ImLib3D : An Efficient, Open Source, Medical Image Processing Framework in C++

Marcel Bosc[1,2], Torbjorn Vik[1,2], Jean-Paul Armspach[2], and Fabrice Heitz[1]

[1] LSIIT UMR-7005 CNRS / Strasbourg I University, 67400 Illkirch, France
fabrice.heitz@ensps.u-strasbg.fr
[2] IPB UMR-7004 CNRS / Strasbourg I University, 67085 Strasbourg, France
{bosc,vik,armspach}@ipb.u-strasbg.fr

Abstract. ImLib3D is a C++ library for 3D medical image processing research. It provides a carefully designed, object-oriented, standards conforming C++ library, as well as a separate visualization system. Focus has been put on simplicity for the researcher who is considered to be the end-user. Source code is freely available and has been placed in an open collaborative development environment.

1 Introduction

The rapidly increasing complexity of medical image processing systems is a threat to the reproducibility of scientific work in this field. Reimplementing published work is often not feasible. Shared source code and shared development platforms are therefore becoming a necessity. Open source provides a well established framework for cooperative development, including high quality tools such as SourceForge.net. The ongoing success of systems such as Linux has proved that the Open Source model favors software quality. For the neuroscience community, Matlab® (*mathworks.com*) based Statistical Parametric Mapping (SPM: http://www.fil.ion.ucl.ac.uk/spm) is an important effort in this direction. However, ImLib3D, which is a stand-alone C++ library, has inherent performance and software engineering advantages over Matlab based code. ImLib3D is an alternative to The Insight Toolkit (ITK: *itk.org*), which also provides an interesting C++ framework designed for 3D medical image processing.

ImLib3D (*imlib3d.sourceforge.net*) provides an object oriented, C++, 3D image processing library with a separate visualization system. The objective of the library is to create a a framework that motivates collaborative work, is simple to use and provides a coherent basis for research and development in medical image processing. The library is currently at a mature stage of development and is actively constructing a user base.

2 Description of ImLib3D

An overview of ImLib3D is shown in Fig. 1. The fundamental elements are the image classes that have been implemented as STL-like templated containers with

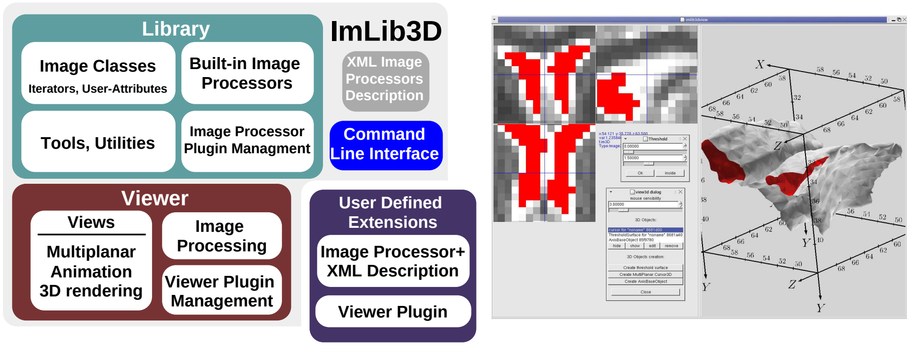


Fig. 1. (left) Overview of elements in ImLib3D. (right) Example of viewer rendering

STL-compliant iterators for traversing images or sub-images in various manners. It is therefore simple for the end-user to create images of arbitrary types, such as floating point images, complex-valued images, or deformation fields whose elements are 3D vectors. The use of templates also results in highly efficient code. Extensible attributes allow the user to dynamically add information to images, such as regions of interest, background values, and patient information.

ImLib3D defines a unified framework for describing image processing operators in XML format. Each operator has a full documentation as well as a precise description of argument types, allowing for automatic generation of fully documented viewer dialogs, command line interfaces, as well as dynamic extension with user defined image processors (image processor plugins). The command-line interface is particularly useful for high-level scripting, database interfacing and easy experimentation. Many low-level image processors are currently implemented: arithmetic, image-statistics (average, median, ...), morphological operations (erosion, distance transforms, connected component labeling, skeleton) with customizable structuring elements (neighborhoods), thresholding (simple, Otsu, ...), image transformations (affine or transformation field) with several methods of interpolation (including β-spline), linear filtering (arbitrary or separable convolutions, very fast Fourier domain filtering: *fftw.org*), image normalization and more.

The separate viewer features multi-planar (3 cross slices through a volume) views as well as surface rendering. Serial MRI may be visualized using animations. With the annotation system the user may interactively add and visualize textual or graphical information on images. The viewer is also dynamical extensible by user defined viewer plugins (dynamic libraries).

ImLib3D : An Efficient, Open Source, Medical Image Processing Framework in C++

Marcel Bosc,[1,2] Torbjørn Vik,[1,2] Jean-Paul Armspach,[2] Fabrice Heitz[1]

(1) LSIIT UMR-7005 CNRS / Strasbourg I University, 67400 Illkirch, France
(2) IPB UMR-7004 CNRS / Strasbourg I University, 67085 Strasbourg, France

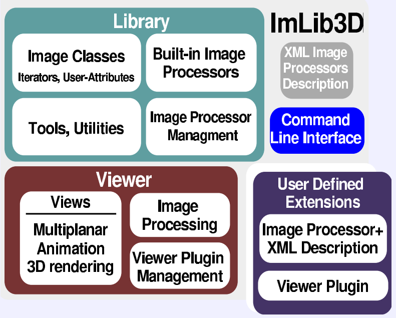
Design goals

Easy to use. ImLib3D is focused on researchers and programmers in 3D medical image processing. Intuitive operators and standards compliance make it easy to use even for unexperienced programmers.

Open Source. Software is a major aspect of image processing research. Distributing software is essential for scientific reproducibility. Open Source goes a step further, giving freedom to modify, adapt and improve software. ImLib3D is placed in a collaborative environment.

Careful design. Object-oriented and modular design, generic (templated) images, STL-compliant iterators and user extensibility help writing efficient and reusable code.

Overview



ImLib3D

Library

Image Classes
Iterators, User-Attributes

Built-in Image Processors

Tools, Utilities

Image Processor Managment

XML Image Processors Description

Command Line Interface

Viewer

Views
Multiplanar
Animation
3D rendering

Image Processing

Viewer Plugin Management

User Defined Extensions

Image Processor+ XML Description

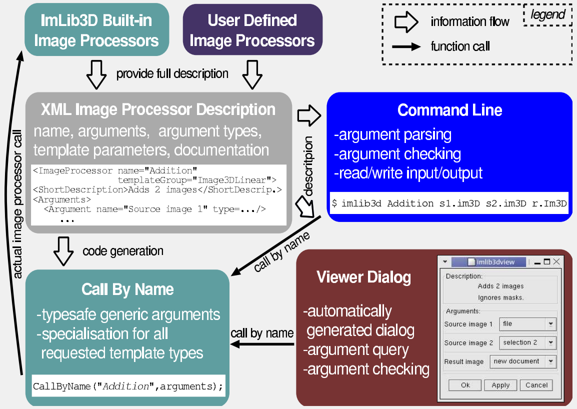
Viewer Plugin

ImLib3D is an open source C++ library and a separate image viewer. The library provides the essential infrastructure for volumetric image processing, including image classes, tools and built-in image processors.

http://imlib3d.sourceforge.net

Image processors

Image processing requires managing many image processing operators. In ImLib3D interfaces like viewer dialogs, and command line interfaces are automatically built from a detailed XML description of each image processor. Users may also **dynamically extend** ImLib3D with their own image processors. Generic processors are assigned template-groups so that they may be instantiated for all required types at compile-time.



XML Image Processor Description

name, arguments, argument types, template parameters, documentation

```
<ImageProcessor name="Addition"
                templateGroup="Image3DLinear">
<ShortDescription>Adds 2 images</ShortDescrip.>
<Arguments>
  <Argument name="Source image 1" type=.../>
    ...
```

Command Line

-argument parsing
-argument checking
-read/write input/output

```
$ imlib3d Addition s1.im3D s2.im3D r.Im3D
```

Call By Name

-typesafe generic arguments
-specialisation for all requested template types

```
CallByName("Addition",arguments);
```

Viewer Dialog

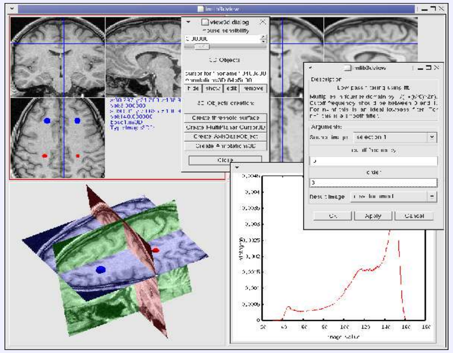
-automatically generated dialog
-argument query
-argument checking

imlib3dview
Description:
Adds 2 images
Ignores masks.
Arguments:
Source image 1: file
Source image 2: selection 2
Result image: new document

Many low-level image processors are currently implemented: arithmetic, image statistics, morphological operations (erosion, distance transforms, etc.) with customizable structuring elements, thresholding, image transformations (affine or transformation field) with several methods of interpolation, linear filtering (arbitrary or separable convolutions, very fast Fourier domain filtering: fftw.org), intensity normalization and more.

viewer

ImLib3D provides an optional viewer featuring multiplanar views, animations and 3D visualization. Images may be edited and annotated. All image processing operators (included user extensions) can be interactively called from the viewer. The viewer is may also be extended by user-supplied dynamically loaded modules.



ImLib3D Images

ImLib3D provides image classes that have been implemented as STL-like templated containers. They have been designed to be easy to use. Here is an example of basic image manipulation:

```
// creates a floating point image
Image3Df myImage(100,100,100);
// changes image value at (5,2,3)
myImage(5,2,3)=100;
myImage.WriteToFile("test.im3D");
```

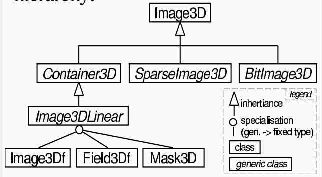
ImLib3D images have optional attributes such as: **masks** for defining regions of interest, extensible user defined **named attributes** of arbitrary types (ex: patient info.), **interpolators** (including B-spline), etc.

File format. Existing file formats cannot store generic image types. ImLib3D files are XML based. Tools are available for conversion from standard file formats.

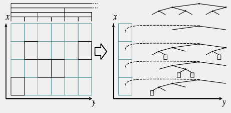
Genericity. Image processing requires manipulating images (3D containers) of many different types, such as as floating point images, complex-valued images, or deformation fields. Rewriting classes and image processors for each type is not feasible. Generic programming (templates) provides an elegant solution. In the following example a user creates an image of a custom type:

```
struct SatisticalModel {float average,variance;};
...
Container3D<StatisticalModel> atlas(50,50,50);
atlas(5,2,3).average=100;
atlas(5,2,3).variance=15;
```

Class Hierarchy. ImLib3D provides both standard (`Container3D`) and compact images, organized in the following, simplified hierarchy:

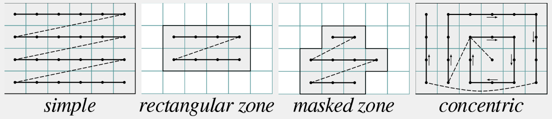


Compact images include bit-wise images and sparse images. For sparse images, non-zero elements are stored in binary trees (`std::map`).



Iterators

Iterators are tools for simple and efficient traversal of an image. ImLib3D uses STL-like iterators, offering a standard, well established syntax. Iterators may be used for walking through an image or a sub-image in many ways:



Iterators hide implementation details (like data storage format), improving performance and greatly simplifying code. The following example shows how to walk a masked zone using iterators:

```
Image3Df::iteratorMasked p;
for(p=image.begin(); p!=image.end(); p++)
{
    *p1=rand();
}
```

Without iterators this would be:

```
for(int z=0; z<image.Depth(); z++){
  for(int y=0; y<image.Height(); y++){
    for(int x=0; x<image.Width(); x++){
      if(image.Mask(x,y,z)){
        image(x,y,z)=rand();
}}}}
```

Code Example

Complex types, such as deformation fields (`Field3Df`), are easily created from generic images. Operators and generic image processors may be applied to any type of image, as long as they are compatible with the operations involved. This makes it easy to manipulate such complex types. Here, an average brain shape is determined by averaging and smoothing an inter-image deformation field, computed using deformable registration.

```
void TransformToAverageBrainShape(Image3Df source,
          const vector<Image3Df> &atlas, Image3Df &result)
{
  Field3Df average(atlas[0].Size());
  average.Fill(Vect3Df(0,0,0));
  // for all atlas images, add interimage deformation field
  for(size_t i=0; i<atlas.size(); i++)
  {
    Field3Df deformation;
    // registration actually computes inverse field
    IP3D::DeformableRegistration(source,atlas[i],
                               NULL,&deformation);
    average+=deformation;
  }
  average*=1.0/atlas.size();
  // smooth the resulting field
  IP3D::GaussianApproxFilter(average,3,3,average);
  // compute average brain by applying transformation
  IP3D::TransformWithInverseField(source,average,result);
}
```

bosc@ipb.u-strasbg.fr