The CImg Library

1.4.5

Generated by Doxygen 1.7.1

Thu Nov 11 2010 12:25:46

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.

A complete PDF version of this reference documentation is available here for off-line reading.

A partial translation in Chinese is available here.

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

Class 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<T> 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<T> 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" a
    t coordinates (100,100).
  img.display("My first CImg code");           // Display the image in a displa
    y 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).di
    splay("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 -L/usr/X11R6/lib -lm -l
    pthread -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 CImg<T> : The image structure.

Description of the CImg<T> structure

6.2.1 Structure overview

6.2.2 Image construction/destruction/copy

6.2.3 Image methods

6.2.4 Shared images

6.2.5 Low-level structure

6.3 CImgList<T> : The image list structure.

Description of the CImgList<T> structure

6.3.1 Structure overview

6.3.2 Image list construction/destruction/copy

6.3.3 Image methods

6.3.4 Low-level structure

6.4 CImgDisplay : The image display structure.

Description of the CImgDisplay structure

6.4.1 Structure overview

6.4.2 Image display construction/destruction/copy

6.4.3 Image methods

6.4.4 Low-level structure

6.5 CImgException : The library exception structure.

Description of the CImgException structure

6.5.1 Structure overview

6.6 FAQ : Frequently Asked Questions.

6.6.1 FAQ Summary

- General information and availability

6.6.2 1. General information and availability

6.6.2.1 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.

6.6.2.2 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).

6.6.2.3 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.

6.6.2.4 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.

6.6.2.5 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.

6.6.2.6 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.

6.6.3 2. C++ related questions

6.6.3.1 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.

6.6.3.2 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;
```

6.6.3.3 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. Think seriously about it, and if you have a better solution to provide, let me know so we can discuss about it.

6.7 Setting Environment Variables

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

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

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

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

- **`cimg_OS`** : This variable defines the type of your Operating System. It can be set to **1** (*Unix*), **2** (*Windows*), or **0** (*Other configuration*). It should be actually auto-detected by the CImg library. If this is not the case (`cimg_OS=0`), you will probably have to tune the environment variables described below.

- **`cimg_display`** : This variable defines the type of graphical library used to display images in windows. It can be set to 0 (no display library available), **1** (X11-based display) or **2** (Windows-GDI display). If you are running on a system without X11 or Windows-GDI ability, please set this variable to `0`. This will disable the display support, since the CImg Library doesn't contain the necessary code to display images on systems other than X11 or Windows GDI.

- **`cimg_use_vt100`** : This variable tells the library if the system terminal has VT100 color capabilities. It can be *defined* or *not defined*. Define this variable to get colored output on your terminal, when using the CImg Library.

- **`cimg_verbosity`** : This variable defines the level of run-time debug messages that will be displayed by the CImg Library. It can be set to 0 (no debug messages), 1 (normal debug messages displayed on standard error), 2 (normal debug messages displayed in modal windows, which is the default value), or 3 (high debug messages). Note that setting this value to 3 may slow down your program since more debug tests are made by the library (particularly to check if pixel access is made outside image boundaries). See also CImgException to better understand how debug messages are working.

- **cimg_plugin** : This variable tells the library to use a plugin file to add features to the CImg<T> class. Define it with the path of your plugin file, if you want to add member functions to the CImg<T> class, without having to modify directly the "CImg.h" file. An include of the plugin file is performed in the CImg<T> class. If cimg_plugin 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.

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

6.8.1 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: Files Platform SDK (adapt if needed)
  - 4. In the select liste "Show directories for", choose "library files", and add C: Files Platform SDK (adapt if needed) Edit file C: Files Visual Studio 8\VC\VCProjectDefaults\corewin_express.vsprops (adapt if needed)
  - 6. 7. Remplace the line AdditionalDependencies="kernel32.lib" /> by AdditionalDependencies="kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib" />
- Restart Visual C++
- Import CImg library in your main file

6.9 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 profi
    le");
  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[] = { 0,0,2
    55 };
```

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,1,0,25
    6,0);
```

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).di
    splay(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.

6.10 Using Drawing Functions.

6.10.1 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

6.11 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.

6.11.1 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 end of the buffer (last pixel) till the beginning 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_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);'
```

6.11.2 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");
```

6.11.3 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();
```

6.11.4 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.

6.11.4.1 Neighborhood-based loops for 2D images

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

- **cimg_for2x2(img,x,y,z,v,I)** : Loop along the (x,y)-axes using a centered 2x2 neighborhood.
- **cimg_for3x3(img,x,y,z,v,I)** : Loop along the (x,y)-axes using a centered 3x3 neighborhood.
- **cimg_for4x4(img,x,y,z,v,I)** : Loop along the (x,y)-axes using a centered 4x4 neighborhood.
- **cimg_for5x5(img,x,y,z,v,I)** : 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 that will be updated with the correct pixel values during the loop (see Defining neighborhoods).

6.11.4.2 Neighborhood-based loops for 3D images

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

- **cimg_for2x2x2(img,x,y,z,v,I)** : Loop along the (x,y,z)-axes using a centered 2x2x2 neighborhood.
- **cimg_for3x3x3(img,x,y,z,v,I)** : 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 that will be updated with the correct pixel values during the loop (see Defining neighborhoods).

6.11.4.3 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.

6.11.4.4 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*).

6.11.4.5 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 'v
    olume'
cimg_for3x3x3(volume,x,y,z,0,I,float) { // Loop over the volume, using the neig
    hborhood I
  const float ix = 0.5f*(Incc-Ipcc);    // Compute the derivative along the x-a
    xis.
  const float iy = 0.5f*(Icnc-Icpc);    // Compute the derivative along the y-a
    xis.
```

```
  const float iz = 0.5f*(Iccn-Iccp);    // Compute the derivative along the z-a
    xis.
  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 colo
    r 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 !

6.12 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.

6.13 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).

6.14 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,...

6.15 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.

6.15.1 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.

6.15.2 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.

6.15.3 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.

6.15.4 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 gauss
   ian smoothing");
  const  int nblevels  = cimg_option("-n",16,"Number of quantification levels"
   );
  const bool hidden    = cimg_option("-hidden",false,0);      // This is a hid
   den 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.

6.15.5 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

This namespace encompasses all classes and functions of the CImg library.

Namespaces

- namespace cimg

  Namespace that encompasses low-level functions and variables of the CImg Library.

Classes

- struct CImgException

  Instances of this class are thrown when errors occur during a CImg library function call.

- struct CImgDisplay

  This class represents a window which can display CImg images and handles mouse and keyboard events.

- struct CImg

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

- struct CImgList

  Class representing list of images CImg<T>.

7.1.1 Detailed Description

This namespace encompasses all classes and functions of the CImg library. This namespace is defined to avoid functions and class names collisions that could happen with the include 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 variables afterwards.

7.2 cimg_library::cimg Namespace Reference

Namespace that encompasses *low-level* functions and variables of the CImg Library.

Functions

- std::FILE * output (std::FILE *file)

  Set/get output stream for CImg library messages.

- void info ()

  Print informations about CImg environement variables.

- double eval (const char *const expression, const double x=0, const double y=0, const double z=0, const double v=0)

  Evaluate math expression.

- void warn (const char *const format,...)

  Display a warning message.

- 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 current endianness of the CPU.

- template<typename T >
  void invert_endianness (T *const buffer, const unsigned int size)

  Invert endianness of a memory buffer.

- template<typename T >
  T & invert_endianness (T &a)

  Invert endianness of a single variable.

- unsigned long time ()

  Get the value of a system timer with a millisecond precision.

- void sleep (const unsigned int milliseconds)

  Sleep for a certain numbers of milliseconds.

- unsigned int wait (const unsigned int milliseconds)

  Wait for a certain number of milliseconds since the last call.

- template<typename T >
  T rol (const T a, const unsigned int n=1)

  Return a left bitwise-rotated number.

- template<typename T >
  T ror (const T a, const unsigned int n=1)

  Return a right bitwise-rotated number.

- template<typename T >
  T abs (const T a)

  Return the absolute value of a number.

- template<typename T >
  T sqr (const T val)

  Return the square of a number.

- int xln (const int x)

  Return 1 + log_10(x).

- template<typename t1 , typename t2 >
  cimg::superset< t1, t2 >::type min (const t1 &a, const t2 &b)

  Return the minimum value between two numbers.

- 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 value between three numbers.

- 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 value between four numbers.

- template<typename t1 , typename t2 >
  cimg::superset< t1, t2 >::type max (const t1 &a, const t2 &b)

  Return the maximum value between two numbers.

- 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 value between three numbers.

- 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 value between four numbers.

- template<typename T >
  T sign (const T x)

  Return the sign of a number.

- template<typename T >
  unsigned int nearest_pow2 (const T x)

  Return the nearest power of 2 higher than a given number.

- double sinc (const double x)

  Return the sinc() of a given number.

- template<typename T >
  T mod (const T &x, const T &m)

  Return the modulo of a number.

- template<typename T >
  T minmod (const T a, const T b)

  Return the minmod of two numbers.

- 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 round (const T x, const double y=1, const int rounding_type=0)

  Return a rounded number.

- char uncase (const char x)

  Remove the 'case' of an ASCII character.

- void uncase (char *const string)

  Remove the 'case' of a C string.

- double atof (const char *const str)

  Read a double number from a C-string.

- int strncasecmp (const char *const s1, const char *const s2, const int l)

  Compare the first `n` characters of two C-strings, ignoring the case.

- int strcasecmp (const char *const s1, const char *const s2)

  Compare two C-strings, ignoring the case.

- bool strpare (char *const s, const char delimiter=' ', const bool symmetric=false, const bool is_iterative=false)

  Remove useless delimiters on the borders of a C-string.

- void strescape (char *const s)

  Replace explicit escape sequences " in C-strings.

- const char * basename (const char *const s)

  Compute the basename of a filename.

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

  Return or set path to store temporary files.

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

  Return or set path to the ImageMagick's `convert` tool.

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

  Return path of the GraphicsMagick's `gm` tool.

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

  Return or set path of the `XMedcon` tool.

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

  Return or set path to the 'ffmpeg' command.

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

  Return or set path to the 'gzip' command.

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

  Return or set path to the 'gunzip' command.

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

  Return or set path to the 'dcraw' command.

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

  Split a filename into two strings 'body' and 'extension'.

- char * number_filename (const char *const filename, const int number, const unsigned int n, char *const string)

  Create a numbered version of a filename.

- std::FILE * fopen (const char *const path, const char *const mode)

  Open a file, and check for possible errors.

- int fclose (std::FILE *file)

  Close a file, and check for possible errors.

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

  Try to guess the image format of a filename, using the magic numbers in its header.

- template<typename T >
  int fread (T *const ptr, const unsigned int nmemb, std::FILE *stream)

  Read file data, and check for possible errors.

- template<typename T >
  int fwrite (const T *ptr, const unsigned int nmemb, std::FILE *stream)

  Write data to a file, and check for possible errors.

- 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 centering=false)

  Display a dialog box, where a user can click standard buttons.

Variables

- const double PI = 3.14159265358979323846

  Definition of the mathematical constant PI.

7.2.1 Detailed Description

Namespace that encompasses *low-level* functions and variables of the CImg Library. Most of the functions and variables within this namespace are used by the library for low-level processing. Nevertheless, documented variables and functions of this namespace may be used safely in your own source code.

Warning

Never write `using namespace cimg_library::cimg;` in your source code, since a lot of functions of the `cimg::` namespace have prototypes similar to standard C functions that could defined in the global namespace `::`.

7.2.2 Function Documentation

7.2.2.1 void info ( )

Print informations about CImg environement variables.

Printing is done on the standard error output.

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

Display a warning message.

Parameters

format is a C-string describing the format of the message, as in `std::printf()`.

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

Note

This function is similar to `std::system()` and is here because using the `std::` version on Windows may open undesired consoles.

7.2.2.4 bool cimg_library::cimg::endianness ( )

Return the current endianness of the CPU.

Returns

`false` for "Little Endian", `true` for "Big Endian".

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

Sleep for a certain numbers of milliseconds.

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

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

Wait for a certain number of milliseconds since the last call.

This function is equivalent to sleep() but the waiting time is computed with regard to the last call of wait(). It may be used to temporize your program properly.

7.2.2.7 T cimg_library::cimg::abs (const T *a*)

Return the absolute value of a number.

Note

This function is different from `std::abs()` or `std::fabs()` because it is able to consider a variable of any type, without cast needed.

7.2.2.8 T cimg_library::cimg::mod (const T & *x*, const T & *m*)

Return the modulo of a number.

Note

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

7.2.2.9 T cimg_library::cimg::minmod (const T *a*, const T *b*)

Return the minmod of two numbers.

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.

7.2.2.10 T cimg_library::cimg::round (const T *x*, const double *y = 1*, const int *rounding_type = 0*)

Return a rounded number.

Parameters

x is the number to be rounded.

y is the rounding precision.

rounding_type defines the type of rounding (0=nearest, -1=backward, 1=forward).

Returns

the rounded value, with the same type as parameter x.

7.2.2.11 void cimg_library::cimg::uncase (char *const *string*)

Remove the 'case' of a C string.

Acts in-place.

7.2.2.12 double cimg_library::cimg::atof (const char *const *str*)

Read a double number from a C-string.

Note

This function is quite similar to `std::atof()`, but that it allows the retrieval of fractions as in "1/2".

7.2.2.13 int cimg_library::cimg::strncasecmp (const char *const *s1*, const char *const *s2*, const int *l*)

Compare the first `n` characters of two C-strings, ignoring the case.

Note

This function is defined since it is not provided by all compilers (not an ANSI function).

7.2.2.14 int cimg_library::cimg::strcasecmp (const char *const *s1*, const char *const *s2*)

Compare two C-strings, ignoring the case.

Note

This function is defined since it is not provided by all compilers (not an ANSI function).

7.2.2.15 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 *centering* = `false`)

Display a dialog box, where a user can click standard buttons.

Up to 6 buttons can be defined in the dialog window. This function returns when a user clicked one of the button or closed the dialog window.

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.

button3_label = Label of the 3rd button.

button4_label = Label of the 4th button.

button5_label = Label of the 5th button.

button6_label = Label of the 6th button.

logo = Logo image displayed at the left of the main message. This parameter is optional.

centering = Tell to center the dialog window on the screen.

Returns

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

Note

If a button text is set to 0, then the corresponding button (and the followings) won't appear in the dialog box. At least one button is necessary.

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

  Iterator type for CImg<T>.

- typedef const T * const_iterator

  Const iterator type for CImg<T>.

- typedef T value_type

  Value type.

Constructors / Destructor / Instance Management

- ~CImg ()

  Destructor.

- CImg ()

  Default constructor.

- CImg (const unsigned int dx, const unsigned int dy=1, const unsigned int dz=1, const unsigned int dc=1)

  Constructs a new image with given size (`dx,dy,dz,dc`).

- CImg (const unsigned int dx, const unsigned int dy, const unsigned int dz, const unsigned int dc, const T val)

  Construct an image with given size (`dx,dy,dz,dc`) and with pixel having a default value `val`.

- CImg (const unsigned int dx, const unsigned int dy, const unsigned int dz, const unsigned int dc, const int val0, const int val1,...)

Construct an image with given size (`dx,dy,dz,dc`) and with specified pixel values (int version).

- CImg (const unsigned int dx, const unsigned int dy, const unsigned int dz, const unsigned int dc, const double val0, const double val1,...)

  Construct an image with given size (`dx,dy,dz,dc`) and with specified pixel values (double version).

- CImg (const unsigned int dx, const unsigned int dy, const unsigned int dz, const unsigned int dc, const char *const values, const bool repeat_values)

  Construct an image with given size and with specified values given in a string.

- template<typename t >
  CImg (const t *const data_buffer, const unsigned int dx, const unsigned int dy=1, const unsigned int dz=1, const unsigned int dc=1, const bool shared=false)

  Construct an image from a raw memory buffer.

- **CImg** (const T *const data_buffer, const unsigned int dx, const unsigned int dy=1, const unsigned int dz=1, const unsigned int dc=1, const bool shared=false)
- CImg (const char *const filename)

  Construct an image from an image file.

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

  Default copy constructor.

- **CImg** (const CImg< T > &img)
- template<typename t >
  CImg (const CImg< t > &img, const bool shared)

  Advanced copy constructor.

- **CImg** (const CImg< T > &img, const bool shared)
- template<typename t >
  CImg (const CImg< t > &img, const char *const dimensions)

  Construct an image using dimensions of another image.

- template<typename t >
  CImg (const CImg< t > &img, const char *const dimensions, const T val)

  Construct an image using dimensions of another image, and fill it with given values.

- template<typename t >
  CImg (const CImg< t > &img, const char *const dimensions, const char *const values, const bool repeat_values)

  Construct an image using dimensions of another image, and fill it with given values.

- CImg (const CImgDisplay &disp)

  Construct an image from the content of a CImgDisplay instance.

- CImg< T > & clear ()

  In-place version of the default constructor (STL-compliant name).

- CImg< T > & assign ()

  In-place version of the default constructor/destructor.

- CImg< T > & assign (const unsigned int dx, const unsigned int dy=1, const unsigned int dz=1, const unsigned int dc=1)

  In-place version of the previous constructor.

- CImg< T > & assign (const unsigned int dx, const unsigned int dy, const unsigned int dz, const unsigned int dc, const T val)

  In-place version of the previous constructor.

- CImg< T > & assign (const unsigned int dx, const unsigned int dy, const unsigned int dz, const unsigned int dc, const int val0, const int val1,...)

  In-place version of the previous constructor.

- CImg< T > & assign (const unsigned int dx, const unsigned int dy, const unsigned int dz, const unsigned int dc, const double val0, const double val1,...)

  In-place version of the previous constructor.

- CImg< T > & assign (const unsigned int dx, const unsigned int dy, const unsigned int dz, const unsigned int dc, const char *const values, const bool repeat_values)

  In-place version of the corresponding constructor.

- template<typename t >
  CImg< T > & assign (const t *const data_buffer, const unsigned int dx, const unsigned int dy=1, const unsigned int dz=1, const unsigned int dc=1)

  In-place version of the previous constructor.

- CImg< T > & **assign** (const T *const data_buffer, const unsigned int dx, const unsigned int dy=1, const unsigned int dz=1, const unsigned int dc=1)
- template<typename t >
  CImg< T > & assign (const t *const data_buffer, const unsigned int dx, const unsigned int dy, const unsigned int dz, const unsigned int dc, const bool shared)

  In-place version of the previous constructor, allowing to force the shared state of the instance image.

- CImg< T > & **assign** (const T *const data_buffer, const unsigned int dx, const unsigned int dy, const unsigned int dz, const unsigned int dc, const bool shared)
- CImg< T > & assign (const char *const filename)

  In-place version of the previous constructor.

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

  In-place version of the default copy constructor.

- template<typename t >
  CImg< T > & assign (const CImg< t > &img, const bool shared)

  In-place version of the advanced constructor.

- template<typename t >
  CImg< T > & assign (const CImg< t > &img, const char *const dimensions)

  In-place version of the previous constructor.

- template<typename t >
  CImg< T > & assign (const CImg< t > &img, const char *const dimensions, const T val)

  In-place version of the previous constructor.

- template<typename t >
  CImg< T > & assign (const CImg< t > &img, const char *const dimensions, const char *const values, const bool repeat_values)

  In-place version of the previous constructor.

- CImg< T > & assign (const CImgDisplay &disp)

  In-place version of the previous constructor.

- template<typename t >
  CImg< t > & move_to (CImg< t > &img)

  Move the content of the instance image into another one in a way that memory copies are avoided if possible.

- CImg< T > & **move_to** (CImg< T > &img)
- template<typename t >
  CImgList< t > & **move_to** (CImgList< t > &list, const unsigned int pos=~0U)
- CImg< T > & swap (CImg< T > &img)

  Swap all fields of two images. Use with care !

- 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)

  Fast access to pixel value for reading or writing.

- const T & **operator()** (const unsigned int x, const unsigned int y=0, const unsigned int z=0, const unsigned int c=0) const
- operator const T * () const

  Return address of the pixel buffer.

- **operator T *** ()
- CImg< T > & operator= (const T val)

  Operator=().

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

  Operator=().

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

  Operator=().

- CImg< T > & **operator=** (const CImg< T > &img)
- CImg< T > & operator= (const CImgDisplay &disp)

Operator=().

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

  Operator+=().

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

  Operator+=().

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

  Operator+=().

- CImg< T > & operator++ ()

  Operator++() (prefix).

- CImg< T > operator++ (int)

  Operator++() (postfix).

- CImg< T > operator+ () const

  Operator+() (unary).

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

  Operator+().

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

  Operator+().

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

  Operator+().

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

  Operator-=().

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

  Operator-=().

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

  Operator-=().

- CImg< T > & operator-- ()

  Operator--() (prefix).

- CImg< T > operator-- (int)

  Operator--() (postfix).

- CImg< T > operator- () const

  Operator-() (unary).

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

  Operator-().

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

  Operator-().

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

  Operator-().

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

  Operator=().*

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

  Operator=().*

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

  Operator=().*

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

  Operator().*

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

  Operator().*

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

  Operator().*

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

  Operator/=().

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

  Operator/=().

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

  Operator/=().

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

  Operator/().

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

  Operator/().

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

  Operator/().

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

  Operator=().

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

  Operator=().

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

  Operator=().

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

  Operator%().

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

  Operator%().

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

  Operator%().

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

  Operator&=().

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

  Operator&=().

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

  Operator&=().

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

  Operator&().

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

  Operator|=().

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

Operator|=().

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

  Operator|=().

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

  Operator|().

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

  Operator^=().

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

  Operator^=().

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

  Operator^=().

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

  Operator^().

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

  Operator<<=().

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

  Operator<<=().

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

  Operator<<=().

- template<typename t >
  CImg< T > operator<< (const t val) const

  Operator<<().

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

  Operator>>=().

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

  Operator>>=().

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

  Operator>>=().

- template<typename t >
  CImg< T > operator>> (const t val) const

  Operator>>().

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

  Operator==().

- template<typename t >
  bool operator!= (const CImg< t > &img) const

  Operator!=().

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

  Operator,().

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

  Operator,().

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

  Operator<().

- CImg< T > operator~ () const

  Operator~().

Instance Characteristics

- int width () const

  Return the number of columns of the instance image (size along the X-axis, i.e image width).

- int height () const

  Return the number of rows of the instance image (size along the Y-axis, i.e image height).

- int depth () const

  Return the number of slices of the instance image (size along the Z-axis).

- int spectrum () const

  Return the number of vector channels of the instance image (size along the C-axis).

- unsigned int size () const

  Return the number of image buffer elements.

- T * data ()

  Return a pointer to the pixel buffer.

- const T * **data** () const
- 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 the pixel value located at (x,y,z,v).

- const T * **data** (const unsigned int x, const unsigned int y=0, const unsigned int z=0, const unsigned int c=0) const
- int offset (const int x, const int y=0, const int z=0, const int c=0) const

  Return the offset of the pixel coordinates (x,y,z,v) with respect to the data pointer data.

- iterator begin ()

  Return an iterator to the first image pixel.

- const_iterator **begin** () const
- iterator end ()

  Return an iterator pointing after the last image pixel (STL-compliant name).

- const_iterator **end** () const
- const T & front () const

  Return reference to the first image pixel (STL-compliant name).

- T & **front** ()
- const T & back () const

  Return a reference to the last image pixel (STL-compliant name).

- T & **back** ()
- T & at (const int off, const T out_val)

  Read a pixel value with Dirichlet boundary conditions.

- T **at** (const int off, const T out_val) const
- T & at (const int off)

  Read a pixel value with Neumann boundary conditions.

- T **at** (const int off) const
- T & atX (const int x, const int y, const int z, const int c, const T out_val)

  Read a pixel value with Dirichlet boundary conditions for the first coordinates (x).

- T **atX** (const int x, const int y, const int z, const int c, const T out_val) const
- T & atX (const int x, const int y=0, const int z=0, const int c=0)

  Read a pixel value with Neumann boundary conditions for the first coordinates (x).

- T **atX** (const int x, const int y=0, const int z=0, const int c=0) const
- T & atXY (const int x, const int y, const int z, const int c, const T out_val)

  Read a pixel value with Dirichlet boundary conditions for the two first coordinates (x,y).

- T **atXY** (const int x, const int y, const int z, const int c, const T out_val) const
- T & atXY (const int x, const int y, const int z=0, const int c=0)

  Read a pixel value with Neumann boundary conditions for the two first coordinates (x,y).

- T **atXY** (const int x, const int y, const int z=0, const int c=0) const
- T & atXYZ (const int x, const int y, const int z, const int c, const T out_val)

  Read a pixel value with Dirichlet boundary conditions for the three first coordinates (x,y,z).

- T **atXYZ** (const int x, const int y, const int z, const int c, const T out_val) const
- T & atXYZ (const int x, const int y, const int z, const int c=0)

  Read a pixel value with Neumann boundary conditions for the three first coordinates (x,y,z).

- T **atXYZ** (const int x, const int y, const int z, const int c=0) const
- T & atXYZC (const int x, const int y, const int z, const int c, const T out_val)

  Read a pixel value with Dirichlet boundary conditions.

- T **atXYZC** (const int x, const int y, const int z, const int c, const T out_val) const
- T & atXYZC (const int x, const int y, const int z, const int c)

  Read a pixel value with Neumann boundary conditions.

- T **atXYZC** (const int x, const int y, const int z, const int c) const
- Tfloat linear_atX (const float fx, const int y, const int z, const int c, const T out_val) const

  Read a pixel value using linear interpolation and Dirichlet boundary conditions (first coordinate).

- Tfloat linear_atX (const float fx, const int y=0, const int z=0, const int c=0) const

  Read a pixel value using linear interpolation and Neumann boundary conditions (first coordinate).

- Tfloat linear_atXY (const float fx, const float fy, const int z, const int c, const T out_val) const

  Read a pixel value using linear interpolation and Dirichlet boundary conditions (first two coordinates).

- Tfloat linear_atXY (const float fx, const float fy, const int z=0, const int c=0) const

  Read a pixel value using linear interpolation and Neumann boundary conditions (first two coordinates).

- Tfloat linear_atXYZ (const float fx, const float fy, const float fz, const int c, const T out_val) const

  Read a pixel value using linear interpolation and Dirichlet boundary conditions (first three coordinates).

- Tfloat linear_atXYZ (const float fx, const float fy=0, const float fz=0, const int c=0) const

  Read a pixel value using linear interpolation and Neumann boundary conditions (first three coordinates).

- Tfloat linear_atXYZC (const float fx, const float fy, const float fz, const float fc, const T out_val) const

  Read a pixel value using linear interpolation and Dirichlet boundary conditions.

- Tfloat linear_atXYZC (const float fx, const float fy=0, const float fz=0, const float fc=0) const

  Read a pixel value using linear interpolation and Neumann boundary conditions.

- Tfloat cubic_atX (const float fx, const int y, const int z, const int c, const T out_val) const

  Read a pixel value using cubic interpolation and Dirichlet boundary conditions (first coordinates).

- Tfloat cubic_atX (const float fx, const int y, const int z, const int c, const T out_val, const Tfloat min_val, const Tfloat max_val) const

  Read a pixel value using cubic interpolation and Dirichlet boundary conditions (first coordinates).

- Tfloat cubic_atX (const float fx, const int y=0, const int z=0, const int c=0) const

  Read a pixel value using cubic interpolation and Neumann boundary conditions (first coordinates).

- Tfloat cubic_atX (const float fx, const int y, const int z, const int c, const Tfloat min_val, const Tfloat max_val) const

Read a pixel value using cubic interpolation and Neumann boundary conditions (first coordinates).

- Tfloat cubic_atXY (const float fx, const float fy, const int z, const int c, const T out_val) const

  Read a pixel value using cubic interpolation and Dirichlet boundary conditions.

- Tfloat cubic_atXY (const float fx, const float fy, const int z, const int c, const T out_val, const Tfloat min_val, const Tfloat max_val) const

  Read a pixel value using cubic interpolation and Dirichlet boundary conditions.

- Tfloat cubic_atXY (const float fx, const float fy, const int z=0, const int c=0) const

  Read a pixel value using cubic interpolation and Neumann boundary conditions.

- Tfloat cubic_atXY (const float fx, const float fy, const int z, const int c, const Tfloat min_val, const Tfloat max_val) const

  Read a pixel value using cubic interpolation and Neumann boundary conditions.

- Tfloat cubic_atXYZ (const float fx, const float fy, const float fz, const int c, const T out_val) const

  Read a pixel value using cubic interpolation and Dirichlet boundary conditions.

- Tfloat cubic_atXYZ (const float fx, const float fy, const float fz, const int c, const T out_val, const Tfloat min_val, const Tfloat max_val) const

  Read a pixel value using cubic interpolation and Dirichlet boundary conditions.

- Tfloat cubic_atXYZ (const float fx, const float fy, const float fz, const int c=0) const

  Read a pixel value using cubic interpolation and Neumann boundary conditions.

- Tfloat cubic_atXYZ (const float fx, const float fy, const float fz, const int c, const Tfloat min_val, const Tfloat max_val) const

  Read a pixel value using cubic interpolation and Neumann boundary conditions.

- CImg< T > & set_linear_atXYZ (const T &val, const float fx, const float fy=0, const float fz=0, const int c=0, const bool add=false)

  Set a pixel value, with 3d float coordinates, using linear interpolation.

- CImg< T > & set_linear_atXY (const T &val, const float fx, const float fy=0, const int z=0, const int c=0, const bool add=false)

  Set a pixel value, with 2d float coordinates, using linear interpolation.

- CImg< charT > value_string (const char separator=',', const unsigned int max_size=0) const

  Return a C-string containing the values of the instance image.

- static const char * pixel_type ()

  Return the type of the pixel values.

Instance Checking

- bool is_shared () const

  Return `true` *if current image has shared memory.*

- bool is_empty () const

  Return `true` *if current image is empty.*

- bool is_sameX (const unsigned int dx) const

  Return `true` *if image (*this*) has the specified width.*

- template<typename t >
  bool is_sameX (const CImg< t > &img) const

  Return `true` *if images (*this*) and* `img` *have same width.*

- bool is_sameX (const CImgDisplay &disp) const

  Return `true` *if images (*this*) and the display* `disp` *have same width.*

- bool is_sameY (const unsigned int dy) const

  Return `true` *if image (*this*) has the specified height.*

- template<typename t >
  bool is_sameY (const CImg< t > &img) const

  Return `true` *if images (*this*) and* `img` *have same height.*

- bool is_sameY (const CImgDisplay &disp) const

  Return `true` *if images (*this*) and the display* `disp` *have same height.*

- bool is_sameZ (const unsigned int dz) const

  Return `true` *if image (*this*) has the specified depth.*

- template<typename t >
  bool is_sameZ (const CImg< t > &img) const

  Return `true` *if images (*this*) and* `img` *have same depth.*

- bool is_sameC (const unsigned int dc) const

  Return `true` *if image (*this*) has the specified number of channels.*

- template<typename t >
  bool is_sameC (const CImg< t > &img) const

  Return `true` *if images (*this*) and* `img` *have same _spectrum.*

- bool is_sameXY (const unsigned int dx, const unsigned int dy) const

  Return `true` *if image (*this*) has the specified width and height.*

- template<typename t >
  bool is_sameXY (const CImg< t > &img) const

  Return `true` *if images have same width and same height.*

- bool is_sameXY (const CImgDisplay &disp) const

  Return `true` *if image (*this*) and the display* `disp` *have same width and same height.*

- bool is_sameXZ (const unsigned int dx, const unsigned int dz) const

  Return `true` *if image (*this*) has the specified width and depth.*

- template<typename t >
  bool is_sameXZ (const CImg< t > &img) const

  Return `true` *if images have same width and same depth.*

- bool is_sameXC (const unsigned int dx, const unsigned int dc) const

  Return `true` *if image (*this*) has the specified width and number of channels.*

- template<typename t >
  bool is_sameXC (const CImg< t > &img) const

  Return `true` *if images have same width and same number of channels.*

- bool is_sameYZ (const unsigned int dy, const unsigned int dz) const

  Return `true` *if image (*this*) has the specified height and depth.*

- template<typename t >
  bool is_sameYZ (const CImg< t > &img) const

  Return `true` *if images have same height and same depth.*

- bool is_sameYC (const unsigned int dy, const unsigned int dc) const

  Return `true` *if image (*this*) has the specified height and number of channels.*

- template<typename t >
  bool is_sameYC (const CImg< t > &img) const

  Return `true` *if images have same height and same number of channels.*

- bool is_sameZC (const unsigned int dz, const unsigned int dc) const

  Return `true` *if image (*this*) has the specified depth and number of channels.*

- template<typename t >
  bool is_sameZC (const CImg< t > &img) const

  Return `true` *if images have same depth and same number of channels.*

- bool is_sameXYZ (const unsigned int dx, const unsigned int dy, const unsigned int dz) const

  Return `true` *if image (*this*) has the specified width, height and depth.*

- template<typename t >
  bool is_sameXYZ (const CImg< t > &img) const

  Return `true` *if images have same width, same height and same depth.*

- bool is_sameXYC (const unsigned int dx, const unsigned int dy, const unsigned int dc) const

  Return `true` *if image (*this*) has the specified width, height and depth.*

- template<typename t >
  bool is_sameXYC (const CImg< t > &img) const

  Return `true` *if images have same width, same height and same number of channels.*

- bool is_sameXZC (const unsigned int dx, const unsigned int dz, const unsigned int dc) const

  Return `true` *if image (*this*) has the specified width, height and number of channels.*

- template<typename t >
  bool is_sameXZC (const CImg< t > &img) const

Return `true` if images have same width, same depth and same number of channels.

- bool is_sameYZC (const unsigned int dy, const unsigned int dz, const unsigned int dc) const

  Return `true` if image (∗this) has the specified height, depth and number of channels.

- template<typename t >
  bool is_sameYZC (const CImg< t > &img) const

  Return `true` if images have same heigth, same depth and same number of channels.

- bool is_sameXYZC (const unsigned int dx, const unsigned int dy, const unsigned int dz, const unsigned int dc) const

  Return `true` if image (∗this) has the specified width, height, depth and number of channels.

- template<typename t >
  bool is_sameXYZC (const CImg< t > &img) const

  Return `true` if images (∗this) and `img` have same width, same height, same depth and same number of channels.

- bool containsXYZC (const int x, const int y=0, const int z=0, const int c=0) const

  Return `true` if pixel (x,y,z,c) is inside image boundaries.

- template<typename t >
  bool contains (const T &pixel, t &x, t &y, t &z, t &c) const

  Return `true` if specified referenced value is inside image boundaries. If true, returns pixel coordinates in (x,y,z,c).

- template<typename t >
  bool contains (const T &pixel, t &x, t &y, t &z) const

  Return `true` if specified referenced value is inside image boundaries. If true, returns pixel coordinates in (x,y,z).

- template<typename t >
  bool contains (const T &pixel, t &x, t &y) const

  Return `true` if specified referenced value is inside image boundaries. If true, returns pixel coordinates in (x,y).

- template<typename t >
  bool contains (const T &pixel, t &x) const

  Return `true` if specified referenced value is inside image boundaries. If true, returns pixel coordinates in (x).

- bool contains (const T &pixel) const

  Return `true` if specified referenced value is inside the image boundaries.

- template<typename t >
  bool is_overlapped (const CImg< t > &img) const

  Return `true` if the memory buffers of the two images overlaps.

- 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

Return true if the set (instance,primitives,colors,opacities) stands for a valid 3d object.

- bool is_CImg3d (const bool full_check=true, char *const error_message=0) const

  Test if an image is a valid CImg3d object.

Mathematical Functions

- CImg< T > & sqr ()

  Compute the square value of each pixel.

- CImg< Tfloat > **get_sqr** () const
- CImg< T > & sqrt ()

  Compute the square root of each pixel value.

- CImg< Tfloat > **get_sqrt** () const
- CImg< T > & exp ()

  Compute the exponential of each pixel value.

- CImg< Tfloat > **get_exp** () const
- CImg< T > & log ()

  Compute the log of each each pixel value.

- CImg< Tfloat > **get_log** () const
- CImg< T > & log10 ()

  Compute the log10 of each each pixel value.

- CImg< Tfloat > **get_log10** () const
- CImg< T > & abs ()

  Compute the absolute value of each pixel value.

- CImg< Tfloat > **get_abs** () const
- CImg< T > & sign ()

  Compute the sign of each pixel value.

- CImg< Tfloat > **get_sign** () const
- CImg< T > & cos ()

  Compute the cosinus of each pixel value.

- CImg< Tfloat > **get_cos** () const
- CImg< T > & sin ()

  Compute the sinus of each pixel value.

- CImg< Tfloat > **get_sin** () const
- CImg< T > & sinc ()

  Compute the sinus cardinal of each pixel value.

- CImg< Tfloat > **get_sinc** () const
- CImg< T > & tan ()

Compute the tangent of each pixel.

- CImg< Tfloat > **get_tan** () const
- CImg< T > & cosh ()

  Compute the hyperbolic cosine of each pixel value.

- CImg< Tfloat > **get_cosh** () const
- CImg< T > & sinh ()

  Compute the hyperbolic sine of each pixel value.

- CImg< Tfloat > **get_sinh** () const
- CImg< T > & tanh ()

  Compute the hyperbolic tangent of each pixel value.

- CImg< Tfloat > **get_tanh** () const
- CImg< T > & acos ()

  Compute the arc-cosine of each pixel value.

- CImg< Tfloat > **get_acos** () const
- CImg< T > & asin ()

  Compute the arc-sinus of each pixel value.

- CImg< Tfloat > **get_asin** () const
- CImg< T > & atan ()

  Compute the arc-tangent of each pixel.

- CImg< Tfloat > **get_atan** () const
- template<typename t >
  CImg< T > & atan2 (const CImg< t > &img)

  Compute the arc-tangent of each pixel.

- template<typename t >
  CImg< Tfloat > **get_atan2** (const CImg< t > &img) const
- CImg< T > & mul (const char *const expression)

  Pointwise multiplication between an image and an expression.

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

  Pointwise multiplication between two images.

- template<typename t >
  CImg< typename cimg::superset< T, t >::type > **get_mul** (const CImg< t > &img) const
- CImg< T > & div (const char *const expression)

  Pointwise division between an image and an expression.

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

  Pointwise division between two images.

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

- CImg< T > & pow (const double p)

  Compute the power by p of each pixel value.

- CImg< Tfloat > **get_pow** (const double p) const
- template<typename t >
  CImg< T > & pow (const CImg< t > &img)

  Compute the power of each pixel value.

- template<typename t >
  CImg< Tfloat > **get_pow** (const CImg< t > &img) const
- CImg< T > & pow (const char *const expression)

  Compute the power of each pixel value.

- CImg< Tfloat > **get_pow** (const char *const expression) const
- 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
- template<typename t >
  CImg< T > & **rol** (const CImg< t > &img)
- template<typename t >
  CImg< T > **get_rol** (const CImg< t > &img) const
- CImg< T > & **rol** (const char *const expression)
- CImg< T > **get_rol** (const char *const expression) const
- 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
- template<typename t >
  CImg< T > & **ror** (const CImg< t > &img)
- template<typename t >
  CImg< T > **get_ror** (const CImg< t > &img) const
- CImg< T > & **ror** (const char *const expression)
- CImg< T > **get_ror** (const char *const expression) const
- CImg< T > & min (const T val)

  Pointwise min operator between an image and a value.

- CImg< T > **get_min** (const T val) const
- 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
- CImg< T > & min (const char *const expression)

  Pointwise min operator between an image and a string.

- CImg< Tfloat > **get_min** (const char *const expression) const
- CImg< T > & max (const T val)

  Pointwise max operator between an image and a value.

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

  Pointwise max operator between an image and a string.

- CImg< Tfloat > **get_max** (const char *const expression) const
- T & min ()

  Return a reference to the minimum pixel value of the instance image.

- const T & **min** () const
- T & max ()

  Return a reference to the maximum pixel value of the instance image.

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

  Return a reference to the minimum pixel value and return also the maximum pixel value.

- template<typename t >
  const T & **min_max** (t &max_val) const
- template<typename t >
  T & max_min (t &min_val)

  Return a reference to the maximum pixel value and return also the minimum pixel value.

- template<typename t >
  const T & **max_min** (t &min_val) const
- T kth_smallest (const unsigned int k) const

  Return the kth smallest element of the image.

- T median () const

  Return the median value of the image.

- Tdouble sum () const

  Return the sum of all the pixel values in an image.

- Tdouble mean () const

  Return the mean pixel value of the instance image.

- Tdouble variance (const unsigned int variance_method=1) const

  Return the variance of the image.

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

  Return the variance and the mean of the image.

- Tdouble variance_noise (const unsigned int variance_method=1) const

  Estimate noise variance of the instance image.

- 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 valmax=255) const

  Compute the PSNR 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 expression.

- CImg< T > & stats (const unsigned int variance_method=1)

  Compute a statistics vector (min,max,mean,variance,xmin,ymin,zmin,cmin,xmax,ymax,zmax,cmax).

- CImg< Tdouble > **get_stats** (const unsigned int variance_method=1) const

Vector / Matrix Operations

- Tdouble magnitude (const int magnitude_type=2) const

  Return the norm of the current vector/matrix. `ntype` = norm type (0=L2, 1=L1, -1=Linf).

- Tdouble trace () const

  Return the trace of the image, viewed as a matrix.

- Tdouble det () const

  Return the determinant of the image, viewed as a matrix.

- template<typename t >
  Tdouble dot (const CImg< t > &img) const

  Return the dot product of the current vector/matrix with the vector/matrix `img`.

- CImg< T > get_vector_at (const unsigned int x, const unsigned int y=0, const unsigned int z=0) const

  Return a new image corresponding to the vector located at (`x,y,z`) of the current vector-valued image.

- CImg< T > get_matrix_at (const unsigned int x=0, const unsigned int y=0, const unsigned int z=0) const

  Return a new image corresponding to the square matrix located at (`x,y,z`) of the current vector-valued image.

- CImg< T > get_tensor_at (const unsigned int x, const unsigned int y=0, const unsigned int z=0) const

  Return a new image corresponding to the diffusion tensor located at (`x,y,z`) of the current vector-valued image.

- 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 the image `vec` as the vector valued pixel located at (`x,y,z`) of the current vector-valued image.

- 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 the image `vec` as the square matrix-valued pixel located at (`x,y,z`) of the current vector-valued image.

- 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 the image `vec` as the tensor valued pixel located at (`x,y,z`) of the current vector-valued image.

- CImg< T > & vector ()

  Unroll all images values into a one-column vector.

- CImg< T > **get_vector** () const
- CImg< T > & matrix ()

  Realign pixel values of the instance image as a square matrix.

- CImg< T > **get_matrix** () const
- CImg< T > & tensor ()

  Realign pixel values of the instance image as a symmetric tensor.

- CImg< T > **get_tensor** () const
- CImg< T > & diagonal ()

  Get a diagonal matrix, whose diagonal coefficients are the coefficients of the input image.

- CImg< T > **get_diagonal** () const
- CImg< T > & identity_matrix ()

  Get an identity matrix having same dimension than instance image.

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

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

- CImg< T > **get_sequence** (const T a0, const T a1) const
- CImg< T > & transpose ()

  Transpose the current matrix.

- CImg< T > **get_transpose** () const
- template<typename t >
  CImg< T > & cross (const CImg< t > &img)

  Compute the cross product between two 3d vectors.

- template<typename t >
  CImg< typename cimg::superset< T, t >::type > **get_cross** (const CImg< t > &img) const
- CImg< T > & invert (const bool use_LU=true)

Invert the current matrix.

- CImg< Tfloat > **get_invert** (const bool use_LU=true) const
- CImg< T > & pseudoinvert ()

  Compute the pseudo-inverse (Moore-Penrose) of the matrix.

- CImg< Tfloat > **get_pseudoinvert** () const
- template<typename t >
  CImg< T > & solve (const CImg< t > &A)

  Solve a linear system AX=B where B=∗this.

- template<typename t >
  CImg< typename cimg::superset2< T, t, float >::type > **get_solve** (const CImg< t > &A) const
- template<typename t >
  CImg< T > & solve_tridiagonal (const CImg< t > &a, const CImg< t > &b, const CImg< t > &c)

  Solve a linear system AX=B where B=∗this and A is a tridiagonal matrix A = [b0,c0,0,...; a1,b1,c1,0,... ; ... ; ...,0,aN,bN].

- template<typename t >
  CImg< typename cimg::superset2< T, t, float >::type > **get_solve_tridiagonal** (const CImg< t > &a, const CImg< t > &b, const CImg< t > &c) const
- template<typename t >
  const CImg< T > & eigen (CImg< t > &val, CImg< t > &vec) const

  Compute the eigenvalues and eigenvectors of a matrix.

- CImgList< Tfloat > **get_eigen** () const
- template<typename t >
  const CImg< T > & symmetric_eigen (CImg< t > &val, CImg< t > &vec) const

  Compute the eigenvalues and eigenvectors of a symmetric matrix.

- CImgList< Tfloat > **get_symmetric_eigen** () const
- template<typename t >
  CImg< T > & sort (CImg< t > &permutations, const bool increasing=true)

  Sort values of a vector and get corresponding permutations.

- template<typename t >
  CImg< T > **get_sort** (CImg< t > &permutations, const bool increasing=true) const
- CImg< T > & sort (const bool increasing=true, const char axis=0)

  Sort image values.

- CImg< T > **get_sort** (const bool increasing=true, const char axis=0) const
- 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 a general matrix.

- CImgList< Tfloat > **get_SVD** (const bool sorting=true, const unsigned int max_iteration=40, const float lambda=0) const
- template<typename t >
  CImg< T > & dijkstra (const unsigned int starting_node, const unsigned int ending_node, CImg< t > &previous)

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) const
- 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
- 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.

- 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)

  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.

- 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 vector field.

- static CImg< T > string (const char *const str)

  Return an image containing the specified string.

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

  Return a vector with specified coefficients.

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

  Return a vector with specified coefficients.

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

  Return a vector with specified coefficients.

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

Return a vector with specified coefficients.

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

  Return a vector with specified coefficients.

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

  Return a vector with specified coefficients.

- 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 vector with specified coefficients.

- 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 vector with specified coefficients.

- 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 vector with specified coefficients.

- 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 vector with specified coefficients.

- 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 vector with specified coefficients.

- 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 vector with specified coefficients.

- 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 vector with specified coefficients.

- 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 vector with specified coefficients.

- 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 vector with specified coefficients.

- 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 vector with specified coefficients.

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

  Return a 1x1 square matrix with specified coefficients.

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

  Return a 2x2 square matrix with 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 square matrix with 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 square matrix with 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 square matrix with specified coefficients.

- static CImg< T > tensor (const T &a1)

  Return a 1x1 symmetric matrix with specified coefficients.

- static CImg< T > tensor (const T &a1, const T &a2, const T &a3)

  Return a 2x2 symmetric matrix tensor with specified coefficients.

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

  Return a 3x3 symmetric matrix with specified coefficients.

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

  Return a 1x1 diagonal matrix with specified coefficients.

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

  Return a 2x2 diagonal matrix with specified coefficients.

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

  Return a 3x3 diagonal matrix with specified coefficients.

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

  Return a 4x4 diagonal matrix with 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 with 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 quaternion_data=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 an image by a value `val`.

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

  Fill sequentially all pixel values with values val0 and val1 respectively.

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

  Fill sequentially all pixel values with values val0 and val1 and val2.

- CImg< T > **get_fill** (const T val0, const T val1, const T val2) const
- CImg< T > & fill (const T val0, const T val1, const T val2, const T val3)

  Fill sequentially all pixel values with values val0 and val1 and val2 and val3.

- CImg< T > **get_fill** (const T val0, const T val1, const T val2, const T val3) const
- CImg< T > & fill (const T val0, const T val1, const T val2, const T val3, const T val4)

  Fill sequentially all pixel values with values val0 and val1 and val2 and val3 and val4.

- CImg< T > **get_fill** (const T val0, const T val1, const T val2, const T val3, const T val4) const
- 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 values val0 and val1 and val2 and val3 and val4 and val5.

- CImg< T > **get_fill** (const T val0, const T val1, const T val2, const T val3, const T val4, const T val5) const
- 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 pixel values.

- 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
- 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 pixel values.

- 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

- 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 pixel values.

- 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
- 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 pixel values.

- 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
- 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 pixel values.

- 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
- 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 pixel values.

- 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
- 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 pixel values.

- 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
- 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 pixel values.

- 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
- 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 pixel values.

- 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
- 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 pixel values.

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

  Fill image values according to the given expression, which can be a formula or a list of values.

- CImg< T > **get_fill** (const char *const values, const bool repeat_values) const
- template<typename t >
  CImg< T > & fill (const CImg< t > &values, const bool repeat_values=true)

  Fill image values according to the values found in the specified image.

- template<typename t >
  CImg< T > **get_fill** (const CImg< t > &values, const bool repeat_values=true) const
- CImg< T > & fillX (const unsigned int y, const unsigned int z, const unsigned int c, const int a0,...)

  Fill image values along the X-axis at the specified pixel position (y,z,c).

- CImg< T > & **fillX** (const unsigned int y, const unsigned int z, const unsigned int c, const double a0,...)
- CImg< T > & fillY (const unsigned int x, const unsigned int z, const unsigned int c, const int a0,...)

  Fill image values along the Y-axis at the specified pixel position (x,z,c).

- CImg< T > & **fillY** (const unsigned int x, const unsigned int z, const unsigned int c, const double a0,...)
- CImg< T > & fillZ (const unsigned int x, const unsigned int y, const unsigned int c, const int a0,...)

  Fill image values along the Z-axis at the specified pixel position (x,y,c).

- CImg< T > & **fillZ** (const unsigned int x, const unsigned int y, const unsigned int c, const double a0,...)
- CImg< T > & fillC (const unsigned int x, const unsigned int y, const unsigned int z, const int a0,...)

  Fill image values along the C-axis at the specified pixel position (x,y,z).

- CImg< T > & **fillC** (const unsigned int x, const unsigned int y, const unsigned int z, const double a0,...)
- CImg< T > & invert_endianness ()

  Invert endianness of the image buffer.

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

  Fill the instance image with random values between specified range.

- CImg< T > **get_rand** (const T val_min, const T val_max) const
- CImg< T > & round (const double y=1, const int rounding_type=0)

  Compute image with rounded pixel values.

- CImg< T > **get_round** (const double y=1, const unsigned int rounding_type=0) const
- CImg< T > & noise (const double sigma, const unsigned int noise_type=0)

  Add random noise to the values of the instance image.

- CImg< T > **get_noise** (const double sigma, const unsigned int noise_type=0) const

- CImg< T > & normalize (const T value_min, const T value_max)

  Linearly normalize values of the instance image between `value_min` *and* `value_max`*.*

- CImg< Tfloat > **get_normalize** (const T value_min, const T value_max) const
- CImg< T > & normalize ()

  Normalize multi-valued pixels of the instance image, with respect to their L2-norm.

- CImg< Tfloat > **get_normalize** () const
- CImg< T > & norm (const int norm_type=2)

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

- CImg< Tfloat > **get_norm** (const int norm_type=2) const
- CImg< T > & cut (const T value_min, const T value_max)

  Cut values of the instance image between `value_min` *and* `value_max`*.*

- CImg< T > **get_cut** (const T value_min, const T value_max) const
- CImg< T > & quantize (const unsigned int nb_levels, const bool keep_range=true)

  Uniformly quantize values of the instance image into `nb_levels` *levels.*

- CImg< T > **get_quantize** (const unsigned int n, const bool keep_range=true) const
- CImg< T > & threshold (const T value, const bool soft_threshold=false, const bool strict_threshold=false)

  Threshold values of the instance image.

- CImg< T > **get_threshold** (const T value, const bool soft_threshold=false, const bool strict_threshold=false) const
- CImg< T > & histogram (const unsigned int nb_levels, const T value_min=(T) 0, const T value_max=(T) 0)

  Compute the histogram of the instance image.

- CImg< floatT > **get_histogram** (const unsigned int nb_levels, const T value_min=(T) 0, const T value_max=(T) 0) const
- CImg< T > & equalize (const unsigned int nb_levels, const T value_min=(T) 0, const T value_max=(T) 0)

  Compute the histogram-equalized version of the instance image.

- CImg< T > **get_equalize** (const unsigned int nblevels, const T val_min=(T) 0, const T val_max=(T) 0) const
- template<typename t >
  CImg< T > & index (const CImg< t > &palette, const bool dithering=false, const bool map_indexes=false)

  Index multi-valued pixels of the instance image, regarding to a predefined palette.

- template<typename t >
  CImg< typename CImg< t >::Tuint > **get_index** (const CImg< t > &palette, const bool dithering=false, const bool map_indexes=true) const
- template<typename t >
  CImg< T > & map (const CImg< t > &palette)

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

- template<typename t >
  CImg< t > **get_map** (const CImg< t > &palette) const
- CImg< T > & label_regions ()

  Create a map of indexed labels counting disconnected regions with same intensities.

- CImg< uintT > **get_label_regions** () const

Color Base Management

- CImg< T > & RGBtoHSV ()

  Convert color pixels from (R,G,B) to (H,S,V).

- CImg< Tfloat > **get_RGBtoHSV** () const
- CImg< T > & HSVtoRGB ()

  Convert color pixels from (H,S,V) to (R,G,B).

- CImg< Tuchar > **get_HSVtoRGB** () const
- CImg< T > & RGBtoHSL ()

  Convert color pixels from (R,G,B) to (H,S,L).

- CImg< Tfloat > **get_RGBtoHSL** () const
- CImg< T > & HSLtoRGB ()

  Convert color pixels from (H,S,L) to (R,G,B).

- CImg< Tuchar > **get_HSLtoRGB** () const
- CImg< T > & RGBtoHSI ()
- CImg< Tfloat > **get_RGBtoHSI** () const
- CImg< T > & HSItoRGB ()

  Convert color pixels from (H,S,I) to (R,G,B).

- CImg< Tfloat > **get_HSItoRGB** () const
- CImg< T > & RGBtoYCbCr ()

  Convert color pixels from (R,G,B) to (Y,Cb,Cr).

- CImg< Tuchar > **get_RGBtoYCbCr** () const
- CImg< T > & YCbCrtoRGB ()

  Convert color pixels from (R,G,B) to (Y,Cb,Cr).

- CImg< Tuchar > **get_YCbCrtoRGB** () const
- CImg< T > & RGBtoYUV ()

  Convert color pixels from (R,G,B) to (Y,U,V).

- CImg< Tfloat > **get_RGBtoYUV** () const
- CImg< T > & YUVtoRGB ()

  Convert color pixels from (Y,U,V) to (R,G,B).

- CImg< Tuchar > **get_YUVtoRGB** () const
- CImg< T > & RGBtoCMY ()

  Convert color pixels from (R,G,B) to (C,M,Y).

- CImg< Tuchar > **get_RGBtoCMY** () const
- CImg< T > & CMYtoRGB ()

  Convert (C,M,Y) pixels of a color image into the (R,G,B) color space.

- CImg< Tuchar > **get_CMYtoRGB** () const
- CImg< T > & CMYtoCMYK ()

  Convert color pixels from (C,M,Y) to (C,M,Y,K).

- CImg< Tuchar > **get_CMYtoCMYK** () const
- CImg< T > & CMYKtoCMY ()

  Convert (C,M,Y,K) pixels of a color image into the (C,M,Y) color space.

- CImg< Tfloat > **get_CMYKtoCMY** () const
- CImg< T > & RGBtoXYZ ()

  Convert color pixels from (R,G,B) to (X,Y,Z)_709.

- CImg< Tfloat > **get_RGBtoXYZ** () const
- CImg< T > & XYZtoRGB ()

  Convert (X,Y,Z)_709 pixels of a color image into the (R,G,B) color space.

- CImg< Tuchar > **get_XYZtoRGB** () const
- CImg< T > & XYZtoLab ()

  Convert (X,Y,Z)_709 pixels of a color image into the (L∗,a∗,b∗) color space.

- CImg< Tfloat > **get_XYZtoLab** () const
- CImg< T > & LabtoXYZ ()

  Convert (L,a,b) pixels of a color image into the (X,Y,Z) color space.

- CImg< Tfloat > **get_LabtoXYZ** () const
- CImg< T > & XYZtoxyY ()

  Convert (X,Y,Z)_709 pixels of a color image into the (x,y,Y) color space.

- CImg< Tfloat > **get_XYZtoxyY** () const
- CImg< T > & xyYtoXYZ ()

  Convert (x,y,Y) pixels of a color image into the (X,Y,Z)_709 color space.

- CImg< Tfloat > **get_xyYtoXYZ** () const
- CImg< T > & RGBtoLab ()

  Convert a (R,G,B) image to a (L,a,b) one.

- CImg< Tfloat > **get_RGBtoLab** () const
- CImg< T > & LabtoRGB ()

  Convert a (L,a,b) image to a (R,G,B) one.

- CImg< Tuchar > **get_LabtoRGB** () const
- CImg< T > & RGBtoxyY ()

  Convert a (R,G,B) image to a (x,y,Y) one.

- CImg< Tfloat > **get_RGBtoxyY** () const

- CImg< T > & xyYtoRGB ()

  Convert a (x,y,Y) image to a (R,G,B) one.

- CImg< Tuchar > **get_xyYtoRGB** () const
- CImg< T > & RGBtoCMYK ()

  Convert a (R,G,B) image to a (C,M,Y,K) one.

- CImg< Tfloat > **get_RGBtoCMYK** () const
- CImg< T > & CMYKtoRGB ()

  Convert a (C,M,Y,K) image to a (R,G,B) one.

- CImg< Tuchar > **get_CMYKtoRGB** () const
- CImg< T > & RGBtoBayer ()

  Convert a (R,G,B) image to a Bayer-coded representation.

- CImg< T > **get_RGBtoBayer** () const
- CImg< T > & BayertoRGB (const unsigned int interpolation_type=3)

  Convert a Bayer-coded image to a (R,G,B) color image.

- CImg< Tuchar > **get_BayertoRGB** (const unsigned int interpolation_type=3) const
- static const CImg< Tuchar > & default_LUT256 ()

  Return a palette 'default' with 256 (R,G,B) entries.

- static const CImg< Tuchar > & HSV_LUT256 ()

  Return palette 'HSV' with 256 (R,G,B) entries.

- static const CImg< Tuchar > & lines_LUT256 ()

  Return palette 'lines' with 256 (R,G,B) entries.

- static const CImg< Tuchar > & hot_LUT256 ()

  Return the palette 'hot' with 256 (R,G,B) entries.

- static const CImg< Tuchar > & cool_LUT256 ()

  Return the palette 'cool' with 256 (R,G,B) entries.

- static const CImg< Tuchar > & jet_LUT256 ()

  Return palette 'jet' with 256 (R,G,B) entries.

- static const CImg< Tuchar > & flag_LUT256 ()

  Return palette 'flag' with 256 (R,G,B) entries.

- static const CImg< Tuchar > & cube_LUT256 ()

  Return palette 'cube' with 256 (R,G,B) entries.

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 border_conditions=0, const float centering_x=0, const float centering_y=0, const float centering_z=0, const float centering_c=0)

  Resize an image.

- 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 border_conditions=0, const float centering_x=0, const float centering_y=0, const float centering_z=0, const float centering_c=0) const

- template<typename t >
  CImg< T > & resize (const CImg< t > &src, const int interpolation_type=1, const unsigned int border_conditions=0, const float centering_x=0, const float centering_y=0, const float centering_z=0, const float centering_c=0)

  Resize an image.

- template<typename t >
  CImg< T > **get_resize** (const CImg< t > &src, const int interpolation_type=1, const unsigned int border_conditions=0, const float centering_x=0, const float centering_y=0, const float centering_z=0, const float centering_c=0) const
- CImg< T > & resize (const CImgDisplay &disp, const int interpolation_type=1, const unsigned int border_conditions=0, const float centering_x=0, const float centering_y=0, const float centering_z=0, const float centering_c=0)

  Resize an image.

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

  Half-resize an image, using a special optimized filter.

- CImg< T > **get_resize_halfXY** () const
- CImg< T > & resize_doubleXY ()

  Upscale an image by a factor 2x.

- CImg< T > **get_resize_doubleXY** () const
- CImg< T > & resize_tripleXY ()

  Upscale an image by a factor 3x.

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

  Mirror an image along the specified axis.

- CImg< T > **get_mirror** (const char axis) const
- CImg< T > & shift (const int deltax, const int deltay=0, const int deltaz=0, const int deltac=0, const int border_condition=0)

  Shift the image.

- CImg< T > **get_shift** (const int deltax, const int deltay=0, const int deltaz=0, const int deltac=0, const int border_condition=0) const

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

  Permute axes order.

- CImg< T > **get_permute_axes** (const char *const order) const
- CImg< T > & unroll (const char axis)

  Unroll all images values into specified axis.

- CImg< T > **get_unroll** (const char axis) const
- CImg< T > & rotate (const float angle, const unsigned int border_conditions=0, const unsigned int interpolation=1)

  Rotate an image.

- CImg< T > **get_rotate** (const float angle, const unsigned int border_conditions=0, const unsigned int interpolation=1) const
- CImg< T > & rotate (const float angle, const float cx, const float cy, const float zoom, const unsigned int border_conditions=3, const unsigned int interpolation=1)

  Rotate an image around a center point (`cx,cy`*).*

- CImg< T > **get_rotate** (const float angle, const float cx, const float cy, const float zoom, const unsigned int border_conditions=3, const unsigned int interpolation=1) const
- template<typename t >
  CImg< T > & warp (const CImg< t > &warp, const bool relative=false, const bool interpolation=true, const unsigned int border_conditions=0)

  Warp an image.

- template<typename t >
  CImg< T > **get_warp** (const CImg< t > &warp, const bool is_relative=false, const bool interpolation=true, const unsigned int border_conditions=0) const
- CImg< T > & projections2d (const unsigned int x0, const unsigned int y0, const unsigned int z0, const int dx=-100, const int dy=-100, const int dz=-100)

  Return a 2d representation of a 3d image, with three slices.

- CImg< T > **get_projections2d** (const unsigned int x0, const unsigned int y0, const unsigned int z0, const int dx=-100, const int dy=-100, const int dz=-100) const
- 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 border_condition=false)

  Get a square region of the image.

- 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 border_condition=false) const
- CImg< T > & crop (const int x0, const int y0, const int z0, const int x1, const int y1, const int z1, const bool border_condition=false)

  Get a rectangular part of the instance image.

- CImg< T > **get_crop** (const int x0, const int y0, const int z0, const int x1, const int y1, const int z1, const bool border_condition=false) const
- CImg< T > & crop (const int x0, const int y0, const int x1, const int y1, const bool border_condition=false)

  Get a rectangular part of the instance image.

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

  Get a rectangular part of the instance image.

- CImg< T > **get_crop** (const int x0, const int x1, const bool border_condition=false) const
- CImg< T > & autocrop (const T value, const char *const axes="czyx")

  Autocrop an image, regarding of the specified backround value.

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

  Autocrop an image, regarding of the specified backround color.

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

  Autocrop an image, regarding of the specified backround color.

- template<typename t >
  CImg< T > **get_autocrop** (const CImg< t > &color, const char *const axes="zyx") const
- CImg< T > & column (const unsigned int x0)

  Get one column.

- CImg< T > **get_column** (const unsigned int x0) const
- CImg< T > & columns (const unsigned int x0, const unsigned int x1)

  Get a set of columns.

- CImg< T > **get_columns** (const unsigned int x0, const unsigned int x1) const
- CImg< T > & line (const unsigned int y0)

  Get a line.

- CImg< T > **get_line** (const unsigned int y0) const
- CImg< T > & lines (const unsigned int y0, const unsigned int y1)

  Get a set of lines.

- CImg< T > **get_lines** (const unsigned int y0, const unsigned int y1) const
- CImg< T > & slice (const unsigned int z0)

  Get a slice.

- CImg< T > **get_slice** (const unsigned int z0) const
- CImg< T > & slices (const unsigned int z0, const unsigned int z1)

  Get a set of slices.

- CImg< T > **get_slices** (const unsigned int z0, const unsigned int z1) const
- CImg< T > & channel (const unsigned int c0)

  Get a channel.

- CImg< T > **get_channel** (const unsigned int c0) const
- CImg< T > & channels (const unsigned int c0, const unsigned int c1)

  Get a set of channels.

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

  Get a shared-memory image referencing a set of points of the instance image.

- 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
- CImg< T > get_shared_lines (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 set of lines of the instance image.

- const CImg< T > **get_shared_lines** (const unsigned int y0, const unsigned int y1, const unsigned int z0=0, const unsigned int c0=0) const
- CImg< T > get_shared_line (const unsigned int y0, const unsigned int z0=0, const unsigned int c0=0)

  Return a shared-memory image referencing one particular line (y0,z0,c0) of the instance image.

- const CImg< T > **get_shared_line** (const unsigned int y0, const unsigned int z0=0, const unsigned int c0=0) const
- CImg< T > get_shared_planes (const unsigned int z0, const unsigned int z1, const unsigned int c0=0)

  Return a shared memory image referencing a set of planes (z0->z1,c0) of the instance image.

- const CImg< T > **get_shared_planes** (const unsigned int z0, const unsigned int z1, const unsigned int c0=0) const
- CImg< T > get_shared_plane (const unsigned int z0, const unsigned int c0=0)

  Return a shared-memory image referencing one plane (z0,c0) of the instance image.

- const CImg< T > **get_shared_plane** (const unsigned int z0, const unsigned int c0=0) const
- CImg< T > get_shared_channels (const unsigned int c0, const unsigned int c1)

  Return a shared-memory image referencing a set of channels (c0->c1) of the instance image.

- const CImg< T > **get_shared_channels** (const unsigned int c0, const unsigned int c1) const
- CImg< T > get_shared_channel (const unsigned int c0)

  Return a shared-memory image referencing one channel c0 of the instance image.

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

  Return a shared version of the instance image.

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

  Split image into a list.

- CImgList< T > **get_split** (const T value, const bool keep_values, const bool shared, const char axis='y') const
- template<typename t >
  CImg< T > & append (const CImg< t > &img, const char axis='x', const char align='p')

  Append an image.

- CImg< T > & **append** (const CImg< T > &img, const char axis='x', const char align='p')
- template<typename t >
  CImg< typename cimg::superset< T, t >::type > **get_append** (const CImg< T > &img, const char axis='x', const char align='p') const
- CImg< T > **get_append** (const CImg< T > &img, const char axis='x', const char align='p') const

Filtering / Transforms

- template<typename t >
  CImg< T > & correlate (const CImg< t > &mask, const unsigned int border_conditions=1, const bool is_normalized=false)

  Compute the correlation of the instance image by a mask.

- template<typename t >
  CImg< typename cimg::superset2< T, t, float >::type > **get_correlate** (const CImg< t > &mask, const unsigned int border_conditions=1, const bool is_normalized=false) const
- template<typename t >
  CImg< T > & convolve (const CImg< t > &mask, const unsigned int border_conditions=1, const bool is_normalized=false)

  Compute the convolution of the image by a mask.

- template<typename t >
  CImg< typename cimg::superset2< T, t, float >::type > **get_convolve** (const CImg< t > &mask, const unsigned int border_conditions=1, const bool is_normalized=false) const
- template<typename t >
  CImg< T > & erode (const CImg< t > &mask, const unsigned int border_conditions=1, const bool is_normalized=false)

  Return the erosion of the image by a structuring element.

- template<typename t >
  CImg< typename cimg::superset< T, t >::type > **get_erode** (const CImg< t > &mask, const unsigned int border_conditions=1, const bool is_normalized=false) const
- CImg< T > & erode (const unsigned int sx, const unsigned int sy, const unsigned int sz=1)

  Erode the image by a rectangular structuring element of size sx,sy,sz.

- CImg< T > **get_erode** (const unsigned int sx, const unsigned int sy, const unsigned int sz=1) const
- CImg< T > & erode (const unsigned int s)

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

- CImg< T > **get_erode** (const unsigned int s) const
- template<typename t >
  CImg< T > & dilate (const CImg< t > &mask, const unsigned int border_conditions=1, const bool is_normalized=false)

  Dilate the image by a structuring element.

- template<typename t >
  CImg< typename cimg::superset< T, t >::type > **get_dilate** (const CImg< t > &mask, const unsigned int border_conditions=1, const bool is_normalized=false) const
- CImg< T > & dilate (const unsigned int sx, const unsigned int sy, const unsigned int sz=1)

  Dilate the image by a rectangular structuring element of size sx,sy,sz.

- CImg< T > **get_dilate** (const unsigned int sx, const unsigned int sy, const unsigned int sz=1) const
- CImg< T > & dilate (const unsigned int s)

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

- CImg< T > **get_dilate** (const unsigned int s) const
- template<typename t >
  CImg< T > & watershed (const CImg< t > &priority, const bool fill_lines=true)

  Compute the watershed transform, from an instance image of non-zero labels.

- template<typename t >
  CImg< T > **get_watershed** (const CImg< t > &priority, const bool fill_lines=true) const
- CImg< T > & deriche (const float sigma, const int order=0, const char axis='x', const bool cond=true)

  Compute the result of the Deriche filter.

- CImg< Tfloat > **get_deriche** (const float sigma, const int order=0, const char axis='x', const bool cond=true) const
- CImg< T > & blur (const float sigmax, const float sigmay, const float sigmaz, const bool cond=true)

  Return a blurred version of the image, using a Canny-Deriche filter.

- CImg< Tfloat > **get_blur** (const float sigmax, const float sigmay, const float sigmaz, const bool cond=true) const
- CImg< T > & blur (const float sigma, const bool cond=true)

  Return a blurred version of the image, using a Canny-Deriche filter.

- CImg< Tfloat > **get_blur** (const float sigma, const bool cond=true) const
- 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 fast_approx=1)

  Blur the image anisotropically following 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 fast_approx=true) const
- 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 fast_approx=true)

  Blur an image following in an anisotropic 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 fast_approx=true) const
- CImg< T > & blur_bilateral (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 an image using the bilateral filter.

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

  Blur an image using the bilateral filter.

- CImg< T > **get_blur_bilateral** (const float sigma_s, const float sigma_r, const int bgrid_s=-33, const int bgrid_r=32, const bool interpolation_type=true) const
- 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 fast_approx=true)

  Blur an image in its 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 fast_approx=true) const
- CImg< T > & blur_median (const unsigned int n)

  Apply a median filter.

- CImg< T > **get_blur_median** (const unsigned int n) const
- 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 using anisotropic shock filters or inverse diffusion.

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

  Compute the list of images, corresponding to the XY-gradients of an image.

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

  Get components of the Hessian matrix of an image.

- CImg< T > & laplacian ()

  Compute the laplacian of the instance image.

- CImg< Tfloat > **get_laplacian** () const
- CImg< T > & structure_tensors (const unsigned int scheme=1)

  Compute the structure tensor field of an image.

- CImg< Tfloat > **get_structure_tensors** (const unsigned int scheme=1) const
- CImg< T > & edge_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)

  Get a diffusion tensor for edge-preserving anisotropic smoothing of an image.

- CImg< Tfloat > **get_edge_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
- CImg< T > & displacement (const CImg< T > &target, const float smooth=0.1f, const float precision=0.1f, const unsigned int nb_scales=0, const unsigned int iteration_max=1000, const bool backward=true)

  Estimate a displacement field between instance image and given target image.

- CImg< Tfloat > **get_displacement** (const CImg< T > &target, const float smoothness=0.1f, const float precision=0.1f, const unsigned int nb_scales=0, const unsigned int iteration_max=1000, const bool backward=true) const
- CImg< T > & distance (const T isovalue, const float sizex=1, const float sizey=1, const float sizez=1, const bool compute_sqrt=true)

  Compute the Euclidean distance map to a shape of specified isovalue.

- CImg< floatT > **get_distance** (const T isovalue, const float sizex=1, const float sizey=1, const float sizez=1, const bool compute_sqrt=true) const
- CImg< T > & distance_eikonal (const unsigned int nb_iterations, const float band_size=0, const float time_step=0.5f)

  Compute distance function from 0-valued isophotes by the application of an Eikonal PDE.

- CImg< Tfloat > **get_distance_eikonal** (const unsigned int nb_iterations, const float band_size=0, const float time_step=0.5f) const
- CImg< T > & haar (const char axis, const bool invert=false, const unsigned int nb_scales=1)

  Compute the Haar multiscale wavelet transform (monodimensional version).

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

  Compute the Haar multiscale wavelet transform.

- CImg< Tfloat > **get_haar** (const bool invert=false, const unsigned int nb_scales=1) const
- CImgList< Tfloat > get_FFT (const char axis, const bool invert=false) const

  Compute a 1d Fast Fourier Transform, along a specified axis.

- CImgList< Tfloat > get_FFT (const bool invert=false) const

  Compute a n-d Fast-Fourier Transform.

- static void FFT (CImg< T > &real, CImg< T > &imag, const char axis, const bool invert=false)

  Compute a 1d Fast Fourier Transform, along a specified axis.

- static void FFT (CImg< T > &real, CImg< T > &imag, const bool invert=false)

  Compute a n-d Fast Fourier Transform.

3d Objects Management

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

  Shift a 3d object.

- CImg< Tfloat > **get_shift_object3d** (const float tx, const float ty=0, const float tz=0) const
- CImg< T > & shift_object3d ()

  Shift a 3d object so that it becomes centered.

- CImg< Tfloat > **get_shift_object3d** () const
- CImg< T > & resize_object3d (const float sx, const float sy=-100, const float sz=-100)

  Resize a 3d object.

- CImg< Tfloat > **get_resize_object3d** (const float sx, const float sy=-100, const float sz=-100) const
- CImg< T > resize_object3d () const

  Resize a 3d object so that its max dimension if one.
- CImg< Tfloat > **get_resize_object3d** () const
- template<typename tf , typename tp , typename tff >
  CImg< T > & append_object3d (CImgList< tf > &primitives, const CImg< tp > &obj_vertices, const CImgList< tff > &obj_primitives)

  Append a 3d object to another one.
- 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

  Create and return a 3d elevation of the instance image.
- template<typename tf >
  CImg< floatT > get_isoline3d (CImgList< tf > &primitives, const float isovalue, const int size_x=-100, const int size_y=-100) const

  Create and return a isoline of the instance image 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

  Create and return a isosurface of the instance image 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)

  Convert a 3d object into a CImg3d.
- template<typename tp , typename tc >
  CImg< T > & **object3dtoCImg3d** (const CImgList< tp > &primitives, const CImgList< tc > &colors)
- template<typename tp >
  CImg< T > & **object3dtoCImg3d** (const CImgList< tp > &primitives)
- CImg< T > & **object3dtoCImg3d** ()
- template<typename tp , typename tc , typename to >
  CImg< floatT > **get_object3dtoCImg3d** (const CImgList< tp > &primitives, const CImgList< tc > &colors, const to &opacities) const
- template<typename tp , typename tc >
  CImg< floatT > **get_object3dtoCImg3d** (const CImgList< tp > &primitives, const CImgList< tc > &colors) const
- template<typename tp >
  CImg< floatT > **get_object3dtoCImg3d** (const CImgList< tp > &primitives) const

- CImg< floatT > **get_object3dtoCImg3d** () const
- template<typename tp , typename tc , typename to >
  CImg< T > get_CImg3dtoobject3d (CImgList< tp > &primitives, CImgList< tc > &colors, CImgList< to > &opacities) const

  Convert a CImg3d (one-column image) into a 3d object.

- template<typename tp , typename tc , typename to >
  CImg< T > & **CImg3dtoobject3d** (CImgList< tp > &primitives, CImgList< tc > &colors, CImgList< to > &opacities)
- 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)

  Get elevation3d of a function.

- 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 sizex=256, const int sizey=256)
- 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 sizex=256, const int sizey=256)

  Get isoline 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 sizex=256, const int sizey=256)
- 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)

  Get isosurface 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)
- 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)

  Create and return 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)

  Create and return 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)

  Create and return 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)

  Create and return 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)

  Create and return a 3d XY-plane.

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

  Create and return a 3d sphere.

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

  Create and return a 3d ellipsoid.

Drawing Functions

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

  Draw a 2d colored point (pixel).

- 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 colored point (voxel).

- 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 colored 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 colored 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 colored 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 2d textured 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 2d textured 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 2d textured line, with z-buffering and perspective correction.

- 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 colored lines in the instance image.

- 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 colored arrow in the instance image.

- 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 cubic spline curve in the instance image.

- 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 cubic spline curve in the instance image (for volumetric images).

- 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 cubic spline curve in the instance image.

- 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 close_set=false, const float precision=4, const unsigned int pattern=~0U, const bool init_hatch=true)
- template<typename tp , typename tc >
  CImg< T > & draw_spline (const CImg< tp > &points, const tc *const color, const float opacity=1, const bool close_set=false, const float precision=4, const unsigned int pattern=~0U, const bool init_hatch=true)

  Draw a set of consecutive colored splines in the instance image.

- 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 2d filled colored 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 2d outlined colored 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 2d filled colored 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 2d Gouraud-shaded colored 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 2d Gouraud-shaded colored triangle, with z-buffering.

- 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 colored triangle with interpolated colors.

- 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 2d textured 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 2d textured triangle, with z-buffering and perspective correction.

- 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 2d Pseudo-Phong-shaded 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 2d Pseudo-Phong-shaded 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 2d Gouraud-shaded textured 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 2d Gouraud-shaded 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 brightness0, const float brightness1, const float brightness2, const float opacity=1)

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

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

- 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 4d filled rectangle in the instance image, at coordinates (x0,y0,z0,c0)-(x1,y1,z1,c1).

- 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 3d filled colored rectangle in the instance image, at coordinates (x0,y0,z0)-(x1,y1,z1).

- 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 a 3d outlined colored rectangle in the instance image.

- 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 2d filled colored rectangle in the instance image, at coordinates (x0,y0)-(x1,y1).

- 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 2d outlined colored rectangle.

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

  Draw a filled polygon in the instance image.

- 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 polygon in the instance image.

- 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 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)

  Draw an outlined circle.

- 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 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 ellipse.

- 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 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 ellipse.

- 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)
- 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.

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

  Draw an image.

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

  Draw an image.

- template<typename t >
  CImg< T > & draw_image (const CImg< t > &sprite, const float opacity=1)

  Draw an image.

- 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_valmax=1)

  Draw a sprite image in the instance image (masked version).

- 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_valmax=1)

  Draw an image.

- 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_valmax=1)

  Draw an image.

- 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_valmax=1)

  Draw an image.

- template<typename ti , typename tm >
  CImg< T > & draw_image (const CImg< ti > &sprite, const CImg< tm > &mask, const float opacity=1, const float mask_valmax=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.

- 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,...)
- 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,...)
- 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.

- 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,...)

- 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,...)
- 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 arrows=true, const unsigned int pattern=~0U)

  Draw a vector field in the instance image, using a colormap.

- 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 arrows=true, const unsigned int pattern=~0U)

  Draw a vector field in the instance image, using a colormap.

- template<typename t , typename tc >
  CImg< T > & draw_axis (const CImg< t > &xvalues, const int y, const tc *const color, const float opacity=1, const unsigned int pattern=~0U)

  Draw a labeled horizontal axis on the instance image.

- template<typename t , typename tc >
  CImg< T > & draw_axis (const int x, const CImg< t > &yvalues, const tc *const color, const float opacity=1, const unsigned int pattern=~0U)

  Draw a labeled vertical axis on the instance image.

- template<typename tx , typename ty , typename tc >
  CImg< T > & draw_axes (const CImg< tx > &xvalues, const CImg< ty > &yvalues, const tc *const color, const float opacity=1, const unsigned int patternx=~0U, const unsigned int patterny=~0U)

  Draw a labeled horizontal+vertical axis on the instance image.

- 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 patternx=~0U, const unsigned int patterny=~0U)

  Draw a labeled horizontal+vertical axis on the instance image.

- template<typename tx , typename ty , typename tc >
  CImg< T > & draw_grid (const CImg< tx > &xvalues, const CImg< ty > &yvalues, const tc *const color, const float opacity=1, const unsigned int patternx=~0U, const unsigned int patterny=~0U)

  Draw grid.

- template<typename tc >
  CImg< T > & draw_grid (const float deltax, const float deltay, const float offsetx, const float offsety, const bool invertx, const bool inverty, const tc *const color, const float opacity=1, const unsigned int patternx=~0U, const unsigned int patterny=~0U)

  Draw grid.

- 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 a 1d graph on the instance image.

- 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 high_connexity=false)

  Draw a 3d filled region starting from a point (x,y,\ z) in the instance image.

- 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 high_connexity=false)

  Draw a 3d filled region starting from a point (x,y,\ z) in the instance image.

- 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 high_connexity=false)

  Draw a 2d filled region starting from a point (x,y) in the instance image.

- CImg< T > & draw_plasma (const int x0, const int y0, const int x1, const int y1, const float alpha=1, const float beta=1, const float opacity=1)

  Draw a plasma random texture.

- CImg< T > & draw_plasma (const float alpha=1, const float beta=1, const float opacity=1)

  Draw a plasma random texture.

- template<typename tc >
  CImg< T > & draw_mandelbrot (const int x0, const int y0, const int x1, const int y1, const CImg< tc > &color_palette, 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 normalized_iteration=false, const bool julia_set=false, const double paramr=0, const double parami=0)

  Draw a quadratic Mandelbrot or Julia fractal set, computed using the Escape Time Algorithm.

- template<typename tc >
  CImg< T > & draw_mandelbrot (const CImg< tc > &color_palette, 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 normalized_iteration=false, const bool julia_set=false, const double paramr=0, const double parami=0)

  Draw a quadratic Mandelbrot or Julia fractal set, computed using the Escape Time Algorithm.

- 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 in the instance image.

- 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 an anisotropic 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 an anisotropic 2d gaussian function.

- 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 an isotropic 2d gaussian function.

- 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 an anisotropic 3d gaussian function.

- 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 an isotropic 3d gaussian function.

- 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 double_sided=false, const float focale=500, const float lightx=0, const float lighty=0, const float lightz=-5e8, const float specular_light=0.2f, const float specular_shine=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 double_sided, const float focale, const float lightx, const float lighty, const float lightz, const float specular_light, const float specular_shine, CImg< tz > &zbuffer)
- 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 double_sided=false, const float focale=500, const float lightx=0, const float lighty=0, const float lightz=-5e8, const float specular_light=0.2f, const float specular_shine=0.1f)
- 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 double_sided, const float focale, const float lightx, const float lighty, const float lightz, const float specular_light, const float specular_shine, CImg< tz > &zbuffer)
- 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 double_sided=false, const float focale=500, const float lightx=0, const float lighty=0, const float lightz=-5e8, const float specular_light=0.2f, const float specular_shine=0.1f)

  Draw a 3d object.

- 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 double_sided, const float focale, const float lightx, const float lighty, const float lightz, const float specular_light, const float specular_shine, CImg< tz > &zbuffer)

Data Input

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

  Simple interface to select a shape from an image.

- CImg< T > & select (const char *const title, const int select_type=2, unsigned int *const XYZ=0, const unsigned char *const color=0)

  Simple interface to select a shape from an image.

- CImg< intT > get_select (CImgDisplay &disp, const int select_type=2, unsigned int *const XYZ=0, const unsigned char *const color=0) const

  Simple interface to select a shape from an image.

- CImg< intT > get_select (const char *const title, const int select_type=2, unsigned int *const XYZ=0, const unsigned char *const color=0) const

  Simple interface to select a shape from an image.

- 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 an image from a file.

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

  Load an image from an ASCII file.

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

  Load an image from an ASCII file.

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

  Load an image from a DLM file.

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

  Load an image from a DLM file.

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

  Load an image from a BMP file.

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

  Load an image from a BMP file.

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

  Load an image from a JPEG file.

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

  Load an image from a JPEG file.

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

  Load an image from a file, using Magick++ library.

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

  Load an image from a PNG file.

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

  Load an image from a PNG file.

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

  Load an image from a PNM file.

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

  Load an image from a PNM file.

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

  Load an image from a PFM file.

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

  Load an image from a PFM file.

- CImg< T > & load_rgb (const char *const filename, const unsigned int dimw, const unsigned int dimh=1)

  Load an image from a RGB file.

- CImg< T > & load_rgb (std::FILE *const file, const unsigned int dimw, const unsigned int dimh=1)

  Load an image from a RGB file.

- CImg< T > & load_rgba (const char *const filename, const unsigned int dimw, const unsigned int dimh=1)

  Load an image from a RGBA file.

- CImg< T > & load_rgba (std::FILE *const file, const unsigned int dimw, const unsigned int dimh=1)

  Load an image from a RGBA file.

- 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 an image from a TIFF file.

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

  Load an image from an ANALYZE7.5/NIFTI file.

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

  Load an image from an ANALYZE7.5/NIFTI file.

- CImg< T > & load_cimg (const char *const filename, const char axis='z', const char align='p')

  Load an image (list) from a .cimg file.

- CImg< T > & load_cimg (std::FILE *const file, const char axis='z', const char align='p')

  Load an image (list) from a .cimg file.

- 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 char align='p')

  Load a sub-image (list) from 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 char align='p')

  Load a sub-image (list) from a non-compressed .cimg file.

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

  Load an image from an INRIMAGE-4 file.

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

  Load an image from an INRIMAGE-4 file.

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

  Load an image from a EXR file.

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

  Load an image from a PANDORE file.

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

  Load an image from a PANDORE file.

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

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

- CImg< T > & load_raw (const char *const filename, const unsigned int sizex, const unsigned int sizey=1, const unsigned int sizez=1, const unsigned int sizev=1, const bool multiplexed=false, const bool invert_endianness=false)

  Load an image from a .RAW file.

- CImg< T > & load_raw (std::FILE *const file, const unsigned int sizex, const unsigned int sizey=1, const unsigned int sizez=1, const unsigned int sizev=1, const bool multiplexed=false, const bool invert_endianness=false)

  Load an image from a .RAW file.

- 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 char align='p')

  Load a video sequence using FFMPEG av's libraries.

- CImg< T > & load_yuv (const char *const filename, const unsigned int sizex, const unsigned int sizey=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', const char align='p')

  Load an image sequence from a YUV file.

- CImg< T > & load_yuv (std::FILE *const file, const unsigned int sizex, const unsigned int sizey=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', const char align='p')

  Load an image sequence from a YUV file.

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

  Load a 3d object from a .OFF file.

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

  Load a 3d object from a .OFF file.

- CImg< T > & load_ffmpeg_external (const char *const filename, const char axis='z', const char align='p')

  Load a video sequence using FFMPEG's external tool 'ffmpeg'.

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

  Load an image using GraphicsMagick's external tool 'gm'.

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

  Load a gzipped image file, using external tool 'gunzip'.

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

  Load an image using ImageMagick's external tool 'convert'.

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

  Load a DICOM image file, using XMedcon's external tool 'medcon'.

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

  Load a RAW Color Camera image file, using external tool 'dcraw'.

- CImg< T > & load_camera (const int camera_index=-1, const unsigned int skip_frames=0, const bool release_camera=false)

  Load an image from a camera stream, using OpenCV.

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

  Load an image using ImageMagick's or GraphicsMagick's executables.

- static CImg< T > **get_load** (const char *const filename)
- static CImg< T > **get_load_ascii** (const char *const filename)
- static CImg< T > **get_load_ascii** (std::FILE *const file)
- static CImg< T > **get_load_dlm** (const char *const filename)
- static CImg< T > **get_load_dlm** (std::FILE *const file)
- static CImg< T > **get_load_bmp** (const char *const filename)
- static CImg< T > **get_load_bmp** (std::FILE *const file)
- static CImg< T > **get_load_jpeg** (const char *const filename)
- static CImg< T > **get_load_jpeg** (std::FILE *const file)
- static CImg< T > **get_load_magick** (const char *const filename)
- static CImg< T > **get_load_png** (const char *const filename)
- static CImg< T > **get_load_png** (std::FILE *const file)
- static CImg< T > **get_load_pnm** (const char *const filename)
- static CImg< T > **get_load_pnm** (std::FILE *const file)
- static CImg< T > **get_load_pfm** (const char *const filename)
- static CImg< T > **get_load_pfm** (std::FILE *const file)
- static CImg< T > **get_load_rgb** (const char *const filename, const unsigned int dimw, const unsigned int dimh=1)
- static CImg< T > **get_load_rgb** (std::FILE *const file, const unsigned int dimw, const unsigned int dimh=1)
- static CImg< T > **get_load_rgba** (const char *const filename, const unsigned int dimw, const unsigned int dimh=1)
- static CImg< T > **get_load_rgba** (std::FILE *const file, const unsigned int dimw, const unsigned int dimh=1)
- 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)
- static CImg< T > **get_load_analyze** (const char *const filename, float *const voxsize=0)
- static CImg< T > **get_load_analyze** (std::FILE *const file, float *const voxsize=0)
- static CImg< T > **get_load_cimg** (const char *const filename, const char axis='z', const char align='p')
- static CImg< T > **get_load_cimg** (std::FILE *const file, const char axis='z', const char align='p')
- 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 char align='p')
- 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 char align='p')
- static CImg< T > **get_load_inr** (const char *const filename, float *const voxsize=0)
- static CImg< T > **get_load_inr** (std::FILE *const file, float *voxsize=0)
- static CImg< T > **get_load_exr** (const char *const filename)
- static CImg< T > **get_load_pandore** (const char *const filename)
- static CImg< T > **get_load_pandore** (std::FILE *const file)
- static CImg< T > **get_load_parrec** (const char *const filename, const char axis='c', const char align='p')
- static CImg< T > **get_load_raw** (const char *const filename, const unsigned int sizex, const unsigned int sizey=1, const unsigned int sizez=1, const unsigned int sizev=1, const bool multiplexed=false, const bool invert_endianness=false)
- static CImg< T > **get_load_raw** (std::FILE *const file, const unsigned int sizex, const unsigned int sizey=1, const unsigned int sizez=1, const unsigned int sizev=1, const bool multiplexed=false, const bool invert_endianness=false)

- 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 char align='p')
- static CImg< T > **get_load_yuv** (const char *const filename, const unsigned int sizex, const unsigned int sizey=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', const char align='p')
- static CImg< T > **get_load_yuv** (std::FILE *const file, const unsigned int sizex, const unsigned int sizey=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', const char align='p')
- template<typename tf , typename tc >
  static CImg< T > **get_load_off** (const char *const filename, CImgList< tf > &primitives, CImgList< tc > &colors)
- template<typename tf , typename tc >
  static CImg< T > **get_load_off** (std::FILE *const file, CImgList< tf > &primitives, CImgList< tc > &colors)
- static CImg< T > **get_load_ffmpeg_external** (const char *const filename, const char axis='z', const char align='p')
- static CImg< T > **get_load_graphicsmagick_external** (const char *const filename)
- static CImg< T > **get_load_gzip_external** (const char *const filename)
- static CImg< T > **get_load_imagemagick_external** (const char *const filename)
- static CImg< T > **get_load_medcon_external** (const char *const filename)
- static CImg< T > **get_load_dcraw_external** (const char *const filename)
- static CImg< T > **get_load_camera** (const int camera_index=-1, const unsigned int skip_frames=0, const bool release_camera=false)
- static CImg< T > **get_load_other** (const char *const filename)

Data Output

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

  Display informations about the image on the standard error output.

- const CImg< T > & display (CImgDisplay &disp) const

  Display an image into a CImgDisplay window.

- const CImg< T > & display (CImgDisplay &disp, const bool display_info) const

  Display an image in a window with a title `title`, and wait a '_is_closed' or 'keyboard' event.
  .

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

  Display an image in a window with a title `title`, and wait a '_is_closed' or 'keyboard' event.
  .

- 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 double_sided=true, const float focale=500, const float light_x=0, const float light_y=0, const float light_z=-5000, const float specular_light=0.2f, const float specular_shine=0.1f, const bool display_axes=true, float *const pose_matrix=0) const

  High-level interface for displaying a 3d object.

- 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 double_sided=true, const float focale=500, const float light_x=0, const float light_y=0, const float light_z=-5000, const float specular_light=0.2f, const float specular_shine=0.1f, const bool display_axes=true, float *const pose_matrix=0) const

  High-level interface for displaying a 3d object.

- 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 double_sided=true, const float focale=500, const float light_x=0, const float light_y=0, const float light_z=-5000, const float specular_light=0.2f, const float specular_shine=0.1f, const bool display_axes=true, float *const pose_matrix=0) const

  High-level interface for displaying a 3d object.

- 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 double_sided=true, const float focale=500, const float light_x=0, const float light_y=0, const float light_z=-5000, const float specular_light=0.2f, const float specular_shine=0.1f, const bool display_axes=true, float *const pose_matrix=0) const

  High-level interface for displaying a 3d object.

- 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 double_sided=true, const float focale=500, const float light_x=0, const float light_y=0, const float light_z=-5000, const float specular_light=0.2f, const float specular_shine=0.1f, const bool display_axes=true, float *const pose_matrix=0) const

  High-level interface for displaying a 3d object.

- 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 double_sided=true, const float focale=500, const float light_x=0, const float light_y=0, const float light_z=-5000, const float specular_light=0.2f, const float specular_shine=0.1f, const bool display_axes=true, float *const pose_matrix=0) const

  High-level interface for displaying a 3d object.

- 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 double_sided=true, const float focale=500, const float light_x=0, const float light_y=0, const float light_z=-5000, const float specular_light=0.2f, const float specular_shine=0.1f, const bool display_axes=true, float *const pose_matrix=0) const

  High-level interface for displaying a 3d object.

- 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 double_-sided=true, const float focale=500, const float light_x=0, const float light_y=0, const float light_z=-5000, const float specular_light=0.2f, const float specular_shine=0.1f, const bool display_axes=true, float *const pose_matrix=0) const

High-level interface for displaying a 3d object.

- 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

  High-level interface for displaying a graph.

- 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

  High-level interface for displaying a graph.

- const CImg< T > & save (const char *const filename, const int number=-1) const

  Save the image as a file.

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

  Save the image as an ASCII file (ASCII Raw + simple header).

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

  Save the image as an ASCII file (ASCII Raw + simple header).

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

  Save the image as a CPP source file.

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

  Save the image as a CPP source file.

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

  Save the image as a DLM file.

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

  Save the image as a DLM file.

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

  Save the image as a BMP file.

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

  Save the image as a BMP file.

- const CImg< T > & save_jpeg (const char *const filename, const unsigned int quality=100) const

  Save a file in JPEG format.

- const CImg< T > & save_jpeg (std::FILE *const file, const unsigned int quality=100) const

  Save a file in JPEG format.

- const CImg< T > & save_magick (const char *const filename, const unsigned int bytes_per_-pixel=0) const

Save the image using built-in ImageMagick++ library.

- const CImg< T > & save_png (const char *const filename, const unsigned int bytes_per_pixel=0) const

  Save a file in PNG format.

- const CImg< T > & save_png (std::FILE *const file, const unsigned int bytes_per_pixel=0) const

  Save a file in PNG format.

- const CImg< T > & save_pnm (const char *const filename, const unsigned int bytes_per_pixel=0) const

  Save the image as a PNM file.

- const CImg< T > & save_pnm (std::FILE *const file, const unsigned int bytes_per_pixel=0) const

  Save the image as a PNM file.

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

  Save the image as a PFM file.

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

  Save the image as a PFM file.

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

  Save the image as a RGB file.

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

  Save the image as a RGB file.

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

  Save the image as a RGBA file.

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

  Save the image as a RGBA file.

- const CImg< T > & save_tiff (const char *const filename, const unsigned int compression=0) const

  Save a file in TIFF format.

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

  Save the image as an ANALYZE7.5 or NIFTI file.

- const CImg< T > & save_cimg (const char *const filename, const bool compression=false) const

  Save the image as a .cimg file.

- const CImg< T > & **save_cimg** (std::FILE *const file, const bool compression=false) const
- 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

  Insert the image into an existing .cimg file, at specified coordinates.

- 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

Insert the image into an existing .cimg file, at specified coordinates.

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

  Save the image as an INRIMAGE-4 file.

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

  Save the image as an INRIMAGE-4 file.

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

  Save the image as a EXR file.

- const CImg< T > & save_pandore (const char *const filename, const unsigned int colorspace=0) const

  Save the image as a PANDORE-5 file.

- const CImg< T > & save_pandore (std::FILE *const file, const unsigned int colorspace=0) const

  Save the image as a PANDORE-5 file.

- const CImg< T > & save_raw (const char *const filename, const bool multiplexed=false) const

  Save the image as a RAW file.

- const CImg< T > & save_raw (std::FILE *const file, const bool multiplexed=false) const

  Save the image as a RAW file.

- const CImg< T > & save_ffmpeg (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int fps=25, const unsigned int bitrate=2048) const

  Save the image as a video sequence file, using FFMPEG library.

- const CImg< T > & save_yuv (const char *const filename, const bool rgb2yuv=true) const

  Save the image as a YUV video sequence file.

- const CImg< T > & save_yuv (std::FILE *const file, const bool rgb2yuv=true) const

  Save the image as a YUV video sequence file.

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

  Save OFF files.

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

  Save OFF files.

- const CImg< T > & save_ffmpeg_external (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const char *const codec="mpeg2video", const unsigned int fps=25, const unsigned int bitrate=2048) const

  Save the image as a video sequence file, using the external tool 'ffmpeg'.

- const CImg< T > & save_graphicsmagick_external (const char *const filename, const unsigned int quality=100) const

  Save the image using GraphicsMagick's gm.

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

  Save an image as a gzipped file, using external tool 'gzip'.

- const CImg< T > & save_imagemagick_external (const char *const filename, const unsigned int quality=100) const

  Save the image using ImageMagick's convert.

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

  Save an image as a Dicom file (need '(X)Medcon' : `http://xmedcon.sourceforge.net` *).*

- 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 an empty .cimg file with specified dimensions.

- 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 an empty .cimg file with specified dimensions.

- static CImg< T > **logo40x38** ()

8.1.1 Detailed Description

template<typename T> struct cimg_library::CImg< T >

Class representing an image (up to 4 dimensions wide), each pixel being of type `T`. This is the main class of the CImg Library. It declares and constructs an image, allows access to its pixel values, and is able to perform various image operations.

Image representation

A CImg image is defined as an instance of the container CImg<T>, which contains a regular grid of pixels, each pixel value being of type `T`. The image grid can have up to 4 dimensions : width, height, depth and number of channels. Usually, the three first dimensions are used to describe spatial coordinates `(x,y,z)`, while the number of channels is rather used as a vector-valued dimension (it may describe the R,G,B color channels for instance). If you need a fifth dimension, you can use image lists CImgList<T> rather than simple images CImg<T>.

Thus, the CImg<T> class is able to represent volumetric images of vector-valued pixels, as well as images with less dimensions (1d scalar signal, 2d color images, ...). Most member functions of the class CImg<T> are designed to handle this maximum case of (3+1) dimensions.

Concerning the pixel value type `T` : fully supported template types are the basic C++ types : `unsigned char, char, short, unsigned int, int, unsigned long, long, float, double, ...` . Typically, fast image display can be done using `CImg<unsigned`

char> images, while complex image processing algorithms may be rather coded using CImg<float> or CImg<double> images that have floating-point pixel values. The default value for the template T is float. Using your own template types may be possible. However, you will certainly have to define the complete set of arithmetic and logical operators for your class.

Image structure

The CImg<T> structure contains *six* fields :

- width defines the number of *columns* of the image (size along the X-axis).
- height defines the number of *rows* of the image (size along the Y-axis).
- depth defines the number of *slices* of the image (size along the Z-axis).
- spectrum defines the number of *channels* of the image (size along the C-axis).
- data defines a *pointer* to the *pixel data* (of type T).
- is_shared is a boolean that tells if the memory buffer data is shared with another image.

You can access these fields publicly although it is recommended to use the dedicated functions width(), height(), depth(), spectrum() and ptr() to do so. Image dimensions are not limited to a specific range (as long as you got enough available memory). A value of *1* usually means that the corresponding dimension is *flat*. If one of the dimensions is *0*, or if the data pointer is null, the image is considered as *empty*. Empty images should not contain any pixel data and thus, will not be processed by CImg member functions (a CImgInstanceException will be thrown instead). Pixel data are stored in memory, in a non interlaced mode (See How pixel data are stored with CImg.).

Image declaration and construction

Declaring an image can be done by using one of the several available constructors. Here is a list of the most used :

- Construct images from arbitrary dimensions :
  - CImg<char> img; declares an empty image.
  - CImg<unsigned char> img(128,128); declares a 128x128 greyscale image with unsigned char pixel values.
  - CImg<double> img(3,3); declares a 3x3 matrix with double coefficients.
  - CImg<unsigned char> img(256,256,1,3); declares a 256x256x1x3 (color) image (colors are stored as an image with three channels).
  - CImg<double> img(128,128,128); declares a 128x128x128 volumetric and greyscale image (with double pixel values).
  - CImg<> img(128,128,128,3); declares a 128x128x128 volumetric color image (with float pixels, which is the default value of the template parameter T).
  - **Note** : images pixels are **not automatically initialized to 0**. You may use the function fill() to do it, or use the specific constructor taking 5 parameters like this : CImg<> img(128,128,128,3,0); declares a 128x128x128 volumetric color image with all pixel values to 0.

- Construct images from filenames :
  - `CImg<unsigned char> img("image.jpg");` reads a JPEG color image from the file "image.jpg".
  - `CImg<float> img("analyze.hdr");` reads a volumetric image (ANALYZE7.5 format) from the file "analyze.hdr".
  - **Note** : You need to install `ImageMagick` to be able to read common compressed image formats (JPG,PNG, ...) (See Files IO in CImg.).

- Construct images from C-style arrays :
  - `CImg<int> img(data_buffer,256,256);` constructs a 256x256 greyscale image from a `int*` buffer `data_buffer` (of size 256x256=65536).
  - `CImg<unsigned char> img(data_buffer,256,256,1,3,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

Iterator type for CImg<T>.

Remarks

- An `iterator` is a `T*` pointer (address of a pixel value in the pixel buffer).
- Iterators are not directly used in CImg functions, they have been introduced for compatibility with the STL.

8.1.2.2 typedef const T* const_iterator

Const iterator type for CImg<T>.

Remarks

- A `const_iterator` is a `const T*` pointer (address of a pixel value in the pixel buffer).
- Iterators are not directly used in CImg functions, they have been introduced for compatibility with the STL.

8.1.3 Constructor & Destructor Documentation

8.1.3.1 ~CImg ( )

Destructor.

The destructor destroys the instance image.

Remarks

- Destructing an empty or shared image does nothing.
- Otherwise, all memory used to store the pixel data of the instance image is freed.
- When destroying a non-shared image, be sure that every shared instances of the same image are also destroyed to avoid further access to desallocated memory buffers.

8.1.3.2 CImg ( )

Default constructor.

The default constructor creates an empty instance image.

Remarks

- An empty image does not contain any data and has all of its dimensions width, height, depth, spectrum set to 0 as well as its pointer to the pixel buffer data.
- An empty image is non-shared.

8.1.3.3 CImg (const unsigned int *dx*, const unsigned int *dy* = *1*, const unsigned int *dz* = *1*, const unsigned int *dc* = *1*) [explicit]

Constructs a new image with given size (dx,dy,dz,dc).

This constructors create an instance image of size (dx,dy,dz,dc) with pixels of type T.

Parameters

dx Desired size along the X-axis, i.e. the width of the image.

dy Desired size along the Y-axis, i.e. the height of the image.

dz Desired size along the Z-axis, i.e. the depth of the image.

dc Desired size along the C-axis, i.e. the number of image channels spectrum.

Remarks

- If one of the input dimension dx,dy,dz or dc is set to 0, the created image is empty and all has its dimensions set to 0. No memory for pixel data is then allocated.
- This constructor creates only non-shared images.
- Image pixels allocated by this constructor are **not initialized**. Use the constructor CImg(const unsigned int,const unsigned int,const unsigned int,const unsigned int,const T) to get an image of desired size with pixels set to a particular value.

8.1.3.4 CImg (const unsigned int *dx*, const unsigned int *dy*, const unsigned int *dz*, const unsigned int *dc*, const T *val*)

Construct an image with given size (dx,dy,dz,dc) and with pixel having a default value val.

This constructor creates an instance image of size (dx,dy,dz,dc) with pixels of type T and sets all pixel values of the created instance image to val.

Parameters

dx Desired size along the X-axis, i.e. the width of the image.

dy Desired size along the Y-axis, i.e. the height of the image.

dz Desired size along the Z-axis, i.e. the depth of the image.

dc Desired size along the C-axis, i.e. the number of image channels spectrum.

val Default value for image pixels.

Remarks

- This constructor has the same properties as CImg(const unsigned int,const unsigned int,const unsigned int,const unsigned int).

8.1.3.5 CImg (const t *const *data_buffer*, const unsigned int *dx*, const unsigned int *dy* = *1*, const unsigned int *dz* = *1*, const unsigned int *dc* = *1*, const bool *shared* = *false*)

Construct an image from a raw memory buffer.

This constructor creates an instance image of size (dx,dy,dz,dc) and fill its pixel buffer by copying data values from the input raw pixel buffer data_buffer.

8.1.3.6 CImg (const char *const *filename*) [explicit]

Construct an image from an image file.

This constructor creates an instance image by reading it from a file.

Parameters

filename Filename of the image file.

Remarks

- The image format is deduced from the filename only by looking for the filename extension i.e. without analyzing the file itself.
- Recognized image formats depend on the tools installed on your system or the external libraries you use to link your code with. More informations on this topic can be found in cimg_files_io.
- If the filename is not found, a CImgIOException is thrown by this constructor.

8.1.3.7 CImg (const CImg< t > & *img*)

Default copy constructor.

The default copy constructor creates a new instance image having same dimensions (width, height, depth, _spectrum) and same pixel values as the input image img.

Parameters

img The input image to copy.

Remarks

- If the input image `img` is non-shared or have a different template type `t` != `T`, the default copy constructor allocates a new pixel buffer and copy the pixel data of `img` into it. In this case, the pointers data to the pixel buffers of the two images are different and the resulting instance image is non-shared.
- If the input image `img` is shared and has the same template type `t` == `T`, the default copy constructor does not allocate a new pixel buffer and the resulting instance image shares its pixel buffer with the input image `img`, which means that modifying pixels of `img` also modifies the created instance image.
- Copying an image having a different template type `t` != `T` performs a crude static cast conversion of each pixel value from type `t` to type `T`.
- Copying an image having the same template type `t` == `T` is significantly faster.

8.1.3.8 CImg (const CImg< t > & *img*, const bool *shared*)

Advanced copy constructor.

The advanced copy constructor - as the default constructor CImg(const CImg< t >&) - creates a new instance image having same dimensions width, height, depth, spectrum and same pixel values as the input image `img`. But it also decides if the created instance image shares its memory with the input image `img` (if the input parameter `shared` is set to `true`) or not (if the input parameter `shared` is set to `false`).

Parameters

img The input image to copy.

shared Boolean flag that decides if the copy is shared on non-shared.

Remarks

- It is not possible to create a shared copy if the input image `img` is empty or has a different pixel type `t` != `T`.
- If a non-shared copy of the input image `img` is created, a new memory buffer is allocated for pixel data.
- If a shared copy of the input image `img` is created, no extra memory is allocated and the pixel buffer of the instance image is the same as the one used by the input image `img`.

8.1.4 Member Function Documentation

8.1.4.1 CImg<T>& clear ( )

In-place version of the default constructor (STL-compliant name).

This function is strictly equivalent to assign() and has been introduced for having a STL-compatible function name.

8.1.4.2 CImg<T>& assign ( )

In-place version of the default constructor/destructor.

This function replaces the instance image by an empty image.

Remarks

- Memory used by the previous content of the instance image is freed if necessary.
- If the instance image was initially shared, it is replaced by a (non-shared) empty image.
- This function is useful to free memory used by an image that is not of use, but which has been created in the current code scope (i.e. not destroyed yet).

8.1.4.3 CImg<T>& assign (const unsigned int *dx*, const unsigned int *dy* = *1*, const unsigned int *dz* = *1*, const unsigned int *dc* = *1*)

In-place version of the previous constructor.

This function replaces the instance image by a new image of size (dx,dy,dz,dc) with pixels of type T.

Parameters

dx Desired size along the X-axis, i.e. the width of the image.

dy Desired size along the Y-axis, i.e. the height of the image.

dz Desired size along the Z-axis, i.e. the depth of the image.

dc Desired size along the C-axis, i.e. the number of image channels _spectrum.

- If one of the input dimension dx,dy,dz or dc is set to 0, the instance image becomes empty and all has its dimensions set to 0. No memory for pixel data is then allocated.
- Memory buffer used to store previous pixel values is freed if necessary.
- If the instance image is shared, this constructor actually does nothing more than verifying that new and old image dimensions fit.
- Image pixels allocated by this function are **not initialized**. Use the function assign(const unsigned int,const unsigned int,const unsigned int,const unsigned int,const T) to assign an image of desired size with pixels set to a particular value.

8.1.4.4 CImg<T>& assign (const unsigned int *dx*, const unsigned int *dy*, const unsigned int *dz*, const unsigned int *dc*, const T *val*)

In-place version of the previous constructor.

This function replaces the instance image by a new image of size (dx,dy,dz,dc) with pixels of type T and sets all pixel values of the instance image to val.

Parameters

dx Desired size along the X-axis, i.e. the width of the image.

dy Desired size along the Y-axis, i.e. the height of the image.

dz Desired size along the Z-axis, i.e. the depth of the image.

dc Desired size along the C-axis, i.e. the number of image channels _spectrum.

val Default value for image pixels.

Remarks

- This function has the same properties as assign(const unsigned int,const unsigned int,const unsigned int,const unsigned int).

8.1.4.5 CImg<T>& assign (const char *const *filename*)

In-place version of the previous constructor.

This function replaces the instance image by the one that have been read from the given file.

Parameters

filename Filename of the image file.

- The image format is deduced from the filename only by looking for the filename extension i.e. without analyzing the file itself.
- Recognized image formats depend on the tools installed on your system or the external libraries you use to link your code with. More informations on this topic can be found in cimg_files_io.
- If the filename is not found, a CImgIOException is thrown by this constructor.

8.1.4.6 CImg<T>& assign (const CImg< t > & *img*)

In-place version of the default copy constructor.

This function assigns a copy of the input image `img` to the current instance image.

Parameters

img The input image to copy.

Remarks

- If the instance image is non-shared, the content of the input image `img` is copied into a new buffer becoming the new pixel buffer of the instance image, while the old pixel buffer is freed if necessary.
- If the instance image is shared, the content of the input image `img` is copied into the current (shared) pixel buffer of the instance image, modifying then the image referenced by the shared instance image. The instance image still remains shared.

8.1.4.7 CImg<T>& assign (const CImg< t > & *img*, const bool *shared*)

In-place version of the advanced constructor.

This function - as the simpler function assign(const CImg< t >&) - assigns a copy of the input image `img` to the current instance image. But it also decides if the copy is shared (if the input parameter `shared` is set to `true`) or non-shared (if the input parameter `shared` is set to `false`).

Parameters

img The input image to copy.

shared Boolean flag that decides if the copy is shared or non-shared.

Remarks

- It is not possible to assign a shared copy if the input image `img` is empty or has a different pixel type `t` != `T`.
- If a non-shared copy of the input image `img` is assigned, a new memory buffer is allocated for pixel data.
- If a shared copy of the input image `img` is assigned, no extra memory is allocated and the pixel buffer of the instance image is the same as the one used by the input image `img`.

8.1.4.8 CImg<t>& move_to (CImg< t > & *img*)

Move the content of the instance image into another one in a way that memory copies are avoided if possible.

The instance image is always empty after a call to this function.

8.1.4.9 T& operator() (const unsigned int *x*, const unsigned int *y = 0*, const unsigned int *z = 0*, const unsigned int *c = 0*)

Fast access to pixel value for reading or writing.

Parameters

x X-coordinate of the pixel.

y Y-coordinate of the pixel.

z Z-coordinate of the pixel.

v C-coordinate of the pixel.

- If one image dimension is equal to 1, it can be omitted in the coordinate list (see example below).
- If the macro `'cimg_verbosity'>=3`, boundary checking is performed and warning messages may appear (but function performances decrease).

example:

```
 CImg<float> img(100,100,1,3,0);                       // Define a 100x100
color image with float-valued black pixels.
 const float valR = img(10,10,0,0);                    // Read the red comp
onent at coordinates (10,10).
 const float valG = img(10,10,0,1);                    // Read the green co
mponent at coordinates (10,10)
 const float valB = img(10,10,2);                      // Read the blue com
ponent at coordinates (10,10) (Z-coordinate omitted here).
 const float avg = (valR + valG + valB)/3;             // Compute average p
ixel value.
 img(10,10,0) = img(10,10,1) = img(10,10,2) = avg;     // Replace the pixel
 (10,10) by the average grey value.
```

8.1.4.10 CImg<T>& operator= (const T *val*)

Operator=().

Assignment operator. Fill all pixels of the instance image with the same value. The image size is not modified.

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

Operator=().

Assignment operator. If `expression` is a formula or a list of values, the image pixels are filled according to the expression and the image size is not modified. If `expression` is a filename, the image is replaced by the input file data (so image size is modified).

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

Operator=().

Assignement operator. If instance image is non-shared, replace the instance image by a copy of the argument image. If instance image is shared, replace the image content by the content of the argument image.

8.1.4.13 CImg<T> operator+ ( ) const

Operator+() (unary).

Remarks

- This operator always returns a non-shared copy of an image.

8.1.4.14 static const char* pixel_type ( ) [static]

Return the type of the pixel values.

Returns

a string describing the type of the image pixels (template parameter `T`).

- The string returned may contains spaces (`"unsigned char"`).
- If the template parameter T does not correspond to a registered type, the string `"unknown"` is returned.

8.1.4.15 unsigned int size ( ) const

Return the number of image buffer elements.

- Equivalent to : width() * height() * depth() * spectrum().

example:

```
CImg<> img(100,100,1,3);
if (img.size()==100*100*3) std::fprintf(stderr,"This statement is true");
```

8.1.4.16 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 the pixel value located at (x,y,z,v).

Parameters

- ***x*** X-coordinate of the pixel.
- ***y*** Y-coordinate of the pixel.
- ***z*** Z-coordinate of the pixel.
- ***v*** C-coordinate of the pixel.

- When called without parameters, data() returns a pointer to the begining of the pixel buffer.
- If the macro 'cimg_verbosity'>=3, boundary checking is performed and warning messages may appear if given coordinates are outside the image range (but function performances decrease).

example:

```
CImg<float> img(100,100,1,1,0);   // Define a 100x100 greyscale image with
float-valued pixels.
float *ptr = data(10,10);         // Get a pointer to the pixel located at
(10,10).
float val = *ptr;                 // Get the pixel value.
```

8.1.4.17 int offset (const int *x*, const int *y* = 0, const int *z* = 0, const int *c* = 0) const

Return the offset of the pixel coordinates (x,y,z,v) with respect to the data pointer data.

Parameters

- ***x*** X-coordinate of the pixel.
- ***y*** Y-coordinate of the pixel.
- ***z*** Z-coordinate of the pixel.
- ***v*** C-coordinate of the pixel.

- No checking is done on the validity of the given coordinates.

Example:

```
 CImg<float> img(100,100,1,3,0);         // Define a 100x100 color image wi
th float-valued black pixels.
 long off = img.offset(10,10,0,2);       // Get the offset of the blue valu
e of the pixel located at (10,10).
 float val = img[off];                   // Get the blue value of the pixel
.
```

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

Return the variance of the image.

Parameters

variance_method Determines how to calculate the variance

0	Second moment: $v = 1/N \sum_{k=1}^{N} (x_k - \bar{x})^2 = 1/N \left(\sum_{k=1}^{N} x_k^2 - \left(\sum_{k=1}^{N} x_k \right)^2 /N \right)$ with $\bar{x} = 1/N \sum_{k=1}^{N} x_k$
1	Best unbiased estimator: $v = \frac{1}{N-1} \sum_{k=1}^{N} (x_k - \bar{x})^2$
2	Least median of squares
3	Least trimmed of squares

8.1.4.19 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 expression.

If you make successive evaluations on the same image and with the same expression, you can set 'expr' to 0 after the first call, to skip the math parsing step.

8.1.4.20 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*) [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 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.

8.1.4.21 CImg<T>& dijkstra (const unsigned int *starting_node*, const unsigned int *ending_node*, CImg< t > & *previous*)

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

Instance image corresponds to the adjacency matrix of the graph.

Parameters

starting_node Indice of the starting node.

previous 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.

8.1.4.22 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

interpolation_type Type of interpolation (can be 0=nearest int, 1=linear, 2=2nd-order RK, 3=4th-order RK.

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

Fill an image by a value `val`.

Parameters

val = fill value

Note

All pixel values of the instance image will be initialized by `val`.

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

Compute image with rounded pixel values.

Parameters

y Rounding precision.

rounding_type Roundin type, can be 0 (nearest), 1 (forward), -1(backward).

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

Add random noise to the values of the instance image.

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 instance image.

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 instance image.

Sample code :

```
const CImg<float> img("reference.jpg"), res = img.get_noise(40);
(img,res.normalize(0,255)).display();
```

8.1.4.26 CImg<T>& normalize (const T *value_min*, const T *value_max*)

Linearly normalize values of the instance image between `value_min` and `value_max`.

Parameters

value_min Minimum desired value of the resulting image.

value_max Maximum desired value of the resulting image.

Returns

A reference to the modified instance image.

Note

- Function `CImg<T>::get_normalize()` is also defined. It returns a non-shared modified copy of the instance image.

Sample code :

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

8.1.4.27 CImg<T>& normalize ( )

Normalize multi-valued pixels of the instance image, with respect to their L2-norm.

Returns

A reference to the modified instance image.

Note

- Function `CImg<T>::get_normalize()` is also defined. It returns a non-shared modified copy of the instance image.

Sample code :

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

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

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

Parameters

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

Returns

A reference to the modified instance image.

Note

- Function `CImg<T>::get_norm()` is also defined. It returns a non-shared modified copy of the instance image.

Sample code :

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

8.1.4.29 CImg<T>& cut (const T *value_min*, const T *value_max*)

Cut values of the instance image between `value_min` and `value_max`.

Parameters

value_min Minimum desired value of the resulting image.

value_max Maximum desired value of the resulting image.

Returns

A reference to the modified instance image.

Note

- Function `CImg<T>::get_cut()` is also defined. It returns a non-shared modified copy of the instance image.

Sample code :

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

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

Uniformly quantize values of the instance image into `nb_levels` levels.

Parameters

nb_levels Number of quantization levels.

keep_range Tells if resulting values keep the same range as the original ones.

Returns

A reference to the modified instance image.

Note

- Function `CImg<T>::get_quantize()` is also defined. It returns a non-shared modified copy of the instance image.

Sample code :

```
const CImg<float> img("reference.jpg"), res = img.get_quantize(4);
(img,res).display();
```

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

Threshold values of the instance image.

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.

Returns

A reference to the modified instance image. Resulting pixel values are either equal to 0 or 1.

Note

- Function `CImg<T>::get_threshold()` is also defined. It returns a non-shared modified copy of the instance image.

Sample code :

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

8.1.4.32 CImg<T>& histogram (const unsigned int *nb_levels*, const T *value_min* = `(T)0`, const T *value_max* = `(T)0`)

Compute the histogram of the instance image.

Parameters

nb_levels Number of desired histogram levels.

value_min Minimum pixel value considered for the histogram computation. All pixel values lower than `value_min` will not be counted.

value_max Maximum pixel value considered for the histogram computation. All pixel values higher than `value_max` will not be counted.

Returns

Instance image is replaced by its histogram, defined as a `CImg<T>(nb_levels)` image.

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 `value_min==value_max==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.
- Function `CImg<T>::get_histogram()` is also defined. It returns a non-shared modified copy of the instance image.

Sample code :

```
const CImg<float> img = CImg<float>("reference.jpg").histogram(256);
img.display_graph(0,3);
```

8.1.4.33 CImg<T>& equalize (const unsigned int *nb_levels*, const T *value_min* = `(T)0`, const T *value_max* = `(T)0`)

Compute the histogram-equalized version of the instance image.

Parameters

nb_levels Number of histogram levels used for the equalization.

value_min Minimum pixel value considered for the histogram computation. All pixel values lower than `value_min` will not be counted.

value_max Maximum pixel value considered for the histogram computation. All pixel values higher than `value_max` will not be counted.

Returns

A reference to the modified instance image.

Note

- If `value_min==value_max==0` (default behavior), the function first estimates the whole range of pixel values then uses it to equalize the histogram.
- Function `CImg<T>::get_equalize()` is also defined. It returns a non-shared modified copy of the instance image.

Sample code :

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

8.1.4.34 CImg<T>& index (const CImg< t > & *palette*, const bool *dithering* = `false`, const bool *map_indexes* = `false`)

Index multi-valued pixels of the instance image, regarding to a predefined palette.

Parameters

palette Multi-valued palette used as the basis for multi-valued pixel indexing.

dithering Tells if Floyd-Steinberg dithering is activated or not.

map_indexes Tell if the values of the resulting image are the palette indices or the palette vectors.

Returns

A reference to the modified instance image.

Note

- `img.index(palette,dithering,1)` is equivalent to `img.index(palette,dithering,0).map(palette)`.
- Function `CImg<T>::get_index()` is also defined. It returns a non-shared modified copy of the instance image.

Sample code :

```
 const CImg<float> img("reference.jpg"), palette(3,1,1,3, 0,128,255, 0,128,
255, 0,128,255);
 const CImg<float> res = img.get_index(palette,true,true);
 (img,res).display();
```

8.1.4.35 CImg<T>& map (const CImg< t > & *palette*)

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

Parameters

palette Multi-valued palette used for mapping the indexes.

Returns

A reference to the modified instance image.

Note

- Function `CImg<T>::get_map()` is also defined. It returns a non-shared modified copy of the instance image.

Sample code :

```
const CImg<float> img("reference.jpg"),
                  palette1(3,1,1,3, 0,128,255, 0,128,255, 0,128,255),
                  palette2(3,1,1,3, 255,0,0, 0,255,0, 0,0,255),
                  res = img.get_index(palette1,false).map(palette2);
(img,res).display();
```

8.1.4.36 CImg<T>& label_regions ( )

Create a map of indexed labels counting disconnected regions with same intensities.

Returns

A reference to the modified instance image.

Note

- Function `CImg<T>::get_label_regions()` is also defined. It returns a non-shared modified copy of the instance image.

Sample code :

```
 const CImg<float> img = CImg<float>("reference.jpg").norm().quantize(4),
                   palette = CImg<float>::default_LUT256(),
                   res = img.get_label_regions().normalize(0,255).map(palet
te);
 (img,res).display();
```

8.1.4.37 CImg<T>& RGBtoHSI ( )

Convert color pixels from (R,G,B) to (H,S,I). Reference: "Digital Image Processing, 2nd. edition", R. Gonzalez and R. Woods. Prentice Hall, 2002.

8.1.4.38 CImg<T>& RGBtoBayer ( )

Convert a (R,G,B) image to a Bayer-coded representation.

Note

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

8.1.4.39 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 *border_conditions* = *0*, const float *centering_x* = *0*, const float *centering_y* = *0*, const float *centering_z* = *0*, const float *centering_c* = *0*)

Resize an image.

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 `border_condition`.
- 1 = nearest-neighbor interpolation.
- 2 = moving average interpolation.
- 3 = linear interpolation.
- 4 = grid interpolation.
- 5 = bicubic interpolation.
- 6 = lanczos interpolation.

border_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).

8.1.4.40 CImg<T>& resize_doubleXY ( )

Upscale an image by a factor 2x.

Use anisotropic upscaling algorithm described at http://scale2x.sourceforge.net/algorithm.html

8.1.4.41 CImg<T>& resize_tripleXY ( )

Upscale an image by a factor 3x.

Use anisotropic upscaling algorithm described at http://scale2x.sourceforge.net/algorithm.html

8.1.4.42 CImg<T>& shift (const int *deltax*, const int *deltay* = *0*, const int *deltaz* = *0*, const int *deltac* = *0*, const int *border_condition* = *0*)

Shift the image.

Parameters

deltax Amount of displacement along the X-axis.
deltay Amount of displacement along the Y-axis.
deltaz Amount of displacement along the Z-axis.
deltac Amount of displacement along the C-axis.
border_condition Border condition.

- `border_condition` can be :
  - 0 : Zero border condition (Dirichlet).
  - 1 : Nearest neighbors (Neumann).
  - 2 : Repeat Pattern (Fourier style).

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

Permute axes order.

This function permutes image axes.

Parameters

permut = String describing the permutation (4 characters).

8.1.4.44 CImg<T>& rotate (const float *angle*, const unsigned int *border_conditions* = 0, const unsigned int *interpolation* = 1)

Rotate an image.

Parameters

angle = rotation angle (in degrees).

cond = rotation type. can be :

- 0 = zero-value at borders
- 1 = nearest pixel.
- 2 = cyclic.

Note

Returned image will probably have a different size than the instance image ∗this.

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

Rotate an image around a center point (`cx`,`cy`).

Parameters

angle = rotation angle (in degrees).

cx = X-coordinate of the rotation center.

cy = Y-coordinate of the rotation center.

zoom = zoom.

cond = rotation type. can be :

- 0 = zero-value at borders
- 1 = repeat image at borders
- 2 = zero-value at borders and linear interpolation

8.1.4.46 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 *border_condition* = false)

Get a square region of the image.

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.

border_condition = Dirichlet (false) or Neumann border conditions.

8.1.4.47 CImg<T>& crop (const int *x0*, const int *y0*, const int *z0*, const int *x1*, const int *y1*, const int *z1*, const bool *border_condition* = `false`)

Get a rectangular part of the instance image.

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.

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.

border_condition = determine the type of border condition if some of the desired region is outside the image.

8.1.4.48 CImg<T>& crop (const int *x0*, const int *y0*, const int *x1*, const int *y1*, const bool *border_condition* = `false`)

Get a rectangular part of the instance image.

Parameters

x0 = X-coordinate of the upper-left crop rectangle corner.

y0 = Y-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.

border_condition = determine the type of border condition if some of the desired region is outside the image.

8.1.4.49 CImg<T>& crop (const int *x0*, const int *x1*, const bool *border_condition* = `false`)

Get a rectangular part of the instance image.

Parameters

x0 = X-coordinate of the upper-left crop rectangle corner.

x1 = X-coordinate of the lower-right crop rectangle corner.

border_condition = determine the type of border condition if some of the desired region is outside the image.

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

Compute the correlation of the instance image by a mask.

The correlation of the instance image `*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)

Parameters

mask = the correlation kernel.

border_conditions = the border condition type (0=zero, 1=dirichlet)

is_normalized = enable local normalization.

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

Compute the convolution of the image by a mask.

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)

Parameters

mask = the correlation kernel.

border_conditions = the border condition type (0=zero, 1=dirichlet)

is_normalized = enable local normalization.

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

Compute the result of the Deriche filter.

The Canny-Deriche filter is a recursive algorithm allowing to compute blurred derivatives of order 0,1 or 2 of an image.

8.1.4.53 CImg<T>& blur (const float *sigmax*, const float *sigmay*, const float *sigmaz*, const bool *cond* = `true`)

Return a blurred version of the image, using a Canny-Deriche filter.

Blur the image with an anisotropic exponential filter (Deriche filter of order 0).

8.1.4.54 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 *fast_approx* = *1*)

Blur the image anisotropically following 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 gaussian function.

interpolation Used interpolation scheme (0 = nearest-neighbor, 1 = linear, 2 = Runge-Kutta)

fast_approx = Tell to use the fast approximation or not.

8.1.4.55 CImg<T>& blur_bilateral (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 an image using the bilateral filter.

Parameters

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 range 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 range 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).

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

Compute the list of images, corresponding to the XY-gradients of an image.

Parameters

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.

8.1.4.57 CImg<T>& displacement (const CImg< T > & *target*, const float *smooth* = *0.1f*, const float *precision* = *0.1f*, const unsigned int *nb_scales* = *0*, const unsigned int *iteration_max* = *1000*, const bool *backward* = `true`)

Estimate a displacement field between instance image and given target image.

Parameters

backward : if false, match I2(X+U(X)) = I1(X), else match I2(X) = I1(X-U(X)).

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

Compute the Haar multiscale wavelet transform (monodimensional version).

Parameters

axis Axis considered for the transform.

invert Set inverse of direct transform.

nb_scales Number of scales used for the transform.

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

Compute the Haar multiscale wavelet transform.

Parameters

invert Set inverse of direct transform.

nb_scales Number of scales used for the transform.

8.1.4.60 CImg<floatT> get_elevation3d (CImgList< tf > & *primitives*, CImgList< tc > & *colors*, const CImg< te > & *elevation*) const

Create and return a 3d elevation of the instance image.

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).

Sample code :

```
 const CImg<float> img("reference.jpg");
 CImgList<unsigned int> faces3d;
 CImgList<unsigned char> colors3d;
 const CImg<float> points3d = img.get_elevation3d(faces3d,colors,img.get_no
rm()*0.2);
 CImg<unsigned char>().display_object3d("Elevation3d",points3d,faces3d,colo
rs3d);
```

8.1.4.61 CImg<floatT> get_isoline3d (CImgList< tf > & *primitives*, const float *isovalue*, const int *size_x* = `-100`, const int *size_y* = `-100`) const

Create and return a isoline of the instance image 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).

Sample code :

```
 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,colors
3d);
```

8.1.4.62 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

Create and return a isosurface of the instance image 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).

Sample code :

```
 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,col
ors3d);
```

8.1.4.63 static CImg<floatT> box3d (CImgList< tf > & *primitives*, const float *size_x* = *200*, const float *size_y* = *100*, const float *size_z* = *100*) [static]

Create and return 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).

Sample code :

```
CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::box3d(faces3d,10,20,30);
CImg<unsigned char>().display_object3d("Box3d",points3d,faces3d);
```

8.1.4.64 static CImg<floatT> cone3d (CImgList< tf > & *primitives*, const float *radius* = *50*, const float *size_z* = *100*, const unsigned int *subdivisions* = *24*) [static]

Create and return 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).

Sample code :

```
CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::cone3d(faces3d,50);
CImg<unsigned char>().display_object3d("Cone3d",points3d,faces3d);
```

8.1.4.65 static CImg<floatT> cylinder3d (CImgList< tf > & *primitives*, const float *radius* = *50*, const float *size_z* = *100*, const unsigned int *subdivisions* = *24*) [static]

Create and return 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).

Sample code :

```
CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::cylinder3d(faces3d,50);
CImg<unsigned char>().display_object3d("Cylinder3d",points3d,faces3d);
```

8.1.4.66 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]

Create and return 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).

Sample code :

```
CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::torus3d(faces3d,20,4);
CImg<unsigned char>().display_object3d("Torus3d",points3d,faces3d);
```

8.1.4.67 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]

Create and return 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).

Sample code :

```
CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::plane3d(faces3d,100,50);
CImg<unsigned char>().display_object3d("Plane3d",points3d,faces3d);
```

8.1.4.68 static CImg<floatT> sphere3d (CImgList< tf > & *primitives*, const float *radius* = 50, const unsigned int *subdivisions* = 3) [static]

Create and return 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).

Sample code :

```
CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::sphere3d(faces3d,100,4);
CImg<unsigned char>().display_object3d("Sphere3d",points3d,faces3d);
```

8.1.4.69 static CImg<floatT> ellipsoid3d (CImgList< tf > & *primitives*, const CImg< t > & *tensor*, const unsigned int *subdivisions* = 3) [static]

Create and return 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).

Sample code :

```
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);
```

8.1.4.70 CImg<T>& draw_point (const int *x0*, const int *y0*, const tc *const *color*, const float *opacity* = 1)

Draw a 2d colored point (pixel).

Parameters

x0 X-coordinate of the point.

y0 Y-coordinate of the point.

color Pointer to `spectrum()` consecutive values, defining the color values.

opacity Drawing opacity (optional).

Note

- Clipping is supported.
- 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);
```

8.1.4.71 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 colored 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 (optional).

pattern An integer whose bits describe the line pattern (optional).

init_hatch Flag telling if a reinitialization of the hash state must be done (optional).

Note

- Clipping is supported.
- 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);
```

8.1.4.72 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 2d textured 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 (optional).

pattern An integer whose bits describe the line pattern (optional).

init_hatch Flag telling if the hash variable must be reinitialized (optional).

Note

- Clipping is supported but not for texture coordinates.
- 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);
```

8.1.4.73 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 colored lines in the instance image.

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 (optional).

pattern An integer whose bits describe the line pattern (optional).

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`.

Example:

```
CImg<unsigned char> img(100,100,1,3,0);
const unsigned char color[] = { 255,128,64 };
CImgList<int> points;
points.insert(CImg<int>::vector(0,0)).
      .insert(CImg<int>::vector(70,10)).
      .insert(CImg<int>::vector(80,60)).
      .insert(CImg<int>::vector(10,90));
img.draw_line(points,color);
```

8.1.4.74 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 colored arrow in the instance image.

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 (optional).

length Length of the arrow head. If negative, describes a percentage of the arrow length (optional).

opacity Drawing opacity (optional).

pattern An integer whose bits describe the line pattern (optional).

Note

- Clipping is supported.

8.1.4.75 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 cubic spline curve in the instance image.

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 (optional).

opacity Drawing opacity (optional).

pattern An integer whose bits describe the line pattern (optional).

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);
```

8.1.4.76 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 cubic spline curve in the instance image (for volumetric images).

Note

- Similar to CImg::draw_spline() for a 3d spline in a volumetric image.

8.1.4.77 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 cubic spline curve in the instance image.

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 (optional).
opacity Drawing opacity (optional).
pattern An integer whose bits describe the line pattern (optional).
init_hatch if `true`, reinit hatch motif.

8.1.4.78 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 2d Gouraud-shaded colored triangle.

Parameters

x0 = X-coordinate of the first corner in the instance image.

y0 = Y-coordinate of the first corner in the instance image.

x1 = X-coordinate of the second corner in the instance image.

y1 = Y-coordinate of the second corner in the instance image.

x2 = X-coordinate of the third corner in the instance image.

y2 = Y-coordinate of the third corner in the instance image.

color = array of spectrum() values of type `T`, defining the global drawing color.

brightness0 = brightness of the first corner (in [0,2]).

brightness1 = brightness of the second corner (in [0,2]).

brightness2 = brightness of the third corner (in [0,2]).

opacity = opacity of the drawing.

Note

Clipping is supported.

8.1.4.79 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 2d textured triangle.

Parameters

x0 = X-coordinate of the first corner in the instance image.

y0 = Y-coordinate of the first corner in the instance image.

x1 = X-coordinate of the second corner in the instance image.

y1 = Y-coordinate of the second corner in the instance image.

x2 = X-coordinate of the third corner in the instance image.

y2 = Y-coordinate of the third corner in the instance image.

texture = texture image used to fill the triangle.

tx0 = X-coordinate of the first corner in the texture image.

ty0 = Y-coordinate of the first corner in the texture image.

tx1 = X-coordinate of the second corner in the texture image.

ty1 = Y-coordinate of the second corner in the texture image.

tx2 = X-coordinate of the third corner in the texture image.

ty2 = Y-coordinate of the third corner in the texture image.

opacity = opacity of the drawing.

brightness = brightness of the drawing (in [0,2]).

Note

Clipping is supported, but texture coordinates do not support clipping.

8.1.4.80 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 2d Pseudo-Phong-shaded triangle.

Parameters

x0 = X-coordinate of the first corner in the instance image.

y0 = Y-coordinate of the first corner in the instance image.

x1 = X-coordinate of the second corner in the instance image.

y1 = Y-coordinate of the second corner in the instance image.

x2 = X-coordinate of the third corner in the instance image.

y2 = Y-coordinate of the third corner in the instance image.

color = array of spectrum() values of type `T`, defining the global drawing color.

light = light image.

lx0 = X-coordinate of the first corner in the light image.

ly0 = Y-coordinate of the first corner in the light image.

lx1 = X-coordinate of the second corner in the light image.

ly1 = Y-coordinate of the second corner in the light image.

lx2 = X-coordinate of the third corner in the light image.

ly2 = Y-coordinate of the third corner in the light image.

opacity = opacity of the drawing.

Note

Clipping is supported, but texture coordinates do not support clipping.

8.1.4.81 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 2d Gouraud-shaded textured triangle.

Parameters

x0 = X-coordinate of the first corner in the instance image.

y0 = Y-coordinate of the first corner in the instance image.

x1 = X-coordinate of the second corner in the instance image.

y1 = Y-coordinate of the second corner in the instance image.

x2 = X-coordinate of the third corner in the instance image.

y2 = Y-coordinate of the third corner in the instance image.

texture = texture image used to fill the triangle.

tx0 = X-coordinate of the first corner in the texture image.

ty0 = Y-coordinate of the first corner in the texture image.

tx1 = X-coordinate of the second corner in the texture image.

ty1 = Y-coordinate of the second corner in the texture image.

tx2 = X-coordinate of the third corner in the texture image.

ty2 = Y-coordinate of the third corner in the texture image.

brightness0 = brightness value of the first corner.

brightness1 = brightness value of the second corner.

brightness2 = brightness value of the third corner.

opacity = opacity of the drawing.

Note

Clipping is supported, but texture coordinates do not support clipping.

8.1.4.82 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 2d Pseudo-Phong-shaded textured triangle.

Parameters

x0 = X-coordinate of the first corner in the instance image.

y0 = Y-coordinate of the first corner in the instance image.

x1 = X-coordinate of the second corner in the instance image.

y1 = Y-coordinate of the second corner in the instance image.

x2 = X-coordinate of the third corner in the instance image.

y2 = Y-coordinate of the third corner in the instance image.

texture = texture image used to fill the triangle.

tx0 = X-coordinate of the first corner in the texture image.

ty0 = Y-coordinate of the first corner in the texture image.

tx1 = X-coordinate of the second corner in the texture image.

ty1 = Y-coordinate of the second corner in the texture image.

tx2 = X-coordinate of the third corner in the texture image.

ty2 = Y-coordinate of the third corner in the texture image.

light = light image.

lx0 = X-coordinate of the first corner in the light image.

ly0 = Y-coordinate of the first corner in the light image.

lx1 = X-coordinate of the second corner in the light image.

ly1 = Y-coordinate of the second corner in the light image.

lx2 = X-coordinate of the third corner in the light image.

ly2 = Y-coordinate of the third corner in the light image.

opacity = opacity of the drawing.

Note

Clipping is supported, but texture coordinates do not support clipping.

8.1.4.83 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 4d filled rectangle in the instance image, at coordinates (x0,y0,z0,c0)-(x1,y1,z1,c1).

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 (optional).

Note

- Clipping is supported.

8.1.4.84 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 *const *color*, const float *opacity* = *1*)

Draw a 3d filled colored rectangle in the instance image, at coordinates (x0,y0,z0)-(x1,y1,z1).

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 (optional).

Note

- Clipping is supported.

8.1.4.85 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 2d filled colored rectangle in the instance image, at coordinates (`x0`,`y0`)-(`x1`,`y1`).

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 (optional).

Note

- Clipping is supported.

8.1.4.86 CImg<T>& draw_circle (const int *x0*, const int *y0*, int *radius*, const tc *const *color*, const float *opacity* = *1*)

Draw a filled circle.

Parameters

x0 X-coordinate of the circle center.

y0 Y-coordinate of the circle center.

radius Circle radius.

color Array of spectrum() values of type `T`, defining the drawing color.

opacity Drawing opacity.

Note

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

8.1.4.87 CImg<T>& draw_circle (const int *x0*, const int *y0*, int *radius*, const tc *const *color*, const float *opacity*, const unsigned *int*)

Draw an outlined circle.

Parameters

x0 X-coordinate of the circle center.

y0 Y-coordinate of the circle center.

radius Circle radius.

color Array of spectrum() values of type `T`, defining the drawing color.

opacity Drawing opacity.

8.1.4.88 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 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 = array of spectrum() values of type `T`, defining the drawing color.

opacity = opacity of the drawing.

8.1.4.89 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 ellipse.

Parameters

x0 = X-coordinate of the ellipse center.

y0 = Y-coordinate of the ellipse center.

tensor = Diffusion tensor describing the ellipse.

color = array of spectrum() values of type `T`, defining the drawing color.

opacity = opacity of the drawing.

8.1.4.90 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 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.

ru = X-coordinate of the orientation vector related to the first radius.

rv = Y-coordinate of the orientation vector related to the first radius.

color = array of spectrum() values of type `T`, defining the drawing color.

pattern = If zero, the ellipse is filled, else pattern is an integer whose bits describe the outline pattern.

opacity = opacity of the drawing.

8.1.4.91 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 ellipse.

Parameters

x0 = X-coordinate of the ellipse center.

y0 = Y-coordinate of the ellipse center.

tensor = Diffusion tensor describing the ellipse.

color = array of spectrum() values of type `T`, defining the drawing color.

opacity = opacity of the drawing.

8.1.4.92 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 (optional).

Note

- Clipping is supported.

8.1.4.93 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_valmax* = *1*)

Draw a sprite image in the instance image (masked version).

Parameters

sprite Sprite image.

mask Mask image.

x0 X-coordinate of the sprite position in the instance image.

y0 Y-coordinate of the sprite position in the instance image.

z0 Z-coordinate of the sprite position in the instance image.

c0 C-coordinate of the sprite position in the instance image.

mask_valmax Maximum pixel value of the mask image `mask` (optional).

opacity Drawing opacity.

Note

- Pixel values of `mask` set the opacity of the corresponding pixels in `sprite`.
- Clipping is supported.
- Dimensions along x,y and z of `sprite` and `mask` must be the same.

8.1.4.94 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.

Parameters

x0 X-coordinate of the text in the instance image.

y0 Y-coordinate of the text in the instance image.

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').

font Font used for drawing text.

opacity Drawing opacity.

format 'printf'-style format string, followed by arguments.

Note

Clipping is supported.

8.1.4.95 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.

Parameters

x0 X-coordinate of the text in the instance image.

y0 Y-coordinate of the text in the instance image.

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').

font_size Size of the font (exact match for 13,24,32,57).

opacity Drawing opacity.

format 'printf'-style format string, followed by arguments.

Note

Clipping is supported.

8.1.4.96 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 *arrows* = true, const unsigned int *pattern* = ~0U)

Draw a vector field in the instance image, using a colormap.

Parameters

flow Image of 2d vectors used as input data.

color Image of spectrum()-D vectors corresponding to the color of each arrow.

sampling Length (in pixels) between each arrow.

factor Length factor of each arrow (if <0, computed as a percentage of the maximum length).

opacity Opacity of the drawing.

pattern Used pattern to draw lines.

Note

Clipping is supported.

8.1.4.97 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 *arrows* = true, const unsigned int *pattern* = ~0U)

Draw a vector field in the instance image, using a colormap.

Parameters

flow Image of 2d vectors used as input data.

color Image of spectrum()-D vectors corresponding to the color of each arrow.

sampling Length (in pixels) between each arrow.

factor Length factor of each arrow (if <0, computed as a percentage of the maximum length).

opacity Opacity of the drawing.

pattern Used pattern to draw lines.

Note

Clipping is supported.

8.1.4.98 CImg<T>& draw_axis (const CImg< t > & *xvalues*, const int *y*, const tc *const *color*, const float *opacity* = 1, const unsigned int *pattern* = ~0U)

Draw a labeled horizontal axis on the instance image.

Parameters

xvalues Lower bound of the x-range.

y Y-coordinate of the horizontal axis in the instance image.

color Array of spectrum() values of type `T`, defining the drawing color.

opacity Drawing opacity.

pattern Drawing pattern.

opacity_out Drawing opacity of 'outside' axes.

Note

if `precision==0`, precision of the labels is automatically computed.

8.1.4.99 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 a 1d graph on the instance image.

Parameters

data Image containing the graph values I = f(x).

color Array of spectrum() values of type `T`, 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.

8.1.4.100 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 *high_connexity* = *false*)

Draw a 3d filled region starting from a point (`x`,`y`,\ `z`) in the instance image.

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 An array of spectrum() values of type `T`, defining the drawing color.

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.

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.

8.1.4.101 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 *high_connexity* = *false*)

Draw a 3d filled region starting from a point (`x`,`y`,\ `z`) in the instance image.

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 = an array of spectrum() values of type `T`, defining the drawing color.

sigma = tolerance concerning neighborhood values.

opacity = opacity of the drawing.

8.1.4.102 CImg<T>& draw_fill (const int *x*, const int *y*, const tc *const *color*, const float *opacity* = 1, const float *sigma* = 0, const bool *high_connexity* = false)

Draw a 2d filled region starting from a point (x,y) in the instance image.

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.
color = an array of spectrum() values of type T, defining the drawing color.
sigma = tolerance concerning neighborhood values.
opacity = opacity of the drawing.

8.1.4.103 CImg<T>& draw_plasma (const int *x0*, const int *y0*, const int *x1*, const int *y1*, const float *alpha* = 1, const float *beta* = 1, const float *opacity* = 1)

Draw a plasma random texture.

Parameters

x0 = X-coordinate of the upper-left corner of the plasma.
y0 = Y-coordinate of the upper-left corner of the plasma.
x1 = X-coordinate of the lower-right corner of the plasma.
y1 = Y-coordinate of the lower-right corner of the plasma.
alpha = Alpha-parameter of the plasma.
beta = Beta-parameter of the plasma.
opacity = opacity of the drawing.

8.1.4.104 CImg<T>& draw_plasma (const float *alpha* = 1, const float *beta* = 1, const float *opacity* = 1)

Draw a plasma random texture.

Parameters

alpha = Alpha-parameter of the plasma.
beta = Beta-parameter of the plasma.
opacity = opacity of the drawing.

8.1.4.105 CImg<T>& draw_gaussian (const float *xc*, const float *sigma*, const tc *const *color*, const float *opacity* = 1)

Draw a 1d gaussian function in the instance image.

Parameters

xc = X-coordinate of the gaussian center.
sigma = Standard variation of the gaussian distribution.
color = array of spectrum() values of type T, defining the drawing color.
opacity = opacity of the drawing.

8.1.4.106 CImg<T>& draw_gaussian (const float *xc*, const float *yc*, const CImg< t > & *tensor*, const tc *const *const *color*, const float *opacity* = *1*)

Draw an anisotropic 2d gaussian function.

Parameters

xc = X-coordinate of the gaussian center.

yc = Y-coordinate of the gaussian center.

tensor = 2x2 covariance matrix.

color = array of spectrum() values of type `T`, defining the drawing color.

opacity = opacity of the drawing.

8.1.4.107 CImg<T>& draw_gaussian (const float *xc*, const float *yc*, const float *sigma*, const tc *const *const *color*, const float *opacity* = *1*)

Draw an isotropic 2d gaussian function.

Parameters

xc = X-coordinate of the gaussian center.

yc = Y-coordinate of the gaussian center.

sigma = standard variation of the gaussian distribution.

color = array of spectrum() values of type `T`, defining the drawing color.

opacity = opacity of the drawing.

8.1.4.108 CImg<T>& draw_gaussian (const float *xc*, const float *yc*, const float *zc*, const CImg< t > & *tensor*, const tc *const *const *color*, const float *opacity* = *1*)

Draw an anisotropic 3d gaussian function.

Parameters

xc = X-coordinate of the gaussian center.

yc = Y-coordinate of the gaussian center.

zc = Z-coordinate of the gaussian center.

tensor = 3x3 covariance matrix.

color = array of spectrum() values of type `T`, defining the drawing color.

opacity = opacity of the drawing.

8.1.4.109 CImg<T>& draw_gaussian (const float *xc*, const float *yc*, const float *zc*, const float *sigma*, const tc *const *const *color*, const float *opacity* = *1*)

Draw an isotropic 3d gaussian function.

Parameters

xc = X-coordinate of the gaussian center.

yc = Y-coordinate of the gaussian center.

zc = Z-coordinate of the gaussian center.

sigma = standard variation of the gaussian distribution.

color = array of spectrum() values of type `T`, defining the drawing color.

opacity = opacity of the drawing.

8.1.4.110 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 *double_sided* = false, const float *focale* = 500, const float *lightx* = 0, const float *lighty* = 0, const float *lightz* = -5e8, const float *specular_light* = 0.2f, const float *specular_shine* = 0.1f)

Draw a 3d object.

Parameters

X = X-coordinate of the 3d object position

Y = Y-coordinate of the 3d object position

Z = 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 = Render type (0=Points, 1=Lines, 2=Faces (no light), 3=Faces (flat), 4=Faces(Gouraud)

double_sided = Tell 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_shine = Shininess of the object

8.1.4.111 CImg<T>& select (CImgDisplay & *disp*, const int *select_type* = 2, unsigned int *const *XYZ* = 0, const unsigned char *const *color* = 0)

Simple interface to select a shape from an image.

Parameters

selection Array of 6 values containing the selection result

coords_type Determine shape type to select (0=point, 1=vector, 2=rectangle, 3=circle)

disp Display window used to make the selection

XYZ Initial XYZ position (for volumetric images only)

color Color of the shape selector.

8.1.4.112 CImg<T>& load (const char *const *filename*)

Load an image from a file.

Parameters

filename is the name of the image file to load.

Note

The extension of `filename` defines the file format. If no filename extension is provided, CImg<T>::get_load() will try to load a .cimg file.

8.1.4.113 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 an image from a TIFF file.

- 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 type.
- If cimg_use_tif is not defined at compilation time the function uses CImg<T>&load_other(const char*).

See also

CImg<T>& load_other(const char*)
CImg<T>& save_tiff(const char*, const unsigned int)

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

Display informations about the image on the standard error output.

Parameters

title Name for the considered image (optional).

display_stats Compute and display image statistics (optional).

8.1.4.115 const CImg<T>& save (const char *const *filename*, const int *number* = -1) const

Save the image as a file.

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.

8.1.4.116 const CImg<T>& save_tiff (const char *const *filename*, const unsigned int *compression* = 0) const

Save a file in TIFF format.

- 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 type.
- If cimg_use_tif is not defined at compilation time the function uses CImg<T>&save_other(const char*).

Parameters

compression 1:None, 2:CCITTRLE, 3:CCITTFAX3, 4:CCITTFAX4, 5:LZW, 6:JPEG

See also

CImg<T>& save_other(const char*)
CImg<T>& load_tiff(const char*)

8.1.4.117 const CImg<T>& save_graphicsmagick_external (const char *const *filename*, const unsigned int *quality* = 100) const

Save the image using GraphicsMagick's gm.

Function that saves the image for other file formats that are not natively handled by CImg, using the tool 'gm' from the GraphicsMagick package.

This is the case for all compressed image formats (GIF,PNG,JPG,TIF, ...). You need to install the GraphicsMagick package in order to get this function working properly (see http://www.graphicsmagick.org).

8.1.4.118 const CImg<T>& save_imagemagick_external (const char *const *filename*, const unsigned int *quality* = 100) const

Save the image using ImageMagick's convert.

Function that saves the image for other file formats that are not natively handled by CImg, using the tool 'convert' from the ImageMagick package.

This is the case for all compressed image formats (GIF,PNG,JPG,TIF, ...). You need to install the ImageMagick package in order to get this function working properly (see http://www.imagemagick.org).

8.2 CImgDisplay Struct Reference

This class represents a window which can display CImg images and handles mouse and keyboard events.

Constructors / Destructor / Instance Management

- ~CImgDisplay ()

  Destructor.

- CImgDisplay ()

  Create an empty display window.

- 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)

  Create a display window with a specified size `pwidth x height`.

- 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)

  Create a display window 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)

  Create a display window from an image list.

- CImgDisplay (const CImgDisplay &disp)

  Create a display window by copying another one.

- CImgDisplay & assign ()

  In-place version of the destructor.

- 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)

  In-place version of the constructor.

- 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)

  In-place version of the constructor.

- 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)

  In-place version of the constructor.

- CImgDisplay & assign (const CImgDisplay &disp)

  In-place version of the constructor.

- static CImgDisplay & empty ()

  Return a reference to an empty display.

Overloaded Operators

- template<typename t >
  CImgDisplay & **operator=** (const CImg< t > &img)
- template<typename t >
  CImgDisplay & **operator=** (const CImgList< t > &list)
- CImgDisplay & operator= (const CImgDisplay &disp)

  Operator=().

- operator bool () const

  Return true if display is not empty.

Instance Checking

- bool is_empty () const

  Return true is display is empty.

- bool **is_closed** () const
- bool **is_resized** () const
- bool **is_moved** () const
- bool **is_event** () const
- bool **is_fullscreen** () const
- bool **is_key** () const
- bool **is_keyESC** () const
- 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
- bool **is_key** (const unsigned int key) const
- bool is_key (const char *const textcode) const

  Get keycode corresponding to given input string.

- bool is_key_sequence (const unsigned int *const key_sequence, const unsigned int length, const bool remove_sequence=false)

  Test if a key sequence has been typed.

Instance Characteristics

- int width () const

  Return display width.

- int height () const

  Return display height.

- 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 or previous 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 current or previous state of the keyboard.

- unsigned int **released_key** (const unsigned int pos=0) const
- unsigned int normalization () const

  Return normalization type of the display.

- const char * title () const

Return title of the display.

- int window_width () const
  Return display window width.
- int window_height () const
  Return display window height.
- int window_x () const
  Return X-coordinate of the window.
- int window_y () const
  Return Y-coordinate of the window.
- float frames_per_second ()
  Return the frame per second rate.
- static unsigned int keycode (const char *const textcode)
  Get keycode corresponding to given input string.
- static int screen_width ()
  Return the width of the screen resolution.
- static int screen_height ()
  Return the height of the screen resolution.

Display Manipulation

- template<typename T >
  CImgDisplay & display (const CImg< T > &img)
  Display an image in a window.
- template<typename T >
  CImgDisplay & display (const CImgList< T > &list, const char axis='x', const char align='p')
  Display an image list CImgList<T> into a display window.
- CImgDisplay & resize (const bool redraw=true)
  Resize a display window in its current size.
- template<typename T >
  CImgDisplay & resize (const CImg< T > &img, const bool redraw=true)
  Resize a display window with the size of an image.
- CImgDisplay & resize (const CImgDisplay &disp, const bool redraw=true)
  Resize a display window using the size of the given display `disp`.
- CImgDisplay & resize (const int width, const int height, const bool redraw=true)
  Resize window.

- CImgDisplay & set_fullscreen (const bool is_fullscreen, const bool redraw=true)
  Set fullscreen mode.
- CImgDisplay & toggle_fullscreen (const bool redraw=true)
  Toggle fullscreen mode.
- CImgDisplay & show ()
  Show a closed display.
- CImgDisplay & close ()
  Close a visible display.
- CImgDisplay & move (const int pos_x, const int pos_y)
  Move window.
- 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 specific location.
- CImgDisplay & set_title (const char *const format,...)
  Set the window title.
- template<typename T >
  CImgDisplay & render (const CImg< T > &img)
  Render image buffer into GDI native image format.
- CImgDisplay & paint ()
  Re-paint image content in window.
- template<typename T >
  const CImgDisplay & snapshot (CImg< T > &img) const
  Take a snapshot of the display in the specified image.
- CImgDisplay & set_button ()
  Simulate a mouse button event.
- CImgDisplay & **set_button** (const unsigned int button, const bool is_pressed=true)
- CImgDisplay & set_wheel ()
  Simulate a mouse wheel event, or flush wheel events.
- CImgDisplay & **set_wheel** (const int amplitude)
- CImgDisplay & set_key ()
  Simulate a keyboard press/release event, or flush all key events.
- CImgDisplay & **set_key** (const unsigned int keycode, const bool pressed=true)

- CImgDisplay & flush ()

  Flush all display events.

- CImgDisplay & wait (const unsigned int milliseconds)

  Synchronized waiting function. Same as cimg::wait().

- CImgDisplay & wait ()

  Wait for an event occuring on the current display.

- 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 a window event in any CImg window.

8.2.1 Detailed Description

This class represents a window which can display CImg images and handles mouse and keyboard events. Creating a `CImgDisplay` instance opens a window that can be used to display a `CImg<T>` image of a `CImgList<T>` image list inside. When a display is created, associated window events (such as mouse motion, keyboard and window size changes) are handled and can be easily detected by testing specific `CImgDisplay` data fields. See Using Display Windows. for a complete tutorial on using the `CImgDisplay` class.

8.2.2 Constructor & Destructor Documentation

8.2.2.1 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*)

Create a display window with a specified size `pwidth` x `height`.

Parameters

width Width of the display window.

height Height of the display window.

title Title of the display window.

normalization Normalization type of the display window (0=none, 1=always, 2=once).

is_fullscreen : Fullscreen mode.

is_closed : Initially visible mode. A black image will be initially displayed in the display window.

8.2.2.2 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]

Create a display window from an image.

Parameters

img : Image that will be used to create the display window.

title : Title of the display window

normalization : Normalization type of the display window.

is_fullscreen : Fullscreen mode.

is_closed : Initially visible mode.

8.2.2.3 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]

Create a display window from an image list.

Parameters

list : The list of images to display.

title : Title of the display window

normalization : Normalization type of the display window.

is_fullscreen : Fullscreen mode.

is_closed : Initially visible mode.

8.2.2.4 CImgDisplay (const CImgDisplay & *disp*)

Create a display window by copying another one.

Parameters

disp : Display window to copy.

8.2.3 Member Function Documentation

8.2.3.1 CImgDisplay& display (const CImgList< T > & *list,* const char *axis* = *'x',* const char *align* = *'p'*)

Display an image list CImgList<T> into a display window.

First, all images of the list are appended into a single image used for visualization, then this image is displayed in the current display window.

Parameters

list : The list of images to display.

axis : The axis used to append the image for visualization. Can be 'x' (default),'y','z' or 'c'.

align : Defines the relative alignment of images when displaying images of different sizes. Can be `'c'` (centered, which is the default), `'p'` (top alignment) and `'n'` (bottom aligment).

8.2.3.2 CImgDisplay& resize (const CImg< T > & *img,* const bool *redraw* = `true`)

Resize a display window with the size of an image.

Parameters

img : Input image. `image.width` and `image.height` give the new dimensions of the display window.

redraw : If `true` (default), the current displayed image in the display window will be bloc-interpolated to fit the new dimensions. If `false`, a black image will be drawn in the resized window.

8.3 CImgException Struct Reference

Instances of this class are thrown when errors occur during a CImg library function call.

Inherited by CImgArgumentException, CImgInstanceException, CImgIOException, and CImg-WarningException.

8.3.1 Detailed Description

Instances of this class are thrown when errors occur during a CImg library function call.

8.3.2 Overview

CImgException is the base class of CImg exceptions. Exceptions are thrown by the CImg Library when an error occured in a CImg library function call. CImgException is seldom thrown itself. Children classes that specify the kind of error encountered are generally used instead. These sub-classes are :

- **CImgInstanceException** : Thrown when the instance associated to the called CImg function is not correctly defined. Generally, this exception is thrown when one tries to process *empty* images. The example below will throw a *CImgInstanceException*.

```
CImg<float> img;        // Construct an empty image.
img.blur(10);           // Try to blur the image.
```

- **CImgArgumentException** : Thrown when one of the arguments given to the called CImg function is not correct. Generally, this exception is thrown when arguments passed to the function are outside an admissible range of values. The example below will throw a *CImgArgumentException*.

```
CImg<float> img(100,100,1,3);   // Define a 100x100 color image with float
pixels.
img = 0;                     // Try to fill pixels from the 0 pointer (inva
lid argument to operator=() ).
```

- **CImgIOException** : Thrown when an error occured when trying to load or save image files. The example below will throw a *CImgIOException*.

```
CImg<float> img("file_doesnt_exist.jpg");    // Try to load a file that doe
sn't exist.
```

The parent class CImgException may be thrown itself when errors that cannot be classified in one of the above type occur. It is recommended not to throw CImgExceptions yourself, since there are normally reserved to CImg Library functions. **CImgInstanceException**, **CImgArgumentException** and **CImgIOException** are simple subclasses of CImgException and are thus not detailled more in this reference documentation.

8.3.3 Exception handling

When an error occurs, the CImg Library first displays the error in a modal window. Then, it throws an instance of the corresponding exception class, generally leading the program to stop (this is the default behavior). You can bypass this default behavior by handling the exceptions yourself, using a code block `try { ... } catch () { ... }`. In this case, you can avoid the apparition of the modal window, by defining the environment variable `cimg_verbosity` to 0 before including the CImg header file. The example below shows how to cleanly handle CImg Library exceptions :

```
#define cimg_verbosity 0     // Disable modal window in CImg exceptions.
#define "CImg.h"
int main() {
  try {
    ...; // Here, do what you want.
  }
  catch (CImgInstanceException &e) {
    std::fprintf(stderr,"CImg Library Error : %s",e.what());  // Display yo
ur own error message
    ...                                                       // Do what yo
u want now.
  }
}
```

8.4 CImgList< T > Struct Template Reference

Class representing list of images CImg<T>.

Public Types

- typedef CImg< T > * iterator

  Define a CImgList<T>::iterator.

- typedef const CImg< T > * const_iterator

  Define a CImgList<T>::const_iterator.

- typedef T value_type

  Value type.

Constructors / Destructor / Instance Management

- ~CImgList ()

  Destructor.

- CImgList ()

  Default constructor.

- CImgList (const unsigned int n)

  Construct an image list containing n 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 an image list containing n images with 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 an image list containing n images with specified size, filled with specified value.

- 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 an image list containing n images with specified size and specified pixel values (int version).

- 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 an image list containing n images with specified size and specified pixel values (double version).

- template<typename t >
  CImgList (const unsigned int n, const CImg< t > &img, const bool shared=false)

  Construct a list containing n copies of the image img.

- template<typename t >
  CImgList (const CImg< t > &img, const bool shared=false)

Construct an image list from one image.

- template<typename t1 , typename t2 >
  CImgList (const CImg< t1 > &img1, const CImg< t2 > &img2, const bool shared=false)

  Construct an image 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 shared=false)

  Construct an image 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 shared=false)

  Construct an image 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 shared=false)

  Construct an image 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 shared=false)

  Construct an image 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 shared=false)

  Construct an image 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 shared=false)

  Construct an image list from eight images.

- template<typename t >
  CImgList (const CImgList< t > &list)

  Default copy constructor.

- **CImgList** (const CImgList< T > &list)
- template<typename t >
  CImgList (const CImgList< t > &list, const bool shared)

  Advanced copy constructor.

- CImgList (const char *const filename)

  Construct an image list from a filename.

- CImgList (const CImgDisplay &disp)

  Construct an image list from a display.

- CImgList< T > get_shared ()

  Return a shared instance of the list.

- const CImgList< T > get_shared () const

  Return a shared instance of the list.

- CImgList< T > & clear ()

  In-place version of the default constructor.

- CImgList< T > & assign ()

  In-place version of the default constructor and default destructor.

- CImgList< T > & assign (const unsigned int n)

  In-place version of the corresponding constructor.

- 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)

  In-place version of the corresponding constructor.

- 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)

  In-place version of the corresponding constructor.

- 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,...)

  In-place version of the corresponding constructor.

- 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,...)

  In-place version of the corresponding constructor.

- CImgList< T > & assign (const CImgList< T > &list, const bool shared=false)

  In-place version of the copy constructor.

- template<typename t >
  CImgList< T > & **assign** (const CImgList< t > &list, const bool shared=false)
- template<typename t >
  CImgList< T > & assign (const unsigned int n, const CImg< t > &img, const bool shared=false)

  In-place version of the corresponding constructor.

- template<typename t >
  CImgList< T > & assign (const CImg< t > &img, const bool shared=false)

  In-place version of the corresponding constructor.

- template<typename t1 , typename t2 >
  CImgList< T > & assign (const CImg< t1 > &img1, const CImg< t2 > &img2, const bool shared=false)

In-place version of the corresponding constructor.

- template<typename t1 , typename t2 , typename t3 >
  CImgList< T > & assign (const CImg< t1 > &img1, const CImg< t2 > &img2, const CImg< t3 > &img3, const bool shared=false)

  In-place version of the corresponding constructor.

- 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 shared=false)

  In-place version of the corresponding constructor.

- 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 shared=false)

  In-place version of the corresponding constructor.

- 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 shared=false)

  In-place version of the corresponding constructor.

- 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 shared=false)

  In-place version of the corresponding constructor.

- 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 shared=false)

  In-place version of the corresponding constructor.

- CImgList< T > & assign (const char *const filename)

  In-place version of the corresponding constructor.

- CImgList< T > & assign (const CImgDisplay &disp)

  In-place version of the corresponding constructor.

- template<typename t >
  CImgList< t > & move_to (CImgList< t > &list)

  Move the content of the instance image list into another one.

- template<typename t >
  CImgList< t > & **move_to** (CImgList< t > &list, const unsigned int pos)
- CImgList< T > & swap (CImgList< T > &list)

  Swap all fields of two CImgList instances (use with care !).

- static CImgList< T > & empty ()

  Return a reference to an empty list.

Overloaded Operators

- CImg< T > & operator() (const unsigned int pos)

  Return a reference to the i-th element of the image list.

- const CImg< T > & **operator()** (const unsigned int pos) 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)

  Return a reference to (x,y,z,c) pixel of the pos-th 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
- operator const CImg< T > * () const

  Return address of the image vector.

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

  Operator=().

- CImgList< T > & operator= (const CImgDisplay &disp)

  Operator=().

- template<typename t >
  CImgList< T > & operator= (const CImgList< t > &list)

  Operator=().

- CImgList< T > & **operator=** (const CImgList< T > &list)
- CImgList< T > & operator= (const char *const filename)

  Operator=().

- CImgList< T > operator+ () const

  Operator+() (unary).

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

  Operator,().

- template<typename t >
  CImgList< T > & operator, (const CImgList< t > &list)

  Operator,().

- CImg< T > operator> (const char axis) const

  Operator>().

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

  Operator<().

Instance Characteristics

- int width () const

  Return the size of the list.

- unsigned int size () const

  Return the size of the list.

- CImg< T > * data ()

  Return a pointer to the image buffer.

- const CImg< T > * **data** () const
- CImg< T > * data (const unsigned int l)

  Return a pointer to the image buffer.

- const CImg< T > * **data** (const unsigned int l) const
- iterator begin ()

  Returns an iterator to the beginning of the vector (STL-compliant name).

- const_iterator **begin** () const
- iterator end ()

  Returns an iterator just past the last element (STL-compliant name).

- const_iterator **end** () const
- CImg< T > & front ()

  Returns a reference to the first element (STL-compliant name).

- const CImg< T > & **front** () const
- const CImg< T > & back () const

  Return a reference to the last image (STL-compliant name).

- CImg< T > & **back** ()
- CImg< T > & at (const int pos)

  Read an image in specified position.

- T & atNXYZC (const int pos, const int x, const int y, const int z, const int c, const T out_val)

  Read a 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_val) const
- T & atNXYZC (const int pos, const int x, const int y, const int z, const int c)

  Read a pixel value with Neumann boundary conditions.

- T **atNXYZC** (const int pos, const int x, const int y, const int z, const int c) const
- T & atNXYZ (const int pos, const int x, const int y, const int z, const int c, const T out_val)

  Read a pixel value with Dirichlet 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, const T out_val) const
- T & atNXYZ (const int pos, const int x, const int y, const int z, const int c=0)

  Read a 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
- T & atNXY (const int pos, const int x, const int y, const int z, const int c, const T out_val)

  Read a 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_val) const
- T & atNXY (const int pos, const int x, const int y, const int z=0, const int c=0)

  Read a 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
- T & atNX (const int pos, const int x, const int y, const int z, const int c, const T out_val)

  Read a 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_val) const
- T & atNX (const int pos, const int x, const int y=0, const int z=0, const int c=0)

  Read a 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
- T & atN (const int pos, const int x, const int y, const int z, const int c, const T out_val)

  Read a 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_val) const
- T & atN (const int pos, const int x=0, const int y=0, const int z=0, const int c=0)

  Read a 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
- 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 a string describing the type of the image pixels in the list (template parameter `T`).

Instance Checking

- bool is_empty () const

  Return `true` if list is empty.

- bool is_sameN (const unsigned int n) const

  Return `true` if list if of specified size.

- template<typename t >
  bool is_sameN (const CImgList< t > &list) const

  Return `true` if list if of specified size.

- 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
- bool **is_sameNXYZC** (const unsigned int n, const unsigned int dx, const unsigned int dy, const unsigned int dz, const unsigned int dc) const
- bool containsNXYZC (const int n, const int x=0, const int y=0, const int z=0, const int c=0) const

Return `true` if the list contains the pixel (n,x,y,z,c).

- bool containsN (const int n) const

  Return `true` if the list contains the image (n).

- template<typename t >
  bool contains (const T &pixel, t &n, t &x, t &y, t &z, t &c) const

  Return `true` if one of the image list contains the specified referenced value. If true, set coordinates (n,x,y,z,c).

- template<typename t >
  bool contains (const T &pixel, t &n, t &x, t &y, t &z) const

  Return `true` if one of the image list contains the specified referenced value. If true, set coordinates (n,x,y,z).

- template<typename t >
  bool contains (const T &pixel, t &n, t &x, t &y) const

  Return `true` if one of the image list contains the specified referenced value. If true, set coordinates (n,x,y).

- template<typename t >
  bool contains (const T &pixel, t &n, t &x) const

  Return `true` if one of the image list contains the specified referenced value. If true, set coordinates (n,x).

- template<typename t >
  bool contains (const T &pixel, t &n) const

  Return `true` if one of the image list contains the specified referenced value. If true, set coordinates (n).

- bool contains (const T &pixel) const

  Return `true` if one of the image list contains the specified referenced value.

- template<typename t >
  bool contains (const CImg< T > &img, t &n) const

  Return `true` if the list contains the image 'img'. If true, returns the position (n) of the image in the list.

- bool contains (const CImg< T > &img) const

  Return `true` 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
- T & max ()

  Return a reference to the maximum pixel value of the instance list.

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

  Return a reference to the minimum pixel value of the instance list.

- template<typename t >
  const T & **min_max** (t &max_val) const
- template<typename t >
  T & max_min (t &min_val)

  Return a reference to the minimum pixel value of the instance list.

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

List Manipulation

- template<typename t >
  CImgList< T > & insert (const CImg< t > &img, const unsigned int pos=~0U, const bool 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 shared=false)
- template<typename t >
  CImgList< T > **get_insert** (const CImg< t > &img, const unsigned int pos=~0U, const bool shared=false) const
- 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
- template<typename t >
  CImgList< T > & insert (const unsigned int n, const CImg< t > &img, const unsigned int pos=~0U, const bool 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 shared=false) const
- template<typename t >
  CImgList< T > & insert (const CImgList< t > &list, const unsigned int pos=~0U, const bool 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 shared=false) const
- template<typename t >
  CImgList< T > & insert (const unsigned int n, const CImgList< t > &list, const unsigned int pos=~0U, const bool 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 shared=false) const
- CImgList< T > & remove (const unsigned int pos1, const unsigned int pos2)

Remove the images from positions `pos1` to `pos2`.

- CImgList< T > **get_remove** (const unsigned int pos1, const unsigned int pos2) const
- CImgList< T > & remove (const unsigned int pos)

  Remove the image at position `pos` from the image list.

- CImgList< T > **get_remove** (const unsigned int pos) const
- CImgList< T > & remove ()

  Remove the last image from the image list.

- CImgList< T > **get_remove** () const
- CImgList< T > & reverse ()

  Reverse list order.

- CImgList< T > **get_reverse** () const
- CImgList< T > & images (const unsigned int i0, const unsigned int i1)

  Get a sub-list.

- CImgList< T > **get_images** (const unsigned int i0, const unsigned int i1) const
- CImgList< T > get_shared_images (const unsigned int i0, const unsigned int i1)

  Get a shared sub-list.

- const CImgList< T > **get_shared_images** (const unsigned int i0, const unsigned int i1) const
- CImg< T > get_append (const char axis, const char align='p') const

  Return a single image which is the concatenation 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
- template<typename t >
  CImgList< T > & push_back (const CImg< t > &img)

  Insert image `img` at the end of the list (STL-compliant name).

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

  Insert image `img` at the front of the list (STL-compliant name).

- template<typename t >
  CImgList< T > & push_back (const CImgList< t > &list)

  Insert list `list` at the end of the current list (STL-compliant name).

- template<typename t >
  CImgList< T > & push_front (const CImgList< t > &list)

  Insert list `list` at the front of the current list (STL-compliant name).

- CImgList< T > & pop_back ()

  Remove last element of the list (STL-compliant name).

- CImgList< T > & pop_front ()

Remove first element of the list (STL-compliant name).

- CImgList< T > & erase (const iterator iter)

  Remove the element pointed by iterator `iter` (STL-compliant name).

Data Input

- CImgList< T > & load (const char *const filename)

  Load an image list from a file.

- CImgList< T > & load_cimg (const char *const filename)

  Load an image list from a .cimg file.

- CImgList< T > & load_cimg (std::FILE *const file)

  Load an image 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 sub-image 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.

- CImgList< T > & load_parrec (const char *const filename)

  Load an image list from a PAR/REC (Philips) file.

- CImgList< T > & load_yuv (const char *const filename, const unsigned int sizex, const unsigned int sizey, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool yuv2rgb=true)

  Load an image sequence from a YUV file.

- CImgList< T > & load_yuv (std::FILE *const file, const unsigned int sizex, const unsigned int sizey, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool yuv2rgb=true)

  Load an image sequence from a YUV file.

- 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 (MPEG,AVI) using the external tool 'ffmpeg'.

- 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 a TIFF file.

- static CImgList< T > **get_load** (const char *const filename)
- static CImgList< T > **get_load_cimg** (const char *const filename)
- static CImgList< T > **get_load_cimg** (std::FILE *const file)
- 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)
- 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)
- static CImgList< T > **get_load_parrec** (const char *const filename)
- static CImgList< T > **get_load_yuv** (const char *const filename, const unsigned int sizex, const unsigned int sizey=1, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool yuv2rgb=true)
- static CImgList< T > **get_load_yuv** (std::FILE *const file, const unsigned int sizex, const unsigned int sizey=1, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool yuv2rgb=true)
- 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)
- static CImgList< T > **get_load_ffmpeg_external** (const char *const filename)
- static CImgList< T > **get_load_gzip_external** (const char *const filename)
- template<typename tf , typename tc >
  static CImgList< T > **get_load_off** (const char *const filename, CImgList< tf > &primitives, CImgList< tc > &colors)
- 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)

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 char align='p') 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 char align='p') 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 char align='p') const

  Display the current CImgList instance in a new display window.

- const CImgList< T > & save (const char *const filename, const int number=-1) const

  Save an image list into a file.

- const CImgList< T > & save_ffmpeg (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int fps=25, const unsigned int bitrate=2048) const

  Save an image sequence, using FFMPEG library.

- const CImgList< T > & save_yuv (const char *const filename=0, const bool rgb2yuv=true) const

  Save an image sequence into a YUV file.

- const CImgList< T > & save_yuv (std::FILE *const file, const bool rgb2yuv=true) const

  Save an image sequence into a YUV file.

- const CImgList< T > & save_cimg (std::FILE *file, const bool compress=false) const

  Save an image list into a CImg file (RAW binary file + simple header).

- const CImgList< T > & save_cimg (const char *const filename, const bool compress=false) const

  Save an image list into a CImg file (RAW binary file + simple header).

- 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 instance image 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 instance image into into an existing .cimg file, at specified coordinates.

- const CImgList< T > & save_gzip_external (const char *const filename) const

  Save a file in TIFF format.

- const CImgList< T > & save_ffmpeg_external (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const char *const codec="mpeg2video", const unsigned int fps=25, const unsigned int bitrate=2048) const

  Save an image sequence using the external tool 'ffmpeg'.

- static bool **is_saveable** (const char *const filename)
- 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)

  Create an empty .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)

  Create an empty .cimg file with specified dimensions.

Others

- CImgList< T > & crop_font ()

  Create an auto-cropped font (along the X axis) from a input font `font`.

- CImgList< T > **get_crop_font** () const
- CImgList< T > & FFT (const char axis, const bool invert=false)

  Compute a 1-D Fast Fourier Transform, along specified axis.

- CImgList< Tfloat > **get_FFT** (const char axis, const bool invert=false) const
- CImgList< T > & FFT (const bool invert=false)

  Compute a n-d Fast Fourier Transform.

- CImgList< Tfloat > **get_FFT** (const bool invert=false) const
- CImgList< T > & reverse_object3d ()

  Invert primitives orientation of a 3d object.

- CImgList< T > **get_reverse_object3d** () const
- static const CImgList< T > & font (const unsigned int font_height, const bool variable_size=true)

  Return a CImg pre-defined font with desired size.

8.4.1 Detailed Description

template<typename T> struct cimg_library::CImgList< T >

Class representing list of images CImg<T>.

8.4.2 Member Function Documentation

8.4.2.1 CImgList<T>& clear ( )

In-place version of the default constructor.

This function is strictly equivalent to assign() and has been introduced for having a STL-compliant function name.

8.4.2.2 CImgList<T> operator+ ( ) const

Operator+() (unary).

Writting '+list' is a convenient shortcut to 'CImgList<T>(list,false)' (forces a copy with non-shared elements).

8.4.2.3 CImg<T> get_append (const char *axis*, const char *align* = `'p'`) const

Return a single image which is the concatenation of all images of the current CImgList instance.

Parameters

axis : specify the axis for image concatenation. Can be 'x','y','z' or 'c'.

align : specify the alignment for image concatenation. Can be 'p' (top), 'c' (center) or 'n' (bottom).

Returns

A CImg<T> image corresponding to the concatenation is returned.

8.4.2.4 const CImgList<T>& display (CImgDisplay & *disp*, const char *axis* = *'x'*, const char *align* = *'p'*) const

Display the current CImgList instance in an existing CImgDisplay window (by reference).

This function displays the list images of the current CImgList instance into an existing CImgDisplay window. Images of the list are concatenated in a single temporarly image for visualization purposes. The function returns immediately.

Parameters

disp : reference to an existing CImgDisplay instance, where the current image list will be displayed.

axis : specify the axis for image concatenation. Can be 'x','y','z' or 'c'.

align : specify the alignment for image concatenation. Can be 'p' (top), 'c' (center) or 'n' (bottom).

Returns

A reference to the current CImgList instance is returned.

8.4.2.5 const CImgList<T>& display (CImgDisplay & *disp*, const bool *display_info*, const char *axis* = *'x'*, const char *align* = *'p'*) const

Display the current CImgList instance in a new display window.

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 concatenated 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.

Parameters

title : specify the title of the opening display window.

axis : specify the axis for image concatenation. Can be 'x','y','z' or 'c'.

align : specify the alignment for image concatenation. Can be 'p' (top), 'c' (center) or 'n' (bottom).

Returns

A reference to the current CImgList instance is returned.

8.4.2.6 const CImgList<T>& save (const char *const *filename*, const int *number* = *-1*) const

Save an image list into a file.

Depending on the extension of the given filename, a file format is chosen for the output file.

8.4.2.7 const CImgList<T>& save_gzip_external (const char *const *filename*) const

Save a file in TIFF format.

Save an image list as a gzipped file, using external tool 'gzip'.

8.4.2.8 static const CImgList<T>& font (const unsigned int *font_height,* const bool *variable_size* = `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)

fixed_size = tell if the font has a fixed or variable width.

Index