The Clmg Library
C++ Template Image Processing Toolkit

G’MIC
GREYC’s Magic Image Converter

David Tschumperlé
Image Team, GREYC / CNRS (UMR 6072)
IPOL Workshop on Image Processing Libraries, Cachan/France, June 2012
Presentation layout

1. Image Processing: Get the Facts
2. The CImg Library: C++ Template Image Processing Library
4. Conclusions
Presentation layout

1. Image Processing: Get the Facts
2. The CImg Library: C++ Template Image Processing Library
4. Conclusions
Fact 1: The image processing research world is wide. Many different people compose it, each with a different scientific background:

- Mathematicians
- Physicists
- Computer geeks
- Biologists

Fact 2: They all work on images, trying to solve many different problems, involving a wide diversity of image data. Photography, medical imaging, astronomy, robot vision, fluid dynamics, etc...
Fact 3: Digital images are generic objects by nature.

On a computer, image data are usually stored as discrete arrays of values (pixels or voxels), but the diversity of acquired images is important.
Diversity of Image Data

$2D \rightarrow [0, 255]$  

$2D \rightarrow [0, 255]^3$  

$(2D + t) \rightarrow [0, 255]^3$

$3D \rightarrow [0, 16383]$  

$3D \rightarrow \mathbb{R}^6$  

$(2D + t) \rightarrow [0, 16384]$
Diversity of Image Data

Acquired digital images may be of different types:

- **Domain dimensions**: $2D$ (static image), $2D + t$ (image sequence), $3D$ (volumetric image), $3D + t$ (sequence of volumetric images), ...
- **Pixel dimensions**: Pixels can be scalars, colors, $N - D$ vectors, matrices, ...
- **Pixel value range**: depends on the sensors used for acquisition, can be $N$-bits (usually 8, 16, 24, 32...), sometimes (often) float-valued.
- **Type of sensor grid**: Square, Rectangular, Octagonal, Graph, ...

All these different image data are digitally stored using dedicated file formats:

- PNG, JPEG, BMP, TIFF, TGA, DICOM, ANALYZE, AVI, MPEG, ...
Fact 4: Usual image processing algorithms are mostly image type independent.

e.g.: Binarization of an image $I : \Omega \rightarrow \Gamma$ by a threshold $\epsilon \in \mathbb{R}$.

\[ \tilde{I} : \Omega \rightarrow \{0, 1\} \quad \text{such that} \quad \forall p \in \Omega, \quad \tilde{I}(p) = \begin{cases} 0 & \text{if} \quad \|I(p)\| < \epsilon \\ 1 & \text{if} \quad \|I(p)\| \geq \epsilon \end{cases} \]

Implementing an image processing algorithm should be as much independent as possible of the image format and coding.
How to help those **various people** implementing **image processing algorithms** working on **generic images** in an **easy way**?
Based on these facts, we designed **CImg** and **G’MIC**, two lightweight image processing toolboxes fitting these constraints:

- **Simplicity**: Easy to install, **easy to take control**. Two different scales of uses (C++ and script).
- **Genericity**: Generic enough for managing a wide variety of data types (template-based).
- **Usefulness**: Provides useful, classical and must-have algorithms and tools.
- **Extensibility**: Extensible frameworks by nature.
- **Portability**: Easy to spread from/to any computer (portable to various architectures and OS).
- **Freedom**: Distributed under open-Source licenses.
Based on these facts, we designed **CImg** and **G’MIC**, two lightweight image processing toolboxes fitting these constraints:

- **Simplicity**: Easy to install, *easy to take control*. Two different scales of uses (C++ and script).
- **Genericity**: *Generic enough* for managing a wide variety of data types (*template-based*).
- **Usefulness**: Provides useful, classical and must-have algorithms and tools.
- **Extensibility**: Extensible frameworks by nature.
- **Portability**: Easy to spread from/to any computer (portable to various architectures and OS).
- **Freedom**: Distributed under open-Source licenses.
Based on these facts, we designed **CImg** and **G’MIC**, two lightweight image processing toolboxes fitting these constraints:

- **Easy to install, easy to take control.** Two different scales of uses (C++ and script).
- **Generic enough** for managing a wide variety of data types (template-based).
- Provides useful, classical and must-have algorithms and tools.
- **Extensible frameworks by nature.**
- **Easy to spread from/to any computer** (portable to various architectures and OS).
- **Distributed under open-Source licenses**
CImg and G’MIC Philosophy

Based on these facts, we designed **CImg** and **G’MIC**, two lightweight image processing toolboxes fitting these constraints:

- **Simplicity**: Easy to install, easy to take control. Two different scales of uses (C++ and script).
- **Genericity**: Generic enough for managing a wide variety of data types (template-based).
- **Usefulness**: Provides useful, classical and must-have algorithms and tools.
- **Extensibility**: Extensible frameworks by nature.
- **Portability**: Easy to spread from/to any computer (portable to various architectures and OS).
- **Freedom**: Distributed under open-Source licenses.
Based on these facts, we designed **CImg** and **G’MIC**, two lightweight image processing toolboxes fitting these constraints:

- **Simplicity**: Easy to install, *easy to take control*. Two different scales of uses (C++ and script).

- **Genericity**: *Generic enough* for managing a wide variety of data types. (template-based).

- **Usefulness**: Provides useful, classical and must-have algorithms and tools.

- **Extensibility**: Extensible frameworks by nature.

- **Portability**: Easy to spread from/to any computer (portable to various architectures and OS).

- **Freedom**: Distributed under open-Source licenses.
CImg and G’MIC Philosophy

- Based on these facts, we designed **CImg** and **G’MIC**, two lightweight image processing toolboxes fitting these constraints:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simplicity</td>
<td>Easy to install, easy to take control. Two different scales of uses (C++ and script).</td>
</tr>
<tr>
<td>Genericity</td>
<td>Generic enough for managing a wide variety of data types. (template-based).</td>
</tr>
<tr>
<td>Usefulness</td>
<td>Provides useful, classical and must-have algorithms and tools.</td>
</tr>
<tr>
<td>Extensibility</td>
<td>Extensible frameworks by nature.</td>
</tr>
<tr>
<td>Portability</td>
<td>Easy to spread from/to any computer (portable to various architectures and OS).</td>
</tr>
<tr>
<td>Freedom</td>
<td>Distributed under open-Source licenses</td>
</tr>
</tbody>
</table>
1. Image Processing: Get the Facts

2. The CImg Library: C++ Template Image Processing Library


4. Conclusions
The CImg Library: Overview

What?: Small C++ library aiming to simplify the development of image processing algorithms for generic-enough datasets.

For whom?: For Researchers and Students in Image Processing and Computer Vision, having basic notions of C++.

How?: Defines a minimal set of templated C++ classes able to manipulate and process image datasets.

Since when?: Started in late 1999, hosted on Sourceforge since December 2003 (about 1200 visits and 100 downloads/day).

http://cimg.sourceforge.net/
- **Easy to get**: CImg is distributed as a .zip package (≈ 12.7 Mo) containing the library code (≈ 40.000 loc), examples of use, documentations and resource files.

- **Easy to use**: Using CImg requires only the include of a single C++ header file. No complex installation, no pre-compilation:

  ```cpp
  #include "CImg.h" // Just do that...
  using namespace cimg_library; // ..Ready to go!
  ```

- **Easy to understand**: It defines only four C++ classes: `CImg<T>`, `CImgList<T>`, `CImgDisplay`, `CImgException` Image processing algorithms are methods of these classes:

  ```cpp
  CImg<T>::blur(), CImgList<T>::insert(), CImgDisplay::resize(), ...
  ```

- **CImg Motto**: KIS(I)S, Keep it Small and (Insanely) Simple.
Easy to get: CImg is distributed as a .zip package (≈ 12.7 Mo) containing the library code (≈ 40,000 loc), examples of use, documentations and resource files.

Easy to use: Using CImg requires only the include of a single C++ header file. No complex installation, no pre-compilation:
```cpp
#include "CImg.h" // Just do that...
using namespace cimg_library; // ..Ready to go!
```

Easy to understand: It defines only four C++ classes: CImg<T>, CImgList<T>, CImgDisplay, CImgException. Image processing algorithms are methods of these classes: CImg<T>::blur(), CImgList<T>::insert(), CImgDisplay::resize(), ...

CImg Motto: KIS(I)S, Keep it Small and (Insanely) Simple.
Easy to get: CImg is distributed as a .zip package (∼12.7 Mo) containing the library code (∼40,000 loc), examples of use, documentations and resource files.

Easy to use: Using CImg requires only the include of a single C++ header file. No complex installation, no pre-compilation:
#include "CImg.h" // Just do that...
using namespace cimg_library; // ..Ready to go!

Easy to understand: It defines only four C++ classes:
CImg<T>, CImgList<T>, CImgDisplay, CImgException
Image processing algorithms are methods of these classes:
CImg<T>::blur(), CImgList<T>::insert(),
CImgDisplay::resize(), ...

CImg Motto: KIS(I)S, Keep it Small and (Insanely) Simple.
Easy to get: CImg is distributed as a .zip package (∼12.7 Mo) containing the library code (∼40,000 loc), examples of use, documentations and resource files.

Easy to use: Using CImg requires only the include of a single C++ header file. No complex installation, no pre-compilation:
```cpp
#include "CImg.h"  // Just do that...
using namespace cimg_library; // ..Ready to go!
```

Easy to understand: It defines only four C++ classes:
- CImg<T>, CImgList<T>, CImgDisplay, CImgException
Image processing algorithms are methods of these classes:
- CImg<T>::blur(), CImgList<T>::insert(),
- CImgDisplay::resize(), ...

CImg Motto: KIS(I)S, Keep it Small and (Insanely) Simple.
CImg is *generic-enough* for most cases:

- CImg implements static genericity using C++ templates.
  
  **KISS philosophy**: One template parameter only!
  
  ⇒ the type of the image pixel (bool, char, int, float, ...).

- A `CImg<T>` instance can handle hyperspectral volumetric images (4D = width×height×depth×spectrum).

- A `CImgList<T>` instance can handle sequences or collections of 4D images.

⇒ CImg covers actually a lot of the image data types found in real world applications, while defining straightforward structures that are still understandable by non computer-geeks.
CImg is **generic-enough** for most cases:

- CImg implements static genericity using C++ templates. **KISS philosophy**: One template parameter only! ➞ the type of the image pixel (bool, char, int, float, ...).
- A `CImg<T>` instance can handle hyperspectral volumetric images (4D = width \times height \times depth \times spectrum).
- A `CImgList<T>` instance can handle sequences or collections of 4D images.

→ CImg covers actually a lot of the image data types found in real world applications, while defining straightforward structures that are still understandable by non computer-geeks.
CImg is **generic-enough** for most cases:

- CImg implements static genericity using C++ templates.
  
  **KISS philosophy**: One template parameter only!
  
  $\Rightarrow$ the type of the image pixel (bool, char, int, float, ...).

- A `CImg<T>` instance can handle **hyperspectral volumetric images**
  $(4D = \text{width} \times \text{height} \times \text{depth} \times \text{spectrum})$.

- A `CImgList<T>` instance can handle **sequences or collections of**
  **4D images**.

$\Rightarrow$ CImg covers actually a lot of the image data types found in real
world applications, while defining straightforward structures that are **still understandable by non computer-geeks**.
What we wanted to avoid at any price!

⇒ Discouraging for any average C++ programmer!!
(i.e. most of the researchers in Image Processing).
What we actually have!

⇒ Looks simpler!
CImg has algorithms/methods everybody is looking for:

- **Data inputs/outputs**: supports a large number of image file formats (e.g. float-valued multi-page tiff files).
- **Usual IP operators**: Convolution, gradients, histograms, color conversions, interpolation, geometric transformations, non-linear blur/sharpening, displacement field estimation, FFT, ...
- **Arithmetic operators**: Most usual mathematical operations between images are defined (e.g. operator+(), sqrt(),...).
- **Vector / matrix operations**: SVD, matrix inversion, linear system solving, eigenvalues, ...
- **Image drawing functions**: Lines, polygons, ellipses, text, vector fields, graphs, 3D objects, ...

All methods and algorithms of CImg are designed to work flawlessly on 4D images CImg<T>.
Methods of CImg<T> can be pipelined to write complex image processing pipelines in few lines:

```cpp
#include "CImg.h"
using namespace cimg_library;
int main() {

    // Load 521x512 lena color image.
    CImg<> img("lena.bmp");

    // Do some weird pipelines.
    img.RGBtoYCbCr().channel(0).quantize(10,false).
    map(CImg<>(3,1,1,3).rand(0,255).resize(10,1,1,3,3));

    // Display result.
    img.display("My nice image");
}
```
CImg owns a mathematical expressions evaluator:

```cpp
#include "CImg.h"
using namespace cimg_library;
int main() {

    // Construct 256x256 color image.
    CImg img(256, 256, 1, 3);

    // Fill pixel values from a formula.
    img = "X=x-w/2;Y=y-h/2;D=sqrt(X^2+Y^2);" 
         "if(D+u*20<80,abs(255*cos(D/(5+c)))," 
         "10*(y%(20+c))");

    // Display result.
    (img, img.get_gradient("xy")).display();
}
```
CImg has a lot of methods to draw things on images, as well as a class (CImgDisplay) to display images on windows and interact with the user.
CImg has its own 3d renderer (kind of mini OpenGL):

```cpp
#include "CImg.h"
using namespace cimg_library;
int main() {

    // Load 3d object from a .off file.
    CImgList<unsigned int> primitives;
    CImgList<unsigned char> colors;
    const CImg<float> points = CImg<>::load_off(primitives, colors, "3dhisto.off");

    // Display 3d object in interactive window.
    CImg<unsigned char>(800, 600, 1, 3, 128).
        display_object3d("3d object", points, primitives, colors);
}
```
You can add your **own methods** in the `CImg<T>` or `CImgList<T>` classes, **without having to modify the library code.**

```cpp
#define cimg_plugin "foo.h"
#include "CImg.h"
using namespace cimg_library;
int main() {

    CImg<> img("lena.bmp");
    img.my_method();
}

⇒ Plug-in mechanism !
Plug-in file `foo.h` contains:

```cpp
CImg<T>& my_method() {
    const CImgList<T> g = get_gradient("xyz");
    (g[0].sqr() + g[1].sqr() + g[2].sqr()).sqrt().move_to(*this);
    return *this;
}
```

Some plug-ins are already distributed within the CImg package: NLmeans, Skeleton, VRML reader, CImg<->Matlab conversion, ...
The CImg Library code is compiled on the fly.

⇒ The library configuration is decided by the CImg users, not by the CImg developers.

Many existing configuration flags, allow to enable/disable extra functionalities, provided by external libraries:

cimg_use_png, cimg_use_openmp, cimg_use_lapack, cimg_use_fftw3, cimg_use_opencv, cimg_use_jpeg, cimg_use_tiff, cimg_use_ffmpeg, cimg_use_zlib, cimg_use_openexr, ....
CImg is distributed under the CeCILL-C license (permissive, LGPL-like).

The code of CImg is **small** and **easy to maintain**.
→ portable library (multi-CPU, multi-OS, multi-compilers).

The CImg structures are **insanely simple**
→ CImg is easy to integrate and to communicate with other image processing libraries.

⇒ Isn’t it the perfect image processing library? 😊
1. Image Processing : Get the Facts
2. The CImg Library : C++ Template Image Processing Library
4. Conclusions
**Motivations**

- **Observation 1**: CImg requires (basic) C++ knowledge. Some people don’t know C++ but could be interested by the CImg capabilities anyway.

- **Observation 2**: When we get new image data, we often want to perform the same basic operations on them (visualization, gradient computation, noise reduction, ...).

- **Observation 3**: It is not optimal to create C++ code specifically for these minor tasks (requires code edition, compilation time, ..).

⇒ **G’MIC defines a script language which interfaces the CImg functionalities.**

⇒ **No compilation required, all CImg features usable from the shell.**
Motivations

- **Observation 1**: CImg requires (basic) C++ knowledge. Some people don’t know C++ but could be interested by the CImg capabilities anyway.

- **Observation 2**: When we get new image data, we often want to perform the **same basic operations** on them (visualization, gradient computation, noise reduction, ...).

- **Observation 3**: It is not optimal to create C++ code specifically for these minor tasks (requires code edition, compilation time, ..).

  ➔ G’MIC defines a script language which interfaces the CImg functionalities.

  ➔ No compilation required, all CImg features usable **from the shell**.
Motivations

Observation 1: CImg requires (basic) C++ knowledge. Some people don’t know C++ but could be interested by the CImg capabilities anyway.

Observation 2: When we get new image data, we often want to perform the same basic operations on them (visualization, gradient computation, noise reduction, ...).

Observation 3: It is not optimal to create C++ code specifically for these minor tasks (requires code edition, compilation time, ..).

⇒ G’MIC defines a script language which interfaces the CImg functionalities.

⇒ No compilation required, all CImg features usable from the shell.
Motivations

- **Observation 1**: CImg requires (basic) C++ knowledge. Some people don’t know C++ but could be interested by the CImg capabilities anyway.

- **Observation 2**: When we get new image data, we often want to perform the **same basic operations** on them (visualization, gradient computation, noise reduction, ...).

- **Observation 3**: It is not optimal to create C++ code specifically for these minor tasks (requires code edition, compilation time, ..).

⇒ G’MIC defines a script language which interfaces the CImg functionalities.

⇒ No compilation required, all CImg features usable **from the shell**.
G’MIC : Language properties

- G’MIC manage a list of images (i.e. an instance of CImgList<T>).
- Each G’MIC instruction runs an image processing algorithm, or control the program execution: `-blur`, `-rgb2hsv`, `-isosurface3d`, `-if`, `-endif` ...
- A G’MIC pipeline is executed by calls to CImg methods.
- User-defined functions can be saved as G’MIC script files.
- The G’MIC interpreter can be called from the command line or from any external project (itself provided as a stand-alone library).
gmic lena.bmp -blur 3 -sharpen 1000 -noise 30 --
"
'\cos(x/3) \times 30'"
gmic reference.inr -flood 23,53,30,50,1,1000 -flood[-2] 0,0,0,30,1,1000 -blur 1 -isosurface3d 900 -opacity3d[-2] 0.2 -color3d[-1] 255,128,0 -+3d
gmic -isosurface3d
"'\sin(x*y*z)'",0,-10,-10,-10,10,10,10,128,128,64
gmic milla.bmp -f '255*(i/255)^1.7' -histogram 128,0,255 -a c -plot

is the G'MIC equivalent code to

#include "CImg.h"
using namespace cimg_library;
int main(int argc,char **argv) {
const CImg<> img("milla.bmp"),
hist = img.get_histogram(128,0,255),
img2 = img.get_fill("255*((i/255)^1.7)",true),
hist2 = img2.get_histogram(128,0,255);
(hist,hist2).get_append('c').display_graph("Histograms");
return 0;
}
gmic lena.jpg -pencilbw 0.3 -o gmic_lena1.jpg; gmic lena.jpg -cubism 160 -o gmic_lena3.jpg

A better ImageMagick’s “convert”?

😊
Plug-in G’MIC for GIMP

CImg functionalities available for everyone!

⇒ ≈ 400-500 downloads/day (+600,000 dl since 2008).
1. Image Processing: Get the Facts
2. The CLimg Library: C++ Template Image Processing Library
4. Conclusions
The CImg Library is a very small and easy-to-use C++ library that eases the coding of image processing algorithms.

http://cimg.sourceforge.net/

G’MIC is the script-based counterpart of CImg.

http://gmic.sourceforge.net/

These projects are Open-Source and can be used, modified and redistributed without hard restrictions.

Generic (enough) libraries to do generic things.

Small, open and easily embeddable libraries: can be integrated in third parties applications.
The end

Thank you for your attention.

Time for questions if any ..