The CImg Library Reference Manual

1.1.6

Generated by Doxygen 1.4.6

Thu Sep 21 15:29:00 2006

Contents

1 The CImg Library Main Page 1

2 The CImg Library Module Documentation 1

3 The CImg Library Namespace Documentation 15

4 The CImg Library Class Documentation 21

1 The CImg Library Main Page

This is the reference documentation of `the CImg Library`. These HTML pages have been generated using `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/documentation/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.

2 The CImg Library Module Documentation

2.1 Introduction to the CImg Library

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.

2.1.1 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**(p. 15). 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**(p. 16) 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**(p. 16) 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**(p. 21)<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::CImgl**(p. 119)<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**(p. 112) 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**(p. 4)).

- The class **cimg_library::CImgStats**(p. 131) represents image statistics. Use it to compute the minimum, maximum, mean and variance of pixel values of images, as well as the corresponding min/max pixel location.

- The class **cimg_library::CImgException**(p. 118) (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 five classes is **enough** to get benefit of the CImg Library functionalities.

2.1.2 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 com
  img.fill(0);                                 // Set pixel values to 0 (color : black)
  unsigned char purple[3]={255,0,255};         // Define a purple color
  img.draw_text("Hello World",100,100,purple); // Draw a purple "Hello world" at coordinates (100,100).
  img.display("My first CImg code");           // Display the image in a display window.
  return 0;
}
```

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

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

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

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

2.1.3 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 and Visual Studio.NET** : 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 -lpthread -lX11
  ```

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

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

2.1.4 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.**(p. 5) section.

2.2 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_type` would be done like this :

```
#define cimg_display_type 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_type` : 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_color_terminal` : 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_debug` : 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, which is the default value), or 2 (high debug messages). Note that setting this value to 2 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_convert_path` : This variables tells the library where the ImageMagick's *convert* tool is located. Setting this variable should not be necessary if ImageMagick is installed on a standard directory, or if *convert* is in your system PATH variable. This macro should be defined only if the ImageMagick's *convert* tool is not found automatically, when trying to read compressed image format (GIF,PNG,...). See also **cimg_library::CImg::get_load_convert()**(p. 54) and **cimg_library::CImg::save_convert()**(p. 110) for more informations.

- `cimg_temporary_path` : This variable tells the library where it can find a directory to store temporary files. Setting this variable should not be necessary if you are running on a standard system. This macro should be defined only when troubles are encountered when trying to read compressed image format (GIF,PNG,...). See also **cimg_library::CImg::get_load_convert()**(p. 54) and **cimg_library::CImg::save_convert()**(p. 110) for more informations.

- `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.

- `cimgl_plugin` : Same as `cimg_plugin`, but to add features to the CImgl<T> class.

- `cimgdisplay_plugin` : Same as `cimg_plugin`, but to add features to the CImgDisplay<T> class.

- `cimgstats_plugin` : Same as `cimg_plugin`, but to add features to the CImgStats<T> class.

All these compilation variables can be checked, using the function **cimg_library::cimg::info()**(p. 19), which displays a list of the different configuration variables and their values on the standard error output.

2.3 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 with the mouse will draw the intensity profiles of (R,G,B) 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[3]={255,0,0}, green[3]={0,255,0}, blue[3]={0,0,255};
  image.blur(2.5);
  CImgDisplay main_disp(image,"Click a point"), draw_disp(visu,"Intensity profile");
  while (!main_disp.is_closed && !draw_disp.is_closed) {
    main_disp.wait();
    if (main_disp.button && main_disp.mouse_y>=0) {
      const int y = main_disp.mouse_y;
      visu.fill(0).draw_graph(image.get_crop(0,y,0,0,image.dimx()-1,y,0,0),red,0,256,0);
      visu.draw_graph(image.get_crop(0,y,0,1,image.dimx()-1,y,0,1),green,0,256,0);
      visu.draw_graph(image.get_crop(0,y,0,2,image.dimx()-1,y,0,2),blue,0,256,0).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[3]={255,0,0}, green[3]={0,255,0}, blue[3]={0,0,255};
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

I don't think you need more explanations !

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

2.4 Using Drawing Functions.

2.4.1 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

2.5 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**(p. 8)
- **Loops over image dimensions**(p. 8)
- **Loops over interior regions and borders.**(p. 9)
- **Loops using neighborhoods.**(p. 10)

2.5.1 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**(p. 21) image. Two macros are defined for this purpose :

- **cimg_map(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**(p. 21) 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_map(img,ptr,float) { *ptr=0; }      // Equivalent to 'img.fill(0);'
```

- **cimg_mapoff(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_mapoff(img,off) { img[off]=0; }  // Equivalent to 'img.fill(0);'
```

2.5.2 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_mapX(img,x)** : equivalent to : `for (int x=0; x<img.dimx(); x++).`
- **cimg_mapY(img,y)** : equivalent to : `for (int y=0; y<img.dimy(); y++).`
- **cimg_mapZ(img,z)** : equivalent to : `for (int z=0; z<img.dimz(); z++).`

- **cimg_mapV(img,v)** : equivalent to : `for (int v=0; v<img.dimv(); v++)`.

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

- **cimg_mapXY(img,x,y)** : equivalent to : `cimg_mapY(img,y) cimg_mapX(img,x)`.
- **cimg_mapXZ(img,x,z)** : equivalent to : `cimg_mapZ(img,z) cimg_mapX(img,x)`.
- **cimg_mapYZ(img,y,z)** : equivalent to : `cimg_mapZ(img,z) cimg_mapY(img,y)`.
- **cimg_mapXV(img,x,v)** : equivalent to : `cimg_mapV(img,v) cimg_mapX(img,x)`.
- **cimg_mapYV(img,y,v)** : equivalent to : `cimg_mapV(img,v) cimg_mapY(img,y)`.
- **cimg_mapZV(img,z,v)** : equivalent to : `cimg_mapV(img,v) cimg_mapZ(img,z)`.
- **cimg_mapXYZ(img,x,y,z)** : equivalent to : `cimg_mapZ(img,z) cimg_mapXY(img,x,y)`.
- **cimg_mapXYV(img,x,y,v)** : equivalent to : `cimg_mapV(img,v) cimg_mapXY(img,x,y)`.
- **cimg_mapXZV(img,x,z,v)** : equivalent to : `cimg_mapV(img,v) cimg_mapXZ(img,x,z)`.
- **cimg_mapYZV(img,y,z,v)** : equivalent to : `cimg_mapV(img,v) cimg_mapYZ(img,y,z)`.
- **cimg_mapXYZV(img,x,y,z,v)** : equivalent to : `cimg_mapV(img,v) cimg_mapXYZ(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**(p. 21) 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_mapXYV(img,x,y,v) { img(x,y,v) = (x+y)*(v+1)/6; }
img.display("Color gradient");
```

2.5.3 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_imapX(img,x,n)** : Loop along the x-axis, except for pixels inside a border of `n` pixels wide.
- **cimg_imapY(img,y,n)** : Loop along the y-axis, except for pixels inside a border of `n` pixels wide.
- **cimg_imapZ(img,z,n)** : Loop along the z-axis, except for pixels inside a border of `n` pixels wide.
- **cimg_imapV(img,v,n)** : Loop along the v-axis, except for pixels inside a border of `n` pixels wide.
- **cimg_imapXY(img,x,y,n)** : Loop along the (x,y)-axes, excepted for pixels inside a border of `n` pixels wide.
- **cimg_imapXYZ(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_bmapX(img,x,n)** : Loop along the x-axis, only for pixels inside a border of `n` pixels wide.
- **cimg_bmapY(img,y,n)** : Loop along the y-axis, only for pixels inside a border of `n` pixels wide.
- **cimg_bmapZ(img,z,n)** : Loop along the z-axis, only for pixels inside a border of `n` pixels wide.
- **cimg_bmapV(img,v,n)** : Loop along the z-axis, only for pixels inside a border of `n` pixels wide.
- **cimg_bmapXY(img,x,y,n)** : Loop along the (x,y)-axes, only for pixels inside a border of `n` pixels wide.
- **cimg_bmapXYZ(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**(p. 21) 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_imapXY(img,x,y,50) img(x,y) = x+y;
cimg_bmapXY(img,x,y,50) img(x,y) = x-y;
img.display();
```

2.5.4 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.

2.5.4.1 Neighborhood-based loops for 2D images For 2D images, the neighborhood-based loop macros are :

- **cimg_map2x2x1(img,x,y,z,v,I)** : Loop along the (x,y)-axes using a centered 2x2 neighborhood.
- **cimg_map3x3x1(img,x,y,z,v,I)** : Loop along the (x,y)-axes using a centered 3x3 neighborhood.
- **cimg_map4x4x1(img,x,y,z,v,I)** : Loop along the (x,y)-axes using a centered 4x4 neighborhood.
- **cimg_map5x5x1(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**(p. 11)).

2.5.4.2 Neighborhood-based loops for 3D images For 3D images, the neighborhood-based loop macros are :

- **cimg_map2x2x2(img,x,y,z,v,I)** : Loop along the (x,y,z)-axes using a centered 2x2x2 neighborhood.
- **cimg_map3x3x3(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**(p. 11)).

2.5.4.3 Defining neighborhoods The CImg library defines a neighborhood as a set of named *variables* or *references*, declared using specific CImg macros :

- **CImg_2x2x1(I,type)** : Define a 2x2 neighborhood named `I`, of type `type`.
- **CImg_3x3x1(I,type)** : Define a 3x3 neighborhood named `I`, of type `type`.
- **CImg_4x4x1(I,type)** : Define a 4x4 neighborhood named `I`, of type `type`.
- **CImg_5x5x1(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_3x3x1(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*).

2.5.4.4 Neighborhood as a reference It is also possible to define neighborhood variables as references to classical C-arrays or CImg<T> images, instead of allocating new variables. This is done by adding `_ref` to the macro names used for the neighborhood definition :

- **CImg_2x2x1_ref(I,type,tab)** : Define a 2x2 neighborhood named `I`, of type `type`, as a reference to `tab`.
- **CImg_3x3x1_ref(I,type,tab)** : Define a 3x3 neighborhood named `I`, of type `type`, as a reference to `tab`.
- **CImg_4x4x1_ref(I,type,tab)** : Define a 4x4 neighborhood named `I`, of type `type`, as a reference to `tab`.
- **CImg_5x5x1_ref(I,type,tab)** : Define a 5x5 neighborhood named `I`, of type `type`, as a reference to `tab`.
- **CImg_2x2x2_ref(I,type,tab)** : Define a 2x2x2 neighborhood named `I`, of type `type`, as a reference to `tab`.
- **CImg_3x3x3_ref(I,type,tab)** : Define a 3x3x3 neighborhood named `I`, of type `type`, as a reference to `tab`.

`tab` can be a one-dimensionnal C-style array, or a non empty `CImg<T>` image. Both objects must have same sizes as the considered neighborhoods.

2.5.4.5 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_map3x3x3()` loop macro :

```
CImg<float> volume("IRM.hdr");        // Load an IRM volume from an Analyze7.5 file
CImg_3x3x3(I,float);                  // Define a 3x3x3 neighborhood
CImg<float> gradnorm(volume);         // Create an image with same size as 'volume'
cimg_map3x3x3(volume,x,y,z,0,I) {     // Loop over the volume, using the neighborhood I
  const float ix = 0.5f*(Incc-Ipcc);  // Compute the derivative along the x-axis.
  const float iy = 0.5f*(Icnc-Icpc);  // Compute the derivative along the y-axis.
  const float iz = 0.5f*(Iccn-Iccp);  // Compute the derivative along the z-axis.
  gradnorm(x,y,z) = std::sqrt(ix*ix+iy*iy+iz*iz);  // Set the gradient norm in the destination image
}
gradnorm.display("Gradient norm");
```

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

```
CImg<unsigned char> src("image_color.jpg"), dest(src,false), neighbor(5,5);  // Image definitions.
typedef unsigned char uchar;             // Avoid space in the second parameter of the macro CImg_5x5x1
CImg_5x5x1_ref(N,uchar,neighbor);        // Define a 5x5 neighborhood as a reference to the 5x5 image ne
cimg_mapV(src,k)                         // Standard loop on color channels
   cimg_map5x5x1(src,x,y,0,k,N)          // 5x5 neighborhood loop.
     dest(x,y,k) = neighbor.sum()/(5*5); // Averaging pixels to filter the color image.
CImgl<unsigned char> visu(src,dest);
visu.display("Original + Filtered");     // Display both original and filtered image.
```

Note that in this example, we didn't use directly the variables Nbb,Nbp,..,Ncc,... since there are only references to the neighborhood image `neighbor`. We rather used a member function of `neighbor`.

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

2.6 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.

2.7 How pixel data are stored with CImg.

TODO

2.8 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,...

2.9 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. Two macros `cimg_usage()` and `cimg_option()` are defined for this purpose. Using these macros allows to easily retrieve options values from the command line. Moreover, invoking the corresponding executable with the option `-h` or `-help` will automatically display the program usage, followed by the list of requested options.

2.9.1 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.

2.9.2 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.

2.9.3 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",(const char*)0,"Output image file");
  const double sigma   = cimg_option("-s",1.0,"Standard variation of the gaussian smoothing");
  const  int nblevels  = cimg_option("-n",16,"Number of quantification levels");
  const bool hidden    = cimg_option("-hidden",false,0);      // This is a hidden option

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

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

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

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

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

You found me !
```

Warning:

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

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

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

2.9.4 How to learn more about command line options ?

You should take a look at the examples `examples/inrcast.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.

3 The CImg Library Namespace Documentation

3.1 cimg_library Namespace Reference

Namespace that encompasses all classes and functions of the CImg library.

Classes

- struct **CImgException**

  Class which is thrown when an error occured during a CImg library function call.

- struct **CImgStats**

  Class used to compute basic statistics on pixel values of a **CImg**(p. 21)*<T> image.*

- struct **CImgDisplay**

  This class represents a window which can display **CImg**(p. 21)*<T> 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 **CImgl**

  Class representing list of images CImg<T>.

Namespaces

- namespace **cimg**

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

3.1.1 Detailed Description

Namespace that encompasses all classes and functions of the CImg library.

This namespace is defined to avoid class names collisions that could happen with the include of other C++ header files. Anyway, it should not happen very often and you may start most of your programs with

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

to simplify the declaration of CImg Library objects variables afterward.

3.2 cimg_library::cimg Namespace Reference

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

Functions

- template<typename tfunc, typename tp, typename tf> void **marching_cubes** (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 float resx, const float resy, const float resz, **CImgl**< tp > &points, **CImgl**< tf > &primitives, const bool invert_faces)

  Polygonize an implicit function.

- template<typename tfunc, typename tp, typename tf> void **marching_squares** (const tfunc &func, const float isovalue, const float x0, const float y0, const float x1, const float y1, const float resx, const float resy, **CImgl**< tp > &points, **CImgl**< tf > &primitives)

  Polygonize an implicit 2D function by the marching squares algorithm.

- const char * **convert_path** ()

  Return path of the ImageMagick's `convert` *tool.*

- const char * **medcon_path** ()

  Return path of the `XMedcon` *tool.*

- const char * **temporary_path** ()

  Return path to store temporary files.

- bool **endian** ()

  Return `false` *for little endian CPUs (Intel),* `true` *for big endian CPUs (Motorola).*

- void **info** ()

  Print informations about CImg environement variables.

- 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 **abs** (const T &a)

  Return the absolute value of a.

- template<typename T> const T & **min** (const T &a, const T &b)

  Return the minimum between a *and* b.

- template<typename T> const T & **min** (const T &a, const T &b, const T &c)

  Return the minimum between a,b *and* c.

- template<typename T> const T & **min** (const T &a, const T &b, const T &c, const T &d)

  Return the minimum between a,b,c *and* d.

- template<typename T> const T & **max** (const T &a, const T &b)

  Return the maximum between a *and* b.

- template<typename T> const T & **max** (const T &a, const T &b, const T &c)

  Return the maximum between a,b *and* c.

- template<typename T> const T & **max** (const T &a, const T &b, const T &c, const T &d)

  Return the maximum between a,b,c *and* d.

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

  Return the sign of x.

- template<typename T> unsigned long **nearest_pow2** (const T &x)

  Return the nearest power of 2 higher than x.

- double **mod** (const double &x, const double &m)

  Return x *modulo* m *(generic modulo).*

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

  *Return minmod(*a,b*).*

- double **rand** ()

  Return a random variable between [0,1], followin a uniform distribution.

- double **crand** ()

  Return a random variable between [-1,1], following a uniform distribution.

- double **grand** ()

  Return a random variable following a gaussian distribution and a standard deviation of 1.

- template<typename t> int **dialog** (const char *title, const char *msg, const char *button1_txt, const char *button2_txt, const char *button3_txt, const char *button4_txt, const char *button5_txt, const char *button6_txt, 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.

3.2.1 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**(p. 16)`;` in your source code, since a lot of functions of the `cimg::` namespace have prototypes similar to standard C functions defined in the global namespace `::`.

3.2.2 Function Documentation

3.2.2.1 const char* cimg_library::cimg::convert_path ()

Return path of the ImageMagick's `convert` tool.

If you have installed the `ImageMagick package` in a standard directory, this function should return the correct path of the `convert` tool used by the CImg Library to load and save compressed image formats. Conversely, if the `convert` executable is not auto-detected by the function, you can define the macro `cimg_convert_path` with the correct path of the `convert` executable, before including `CImg.h` in your program :

```
#define cimg_convert_path "/users/thatsme/local/bin/convert"
#include "CImg.h"

int main() {
  CImg<> img("my_image.jpg");     // Read a JPEG image file.
  return 0;
}
```

Note that non compressed image formats can be read without installing ImageMagick.

See also:
temporary_path()(p. 19), get_load_convert(), load_convert(), save_convert().

3.2.2.2 const char* cimg_library::cimg::medcon_path ()

Return path of the `XMedcon` tool.

If you have installed the `XMedcon package` in a standard directory, this function should return the correct path of the `medcon` tool used by the CIg Library to load DICOM image formats. Conversely, if the `medcon` executable is not auto-detected by the function, you can define the macro `cimg_medcon_path` with the correct path of the `medcon` executable, before including `CImg.h` in your program :

```
#define cimg_medcon_path "/users/thatsme/local/bin/medcon"
#include "CImg.h"

int main() {
  CImg<> img("my_image.dcm");     // Read a DICOM image file.
  return 0;
}
```

Note that `medcon` is only needed if you want to read DICOM image formats.

See also:
temporary_path()(p. 19), get_load_dicom(), load_dicom().

3.2.2.3 const char* cimg_library::cimg::temporary_path ()

Return path to store temporary files.

If you are running on a standard Unix or Windows system, this function should return a correct path where temporary files can be stored. If such a path is not auto-detected by this function, you can define the macro `cimg_temporary_path` with a correct path, before including `CImg.h` in your program :

```
#define cimg_temporary_path "/users/thatsme/tmp"
#include "CImg.h"

int main() {
  CImg<> img("my_image.jpg");   // Read a JPEG image file (using the defined temporay path).
  return 0;
}
```

A temporary path is necessary to load and save compressed image formats, using `convert` or `medcon`.

See also:
convert_path()(p. 18), get_load_convert(), load_convert(), save_convert(), get_load_dicom(), load_dicom().

3.2.2.4 void cimg_library::cimg::info ()

Print informations about CImg environement variables.

Printing is done on the standart error output.

3.2.2.5 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.

See also:
wait(), **time()**(p. 17).

3.2.2.6 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**()(p. 19) but the waiting time is computed with regard to the last call of wait(). It may be used to temporize your program properly.

See also:
sleep()(p. 19), **time**()(p. 17).

3.2.2.7 double cimg_library::cimg::mod (const double & *x*, const double & *m*)

Return `x` modulo `m` (generic modulo).

This modulo function accepts negative and floating-points modulo numbers `m`.

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

Return minmod(`a`,`b`).

The operator minmod(`a`,`b`) is defined to be :

- minmod(`a`,`b`) = min(`a`,`b`), if (`a * b`)>0.
- minmod(`a`,`b`) = 0, if (`a * b`)<=0

3.2.2.9 int cimg_library::cimg::dialog (const char * *title*, const char * *msg*, const char * *button1_txt*, const char * *button2_txt*, const char * *button3_txt*, const char * *button4_txt*, const char * *button5_txt*, const char * *button6_txt*, 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_txt = Label of the 1st button.
button2_txt = Label of the 2nd button.
button3_txt = Label of the 3rd button.
button4_txt = Label of the 4th button.
button5_txt = Label of the 5th button.
button6_txt = 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.

4 The CImg Library Class Documentation

4.1 CImg Struct Template Reference

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

Constructors-Destructor-Copy

- **CImg** ()

  Default constructor.

- ~**CImg** ()

  Destructor.

- **CImg** & **assign** ()

  In-place version of the default constructor and the default destructor.

- **CImg** & **clear** ()

  Equivalent to **assign()**(p. 21) *(introduced as a STL-like function name).*

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

  Copy constructor.

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

  In-place version of the copy constructor.

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

  Copy constructor allowing to force the shared state of the instance image.

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

  Assignement operator.

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

  Construct an image of size (`dx,dy,dz,dv`*) with pixels of type* `T`.

- **CImg** & **assign** (const unsigned int dx, const unsigned int dy=1, const unsigned int dz=1, const unsigned int dv=1)

  In-place version of the previous constructor.

- **CImg** (const unsigned int dx, const unsigned int dy, const unsigned int dz, const unsigned int dv, const T &val)

  Construct an image of size (`dx,dy,dz,dv`*) with pixels of type* `T` *and set all pixel values to* `val`.

- **CImg** & **assign** (const unsigned int dx, const unsigned int dy, const unsigned int dz, const unsigned int dv, const T &val)

  In-place version of the previous constructor.

- **CImg** (const char *const filename)

  Construct an image from the filename `filename`.

- **CImg** & **assign** (const char *const filename)

  In-place version of the previous constructor.

- 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 dv=1, const bool shared=false)

  Construct an image of size (`dx,dy,dz,dv`*) with pixels of type* `T`*, from a raw pixel buffer* `data_buffer`.

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

  In-place version of the previous constructor.

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

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

Image Informations

- unsigned long **size** () const

  Return the total number of pixel values in an image.

- int **dimx** () const

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

- int **dimy** () const

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

- int **dimz** () const

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

- int **dimv** () const

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

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

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

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

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

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

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

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

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

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

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

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

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

- template<typename t> bool **is_sameXYZV** (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 **is_empty** () const

  Return `true` *if image is empty.*

- long **offset** (const int x=0, const int y=0, const int z=0, const int v=0) const

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

- T ∗ **ptr** (const unsigned int x=0, const unsigned int y=0, const unsigned int z=0, const unsigned int v=0)

  Return a pointer to the pixel value located at (`x,y,z,v`*).*

- **iterator begin** ()

  Return an iterator to the first image pixel.

- **iterator end** ()

  Return an iterator to the last image pixel.

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

  Fast access to pixel value for reading or writing.

- T & **at** (const unsigned int x, const unsigned int y=0, const unsigned int z=0, const unsigned int v=0)

  Return pixel value at a given position. Equivalent to operator().

- T & **operator[ ]** (const unsigned long off)

  Fast access to pixel value for reading or writing, using an offset to the image pixel.

- T & **back** ()

  Return a reference to the last image value.

- T & **front** ()

  Return a reference to the first image value.

- T **pix4d** (const int x, const int y, const int z, const int v, const T &out_val) const

  Read a pixel value with Dirichlet or Neumann boundary conditions.

- T **pix3d** (const int x, const int y, const int z, const int v, const T &out_val) const

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

- T **pix2d** (const int x, const int y, const int z, const int v, const T &out_val) const

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

- T **pix1d** (const int x, const int y, const int z, const int v, const T &out_val) const

  Read a pixel value with Dirichlet or Neumann boundary conditions for the first coordinate `x`*.*

- cimg::largest< T, float >::type **linear_pix4d** (const float fx, const float fy, const float fz, const float fv, const T &out_val) const

  Read a pixel value using linear interpolation.

- cimg::largest< T, float >::type **linear_pix3d** (const float fx, const float fy, const float fz, const int v, const T &out_val) const

  Read a pixel value using linear interpolation for the three first coordinates (`cx,cy,cz`*).*

- cimg::largest< T, float >::type **linear_pix2d** (const float fx, const float fy, const int z, const int v, const T &out_val) const

  Read a pixel value using linear interpolation for the two first coordinates (`cx,cy`*).*

- cimg::largest< T, float >::type **linear_pix1d** (const float fx, const int y, const int z, const int v, const T &out_val) const

  Read a pixel value using linear interpolation for the first coordinate `cx`*.*

- cimg::largest< T, float >::type **cubic_pix1d** (const float fx, const int y, const int z, const int v, const T &out_val) const

  Read a pixel value using cubic interpolation for the first coordinate `cx`*.*

- cimg::largest< T, float >::type **cubic_pix2d** (const float fx, const float fy, const int z, const int v, const T &out_val) const

  Read a pixel value using bicubic interpolation.

- const **CImg** & **print** (const char *title=0, const unsigned int print_flag=1) const

  Display informations about the image on the standard error output.

- const **CImg** & **print** (const unsigned int print_flag) const

  Display informations about the image on the standart output.

- static const char * **pixel_type** ()

  Return the type of the pixel values.

Arithmetic and Boolean Operators

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

  Assign a value to each image pixel of the instance image.

- **CImg** & **operator=** (const T *buf)

  Assign values of a C-array to the instance image.

- **CImg operator+** (const T &val) const

  Addition.

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

  Addition.

- **CImg** & **operator+=** (const T &val)

In-place addition.

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

  In-place addition.

- **CImg** & **operator++** ()

  In-place increment.

- **CImg operator-** (const T &val) const

  Substraction.

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

  Substraction.

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

  In-place substraction.

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

  In-place substraction.

- **CImg** & **operator–** ()

  In-place decrement.

- **CImg operator** ∗ (const double val) const

  Multiplication.

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

  Multiplication.

- template<typename t> **CImgl**< typename cimg::largest< T, t >::type > **operator** ∗ (const **CImgl**< t > &list) const

  Matrix multiplication with an image list.

- **CImg** & **operator** ∗**=** (const double val)

  In-place multiplication.

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

  In-place multiplication.

- **CImg operator/** (const double val) const

  Division.

- **CImg** & **operator/=** (const double val)

  In-place division.

- **CImg operator%** (const T &val) const

  Modulo.

- **CImg operator%** (const **CImg** &img) const
  Modulo.
- **CImg** & **operator%=** (const T &val)
  In-place modulo.
- **CImg** & **operator%=** (const **CImg** &img)
  In-place modulo.
- **CImg operator &** (const T &val) const
  Bitwise AND.
- **CImg operator &** (const **CImg** &img) const
  Bitwise AND.
- **CImg** & **operator &=** (const T &val)
  In-place bitwise AND.
- **CImg** & **operator &=** (const **CImg** &img)
  In-place bitwise AND.
- **CImg operator|** (const T &val) const
  Bitwise OR.
- **CImg operator|** (const **CImg** &img) const
  Bitwise OR.
- **CImg** & **operator|=** (const T &val)
  In-place bitwise OR.
- **CImg** & **operator|=** (const **CImg** &img)
  In-place bitwise OR.
- **CImg operator^** (const T &val) const
  Bitwise XOR.
- **CImg operator^** (const **CImg** &img) const
  Bitwise XOR.
- **CImg** & **operator^=** (const T &val)
  In-place bitwise XOR.
- **CImg** & **operator^=** (const **CImg** &img)
  In-place bitwise XOR.
- **CImg operator!** () const
  Boolean NOT.
- **CImg operator~** () const
  Bitwise NOT.

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

  Boolean equality.

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

  Boolean difference.

- **CImg operator+** (const T &val, const **CImg**< T > &img)

  Addition.

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

  Substraction.

- **CImg operator** ∗ (const double val, const **CImg** &img)

  Multiplication.

Usual Mathematics

- template<typename ts, typename td> **CImg** & **apply** (td(∗func)(ts))

  Apply a R->R function on all image value.

- template<typename ts, typename td> **CImg**< typename cimg::largest< T, td >::type > **get_apply** (td(∗func)(ts))

  Return an image where each pixel value is equal to func(x).

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

  In-place pointwise multiplication between `∗this` *and* `img`*.*

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

  Pointwise multiplication between `∗this` *and* `img`*.*

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

  Replace the image by the pointwise division between `∗this` *and* `img`*.*

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

  Return an image from a pointwise division between `∗this` *and* `img`*.*

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

  Replace the image by the pointwise max operator between `∗this` *and* `img`*.*

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

  Return the image corresponding to the max value for each pixel.

- **CImg** & **max** (const T &val)

  Replace the image by the pointwise max operator between `∗this` *and* `val`*.*

- **CImg get_max** (const T &val) const

  Return the image corresponding to the max value for each pixel.

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

  Replace the image by the pointwise min operator between `*this` *and* `img`.

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

  Return the image corresponding to the min value for each pixel.

- **CImg** & **min** (const T &val)

  Replace the image by the pointwise min operator between `*this` *and* `val`.

- **CImg get_min** (const T &val) const

  Return the image corresponding to the min value for each pixel.

- **CImg** & **sqrt** ()

  Replace each image pixel by its square root.

- **CImg**< typename cimg::largest< T, float >::type > **get_sqrt** () const

  Return the image of the square root of the pixel values.

- **CImg** & **log** ()

  Replace each image pixel by its log.

- **CImg**< typename cimg::largest< T, float >::type > **get_log** () const

  Return the image of the log of the pixel values.

- **CImg** & **log10** ()

  Replace each image pixel by its log10.

- **CImg**< typename cimg::largest< T, float >::type > **get_log10** () const

  Return the image of the log10 of the pixel values.

- **CImg** & **pow** (const double p)

  Replace each image pixel by its power by `p`.

- **CImg**< typename cimg::largest< T, float >::type > **get_pow** (const double p) const

  Return the image of the square root of the pixel values.

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

  *Return each image pixel (*this)(x,y,z,k) by its power by* `img(x,y,z,k)`.

- template<typename t> **CImg**< typename cimg::largest< T, float >::type > **get_pow** (const **CImg**< t > &img) const

  *Return each image pixel (*this)(x,y,z,k) by its power by* `img(x,y,z,k)`.

- **CImg** & **abs** ()

  Replace each pixel value by its absolute value.

- **CImg**< typename cimg::largest< T, float >::type > **get_abs** () const
  Return the image of the absolute value of the pixel values.
- **CImg** & **cos** ()
  Replace each image pixel by its cosinus.
- **CImg**< typename cimg::largest< T, float >::type > **get_cos** () const
  Return the image of the cosinus of the pixel values.
- **CImg** & **sin** ()
  Replace each image pixel by its sinus.
- **CImg**< typename cimg::largest< T, float >::type > **get_sin** () const
  Return the image of the sinus of the pixel values.
- **CImg** & **tan** ()
  Replace each image pixel by its tangent.
- **CImg**< typename cimg::largest< T, float >::type > **get_tan** () const
  Return the image of the tangent of the pixel values.
- template<typename t> double **MSE** (const **CImg**< t > &img) const
  Return the MSE (Mean-Squared Error) between two images.
- template<typename t> double **PSNR** (const **CImg**< t > &img, const double valmax=255.0) const
  Return the PSNR between two images.

Usual Image Transformations

- **CImg** & **fill** (const T &val)
  Fill an image by a value `val`.
- **CImg** & **fill** (const T &val0, const T &val1)
  Fill sequentially all pixel values with values val0 *and* val1 *respectively.*
- **CImg** & **fill** (const T &val0, const T &val1, const T &val2)
  Fill sequentially all pixel values with values val0 *and* val1 *and* val2.
- **CImg** & **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** & **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** & **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** & **fill** (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6)

  Fill sequentially all pixel values with values val0 *and* val1 *and* val2 *and* val3 *and* val4 *and* val5.

- **CImg** & **fill** (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6, const T &val7)

  Fill sequentially all pixel values with values val0 *and* val1 *and* val2 *and* val3 *and ... and* val7.

- **CImg** & **fill** (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6, const T &val7, const T &val8)

  Fill sequentially all pixel values with values val0 *and* val1 *and* val2 *and* val3 *and ... and* val8.

- **CImg** & **fill** (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6, const T &val7, const T &val8, const T &val9)

  Fill sequentially all pixel values with values val0 *and* val1 *and* val2 *and* val3 *and ... and* val9.

- **CImg** & **fill** (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6, const T &val7, const T &val8, const T &val9, const T &val10, const T &val11)

  Fill sequentially all pixel values with values val0 *and* val1 *and* val2 *and* val3 *and ... and* val11.

- **CImg** & **fill** (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6, const T &val7, const T &val8, const T &val9, const T &val10, const T &val11, const T &val12)

  Fill sequentially all pixel values with values val0 *and* val1 *and* val2 *and* val3 *and ... and* val11.

- **CImg** & **fill** (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6, const T &val7, const T &val8, const T &val9, const T &val10, const T &val11, const T &val12, const T &val13, const T &val14, const T &val15)

  Fill sequentially all pixel values with values val0 *and* val1 *and* val2 *and* val3 *and ... and* val15.

- **CImg** & **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 T &val16, const T &val17, const T &val18, const T &val19, const T &val20, const T &val21, const T &val22, const T &val23, const T &val24)

  Fill sequentially all pixel values with values val0 *and* val1 *and* val2 *and* val3 *and ... and* val24.

- **CImg** & **normalize** (const T &a, const T &b)

  Linear normalization of the pixel values between a *and* b.

- **CImg get_normalize** (const T &a, const T &b) const

  Return the image of normalized values.

- **CImg** & **cut** (const T &a, const T &b)

  Cut pixel values between a *and* b.

- **CImg get_cut** (const T &a, const T &b) const

  Return the image of cutted values.

- **CImg** & **quantize** (const unsigned int n=256)

Quantize pixel values into levels.

- **CImg get_quantize** (const unsigned int n=256) const

  Return a quantified image, with levels.

- **CImg** & **threshold** (const T &thres)

  Threshold the image.

- **CImg get_threshold** (const T &thres) const

  Return a thresholded image.

- **CImg get_rotate** (const float angle, const unsigned int cond=3) const

  Return a rotated image.

- **CImg** & **rotate** (const float angle, const unsigned int cond=3)

  Rotate the image.

- **CImg get_rotate** (const float angle, const float cx, const float cy, const float zoom=1, const unsigned int cond=3) const

  Return a rotated image around the point (`cx,cy`*).*

- **CImg** & **rotate** (const float angle, const float cx, const float cy, const float zoom=1, const unsigned int cond=3)

  Rotate the image around the point (`cx,cy`*).*

- **CImg get_resize** (const int pdx=-100, const int pdy=-100, const int pdz=-100, const int pdv=-100, const unsigned int interp=1) const

  Return a resized image.

- template<typename t> **CImg get_resize** (const **CImg**< t > &src, const unsigned int interp=1) const

  Return a resized image.

- **CImg get_resize** (const **CImgDisplay** &disp, const unsigned int interp=1) const

  Return a resized image.

- **CImg** & **resize** (const int pdx=-100, const int pdy=-100, const int pdz=-100, const int pdv=-100, const unsigned int interp=1)

  Resize the image.

- template<typename t> **CImg** & **resize** (const **CImg**< t > &src, const unsigned int interp=1)

  Resize the image.

- **CImg** & **resize** (const **CImgDisplay** &disp, const unsigned int interp=1)

  Resize the image.

- **CImg get_resize_halfXY** () const

  Return an half-resized image, using a special filter.

- **CImg** & **resize_halfXY** ()

  Half-resize the image, using a special filter.

- **CImg get_crop** (const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int v0, const unsigned int x1, const unsigned int y1, const unsigned int z1, const unsigned int v1, const bool border_condition=false) const

  Return a square region of the image, as a new image.

- **CImg get_crop** (const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int x1, const unsigned int y1, const unsigned int z1, const bool border_condition=false) const

  Return a square region of the image, as a new image.

- **CImg get_crop** (const unsigned int x0, const unsigned int y0, const unsigned int x1, const unsigned int y1, const bool border_condition=false) const

  Return a square region of the image, as a new image.

- **CImg get_crop** (const unsigned int x0, const unsigned int x1, const bool border_condition=false) const

  Return a square region of the image, as a new image.

- **CImg** & **crop** (const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int v0, const unsigned int x1, const unsigned int y1, const unsigned int z1, const unsigned int v1, const bool border_condition=false)

  Replace the image by a square region of the image.

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

  Replace the image by a square region of the image.

- **CImg** & **crop** (const unsigned int x0, const unsigned int y0, const unsigned int x1, const unsigned int y1, const bool border_condition=false)

  Replace the image by a square region of the image.

- **CImg** & **crop** (const unsigned int x0, const unsigned int x1, const bool border_condition=false)

  Replace the image by a square region of the image.

- **CImg get_shared_points** (const unsigned int x0, const unsigned int x1, const unsigned int y0=0, const unsigned int z0=0, const unsigned int v0=0)

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

- const **CImg get_shared_points** (const unsigned int x0, const unsigned int x1, const unsigned int y0=0, const unsigned int z0=0, const unsigned int v0=0) const

  Get a shared-memory image referencing a set of points of the instance image (const version).

- **CImg get_points** (const unsigned int x0, const unsigned int x1, const unsigned int y0=0, const unsigned int z0=0, const unsigned int v0=0) const

  Get a copy of a set of points of the instance image.

- **CImg get_shared_lines** (const unsigned int y0, const unsigned int y1, const unsigned int z0=0, const unsigned int v0=0)

  Return a shared-memory image referencing a set of lines of the instance image.

- const **CImg get_shared_lines** (const unsigned int y0, const unsigned int y1, const unsigned int z0=0, const unsigned int v0=0) const

  Return a shared-memory image referencing a set of lines of the instance image (const version).

- **CImg get_lines** (const unsigned int y0, const unsigned int y1, const unsigned int z0=0, const unsigned int v0=0) const

  Get a copy of a set of lines of the instance image.

- **CImg** & **lines** (const unsigned int y0, const unsigned int y1, const unsigned int z0=0, const unsigned int v0=0) const

  Replace the instance image by a set of lines of the instance image.

- **CImg get_shared_line** (const unsigned int y0, const unsigned int z0=0, const unsigned int v0=0)

  Return a shared-memory image referencing one particular line (y0,z0,v0) of the instance image.

- const **CImg get_shared_line** (const unsigned int y0, const unsigned int z0=0, const unsigned int v0=0) const

  Return a shared-memory image referencing one particular line (y0,z0,v0) of the instance image (const version).

- **CImg get_line** (const unsigned int y0, const unsigned int z0=0, const unsigned int v0=0) const

  Get a copy of a line of the instance image.

- **CImg** & **line** (const unsigned int y0, const unsigned int z0=0, const unsigned int v0=0)

  Replace the instance image by one of its line.

- **CImg get_shared_planes** (const unsigned int z0, const unsigned int z1, const unsigned int v0=0)

  Return a shared memory image referencing a set of planes (z0->z1,v0) of the instance image.

- const **CImg get_shared_planes** (const unsigned int z0, const unsigned int z1, const unsigned int v0=0) const

  Return a shared-memory image referencing a set of planes (z0->z1,v0) of the instance image (const version).

- **CImg get_planes** (const unsigned int z0, const unsigned int z1, const unsigned int v0=0) const

  Return a copy of a set of planes of the instance image.

- **CImg** & **planes** (const unsigned int z0, const unsigned int z1, const unsigned int v0=0)

  Replace the instance image by a set of planes of the instance image.

- **CImg get_shared_plane** (const unsigned int z0, const unsigned int v0=0)

  Return a shared-memory image referencing one plane (z0,v0) of the instance image.

- const **CImg get_shared_plane** (const unsigned int z0, const unsigned int v0=0) const

  Return a shared-memory image referencing one plane (z0,v0) of the instance image (const version).

- **CImg get_plane** (const unsigned int z0, const unsigned int v0=0) const

  Return a copy of a plane of the instance image.

- **CImg** & **plane** (const unsigned int z0, const unsigned int v0=0)

Replace the instance image by one plane of the instance image.

- **CImg get_shared_channels** (const unsigned int v0, const unsigned int v1)

  Return a shared-memory image referencing a set of channels (v0->v1) of the instance image.

- const **CImg get_shared_channels** (const unsigned int v0, const unsigned int v1) const

  Return a shared-memory image referencing a set of channels (v0->v1) of the instance image (const version).

- **CImg get_channels** (const unsigned int v0, const unsigned int v1) const

  Return a copy of a set of channels of the instance image.

- **CImg** & **channels** (const unsigned int v0, const unsigned int v1)

  Replace the instance image by a set of channels of the instance image.

- **CImg get_shared_channel** (const unsigned int v0)

  Return a shared-memory image referencing one channel v0 of the instance image.

- const **CImg get_shared_channel** (const unsigned int v0) const

  Return a shared-memory image referencing one channel v0 of the instance image (const version).

- **CImg get_channel** (const unsigned int v0) const

  Return a copy of a channel of the instance image.

- **CImg** & **channel** (const unsigned int v0)

  Replace the instance image by one of its channel.

- **CImg get_shared** ()

  Return a shared version of the instance image.

- const **CImg get_shared** () const

  Return a shared version of the instance image (const version).

- **CImg get_slice** (const unsigned int z0=0) const

  Get the z-slice z *of* ∗*this, as a new image.*

- **CImg** & **slice** (const unsigned int z0)

  Replace the image by one of its slice.

- **CImg** & **mirror** (const char axe='x')

  Mirror an image along the specified axis.

- **CImg get_mirror** (const char axe='x')

  Get a mirrored version of the image, along the specified axis.

- **CImg** & **scroll** (const int scrollx, const int scrolly=0, const int scrollz=0, const int scrollv=0, const int border_condition=0)

  Scroll the image.

- **CImg get_scroll** (const int scrollx, const int scrolly=0, const int scrollz=0, const int scrollv=0, const int border_condition=0) const

Return a scrolled image.

- **CImg get_projections2d** (const unsigned int px0, const unsigned int py0, const unsigned int pz0) const

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

- **CImg**< float > **get_histogram** (const unsigned int nblevels=256, const T val_min=(T) 0, const T val_max=(T) 0) const

  Return the image histogram.

- **CImg** & **equalize_histogram** (const unsigned int nblevels=256, const T val_min=(T) 0, const T val_max=(T) 0)

  Equalize the image histogram.

- **CImg get_equalize_histogram** (const unsigned int nblevels=256, const T val_min=(T) 0, const T val_max=(T) 0) const

  Return the histogram-equalized version of the current image.

- **CImg**< typename cimg::largest< T, float >::type > **get_norm_pointwise** (int norm_type=2) const

  Return the scalar image of vector norms.

- **CImg** & **norm_pointwise** ()

  Replace each pixel value with its vector norm.

- **CImg**< typename cimg::largest< T, float >::type > **get_orientation_pointwise** () const

  Return the image of normalized vectors.

- **CImg** & **orientation_pointwise** ()

  Replace each pixel value by its normalized vector.

- **CImgl**< T > **get_split** (const char axe='x', const unsigned int nb=0) const

  Split image into a list CImgl<>.

- **CImg get_append** (const **CImg**< T > &img, const char axis='x', const char align='c') const

  Append an image to another one.

- **CImg** & **append** (const **CImg**< T > &img, const char axis='x', const char align='c')

  Append an image to another one (in-place version).

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

  Append an image to another one (in-place operator<< version).

- **CImgl**< typename cimg::largest< T, float >::type > **get_gradientXY** (const int scheme=0) const

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

- **CImgl**< typename cimg::largest< T, float >::type > **get_gradientXYZ** (const int scheme=0) const

  Return a list of images, corresponding to the XYZ-gradients of an image.

Meshes and Triangulations

- template<typename tp, typename tf> const **CImg** & **marching_squares** (const float isovalue, **CImgl**< tp > &points, **CImgl**< tf > &primitives) const

  Get a vectorization of an implicit function defined by the instance image.

- template<typename tp, typename tf> const **CImg** & **marching_squares** (const float isovalue, const float resx, const float resy, **CImgl**< tp > &points, **CImgl**< tf > &primitives) const

  Get a vectorization of an implicit function defined by the instance image.

- template<typename tp, typename tf> const **CImg** & **marching_cubes** (const float isovalue, **CImgl**< tp > &points, **CImgl**< tf > &primitives, const bool invert_faces=false) const

  Get a triangulation of an implicit function defined by the instance image.

- template<typename tp, typename tf> const **CImg** & **marching_cubes** (const float isovalue, const float resx, const float resy, const float resz, **CImgl**< tp > &points, **CImgl**< tf > &primitives, const bool invert_faces=false) const

  Get a triangulation of an implicit function defined by the instance image.

Color conversions

- template<typename t> **CImg**< t > **get_RGBtoLUT** (const **CImg**< t > &palette, const bool dithering=true, const bool indexing=false) const

  Convert color pixels from (R,G,B) to match a specified palette.

- **CImg**< T > **get_RGBtoLUT** (const bool dithering=true, const bool indexing=false) const

  Convert color pixels from (R,G,B) to match the default 256 colors palette.

- **CImg** & **RGBtoLUT** (const **CImg**< T > &palette, const bool dithering=true, const bool indexing=false)

  Convert color pixels from (R,G,B) to match the specified color palette.

- **CImg** & **RGBtoLUT** (const bool dithering=true, const bool indexing=false)

  Convert color pixels from (R,G,B) to match the specified color palette.

- template<typename t> **CImg**< t > **get_LUTtoRGB** (const **CImg**< t > &palette) const

  Convert an indexed image to a (R,G,B) image using the specified color palette.

- **CImg**< T > **get_LUTtoRGB** () const

  Convert an indexed image (with the default palette) to a (R,G,B) image.

- **CImg** & **LUTtoRGB** (const **CImg**< T > &palette)

  In-place version of **get_LUTtoRGB**()(p. 36).

- **CImg** & **LUTtoRGB** ()

  In-place version of get_LUTroRGB().

- **CImg** & **RGBtoHSV** ()

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

- **CImg** & **HSVtoRGB** ()

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

- **CImg** & **RGBtoYCbCr** ()

  Convert color pixels from (R,G,B) to (Y,Cb,Cr)_8 (Thanks to Chen Wang).

- **CImg** & **YCbCrtoRGB** ()

  Convert color pixels from (Y,Cb,Cr)_8 to (R,G,B).

- **CImg** & **RGBtoYUV** ()

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

- **CImg** & **YUVtoRGB** ()

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

- **CImg** & **RGBtoXYZ** ()

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

- **CImg** & **XYZtoRGB** ()

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

- **CImg** & **LabtoXYZ** ()

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

- **CImg** & **XYZtoxyY** ()

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

- **CImg** & **xyYtoXYZ** ()

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

- **CImg** & **RGBtoLab** ()

  In-place version of **get_RGBtoLab()**(p. 38).

- **CImg** & **LabtoRGB** ()

  In-place version of get_LabtoRGb().

- **CImg** & **RGBtoxyY** ()

  In-place version of **get_RGBtoxyY()**(p. 38).

- **CImg** & **xyYtoRGB** ()

  In-place version of **get_xyYtoRGB()**(p. 38).

- **CImg get_RGBtoHSV** () const

  Convert a (R,G,B) image to a (H,S,V) one.

- **CImg get_HSVtoRGB** () const

  Convert a (H,S,V) image to a (R,G,B) one.

- **CImg get_RGBtoYCbCr** () const

Convert a (R,G,B) image to a (Y,Cb,Cr) one.

- **CImg get_YCbCrtoRGB** () const
  Convert a (Y,Cb,Cr) image to a (R,G,B) one.
- **CImg**< typename cimg::largest< T, float >::type > **get_RGBtoYUV** () const
  Convert a (R,G,B) image into a (Y,U,V) one.
- **CImg get_YUVtoRGB** () const
  Convert a (Y,U,V) image into a (R,G,B) one.
- **CImg**< typename cimg::largest< T, float >::type > **get_RGBtoXYZ** () const
  Convert a (R,G,B) image to a (X,Y,Z) one.
- **CImg get_XYZtoRGB** () const
  Convert a (X,Y,Z) image to a (R,G,B) one.
- **CImg get_XYZtoLab** () const
  Convert a (X,Y,Z) image to a (L,a,b) one.
- **CImg get_LabtoXYZ** () const
  Convert a (L,a,b) image to a (X,Y,Z) one.
- **CImg get_XYZtoxyY** () const
  Convert a (X,Y,Z) image to a (x,y,Y) one.
- **CImg get_xyYtoXYZ** () const
  Convert a (x,y,Y) image to a (X,Y,Z) one.
- **CImg get_RGBtoLab** () const
  Convert a (R,G,B) image to a (L,a,b) one.
- **CImg get_LabtoRGB** () const
  Convert a (L,a,b) image to a (R,G,B) one.
- **CImg get_RGBtoxyY** () const
  Convert a (R,G,B) image to a (x,y,Y) one.
- **CImg get_xyYtoRGB** () const
  Convert a (x,y,Y) image to a (R,G,B) one.
- static **CImg**< T > **get_default_LUT8** ()
  Return the default 256 colors palette.

Drawing

- **CImg** & **draw_point** (const int x0, const int y0, const int z0, const T *const color, const float opacity=1)
  Draw a colored point in the instance image, at coordinates (`x0,y0,z0`*).*

- **CImg** & **draw_point** (const int x0, const int y0, const T ∗const color, const float opacity=1)

  *Draw a colored point in the instance image, at coordinates (*x0,y0*).*

- **CImg** & **draw_line** (const int x0, const int y0, const int x1, const int y1, const T ∗const color, const unsigned int pattern=∼0L, const float opacity=1)

  *Draw a 2D colored line in the instance image, at coordinates (*x0,y0*)-(*x1,y1*).*

- **CImg** & **draw_line** (const int x0, const int y0, const int z0, const int x1, const int y1, const int z1, const T ∗const color, const unsigned int pattern=∼0L, const float opacity=1)

  *Draw a 3D colored line in the instance image, at coordinates (*x0,y0,z0*)-(*x1,y1,z1*).*

- template<typename t> **CImg** & **draw_line** (const int x0, const int y0, const int x1, const int y1, const **CImg**< t > &texture, const int tx0, const int ty0, const int tx1, const int ty1, const float opacity=1)

  *Draw a 2D textured line in the instance image, at coordinates (*x0,y0*)-(*x1,y1*).*

- **CImg** & **draw_arrow** (const int x0, const int y0, const int x1, const int y1, const T ∗const color, const float angle=30, const float length=-10, const unsigned int pattern=∼0L, const float opacity=1)

  *Draw a 2D colored arrow in the instance image, at coordinates (*x0,y0*)->(*x1,y1*).*

- template<typename t> **CImg** & **draw_image** (const **CImg**< t > &sprite, const int x0=0, const int y0=0, const int z0=0, const int v0=0, const float opacity=1)

  *Draw a sprite image in the instance image, at coordinates (*x0,y0,z0,v0*).*

- template<typename ti, typename tm> **CImg** & **draw_image** (const **CImg**< ti > &sprite, const **CImg**< tm > &mask, const int x0=0, const int y0=0, const int z0=0, const int v0=0, const tm mask_valmax=1, const float opacity=1)

  *Draw a masked sprite image in the instance image, at coordinates (*x0,y0,z0,v0*).*

- **CImg** & **draw_rectangle** (const int x0, const int y0, const int z0, const int v0, const int x1, const int y1, const int z1, const int v1, const T &val, const float opacity=1.0f)

  *Draw a 4D filled rectangle in the instance image, at coordinates (*x0,y0,z0,v0*)-(*x1,y1,z1,v1*).*

- **CImg** & **draw_rectangle** (const int x0, const int y0, const int z0, const int x1, const int y1, const int z1, const T ∗const color, const float opacity=1)

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

- **CImg** & **draw_rectangle** (const int x0, const int y0, const int x1, const int y1, const T ∗const color, const float opacity=1)

  *Draw a 2D filled colored rectangle in the instance image, at coordinates (*x0,y0*)-(*x1,y1*).*

- **CImg** & **draw_triangle** (const int x0, const int y0, const int x1, const int y1, const int x2, const int y2, const T ∗const color, const float opacity=1, const float brightness=1)

  *Draw a 2D filled colored triangle in the instance image, at coordinates (*x0,y0*)-(*x1,y1*)-(*x2,y2*).*

- **CImg** & **draw_triangle** (const int x0, const int y0, const int x1, const int y1, const int x2, const int y2, const T ∗const color, const float c0, const float c1, const float c2, const float opacity=1)

  *Draw a 2D Gouraud-filled triangle in the instance image, at coordinates (*x0,y0*)-(*x1,y1*)-(*x2,y2*).*

- template<typename t> **CImg** & **draw_triangle** (const int x0, const int y0, const int x1, const int y1, const int x2, const int y2, const T *const color, const **CImg**< t > &light, const int lx0, const int ly0, const int lx1, const int ly1, const int lx2, const int ly2, const float opacity=1.0f)

  *Draw a 2D phong-shaded triangle in the instance image, at coordinates (*x0,y0*)-(*x1,y1*)-(*x2,y2*).*

- template<typename t> **CImg** & **draw_triangle** (const int x0, const int y0, const int x1, const int y1, const int x2, const int y2, const **CImg**< t > &texture, const int tx0, const int ty0, const int tx1, const int ty1, const int tx2, const int ty2, const float opacity=1.0f, const float brightness=1.0f)

  *Draw a 2D textured triangle in the instance image, at coordinates (*x0,y0*)-(*x1,y1*)-(*x2,y2*).*

- template<typename t> **CImg** & **draw_triangle** (const int x0, const int y0, const int x1, const int y1, const int x2, const int y2, const **CImg**< t > &texture, const int tx0, const int ty0, const int tx1, const int ty1, const int tx2, const int ty2, const float c0, const float c1, const float c2, const float opacity=1)

  *Draw a 2D textured triangle with Gouraud-Shading in the instance image, at coordinates (*x0,y0*)-(*x1,y1*)-(*x2,y2*).*

- template<typename t, typename tl> **CImg** & **draw_triangle** (const int x0, const int y0, const int x1, const int y1, const int x2, const int y2, const **CImg**< t > &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=1.0f)

  *Draw a phong-shaded 2D textured triangle in the instance image, at coordinates (*x0,y0*)-(*x1,y1*)-(*x2,y2*).*

- **CImg** & **draw_ellipse** (const int x0, const int y0, const float r1, const float r2, const float ru, const float rv, const T *const color, const unsigned int pattern=0L, const float opacity=1)

  Draw an ellipse on the instance image.

- template<typename t> **CImg** & **draw_ellipse** (const int x0, const int y0, const **CImg**< t > &tensor, const T *color, const unsigned int pattern=0L, const float opacity=1)

  Draw an ellipse on the instance image.

- **CImg** & **draw_circle** (const int x0, const int y0, float r, const T *const color, const unsigned int pattern=0L, const float opacity=1)

  Draw a circle on the instance image.

- template<typename t> **CImg** & **draw_text** (const char *const text, const int x0, const int y0, const T *const fgcolor, const T *const bgcolor, const **CImgl**< t > &font, const float opacity=1)

  Draw a text into the instance image.

- **CImg** & **draw_text** (const char *const text, const int x0, const int y0, const T *const fgcolor, const T *const bgcolor=0, const unsigned int font_size=11, const float opacity=1.0f)

  Draw a text into the instance image.

- **CImg** & **draw_text** (const int x0, const int y0, const T *const fgcolor, const T *const bgcolor, const unsigned int font_size, const float opacity, const char *format,...)

  Draw a text into the instance image.

- template<typename t> **CImg** & **draw_quiver** (const **CImg**< t > &flow, const T *const color, const unsigned int sampling=25, const float factor=-20, const int quiver_type=0, const float opacity=1)

  Draw a vector field in the instance image.

- template<typename t1, typename t2> **CImg** & **draw_quiver** (const **CImg**< t1 > &flow, const **CImg**< t2 > &color, const unsigned int sampling=25, const float factor=-20, const int quiver_type=0, const float opacity=1)

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

- template<typename t> **CImg** & **draw_graph** (const **CImg**< t > &**data**, const T *const color, const unsigned int gtype=0, const double ymin=0, const double ymax=0, const float opacity=1)

  Draw a 1D graph on the instance image.

- template<typename t> **CImg** & **draw_axe** (const **CImg**< t > &xvalues, const int y, const T *const color, const int precision=-1, const float opacity=1.0f)

  Draw a labelled horizontal axis on the instance image.

- template<typename t> **CImg** & **draw_fill** (const int x, const int y, const int z, const T *const color, **CImg**< t > &region, const float sigma=0, const float opacity=1)

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

- **CImg** & **draw_fill** (const int x, const int y, const int z, const T *const color, const float sigma=0, const float opacity=1)

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

- **CImg** & **draw_fill** (const int x, const int y, const T *const color, const float sigma=0, const float opacity=1)

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

- **CImg** & **draw_plasma** (const int x0, const int y0, const int x1, const int y1, const double alpha=1.0, const double beta=1.0, const float opacity=1)

  Draw a plasma square in the instance image.

- **CImg** & **draw_plasma** (const double alpha=1.0, const double beta=1.0, const float opacity=1)

  Draw a plasma in the instance image.

- **CImg** & **draw_gaussian** (const float xc, const double sigma, const T *const color, const float opacity=1)

  Draw a 1D gaussian function in the instance image.

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

  Draw an anisotropic 2D gaussian function in the instance image.

- **CImg** & **draw_gaussian** (const float xc, const float yc, const float sigma, const T *const color, const float opacity=1)

  Draw an isotropic 2D gaussian function in the instance image.

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

  Draw an anisotropic 3D gaussian function in the instance image.

- **CImg** & **draw_gaussian** (const float xc, const float yc, const float zc, const double sigma, const T *const color, const float opacity=1)

  Draw an isotropic 3D gaussian function in the instance image.

- template<typename tp, typename tf, typename to> **CImg** & **draw_object3d** (const float X, const float Y, const float Z, const **CImg**< tp > &points, const **CImgl**< tf > &primitives, const **CImgl**< T > &colors, const **CImgl**< 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=-5000, const float ambiant_light=0.05f)

  Draw a 3D object in the instance image.

- template<typename tp, typename tf, typename to> **CImg** & **draw_object3d** (const float X, const float Y, const float Z, const **CImgl**< tp > &points, const **CImgl**< tf > &primitives, const **CImgl**< T > &colors, const **CImgl**< 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=-5000, const float ambiant_light=0.05f)

  Draw a 3D object in the instance image.

- template<typename tp, typename tf, typename to> **CImg** & **draw_object3d** (const float X, const float Y, const float Z, const **CImg**< tp > &points, const **CImgl**< tf > &primitives, const **CImgl**< T > &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=-5000, const float ambiant_light=0.05f)

  Draw a 3D object in the instance image.

- template<typename tp, typename tf, typename to> **CImg** & **draw_object3d** (const float X, const float Y, const float Z, const **CImgl**< tp > &points, const **CImgl**< tf > &primitives, const **CImgl**< T > &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=-5000, const float ambiant_light=0.05f)

  Draw a 3D object in the instance image.

- template<typename tp, typename tf> **CImg** & **draw_object3d** (const float X, const float Y, const float Z, const tp &points, const **CImgl**< tf > &primitives, const **CImgl**< T > &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=-5000, const float ambiant_light=0.05f, const float opacity=1.0f)

  Draw a 3D object in the instance image.

Image Filtering

- template<typename t> **CImg**< typename cimg::largest< T, t >::type > **get_correlate** (const **CImg**< t > &mask, const unsigned int cond=1, const bool weighted_correl=false) const

  Return the correlation of the image by a mask.

- template<typename t> **CImg** & **correlate** (const **CImg**< t > &mask, const unsigned int cond=1, const bool weighted_correl=false)

  Correlate the image by a mask.

- template<typename t> **CImg**< typename cimg::largest< T, t >::type > **get_convolve** (const **CImg**< t > &mask, const unsigned int cond=1, const bool weighted_convol=false) const

  Return the convolution of the image by a mask.

- template<typename t> **CImg** & **convolve** (const **CImg**< t > &mask, const unsigned int cond=1, const bool weighted_convol=false)

  Convolve the image by a mask.

- template<typename t> **CImg**< typename cimg::largest< T, t >::type > **get_erode** (const **CImg**< t > &mask, const unsigned int cond=1, const bool weighted_erosion=false) const

  Return the erosion of the image by a structuring element.

- template<typename t> **CImg** & **erode** (const **CImg**< t > &mask, const unsigned int cond=1, const bool weighted_erosion=false)

  Erode the image by a structuring element.

- **CImg get_erode** (const unsigned int n=1, const unsigned int cond=1) const

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

- **CImg** & **erode** (const unsigned int n=1, const unsigned int cond=1)

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

- template<typename t> **CImg**< typename cimg::largest< T, t >::type > **get_dilate** (const **CImg**< t > &mask, const unsigned int cond=1, const bool weighted_dilatation=false) const

  Return the dilatation of the image by a structuring element.

- template<typename t> **CImg** & **dilate** (const **CImg**< t > &mask, const unsigned int cond=1, const bool weighted_dilatation=false)

  Dilate the image by a structuring element.

- **CImg get_dilate** (const unsigned int n=1, const unsigned int cond=1) const

  Dilate the image by a square structuring element of size n.

- **CImg** & **dilate** (const unsigned int n=1, const unsigned int cond=1)

  Dilate the image by a square structuring element of size n.

- **CImg** & **noise** (const double sigma=-20, const unsigned int ntype=0)

  Add noise to the image.

- **CImg get_noise** (const double sigma=-20, const unsigned int ntype=0) const

  Return a noisy image.

- **CImg** & **deriche** (const float sigma=1, const int order=0, const char axe='x', const unsigned int cond=1)

  Apply a deriche filter on the image.

- **CImg get_deriche** (const float sigma=1, const int order=0, const char axe='x', const unsigned int cond=1) const

  Return the result of the Deriche filter.

- **CImg** & **blur** (const float sigmax, const float sigmay, const float sigmaz, const unsigned int cond=1)

  Blur the image with a Deriche filter (anisotropically).

- **CImg** & **blur** (const float sigma, const unsigned int cond=1)

  Blur the image with a Canny-Deriche filter.

- **CImg get_blur** (const float sigmax, const float sigmay, const float sigmaz, const unsigned int cond=1) const

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

- **CImg get_blur** (const float sigma, const unsigned int cond=1) const

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

- template<typename t> **CImg** & **blur_anisotropic** (const **CImg**< t > &G, const float amplitude=30.0f, const float dl=0.8f, const float da=30.0f, const float gauss_prec=2.0f, const unsigned int interpolation=0, const bool fast_approx=true)

  Blur an image following a field of diffusion tensors.

- template<typename t> **CImg get_blur_anisotropic** (const **CImg**< t > &G, const float amplitude=30.0f, const float dl=0.8f, const float da=30.0f, const float gauss_prec=2.0f, const unsigned int interpolation=0, const bool fast_approx=true) const

  Get a blurred version of an image following a field of diffusion tensors.

- **CImg** & **blur_anisotropic** (const float amplitude, const float sharpness=0.8f, const float anisotropy=0.3f, const float alpha=1.2f, const float sigma=0.8f, const float dl=0.8f, const float da=30.0f, const float gauss_prec=2.0f, const unsigned int interpolation=0, const bool fast_approx=true)

  Blur an image following a field of diffusion tensors.

- **CImg get_blur_anisotropic** (const float amplitude, const float sharpness=0.8f, const float anisotropy=0.3f, const float alpha=1.2f, const float sigma=0.8f, const float dl=0.8f, const float da=30.0f, const float gauss_prec=2.0f, const unsigned int interpolation=0, const bool fast_approx=true) const

  Blur an image in an anisotropic way.

- **CImgl**< typename cimg::largest< T, float >::type > **get_FFT** (const char axe, const bool inverse=false) const

  Return the Fast Fourier Transform of an image (along a specified axis).

- **CImgl**< typename cimg::largest< T, float >::type > **get_FFT** (const bool inverse=false) const

  Return the Fast Fourier Transform on an image.

- **CImg get_blur_median** (const unsigned int n=3)

  Apply a median filter.

- **CImg** & **blur_median** (const unsigned int n=3)

  Apply a median filter.

Matrix and Vectors

- **CImg** & **unroll** (const char axe='x')

  Unroll all images values into specified axis.

- **CImg get_diagonal** () const

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

- **CImg** & **diagonal** ()

  Replace a vector by a diagonal matrix containing the original vector coefficients.

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

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

- **CImg get_vector_at** (const unsigned int x=0, 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 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 get_tensor_at** (const unsigned int x=0, 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.*

- **CImg** & **set_vector_at** (const **CImg** &vec, const unsigned int x=0, 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.*

- **CImg** & **set_matrix_at** (const **CImg** &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.*

- **CImg** & **set_tensor_at** (const **CImg** &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 get_transpose** () const

  Return the transpose version of the current matrix.

- **CImg** & **transpose** ()

  Replace the current matrix by its transpose.

- **CImg** & **inverse** (const bool use_LU=true)

  Inverse the current matrix.

- **CImg**< typename cimg::largest< T, float >::type > **get_inverse** (const bool use_LU=true) const

  Return the inverse of the current matrix.

- **CImg**< typename cimg::largest< T, float >::type > **get_pseudoinverse** () const

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

- **CImg** & **pseudoinverse** ()

Replace the matrix by its pseudo-inverse.

- double **trace** () const

  Return the trace of the current matrix.

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

  Return the kth smallest element of the image.

- const T **median** () const

  Return the median of the image.

- double **dot** (const **CImg** &img) const

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

- **CImg** & **cross** (const **CImg** &img)

  Return the cross product between two 3d vectors.

- **CImg get_cross** (const **CImg** &img) const

  Return the cross product between two 3d vectors.

- double **det** () const

  Return the determinant of the current matrix.

- double **norm** (const int ntype=2) const

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

- double **sum** () const

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

- template<typename t> const **CImg** & **SVD** (**CImg**< t > &U, **CImg**< t > &S, **CImg**< t > &V, const bool sorting=true, const unsigned int max_iter=40) const

  Compute the SVD of a general matrix.

- template<typename t> const **CImg** & **SVD** (**CImgl**< t > &USV) const

  Compute the SVD of a general matrix.

- **CImgl**< typename cimg::largest< T, float >::type > **get_SVD** (const bool sorting=true) const

  Compute the SVD of a general matrix.

- **CImg** & **solve** (const **CImg** &A)

  Solve a linear system AX=B where B=∗this. (in-place version).

- **CImg**< typename cimg::largest< T, float >::type > **get_solve** (const **CImg** &A) const

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

- template<typename t> const **CImg**< T > & **eigen** (**CImg**< t > &val, **CImg**< t > &vec) const

  Compute the eigenvalues and eigenvectors of a matrix.

- **CImgl**< typename cimg::largest< T, float >::type > **get_eigen** () const

  Return the eigenvalues and eigenvectors of a matrix.

- template<typename t> const **CImg**< T > & **eigen** (**CImgl**< t > &eig) const

  Compute the eigenvalues and eigenvectors of a matrix.

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

  Compute the eigenvalues and eigenvectors of a symmetric matrix.

- **CImgl**< typename cimg::largest< T, float >::type > **get_symmetric_eigen** () const

  Compute the eigenvalues and eigenvectors of a symmetric matrix.

- template<typename t> const **CImg**< T > & **symmetric_eigen** (**CImgl**< t > &eig) const

  Compute the eigenvalues and eigenvectors of a symmetric matrix.

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

  Sort values of a vector and get permutations.

- **CImg**< T > & **sort** (const bool increasing=true)

  Sort values of a vector.

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

  Get a sorted version a of vector, with permutations.

- **CImg**< T > **get_sort** (const bool increasing=true)

  Get a sorted version of a vector.

- template<typename t> **CImg**< T > **get_permute** (const **CImg**< t > &permutation) const

  Get a permutation of the pixels.

- template<typename t> **CImg**< T > & **permute** (const **CImg**< t > &permutation)

  In-place version of the previous function.

- static **CImg vector** (const T &a1)

  Return a vector with specified coefficients.

- static **CImg vector** (const T &a1, const T &a2)

  Return a vector with specified coefficients.

- static **CImg vector** (const T &a1, const T &a2, const T &a3)

  Return a vector with specified coefficients.

- static **CImg vector** (const T &a1, const T &a2, const T &a3, const T &a4)

  Return a vector with specified coefficients.

- static **CImg vector** (const T &a1, const T &a2, const T &a3, const T &a4, const T &a5)

  Return a vector with specified coefficients.

- static **CImg vector** (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 vector** (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 vector** (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 vector** (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 vector** (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 vector** (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 vector** (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 vector** (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 matrix** (const T &a1)

  Return a 1x1 square matrix with specified coefficients.

- static **CImg matrix** (const T &a1, const T &a2, const T &a3, const T &a4)

  Return a 2x2 square matrix with specified coefficients.

- static **CImg matrix** (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 3x3 square matrix with specified coefficients.

- static **CImg matrix** (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)

  Return a 4x4 square matrix with specified coefficients.

- static **CImg matrix** (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, const T &a25)

Return a 5x5 square matrix with specified coefficients.

- static **CImg tensor** (const T &a1)

  Return a 1x1 symmetric matrix with specified coefficients.

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

  Return a 2x2 symmetric matrix tensor with specified coefficients.

- static **CImg 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 diagonal** (const T &a1)

  Return a 1x1 diagonal matrix with specified coefficients.

- static **CImg diagonal** (const T &a1, const T &a2)

  Return a 2x2 diagonal matrix with specified coefficients.

- static **CImg diagonal** (const T &a1, const T &a2, const T &a3)

  Return a 3x3 diagonal matrix with specified coefficients.

- static **CImg diagonal** (const T &a1, const T &a2, const T &a3, const T &a4)

  Return a 4x4 diagonal matrix with specified coefficients.

- static **CImg diagonal** (const T &a1, const T &a2, const T &a3, const T &a4, const T &a5)

  Return a 5x5 diagonal matrix with specified coefficients.

- static **CImg identity_matrix** (const unsigned int N)

  Return a NxN identity matrix.

- static **CImg 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.

Display

- const **CImg** & **display** (**CImgDisplay** &disp) const

  Display an image into a **CImgDisplay**(p. 112) *window.*

- const **CImg** & **display** (const char ∗title, const int min_size=128, const int max_size=1024) const

  If negative, they corresponds to a percentage of the original image size.

- const **CImg** & **display** (const int min_size=128, const int max_size=1024) const

  Display an image in a window, with a default title. See also.

- const **CImg** & **feature_selection** (int ∗const selection, const int feature_type, **CImgDisplay** &disp, unsigned int ∗const XYZ=0, const unsigned char ∗const color=0) const

  High-level interface to select features from images.

- const **CImg** & **feature_selection** (int *const selection, const int feature_type, unsigned int *const XYZ=0, const unsigned char *const color=0) const

  High-level interface to select features in images.

- template<typename tp, typename tf, typename to> const **CImg** & **display_object3d** (const **CImg**< tp > &points, const **CImgl**< tf > &primitives, const **CImgl**< T > &colors, const **CImgl**< to > &opacities, **CImgDisplay** &disp, const bool centering=true, const int render_static=4, const int render_motion=1, const bool double_sided=false, const float focale=500.0f, const float ambiant_light=0.05f, 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 to> const **CImg** & **display_object3d** (const **CImgl**< tp > &points, const **CImgl**< tf > &primitives, const **CImgl**< T > &colors, const **CImgl**< to > &opacities, **CImgDisplay** &disp, const bool centering=true, const int render_static=4, const int render_motion=1, const bool double_sided=false, const float focale=500.0f, const float ambiant_light=0.05f, 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 to> const **CImg** & **display_object3d** (const **CImgl**< tp > &points, const **CImgl**< tf > &primitives, const **CImgl**< T > &colors, const **CImg**< to > &opacities, **CImgDisplay** &disp, const bool centering=true, const int render_static=4, const int render_motion=1, const bool double_sided=false, const float focale=500.0f, const float ambiant_light=0.05f, 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 to> const **CImg** & **display_object3d** (const **CImgl**< tp > &points, const **CImgl**< tf > &primitives, const **CImgl**< T > &colors, const **CImg**< to > &opacities, **CImgDisplay** &disp, const bool centering=true, const int render_static=4, const int render_motion=1, const bool double_sided=false, const float focale=500.0f, const float ambiant_light=0.05f, 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 to> const **CImg** & **display_object3d** (const tp &points, const **CImgl**< tf > &primitives, const **CImgl**< T > &colors, const to &opacities, const bool centering=true, const int render_static=4, const int render_motion=1, const bool double_sided=false, const float focale=500.0f, const float ambiant_light=0.05f, 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** & **display_object3d** (const tp &points, const **CImgl**< tf > &primitives, const **CImgl**< T > &colors, const bool centering=true, const int render_static=4, const int render_motion=1, const bool double_sided=false, const float focale=500.0f, const float ambiant_light=0.05f, const float opacity=1.0f, 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** & **display_object3d** (const tp &points, const **CImgl**< tf > &primitives, const **CImgl**< T > &colors, **CImgDisplay** &disp, const bool centering=true, const int render_static=4, const int render_motion=1, const bool double_sided=false, const float focale=500.0f, const float ambiant_light=0.05f, const float opacity=1.0f, const bool display_axes=true, float *const pose_matrix=0) const

  High-level interface for displaying a 3d object.

Input-Output

- **CImg** & **load** (const char *filename)

  Load an image from a file.

- **CImg** & **load_ascii** (const char *filename)

  Load an image from an ASCII file (in-place version).

- **CImg** & **load_dlm** (const char *filename)

  Load an image from a DLM file (in-place version).

- **CImg** & **load_pnm** (const char *filename)

  Load an image from a PNM file (in-place version).

- **CImg** & **load_yuv** (const char *filename, const unsigned int sizex, const unsigned int sizey, const unsigned int first_frame=0, const int last_frame=-1, const bool yuv2rgb=true)

  Load a YUV image sequence file (in-place).

- **CImg** & **load_bmp** (const char *filename)

  Load an image from a BMP file.

- **CImg** & **load_png** (const char *filename)

  Load an image from a PNG file.

- **CImg** & **load_tiff** (const char *filename)

  Load an image from a PNG file.

- **CImg** & **load_jpeg** (const char *filename)

  Load an image from a JPEG file.

- **CImg** & **load_raw** (const char *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 endian_-swap=false)

  In-place version of **get_load_raw**()(p. 54).

- **CImg** & **load_rgba** (const char *filename, const unsigned int dimw, const unsigned int dimh)

  In-place version of **get_load_rgba**()(p. 54).

- **CImg** & **load_rgb** (const char *filename, const unsigned int dimw, const unsigned int dimh)

  In-place version of **get_load_rgb**()(p. 54).

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

  In-place version of **get_load_inr**()(p. 54).

- **CImg** & **load_pandore** (const char *filename)

  In-place version of **get_load_pandore**()(p. 54).

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

  In-place version of **get_load_analyze**()(p. 54).

- **CImg** & **load_parrec** (const char *filename, const char axis='v', const char align='p')
  In-place version of **get_load_parrec**()(p. 54).
- **CImg** & **load_cimg** (const char *filename, const char axis='v', const char align='p')
  In-place version of **get_load_cimg**()(p. 54).
- **CImg** & **load_convert** (const char *filename)
  In-place version of **get_load_convert**()(p. 54).
- **CImg** & **load_dicom** (const char *filename)
  In-place version of **get_load_dicom**()(p. 54).
- template<typename tf, typename tc> **CImg** & **load_off** (const char *filename, **CImgl**< tf > &primitives, **CImgl**< tc > &colors, const bool invert_faces=false)
  In-place version of **get_load_off**()(p. 54).
- const **CImg** & **save** (const char *filename, const int number=-1) const
  Save the image as a file.
- const **CImg** & **save_ascii** (const char *filename) const
  Save the image as an ASCII file.
- const **CImg** & **save_dlm** (const char *filename) const
  Save the image as a DLM file.
- const **CImg** & **save_pnm** (const char *filename) const
  Save the image as a PNM file.
- const **CImg** & **save_dicom** (const char *filename) const
  Save an image as a Dicom file (need '(X)Medcon' : `http://xmedcon.sourceforge.net` *).*
- const **CImg** & **save_analyze** (const char *filename, const float *const voxsize=0) const
  Save the image as an ANALYZE7.5 file.
- const **CImg** & **save_cimg** (const char *filename) const
  Save the image as a **CImg**(p. 21) *RAW file.*
- const **CImg** & **save_raw** (const char *filename, const bool multiplexed=false) const
  Save the image as a RAW file.
- const **CImg** & **save_convert** (const char *filename, const unsigned int quality=100) const
  Save the image using ImageMagick's convert.
- const **CImg** & **save_inr** (const char *filename, const float *const voxsize=0) const
  Save the image as an INRIMAGE-4 file.
- const **CImg** & **save_pandore** (const char *filename, const unsigned int colorspace=0) const
  Save the image as a PANDORE-5 file.
- const **CImg** & **save_yuv** (const char *filename, const bool rgb2yuv=true) const

Save the image as a YUV file.

- const **CImg** & **save_bmp** (const char ∗filename) const

  Save the image as a BMP file.

- const **CImg** & **save_png** (const char ∗filename) const

  Save an image to a PNG file.

- const **CImg** & **save_tiff** (const char ∗filename) const

  Save a file in TIFF format.

- const **CImg**< T > & **save_jpeg** (const char ∗filename, const unsigned int quality=100) const

  Save a file in JPEG format.

- const **CImg** & **save_rgba** (const char ∗filename) const

  Save the image as a RGBA file.

- const **CImg** & **save_rgb** (const char ∗filename) const

  Save the image as a RGB file.

- template<typename tf, typename tc> const **CImg** & **save_off** (const char ∗filename, const **CImgl**< tf > &primitives, const **CImgl**< tc > &colors, const bool invert_faces=false) const

  Save OFF files (GeomView 3D object files).

- static **CImg get_load** (const char ∗filename)

  Load an image from a file.

- static **CImg get_load_ascii** (const char ∗filename)

  Load an image from an ASCII file.

- static **CImg get_load_dlm** (const char ∗filename)

  Load an image from a DLM file.

- static **CImg get_load_pnm** (const char ∗filename)

  Load an image from a PNM file.

- static **CImg get_load_yuv** (const char ∗filename, const unsigned int sizex, const unsigned int sizey, const unsigned int first_frame=0, const int last_frame=-1, const bool yuv2rgb=true)

  Load a YUV image sequence file.

- static **CImg get_load_bmp** (const char ∗filename)

  Load an image from a BMP file.

- static **CImg get_load_png** (const char ∗filename)

  Load an image from a PNG file.

- static **CImg get_load_tiff** (const char ∗filename)

  Load an image in TIFF format.

- static **CImg get_load_jpeg** (const char ∗filename)

Load a file in JPEG format.

- static **CImg get_load_raw** (const char *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 endian_swap=false)

  Load an image from a RAW file.

- static **CImg get_load_rgba** (const char *filename, const unsigned int dimw, const unsigned int dimh)

  Load an image from a RGBA file.

- static **CImg get_load_rgb** (const char *filename, const unsigned int dimw, const unsigned int dimh)

  Load an image from a RGB file.

- static **CImg get_load_inr** (const char *filename, float *voxsize=0)

  Load an image from an INRIMAGE-4 file.

- static **CImg get_load_pandore** (const char *filename)

  Load an image from a PANDORE-5 file.

- static **CImg get_load_analyze** (const char *filename, float *voxsize=0)

  Load an image from an ANALYZE7.5 file.

- static **CImg get_load_parrec** (const char *filename, const char axe='v', const char align='p')

  Load PAR-REC (Philips) image file.

- static **CImg get_load_cimg** (const char *filename, const char axis='v', const char align='p')

  Load an image from a **CImg**(p. 21) *RAW file.*

- static **CImg get_load_convert** (const char *filename)

  this function working properly (see `http://www.imagemagick.org` *).*

- static **CImg get_load_dicom** (const char *filename)

  Load an image from a Dicom file (need '(X)Medcon' : `http://xmedcon.sourceforge.net` *).*

- template<typename tf, typename tc> static **CImg**< T > **get_load_off** (const char *filename, **CImgl**< tf > &primitives, **CImgl**< tc > &colors, const bool invert_faces=false)

  Load OFF files (GeomView 3D object files).

- static **CImg get_logo40x38** ()

  Get a 40x38 color logo of a 'danger' item.

Public Types

- typedef T * **iterator**

  Iterator type for CImg<T>.

- typedef const T * **const_iterator**

  Const iterator type for CImg<T>.

Public Attributes

- unsigned int **width**

  Variable representing the width of the instance image (i.e. dimensions along the X-axis).

- unsigned int **height**

  Variable representing the height of the instance image (i.e. dimensions along the Y-axis).

- unsigned int **depth**

  Variable representing the depth of the instance image (i.e. dimensions along the Z-axis).

- unsigned int **dim**

  Variable representing the number of channels of the instance image (i.e. dimensions along the V-axis).

- bool **is_shared**

  Variable telling if pixel buffer of the instance image is shared with another one.

- T * **data**

  Pointer to the first pixel of the pixel buffer.

4.1.1 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 structure of the CImg Library. It allows the declaration and construction of an image, the access to its pixel values, and the application of various operations on it.

Image representation

A CImg image is defined as an instance of the container **CImg**(p. 21)<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 **CImgl**(p. 119)<T> rather than simple images **CImg**(p. 21)<T>.

Thus, the **CImg**(p. 21)<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**(p. 21)<T> structure contains *five* fields :

- **width**(p. 111) defines the number of *columns* of the image (size along the X-axis).
- **height**(p. 111) defines the number of *rows* of the image (size along the Y-axis).
- **depth**(p. 111) defines the number of *slices* of the image (size along the Z-axis).
- **dim**(p. 112) defines the number of *channels* of the image (size along the V-axis).
- **data**(p. 55) defines a *pointer* to the *pixel data* (of type T).

You can access these fields publicly although it is recommended to use the dedicated functions **dimx**()(p. 58), **dimy**()(p. 58), **dimz**()(p. 58), **dimv**()(p. 59) and **ptr**()(p. 59) 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**(p. 21) 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.**(p. 13)).

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**()(p. 73) 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.**(p. 13)).

- 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**(p. 21)<T> class contains a lot of functions that operates on images. Some of the most useful are :

- **operator()**()(p. 60), **operator[ ]**()(p. 60) : allows to access or write pixel values.
- **display**()(p. 49) : displays the image in a new window.

See also:
CImgl(p. 119), **CImgStats**(p. 131), **CImgDisplay**(p. 112), **CImgException**(p. 118).

4.1.2 Member Typedef Documentation

4.1.2.1 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.

4.1.2.2 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.

4.1.3 Member Function Documentation

4.1.3.1 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.

4.1.3.2 unsigned long size () const

Return the total number of pixel values in an image.

- Equivalent to : **dimx()**(p. 58) * **dimy()**(p. 58) * **dimz()**(p. 58) * **dimv()**(p. 59).

example:

```
CImg<> img(100,100,1,3);
if (img.size()==100*100*3) std::fprintf(stderr,"This statement is true");
```

See also:
dimx()(p. 58), **dimy()**(p. 58), **dimz()**(p. 58), **dimv()**(p. 59)

4.1.3.3 int dimx () const

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

See also:
width(p. 111), **dimy()**(p. 58), **dimz()**(p. 58), **dimv()**(p. 59), **size()**(p. 58).

4.1.3.4 int dimy () const

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

See also:
height(p. 111), **dimx()**(p. 58), **dimz()**(p. 58), **dimv()**(p. 59), **size()**(p. 58).

4.1.3.5 int dimz () const

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

See also:
depth(p. 111), **dimx()**(p. 58), **dimy()**(p. 58), **dimv()**(p. 59), **size()**(p. 58).

4.1.3.6 int dimv () const

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

See also:
dim(p. 112), **dimx()**(p. 58), **dimy()**(p. 58), **dimz()**(p. 58), **size()**(p. 58).

4.1.3.7 long offset (const int *x* = 0, const int *y* = 0, const int *z* = 0, const int *v* = 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 V-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 with float-valued black
long off = img.offset(10,10,0,2);       // Get the offset of the blue value of the pixel locate
float val = img[off];                   // Get the blue value of the pixel.
```

See also:
ptr()(p. 59), **operator()()**(p. 60), **operator[ ]()**(p. 60), **How pixel data are stored with CImg.**(p. 13).

4.1.3.8 T* ptr (const unsigned int *x* = 0, const unsigned int *y* = 0, const unsigned int *z* = 0, const unsigned int *v* = 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 V-coordinate of the pixel.

- When called without parameters, **ptr()**(p. 59) returns a pointer to the begining of the pixel buffer.
- If the macro `cimg_debug` == 2, 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 = ptr(10,10);          // Get a pointer to the pixel located at (10,10).
float val = *ptr;                 // Get the pixel value.
```

See also:
data(p. 55), **offset()**(p. 59), **operator()()**(p. 60), **operator[ ]()**(p. 60), **How pixel data are stored with CImg.**(p. 13), **Setting Environment Variables**(p. 4).

4.1.3.9 T& operator() (const unsigned int *x*, const unsigned int *y* = 0, const unsigned int *z* = 0, const unsigned int *v* = 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 V-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_debug` == 2, 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 floa
const float valR = img(10,10,0,0);                   // Read the red component at coordinates
const float valG = img(10,10,0,1);                   // Read the green component at coordinate
const float valB = img(10,10,2);                     // Read the blue component at coordinates
const float avg = (valR + valG + valB)/3;            // Compute average pixel value.
img(10,10,0) = img(10,10,1) = img(10,10,2) = avg;    // Replace the pixel (10,10) by the avera
```

See also:

operator[ ]()(p. 60), **ptr**()(p. 59), **offset**()(p. 59), **How pixel data are stored with CImg.**(p. 13), **Setting Environment Variables**(p. 4).

4.1.3.10 T& operator[ ] (const unsigned long *off*)

Fast access to pixel value for reading or writing, using an offset to the image pixel.

Parameters:

off Offset of the pixel according to the begining of the pixel buffer, given by **ptr**()(p. 59).

- If the macro `cimg_debug==2`, boundary checking is performed and warning messages may appear (but function performances decrease).
- As pixel values are aligned in memory, this operator can sometime useful to access values easier than with **operator()**()(p. 60) (see example below).

example:

```
CImg<float> vec(1,10);        // Define a vector of float values (10 lines, 1 row).
const float val1 = vec(0,4);  // Get the fifth element using operator()().
const float val2 = vec[4];    // Get the fifth element using operator[]. Here, val2==val1.
```

See also:

operator()()(p. 60), **ptr**()(p. 59), **offset**()(p. 59), **How pixel data are stored with CImg.**(p. 13), **Setting Environment Variables**(p. 4).

4.1.3.11 T pix4d (const int *x*, const int *y*, const int *z*, const int *v*, const T & *out_val*) const

Read a pixel value with Dirichlet or Neumann boundary conditions.

Parameters:

x X-coordinate of the pixel.

y Y-coordinate of the pixel.

z Z-coordinate of the pixel.

v V-coordinate of the pixel.

out_val Desired value if pixel coordinates are outside the image range (optional parameter).

- This function allows to read pixel values with boundary checking on all coordinates.
- If given coordinates are outside the image range and the parameter out_val is specified, the value `out_val` is returned.
- If given coordinates are outside the image range and the parameter out_val is not specified, the closest pixel value is returned.

example:

```
CImg<float> img(100,100,1,1,128);                    // Define a 100x100 images with all pixel
const float val1 = img.pix4d(10,10,0,0,0);  // Equivalent to val1=img(10,10) (but slower).
const float val2 = img.pix4d(-4,5,0,0,0);   // Return 0, since coordinates are outside the imag
const float val3 = img.pix4d(10,10,5,0,64); // Return 64, since coordinates are outside the ima
```

See also:

operator()()(p. 60), **linear_pix4d**()(p. 61), **cubic_pix2d**()(p. 63).

4.1.3.12 cimg::largest<T,float>::type linear_pix4d (const float *fx*, const float *fy*, const float *fz*, const float *fv*, const T & *out_val*) const

Read a pixel value using linear interpolation.

Parameters:

ffx X-coordinate of the pixel (float-valued).

ffy Y-coordinate of the pixel (float-valued).

ffz Z-coordinate of the pixel (float-valued).

ffv V-coordinate of the pixel (float-valued).

out_val Out-of-border pixel value

- This function allows to read pixel values with boundary checking on all coordinates.
- If given coordinates are outside the image range, the value of the nearest pixel inside the image is returned (Neumann boundary conditions).
- If given coordinates are float-valued, a linear interpolation is performed in order to compute the returned value.

example:

```
CImg<float> img(2,2);     // Define a greyscale 2x2 image.
img(0,0) = 0;             // Fill image with specified pixel values.
img(1,0) = 1;
img(0,1) = 2;
img(1,1) = 3;
const double val = img.linear_pix4d(0.5,0.5);  // Return val=1.5, which is the average intensit
```

See also:
operator()()(p. 60), **linear_pix3d()**(p. 62), **linear_pix2d**()(p. 62), **linear_pix1d**()(p. 62), **cubic_pix2d()**(p. 63).

4.1.3.13 cimg::largest<T,float>::type linear_pix3d (const float *fx*, const float *fy*, const float *fz*, const int *v*, const T & *out_val*) const

Read a pixel value using linear interpolation for the three first coordinates (cx,cy,cz).

- Same as **linear_pix4d()**(p. 61), except that linear interpolation and boundary checking is performed only on the three first coordinates.

See also:
operator()()(p. 60), **linear_pix4d()**(p. 61), **linear_pix2d**()(p. 62), **linear_pix1d**()(p. 62), **linear_pix3d()**(p. 62), **cubic_pix2d()**(p. 63).

4.1.3.14 cimg::largest<T,float>::type linear_pix2d (const float *fx*, const float *fy*, const int *z*, const int *v*, const T & *out_val*) const

Read a pixel value using linear interpolation for the two first coordinates (cx,cy).

- Same as **linear_pix4d()**(p. 61), except that linear interpolation and boundary checking is performed only on the two first coordinates.

See also:
operator()()(p. 60), **linear_pix4d()**(p. 61), **linear_pix3d**()(p. 62), **linear_pix1d**()(p. 62), **linear_pix2d()**(p. 62), **cubic_pix2d()**(p. 63).

4.1.3.15 cimg::largest<T,float>::type linear_pix1d (const float *fx*, const int *y*, const int *z*, const int *v*, const T & *out_val*) const

Read a pixel value using linear interpolation for the first coordinate cx.

- Same as **linear_pix4d()**(p. 61), except that linear interpolation and boundary checking is performed only on the first coordinate.

See also:
operator()()(p. 60), **linear_pix4d()**(p. 61), **linear_pix3d**()(p. 62), **linear_pix2d**()(p. 62), **linear_pix1d()**(p. 62), **cubic_pix1d()**(p. 62).

4.1.3.16 cimg::largest<T,float>::type cubic_pix1d (const float *fx*, const int *y*, const int *z*, const int *v*, const T & *out_val*) const

Read a pixel value using cubic interpolation for the first coordinate cx.

- Same as **cubic_pix2d()**(p. 63), except that cubic interpolation and boundary checking is performed only on the first coordinate.

See also:
operator()()(p. 60), **cubic_pix2d()**(p. 63), **linear_pix1d()**(p. 62).

4.1.3.17 cimg::largest<T,float>::type cubic_pix2d (const float *fx*, const float *fy*, const int *z*, const int *v*, const T & *out_val*) const

Read a pixel value using bicubic interpolation.

Parameters:

pfx X-coordinate of the pixel (float-valued).

pfy Y-coordinate of the pixel (float-valued).

z Z-coordinate of the pixel.

v V-coordinate of the pixel.

- This function allows to read pixel values with boundary checking on the two first coordinates.
- If given coordinates are outside the image range, the value of the nearest pixel inside the image is returned (Neumann boundary conditions).
- If given coordinates are float-valued, a cubic interpolation is performed in order to compute the returned value.

See also:

operator()()(p. 60), **cubic_pix1d**()(p. 62), **linear_pix2d**()(p. 62).

4.1.3.18 const CImg& print (const char * *title* = 0, const unsigned int *print_flag* = 1) const

Display informations about the image on the standard error output.

Parameters:

title Name for the considered image (optional).

print_flag Level of informations to be printed.

- The possible values for `print_flag` are :
  - 0 : print only informations about image size and pixel buffer.
  - 1 : print also statistics on the image pixels.
  - 2 : print also the content of the pixel buffer, in a matlab-style.

example:

```
CImg<float> img("foo.jpg");      // Load image from a JPEG file.
img.print("Image : foo.jpg",1);  // Print image informations and statistics.
```

See also:

CImgStats(p. 131)

4.1.3.19 CImg& operator= (const T & *val*)

Assign a value to each image pixel of the instance image.

Parameters:

val Value to assign.

- Replace all pixel values of the instance image by `val`.

- Can be used to easily initialize an image.

example:

```
CImg<float> img(100,100);   // Define a 100x100 greyscale image.
img = 3.14f;                // Set all pixel values to 3.14.
```

See also:
fill()(p. 73).

4.1.3.20 CImg& operator= (const T * *buf*)

Assign values of a C-array to the instance image.

Parameters:
buf Pointer to a C-style array having a size of (at least) `this->`**size**()(p. 58).

- Replace pixel values by the content of the array `buf`.
- Warning : the value types in the array and in the image must be the same.

example:

```
float tab[4*4] = { 1,2,3,4, 5,6,7,8, 9,10,11,12, 13,14,15,16 };  // Define a 4x4 matrix in C-st
CImg<float> matrice(4,4);                                        // Define a 4x4 greyscale imag
matrice = tab;                                                   // Fill the image by the value
```

4.1.3.21 CImg operator+ (const T & *val*) const

Addition.

Parameters:
val Constant value added to each image pixel of the instance image.

4.1.3.22 CImg<typename cimg::largest<T,t>::type> operator+ (const CImg< t > & *img*) const

Addition.

Parameters:
img Image added to the instance image.

4.1.3.23 CImg& operator+= (const T & *val*)

In-place addition.

This is the in-place version of **operator+**()(p. 64).

4.1.3.24 CImg& operator+= (const CImg< t > & *img*)

In-place addition.

This is the in-place version of **operator+**()(p. 64).

4.1.3.25 CImg& operator++ ()

In-place increment.

Equivalent to `operator+=(1)`.

4.1.3.26 CImg operator- (const T & *val*) const

Substraction.

Parameters:
val Constant value substracted to each image pixel of the instance image.

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

Substraction.

Parameters:
img Image substracted to the instance image.

4.1.3.28 CImg& operator-= (const T & *val*)

In-place substraction.

This is the in-place version of **operator-**()(p. 65).

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

In-place substraction.

This is the in-place version of **operator-**()(p. 65).

4.1.3.30 CImg& operator– ()

In-place decrement.

Equivalent to `operator-=(1)`.

4.1.3.31 CImg operator ∗ (const double *val*) const

Multiplication.

Parameters:
val Constant value multiplied to each image pixel of the instance image.

4.1.3.32 CImg<typename cimg::largest<T,t>::type> operator ∗ (const CImg< t > & *img*) const

Multiplication.

Matrix multiplication.

4.1.3.33 CImg& operator *= (const double *val*)

In-place multiplication.

This is the in-place version of **operator***()(p. 65).

4.1.3.34 CImg& operator *= (const CImg< t > & *img*)

In-place multiplication.

This is the in-place version of **operator***()(p. 65).

4.1.3.35 CImg operator/ (const double *val*) const

Division.

Parameters:

val Constant value divided to each image pixel of the instance image.

4.1.3.36 CImg& operator/= (const double *val*)

In-place division.

This is the in-place version of **operator/**()(p. 66).

4.1.3.37 CImg operator% (const T & *val*) const

Modulo.

Parameters:

val Constant valued used for a modulo on each image pixel.

4.1.3.38 CImg operator% (const CImg< T > & *img*) const

Modulo.

Parameters:

img Image whose values are used for a modulo.

4.1.3.39 CImg& operator%= (const T & *val*)

In-place modulo.

This is the in-place version of **operator%**()(p. 66).

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

In-place modulo.

This is the in-place version of **operator%**()(p. 66).

4.1.3.41 CImg operator & (const T & *val*) const

Bitwise AND.

Parameters:
val Constant valued used for a bitwise AND on each image pixel.

4.1.3.42 CImg operator & (const CImg< T > & *img*) const

Bitwise AND.

Parameters:
img Image whose value are used for the AND.

4.1.3.43 CImg& operator &= (const T & *val*)

In-place bitwise AND.

This is the in-place version of **operator&**()(p. 67).

4.1.3.44 CImg& operator &= (const CImg< T > & *img*)

In-place bitwise AND.

This is the in-place version of **operator&=**()(p. 67).

4.1.3.45 CImg operator| (const T & *val*) const

Bitwise OR.

Parameters:
val Constant valued used for a bitwise OR on each image pixel.

4.1.3.46 CImg operator| (const CImg< T > & *img*) const

Bitwise OR.

Parameters:
img Image whose values are used for the OR.

4.1.3.47 CImg& operator|= (const T & *val*)

In-place bitwise OR.

This is the in-place version of **operator|**()(p. 67).

4.1.3.48 CImg& operator|= (const CImg< T > & *img*)

In-place bitwise OR.

This is the in-place version of **operator|**()(p. 67).

4.1.3.49 CImg operator^ (const T & *val*) const

Bitwise XOR.

Parameters:
val Constant valued used for a bitwise XOR on each image pixel.

4.1.3.50 CImg operator^ (const CImg< T > & *img*) const

Bitwise XOR.

Parameters:
img Image whose values are used for the XOR.

4.1.3.51 CImg& operator^= (const T & *val*)

In-place bitwise XOR.

This is the in-place version of **operator^**()(p. 68).

4.1.3.52 CImg& operator^= (const CImg< T > & *img*)

In-place bitwise XOR.

This is the in-place version of **operator^**()(p. 68).

4.1.3.53 CImg& mul (const CImg< t > & *img*)

In-place pointwise multiplication between `*this` and `img`.

This is the in-place version of **get_mul**()(p. 68).

See also:
get_mul()(p. 68).

4.1.3.54 CImg<typename cimg::largest<T,t>::type> get_mul (const CImg< t > & *img*) const

Pointwise multiplication between `*this` and `img`.

Parameters:
img Argument of the multiplication.
- if `*this` and `img` have different size, the multiplication is applied on the maximum possible range.

See also:
get_div()(p. 69),**mul**()(p. 68),**div**()(p. 68)

4.1.3.55 CImg& div (const CImg< t > & *img*)

Replace the image by the pointwise division between `*this` and `img`.

This is the in-place version of **get_div**()(p. 69).

See also:
get_div()(p. 69).

4.1.3.56 CImg<typename cimg::largest<T,t>::type> get_div (const CImg< t > & *img*) const

Return an image from a pointwise division between `*this` and `img`.

Parameters:
img = argument of the division.

Note:
if `*this` and `img` have different size, the division is applied only on possible values.

See also:
get_mul()(p. 68),**mul()**(p. 68),**div()**(p. 68)

4.1.3.57 CImg& max (const CImg< t > & *img*)

Replace the image by the pointwise max operator between `*this` and `img`.

This is the in-place version of **get_max()**(p. 69).

See also:
get_max()(p. 69).

4.1.3.58 CImg<typename cimg::largest<T,t>::type> get_max (const CImg< t > & *img*) const

Return the image corresponding to the max value for each pixel.

Parameters:
img = second argument of the max operator (the first one is *this).

See also:
max()(p. 69), **min()**(p. 70), **get_min()**(p. 70)

4.1.3.59 CImg& max (const T & *val*)

Replace the image by the pointwise max operator between `*this` and `val`.

This is the in-place version of **get_max()**(p. 69).

See also:
get_max()(p. 69).

4.1.3.60 CImg get_max (const T & *val*) const

Return the image corresponding to the max value for each pixel.

Parameters:
val = second argument of the max operator (the first one is *this).

See also:
max()(p. 69), **min()**(p. 70), **get_min()**(p. 70)

4.1.3.61 CImg& min (const CImg< t > & *img*)

Replace the image by the pointwise min operator between `*this` and `img`.

This is the in-place version of **get_min()**(p. 70).

See also:
get_min()(p. 70).

4.1.3.62 CImg<typename cimg::largest<T,t>::type> get_min (const CImg< t > & *img*) const

Return the image corresponding to the min value for each pixel.

Parameters:
img = second argument of the min operator (the first one is *this).

See also:
min()(p. 70), **max()**(p. 69), **get_max()**(p. 69)

4.1.3.63 CImg& min (const T & *val*)

Replace the image by the pointwise min operator between `*this` and `val`.

This is the in-place version of **get_min()**(p. 70).

See also:
get_min()(p. 70).

4.1.3.64 CImg get_min (const T & *val*) const

Return the image corresponding to the min value for each pixel.

Parameters:
val = second argument of the min operator (the first one is *this).

See also:
min()(p. 70), **max()**(p. 69), **get_max()**(p. 69)

4.1.3.65 CImg& sqrt ()

Replace each image pixel by its square root.

See also:
get_sqrt()(p. 70)

4.1.3.66 CImg<typename cimg::largest<T,float>::type> get_sqrt () const

Return the image of the square root of the pixel values.

See also:
sqrt()(p. 70)

4.1.3.67 CImg& log ()

Replace each image pixel by its log.

See also:
get_log()(p. 71), **log10**()(p. 71), **get_log10**()(p. 71)

4.1.3.68 CImg<typename cimg::largest<T,float>::type> get_log () const

Return the image of the log of the pixel values.

See also:
log()(p. 71), **log10**()(p. 71), **get_log10**()(p. 71)

4.1.3.69 CImg& log10 ()

Replace each image pixel by its log10.

See also:
get_log10()(p. 71), **log**()(p. 71), **get_log**()(p. 71)

4.1.3.70 CImg<typename cimg::largest<T,float>::type> get_log10 () const

Return the image of the log10 of the pixel values.

See also:
log10()(p. 71), **log**()(p. 71), **get_log**()(p. 71)

4.1.3.71 CImg& pow (const double *p*)

Replace each image pixel by its power by p.

Parameters:
p = power

See also:
get_pow()(p. 71), **sqrt**()(p. 70), **get_sqrt**()(p. 70)

4.1.3.72 CImg<typename cimg::largest<T,float>::type> get_pow (const double *p*) const

Return the image of the square root of the pixel values.

Parameters:
p = power

See also:
pow()(p. 71), **sqrt**()(p. 70), **get_sqrt**()(p. 70)

4.1.3.73 CImg& pow (const CImg< t > & *img*)

Return each image pixel (*this)(x,y,z,k) by its power by `img(x,y,z,k)`.

In-place version

4.1.3.74 CImg& abs ()

Replace each pixel value by its absolute value.

See also:
get_abs()(p. 72)

4.1.3.75 CImg<typename cimg::largest<T,float>::type> get_abs () const

Return the image of the absolute value of the pixel values.

See also:
abs()(p. 72)

4.1.3.76 CImg& cos ()

Replace each image pixel by its cosinus.

See also:
get_cos()(p. 72), **sin**()(p. 72), **get_sin**()(p. 72), **tan**()(p. 73), **get_tan**()(p. 73)

4.1.3.77 CImg<typename cimg::largest<T,float>::type> get_cos () const

Return the image of the cosinus of the pixel values.

See also:
cos()(p. 72), **sin**()(p. 72), **get_sin**()(p. 72), **tan**()(p. 73), **get_tan**()(p. 73)

4.1.3.78 CImg& sin ()

Replace each image pixel by its sinus.

See also:
get_sin()(p. 72), **cos**()(p. 72), **get_cos**()(p. 72), **tan**()(p. 73), **get_tan**()(p. 73)

4.1.3.79 CImg<typename cimg::largest<T,float>::type> get_sin () const

Return the image of the sinus of the pixel values.

See also:
sin()(p. 72), **cos**()(p. 72), **get_cos**()(p. 72), **tan**()(p. 73), **get_tan**()(p. 73)

4.1.3.80 CImg& tan ()

Replace each image pixel by its tangent.

See also:
get_tan()(p. 73), **cos**()(p. 72), **get_cos**()(p. 72), **sin**()(p. 72), **get_sin**()(p. 72)

4.1.3.81 CImg<typename cimg::largest<T,float>::type> get_tan () const

Return the image of the tangent of the pixel values.

See also:
tan()(p. 73), **cos**()(p. 72), **get_cos**()(p. 72), **sin**()(p. 72), **get_sin**()(p. 72)

4.1.3.82 CImg& 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`.

See also:
operator=()(p. 21).

4.1.3.83 CImg& fill (const T & *val0*, const T & *val1*)

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

Parameters:
val0 = fill value 1
val1 = fill value 2

4.1.3.84 CImg& fill (const T & *val0*, const T & *val1*, const T & *val2*)

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

Parameters:
val0 = fill value 1
val1 = fill value 2
val2 = fill value 3

4.1.3.85 CImg& 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*.

Parameters:

val0 = fill value 1

val1 = fill value 2

val2 = fill value 3

val3 = fill value 4

4.1.3.86 CImg& 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*.

Parameters:

val0 = fill value 1

val1 = fill value 2

val2 = fill value 3

val3 = fill value 4

val4 = fill value 5

4.1.3.87 CImg& 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*.

Parameters:

val0 = fill value 1

val1 = fill value 2

val2 = fill value 3

val3 = fill value 4

val4 = fill value 5

val5 = fill value 6

4.1.3.88 CImg& fill (const T & *val0*, const T & *val1*, const T & *val2*, const T & *val3*, const T & *val4*, const T & *val5*, const T & *val6*)

Fill sequentially all pixel values with values *val0* and *val1* and *val2* and *val3* and *val4* and *val5*.

Parameters:

val0 = fill value 1

val1 = fill value 2

val2 = fill value 3

val3 = fill value 4

val4 = fill value 5

val5 = fill value 6

val6 = fill value 7

4.1.3.89 CImg& fill (const T & *val0*, const T & *val1*, const T & *val2*, const T & *val3*, const T & *val4*, const T & *val5*, const T & *val6*, const T & *val7*)

Fill sequentially all pixel values with values *val0* and *val1* and *val2* and *val3* and ... and *val7*.

Parameters:

val0 = fill value 1
val1 = fill value 2
val2 = fill value 3
val3 = fill value 4
val4 = fill value 5
val5 = fill value 6
val6 = fill value 7
val7 = fill value 8

4.1.3.90 CImg& fill (const T & *val0*, const T & *val1*, const T & *val2*, const T & *val3*, const T & *val4*, const T & *val5*, const T & *val6*, const T & *val7*, const T & *val8*)

Fill sequentially all pixel values with values *val0* and *val1* and *val2* and *val3* and ... and *val8*.

Parameters:

val0 = fill value 1
val1 = fill value 2
val2 = fill value 3
val3 = fill value 4
val4 = fill value 5
val5 = fill value 6
val6 = fill value 7
val7 = fill value 8
val8 = fill value 9

4.1.3.91 CImg& fill (const T & *val0*, const T & *val1*, const T & *val2*, const T & *val3*, const T & *val4*, const T & *val5*, const T & *val6*, const T & *val7*, const T & *val8*, const T & *val9*)

Fill sequentially all pixel values with values *val0* and *val1* and *val2* and *val3* and ... and *val9*.

Parameters:

val0 = fill value 1
val1 = fill value 2
val2 = fill value 3
val3 = fill value 4
val4 = fill value 5
val5 = fill value 6
val6 = fill value 7
val7 = fill value 8
val8 = fill value 9
val9 = fill value 10

4.1.3.92 CImg& fill (const T & *val0*, const T & *val1*, const T & *val2*, const T & *val3*, const T & *val4*, const T & *val5*, const T & *val6*, const T & *val7*, const T & *val8*, const T & *val9*, const T & *val10*, const T & *val11*)

Fill sequentially all pixel values with values *val0* and *val1* and *val2* and *val3* and ... and *val11*.

Parameters:

val0 = fill value 1

val1 = fill value 2

val2 = fill value 3

val3 = fill value 4

val4 = fill value 5

val5 = fill value 6

val6 = fill value 7

val7 = fill value 8

val8 = fill value 9

val9 = fill value 10

val10 = fill value 11

val11 = fill value 12

4.1.3.93 CImg& fill (const T & *val0*, const T & *val1*, const T & *val2*, const T & *val3*, const T & *val4*, const T & *val5*, const T & *val6*, const T & *val7*, const T & *val8*, const T & *val9*, const T & *val10*, const T & *val11*, const T & *val12*)

Fill sequentially all pixel values with values *val0* and *val1* and *val2* and *val3* and ... and *val11*.

Parameters:

val0 = fill value 1

val1 = fill value 2

val2 = fill value 3

val3 = fill value 4

val4 = fill value 5

val5 = fill value 6

val6 = fill value 7

val7 = fill value 8

val8 = fill value 9

val9 = fill value 10

val10 = fill value 11

val11 = fill value 12

val12 = fill value 13

4.1.3.94 CImg& fill (const T & *val0*, const T & *val1*, const T & *val2*, const T & *val3*, const T & *val4*, const T & *val5*, const T & *val6*, const T & *val7*, const T & *val8*, const T & *val9*, const T & *val10*, const T & *val11*, const T & *val12*, const T & *val13*, const T & *val14*, const T & *val15*)

Fill sequentially all pixel values with values *val0* and *val1* and *val2* and *val3* and ... and *val15*.

Parameters:

val0 = fill value 1

val1 = fill value 2

val2 = fill value 3

val3 = fill value 4

val4 = fill value 5

val5 = fill value 6

val6 = fill value 7

val7 = fill value 8

val8 = fill value 9

val9 = fill value 10

val10 = fill value 11

val11 = fill value 12

val12 = fill value 13

val13 = fill value 14

val14 = fill value 15

val15 = fill value 16

4.1.3.95 CImg& 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 T & *val16*, const T & *val17*, const T & *val18*, const T & *val19*, const T & *val20*, const T & *val21*, const T & *val22*, const T & *val23*, const T & *val24*)

Fill sequentially all pixel values with values *val0* and *val1* and *val2* and *val3* and ... and *val24*.

Parameters:

val0 = fill value 1

val1 = fill value 2

val2 = fill value 3

val3 = fill value 4

val4 = fill value 5

val5 = fill value 6

val6 = fill value 7

val7 = fill value 8

val8 = fill value 9

val9 = fill value 10

val10 = fill value 11

val11 = fill value 12

val12 = fill value 13
val13 = fill value 14
val14 = fill value 15
val15 = fill value 16
val16 = fill value 17
val17 = fill value 18
val18 = fill value 19
val19 = fill value 20
val20 = fill value 21
val21 = fill value 22
val22 = fill value 23
val23 = fill value 24
val24 = fill value 25

4.1.3.96 CImg& normalize (const T & *a*, const T & *b*)

Linear normalization of the pixel values between *a* and *b*.

Parameters:
a = minimum pixel value after normalization.
b = maximum pixel value after normalization.

See also:
get_normalize()(p. 78), **cut**()(p. 78), **get_cut**()(p. 79).

4.1.3.97 CImg get_normalize (const T & *a*, const T & *b*) const

Return the image of normalized values.

Parameters:
a = minimum pixel value after normalization.
b = maximum pixel value after normalization.

See also:
normalize()(p. 78), **cut**()(p. 78), **get_cut**()(p. 79).

4.1.3.98 CImg& cut (const T & *a*, const T & *b*)

Cut pixel values between *a* and *b*.

Parameters:
a = minimum pixel value after cut.
b = maximum pixel value after cut.

See also:
get_cut()(p. 79), **normalize**()(p. 78), **get_normalize**()(p. 78).

4.1.3.99 CImg get_cut (const T & *a*, const T & *b*) const

Return the image of cutted values.

Parameters:
a = minimum pixel value after cut.
b = maximum pixel value after cut.

See also:
cut()(p. 78), **normalize()**(p. 78), **get_normalize()**(p. 78).

4.1.3.100 CImg& quantize (const unsigned int *n* = 256)

Quantize pixel values into

levels.

Parameters:
n = number of quantification levels

See also:
get_quantize()(p. 79).

4.1.3.101 CImg get_quantize (const unsigned int *n* = 256) const

Return a quantified image, with

levels.

Parameters:
n = number of quantification levels

See also:
quantize()(p. 79).

4.1.3.102 CImg& threshold (const T & *thres*)

Threshold the image.

Parameters:
thres = threshold

See also:
get_threshold()(p. 79).

4.1.3.103 CImg get_threshold (const T & *thres*) const

Return a thresholded image.

Parameters:
thres = threshold.

See also:
threshold()(p. 79).

4.1.3.104 CImg get_rotate (const float *angle*, const unsigned int *cond* = 3) const

Return a rotated image.

Parameters:

angle = rotation angle (in degrees).

cond = rotation type. can be :

- 0 = zero-value at borders
- 1 = repeat image at borders
- 2 = zero-value at borders and linear interpolation

Note:

Returned image will probably have a different size than the instance image ∗this.

See also:

rotate()(p. 80)

4.1.3.105 CImg& rotate (const float *angle*, const unsigned int *cond* = 3)

Rotate the image.

Parameters:

angle = rotation angle (in degrees).

cond = rotation type. can be :

- 0 = zero-value at borders
- 1 = repeat image at borders
- 2 = zero-value at borders and linear interpolation

See also:

get_rotate()(p. 80)

4.1.3.106 CImg get_rotate (const float *angle*, const float *cx*, const float *cy*, const float *zoom* = 1, const unsigned int *cond* = 3) const

Return a rotated image around the 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

See also:

rotate()(p. 80)

4.1.3.107 CImg& rotate (const float *angle*, const float *cx*, const float *cy*, const float *zoom* = 1, const unsigned int *cond* = 3)

Rotate the image around the 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

Note:

Rotation does not change the image size. If you want to get an image with a new size, use **get_rotate**()(p. 80) instead.

See also:

get_rotate()(p. 80)

4.1.3.108 CImg get_resize (const int *pdx* = -100, const int *pdy* = -100, const int *pdz* = -100, const int *pdv* = -100, const unsigned int *interp* = 1) const

Return a resized image.

Parameters:

pdx = Number of columns (new size along the X-axis).

pdy = Number of rows (new size along the Y-axis).

pdz = Number of slices (new size along the Z-axis).

pdv = Number of vector-channels (new size along the V-axis).

interp = Resizing type :

- 0 = no interpolation : additionnal space is filled with 0.
- 1 = bloc interpolation (nearest point).
- 2 = mosaic : image is repeated if necessary.
- 3 = linear interpolation.
- 4 = grid interpolation.
- 5 = bi-cubic interpolation.

Note:

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

4.1.3.109 CImg get_resize (const CImg< t > & *src*, const unsigned int *interp* = 1) const

Return a resized image.

Parameters:

src = Image giving the geometry of the resize.

interp = Resizing type :

- 0 = no interpolation : additionnal space is filled with 0.
- 1 = bloc interpolation (nearest point).
- 2 = mosaic : image is repeated if necessary.
- 3 = linear interpolation.
- 4 = grid interpolation.
- 5 = bi-cubic interpolation.

Note:

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

4.1.3.110 CImg get_resize (const CImgDisplay & *disp*, const unsigned int *interp* = 1) const

Return a resized image.

Parameters:

disp = Display giving the geometry of the resize.

interp = Resizing type :

- 0 = no interpolation : additionnal space is filled with 0.
- 1 = bloc interpolation (nearest point).
- 2 = mosaic : image is repeated if necessary.
- 3 = linear interpolation.
- 4 = grid interpolation.
- 5 = bi-cubic interpolation.

Note:

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

4.1.3.111 CImg& resize (const int *pdx* = -100, const int *pdy* = -100, const int *pdz* = -100, const int *pdv* = -100, const unsigned int *interp* = 1)

Resize the image.

Parameters:

pdx = Number of columns (new size along the X-axis).

pdy = Number of rows (new size along the Y-axis).

pdz = Number of slices (new size along the Z-axis).

pdv = Number of vector-channels (new size along the V-axis).

interp = Resizing type :

- 0 = no interpolation : additionnal space is filled with 0.
- 1 = bloc interpolation (nearest point).

- 2 = mosaic : image is repeated if necessary.
- 3 = linear interpolation.
- 4 = grid interpolation.
- 5 = bi-cubic interpolation.

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

4.1.3.112 CImg& resize (const CImg< t > & *src*, const unsigned int *interp* = 1)

Resize the image.

Parameters:
src = Image giving the geometry of the resize.
interp = Resizing type :

- 0 = no interpolation : additionnal space is filled with 0.
- 1 = bloc interpolation (nearest point).
- 2 = mosaic : image is repeated if necessary.
- 3 = linear interpolation.
- 4 = grid interpolation.
- 5 = bi-cubic interpolation.

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

4.1.3.113 CImg& resize (const CImgDisplay & *disp*, const unsigned int *interp* = 1)

Resize the image.

Parameters:
disp = Display giving the geometry of the resize.
interp = Resizing type :

- 0 = no interpolation : additionnal space is filled with 0.
- 1 = bloc interpolation (nearest point).
- 2 = mosaic : image is repeated if necessary.
- 3 = linear interpolation.
- 4 = grid interpolation.
- 5 = bi-cubic interpolation.

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

4.1.3.114 CImg get_resize_halfXY () const

Return an half-resized image, using a special filter.

See also:
resize_halfXY()(p. 84), **resize**()(p. 82), **get_resize**()(p. 81).

4.1.3.115 CImg& resize_halfXY ()

Half-resize the image, using a special filter.

See also:
get_resize_halfXY()(p. 83), **resize()**(p. 82), **get_resize()**(p. 81).

4.1.3.116 CImg get_crop (const unsigned int *x0*, const unsigned int *y0*, const unsigned int *z0*, const unsigned int *v0*, const unsigned int *x1*, const unsigned int *y1*, const unsigned int *z1*, const unsigned int *v1*, const bool *border_condition* = `false`) const

Return a square region of the image, as a new 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.
v0 = V-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.
v1 = V-coordinate of the lower-right crop rectangle corner.
border_condition = Dirichlet (false) or Neumann border conditions.

See also:
crop()(p. 85)

4.1.3.117 CImg get_crop (const unsigned int *x0*, const unsigned int *y0*, const unsigned int *z0*, const unsigned int *x1*, const unsigned int *y1*, const unsigned int *z1*, const bool *border_condition* = `false`) const

Return a square region of the image, as a new 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.

See also:
crop()(p. 85)

4.1.3.118 CImg get_crop (const unsigned int *x0*, const unsigned int *y0*, const unsigned int *x1*, const unsigned int *y1*, const bool *border_condition* = `false`) const

Return a square region of the image, as a new 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.

See also:

crop()(p. 85)

4.1.3.119 CImg get_crop (const unsigned int *x0*, const unsigned int *x1*, const bool *border_condition* = `false`) const

Return a square region of the image, as a new 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.

See also:

crop()(p. 85)

4.1.3.120 CImg& crop (const unsigned int *x0*, const unsigned int *y0*, const unsigned int *z0*, const unsigned int *v0*, const unsigned int *x1*, const unsigned int *y1*, const unsigned int *z1*, const unsigned int *v1*, const bool *border_condition* = `false`)

Replace the image by 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.

v0 = V-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.

v1 = V-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.

See also:

get_crop()(p. 84)

4.1.3.121 CImg& crop (const unsigned int *x0*, const unsigned int *y0*, const unsigned int *z0*, const unsigned int *x1*, const unsigned int *y1*, const unsigned int *z1*, const bool *border_condition* = `false`)

Replace the image by 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.

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.

See also:

get_crop()(p. 84)

4.1.3.122 CImg& crop (const unsigned int *x0*, const unsigned int *y0*, const unsigned int *x1*, const unsigned int *y1*, const bool *border_condition* = `false`)

Replace the image by 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.

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.

See also:

get_crop()(p. 84)

4.1.3.123 CImg& crop (const unsigned int *x0*, const unsigned int *x1*, const bool *border_condition* = `false`)

Replace the image by a square region of the 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.

See also:

get_crop()(p. 84)

4.1.3.124 CImg get_slice (const unsigned int *z0* = 0) const

Get the z-slice z of $*$this, as a new image.

Parameters:
z0 = Z-slice to return.

See also:
slice()(p. 34), **get_channel()**(p. 34), **channel()**(p. 34), **get_plane()**(p. 33), **plane()**(p. 33).

4.1.3.125 CImg& mirror (const char *axe* = 'x')

Mirror an image along the specified axis.

This is the in-place version of **get_mirror()**(p. 87).

See also:
get_mirror()(p. 87).

4.1.3.126 CImg get_mirror (const char *axe* = 'x')

Get a mirrored version of the image, along the specified axis.

Parameters:
axe Axe used to mirror the image. Can be 'x', 'y', 'z' or 'v'.

See also:
mirror()(p. 87).

4.1.3.127 CImg& scroll (const int *scrollx*, const int *scrolly* = 0, const int *scrollz* = 0, const int *scrollv* = 0, const int *border_condition* = 0)

Scroll the image.

This is the in-place version of **get_scroll()**(p. 87).

See also:
get_scroll()(p. 87).

4.1.3.128 CImg get_scroll (const int *scrollx*, const int *scrolly* = 0, const int *scrollz* = 0, const int *scrollv* = 0, const int *border_condition* = 0) const

Return a scrolled image.

Parameters:
scrollx Amount of displacement along the X-axis.
scrolly Amount of displacement along the Y-axis.
scrollz Amount of displacement along the Z-axis.
scrollv Amount of displacement along the V-axis.
border_condition Border condition.

- `border_condition` can be :
  - 0 : Zero border condition (Dirichlet).
  - 1 : Nearest neighbors (Neumann).
  - 2 : Repeat Pattern (Fourier style).

4.1.3.129 CImg<float> get_histogram (const unsigned int *nblevels* = 256, const T *val_min* = (T)0, const T *val_max* = (T)0) const

Return the image histogram.

The histogram H of an image I is a 1D-function where H(x) is the number of occurences of the value x in I.

Parameters:

nblevels = Number of different levels of the computed histogram. For classical images, this value is 256 (default value). You should specify more levels if you are working with CImg<float> or images with high range of pixel values.

val_min = Minimum value considered for the histogram computation. All pixel values lower than val_min won't be counted.

val_max = Maximum value considered for the histogram computation. All pixel values higher than val_max won't be counted.

Note:

If val_min==val_max==0 (default values), the function first estimates the minimum and maximum pixel values of the current image, then uses these values for the histogram computation.

Returns:

The histogram is returned as a 1D CImg<float> image H, having a size of (nblevels,1,1,1) such that H(0) and H(nblevels-1) are respectively equal to the number of occurences of the values val_min and val_max in I.

Note:

Histogram computation always returns a 1D function. Histogram of multi-valued (such as color) images are not multi-dimensional.

See also:

get_equalize_histogram()(p. 88), **equalize_histogram**()(p. 88)

4.1.3.130 CImg& equalize_histogram (const unsigned int *nblevels* = 256, const T *val_min* = (T)0, const T *val_max* = (T)0)

Equalize the image histogram.

This is the in-place version of **get_equalize_histogram**()(p. 88)

4.1.3.131 CImg get_equalize_histogram (const unsigned int *nblevels* = 256, const T *val_min* = (T)0, const T *val_max* = (T)0) const

Return the histogram-equalized version of the current image.

The histogram equalization is a classical image processing algorithm that enhances the image contrast by expanding its histogram.

Parameters:

nblevels = Number of different levels of the computed histogram. For classical images, this value is 256 (default value). You should specify more levels if you are working with CImg<float> or images with high range of pixel values.

val_min = Minimum value considered for the histogram computation. All pixel values lower than val_min won't be changed.

val_max = Maximum value considered for the histogram computation. All pixel values higher than val_max won't be changed.

Note:

If val_min==val_max==0 (default values), the function acts on all pixel values of the image.

Returns:

A new image with same size is returned, where pixels have been equalized.

See also:

get_histogram()(p. 88), **equalize_histogram()**(p. 88)

4.1.3.132 CImg<typename cimg::largest<T,float>::type> get_norm_pointwise (int *norm_type* = 2) const

Return the scalar image of vector norms.

When dealing with vector-valued images (i.e images with **dimv()**(p. 59)>1), this function computes the L1,L2 or Linf norm of each vector-valued pixel.

Parameters:

norm_type = Type of the norm being computed (1 = L1, 2 = L2, -1 = Linf).

Returns:

A scalar-valued image CImg<float> with size (**dimx()**(p. 58),**dimy()**(p. 58),**dimz()**(p. 58),1), where each pixel is the norm of the corresponding pixels in the original vector-valued image.

See also:

get_orientation_pointwise(p. 89), **orientation_pointwise**(p. 90), **norm_pointwise**(p. 89).

4.1.3.133 CImg& norm_pointwise ()

Replace each pixel value with its vector norm.

This is the in-place version of **get_norm_pointwise()**(p. 89).

Note:

Be careful when using this function on CImg<T> with T=char, unsigned char,unsigned int or int. The vector norm is usually a floating point value, and a rough cast will be done here.

4.1.3.134 CImg<typename cimg::largest<T,float>::type> get_orientation_pointwise () const

Return the image of normalized vectors.

When dealing with vector-valued images (i.e images with **dimv()**(p. 59)>1), this function return the image of normalized vectors (unit vectors). Null vectors are unchanged. The L2-norm is computed for the normalization.

Returns:
A new vector-valued image with same size, where each vector-valued pixels have been normalized.

See also:
get_norm_pointwise(p. 89), **norm_pointwise**(p. 89), **orientation_pointwise**(p. 90).

4.1.3.135 CImg& orientation_pointwise ()

Replace each pixel value by its normalized vector.

This is the in-place version of **get_orientation_pointwise()**(p. 89)

4.1.3.136 CImgl<typename cimg::largest<T,float>::type> get_gradientXY (const int *scheme* = 0) const

Return a 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.

4.1.3.137 CImgl<typename cimg::largest<T,float>::type> get_gradientXYZ (const int *scheme* = 0) const

Return a list of images, corresponding to the XYZ-gradients of an image.

See also:
get_gradientXY()(p. 90).

4.1.3.138 const CImg& marching_squares (const float *isovalue*, const float *resx*, const float *resy*, CImgl< tp > & *points*, CImgl< tf > & *primitives*) const

Get a vectorization of an implicit function defined by the instance image.

This version allows to specify the marching squares resolution along x,y, and z.

4.1.3.139 const CImg& marching_cubes (const float *isovalue*, const float *resx*, const float *resy*, const float *resz*, CImgl< tp > & *points*, CImgl< tf > & *primitives*, const bool *invert_faces* = false) const

Get a triangulation of an implicit function defined by the instance image.

This version allows to specify the marching cube resolution along x,y and z.

4.1.3.140 static CImg<T> get_default_LUT8 () [static]

Return the default 256 colors palette.

The default color palette is used by CImg when displaying images on 256 colors displays. It consists in the quantification of the (R,G,B) color space using 3:3:2 bits for color coding (i.e 8 levels for the Red and Green and 4 levels for the Blue).

Returns:
A 256x1x1x3 color image defining the palette entries.

4.1.3.141 CImg<t> get_RGBtoLUT (const CImg< t > & *palette*, const bool *dithering* = true, const bool *indexing* = false) const

Convert color pixels from (R,G,B) to match a specified palette.

This function return a (R,G,B) image where colored pixels are constrained to match entries of the specified color `palette`.

Parameters:
palette User-defined palette that will constraint the color conversion.

dithering Enable/Disable Floyd-Steinberg dithering.

indexing If `true`, each resulting image pixel is an index to the given color palette. Otherwise, (R,G,B) values of the palette are copied instead.

4.1.3.142 CImg<T> get_RGBtoLUT (const bool *dithering* = true, const bool *indexing* = false) const

Convert color pixels from (R,G,B) to match the default 256 colors palette.

Same as **get_RGBtoLUT()**(p. 91) with the default color palette given by **get_default_LUT8()**(p. 91).

4.1.3.143 CImg& RGBtoLUT (const CImg< T > & *palette*, const bool *dithering* = true, const bool *indexing* = false)

Convert color pixels from (R,G,B) to match the specified color palette.

This is the in-place version of **get_RGBtoLUT()**(p. 91).

4.1.3.144 CImg& RGBtoLUT (const bool *dithering* = true, const bool *indexing* = false)

Convert color pixels from (R,G,B) to match the specified color palette.

This is the in-place version of **get_RGBtoLUT()**(p. 91).

4.1.3.145 CImg& draw_point (const int *x0*, const int *y0*, const int *z0*, const T *const *color*, const float *opacity* = 1)

Draw a colored point in the instance image, at coordinates (`x0`,`y0`,`z0`).

Parameters:
x0 = X-coordinate of the vector-valued pixel to plot.

y0 = Y-coordinate of the vector-valued pixel to plot.

z0 = Z-coordinate of the vector-valued pixel to plot.

color = array of **dimv**()(p. 59) values of type `T`, defining the drawing color.

opacity = opacity of the drawing.

Note:
Clipping is supported.

4.1.3.146 CImg& draw_point (const int *x0*, const int *y0*, const T *const *color*, const float *opacity* = 1)

Draw a colored point in the instance image, at coordinates (`x0`,`y0`).

Parameters:
x0 = X-coordinate of the vector-valued pixel to plot.

y0 = Y-coordinate of the vector-valued pixel to plot.

color = array of **dimv**()(p. 59) values of type `T`, defining the drawing color.

opacity = opacity of the drawing.

Note:
Clipping is supported.

4.1.3.147 CImg& draw_line (const int *x0*, const int *y0*, const int *x1*, const int *y1*, const T *const *color*, const unsigned int *pattern* = ~0L, const float *opacity* = 1)

Draw a 2D colored line in the instance image, at coordinates (`x0`,`y0`)-(`x1`,`y1`).

Parameters:
x0 = X-coordinate of the starting point of the line.

y0 = Y-coordinate of the starting point of the line.

x1 = X-coordinate of the ending point of the line.

y1 = Y-coordinate of the ending point of the line.

color = array of **dimv**()(p. 59) values of type `T`, defining the drawing color.

pattern = An integer whose bits describes the line pattern.

opacity = opacity of the drawing.

Note:
Clipping is supported.

4.1.3.148 CImg& draw_line (const int *x0*, const int *y0*, const int *z0*, const int *x1*, const int *y1*, const int *z1*, const T *const *color*, const unsigned int *pattern* = ~0L, const float *opacity* = 1)

Draw a 3D colored line in the instance image, at coordinates (`x0`,`y0`,`z0`)-(`x1`,`y1`,`z1`).

Parameters:
x0 = X-coordinate of the starting point of the line.

y0 = Y-coordinate of the starting point of the line.

z0 = Z-coordinate of the starting point of the line.

x1 = X-coordinate of the ending point of the line.
y1 = Y-coordinate of the ending point of the line.
z1 = Z-coordinate of the ending point of the line.
color = array of **dimv**()(p. 59) values of type `T`, defining the drawing color.
pattern = An integer whose bits describes the line pattern.
opacity = opacity of the drawing.

Note:
Clipping is supported.

4.1.3.149 CImg& draw_line (const int *x0*, const int *y0*, const int *x1*, const int *y1*, const CImg< t > & *texture*, const int *tx0*, const int *ty0*, const int *tx1*, const int *ty1*, const float *opacity* = 1)

Draw a 2D textured line in the instance image, at coordinates (`x0`,`y0`)-(`x1`,`y1`).

Parameters:
x0 = X-coordinate of the starting point of the line.
y0 = Y-coordinate of the starting point of the line.
x1 = X-coordinate of the ending point of the line.
y1 = Y-coordinate of the ending point of the line.
texture = a colored texture image used to draw the line color.
tx0 = X-coordinate of the starting point of the texture.
ty0 = Y-coordinate of the starting point of the texture.
tx1 = X-coordinate of the ending point of the texture.
ty1 = Y-coordinate of the ending point of the texture.
opacity = opacity of the drawing.

Note:
Clipping is supported, but texture coordinates do not support clipping.

4.1.3.150 CImg& draw_arrow (const int *x0*, const int *y0*, const int *x1*, const int *y1*, const T *const *color*, const float *angle* = 30, const float *length* = -10, const unsigned int *pattern* = ~0L, const float *opacity* = 1)

Draw a 2D colored arrow in the instance image, at coordinates (`x0`,`y0`)->(`x1`,`y1`).

Parameters:
x0 = X-coordinate of the starting point of the arrow (tail).
y0 = Y-coordinate of the starting point of the arrow (tail).
x1 = X-coordinate of the ending point of the arrow (head).
y1 = Y-coordinate of the ending point of the arrow (head).
color = array of **dimv**()(p. 59) values of type `T`, defining the drawing color.
angle = aperture angle of the arrow head
length = length of the arrow head. If <0, described as a percentage of the arrow length.
pattern = An integer whose bits describes the line pattern.
opacity = opacity of the drawing.

Note:
Clipping is supported.

4.1.3.151 CImg& draw_image (const CImg< t > & *sprite*, const int *x0* = 0, const int *y0* = 0, const int *z0* = 0, const int *v0* = 0, const float *opacity* = 1)

Draw a sprite image in the instance image, at coordinates (x0,y0,z0,v0).

Parameters:

sprite = sprite 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.

v0 = V-coordinate of the sprite position in the instance image.

opacity = opacity of the drawing.

Note:

Clipping is supported.

4.1.3.152 CImg& draw_image (const CImg< ti > & *sprite*, const CImg< tm > & *mask*, const int *x0* = 0, const int *y0* = 0, const int *z0* = 0, const int *v0* = 0, const tm *mask_valmax* = 1, const float *opacity* = 1)

Draw a masked sprite image in the instance image, at coordinates (x0,y0,z0,v0).

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.

v0 = V-coordinate of the sprite position in the instance image.

mask_valmax = Maximum pixel value of the mask image mask.

opacity = opacity of the drawing.

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.

4.1.3.153 CImg& draw_rectangle (const int *x0*, const int *y0*, const int *z0*, const int *v0*, const int *x1*, const int *y1*, const int *z1*, const int *v1*, const T & *val*, const float *opacity* = 1.0f)

Draw a 4D filled rectangle in the instance image, at coordinates (x0,y0,z0,v0)-(x1,y1,z1,v1).

Parameters:

x0 = X-coordinate of the upper-left rectangle corner in the instance image.

y0 = Y-coordinate of the upper-left rectangle corner in the instance image.

z0 = Z-coordinate of the upper-left rectangle corner in the instance image.

v0 = V-coordinate of the upper-left rectangle corner in the instance image.

x1 = X-coordinate of the lower-right rectangle corner in the instance image.

y1 = Y-coordinate of the lower-right rectangle corner in the instance image.

z1 = Z-coordinate of the lower-right rectangle corner in the instance image.

v1 = V-coordinate of the lower-right rectangle corner in the instance image.

val = scalar value used to fill the rectangle area.

opacity = opacity of the drawing.

Note:
Clipping is supported.

4.1.3.154 CImg& draw_rectangle (const int *x0*, const int *y0*, const int *z0*, const int *x1*, const int *y1*, const int *z1*, const T *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 in the instance image.

y0 = Y-coordinate of the upper-left rectangle corner in the instance image.

z0 = Z-coordinate of the upper-left rectangle corner in the instance image.

x1 = X-coordinate of the lower-right rectangle corner in the instance image.

y1 = Y-coordinate of the lower-right rectangle corner in the instance image.

z1 = Z-coordinate of the lower-right rectangle corner in the instance image.

color = array of **dimv**()(p. 59) values of type T, defining the drawing color.

opacity = opacity of the drawing.

Note:
Clipping is supported.

4.1.3.155 CImg& draw_rectangle (const int *x0*, const int *y0*, const int *x1*, const int *y1*, const T *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 in the instance image.

y0 = Y-coordinate of the upper-left rectangle corner in the instance image.

x1 = X-coordinate of the lower-right rectangle corner in the instance image.

y1 = Y-coordinate of the lower-right rectangle corner in the instance image.

color = array of **dimv**()(p. 59) values of type T, defining the drawing color.

opacity = opacity of the drawing.

Note:
Clipping is supported.

4.1.3.156 CImg& draw_triangle (const int *x0*, const int *y0*, const int *x1*, const int *y1*, const int *x2*, const int *y2*, const T *const *color*, const float *opacity* = 1, const float *brightness* = 1)

Draw a 2D filled colored triangle in the instance image, at coordinates (x0,y0)-(x1,y1)-(x2,y2).

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 **dimv**()(p. 59) values of type T, defining the drawing color.

opacity = opacity of the drawing (<1)

brightness = brightness of the drawing (in [0,1])

Note:

Clipping is supported.

4.1.3.157 CImg& draw_triangle (const int *x0*, const int *y0*, const int *x1*, const int *y1*, const int *x2*, const int *y2*, const T *const *color*, const float *c0*, const float *c1*, const float *c2*, const float *opacity* = 1)

Draw a 2D Gouraud-filled triangle in the instance image, at coordinates (x0,y0)-(x1,y1)-(x2,y2).

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 **dimv**()(p. 59) values of type T, defining the global drawing color.

c0 = brightness of the first corner.

c1 = brightness of the second corner.

c2 = brightness of the third corner.

opacity = opacity of the drawing.

Note:

Clipping is supported.

4.1.3.158 CImg& draw_triangle (const int *x0*, const int *y0*, const int *x1*, const int *y1*, const int *x2*, const int *y2*, const T *const *color*, const CImg< t > & *light*, const int *lx0*, const int *ly0*, const int *lx1*, const int *ly1*, const int *lx2*, const int *ly2*, const float *opacity* = 1.0f)

Draw a 2D phong-shaded triangle in the instance image, at coordinates (x0,y0)-(x1,y1)-(x2,y2).

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 **dimv**()(p. 59) 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.

4.1.3.159 CImg& draw_triangle (const int *x0*, const int *y0*, const int *x1*, const int *y1*, const int *x2*, const int *y2*, const CImg< t > & *texture*, const int *tx0*, const int *ty0*, const int *tx1*, const int *ty1*, const int *tx2*, const int *ty2*, const float *opacity* = 1.0f, const float *brightness* = 1.0f)

Draw a 2D textured triangle in the instance image, at coordinates (x0,y0)-(x1,y1)-(x2,y2).

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.

Note:
Clipping is supported, but texture coordinates do not support clipping.

4.1.3.160 CImg& draw_triangle (const int *x0*, const int *y0*, const int *x1*, const int *y1*, const int *x2*, const int *y2*, const CImg< t > & *texture*, const int *tx0*, const int *ty0*, const int *tx1*, const int *ty1*, const int *tx2*, const int *ty2*, const float *c0*, const float *c1*, const float *c2*, const float *opacity* = 1)

Draw a 2D textured triangle with Gouraud-Shading in the instance image, at coordinates (x0,y0)-(x1,y1)-(x2,y2).

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.

c0 = brightness value of the first corner.

c1 = brightness value of the second corner.

c2 = brightness value of the third corner.

opacity = opacity of the drawing.

Note:
Clipping is supported, but texture coordinates do not support clipping.

4.1.3.161 CImg& draw_triangle (const int *x0*, const int *y0*, const int *x1*, const int *y1*, const int *x2*, const int *y2*, const CImg< t > & *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* = 1.0f)

Draw a phong-shaded 2D textured triangle in the instance image, at coordinates (x0,y0)-(x1,y1)-(x2,y2).

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.

4.1.3.162 CImg& draw_ellipse (const int *x0*, const int *y0*, const float *r1*, const float *r2*, const float *ru*, const float *rv*, const T *const *color*, const unsigned int *pattern* = 0L, const float *opacity* = 1)

Draw an ellipse on the instance image.

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 **dimv**()(p. 59) 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.

4.1.3.163 CImg& draw_ellipse (const int *x0*, const int *y0*, const CImg< t > & *tensor*, const T * *color*, const unsigned int *pattern* = 0L, const float *opacity* = 1)

Draw an ellipse on the instance image.

Parameters:

x0 = X-coordinate of the ellipse center.

y0 = Y-coordinate of the ellipse center.

tensor = Diffusion tensor describing the ellipse.

color = array of **dimv**()(p. 59) 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.

4.1.3.164 CImg& draw_circle (const int *x0*, const int *y0*, float *r*, const T *const *color*, const unsigned int *pattern* = 0L, const float *opacity* = 1)

Draw a circle on the instance image.

Parameters:

x0 = X-coordinate of the circle center.

y0 = Y-coordinate of the circle center.

r = radius of the circle.

color = an array of **dimv**()(p. 59) values of type T, defining the drawing color.

pattern = If zero, the circle is filled, else pattern is an integer whose bits describe the outline pattern.

opacity = opacity of the drawing.

4.1.3.165 CImg& draw_text (const char *const *text*, const int *x0*, const int *y0*, const T *const *fgcolor*, const T *const *bgcolor*, const CImgl< t > & *font*, const float *opacity* = 1)

Draw a text into the instance image.

Parameters:

text = a C-string containing the text to display.

x0 = X-coordinate of the text in the instance image.

y0 = Y-coordinate of the text in the instance image.

fgcolor = an array of **dimv**()(p. 59) values of type T, defining the foreground color (0 means 'transparent').

bgcolor = an array of **dimv**()(p. 59) values of type T, defining the background color (0 means 'transparent').

font = List of font characters used for the drawing.

opacity = opacity of the drawing.

Note:

Clipping is supported.

See also:

get_font().

4.1.3.166 CImg& draw_text (const char *const *text*, const int *x0*, const int *y0*, const T *const *fgcolor*, const T *const *bgcolor* = 0, const unsigned int *font_size* = 11, const float *opacity* = 1.0f)

Draw a text into the instance image.

Parameters:

text = a C-string containing the text to display.

x0 = X-coordinate of the text in the instance image.

y0 = Y-coordinate of the text in the instance image.

fgcolor = an array of **dimv()**(p. 59) values of type T, defining the foreground color (0 means 'transparent').

bgcolor = an array of **dimv()**(p. 59) values of type T, defining the background color (0 means 'transparent').

font_size = Height of the desired font (11,13,24,38 or 57)

opacity = opacity of the drawing.

Note:

Clipping is supported.

See also:

get_font().

4.1.3.167 CImg& draw_text (const int *x0*, const int *y0*, const T *const *fgcolor*, const T *const *bgcolor*, const unsigned int *font_size*, const float *opacity*, const char * *format*, ...)

Draw a text into the instance image.

Parameters:

x0 = X-coordinate of the text in the instance image.

y0 = Y-coordinate of the text in the instance image.

fgcolor = an array of **dimv()**(p. 59) values of type T, defining the foreground color (0 means 'transparent').

bgcolor = an array of **dimv()**(p. 59) values of type T, defining the background color (0 means 'transparent').

opacity = opacity of the drawing.

format = a 'printf'-style format, followed by arguments.

Note:

Clipping is supported.

4.1.3.168 CImg& draw_quiver (const CImg< t > & *flow*, const T *const *color*, const unsigned int *sampling* = 25, const float *factor* = -20, const int *quiver_type* = 0, const float *opacity* = 1)

Draw a vector field in the instance image.

Parameters:

flow = a 2d image of 2d vectors used as input data.

color = an array of **dimv()**(p. 59) values of type T, defining the drawing color.

sampling = length (in pixels) between each arrow.

factor = length factor of each arrow (if <0, computed as a percentage of the maximum length).

quiver_type = type of plot. Can be 0 (arrows) or 1 (segments).

opacity = opacity of the drawing.

Note:
Clipping is supported.

4.1.3.169 CImg& draw_quiver (const CImg< t1 > & *flow*, const CImg< t2 > & *color*, const unsigned int *sampling* = 25, const float *factor* = -20, const int *quiver_type* = 0, const float *opacity* = 1)

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

Parameters:
flow = a 2d image of 2d vectors used as input data.

color = a 2d image of **dimv()**(p. 59)-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).

quiver_type = type of plot. Can be 0 (arrows) or 1 (segments).

opacity = opacity of the drawing.

Note:
Clipping is supported.

4.1.3.170 CImg& draw_graph (const CImg< t > & *data*, const T *const *color*, const unsigned int *gtype* = 0, const double *ymin* = 0, const double *ymax* = 0, const float *opacity* = 1)

Draw a 1D graph on the instance image.

Parameters:
data = an image containing the graph values I = f(x).

color = an array of **dimv()**(p. 59) values of type T, defining the drawing color.

gtype = define the type of the plot :

- 0 = Plot using linear interpolation (segments).
- 1 = Plot with bars.
- 2 = Plot using cubic interpolation (3-polynomials).

ymin = lower bound of the y-range.

ymax = upper bound of the y-range.

opacity = opacity of the drawing.

Note:
- if ymin==ymax==0, the y-range is computed automatically from the input sample.

See also:
draw_axe()(p. 103).

4.1.3.171 CImg& draw_axe (const CImg< t > & *xvalues*, const int *y*, const T *const *color*, const int *precision* = -1, const float *opacity* = 1.0f)

Draw a labelled horizontal axis on the instance image.

Parameters:

x0 = lower bound of the x-range.

x1 = upper bound of the x-range.

y = Y-coordinate of the horizontal axis in the instance image.

color = an array of **dimv()**(p. 59) values of type `T`, defining the drawing color.

precision = precision of the labels.

opacity = opacity of the drawing.

Note:

if `precision==0`, precision of the labels is automatically computed.

See also:

draw_graph()(p. 102).

4.1.3.172 CImg& draw_fill (const int *x*, const int *y*, const int *z*, const T *const *color*, CImg< t > & *region*, const float *sigma* = 0, const float *opacity* = 1)

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 **dimv()**(p. 59) 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.

Returns:

`region` is initialized with the binary mask of the filled region.

4.1.3.173 CImg& draw_fill (const int *x*, const int *y*, const int *z*, const T *const *color*, const float *sigma* = 0, const float *opacity* = 1)

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 **dimv()**(p. 59) values of type `T`, defining the drawing color.

sigma = tolerance concerning neighborhood values.

opacity = opacity of the drawing.

4.1.3.174 CImg& draw_fill (const int *x*, const int *y*, const T *const *color*, const float *sigma* = 0, const float *opacity* = 1)

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 **dimv**()(p. 59) values of type T, defining the drawing color.

sigma = tolerance concerning neighborhood values.

opacity = opacity of the drawing.

4.1.3.175 CImg& draw_plasma (const int *x0*, const int *y0*, const int *x1*, const int *y1*, const double *alpha* = 1.0, const double *beta* = 1.0, const float *opacity* = 1)

Draw a plasma square in the instance image.

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.

4.1.3.176 CImg& draw_plasma (const double *alpha* = 1.0, const double *beta* = 1.0, const float *opacity* = 1)

Draw a plasma in the instance image.

Parameters:

alpha = Alpha-parameter of the plasma.

beta = Beta-parameter of the plasma.

opacity = opacity of the drawing.

4.1.3.177 CImg& draw_gaussian (const float *xc*, const double *sigma*, const T *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 **dimv**()(p. 59) values of type T, defining the drawing color.

opacity = opacity of the drawing.

4.1.3.178 CImg& draw_gaussian (const float *xc*, const float *yc*, const CImg< t > & *tensor*, const T *const *color*, const float *opacity* = 1)

Draw an anisotropic 2D gaussian function in the instance image.

Parameters:

xc = X-coordinate of the gaussian center.

yc = Y-coordinate of the gaussian center.

tensor = 2x2 covariance matrix.

color = array of **dimv**()(p. 59) values of type `T`, defining the drawing color.

opacity = opacity of the drawing.

4.1.3.179 CImg& draw_gaussian (const float *xc*, const float *yc*, const float *sigma*, const T *const *color*, const float *opacity* = 1)

Draw an isotropic 2D gaussian function in the instance image.

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 **dimv**()(p. 59) values of type `T`, defining the drawing color.

opacity = opacity of the drawing.

4.1.3.180 CImg& draw_gaussian (const float *xc*, const float *yc*, const float *zc*, const CImg< t > & *tensor*, const T *const *color*, const float *opacity* = 1)

Draw an anisotropic 3D gaussian function in the instance image.

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 **dimv**()(p. 59) values of type `T`, defining the drawing color.

opacity = opacity of the drawing.

4.1.3.181 CImg& draw_gaussian (const float *xc*, const float *yc*, const float *zc*, const double *sigma*, const T *const *color*, const float *opacity* = 1)

Draw an isotropic 3D gaussian function in the instance image.

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 **dimv**()(p. 59) values of type `T`, defining the drawing color.

opacity = opacity of the drawing.

4.1.3.182 CImg& draw_object3d (const float *X*, const float *Y*, const float *Z*, const CImg< tp > & *points*, const CImgl< tf > & *primitives*, const CImgl< T > & *colors*, const CImgl< 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* = -5000, const float *ambiant_light* = 0.05f)

Draw a 3D object in the instance image.

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

points = Image N∗3 describing 3D point coordinates

primitives = List of P primitives

colors = List of P color (or textures)

opacities = Image 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

lightx = X-coordinate of the light

lighty = Y-coordinate of the light

lightz = Z-coordinate of the light

ambiant_light = Brightness (between 0..1) of the ambiant light

4.1.3.183 CImg<typename cimg::largest<T,t>::type> get_correlate (const CImg< t > & *mask*, const unsigned int *cond* = 1, const bool *weighted_correl* = false) const

Return the correlation of the image by a mask.

The result res of the correlation 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.

cond = the border condition type (0=zero, 1=dirichlet)

weighted_correl = enable local normalization.

4.1.3.184 CImg& correlate (const CImg< t > & *mask*, const unsigned int *cond* = 1, const bool *weighted_correl* = false)

Correlate the image by a mask.

This is the in-place version of get_correlate.

See also:

get_correlate(p. 106)

4.1.3.185 CImg<typename cimg::largest<T,t>::type> get_convolve (const CImg< t > & *mask*, const unsigned int *cond* = 1, const bool *weighted_convol* = false) const

Return 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.

cond = the border condition type (0=zero, 1=dirichlet)

weighted_convol = enable local normalization.

4.1.3.186 CImg& convolve (const CImg< t > & *mask*, const unsigned int *cond* = 1, const bool *weighted_convol* = false)

Convolve the image by a mask.

This is the in-place version of **get_convolve()**(p. 107).

See also:

get_convolve()(p. 107)

4.1.3.187 CImg& erode (const CImg< t > & *mask*, const unsigned int *cond* = 1, const bool *weighted_erosion* = false)

Erode the image by a structuring element.

This is the in-place version of **get_erode()**(p. 43).

See also:

get_erode()(p. 43)

4.1.3.188 CImg& dilate (const CImg< t > & *mask*, const unsigned int *cond* = 1, const bool *weighted_dilatation* = false)

Dilate the image by a structuring element.

This is the in-place version of **get_dilate()**(p. 43).

See also:

get_dilate()(p. 43)

4.1.3.189 CImg& noise (const double *sigma* = -20, const unsigned int *ntype* = 0)

Add noise to the image.

This is the in-place version of get_noise.

See also:

get_noise(p. 108).

4.1.3.190 CImg get_noise (const double *sigma* = -20, const unsigned int *ntype* = 0) const

Return a noisy image.

Parameters:

sigma = power of the noise. if sigma<0, it corresponds to the percentage of the maximum image value.

ntype = noise type. can be 0=gaussian, 1=uniform or 2=Salt and Pepper.

Returns:

A noisy version of the instance image.

4.1.3.191 CImg& deriche (const float *sigma* = 1, const int *order* = 0, const char *axe* = 'x', const unsigned int *cond* = 1)

Apply a deriche filter on the image.

This is the in-place version of get_deriche

See also:

get_deriche(p. 108).

4.1.3.192 CImg get_deriche (const float *sigma* = 1, const int *order* = 0, const char *axe* = 'x', const unsigned int *cond* = 1) const

Return 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.

See also:

blur(p. 108)

4.1.3.193 CImg& blur (const float *sigmax*, const float *sigmay*, const float *sigmaz*, const unsigned int *cond* = 1)

Blur the image with a Deriche filter (anisotropically).

This is the in-place version of **get_blur()**(p. 108).

See also:

get_blur()(p. 108).

4.1.3.194 CImg& blur (const float *sigma*, const unsigned int *cond* = 1)

Blur the image with a Canny-Deriche filter.

This is the in-place version of **get_blur()**(p. 108).

4.1.3.195 CImg get_blur (const float *sigmax*, const float *sigmay*, const float *sigmaz*, const unsigned int *cond* = 1) const

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).

4.1.3.196 CImg& blur_anisotropic (const CImg< t > & *G*, const float *amplitude* = 30.0f, const float *dl* = 0.8f, const float *da* = 30.0f, const float *gauss_prec* = 2.0f, const unsigned int *interpolation* = 0, const bool *fast_approx* = true)

Blur an image following a field of diffusion tensors.

This is the in-place version of **get_blur_anisotropic()**(p. 109).

4.1.3.197 CImg get_blur_anisotropic (const CImg< t > & *G*, const float *amplitude* = 30.0f, const float *dl* = 0.8f, const float *da* = 30.0f, const float *gauss_prec* = 2.0f, const unsigned int *interpolation* = 0, const bool *fast_approx* = true) const

Get a blurred version of an image following a field of diffusion tensors.

Parameters:

G = Field of square roots of diffusion tensors 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.

4.1.3.198 CImg get_blur_anisotropic (const float *amplitude*, const float *sharpness* = 0.8f, const float *anisotropy* = 0.3f, const float *alpha* = 1.2f, const float *sigma* = 0.8f, const float *dl* = 0.8f, const float *da* = 30.0f, const float *gauss_prec* = 2.0f, const unsigned int *interpolation* = 0, const bool *fast_approx* = true) const

Blur an image in an anisotropic way.

Parameters:

amplitude = amplitude of the anisotropic blur.

sharpness = define the contour preservation.

anisotropy = define the smoothing anisotropy.

alpha = image pre-blurring (gaussian).

sigma = regularity of the tensor-valued geometry.

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

4.1.3.199 const CImg& display (const int *min_size* = 128, const int *max_size* = 1024) const

Display an image in a window, with a default title. See also.

See also:

display()(p. 49) for details on parameters.

4.1.3.200 static CImg get_load (const char * *filename*) [static]

Load an image from a file.

Parameters:
filename = name of the image file to load.

Returns:
A CImg<T> instance containing the pixel data defined in the image file.

Note:
The extension of `filename` defines the file format. If no filename extension is provided, **CImg<T>::get_load()**(p. 110) will try to load a CRAW file (**CImg**(p. 21) Raw file).

4.1.3.201 CImg& load (const char * *filename*)

Load an image from a file.

This is the in-place version of **get_load()**(p. 110).

4.1.3.202 CImg& load_ascii (const char * *filename*)

Load an image from an ASCII file (in-place version).

This is the in-place version of **get_load_ascii()**(p. 53).

4.1.3.203 CImg& load_dlm (const char * *filename*)

Load an image from a DLM file (in-place version).

This is the in-place version of **get_load_dlm()**(p. 53).

4.1.3.204 const CImg& save (const char * *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.

If `normalize` is true, a normalized version of the image (between [0,255]) is saved.

4.1.3.205 const CImg& save_convert (const char * *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**(p. 21), 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`).

4.1.3.206 const CImg& save_png (const char * *filename*) const

Save an image to a PNG file.

Parameters:
filename = name of the png image file to load

Returns:
*this

Note:
The png format specifies a variety of possible data formats. Grey scale, Grey scale with Alpha, RGB color, RGB color with Alpha, and Palletized color are supported. Per channel bit depths of 1, 2, 4, 8, and 16 are natively supported. The type of file saved depends on the number of channels in the **CImg**(p. 21) file. If there is 4 or more channels, the image will be saved as an RGB color with Alpha image using the bottom 4 channels. If there are 3 channels, the saved image will be an RGB color image. If 2 channels then the image saved will be Grey scale with Alpha, and if 1 channel will be saved as a Grey scale image.

4.1.4 Member Data Documentation

4.1.4.1 unsigned int width

Variable representing the width of the instance image (i.e. dimensions along the X-axis).

Remarks:
- Prefer using the function **CImg<T>::dimx()**(p. 58) to get information about the width of an image.
- Use function **CImg<T>::resize()**(p. 82) to set a new width for an image. Setting directly the variable `width` would probably result in a library crash.
- Empty images have `width` defined to `0`.

4.1.4.2 unsigned int height

Variable representing the height of the instance image (i.e. dimensions along the Y-axis).

Remarks:
- Prefer using the function **CImg<T>::dimy()**(p. 58) to get information about the height of an image.
- Use function **CImg<T>::resize()**(p. 82) to set a new height for an image. Setting directly the variable `height` would probably result in a library crash.
- 1D signals have `height` defined to `1`.
- Empty images have `height` defined to `0`.

4.1.4.3 unsigned int depth

Variable representing the depth of the instance image (i.e. dimensions along the Z-axis).

Remarks:
- Prefer using the function **CImg<T>::dimz()**(p. 58) to get information about the depth of an image.

- Use function **CImg<T>::resize()**(p. 82) to set a new depth for an image. Setting directly the variable `depth` would probably result in a library crash.
- Classical 2D images have `depth` defined to `1`.
- Empty images have `depth` defined to `0`.

4.1.4.4 unsigned int dim

Variable representing the number of channels of the instance image (i.e. dimensions along the V-axis).

Remarks:

- Prefer using the function **CImg<T>::dimv()**(p. 59) to get information about the depth of an image.
- Use function **CImg<T>::resize()**(p. 82) to set a new vector dimension for an image. Setting directly the variable `dim` would probably result in a library crash.
- Scalar-valued images (one value per pixel) have `dim` defined to `1`.
- Empty images have `depth` defined to `0`.

4.2 CImgDisplay Struct Reference

This class represents a window which can display **CImg**(p. 21)<T> images and handles mouse and keyboard events.

Public Member Functions

- **CImgDisplay** (const unsigned int dimw, const unsigned int dimh, const char *__title__=0, const unsigned int normalization_type=1, const unsigned int events_type=3, const bool fullscreen_flag=false, const bool closed_flag=false)

  Create a display window with a specified size `pwidth` *x* `height`.

- template<typename T> **CImgDisplay** (const **CImg**< T > &img, const char *__title__=0, const unsigned int normalization_type=1, const unsigned int events_type=3, const bool fullscreen_flag=false, const bool closed_flag=false)

  Create a display window from an image.

- template<typename T> **CImgDisplay** (const **CImgl**< T > &list, const char *__title__=0, const unsigned int normalization_type=1, const unsigned int events_type=3, const bool fullscreen_flag=false, const bool closed_flag=false)

  Create a display window from an image list.

- **CImgDisplay** (const **CImgDisplay** &disp)

  Create a display window by copying another one.

- ~**CImgDisplay** ()

  Destructor.

- **CImgDisplay** & **operator=** (const **CImgDisplay** &disp)

  Assignement operator.

- int **dimx** () const

  Return display width.

- int **dimy** () const

  Return display height.

- int **window_dimx** () const

  Return display window width.

- int **window_dimy** () const

  Return display window height.

- int **window_posx** () const

  Return X-coordinate of the window.

- int **window_posy** () const

  Return Y-coordinate of the window.

- **CImgDisplay** & **wait** (const unsigned int milliseconds)

  Synchronized waiting function. Same as cimg::wait().

- **CImgDisplay** & **wait** ()

  Wait for an event.

- float **frames_per_second** ()

  Return the frame per second rate.

- template<typename T> **CImgDisplay** & **display** (const **CImgl**< T > &list, const char axe='x', const char align='c')

  Display an image list CImgl<T> into a display window.

- 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 bool redraw=true)

  Resize a display window in its current size.

- template<typename tp, typename tf, typename T, typename to> **CImgDisplay** & **display_object3d** (const tp &points, const **CImgl**< tf > &primitives, const **CImgl**< T > &colors, const to &opacities, const bool centering=true, const int render_static=4, const int render_motion=1, const bool double_sided=false, const float focale=500.0f, const float ambiant_light=0.05f, const bool display_axes=true, float *const pose_matrix=0)

  Display a 3d object.

- template<typename tp, typename tf, typename T> **CImgDisplay** & **display_object3d** (const tp &points, const **CImgl**< tf > &primitives, const **CImgl**< T > &colors, const bool centering=true, const int render_static=4, const int render_motion=1, const bool double_sided=false, const float focale=500.0f, const float ambiant_light=0.05f, const float opacity=1.0f, const bool display_axes=true, float *const pose_matrix=0)

  Display a 3D object.

- **CImgDisplay** & **toggle_fullscreen** ()

  Toggle fullscreen mode.

- **CImgDisplay** & **assign** (const unsigned int dimw, const unsigned int dimh, const char ***title**=0, const unsigned int normalization_type=1, const unsigned int events_type=3, const bool fullscreen_flag=false, const bool closed_flag=false)

  In-place version of the previous constructor.

- template<typename T> **CImgDisplay** & **assign** (const **CImg**< T > &img, const char ***title**=0, const unsigned int normalization_type=1, const unsigned int events_type=3, const bool fullscreen_flag=false, const bool closed_flag=false)

  In-place version of the previous constructor.

- template<typename T> **CImgDisplay** & **assign** (const **CImgl**< T > &list, const char ***title**=0, const unsigned int normalization_type=1, const unsigned int events_type=3, const bool fullscreen_flag=false, const bool closed_flag=false)

  In-place version of the previous constructor.

- **CImgDisplay** & **assign** (const **CImgDisplay** &disp)

  In-place version of the previous constructor.

- template<typename T> **CImgDisplay** & **display** (const **CImg**< T > &img)

  Display an image in a window.

- **CImgDisplay** & **resize** (const int **width**, const int **height**, const bool redraw=true)

  Resize window.

- **CImgDisplay** & **move** (const int posx, const int posy)

  Move window.

- **CImgDisplay** & **show** ()

  Show a closed display.

- **CImgDisplay** & **close** ()

  Close a visible display.

- **CImgDisplay** & **set_title** (const char *format,...)

  Set the window title.

- **CImgDisplay** & **paint** ()

  Re-paint image content in window.

- template<typename T> **CImgDisplay** & **render** (const **CImg**< T > &img)

  Render image buffer into GDI native image format.

Static Public Member Functions

- static int **screen_dimx** ()

  Return the width of the screen resolution.

- static int **screen_dimy** ()

  Return the height of the screen resolution.

- static void **wait_all** ()

  Wait for a window event in any **CImg**(p. 21) *window.*

Public Attributes

- unsigned int **width**

  Width of the display.

- unsigned int **height**

  Height of the display.

- unsigned int **normalization**

  Normalization type used for the display.

- unsigned int **events**

  Range of events detected by the display.

- bool **is_fullscreen**

  Fullscreen state of the display.

- char ∗ **title**

  Display title.

- volatile int **window_x**

  X-pos of the display on the screen.

- volatile int **window_y**

  Y-pos of the display on the screen.

- volatile unsigned int **window_width**

  Width of the underlying window.

- volatile unsigned int **window_height**

  Height of the underlying window.

- volatile int **mouse_x**

  X-coordinate of the mouse pointer on the display.

- volatile int **mouse_y**

  Y-coordinate of the mouse pointer on the display.

- volatile unsigned int **button**

  Button state of the mouse.

- volatile int **wheel**

  Wheel state of the mouse (Linux only).

- volatile unsigned int **key**

  Key value if pressed.

- volatile bool **is_closed**

  Closed state of the window.

- volatile bool **is_resized**

  Resized state of the window.

- volatile bool **is_moved**

  Moved state of the window.

- volatile bool **is_event**

  Event state of the window.

4.2.1 Detailed Description

This class represents a window which can display **CImg**(p. 21)<T> images and handles mouse and keyboard events.

Creating a **CImgDisplay**(p. 112) instance opens a window that can be used to display a `CImg<T>` image of a `CImgl<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**(p. 112) data fields. See **Using Display Windows.**(p. 13) for a complete tutorial on using the **CImgDisplay**(p. 112) class.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 CImgDisplay (const unsigned int *dimw*, const unsigned int *dimh*, const char * *title* = `0`, const unsigned int *normalization_type* = `1`, const unsigned int *events_type* = `3`, const bool *fullscreen_flag* = `false`, const bool *closed_flag* = `false`)

Create a display window with a specified size `pwidth` x `height`.

Parameters:

dimw : Width of the display window.

dimh : Height of the display window.

title : Title of the display window.

normalization_type : Normalization type of the display window (see CImgDisplay::normalize).

events_type : Type of events handled by the display window.

fullscreen_flag : Fullscreen mode.

closed_flag : Initially visible mode. A black image will be initially displayed in the display window.

4.2.2.2 CImgDisplay (const CImg< T > & *img*, const char * *title* = 0, const unsigned int *normalization_type* = 1, const unsigned int *events_type* = 3, const bool *fullscreen_flag* = false, const bool *closed_flag* = false)

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_type : Normalization type of the display window.

events_type : Type of events handled by the display window.

fullscreen_flag : Fullscreen mode.

closed_flag : Initially visible mode.

4.2.2.3 CImgDisplay (const CImgl< T > & *list*, const char * *title* = 0, const unsigned int *normalization_type* = 1, const unsigned int *events_type* = 3, const bool *fullscreen_flag* = false, const bool *closed_flag* = false)

Create a display window from an image list.

Parameters:

list : The list of images to display.

title : Title of the display window

normalization_type : Normalization type of the display window.

events_type : Type of events handled by the display window.

fullscreen_flag : Fullscreen mode.

closed_flag : Initially visible mode.

4.2.2.4 CImgDisplay (const CImgDisplay & *disp*)

Create a display window by copying another one.

Parameters:

win : Display window to copy.

title : Title of the new display window.

4.2.3 Member Function Documentation

4.2.3.1 CImgDisplay& wait (const unsigned int *milliseconds*)

Synchronized waiting function. Same as cimg::wait().

See also:

cimg::wait()

4.2.3.2 CImgDisplay& display (const CImgl< T > & *list*, const char *axe* = 'x', const char *align* = 'c')

Display an image list CImgl<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.

axe : The axe used to append the image for visualization. Can be 'x' (default),'y','z' or 'v'.

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).

See also:

CImg::get_append()(p. 35)

4.2.3.3 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.

See also:

CImgDisplay::is_resized(p. 116), CImgDisplay::resizedimx(), CImgDisplay::resizedimy()

4.3 CImgException Struct Reference

Class which is thrown when an error occured during a CImg library function call.

Public Attributes

- char **message** [1024]

  Message associated with the error that thrown the exception.

4.3.1 Detailed Description

Class which is thrown when an error occured during a CImg library function call.

4.3.2 Overview

CImgException(p. 118) 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**(p. 118) 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 (invalid argument to oper
```

- **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 doesn't exist.
```

- **CImgDisplayException** : Thrown when an error occured when trying to display an image in a window. This exception is thrown when image display request cannot be satisfied.

The parent class **CImgException**(p. 118) 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**, **CImgIOException** and **CImgDisplayException** are simple subclasses of **CImgException**(p. 118) and are thus not detailled more in this reference documentation.

4.3.3 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_debug` to 0 before including the CImg header file. The example below shows how to cleanly handle CImg Library exceptions :

```
#define cimg_debug 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.message);  // Display your own error message
    ...                                                        // Do what you want now.
  }
}
```

4.4 CImgl Struct Template Reference

Class representing list of images CImg<T>.

Constructors - Destructor - Copy

- **CImgl** ()

  Default constructor.

- ~**CImgl** ()

  Destructor.

- **CImgl** & **assign** ()

  In-place version of the default constructor and default destructor.

- **CImgl** & **clear** ()

  Equivalent to **assign**()(p. 120) *(STL-compliant name).*

- template<typename t> **CImgl** (const **CImgl**< t > &list)

  Copy constructor.

- template<typename t> **CImgl** (const **CImgl**< t > &list, const bool shared)

  Copy constructor that create a shared object.

- template<typename t> **CImgl** & **assign** (const **CImgl**< t > &list, const bool shared=false)

  In-place version of the copy constructor.

- template<typename t> **CImgl** & **operator=** (const **CImgl**< t > &list)

  Assignement operator.

- **CImgl** (const unsigned int n)

  Construct an image list containing n empty images.

- **CImgl** & **assign** (const unsigned int n)

  In-place version of the previous constructor.

- **CImgl** (const unsigned int n, const unsigned int width, const unsigned int height=1, const unsigned int depth=1, const unsigned int dim=1)

  Construct an image list containing n images with specified size.

- **CImgl** & **assign** (const unsigned int n, const unsigned int width, const unsigned int height=1, const unsigned int depth=1, const unsigned int dim=1)

  In-place version of the previous constructor.

- **CImgl** (const unsigned int n, const unsigned int width, const unsigned int height, const unsigned int depth, const unsigned int dim, const T &val)

  Construct an image list containing n images with specified size, filled with val.

- **CImgl** & **assign** (const unsigned int n, const unsigned int width, const unsigned int height, const unsigned int depth, const unsigned int dim, const T &val)

  In-place version of the previous constructor.

- template<typename t> **CImgl** (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> **CImgl** & **assign** (const unsigned int n, const **CImg**< t > &img, const bool shared=false)

  In-place version of the previous constructor.

- **CImgl** (const **CImg**< T > &img, const bool shared=false)

  Construct an image list from one image.

- **CImgl** & **assign** (const **CImg**< T > &img, const bool shared=false)

  In-place version of the previous constructor.

- **CImgl** (const **CImg**< T > &img1, const **CImg**< T > &img2, const bool shared=false)

  Construct an image list from two images.

- **CImgl** & **assign** (const **CImg**< T > &img1, const **CImg**< T > &img2, const bool shared=false)

  In-place version of the previous constructor.

- **CImgl** (const **CImg**< T > &img1, const **CImg**< T > &img2, const **CImg**< T > &img3, const bool shared=false)

  Construct an image list from three images.

- **CImgl** & **assign** (const **CImg**< T > &img1, const **CImg**< T > &img2, const **CImg**< T > &img3, const bool shared=false)

  In-place version of the previous constructor.

- **CImgl** (const **CImg**< T > &img1, const **CImg**< T > &img2, const **CImg**< T > &img3, const **CImg**< T > &img4, const bool shared=false)

  Construct an image list from four images.

- **CImgl** & **assign** (const **CImg**< T > &img1, const **CImg**< T > &img2, const **CImg**< T > &img3, const **CImg**< T > &img4, const bool shared=false)

  In-place version of the previous constructor.

- **CImgl** (const **CImg**< T > &img1, const **CImg**< T > &img2, const **CImg**< T > &img3, const **CImg**< T > &img4, const **CImg**< T > &img5, const bool shared=false)

  Construct an image list from five images.

- **CImgl** & **assign** (const **CImg**< T > &img1, const **CImg**< T > &img2, const **CImg**< T > &img3, const **CImg**< T > &img4, const **CImg**< T > &img5, const bool shared=false)

  In-place version of the previous constructor.

- **CImgl** (const **CImg**< T > &img1, const **CImg**< T > &img2, const **CImg**< T > &img3, const **CImg**< T > &img4, const **CImg**< T > &img5, const **CImg**< T > &img6, const bool shared=false)

  Construct an image list from six images.

- **CImgl** & **assign** (const **CImg**< T > &img1, const **CImg**< T > &img2, const **CImg**< T > &img3, const **CImg**< T > &img4, const **CImg**< T > &img5, const **CImg**< T > &img6, const bool shared=false)

  In-place version of the previous constructor.

- **CImgl** (const **CImg**< T > &img1, const **CImg**< T > &img2, const **CImg**< T > &img3, const **CImg**< T > &img4, const **CImg**< T > &img5, const **CImg**< T > &img6, const **CImg**< T > &img7, const bool shared=false)

  Construct an image list from seven images.

- **CImgl** & **assign** (const **CImg**< T > &img1, const **CImg**< T > &img2, const **CImg**< T > &img3, const **CImg**< T > &img4, const **CImg**< T > &img5, const **CImg**< T > &img6, const **CImg**< T > &img7, const bool shared=false)

  In-place version of the previous constructor.

- **CImgl** (const **CImg**< T > &img1, const **CImg**< T > &img2, const **CImg**< T > &img3, const **CImg**< T > &img4, const **CImg**< T > &img5, const **CImg**< T > &img6, const **CImg**< T > &img7, const **CImg**< T > &img8, const bool shared=false)

  Construct an image list from eight images.

- **CImgl** & **assign** (const **CImg**< T > &img1, const **CImg**< T > &img2, const **CImg**< T > &img3, const **CImg**< T > &img4, const **CImg**< T > &img5, const **CImg**< T > &img6, const **CImg**< T > &img7, const **CImg**< T > &img8, const bool shared=false)

  In-place version of the previous constructor.

- **CImgl** (const **CImg**< T > &img1, const **CImg**< T > &img2, const **CImg**< T > &img3, const **CImg**< T > &img4, const **CImg**< T > &img5, const **CImg**< T > &img6, const **CImg**< T > &img7, const **CImg**< T > &img8, const **CImg**< T > &img9, const bool shared=false)

  Construct an image list from nine images.

- **CImgl** & **assign** (const **CImg**< T > &img1, const **CImg**< T > &img2, const **CImg**< T > &img3, const **CImg**< T > &img4, const **CImg**< T > &img5, const **CImg**< T > &img6, const **CImg**< T > &img7, const **CImg**< T > &img8, const **CImg**< T > &img9, const bool shared=false)

  In-place version of the previous constructor.

- **CImgl** (const **CImg**< T > &img1, const **CImg**< T > &img2, const **CImg**< T > &img3, const **CImg**< T > &img4, const **CImg**< T > &img5, const **CImg**< T > &img6, const **CImg**< T > &img7, const **CImg**< T > &img8, const **CImg**< T > &img9, const **CImg**< T > &img10, const bool shared=false)

  Construct an image list from ten images.

- **CImgl** & **assign** (const **CImg**< T > &img1, const **CImg**< T > &img2, const **CImg**< T > &img3, const **CImg**< T > &img4, const **CImg**< T > &img5, const **CImg**< T > &img6, const **CImg**< T > &img7, const **CImg**< T > &img8, const **CImg**< T > &img9, const **CImg**< T > &img10, const bool shared=false)

  In-place version of the previous constructor.

- **CImgl** (const char ∗filename)

  Construct an image list from a filename.

- **CImgl** & **assign** (const char ∗filename)

  In-place version of the previous constructor.

Arithmetics Operators

- bool **is_empty** () const

  Return `true` *if list is empty.*

- template<typename t> **CImgl** & **operator+=** (const **CImgl**< t > &list)

  Add each image of the current list with the corresponding image in the list `list`.

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

  Add each image of the current list with an image `img`.

- **CImgl** & **operator+=** (const T &val)

  Add each image of the current list with a value `val`.

- template<typename t> **CImgl** & **operator-=** (const **CImgl**< t > &list)

  Subtract each image of the current list with the corresponding image in the list `list`.

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

  Substract each image of the current list with an image `img`.

- **CImgl** & **operator-=** (const T &val)

  Substract each image of the current list with a value `val`.

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

  Multiply each image of the current list by an image `img`.

- **CImgl** & **operator** ∗**=** (const double val)

  Multiply each image of the current list by a value `val`.

- **CImgl** & **operator/=** (const double val)

  Divide each image of the current list by a value `val`.

- template<typename t> **CImgl**< typename cimg::largest< t, T >::type > **operator+** (const **CImgl**< t > &list) const

  Return a new image list corresponding to the addition of each image of the current list with the corresponding image in the list `list`.

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

  Return a new image list corresponding to the addition of each image of the current list with an image `img`.

- **CImgl operator+** (const T &val) const

  Return a new image list corresponding to the addition of each image of the current list with a value `val`.

- template<typename t> **CImgl**< typename cimg::largest< t, T >::type > **operator-** (const **CImgl** &list) const

  Return a new image list corresponding to the substraction of each image of the current list with the corresponding image in the list `list`.

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

Return a new image list corresponding to the substraction of each image of the current list with an image `img`.

- **CImgl operator-** (const T &val) const

  Return a new image list corresponding to the substraction of each image of the current list with a value `val`.

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

  Return a new image list corresponding to the matrix multiplication of each image of the current list by the matrix `img`.

- **CImgl operator** ∗ (const double val) const

  Return a new image list corresponding to the multiplication of each image of the current list by a value `val`.

- **CImgl operator/** (const double val) const

  Return a new image list corresponding to the division of each image of the current list by a value `val`.

- static const char ∗ **pixel_type** ()

  Return a string describing the type of the image pixels in the list (template parameter `T`*).*

- template<typename t> **CImgl**< typename cimg::largest< T, t >::type > **operator+** (const T &val, const **CImgl**< t > &list)

  Return a new image list corresponding to the addition of each image of the current list with a value `val`*;.*

- **CImgl operator** ∗ (const double val, const **CImgl** &list)

  Return a new image list corresponding to the scalar multiplication of each image of the current list by a value `val`.

List Manipulation

- **CImg**< T > & **operator[ ]** (const unsigned int pos)

  Return a reference to the i-th element of the image list.

- **CImg**< T > & **operator()** (const unsigned int pos)

  Equivalent to CImgl<T>::operator[ ].

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

  Return a reference to (x,y,z,v) pixel of the pos-th image of the list.

- **CImg**< T > & **at** (const unsigned int pos)

  Equivalent to CImgl<T>::operator[ ], with boundary checking.

- **CImg**< T > & **back** ()

  Returns a reference to last element.

- **CImg**< T > & **front** ()

  Returns a reference to the first element.

- **iterator begin** ()

  Returns an iterator to the beginning of the vector.

- **iterator end** ()

  Returns an iterator just past the last element.

- **CImgl** & **insert** (const **CImg**< T > &img, const unsigned int pos, const bool shared=false)

  Insert a copy of the image `img` *into the current image list, at position* `pos`.

- **CImgl** & **insert** (const **CImg**< T > &img)

  Insert a copy of the image `img` *at the current image list.*

- **CImgl** & **insert** (const unsigned int n, const **CImg**< T > &img, const unsigned int pos)

  Insert n copies of the image `img` *into the current image list, at position* `pos`.

- **CImgl** & **insert** (const unsigned int n, const **CImg**< T > &img)

  Insert n copies of the image `img` *at the end of the list.*

- **CImgl** & **insert** (const **CImgl**< T > &list, const unsigned int pos)

  Insert a copy of the image list `list` *into the current image list, starting from position* `pos`.

- **CImgl** & **insert** (const **CImgl**< T > &list)

  Append a copy of the image list `list` *at the current image list.*

- **CImgl** & **insert** (const unsigned int n, const **CImgl**< T > &list, const unsigned int pos)

  Insert n copies of the list `list` *at position* `pos` *of the current list.*

- **CImgl** & **insert** (const unsigned int n, const **CImgl**< T > &list)

  Insert n copies of the list at the end of the current list.

- **CImgl** & **push_back** (const **CImg**< T > &img)

  Insert image `img` *at the end of the list.*

- **CImgl** & **push_front** (const **CImg**< T > &img)

  Insert image `img` *at the front of the list.*

- **CImgl** & **push_back** (const **CImgl**< T > &list)

  Insert list `list` *at the end of the current list.*

- **CImgl** & **push_front** (const **CImgl**< T > &list)

  Insert list `list` *at the front of the current list.*

- **CImgl** & **insert_shared** (const **CImg**< T > &img, const unsigned int pos)

  Insert a shared copy of the image `img` *into the current image list, at position* `pos`.

- **CImgl** & **insert_shared** (const **CImg**< T > &img)

  Insert a shared copy of the image `img` *at the current image list.*

- **CImgl** & **insert_shared** (const unsigned int n, const **CImg**< T > &img, const unsigned int pos)

  Insert n shared copies of the image `img` *into the current image list, at position* `pos`.

- **CImgl** & **insert_shared** (const unsigned int n, const **CImg**< T > &img)
  Insert n shared copies of the image `img` *at the end of the list.*
- **CImgl** & **insert_shared** (const **CImgl**< T > &list, const unsigned int pos)
  Insert a shared copy of all image of the list `list` *into the current image list, starting from position* `pos`.
- **CImgl** & **insert_shared** (const **CImgl**< T > &list)
  Append a shared copy of the image list `list` *at the current image list.*
- **CImgl** & **insert_shared** (const unsigned int n, const **CImgl**< T > &list, const unsigned int pos)
  Insert n shared copies of the list `list` *at position* `pos` *of the current list.*
- **CImgl** & **insert_shared** (const unsigned int n, const **CImgl**< T > &list)
  Insert n shared copies of the list `list` *at the end of the list.*
- **CImgl** & **remove** (const unsigned int pos)
  Remove the image at position `pos` *from the image list.*
- **CImgl** & **pop_back** ()
  Remove last element of the list;.
- **CImgl** & **pop_front** ()
  Remove first element of the list;.
- **CImgl** & **erase** (const **iterator** iter)
  Remove the element pointed by iterator `iter`;.
- **CImgl** & **remove** ()
  Remove the last image from the image list.
- **CImgl** & **reverse** ()
  Reverse list order.
- **CImgl** & **get_reverse** () const
  Get reversed list.
- **CImgl** & **operator**<< (const **CImg**< T > &img)
  Insert image at the end of the list.
- **CImgl** & **operator**>> (**CImg**< T > &img)
  Remove last image of the list.
- const **CImgl get_crop** (const unsigned int i0, const unsigned int i1, const bool shared=false) const
  Get a sub-list.
- **CImgl** & **crop** (const unsigned int i0, const unsigned int i1, const bool shared=false)
  Replace a list by its sublist.

Fourier Transforms

- **CImgl** & **FFT** (const char axe, const bool inverse=false)

  Compute the Fast Fourier Transform (along the specified axis).

- **CImgl** & **FFT** (const bool inverse=false)

  Compute the Fast Fourier Transform of a complex image.

- **CImgl**< typename cimg::largest< T, float >::type > **get_FFT** (const bool inverse=false) const

  Return the Fast Fourier Transform of a complex image.

- **CImgl**< typename cimg::largest< T, float >::type > **get_FFT** (const char axe, const bool inverse=false) const

  Return the Fast Fourier Transform of a complex image (along a specified axis).

Input-Output and Display

- const **CImgl** & **print** (const char *title=0, const int print_flag=1) const

  Print informations about the list on the standard error stream.

- **CImgl** & **load** (const char *filename)

  In-place version of **load()**(p. 127).

- **CImgl** & **load_cimg** (const char *filename)

  In-place version of **get_load_cimg()**(p. 128).

- **CImgl** & **load_parrec** (const char *filename)

  In-place version of **get_load_parrec()**(p. 128).

- **CImgl** & **load_yuv** (const char *filename, const unsigned int sizex, const unsigned int sizey, const unsigned int first_frame=0, const int last_frame=-1, const bool yuv2rgb=true)

  In-place version of **get_load_yuv()**(p. 128).

- const **CImgl** & **save_yuv** (const char *filename, const bool rgb2yuv=true) const

  Save an image sequence into a YUV file.

- const **CImgl** & **save** (const char *filename) const

  Save an image list into a file.

- const **CImgl** & **save_cimg** (const char *filename) const

  Save an image list into a **CImg**(p. 21) *RAW file.*

- template<typename tf, typename tc> **CImgl** & **load_off** (const char *filename, **CImgl**< tf > &primitives, **CImgl**< tc > &colors, const bool invert_faces=false)

  In-place version of **get_load_off()**(p. 128).

- template<typename tf, typename tc> const **CImgl** & **save_off** (const char *filename, const **CImgl**< tf > &primitives, const **CImgl**< tc > &colors, const bool invert_faces=false) const

  Save an image list into a OFF file.

- **CImg**< T > **get_append** (const char axe='x', const char align='c') const

  Return a single image which is the concatenation of all images of the current **CImgl**(p. 119) *instance.*

- const **CImgl** & **display** (**CImgDisplay** &disp, const char axe='x', const char align='c') const

  Display the current **CImgl**(p. 119) *instance in an existing* **CImgDisplay**(p. 112) *window (by reference).*

- const **CImgl** & **display** (const char *title, const char axe='x', const char align='c', const int min_size=128, const int max_size=1024) const

  Display the current **CImgl**(p. 119) *instance in a new display window.*

- const **CImgl** & **display** (const char axe='x', const char align='c', const int min_size=128, const int max_size=1024) const

  Display the current **CImgl**(p. 119) *instance in a new display window.*

- static **CImgl get_load** (const char *filename)

  Load an image list from a file.

- static **CImgl get_load_cimg** (const char *filename)

  Load an image list from a file (.raw format).

- static **CImgl get_load_parrec** (const char *filename)

  Load PAR-REC (Philips) image file.

- static **CImgl get_load_yuv** (const char *filename, const unsigned int sizex, const unsigned int sizey, const unsigned int first_frame=0, const int last_frame=-1, const bool yuv2rgb=true)

  Load YUV image sequence.

- template<typename tf, typename tc> static **CImgl**< T > **get_load_off** (const char *filename, **CImgl**< tf > &primitives, **CImgl**< tc > &colors, const bool invert_faces=false)

  Load from OFF file format.

- static **CImgl**< T > **get_font** (const unsigned int font_width, const bool variable_size=true)

  Return a **CImg**(p. 21) *pre-defined font with desired size.*

Public Types

- typedef **CImg**< T > * **iterator**

  Define a **CImgl**<**T**>**::iterator**(p. 128).

- typedef const **CImg**< T > * **const_iterator**

  Define a **CImgl**<**T**>**::const_iterator**(p. 128).

Public Attributes

- unsigned int **size**

  Size of the list (number of elements inside).

- unsigned int **allocsize**
  Allocation size of the list.
- **CImg**< T > ∗ **data**
  Pointer to the first list element.

4.4.1 Detailed Description

template<typename T> struct cimg_library::CImgl< T >

Class representing list of images CImg<T>.

4.4.2 Member Function Documentation

4.4.2.1 const CImgl& save (const char ∗ *filename*) 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.

4.4.2.2 const CImgl& save_cimg (const char ∗ *filename*) const

Save an image list into a **CImg**(p. 21) RAW file.

A **CImg**(p. 21) RAW file is a simple uncompressed binary file that may be used to save list of CImg<T> images.

Parameters:
filename : name of the output file.

Returns:
A reference to the current **CImgl**(p. 119) instance is returned.

4.4.2.3 CImg<T> get_append (const char *axe* = 'x', const char *align* = 'c') const

Return a single image which is the concatenation of all images of the current **CImgl**(p. 119) instance.

Parameters:
axe : specify the axe for image concatenation. Can be 'x','y','z' or 'v'.
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.

4.4.2.4 static CImgl<T> get_font (const unsigned int *font_width*, const bool *variable_size* = true) [static]

Return a **CImg**(p. 21) pre-defined font with desired size.

Parameters:
font_height = height of the desired font (can be 11,13,24,38 or 57)
fixed_size = tell if the font has a fixed or variable width.

4.4.2.5 const CImgl& display (CImgDisplay & *disp*, const char *axe* = 'x', const char *align* = 'c') const

Display the current **CImgl**(p. 119) instance in an existing **CImgDisplay**(p. 112) window (by reference).

This function displays the list images of the current **CImgl**(p. 119) instance into an existing **CImgDisplay**(p. 112) 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**(p. 112) instance, where the current image list will be displayed.

axe : specify the axe for image concatenation. Can be 'x','y','z' or 'v'.

align : specify the alignment for image concatenation. Can be 'p' (top), 'c' (center) or 'n' (bottom).

Returns:

A reference to the current **CImgl**(p. 119) instance is returned.

4.4.2.6 const CImgl& display (const char * *title*, const char *axe* = 'x', const char *align* = 'c', const int *min_size* = 128, const int *max_size* = 1024) const

Display the current **CImgl**(p. 119) instance in a new display window.

This function opens a new window with a specific title and displays the list images of the current **CImgl**(p. 119) 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.

axe : specify the axe for image concatenation. Can be 'x','y','z' or 'v'.

align : specify the alignment for image concatenation. Can be 'p' (top), 'c' (center) or 'n' (bottom).

min_size : specify the minimum size of the opening display window. Images having dimensions below this size will be upscaled.

max_size : specify the maximum size of the opening display window. Images having dimensions above this size will be downscaled.

Returns:

A reference to the current **CImgl**(p. 119) instance is returned.

4.4.2.7 const CImgl& display (const char *axe* = 'x', const char *align* = 'c', const int *min_size* = 128, const int *max_size* = 1024) const

Display the current **CImgl**(p. 119) instance in a new display window.

This function opens a new window and displays the list images of the current **CImgl**(p. 119) 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:

axe : specify the axe for image concatenation. Can be 'x','y','z' or 'v'.

align : specify the alignment for image concatenation. Can be 'p' (top), 'c' (center) or 'n' (bottom).

min_size : specify the minimum size of the opening display window. Images having dimensions below this size will be upscaled.

max_size : specify the maximum size of the opening display window. Images having dimensions above this size will be downscaled.

Returns:
A reference to the current **CImgl**(p. 119) instance is returned.

4.5 CImgStats Struct Reference

Class used to compute basic statistics on pixel values of a **CImg**(p. 21)<T> image.

Public Member Functions

- **CImgStats** ()

  Default constructor.

- **CImgStats** & **assign** ()

  In-place version of the default constructor.

- **CImgStats** (const **CImgStats** &stats)

  Copy constructor.

- **CImgStats** & **assign** (const **CImgStats** &stats)

  In-place version of the copy constructor.

- template<typename T> **CImgStats** (const **CImg**< T > &img, const bool compute_variance=true)

  Constructor that computes statistics of an input image `img`.

- template<typename T> **CImgStats** & **assign** (const **CImg**< T > &img, const bool compute_variance=true)

  In-place version of the previous constructor.

- template<typename T> **CImgStats** (const **CImgl**< T > &list, const bool compute_variance=true)

  Constructor that computes statistics of an input image list `list`.

- template<typename T> **CImgStats** & **assign** (const **CImgl**< T > &list, const bool compute_variance=true)

  In-place version of the previous constructor.

- **CImgStats** & **operator=** (const **CImgStats** &stats)

  Assignement operator.

- const **CImgStats** & **print** (const char *title=0) const

  Print the current statistics.

Public Attributes

- double **min**

  Minimum of the pixel values.

- double **max**

  Maximum of the pixel values.

- double **mean**

  Mean of the pixel values.

- double **variance**

  Variance of the pixel values.

- int **xmin**

  X-coordinate of the pixel with minimum value.

- int **ymin**

  Y-coordinate of the pixel with minimum value.

- int **zmin**

  Z-coordinate of the pixel with minimum value.

- int **vmin**

  V-coordinate of the pixel with minimum value.

- int **lmin**

  Image number (for a list) containing the minimum pixel.

- int **xmax**

  X-coordinate of the pixel with maximum value.

- int **ymax**

  Y-coordinate of the pixel with maximum value.

- int **zmax**

  Z-coordinate of the pixel with maximum value.

- int **vmax**

  V-coordinate of the pixel with maximum value.

- int **lmax**

  Image number (for a list) containing the maximum pixel.

4.5.1 Detailed Description

Class used to compute basic statistics on pixel values of a **CImg**(p. 21)<T> image.

Constructing a **CImgStats**(p. 131) instance from an image CImg<T> or a list CImgl<T> will compute the minimum, maximum and average pixel values of the input object. Optionally, the variance of the pixel values can be computed. Coordinates of the pixels whose values are minimum and maximum are also stored. The example below shows how to use **CImgStats**(p. 131) objects to retrieve simple statistics of an image :

```
const CImg<float> img("my_image.jpg");                // Read JPEG image file.
const CImgStats stats(img);                           // Compute basic statistics on the image.
stats.print("My statistics");                         // Display statistics.
std::printf("Max-Min = %lf",stats.max-stats.min);     // Compute the difference between extremum va
```

Note that statistics are computed by considering the set of *scalar* values of the image pixels. No vector-valued statistics are computed.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 CImgStats (const CImg< T > & *img*, const bool *compute_variance* = `true`)

Constructor that computes statistics of an input image `img`.

Parameters:

img The input image.

compute_variance If true, the `variance` field is computed, else it is set to 0.

4.5.2.2 CImgStats (const CImgl< T > & *list*, const bool *compute_variance* = `true`)

Constructor that computes statistics of an input image list `list`.

Parameters:

list The input list of images.

compute_variance If true, the `variance` field is computed, else it is set to 0.

4.5.3 Member Function Documentation

4.5.3.1 const CImgStats& print (const char * *title* = `0`) const

Print the current statistics.

Printing is done on the standart error output.

Index

abs
    cimg_library::CImg, 72

blur
    cimg_library::CImg, 108
blur_anisotropic
    cimg_library::CImg, 108

cimg_library, 15
cimg_library::CImg, 20
    abs, 72
    blur, 108
    blur_anisotropic, 108
    const_iterator, 57
    convolve, 107
    correlate, 106
    cos, 72
    crop, 85, 86
    cubic_pix1d, 62
    cubic_pix2d, 62
    cut, 78
    depth, 111
    deriche, 108
    dilate, 107
    dim, 112
    dimv, 58
    dimx, 58
    dimy, 58
    dimz, 58
    display, 109
    div, 68
    draw_arrow, 93
    draw_axe, 102
    draw_circle, 100
    draw_ellipse, 99
    draw_fill, 103
    draw_gaussian, 104, 105
    draw_graph, 102
    draw_image, 93, 94
    draw_line, 92, 93
    draw_object3d, 105
    draw_plasma, 104
    draw_point, 91, 92
    draw_quiver, 101, 102
    draw_rectangle, 94, 95
    draw_text, 100, 101
    draw_triangle, 95–98
    equalize_histogram, 88
    erode, 107
    fill, 73–77
    get_abs, 72
    get_blur, 108
    get_blur_anisotropic, 109
    get_convolve, 106
    get_correlate, 106
    get_cos, 72
    get_crop, 84, 85
    get_cut, 78
    get_default_LUT8, 90
    get_deriche, 108
    get_div, 68
    get_equalize_histogram, 88
    get_gradientXY, 90
    get_gradientXYZ, 90
    get_histogram, 88
    get_load, 109
    get_log, 71
    get_log10, 71
    get_max, 69
    get_min, 70
    get_mirror, 87
    get_mul, 68
    get_noise, 107
    get_norm_pointwise, 89
    get_normalize, 78
    get_orientation_pointwise, 89
    get_pow, 71
    get_quantize, 79
    get_resize, 81, 82
    get_resize_halfXY, 83
    get_RGBtoLUT, 91
    get_rotate, 79, 80
    get_scroll, 87
    get_sin, 72
    get_slice, 86
    get_sqrt, 70
    get_tan, 73
    get_threshold, 79
    height, 111
    iterator, 57
    linear_pix1d, 62
    linear_pix2d, 62
    linear_pix3d, 62
    linear_pix4d, 61
    load, 110
    load_ascii, 110
    load_dlm, 110
    log, 70
    log10, 71
    marching_cubes, 90
    marching_squares, 90

max, 69
min, 69, 70
mirror, 87
mul, 68
noise, 107
norm_pointwise, 89
normalize, 78
offset, 59
operator &, 66, 67
operator &=, 67
operator ∗, 65
operator ∗=, 65, 66
operator(), 59
operator+, 64
operator++, 64
operator+=, 64
operator-, 65
operator–, 65
operator-=, 65
operator/, 66
operator/=, 66
operator=, 63, 64
operator[], 60
operator%, 66
operator%=, 66
operator | , 67
operator | =, 67
operator^, 67, 68
operator^=, 68
orientation_pointwise, 90
pix4d, 60
pixel_type, 58
pow, 71
print, 63
ptr, 59
quantize, 79
resize, 82, 83
resize_halfXY, 83
RGBtoLUT, 91
rotate, 80
save, 110
save_convert, 110
save_png, 110
scroll, 87
sin, 72
size, 58
sqrt, 70
tan, 72
threshold, 79
width, 111
cimg_library::cimg, 16
convert_path, 18
dialog, 20
info, 19
medcon_path, 18
minmod, 19
mod, 19
sleep, 19
temporary_path, 19
wait, 19
cimg_library::CImgDisplay, 112
cimg_library::CImgDisplay
CImgDisplay, 116, 117
display, 117
resize, 118
wait, 117
cimg_library::CImgException, 118
cimg_library::CImgl, 119
display, 129, 130
get_append, 129
get_font, 129
save, 129
save_cimg, 129
cimg_library::CImgStats, 131
cimg_library::CImgStats
CImgStats, 133
print, 133
CImgDisplay
cimg_library::CImgDisplay, 116, 117
CImgStats
cimg_library::CImgStats, 133
const_iterator
cimg_library::CImg, 57
convert_path
cimg_library::cimg, 18
convolve
cimg_library::CImg, 107
correlate
cimg_library::CImg, 106
cos
cimg_library::CImg, 72
crop
cimg_library::CImg, 85, 86
cubic_pix1d
cimg_library::CImg, 62
cubic_pix2d
cimg_library::CImg, 62
cut
cimg_library::CImg, 78

depth
cimg_library::CImg, 111
deriche
cimg_library::CImg, 108
dialog
cimg_library::cimg, 20
dilate
cimg_library::CImg, 107

dim
cimg_library::CImg, 112
dimv
cimg_library::CImg, 58
dimx
cimg_library::CImg, 58
dimy
cimg_library::CImg, 58
dimz
cimg_library::CImg, 58
display
cimg_library::CImg, 109
cimg_library::CImgDisplay, 117
cimg_library::CImgl, 129, 130
div
cimg_library::CImg, 68
draw_arrow
cimg_library::CImg, 93
draw_axe
cimg_library::CImg, 102
draw_circle
cimg_library::CImg, 100
draw_ellipse
cimg_library::CImg, 99
draw_fill
cimg_library::CImg, 103
draw_gaussian
cimg_library::CImg, 104, 105
draw_graph
cimg_library::CImg, 102
draw_image
cimg_library::CImg, 93, 94
draw_line
cimg_library::CImg, 92, 93
draw_object3d
cimg_library::CImg, 105
draw_plasma
cimg_library::CImg, 104
draw_point
cimg_library::CImg, 91, 92
draw_quiver
cimg_library::CImg, 101, 102
draw_rectangle
cimg_library::CImg, 94, 95
draw_text
cimg_library::CImg, 100, 101
draw_triangle
cimg_library::CImg, 95–98

equalize_histogram
cimg_library::CImg, 88
erode
cimg_library::CImg, 107

Files IO in CImg., 13
fill
cimg_library::CImg, 73–77

get_abs
cimg_library::CImg, 72
get_append
cimg_library::CImgl, 129
get_blur
cimg_library::CImg, 108
get_blur_anisotropic
cimg_library::CImg, 109
get_convolve
cimg_library::CImg, 106
get_correlate
cimg_library::CImg, 106
get_cos
cimg_library::CImg, 72
get_crop
cimg_library::CImg, 84, 85
get_cut
cimg_library::CImg, 78
get_default_LUT8
cimg_library::CImg, 90
get_deriche
cimg_library::CImg, 108
get_div
cimg_library::CImg, 68
get_equalize_histogram
cimg_library::CImg, 88
get_font
cimg_library::CImgl, 129
get_gradientXY
cimg_library::CImg, 90
get_gradientXYZ
cimg_library::CImg, 90
get_histogram
cimg_library::CImg, 88
get_load
cimg_library::CImg, 109
get_log
cimg_library::CImg, 71
get_log10
cimg_library::CImg, 71
get_max
cimg_library::CImg, 69
get_min
cimg_library::CImg, 70
get_mirror
cimg_library::CImg, 87
get_mul
cimg_library::CImg, 68
get_noise
cimg_library::CImg, 107

get_norm_pointwise
cimg_library::CImg, 89
get_normalize
cimg_library::CImg, 78
get_orientation_pointwise
cimg_library::CImg, 89
get_pow
cimg_library::CImg, 71
get_quantize
cimg_library::CImg, 79
get_resize
cimg_library::CImg, 81, 82
get_resize_halfXY
cimg_library::CImg, 83
get_RGBtoLUT
cimg_library::CImg, 91
get_rotate
cimg_library::CImg, 79, 80
get_scroll
cimg_library::CImg, 87
get_sin
cimg_library::CImg, 72
get_slice
cimg_library::CImg, 86
get_sqrt
cimg_library::CImg, 70
get_tan
cimg_library::CImg, 73
get_threshold
cimg_library::CImg, 79

height
cimg_library::CImg, 111
How pixel data are stored with CImg., 13

info
cimg_library::cimg, 19
Introduction to the CImg Library, 1
iterator
cimg_library::CImg, 57

linear_pix1d
cimg_library::CImg, 62
linear_pix2d
cimg_library::CImg, 62
linear_pix3d
cimg_library::CImg, 62
linear_pix4d
cimg_library::CImg, 61
load
cimg_library::CImg, 110
load_ascii
cimg_library::CImg, 110
load_dlm
cimg_library::CImg, 110
log
cimg_library::CImg, 70
log10
cimg_library::CImg, 71

marching_cubes
cimg_library::CImg, 90
marching_squares
cimg_library::CImg, 90
max
cimg_library::CImg, 69
medcon_path
cimg_library::cimg, 18
min
cimg_library::CImg, 69, 70
minmod
cimg_library::cimg, 19
mirror
cimg_library::CImg, 87
mod
cimg_library::cimg, 19
mul
cimg_library::CImg, 68

noise
cimg_library::CImg, 107
norm_pointwise
cimg_library::CImg, 89
normalize
cimg_library::CImg, 78

offset
cimg_library::CImg, 59
operator &
cimg_library::CImg, 66, 67
operator &=
cimg_library::CImg, 67
operator ∗
cimg_library::CImg, 65
operator ∗=
cimg_library::CImg, 65, 66
operator()
cimg_library::CImg, 59
operator+
cimg_library::CImg, 64
operator++
cimg_library::CImg, 64
operator+=
cimg_library::CImg, 64
operator-
cimg_library::CImg, 65
operator–
cimg_library::CImg, 65

operator-=
cimg_library::CImg, 65
operator/
cimg_library::CImg, 66
operator/=
cimg_library::CImg, 66
operator=
cimg_library::CImg, 63, 64
operator[]
cimg_library::CImg, 60
operator%
cimg_library::CImg, 66
operator%=
cimg_library::CImg, 66
operator|
cimg_library::CImg, 67
operator|=
cimg_library::CImg, 67
operator^
cimg_library::CImg, 67, 68
operator^=
cimg_library::CImg, 68
orientation_pointwise
cimg_library::CImg, 90

pix4d
cimg_library::CImg, 60
pixel_type
cimg_library::CImg, 58
pow
cimg_library::CImg, 71
print
cimg_library::CImg, 63
cimg_library::CImgStats, 133
ptr
cimg_library::CImg, 59

quantize
cimg_library::CImg, 79

resize
cimg_library::CImg, 82, 83
cimg_library::CImgDisplay, 118
resize_halfXY
cimg_library::CImg, 83
Retrieving Command Line Arguments., 13
RGBtoLUT
cimg_library::CImg, 91
rotate
cimg_library::CImg, 80

save
cimg_library::CImg, 110
cimg_library::CImgl, 129
save_cimg
cimg_library::CImgl, 129
save_convert
cimg_library::CImg, 110
save_png
cimg_library::CImg, 110
scroll
cimg_library::CImg, 87
Setting Environment Variables, 3
sin
cimg_library::CImg, 72
size
cimg_library::CImg, 58
sleep
cimg_library::cimg, 19
sqrt
cimg_library::CImg, 70

tan
cimg_library::CImg, 72
temporary_path
cimg_library::cimg, 19
threshold
cimg_library::CImg, 79
Tutorial : Getting Started., 5

Using Display Windows., 12
Using Drawing Functions., 7
Using Image Loops., 7

wait
cimg_library::cimg, 19
cimg_library::CImgDisplay, 117
width
cimg_library::CImg, 111