Anatomist: a python framework for interactive 3D visualization of neuroimaging data

D Rivière[1,2], D Geffroy[1], I Denghien[1], N Souedet[1,3], Y Cointepas[1,2]

[1]IFR 49, Gif-sur-Yvette, France/[2]CEA, I2BM, Neurospin, Gif-sur-Yvette, France/[3]CEA, I2BM, MIRCEN, Fontenay-aux-Roses, France

Introduction

Anatomist is a powerful 3D visualization software dedicated to neuroimaging. It is cross-platform, open-source, distributed under the CeCill-B license [1] (similar to BSD), and available on BrainVISA [2] web site (http://brainvisa.info). Originally developed in C++ language for about 15 years, a set of python bindings and extensions have then been developed for Anatomist, including integration with popular python modules like Numpy [3], Matplotlib [4], or IPython [5].
Anatomist technologies rely on OpenGL [6], Qt [7] and PyQt [8] for the graphics and 3D parts, and on AIMS / PyAims libraries for neuroimaging data structures, IO and algorithms. It is able to display an arbitrary number of objects in an arbitrary number of views which may be embedded in dedicated graphical interfaces. The AIMS library (also part of the BrainVISA project) offers many plugin-based IO formats for several data structures: 3D/4D volumes (supporting DICOM, NIFTI, MINC, ECAT, and many 2D image formats), surfacic meshes (supporting GIFTI, PLY, MNI-Obj and a few other export formats like VRML-1 and POV), textures, voxels lists, more complex structured container objects (graphs, used for ROI drawing, sulci and fibers representation, or cortex parcellations), and a subset of VTK [9] objects. Objects may be combined to form new objects mixing properties of the combined ones, which offer richer display modes, such as overlaying volumes, texturing meshes according to 3D or 4D volumic data and/or to raw texture data, cutting meshes with volumic data along an arbitrary plane, volume rendering, clipping, and so on. Such functionalities may be extended by plugins.

Python control modes

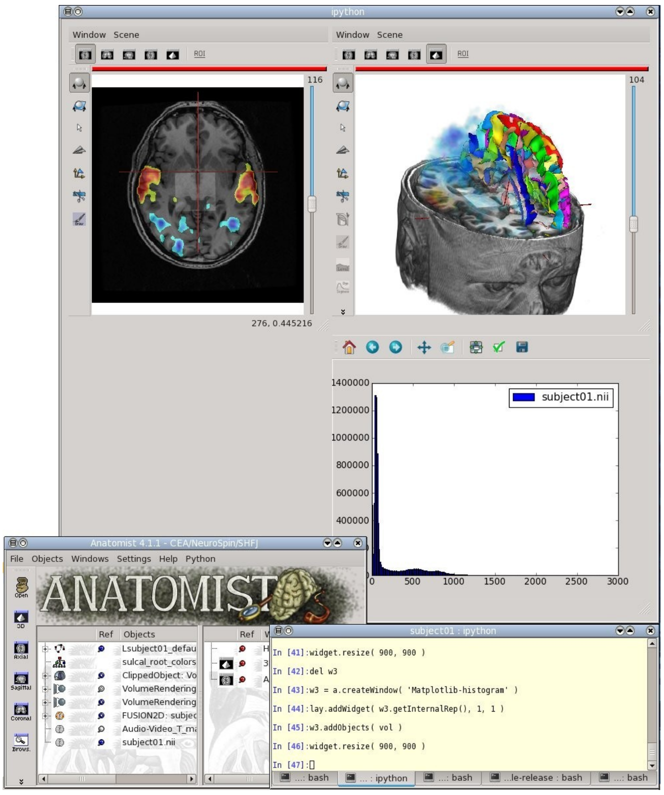
The python API of Anatomist allows two main modes: one is the “direct bindings” mode which offers an access to the underlying C++ data structures in memory, and the other one, “socket” allows to control a remote Anatomist as a separate application via a network connection. Both modes are using the same API, except that the “direct” mode is richer and allows more functionalities, such as GUI embedding and direct data modification in memory for visualization update on the fly.
The high-level python API is based on an “application” object which may address a unique singleton in direct mode application, and possibly several socket-based applications. Through this application object, manipulations use simple functions like `loadObject()`, `createWindow()` and so on (Fig. 1). Objects and windows are handled through dedicated classes (`AObject` and `AWindow`, namely). These objects grant access to various properties: colormaps, materials, referentials, camera properties, display modes, and other parameters.
Documentation can be found at http://brainvisa.info/doc/pyanatomist/sphinx/.

Lower-level API for full customization

In addition to this high-level API, intended to be quickly usable even by novice programmers, the “direct” mode offers an additional lower-level API which is actually the direct bindings of C++ classes. More complex due to its richness, this level of programming offers the possibility to extend many of Anatomist mechanisms: new object types and new window types, may be defined by subclassing in Python the C++ classes. 3D graphical objects may be defined at various levels, either reusing parts of existing graphical primitives of existing objects (geometric properties, vertices etc), or completely defining the 3D rendering at OpenGL level. Interaction controls (keyboard and mouse actions for users) and callbacks may be defined. Complete dedicated GUI applications may be designed. For this last usage, it is possible to use Qt Designer [7] to graphically design the application user interface. The anatomist main window can be suppressed to be replaced by a custom GUI, or by only an interactive python shell if needed.

Conclusion

Anatomist offers a wide range of possibilities for controlling, designing and extending 3D visualization for neuroimaging data in Python language. Combining the existing visualization and interaction features already provide numerous and very powerful capabilities at a low programming cost. Different complexities of customization are addressed at different levels of API to keep a high-level layer as simple as possible, thus still allowing to extend mechanisms at lower level and at lower scales, possibly down to the OpenGL layer.



```
In [41]:widget.resize( 900, 900 )
In [42]:del w3
In [43]:w3 = a.createWindow( 'Matplotlib-histogram' )
In [44]:lay.addWidget( w3.getInternalRep(), 1, 1 )
In [45]:w3.addObjects( vol )
In [46]:widget.resize( 900, 900 )
In [47]:
```

```python
import anatomist.direct.api as ana
import numpy
from PyQt4 import QtGui

# main application
a = ana.Anatomist()
# load an anatomical MRI volume
vol = a.loadObject( 'subject01.nii' )
# setup normalization transformation and referentials
vol.applyBuiltinReferential()
# load a functional T map
func = a.loadObject( 'Audio-Video_T_map.nii' )
func.applyBuiltinReferential()
# display both with appropriate colormaps
f = a.fusionObjects( (vol, func), method='Fusion2DMethod' )
func.setPalette( 'tvalues100-200-100-lfusion', minVal=-5, maxVal=5,
  absoluteMode=True )
widget = QtGui.QWidget() # we use a custom Qt widget to group several views
lay = QtGui.QGridLayout( widget )
w1 = a.createWindow( 'Axial' ) # anatomist window
lay.addWidget( w1.getInternalRep(), 0, 0 )
widget.show()
w1.addObjects( f ) # display fusionned object (anat + functional volumes)
# go to Talairach position (0,0,0)
a.execute( 'LinkedCursor', window=w1, position=(0, 0, 0) )
# use some volume rendering
vr = a.fusionObjects( vol, method='VolumeRenderingFusionMethod' )
vrf = a.fusionObjects( func, method='VolumeRenderingFusionMethod' )
# with clipping on the anatomical volume
cvr = a.fusionObjects( vr, method='FusionClipMethod' )
# display both
w2 = a.createWindow( '3D' )
lay.addWidget( w2.getInternalRep(), 0, 1 )
w2.addObjects( ( cvr, vrf ) )
# load a sulci nomenclature and graph
nom = a.loadObject( 'sulci/sulcal_root_colors.hie' )
sulci = a.loadObject( 'sulci/Lsubject01_default_session_auto.arg' )
sulci.graph()[ 'label_property' ] = 'label'
sulci.setReferential( vol.referential )
sulci.notifyObservers()
w2.addObjects( sulci, add_graph_nodes=True )
# rotate the view for better visibility
w2.camera( view_quaternion=[0.484051, -0.196338, -0.156624, 0.838222] )
# use matplotlib-based histogram module
w3 = a.createWindow( 'Matplotlib-histogram' )
lay.addWidget( w3.getInternalRep(), 1, 1 )
widget.resize( 900, 900 )
w3.addObjects( vol )
# now modify the anatomical volume data using numpy
numpy.array( a.toAimsObject( vol ).volume(), \
  copy=False )[100:150,100:150,40:80,0] += 300
vol.setChanged()
vol.notifyObservers()
```

Fig. 1: Example of Anatomist handling in python, showing a script (right) which may be used in an interactive IPython shell (left, lower), and the visual result (left). Views may be interactively modified, rotated, zoomed etc.

References:

[1] CeCILL licences, http://www.cecill.info/index.en.html
[2] BrainVISA, http://brainvisa.info
[3] Numpy, http://numpy.scipy.org
[4] Matplotlib, http://matplotlib.sourceforge.net
[5] IPython, http://ipython.scipy.org
[6] OpenGL, http://www.opengl.org
[7] Qt, http://qt.nokia.com
[8] PyQt, http://www.riverbankcomputing.co.uk
[9] VTK, http://www.vtk.org