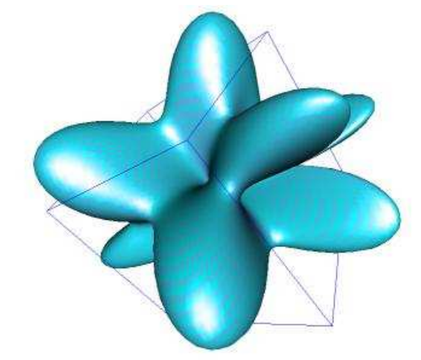
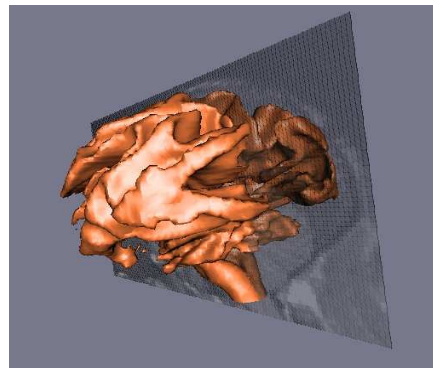
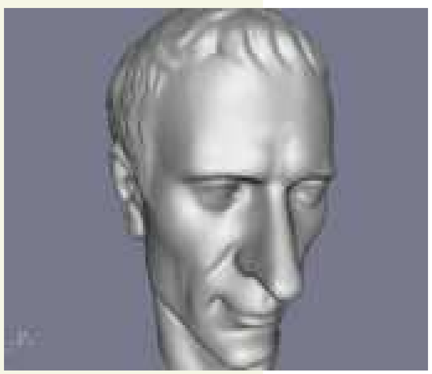
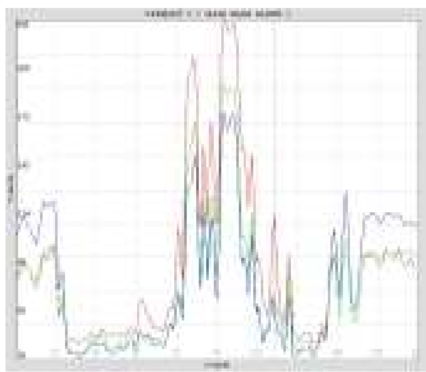
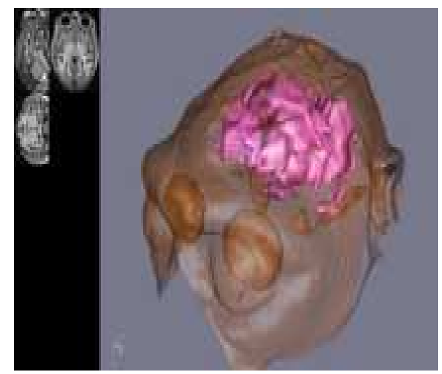
The CImg Library and G'MIC

Open-Source Toolboxes for Image Processing at Different Levels



David Tschumperlé

{ Image Team - GREYC Laboratory (CNRS UMR 6072) - Caen / France}

Séminaire LRDE, Paris / France, Octobre 2009.

Talk outline

- **Context and Philosophy** : Research in Image Processing
- **“Low-level” use (C++)** : The CImg Library
- **“Middle-level” use (script)** : G’MIC
- **“High-level” use** : Providing GUI, and results in real applications

Talk outline

⇒ **Context and Philosophy : Research in Image Processing**

- **“Low-level” use (C++)** : The CImg Library
- **“Middle-level” use (script)** : G’MIC
- **“High-level” use** : Providing GUI, and results in real applications

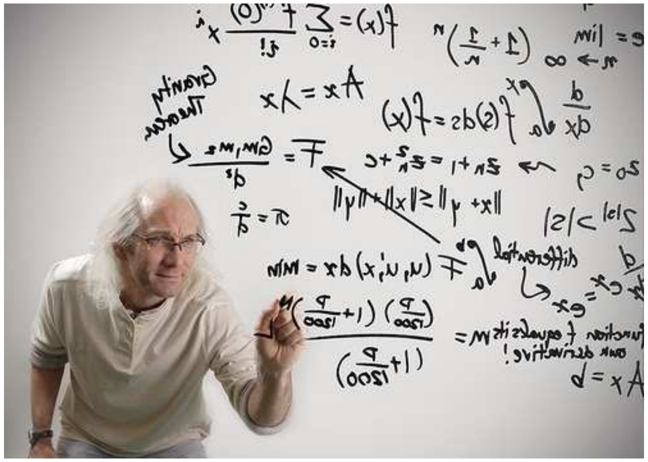
Context : Research in Image Processing

- Fact 1 : The image processing research world is **wide**.

  It is composed of many different people, with different scientific backgrounds :

- Fact 1 : The image processing research world is **wide**.

It is composed of many different people, with different scientific backgrounds :

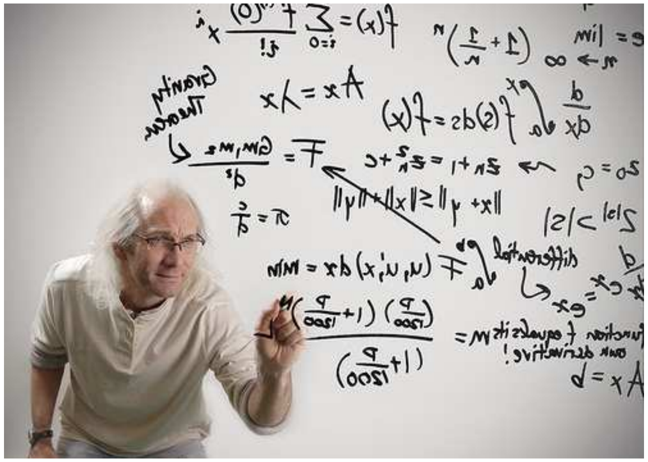


Mathematicians

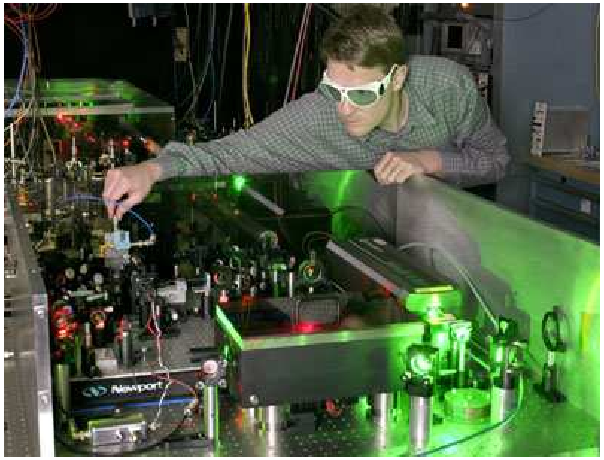
Context : Research in Image Processing

- Fact 1 : The image processing research world is **wide**.

  It is composed of many different people, with different scientific backgrounds :



Mathematicians

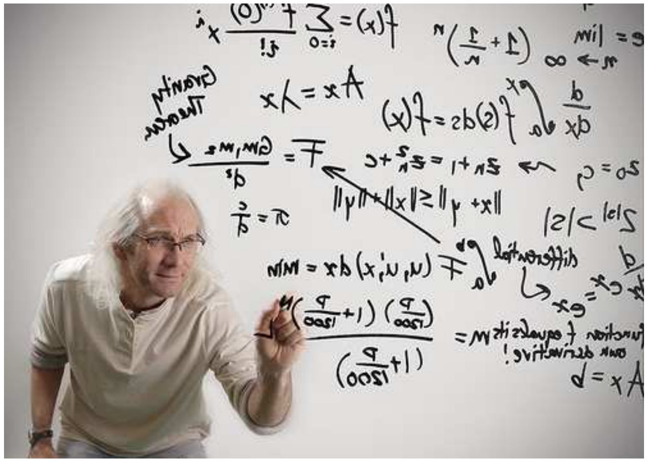


Physicists

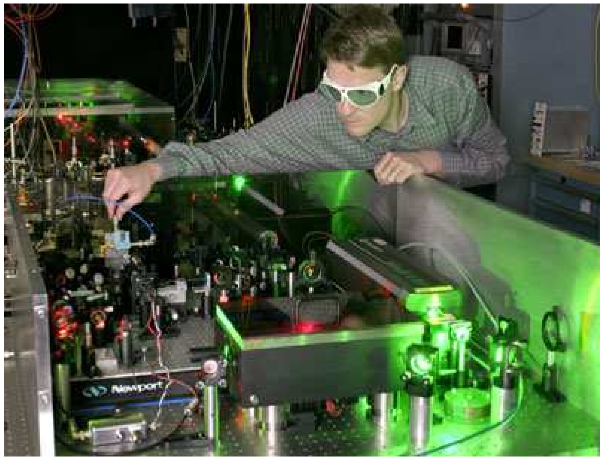
Context : Research in Image Processing

- <u>Fact 1</u> : The image processing research world is **wide**.

  It is composed of many different people, with different scientific backgrounds :



Mathematicians



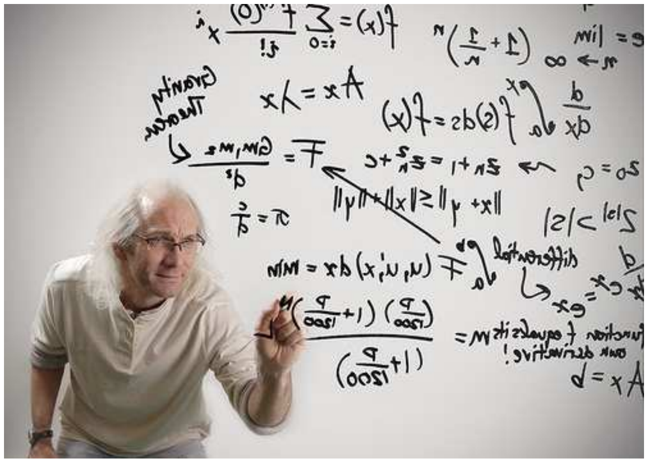
Physicists

Computer Scientists

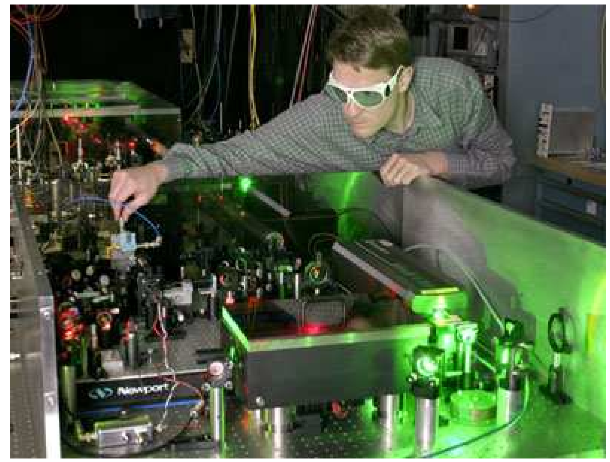
Context : Research in Image Processing

- Fact 1 : The image processing research world is **wide**.

  It is composed of many different people, with different scientific backgrounds :



Mathematicians



Physicists

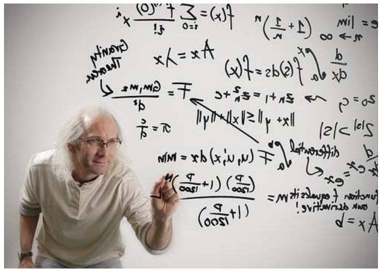
Computer Scientists

Biologists

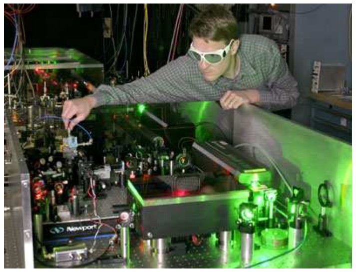
Context : Research in Image Processing

- Fact 1 : The image processing research world is **wide**.

  It is composed of many different people, with different scientific backgrounds :



Mathematicians



Physicists

Computer Scientists

Biologists

....(and others)....

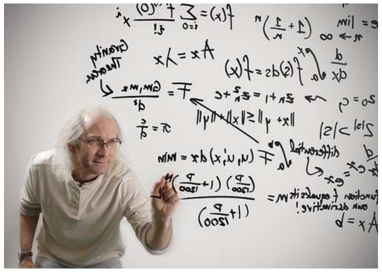
- Fact 2 : These different people work on images for **various reasons**.

  Photography, medical imaging, astronomy, robot vision, fluid dynamics, etc.

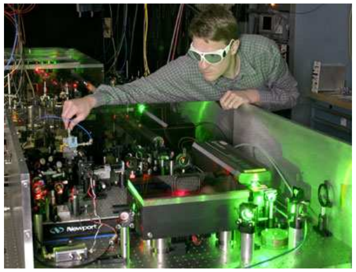
Context : Research in Image Processing

- Fact 1 : The image processing research world is **wide**.

  It is composed of many different people, with different scientific backgrounds :



Mathematicians



Physicists

Computer Scientists

Biologists

....(and others)....

- Fact 2 : These different people work on images for **various reasons**.

  Photography, medical imaging, astronomy, robot vision, fluid dynamics, etc.

$\Longrightarrow$ The numbers of considered problems and image datasets are **actually huge**.

Context : Problematic

⇒ How to design image processing tools which can be helpful for these scientists ?

Context : Problematic

⇒ How to design image processing tools which can be helpful for these scientists ?

Usefulness Should provide useful, classical and all-purpose algorithms.

Context : Problematic

⇒ How to design image processing tools which can be helpful for these scientists ?

Usefulness Should provide useful, classical and all-purpose algorithms.

Simplicity Should be easy to use, to understand, even for non-computer geeks.

Context : Problematic

⇒ How to design image processing tools which can be helpful for these scientists ?

Usefulness Should provide useful, classical and all-purpose algorithms.

Simplicity Should be easy to use, to understand, even for non-computer geeks.

Genericity Should be generic enough to serve for a wide variety of applications.

Context : Problematic

⇒ How to design image processing tools which can be helpful for these scientists ?

Usefulness Should provide useful, classical and all-purpose algorithms.

Simplicity Should be easy to use, to understand, even for non-computer geeks.

Genericity Should be generic enough to serve for a wide variety of applications.

Portability Should be easy to spread from/to any computer.

Context : Problematic

⇒ How to design image processing tools which can be helpful for these scientists ?

Usefulness Should provide useful, classical and all-purpose algorithms.

Simplicity Should be easy to use, to understand, even for non-computer geeks.

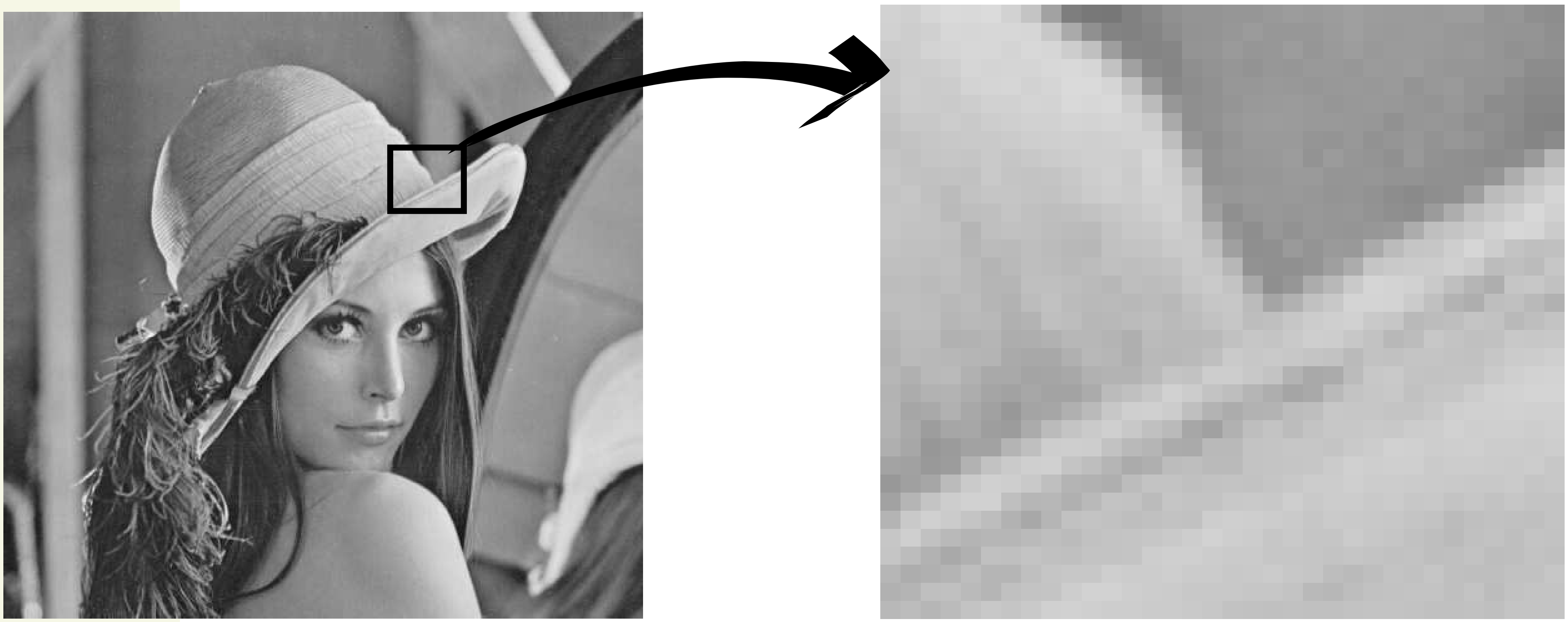
Genericity Should be generic enough to serve for a wide variety of applications.

Portability Should be easy to spread from/to any computer.

Freedom Should be compatible with a scientific spirit, i.e. can be **used, modified and distributed** by anyone, without hard restrictions.
(≠ proprietary software)

Context : About genericity of images

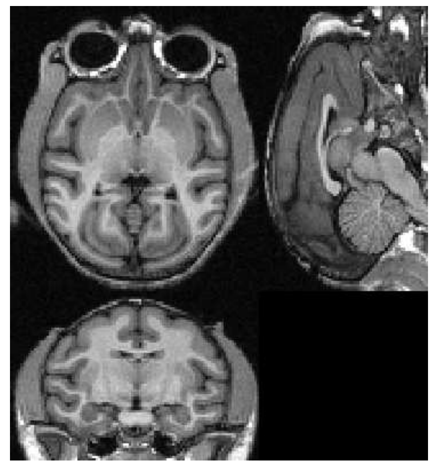
- Fact 3 : Digital Images are **generic objects by nature**.

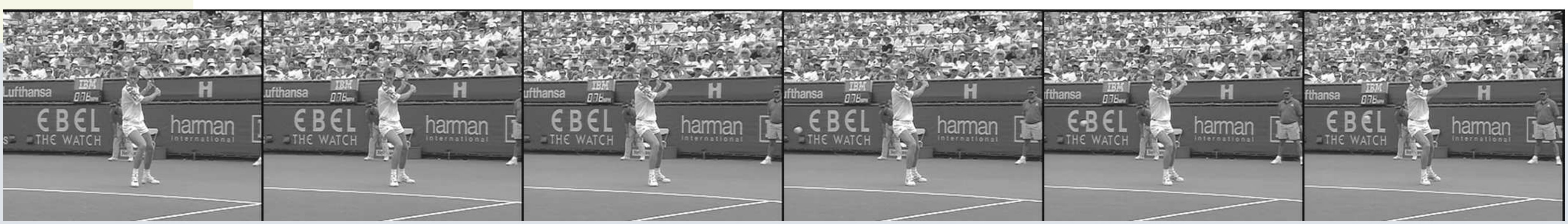


- On a computer, image data are usually stored as a discrete array of values (pixels or voxels).

Context : About genericity of images

- 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), ...

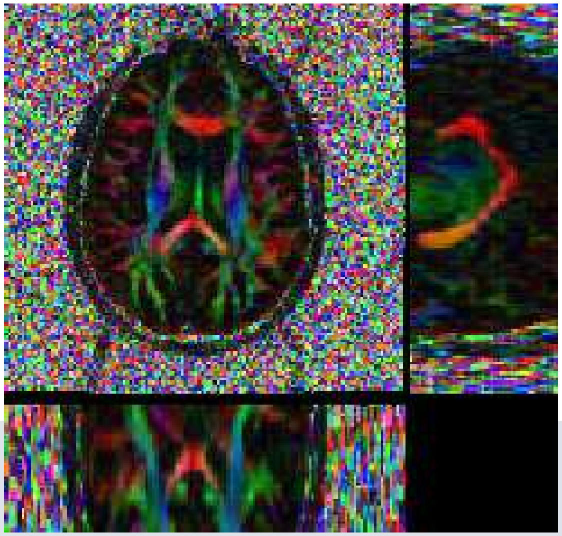




Context : About genericity of images

- 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, ...

(a) $I_1 : W \times H \rightarrow [0, 255]^3$



(b) $I_2 : W \times H \times D \rightarrow [0, 65535]^{32}$

(c) $I_3 : W \times H \times T \rightarrow [0, 4095]$

Context : About genericity of images

- 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.

Context : About genericity of images

- 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, ...

Context : About genericity of images

- 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 specific file formats :
  - PNG, JPEG, BMP, TIFF, TGA, DICOM, ANALYZE, ...

Context : About genericity of images

- Fact 4 : Image formats are just **“technical” solutions** for storing arrays of pixels. They hardly give informations about the image content itself.

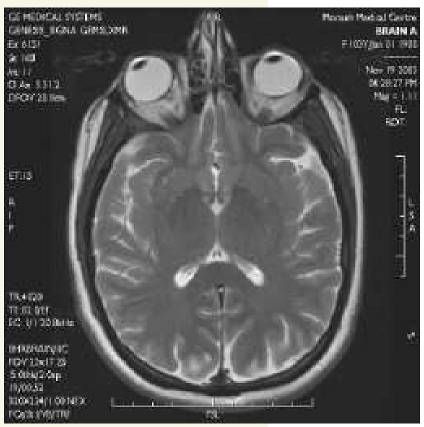
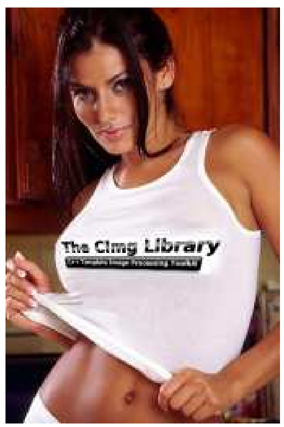
- Image processing and analysis is mainly about algorithms not input/output.

Context : About genericity of images

- Fact 4 : Image formats are just **“technical” solutions** for storing arrays of pixels. They hardly give informations about the image content itself.

- Image processing and analysis is mainly about algorithms not input/output.

- All images below are stored in PNG format :

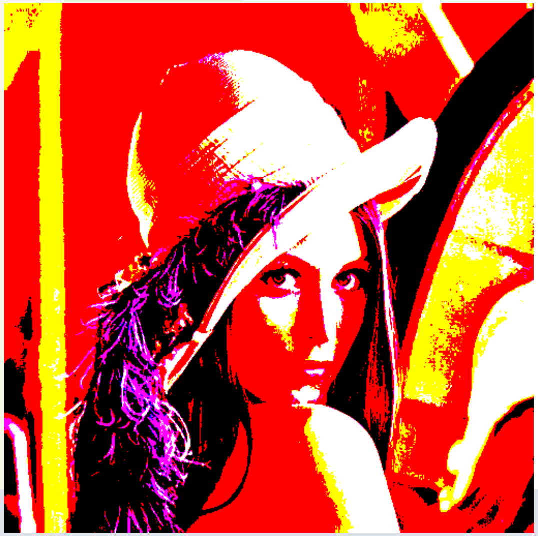

⇒ An image processing library/software should never be “attached” to a particular image format. Image formats are just a way to input/output pixel values.

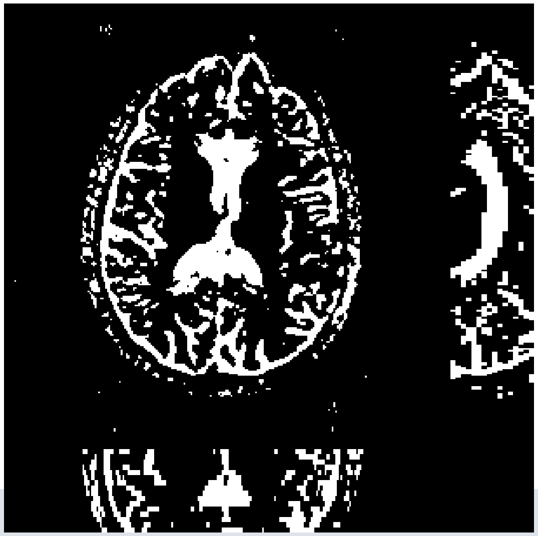
Context : About genericity of algorithms

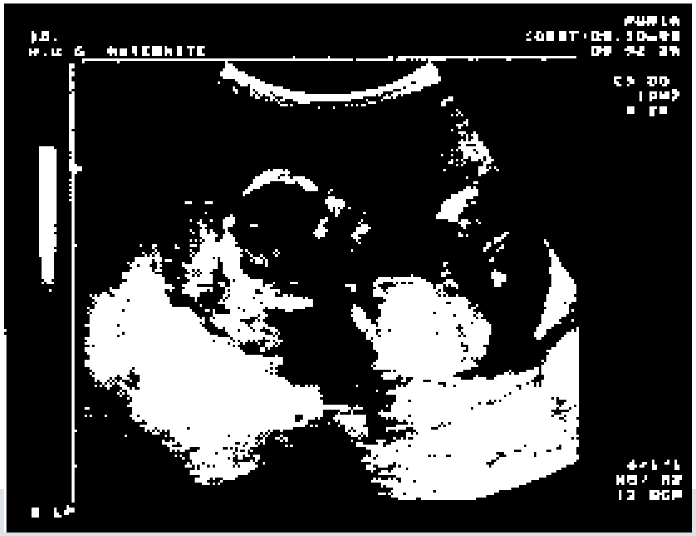
- Fact 5 : Most usual image processing algorithms are **image type independant**.

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

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







$\Rightarrow$ Implementing an image processing algorithm should be independant from the image format and coding.

CImg and G'MIC Philosophy

⇒ We propose **CImg** and **G'MIC**, two small image processing toolboxes based on these facts, which try to fit these constraints :

Usefulness Provides useful, classical and must-have algorithms.

Simplicity Easy to use, to understand, at two different scales (C++ and script).

Genericity Generic enough for a wide variety of applications (templates).

Portability Easy to spread from/to any computer (portable to various OS).

Freedom Distributed under Open-Source licenses

Talk outline

- **Context and Philosophy** : Research in Image Processing

⇒ **“Low-level” use (C++) : The CImg Library**

- **“Middle-level” use (script)** : G’MIC

- **“High-level” use** : Providing GUI, and results in real applications

- **What ?** : An open-source C++ library aiming to **simplify the development of image processing algorithms** for generic datasets (CeCILL-C License).

⇒ **Originally designed for algorithm prototyping.**

- **What ?** : An open-source C++ library aiming to **simplify the development of image processing algorithms** for generic datasets (CeCILL-C License).

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

⇒ **Not intended for C++ gurus.**

- **What ?** : An open-source C++ library aiming to **simplify the development of image processing algorithms** for generic datasets (CeCILL-C License).
- **For whom ?** : Designed for Researchers and Students in Image Processing and Computer Vision, having basic notions of C++.
- **How ?** : Defines a minimal set of C++ classes able to manipulate and process image datasets. Uses template mechanisms to handle pixel value genericity.

$\Rightarrow$ **Easy to apprehend.**

The CImg Library : Introduction

- **What ?** : An open-source C++ library aiming to **simplify the development of image processing algorithms** for generic datasets (CeCILL-C License).

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

- **How ?** : Defines a minimal set of C++ classes able to manipulate and process image datasets. Uses template mechanisms to handle pixel value genericity.

- **When ?** : Started in late 1999, the library is hosted on Sourceforge since December 2003 *(about 1200 visits and 100 downloads/day)*.

`http://cimg.sourceforge.net/`

The CImg Library

C++ Template Image Processing Toolkit

The CImg Library : Characteristics

CImg is lightweight and easy to use :

- **Easy to get :** CImg is distributed as a package (≈ 8.7 Mo) containing the library code (≈ 40000 lines), examples of use, documentations and resource files.

⇒ **Intended to remain small in the future.**

The CImg Library : Characteristics

CImg is lightweight and easy to use :

- **Easy to get :** CImg is distributed as a package (≈ 8.7 Mo) containing the library code (≈ 40000 lines), examples of use, documentations and resource files.

- **Easy to use :** Using CImg requires only the include of a single file :

```
#include "CImg.h"                // Just do that...
using namespace cimg_library; // ...and you are ready to go
```

⇒ **No complex installation required.**

CImg is lightweight and easy to use :

- **Easy to get :** CImg is distributed as a package ($\approx$ 8.7 Mo) containing the library code ($\approx$ 40000 lines), examples of use, documentations and resource files.

- **Easy to use :** Using CImg requires only the include of a single file :

```
#include "CImg.h"                // Just do that...
using namespace cimg_library; // ...and you are ready to go
```

- **Easy to understand :** It defines only four C++ classes :

```
CImg<T>, CImgList<T>, CImgDisplay, CImgException
```

$\Rightarrow$ **Easy to apprehend.**

The CImg Library : Characteristics

CImg is lightweight and easy to use :

- **Easy to get :** CImg is distributed as a package (≈ 8.7 Mo) containing the library code (≈ 40000 lines), examples of use, documentations and resource files.
- **Easy to use :** Using CImg requires only the include of a single file :

```
#include "CImg.h"                // Just do that...
using namespace cimg_library; // ...and you are 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(S)S, Keep it small and (stupidly) simple.**

CImg is (sufficiently) generic :

- CImg implements static genericity by using the C++ template mechanism.
  Keep-it-simple philosophy : One template parameter only !
  $\Longrightarrow$ the type of the image pixel (bool, char, int, float, ...).

CImg is (sufficiently) generic :

- CImg implements static genericity by using the C++ template mechanism.
  Keep-it-simple philosophy : One template parameter only !
  $\Longrightarrow$ 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).

The CImg Library : Characteristics

CImg is (sufficiently) generic :

- CImg implements static genericity by using the C++ template mechanism.
  Keep-it-simple philosophy : One template parameter only !
  $\Longrightarrow$ 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 sets of 4D images.

CImg is (sufficiently) generic :

- CImg implements static genericity by using the C++ template mechanism.
  Keep-it-simple philosophy : One template parameter only !
  $\Longrightarrow$ 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 sets of 4D images.

- ... **But**, CImg is **limited** to images defined on **regular rectangular grids**, and cannot handle image domains higher than 4 dimensions.

$\Rightarrow$ CImg covers actually most of the image types found in real world applications, while remaining understandable by non computer-geeks.

CImg wants to avoid too much genericity...

- CImg is generic, but wants to avoid quirks/difficulties encountered by hyper-generic libraries.

Generic Image Library Class Index

A | B | C | D | E | G | H | I | J | K | L | M | N | P | R | S | T | V | Y

alpha_t (boost::gil)
any_image (boost::gil)
any_image_view (boost::gil)
Assignable (boost::gil)

binary_operation_obj (boost::gil)
bit_aligned_image1_type (boost::gil)
bit_aligned_image2_type (boost::gil)
bit_aligned_image3_type (boost::gil)
bit_aligned_image4_type (boost::gil)
bit_aligned_image5_type (boost::gil)
bit_aligned_image_type (boost::gil)
bit_aligned_pixel_iterator (boost::gil)
bit_aligned_pixel_reference (boost::gil)
black_t (boost::gil)
blue_t (boost::gil)
byte_to_memunit (boost::gil)

channel_converter (boost::gil)
channel_converter_unsigned< bits32, bits32f > (boost::gil)
channel_converter_unsigned< bits32f, bits32 > (boost::gil)
channel_converter_unsigned< bits32f, DstChannelV > (boost::gil)
channel_converter_unsigned< T, T > (boost::gil)
channel_converter_unsigned_impl (boost::gil::detail)
channel_mapping_type< planar_pixel_reference< ChannelReference, ColorSpace > > (boost::gil)
channel_multiplier (boost::gil)
channel_multiplier_unsigned (boost::gil)

devicen_t< 3 > (boost::gil)
devicen_t< 4 > (boost::gil)
devicen_t< 5 > (boost::gil)
dynamic_xy_step_transposed_type (boost::gil)
dynamic_xy_step_type (boost::gil)

element_const_reference_type (boost::gil)
element_reference_type (boost::gil)
element_type (boost::gil)
equal_n_fn< boost::gil::iterator_from_2d< Loc >, I2 > (boost::gil::detail)
equal_n_fn< boost::gil::iterator_from_2d< Loc1 >, boost::gil::iterator_from_2d< Loc2 > > (boost::gil::detail)
equal_n_fn< const pixel< T, Cs > *, const pixel< T, Cs > * > (boost::gil::detail)
equal_n_fn< I1, boost::gil::iterator_from_2d< Loc > > (boost::gil::detail)
equal_n_fn< planar_pixel_iterator< IC, Cs >, planar_pixel_iterator< IC, Cs > > (boost::gil::detail)
EqualityComparable (boost::gil)

gray_color_t (boost::gil)
green_t (boost::gil)

HasDynamicXStepTypeConcept (boost::gil)
HasDynamicYStepTypeConcept (boost::gil)
HasTransposedTypeConcept (boost::gil)
homogeneous_color_base< Element, Layout, 1 > (boost::gil::detail)
homogeneous_color_base< Element, Layout, 2 > (boost::gil::detail)
homogeneous_color_base< Element, Layout, 3 > (boost::gil::detail)
homogeneous_color_base< Element, Layout, 4 > (boost::gil::detail)
homogeneous_color_base< Element, Layout, 5 > (boost::gil::detail)
HomogeneousColorBaseConcept (boost::gil)

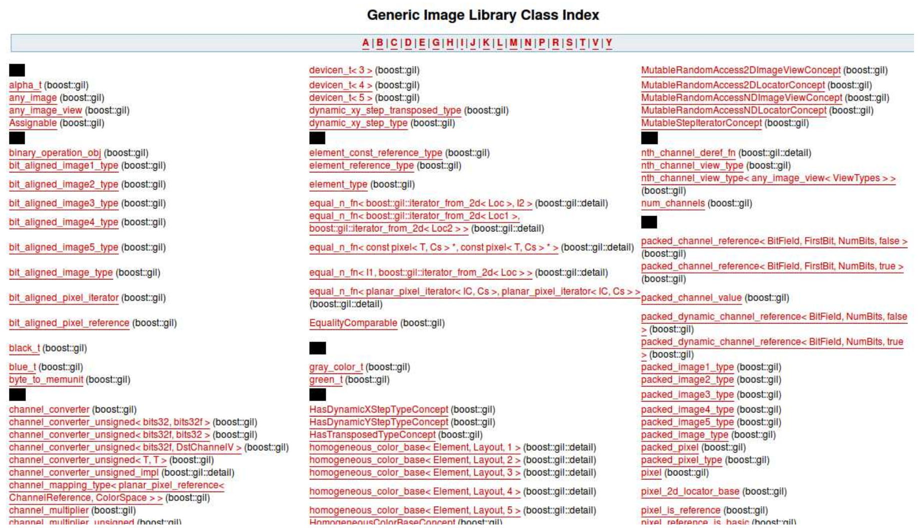
MutableRandomAccess2DImageViewConcept (boost::gil)
MutableRandomAccess2DLocatorConcept (boost::gil)
MutableRandomAccessNDImageViewConcept (boost::gil)
MutableRandomAccessNDLocatorConcept (boost::gil)
MutableStepIteratorConcept (boost::gil)

nth_channel_deref_fn (boost::gil::detail)
nth_channel_view_type (boost::gil)
nth_channel_view_type< any_image_view< ViewTypes > > (boost::gil)
num_channels (boost::gil)

packed_channel_reference< BitField, FirstBit, NumBits, false > (boost::gil)
packed_channel_reference< BitField, FirstBit, NumBits, true > (boost::gil)
packed_channel_value (boost::gil)
packed_dynamic_channel_reference< BitField, NumBits, false > (boost::gil)
packed_dynamic_channel_reference< BitField, NumBits, true > (boost::gil)
packed_image1_type (boost::gil)
packed_image2_type (boost::gil)
packed_image3_type (boost::gil)
packed_image4_type (boost::gil)
packed_image5_type (boost::gil)
packed_image_type (boost::gil)
packed_pixel (boost::gil)
packed_pixel_type (boost::gil)
pixel (boost::gil)
pixel_2d_locator_base (boost::gil)
pixel_is_reference (boost::gil)
pixel_reference_is_basic (boost::gil)

CImg wants to avoid too much genericity...

- CImg is generic, but wants to avoid quirks/difficulties encountered by hyper-generic libraries.



Generic Image Library Class Index

A | B | C | D | E | G | H | I | J | K | L | M | N | P | R | S | T | V | Y

alpha_t (boost::gil)
any_image (boost::gil)
any_image_view (boost::gil)
Assignable (boost::gil)

binary_operation_obj (boost::gil)
bit_aligned_image1_type (boost::gil)
bit_aligned_image2_type (boost::gil)
bit_aligned_image3_type (boost::gil)
bit_aligned_image4_type (boost::gil)
bit_aligned_image5_type (boost::gil)
bit_aligned_image_type (boost::gil)
bit_aligned_pixel_iterator (boost::gil)
bit_aligned_pixel_reference (boost::gil)
black_t (boost::gil)
blue_t (boost::gil)
byte_to_memunit (boost::gil)

channel_converter (boost::gil)
channel_converter_unsigned< bits32, bits32f > (boost::gil)
channel_converter_unsigned< bits32f, bits32 > (boost::gil)
channel_converter_unsigned< bits32f, DstChannelV > (boost::gil)
channel_converter_unsigned< T, T > (boost::gil)
channel_converter_unsigned_impl (boost::gil::detail)
channel_mapping_type< planar_pixel_reference< ChannelReference, ColorSpace > > (boost::gil)
channel_multiplier (boost::gil)

devicen_t< 3 > (boost::gil)
devicen_t< 4 > (boost::gil)
devicen_t< 5 > (boost::gil)
dynamic_xy_step_transposed_type (boost::gil)
dynamic_xy_step_type (boost::gil)

element_const_reference_type (boost::gil)
element_reference_type (boost::gil)
element_type (boost::gil)
equal_n_fn< boost::gil::iterator_from_2d< Loc >, I2 > (boost::gil::detail)
equal_n_fn< boost::gil::iterator_from_2d< Loc1 >, boost::gil::iterator_from_2d< Loc2 > > (boost::gil::detail)
equal_n_fn< const pixel< T, Cs > *, const pixel< T, Cs > * > (boost::gil::detail)
equal_n_fn< I1, boost::gil::iterator_from_2d< Loc > > (boost::gil::detail)
equal_n_fn< planar_pixel_iterator< IC, Cs >, planar_pixel_iterator< IC, Cs > > (boost::gil::detail)
EqualityComparable (boost::gil)

gray_color_t (boost::gil)
green_t (boost::gil)

HasDynamicXStepTypeConcept (boost::gil)
HasDynamicYStepTypeConcept (boost::gil)
HasTransposedTypeConcept (boost::gil)
homogeneous_color_base< Element, Layout, 1 > (boost::gil::detail)
homogeneous_color_base< Element, Layout, 2 > (boost::gil::detail)
homogeneous_color_base< Element, Layout, 3 > (boost::gil::detail)
homogeneous_color_base< Element, Layout, 4 > (boost::gil::detail)
homogeneous_color_base< Element, Layout, 5 > (boost::gil::detail)

MutableRandomAccess2DImageViewConcept (boost::gil)
MutableRandomAccess2DLocatorConcept (boost::gil)
MutableRandomAccessNDImageViewConcept (boost::gil)
MutableRandomAccessNDLocatorConcept (boost::gil)
MutableStepIteratorConcept (boost::gil)

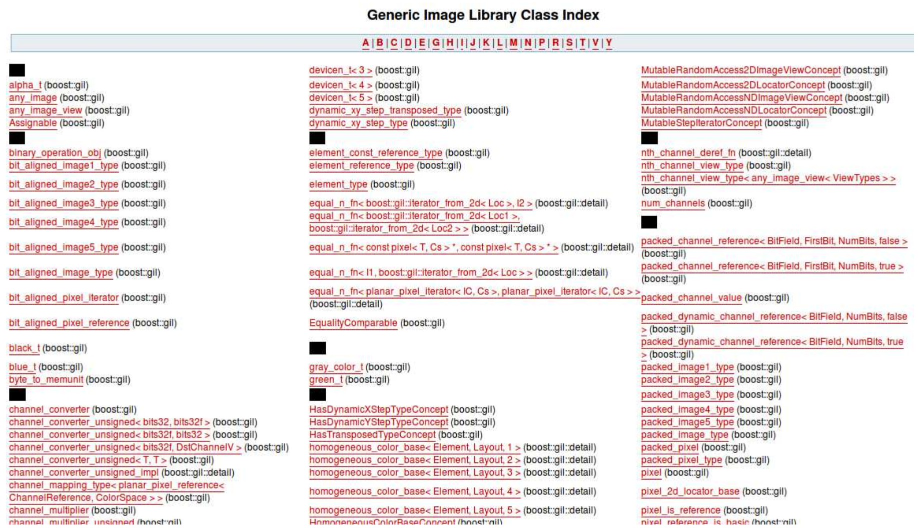
nth_channel_deref_fn (boost::gil::detail)
nth_channel_view_type (boost::gil)
nth_channel_view_type< any_image_view< ViewTypes > > (boost::gil)
num_channels (boost::gil)

packed_channel_reference< BitField, FirstBit, NumBits, false > (boost::gil)
packed_channel_reference< BitField, FirstBit, NumBits, true > (boost::gil)
packed_channel_value (boost::gil)
packed_dynamic_channel_reference< BitField, NumBits, false > (boost::gil)
packed_dynamic_channel_reference< BitField, NumBits, true > (boost::gil)
packed_image1_type (boost::gil)
packed_image2_type (boost::gil)
packed_image3_type (boost::gil)
packed_image4_type (boost::gil)
packed_image5_type (boost::gil)
packed_image_type (boost::gil)
packed_pixel (boost::gil)
packed_pixel_type (boost::gil)
pixel (boost::gil)
pixel_2d_locator_base (boost::gil)
pixel_is_reference (boost::gil)

- Discouraging for any average C++ programmers !! (i.e. most researchers....).

CImg wants to avoid too much genericity...

- CImg is generic, but wants to avoid quirks/difficulties encountered by hyper-generic libraries.



Generic Image Library Class Index

A | B | C | D | E | G | H | I | J | K | L | M | N | P | R | S | T | V | Y

alpha_t (boost::gil)
any_image (boost::gil)
any_image_view (boost::gil)
Assignable (boost::gil)

binary_operation_obj (boost::gil)
bit_aligned_image1_type (boost::gil)
bit_aligned_image2_type (boost::gil)
bit_aligned_image3_type (boost::gil)
bit_aligned_image4_type (boost::gil)
bit_aligned_image5_type (boost::gil)
bit_aligned_image_type (boost::gil)
bit_aligned_pixel_iterator (boost::gil)
bit_aligned_pixel_reference (boost::gil)
black_t (boost::gil)
blue_t (boost::gil)
byte_to_memunit (boost::gil)

channel_converter (boost::gil)
channel_converter_unsigned< bits32, bits32f > (boost::gil)
channel_converter_unsigned< bits32f, bits32 > (boost::gil)
channel_converter_unsigned< bits32f, DstChannelV > (boost::gil)
channel_converter_unsigned< T, T > (boost::gil)
channel_converter_unsigned_impl (boost::gil::detail)
channel_mapping_type< planar_pixel_reference< ChannelReference, ColorSpace > > (boost::gil)
channel_multiplier (boost::gil)
channel_multiplier_unsigned (boost::gil)

devicen_t< 3 > (boost::gil)
devicen_t< 4 > (boost::gil)
devicen_t< 5 > (boost::gil)
dynamic_xy_step_transposed_type (boost::gil)
dynamic_xy_step_type (boost::gil)

element_const_reference_type (boost::gil)
element_reference_type (boost::gil)
element_type (boost::gil)
equal_n_fn< boost::gil::iterator_from_2d< Loc >, I2 > (boost::gil::detail)
equal_n_fn< boost::gil::iterator_from_2d< Loc1 >, boost::gil::iterator_from_2d< Loc2 > > (boost::gil::detail)
equal_n_fn< const pixel< T, Cs > *, const pixel< T, Cs > * > (boost::gil::detail)
equal_n_fn< I1, boost::gil::iterator_from_2d< Loc > > (boost::gil::detail)
equal_n_fn< planar_pixel_iterator< IC, Cs >, planar_pixel_iterator< IC, Cs > > (boost::gil::detail)
EqualityComparable (boost::gil)

gray_color_t (boost::gil)
green_t (boost::gil)

HasDynamicXStepTypeConcept (boost::gil)
HasDynamicYStepTypeConcept (boost::gil)
HasTransposedTypeConcept (boost::gil)
homogeneous_color_base< Element, Layout, 1 > (boost::gil::detail)
homogeneous_color_base< Element, Layout, 2 > (boost::gil::detail)
homogeneous_color_base< Element, Layout, 3 > (boost::gil::detail)
homogeneous_color_base< Element, Layout, 4 > (boost::gil::detail)
homogeneous_color_base< Element, Layout, 5 > (boost::gil::detail)
HomogeneousColorBaseConcept (boost::gil)

MutableRandomAccess2DImageViewConcept (boost::gil)
MutableRandomAccess2DLocatorConcept (boost::gil)
MutableRandomAccessNDImageViewConcept (boost::gil)
MutableRandomAccessNDLocatorConcept (boost::gil)
MutableStepIteratorConcept (boost::gil)

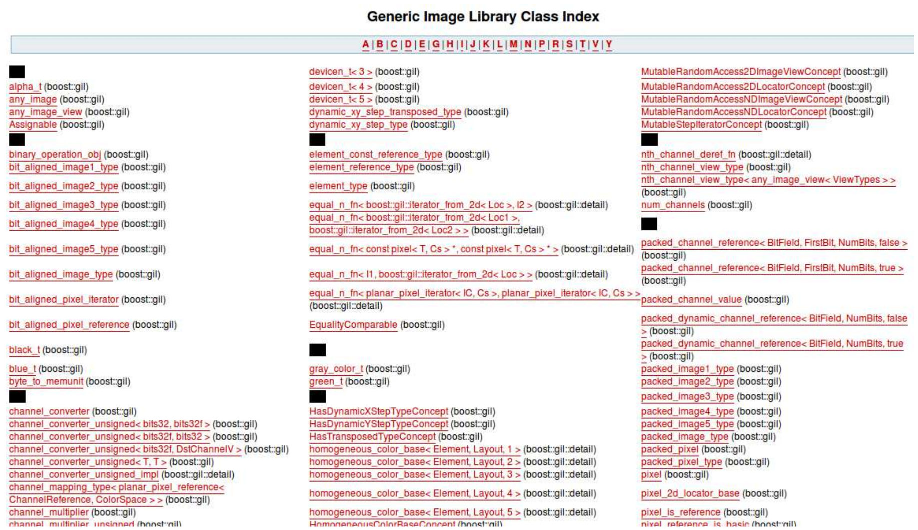
nth_channel_deref_fn (boost::gil::detail)
nth_channel_view_type (boost::gil)
nth_channel_view_type< any_image_view< ViewTypes > > (boost::gil)
num_channels (boost::gil)

packed_channel_reference< BitField, FirstBit, NumBits, false > (boost::gil)
packed_channel_reference< BitField, FirstBit, NumBits, true > (boost::gil)
packed_channel_value (boost::gil)
packed_dynamic_channel_reference< BitField, NumBits, false > (boost::gil)
packed_dynamic_channel_reference< BitField, NumBits, true > (boost::gil)
packed_image1_type (boost::gil)
packed_image2_type (boost::gil)
packed_image3_type (boost::gil)
packed_image4_type (boost::gil)
packed_image5_type (boost::gil)
packed_image_type (boost::gil)
packed_pixel (boost::gil)
packed_pixel_type (boost::gil)
pixel (boost::gil)
pixel_2d_locator_base (boost::gil)
pixel_is_reference (boost::gil)
pixel_reference_is_basic (boost::gil)

- Discouraging for any average C++ programmers !! (i.e. most researchers....).
- What if I want to contribute with non-generic algorithms ?

CImg wants to avoid too much genericity...

- CImg is generic, but wants to avoid quirks/difficulties encountered by hyper-generic libraries.



- Discouraging for any average C++ programmers !! (i.e. most researchers....).
- What if I want to contribute with non-generic algorithms ?
- Several API levels required to get both enough genericity and usability.

⇒ **Requires too much development efforts, regarding the benefits.**

- Link to the documentation web page.

The CImg Library : Characteristics

CImg is multi-platform and extensible :

- CImg does not depend on many libraries.
  It can be compiled only with the standard C++ libraries
  (useful for embedded architectures).

- Successfully tested platforms : Win32, Linux, Solaris, *BSD, Mac OS X.

- CImg is extensible : External tools or libraries may be used to improve CImg capabilities (ImageMagick, XMedcon, libpng, libjpeg, libtiff, libfftw3...), these tools being freely available for any platform.

- CImg defines a simple plug-in mechanism to easily add your own functions to the library core.

The CImg Library : Characteristics

Last but not least, CImg is **very useful** on a daily basis !

- CImg is able to read/write different image formats.
- CImg has lot of classical algorithms for image processing.
- CImg has an integrated parser of mathematical expressions.
- CImg has an integrated renderer of 3D objects.
- CImg has methods dedicated to data visualization.
- CImg has structure and methods to quickly create interactive windows.
- CImg is small and modulable enough to be integrated everywhere.

The CImg Library : Code example (1/5)

```
#include "CImg.h"
using namespace cimg_library;
int main() {
  const CImg<float>
    img1(256,256),
    img2("milla.bmp"),
    img3(256,256,1,3,"128 + 128*cos(x*y*(1+c)/40)",true);
  (img1,img2,img3).display();
  return 0;
}
```

The CImg Library : Code example (2/5)

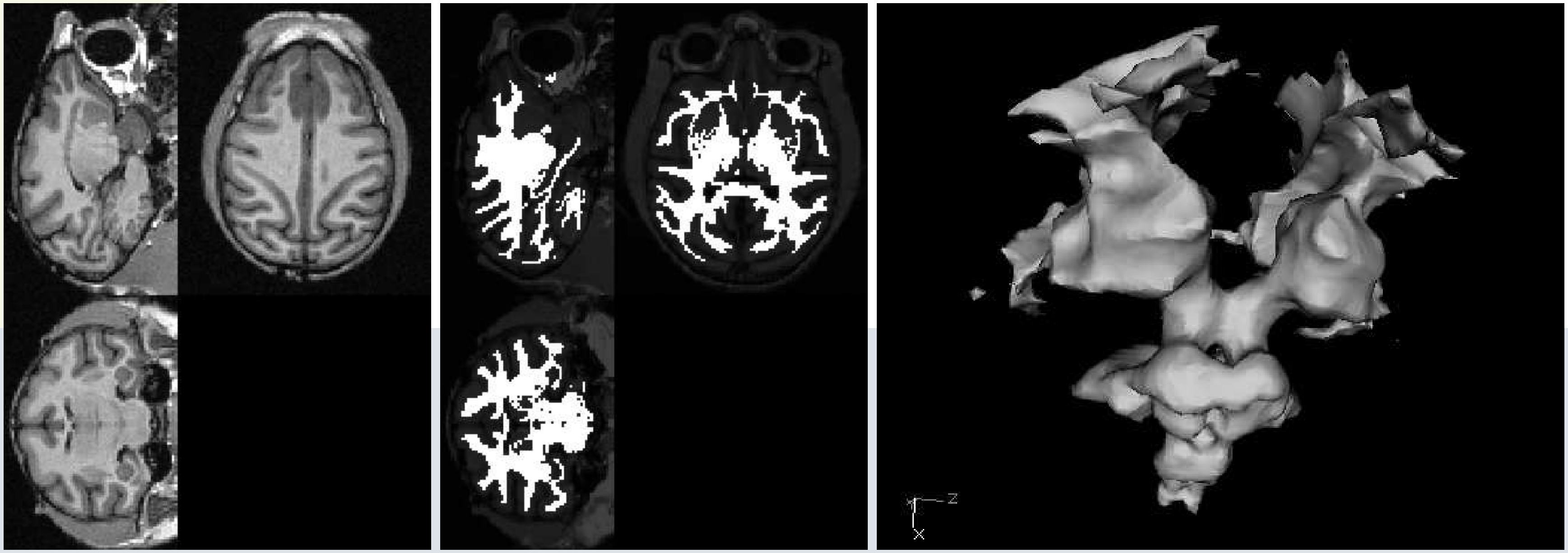
```
#include "CImg.h"
using namespace cimg_library;
int main(int argc,char **argv) {
  const CImg<>
    img("milla.bmp"),
    res = (img + "128 + 128*cos(x*y*(1+c)/40)).normalize(0,255);
  (img,res).display();
  return 0;
}
```

The CImg Library : Code example (3/5)

```
#include "CImg.h"
using namespace cimg_library;
int main(int argc,char **argv) {
  const CImg<>
    img("milla.bmp"),
    colors = CImg<>::contrast_LUT8(),
    res = img.get_norm().blur(1).quantize(4).label_regions().map(colors);
  (img,res).display();
  return 0;
}
```

The CImg Library : Code example (4/5)

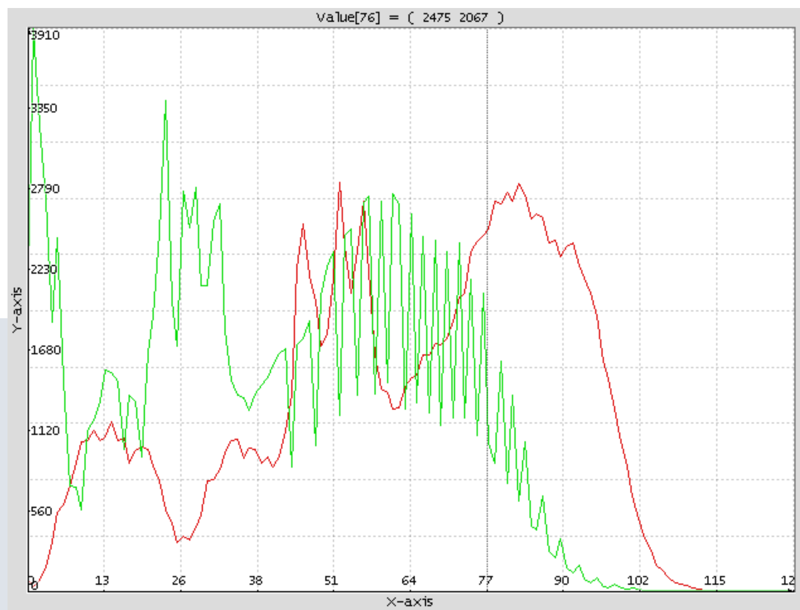
```
#include "CImg.h"
using namespace cimg_library;
int main(int argc,char **argv) {
  CImg<> img("reference.inr");
  CImgDisplay disp(img,"3D volume");
  float color[1] = { 1000 };
  CImgList<unsigned int> faces3d;
  const CImg<> points3d = img.draw_fill(17,58,39,color,1,30).blur(1).get_isosurface3d(faces3d,900);
  CImg<unsigned char>().display_object3d("3D brain",points3d,faces3d);
  return 0;
}
```



The CImg Library : Code example (5/5)

```
#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);
  CImgDisplay disp((img,img2),"Images");
  (hist,hist2).get_append('v').display_graph("Histograms");
  return 0;
}
```





More about the single-file library structure

- The core of the CImg code is all contained in a single header file *CImg.h*.
- It defies classical programming rules in C++ ! **Are the developers stupid ?**

More about the single-file library structure

- The core of the CImg code is all contained in a single header file *CImg.h*.

⇒ This is one technical solution to fit with technical constraints.

More about the single-file library structure

- The core of the CImg code is all contained in a single header file *CImg.h*.

⇒ This is one technical solution to fit with technical constraints.

- Question 1 :
  Why not having a classical library structure based on **header & binary object ?**

More about the single-file library structure

- The core of the CImg code is all contained in a single header file *CImg.h*.

⇒ This is one technical solution to fit with technical constraints.

- Question 1 :
  Why not having a classical library structure based on **header & binary object ?**

⇒ Because the library uses templates : The template type of used instanciated objects are only known during the compilation phase, so one cannot anticipate the types of the functions that will be required to compile one particular code.

⇒ It is quite common in C++ to put implementations of generic functions in headers (e.g. STL).

More about the single-file library structure

- The core of the CImg code is all contained in a single header file *CImg.h*.

⇒ This is one technical solution to fit with technical constraints.

- Question 2 :
  The number of possible template types are actually limited *(bool, char, float, ...)*.
  Why not compiling CImg methods for all possible types as an object/library file ?

More about the single-file library structure

- The core of the CImg code is all contained in a single header file *CImg.h*.

⇒ This is one technical solution to fit with technical constraints.

- Question 2 :
  The number of possible template types are actually limited *(bool, char, float, ...)*.
  Why not compiling CImg methods for all possible types as an object/library file ?

⇒ Because it would be useless : Usually, less than 10% of the CImg methods are used in a given program. Compilers know how to avoid compilation of unused functions in the final binary code, so it remains small.

More about the single-file library structure

- The core of the CImg code is all contained in a single header file *CImg.h*.

⇒ This is one technical solution to fit with technical constraints.

- Question 2 :
  The number of possible template types are actually limited *(bool, char, float, ...)*.
  Why not compiling CImg methods for all possible types as an object/library file ?

⇒ Because it would be useless : Usually, less than 10% of the CImg methods are used in a given program. Compilers know how to avoid compilation of unused functions in the final binary code, so it remains small.

⇒ Because it would be huge : CImg methods often take one or several template images as parameters (e.g. `CImg<T>::draw_image()`). Combinatorially speaking, the number of functions to be compiled is gigantic.

More about the single-file library structure

- The core of the CImg code is all contained in a single header file *CImg.h*.

⇒ This is one technical solution to fit with technical constraints.

- Question 3 :

  But, why having a big single file with all classes/namespaces inside ? Why not splitting it as one header per class ?

More about the single-file library structure

- The core of the CImg code is all contained in a single header file *CImg.h*.

⇒ This is one technical solution to fit with technical constraints.

- Question 3 :
  But, why having a big single file with all classes/namespaces inside ? Why not splitting it as one header per class ?

⇒ Because CImg classes and namespaces are interdependent : Any code would require the systematic inclusion of all these headers.

- This interdependence is due to the fact that algorithms are methods of the CImg classes. It is not possible to apply an algorithm on another container (as the STL is able to do for instance).

More about the single-file library structure

- The core of the CImg code is all contained in a single header file *CImg.h*.

⇒ This is one technical solution to fit with technical constraints.

- Question 3 :
  But, why having a big single file with all classes/namespaces inside ? Why not splitting it as one header per class ?

⇒ Because CImg classes and namespaces are interdependent : Any code would require the systematic inclusion of all these headers.

⇒ Because CImg is a small toolkit and will remain as it. It contains only classical image processing and does not intend to blow-up with billions of different algorithms.

More about the single-file library structure

- The core of the CImg code is all contained in a single header file *CImg.h*.

⇒ This is one technical solution to fit with technical constraints.

- Question 3 :
  But, why having a big single file with all classes/namespaces inside ? Why not splitting it as one header per class ?

⇒ Because CImg classes and namespaces are interdependent : Any code would require the systematic inclusion of all these headers.

⇒ Because CImg is a small toolkit and will remain as it. It contains only classical image processing and does not intend to blow-up with billions of different algorithms.

⇒ Because splitting a header file in several parts does not speed-up the compilation process, nor ease the maintenance or add clarity to the source code.

More about the single-file library structure

- The core of the CImg code is all contained in a single header file *CImg.h*.

⇒ This is one technical solution to fit with technical constraints.

- Question 4 :
  So, you don't allow algorithm reusability in a generic library. Isn't it a design flaw ?

More about the single-file library structure

- The core of the CImg code is all contained in a single header file *CImg.h*.

⇒ This is one technical solution to fit with technical constraints.

- Question 4 :
  So, you don't allow algorithm reusability in a generic library. Isn't it a design flaw ?

⇒ Not at all. Different genericity levels can be considered : Genericity can be focused on structures, algorithms, or both. CImg does not propose generic algorithms, but algorithms working on a given set of generic structures.

More about the single-file library structure

- The core of the CImg code is all contained in a single header file *CImg.h*.

⇒ This is one technical solution to fit with technical constraints.

- Question 4 :
  So, you don't allow algorithm reusability in a generic library. Isn't it a design flaw ?

⇒ Not at all. Different genericity levels can be considered : Genericity can be focused on structures, algorithms, or both. CImg does not propose generic algorithms, but algorithms working on a given set of generic structures.

⇒ **Simplicity** : Having algorithms as methods allows us to write code as :

```
img.blur(3).mirror('x').rotate(90).save(“foo.jpg”);
```

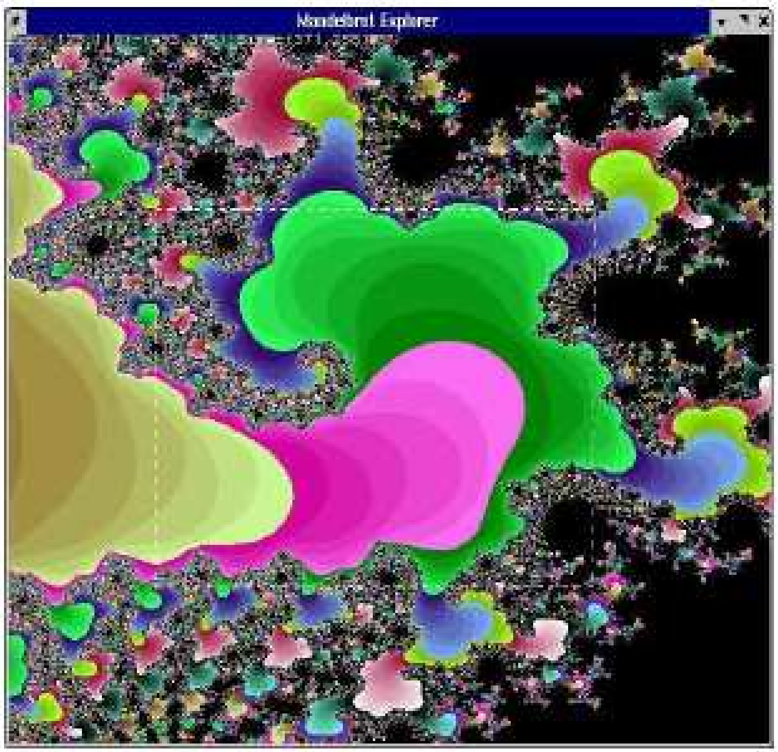
instead of

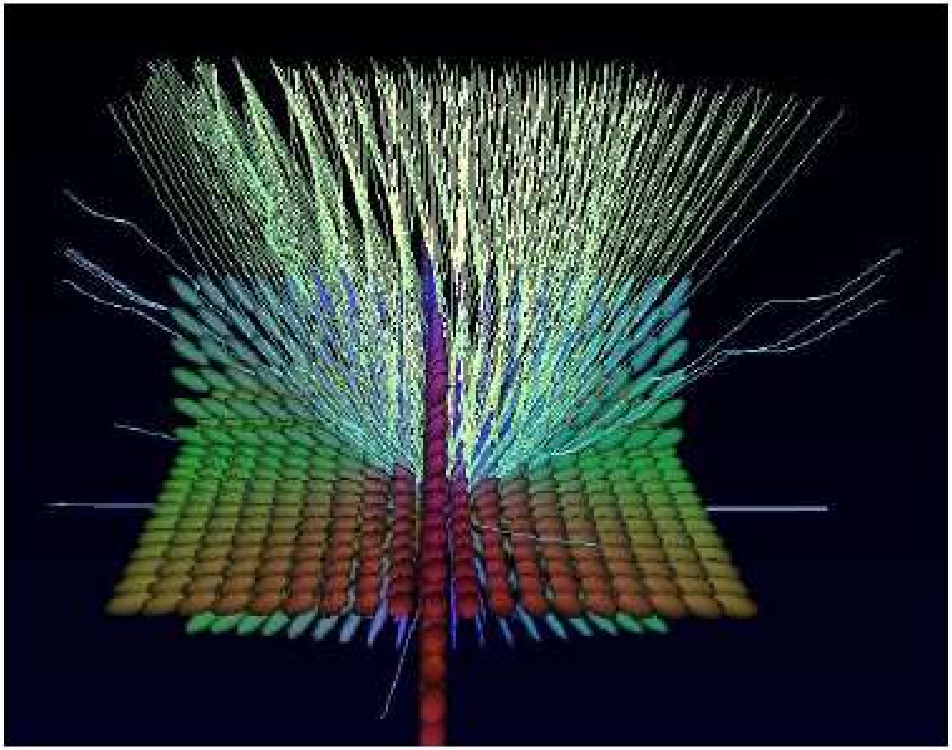
```
save(rotate(mirror(blur(img,3),'x'),90),”foo.jpg”);
```

Live demo

- Here is a quick live demo of CImg.

  It illustrates some of the important characteristics of the CImg Library.





Talk outline

- **Context and Philosophy** : Research in Image Processing
- **“Low-level” use (C++)** : The CImg Library

⇒ **“Middle-level” use (script) : G’MIC**

- **“High-level” use** : Providing GUI, and results in real applications

- Observation 1 : CImg requires (basic) C++ knowledge.

  It eases the implementation of image algorithms *from scratch*, but is still hardly usable by non C++ programmers.

G'MIC : Motivations

- Observation 1 : CImg requires (basic) C++ knowledge.
  It eases the implementation of image algorithms *from scratch*, but is still hardly usable by non C++ programmers.

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

- Observation 1 : CImg requires (basic) C++ knowledge.
  It eases the implementation of image algorithms *from scratch*, but is still hardly usable by non C++ programmers.

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

- Observation 3 : It is not convenient to create C++ programs specifically for this task (requires code edition, compilation time, ...).

- Observation 1 : CImg requires (basic) C++ knowledge.
  It eases the implementation of image algorithms *from scratch*, but is still hardly usable by non C++ programmers.

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

- Observation 3 : It is not convenient to create C++ programs specifically for this task (requires code edition, compilation time, ...).

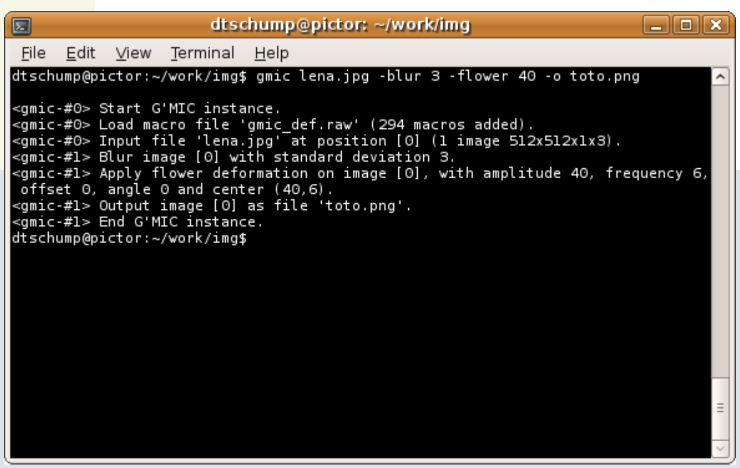
⇒ G'MIC defines a script language which interfaces the CImg functionalities.

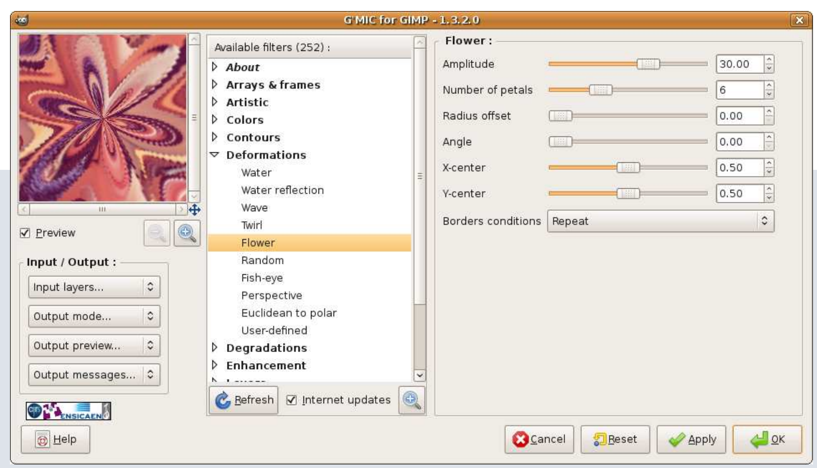
⇒ No compilation required, most of the CImg features available.

⇒ G'MIC is a **"middle-scale"** tool for image processing.

G'MIC : Language properties

- G'MIC input/outputs are lists of numbered images (eq. to `CImgList<T>`).
- Each G'MIC instruction runs an image processing algorithm, or control the program execution : `-blur, -rgb2hsv, -isosurface3d, -if, -endif ...`
- A G'MIC program is interpreted as successive calls of CImg methods.
- Custom G'MIC functions can be written and recognized by the interpreter.
- The G'MIC interpreter can be called from the command line of from any external project (provided as a library).



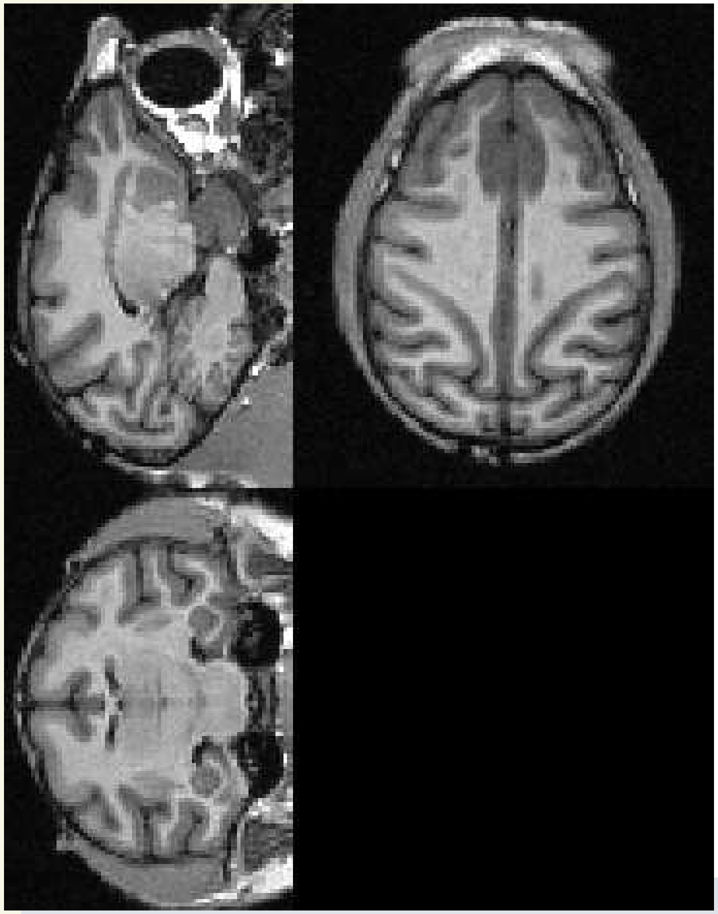


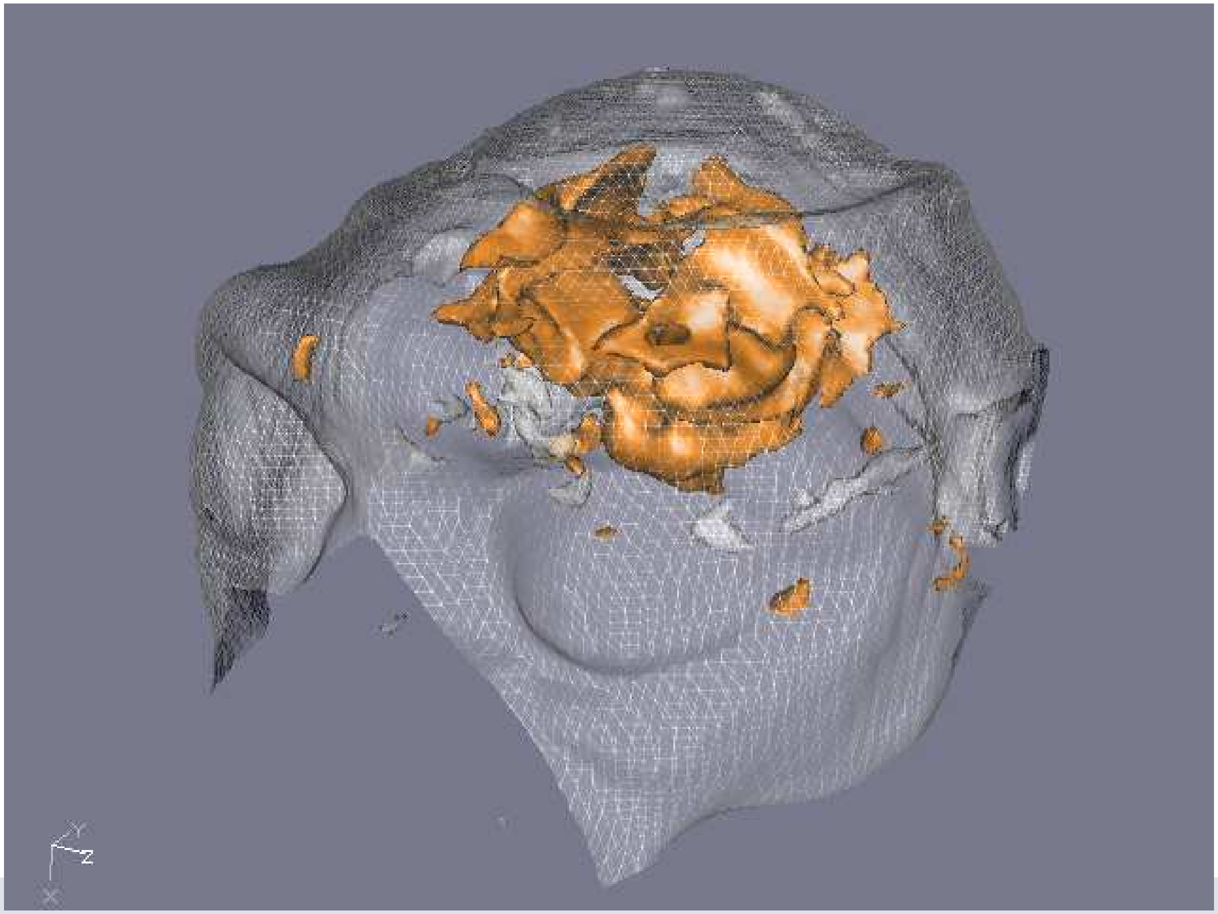
G’MIC : Examples of use (1/6)

```
gmic ~/work/img/lena.jpg -blur 3 -sharpen 1000 -noise 30 -+ "'cos(x/3)*30'"
```

G'MIC : Examples of use (2/6)

```
gmic reference.inr --flood 23,53,30,50,1,1000 -flood[-2] 0,0,0,30,1,1000 -blur 1 -isosurface3d 900
     -o3d[-2] 0.2 -color3d[-1] 255,128,0 -+3d
```



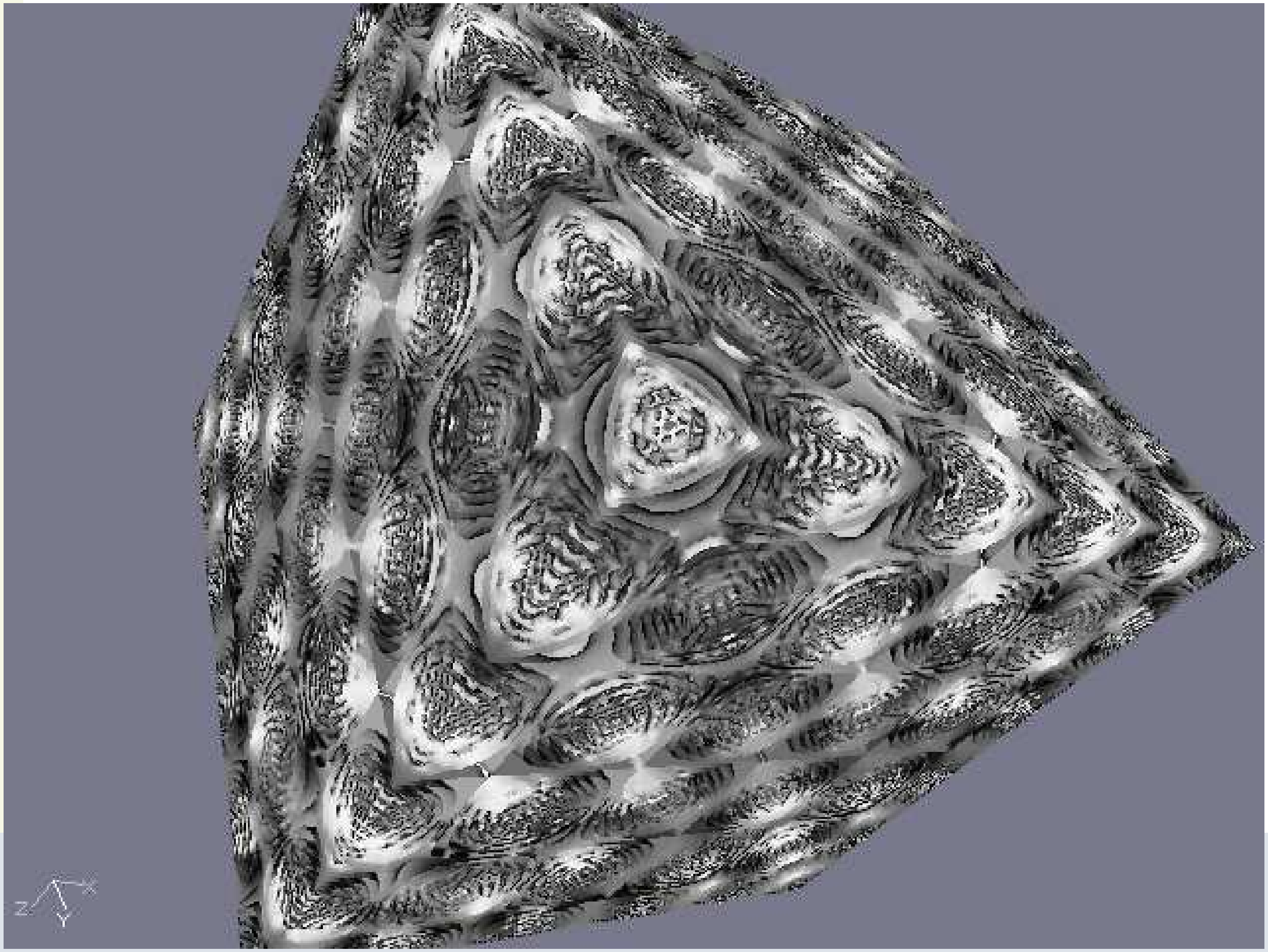


G'MIC : Examples of use (3/6)

```
gmic tunis.jpg -repeat 4 -smooth 30 -done -o tunis2.jpg
```

G’MIC : Examples of use (4/6)

```
gmic -isosurface3d "'sin(x*y*z)'",0,-10,-10,-10,10,10,10,128,128,64
```



G'MIC : Examples of use (5/6)

```
gmic lena.jpg -pencilbw 0.3 -o gmic_lena1.jpg;    gmic lena.jpg -cubism 160 -o gmic_lena3.jpg
gmic lena.jpg -flower 10 -o gmic_lena4.jpg;       gmic lena.jpg -stencibw 30 -o gmic_lena2.jpg
```

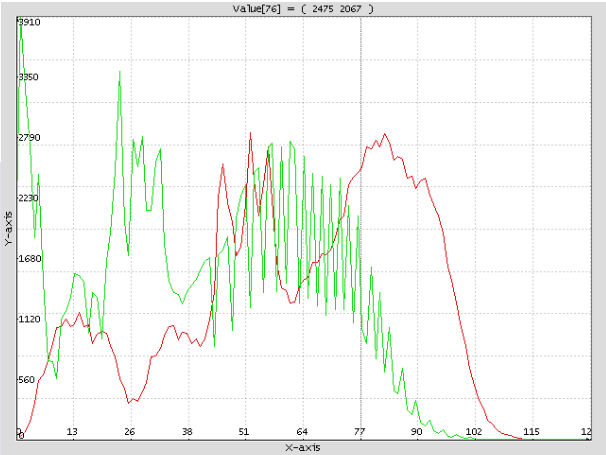
G’MIC : Examples of use (6/6)

```
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;
}
```



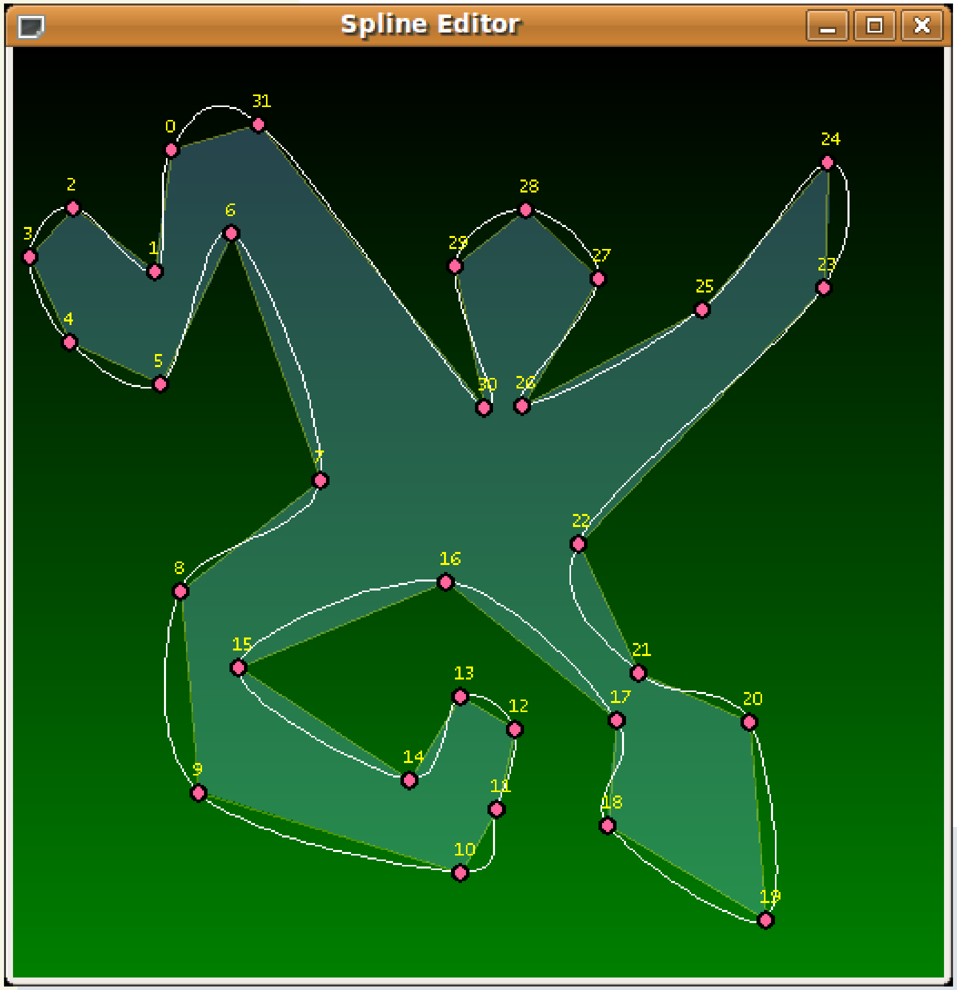


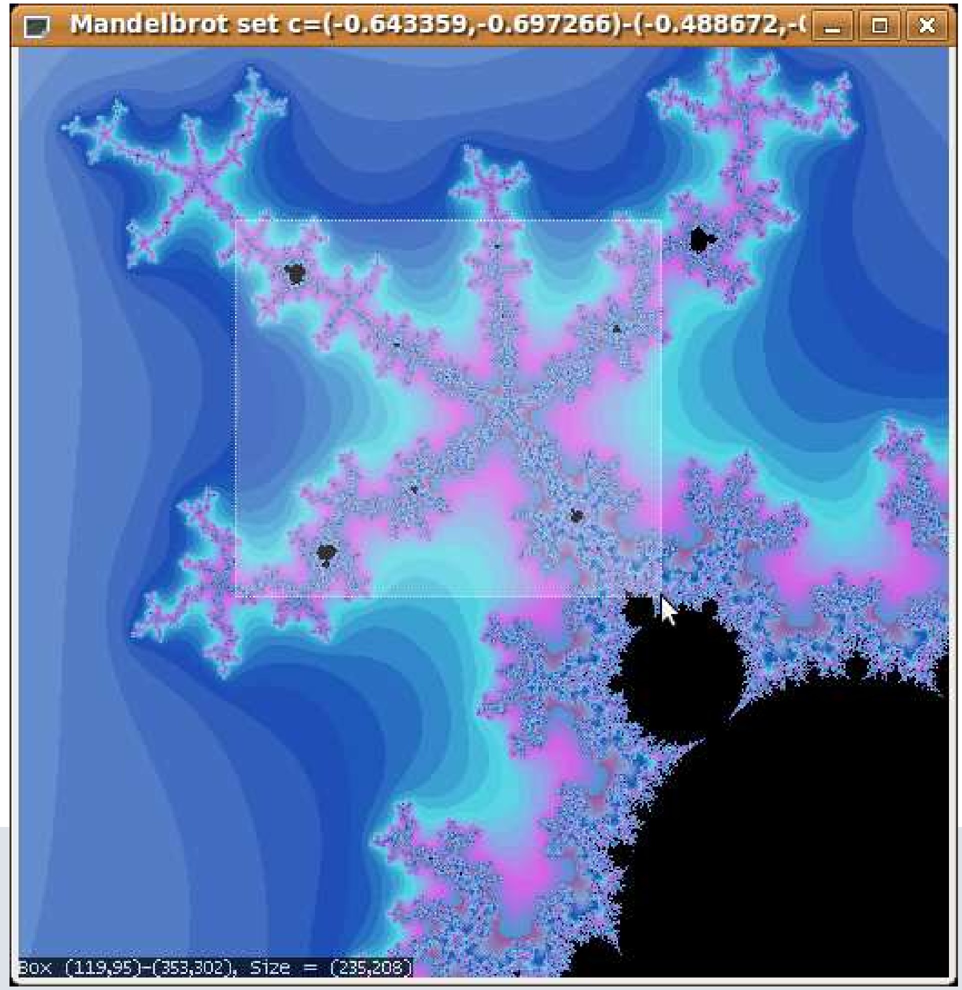
Writting a complex G'MIC function : Fish-Eye demo

```
-v- -type float
-if {@#>0} -a x -n 0,255 -r2dy 220 -else
  120,90,1,3 -rand[-1] 0,255 -plasma[-1] 0.3,3 -n 0,255
  -text "  G'MIC\nFISH-EYE\n  DEMO",15,13,24,1,255 -resize2x -blur 5 -sharpen 1000
  -f i+150-4*abs(y-h/2) -c[-1] 0,255 -frame_fuzzy[-1] 15,10,15,1.5,0 -to_rgb[-1]
-endif
-torus3d 20,6 -col3d[-1] {?(30,255)},{?(30,255)},{?(30,255)} --rot3d[-1] 1,0,0,90
-col3d[-1] {?(30,255)},{?(30,255)},{?(30,255)} -+3d[-1] 15 -+3d[-2,-1] -db3d 0 -c3d[-1]
-p[0] 30 -w[-2] {2*@{-2,w}},{2*@{-2,h}},0,0
-repeat 100000
  -wait 40
  -if {@{!,b}==1} -p[0] {min(80,@{*,0}+8)} -pp[1] -endif
  -if {@{!,b}==2} -p[0] {max(3,@{*,0}-8)} -pp[1] -endif
  --object3d[-2] [-1],{50+30*cos(@{>,-1}/20)}%,{50+30*sin(@{>,-1}/31)}%,{50+330*sin(@{>,-1}/19)},0.7,0
  -rot3d[-2] 1,0.2,0.6,3
  -if {@{!,x}>=0}
  -fish_eye[-1] {@{!,x}*100/@{!,w}},{@{!,y}*100/@{!,h}},@{*,0}
  -endif
  -name[-1] "Fish-Eye Demo" -w[-1] -rm[-1]
  -if {"@!==0 || @{!,ESC} || @{!,Q}"} -rm[-2,-1] -pp[0] -w 0 -v+ -return -endif
-done
```

More G'MIC scripts : Mandelbrot Explorer and Spline Editor

```
gmic -x_spline
gmic -x_mandelbrot
```





Talk outline

- **Context and Philosophy** : Research in Image Processing
- **“Low-level” use (C++)** : The CImg Library
- **“Middle-level” use (script)** : G’MIC

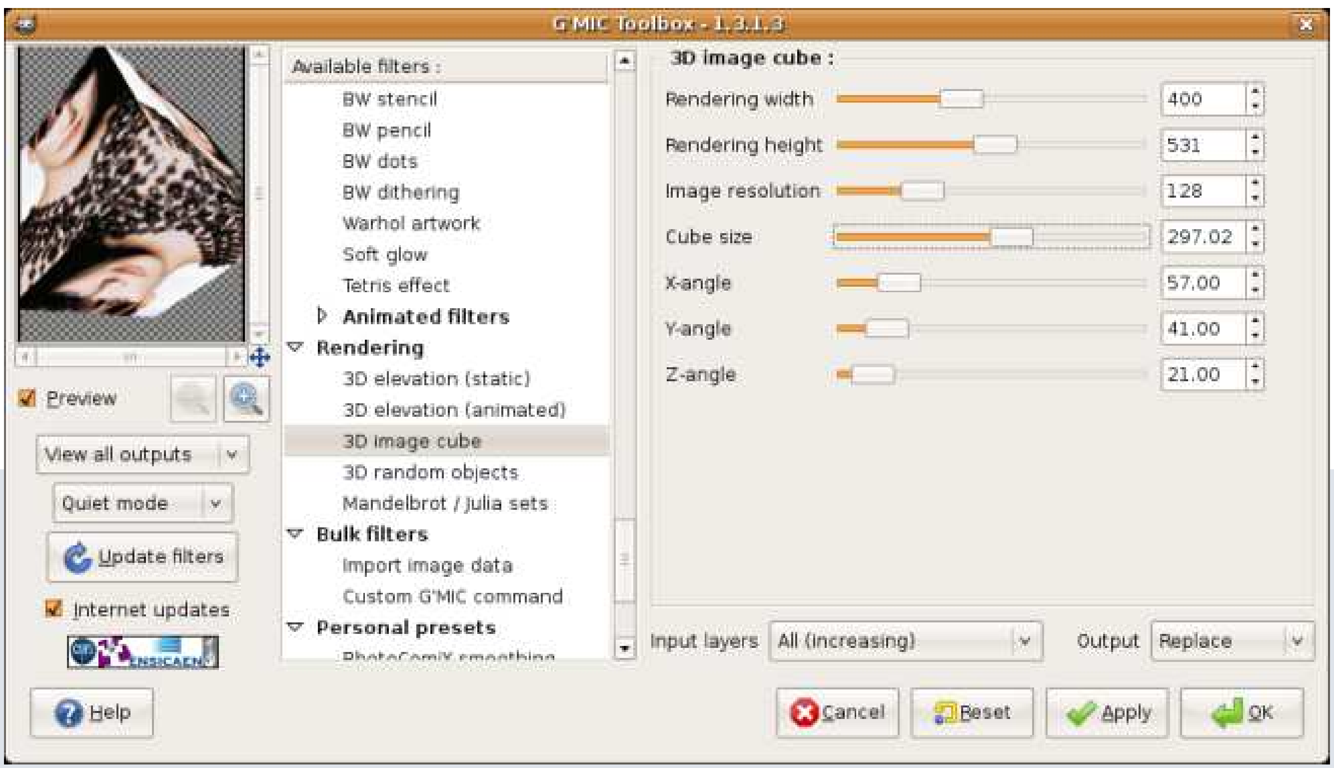
⇒ **“High-level” use : Providing GUI, and results in real applications**

G'MIC : Plug-in for GIMP

- GIMP is an open-source image retouching software with plug-in capabilities.
- The G'MIC interpreter has been embedded in a plug-in for GIMP.

⇒ All G'MIC functionalities are available directly from the GIMP interface.

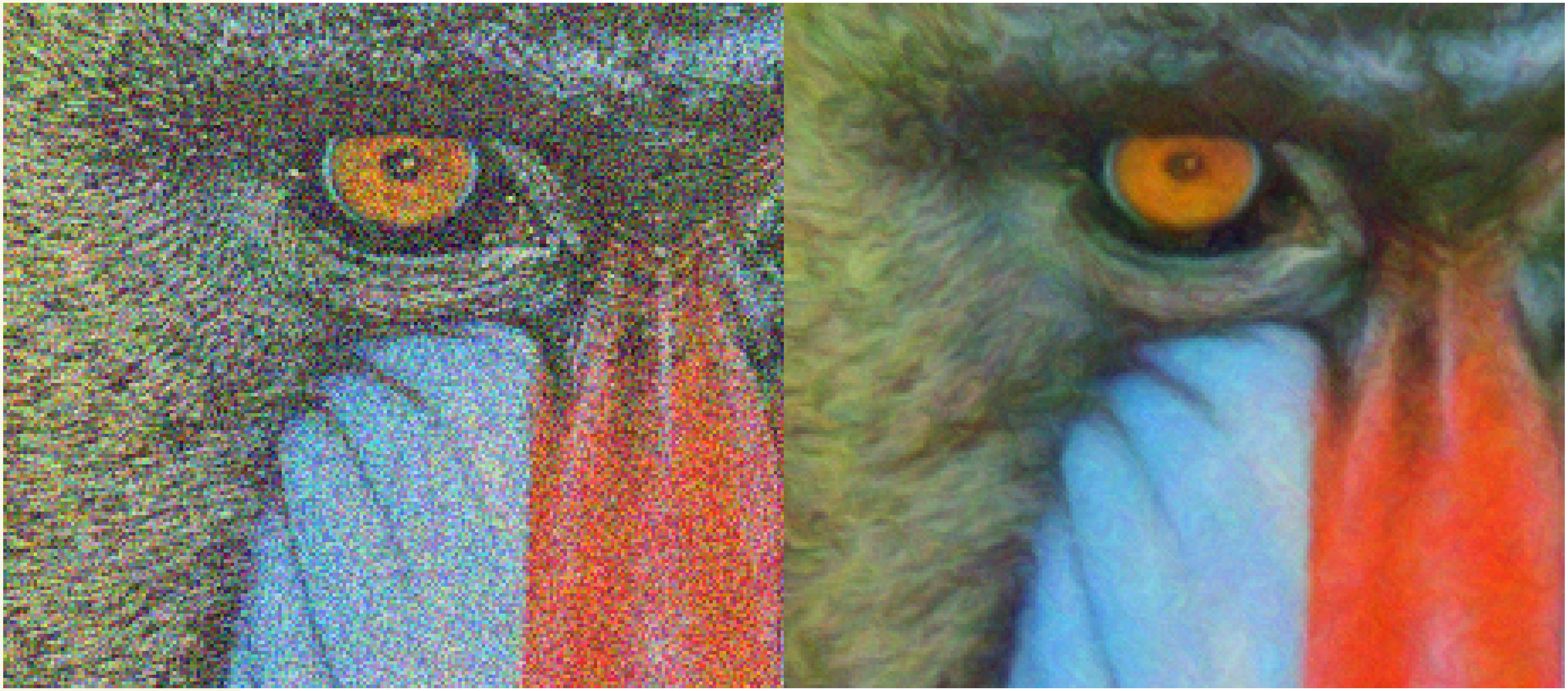
⇒ Management of multiple image input/output via the image layers.



Two CImg applications

on different modalities

Application of `CImg<T>::blur_anisotropic()`



“Babouin” (détail) - 512x512 - (1 iter., 19s)

Application of `CImg<T>::blur_anisotropic()`

“Tunisie” - 555x367

Application of `CImg<T>::blur_anisotropic()`

“Tunisie” - 555x367 - (1 iter., 11s)

Application of `CImg<T>::blur_anisotropic()`

“Tunisie” - 555x367 - (1 iter., 11s)

Application of `CImg<T>::blur_anisotropic()`

“Bébé” - 400x375

Application of `CImg<T>::blur_anisotropic()`

“Bébé” - 400x375 - (2 iter, 5.8s)

Application of `CImg<T>::blur_anisotropic()`

"Bébé" - 400x375 - (2 iter, 5.8s)

Application of `CImg<T>::blur_anisotropic()`

"Van Gogh"

Application of `CImg<T>::blur_anisotropic()`

“Van Gogh” - (1 iter, 5.122s).

Application of `CImg<T>::blur_anisotropic()`

“Fleurs” (JPEG, 10% quality).

Application of `CImg<T>::blur_anisotropic()`

"Corail" (1 iter.)

Application : Image Inpainting with CImg

“Bird”, original color image.

Application : Image Inpainting with CImg

“Bird”, inpainting mask definition.

Application : Image Inpainting with CImg

“Bird”, inpainted with PDE-based diffusion.

Application : Image Inpainting with CImg

“Chloé au zoo”, original color image.

Application : Image Inpainting with CImg

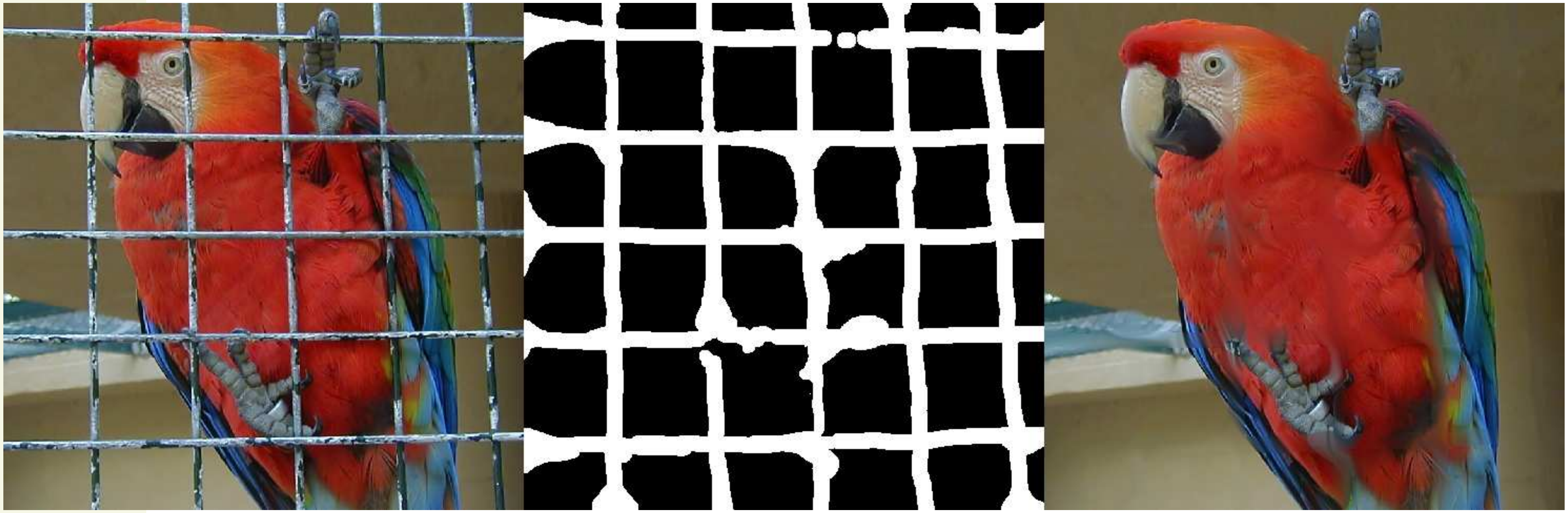
"Chloé au zoo", inpainting mask definition.

Application : Image Inpainting with CImg

“Chloé au zoo”, inpainted with PDE-based diffusion.

Application : Image Inpainting and Reconstruction with CImg

“Parrot”
500x500
(200 iter., 4m11s)



“Owl”
320x246
(10 iter., 1m01s)

Application : Image Resizing with CImg

"Nude" - (1 iter., 20s)

Application : Image Resizing with CImg

“Forest” - (1 iter., 5s)

Application : Image Resizing with CImg

(c) Details from the image resized by bicubic interpolation.

(d) Details from the image resized by a non-linear regularization PDE.

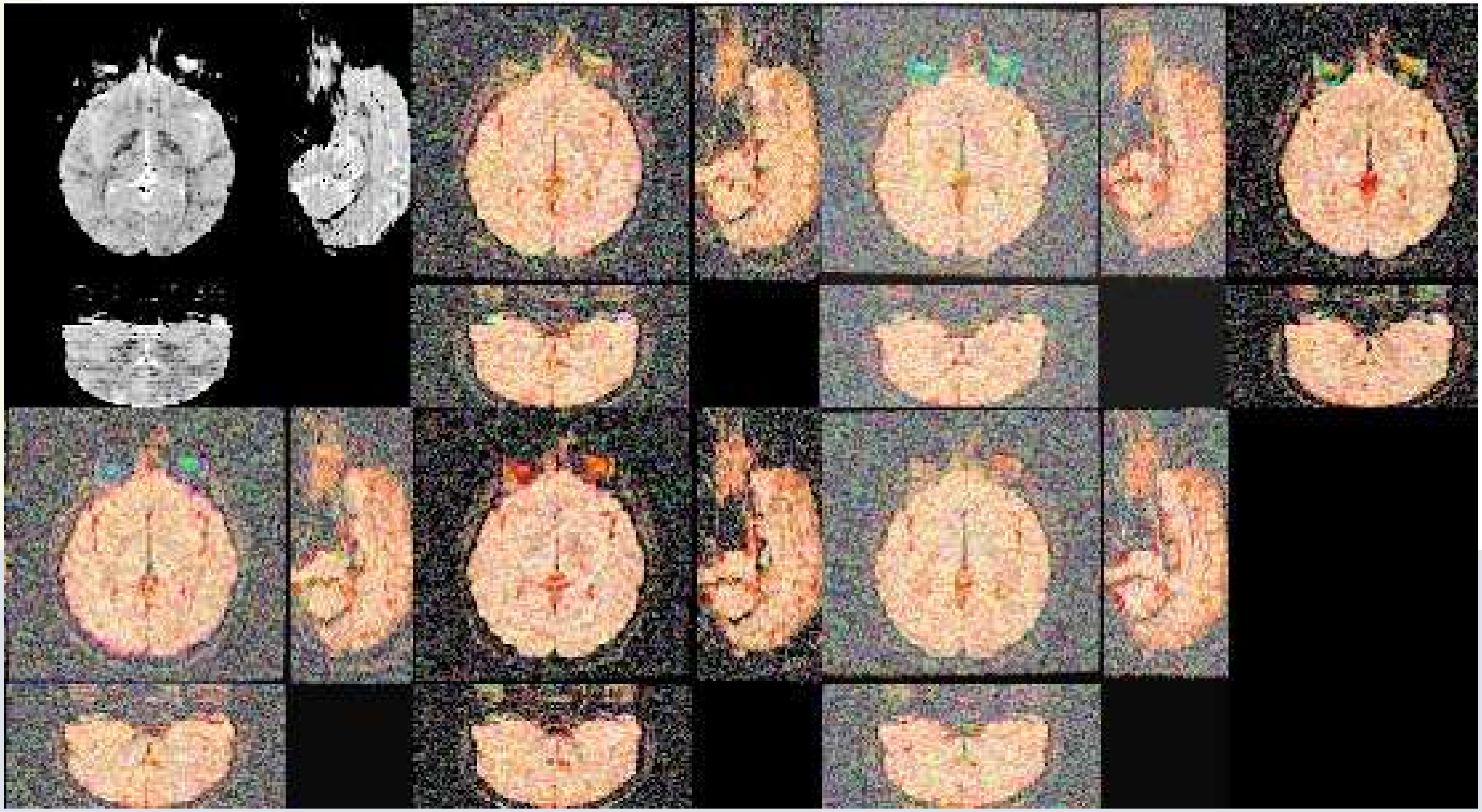
Application : Image Resizing with CImg

(a) Original color image

(b) Bloc Interpolation (c) Linear Interpolation (d) Bicubic Interpolation (e) PDE/LIC Interpolation

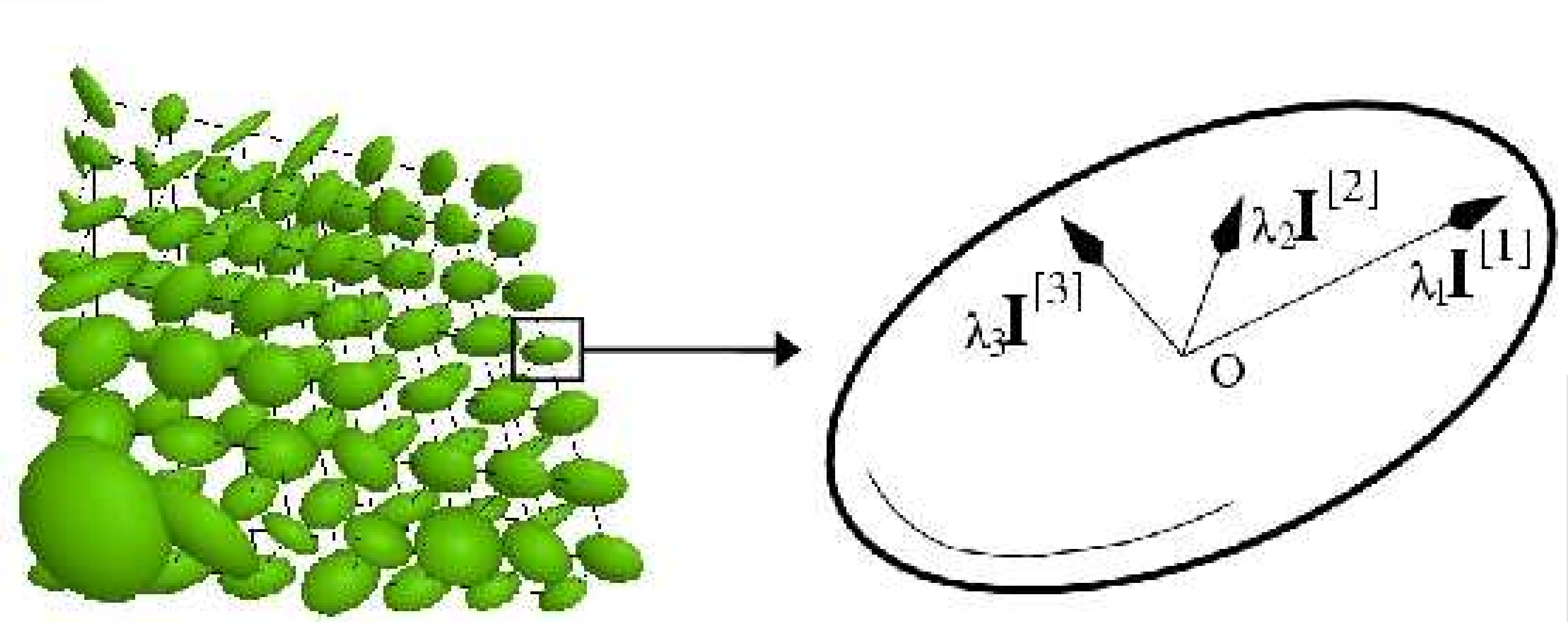
Application for DT-MRI Images processing : Principle

- MRI-based image modality measuring water diffusion within tissues.
- Acquisition of several raw images under different magnetic field magnitudes and orientations.



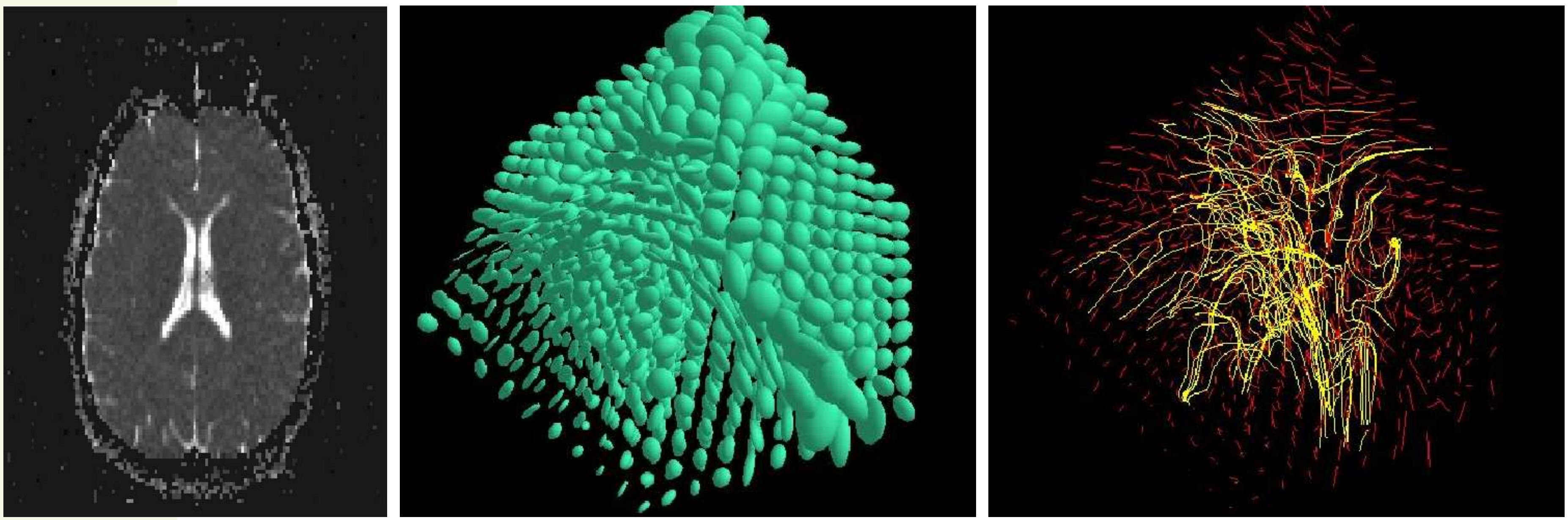
DT-MRI Images : Principle (2)

- A volume of *Diffusion Tensors* can be further estimated from these raw images.
- Diffusion tensors represent gaussian models of the water diffusion in the voxels, and are 3x3 symmetric and positive-definite matrices.
- Representation of a DT-MRI image with a volume of ellipsoids :



DT-MRI Images : Principle (3)

- DT-MRI images give structural informations about fiber networks within tissues.
- Fiber reconstruction can be performed by tracking the principal tensor directions.



- Used for tractography.

DT-MRI Estimation : Variational approach

- Robust tensor estimation by minimizing the following criterion :

$$\min_{\mathbf{D}\in \mathrm{P}(3)} \int_\Omega \sum_{k=1}^{n} \psi\left(\left|\ln\left(\frac{S_0}{S_k}\right) - g_k^T \mathbf{D} g_k\right|\right) + \alpha\, \phi(\|\nabla \mathbf{D}\|)\, d\Omega$$

- The corresponding gradient descent that respect the positive-definite property of the tensors is :

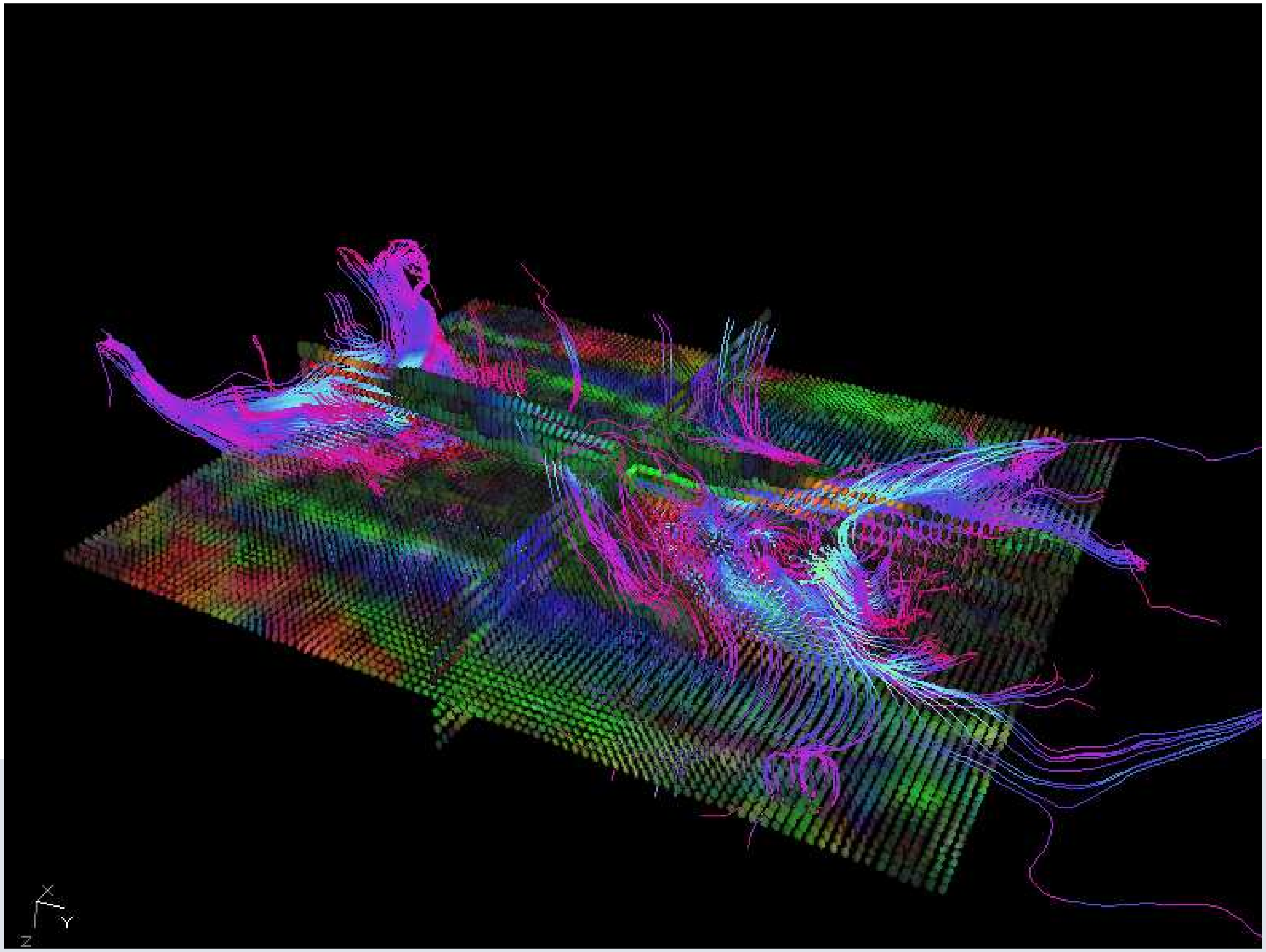
$$\begin{cases} \mathbf{T}_{(t=0)} = \mathbf{Id} \\ \frac{\partial \mathbf{T}}{\partial t} = (\mathbf{G} + \mathbf{G}^T)\mathbf{T}^2 + \mathbf{T}^2(\mathbf{G} + \mathbf{G}^T) \end{cases}$$

where $\mathbf{G}$ corresponds to the unconstrained velocity matrix defined as : $G_{i,j} = \sum_{k=1}^{n} \psi'(|v_k|)\mathrm{sign}(v_k)\left(g_k g_k^T\right)_{i,j} + \alpha \mathrm{div}\left(\frac{\phi'(\|\nabla \mathbf{T}\|)}{\|\nabla \mathbf{T}\|} \nabla T_{i,j}\right)$, with $v_k = \ln\left(\frac{S_0}{S_k}\right) - g_k^T \mathbf{T} g_k$.

⇒ Coded with CImg in less than 300 lines...

DT-MRI Visualization

- DTMRI dataset visualization and fibertracking code is distributed in the CImg package (File examples/dtmri_view.cpp, 823 lines).



Corpus Callosum Fiber Tracking

Conclusion and Links

- The CImg Library is a very small and pleasant C++ library that eases the coding of image processing algorithms.

  http://cimg.sourceforge.net/

- G'MIC is the script-based counterpart of CImg. It can be used for day-to-day image processing needs.

  http://gmic.sourceforge.net/

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

⇒ **Generic** (enough) libraries can 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 ..