3D Processing and Analysis with ImageJ

Philippe Andrey[a] and Thomas Boudier[b]

[a]AMIB, UR1197, Domaine de Vilvert, INRA, 78350 Jouy-en-Josas,France.

[b]Université Pierre et Marie Curie, IFR83, 7 quai St Bernard, 75252 Paris Cedex 05, France.

ABSTRACT

Modern microscopical techniques, such as LSCM, yields 3D and more generally multi-dimensional images and even 4D data when including time, and 5D when including numerous channels. Important developments have been done for visualisation of such images in ImageJ such as *VolumeViewer* or *ImageJ 3D Viewer*. However few plugins focused on the 3D information present in the images. Generally most processing are done in a slice by slice manner, and few plugins (including *ObjectCounter3D*) performs 3D measurements. This workshop will emphasise on tools for visualisation, processing and measurements of volumic data using ImageJ.

Keywords: Image Processing, Image Analysis, 3D, 4D, 5D, LSCM, Microscopy

1. INTRODUCTION

Last decade has seen the development of major microscopic techniques in biology such as LSCM in fluorescence microscopy and tomography for electron microscopy, not mentioning MRI or CT-Scan in medical imaging. These techniques enable a better resolution but also permits to obtain 3D volumetric data. The volumetric data can also be combined with time lapse and multi-channel acquisition leading to what is called 4D and 5D data.

In order to visualise, process and analyse these new type of data, new software had to be developed. Numerous commercial software exists such as Amira, Imaris or Volocity. They allow to perform 3D visualisation of the data, but they often provide only basic processing and analysis features. Furthermore 3D programming is not possible in an easy and portable way contrary to ImageJ. Altough ImageJ was developped with 2D processing and analysis in mind, thanks to plugins, it can however be used as a powerful software for 3D processing and analysis. FIJI (http://pacific.mpi-cbg.de/wiki/index.php/Fiji) offers simple and easy installation of ImageJ including numerous plugins for 3D manipulation and visualisation mentioned in this paper.

We present here basics of 3D imaging including visualisation, processing, analysis and programming within ImageJ.

2. VISUALISATION

As opposed to the 2D case, visualisation of 3D data can be a difficult task. At the same time, however, 3D visualisation now benefits from the development of powerful 3D graphics cards. In ImageJ volume are stored and treated as stack of 2D images. When opening a volume, the first slice will appear inside a window with the possibility to scroll trough the stack (see Fig. 1). For 3D visualisation powerful 3D manipulation and visualisation plugins are available such as *Volume Viewer* or *ImageJ 3D Viewer*.

contact author : email: tboudier@snv.jussieu.fr

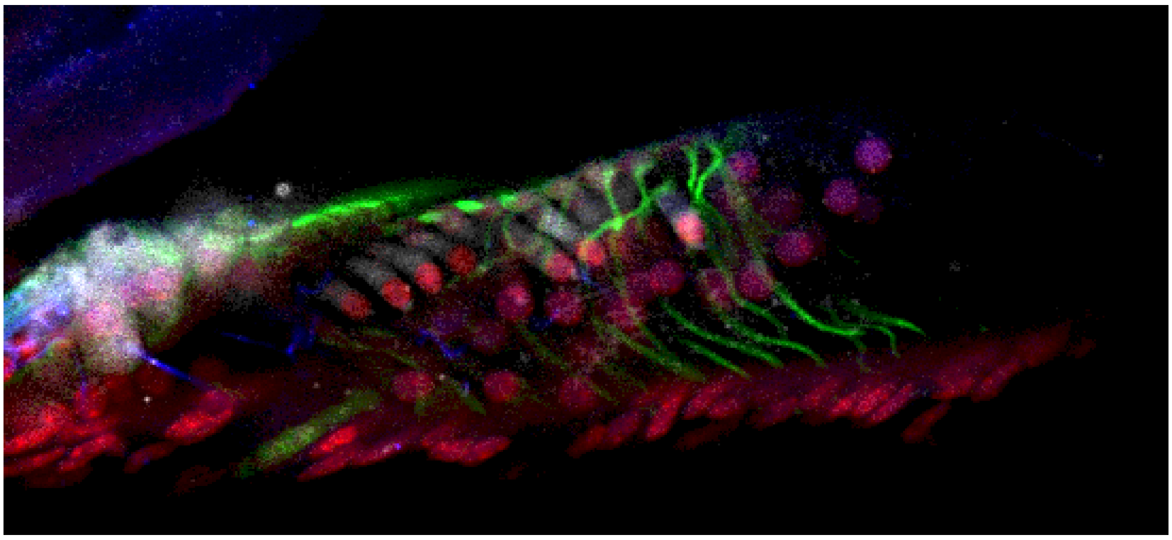


Figure 1. Slice 8 from the 3D stack Organ of Corti that can be found as a sample image in ImageJ.

Brightness and contrast

Brightness and contrast may be adjusted slice by slice to improve the visualisation of objects of interest. However contrast may be different from one slice to another due to the acquisition procedure, like decrease of fluorescence for LSCM or increase of thickness in tilted series for electron tomography. These changes in contrast can be modelled and the contrast can be corrected based on the model. The contrast can be also automatically adjusted from slice to slice thanks to the *Normalise All slice* command of the *Enhance Contrast* function. This procedure will process the data so that all the the slices will have roughly the same mean and standard deviation, correcting hence any changes in contrast trough the data.

3D manipulation and visualisation with ImageJ

ImageJ allows the manipulation of volume data via the *Stacks* submenu. It allows adding or removing slices, and animate the data. New tools have been added to reduce stack in Z or to concatenate two stacks. It allows also to perform projections of the data in the Z direction. Projections permit to visualise all the slices averaged as one slice and increase signal to noise ratio. One can also project the maximal intensity when bright signals are to be seen (or in the same way project the minimal intensity for dark signals). The first action to perform on a volume is the spatial calibration using the *Image Properties* command, this spatial calibration is often directly retrieved from the file corresponding to the volume. The volume can then be resliced with the *Reslice* command, for reslicing it is important to take into account the spacing, *i.e* the size in Z of the voxel compared to its size in $X-Y$. The command *Orthogonal Views* will display automatically the 3 views $X-Y$, $X-Z$ and $Y-Z$. Isotropic data are usually preferred for 3D visualisation and analysis. However generally the Z spacing is much larger than the $X-Y$ spacing. In order to reduce this ratio it is possible to reslice the data using the *Reslice* command twice : first reslice using the new desired spacing in Z, then reslice again to have the correct orientation. A volume with additional interpolated slices will be obtained. Of course these interpolated slices do not increase the qualitative resolution in the Z direction.

In order to have an idea of the 3D presence of objects inside the volume the simplest way is to make series of projections of the data at different orientations. This will create series of projections of the data as if one was

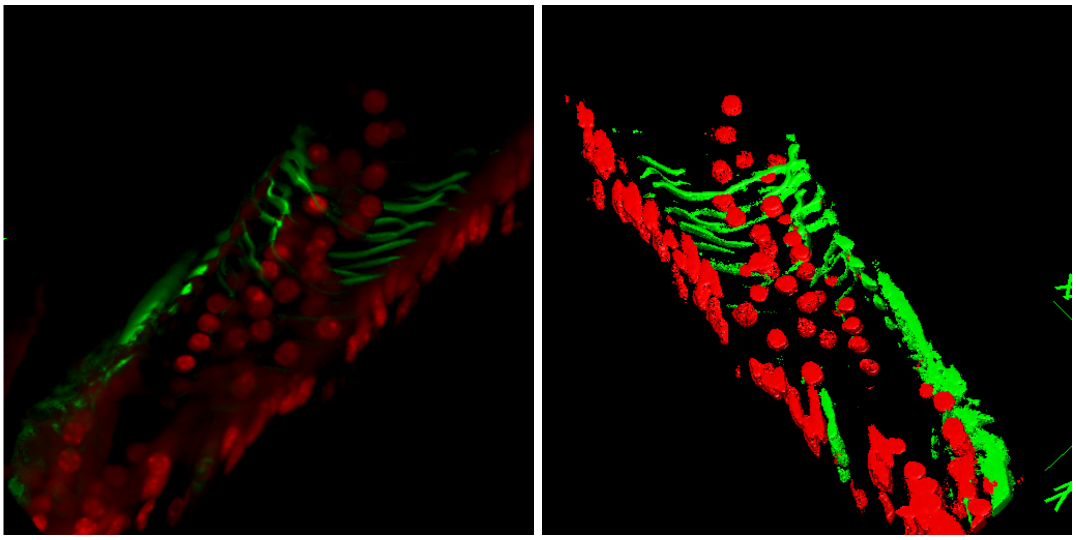


Figure 2. Two views with the *Image3DViewer* plugin from the red and green canal of 4D stack of Fig. 1. Left : volume rendering, all voxels are visualised. Right : surface rendering, only thresholded voxels are seen, note that no variation of intensities is available.

turning around the data and view it transparent at each of these angles, thanks to a animation our brain will reconstruct the whole 3D data. This can be done using the *3D Project* command with the *mean value* and *Y axis* options.

The Volume Viewer plugin

The *Volume Viewer* plugin allows more complex and interactive volume visualisation. Slicing in any direction can be performed inside the volume. The orientation is chosen with the mouse and then one can scroll trough the volume adjusting the *distance* view. For volume visualisation a *threshold* must be chosen, the voxels below this value will be set as transparent, for other voxels the higher values the more opaque. The voxels that are visualised are set with the *depth* slider. Animations can be recorded using the *volume slicer* macro.

The ImageJ 3D Viewer plugin

The *ImageJ 3D Viewer* plugin requires the installation of Java3D (https://java3d.dev.java.net/). It allows the visualisation at the same time of volume data, orthogonal slices and surfaces (see Fig. 2). In order to build surfaces (meshes) a threshold must be chosen. A binary volume is build and voxels at the border of the binarized objects are displayed. Transparencies between the various representations can be adjusted, thus allowing a complete interactive view of multiple objects coming from the same or different volumes. Finally an animation of the 3D visualisation can be recorded and saved as an AVI movie.

HyperStacks manipulation : 4D and 5D

Volume data incorporating time or multiple channels are called *HyperStacks* and can be manipulated within ImageJ. They correspond internally to standard stacks with additional information about their dimensionality as can be seen using the *HyperStack to Stack* and *Stack to HyperStack* commands. Hyperstacks can be projected in *Z* for all channels and times. However 4D or 5D visualisation is not possible with the *3D Project* command, the hyperstack needs to be converted into RGB stacks. For temporal 4D visualisation, that is to say animation of several 3D dataset, the *ImageJ 3D Viewer* plugin is one of the few free software, including Voxx2, available.

3. PROCESSING

Volumetric data may present a signal to noise ratio lower than 2D data since in fluorescence microscopy usually 2D data are projections of 3D data. The theory of processing is essentially the same for 2D and 3D data, it is based on the extraction of neighbouring pixel values, the computation of the processed value from these values, and the assignment of this value to the central pixel of the neighbourhood. Basically in 2D the neighbourhood may be any symmetrical shape, usually a square or a circle. For 3D images generally bricks or ellipsoids are used.

The number of values to process in 3D neighbourhood are quite larger than in 2D, for a neighbourhood of radius 1, 9 values are present in a 2D square but 27 in a 3D cube. The time to process an image is then considerably increased from 2D to 3D, thus algorithms implementation should be optimised. JNI programming can also be used, it consists in writing time-consuming procedures in other languages like C, which is quite faster than Java.

In this section we describe commonly used filters and morphological operators, with first a introduction to deconvolution.

Deconvolution

The optical system introduces deformation of the real signal, and deconvolution aims to restore the original signal from the images. Indeed a fluorescent point will appear as a "blob" in images. In order to perform the deconvolution a perfect model of the deformation induced by the system must be known. In fluorescence microscopy this can be done using fluorescent beads whose size is under the optical resolution of the system. In the image they appear as what is called the "Point Spread Function" (PSF). Some software may estimate this PSF from the microscope information but the result of the deconvolution will not be as good as with PSF estimated from real fluorescent beads.

Two plugins can be used in ImageJ to perform 3D deconvolution, *DeconvolutionLab* and *Parallel Iterative Deconvolution*. They both need a stack containing the PSF, they are based on iterative algorithms, that is to say they first compute the estimated original signal. Then this estimated original signal is convolved with the PSF, so it should reproduce the deformation induced by the system and give the actual image. Different algorithms are then used to minimise this difference. A commonly used algorithm in fluorescence microscopy is "Richardson-Lucy".

Noise reduction

Since volume are treated as stack of 2D images, filtering is performed in 2D in a slice by slice manner. This can produce fast satisfactory results but in some cases, where the signal to noise ratio is quite low, a 3D filtering should be favoured. In some cases the signal may be so weak that in may appear as noise in 2D slices, but when looking in 3D the signal is present. Only a 3D filtering can then successfully reduce noise and enhance this weak 3D signal. Classical filters used for reducing noise are mean, Gaussian and median filters (see Fig. 3). Median filter can be favoured since it is more robust to noise and preserves more the edges of the objects, it is specially powerful on noisy confocal images.

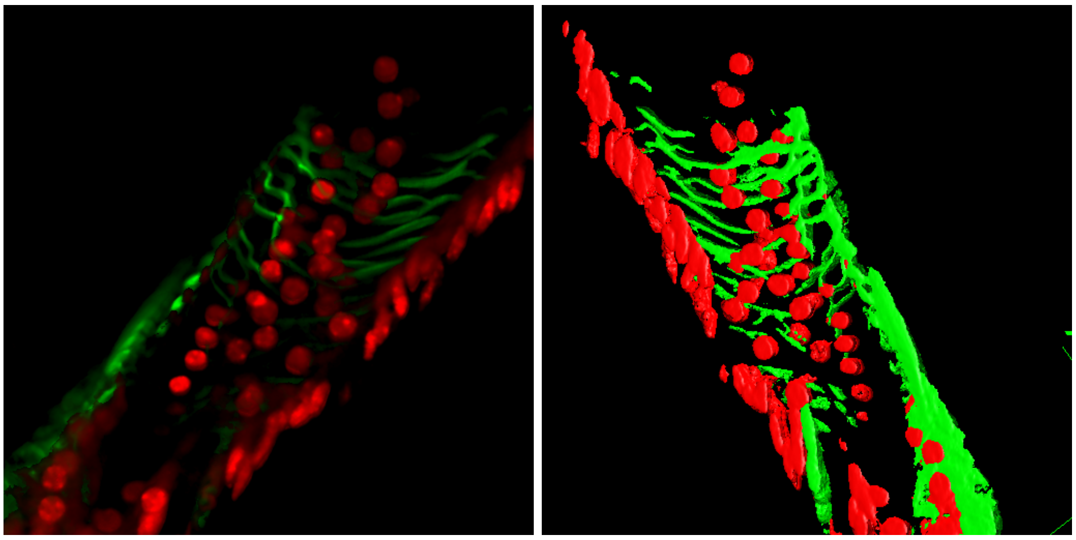


Figure 3. The stack of Fig. 1 was denoised by a 3D median filter of radius 1. Left : volume rendering. Right : surface rendering, note that surfaces are smoother.

As in 2D the radius of filtering should be chosen accordingly to the size of the objects of interest since filtering reduces noise but also may suppress small objects, the smaller the objects the lower the radius. However due to the acquisition systems, objects tend to be a bit elongated in Z compared to $X - Y$, furthermore the spacing is generally different in Z than in $X - Y$ so originally round objects maybe seen as elongated in the Z direction, the radius of filtering should hence be different in the $X - Y$ and Z direction. The kernel can also be optimally computed for each voxel in order to fit inside a probable object and avoiding averaging between signals and background. Such procedures are part of adaptive filtering. Some plugins are available like Kuwahara, anisotropic diffusion or Bilateral Filter. For 3D adaptive filtering, some works have been done for tomographic data, like *TOMOBFLOW*.

Mathematical Morphology

Mathematical morphology is originally based on complex mathematical operations between sets, however applied to images it can be seen as a particular type of filtering. Binary morphological operations are to be applied to binarized images and work on the objects segmented by the binarization. However these operations can be extended to grey-level images. The basic operation of mathematical morphology is called *dilatation* and consists in superimposing a kernel to every pixel of the image, and if any of the pixel in the kernel is part of an object then the centre of the kernel turns to the object value. If objects are considered with high intensity values compared to background (for instance 255 and 0), dilatation actually corresponds to a maximum filtering. The inverse operation of dilatation is called *erosion* and corresponds to a minimum filtering, the centre of the kernel turns to the background value if any of the pixel inside the kernel is part of the background. For 3D mathematical morphology the kernel is a 3D shape, usually an ellipsoid or a 3D brick.

Actually useful morphological operations are a combination of the two basics operation, for instance *closing* consists in a *dilatation* followed by an *erosion* and permits to make the objects more compact and fills small holes inside objects. The dual operation *opening*, a *erosion* followed by a *dilatation*, permits to remove small spots in the background and separate close touching objects. Other useful operations are *fill holes* to suppress any hole present within an object, *distance map* to compute for any pixel its distance to the closest edge or *tophat* to enhance spots. This operation consists in computing a local background image, and then to perform the difference between the original image and the computed background. In the case of bright spots the background is computed by performing a opening filter with a radius larger than the size of the spots. This procedure is quite powerful in 3D to detect fluorescence signals like F.I.S.H. signals.

Other processing

Any procedure that processes a 2D image via a kernel can be quite directly adapted to 3D data, however the main concern is time and memory requirements since a 3D image can be hundred times bigger than a 2D image.

Another purpose of filtering is edge detection, in 2D a classical way to detect edges is to compute edges in both X and Y directions and then combine them to find the norm of the gradient corresponding to the strength of the edges, the angle of the edges can also be computed from the X and Y differentials. For 3D edges the differential in Z is added to the X and Y ones.
$\delta x(i,j,k) \approx I(i+1,j,k) - I(i-1,j,k)$,
$\delta y(i,j,k) \approx I(i,j+1,k) - I(i,j-1,j)$,
$\delta z(i,j,k) \approx I(i,j,k+1) - I(i,j,k-1)$,
$edge = \sqrt{\delta x^2 + \delta y^2 + \delta z^2}$

Fourier processing

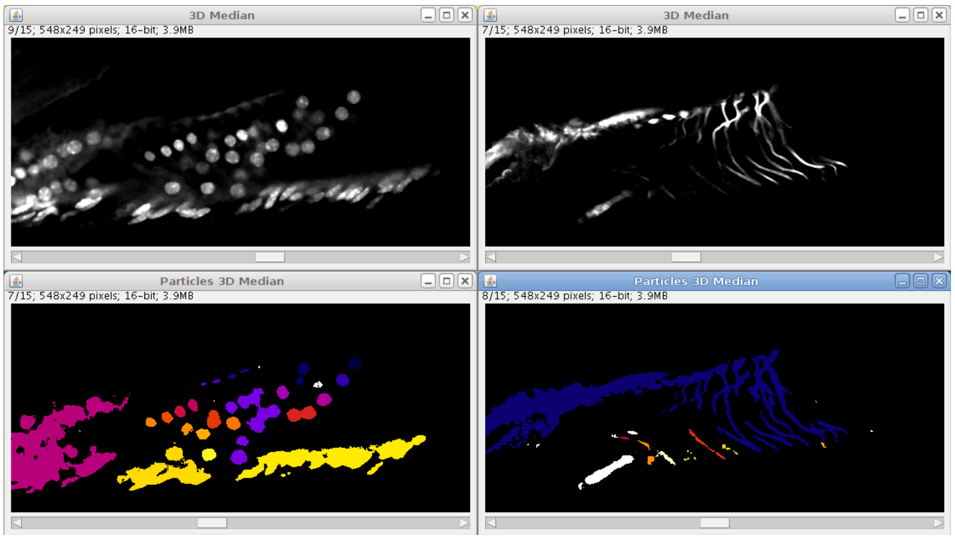
The above cited operations are performed on the input original images, other processing can be performed on the Fourier transform of the image. The Fourier transform of the image corresponds to the frequencies inside the input image, so, in another way, to the various sizes of objects. In order to reduce noise, *i. e.* objects with very small sizes hence high frequencies, a low pass filter can be used. To enhance objects with a particular size, band pass filtering can be performed, see the *Bandpass Filter* in the *FFT* menu. The Fourier Transform can also be computed in 3D, however the computation time can be quite long if image size is not a power of 2. Other applications of Fourier Transform are signal deconvolution and correlation computations, that permits to find the best displacement in X, Y and Z that match two volumes.

Registration

For stack alignment the computation of the 3D translation between two volumes can be easily computed in Fourier Space, however for computation of 3D rotation the problem is quite more complex than in 2D since 3 angles, around each axis, must be computed. Iterative procedures including translation and rotations estimations are generally used, see the *StackAlignment* plugin. *ImageJ 3D Viewer* can also perform 3D registration based on 3D points used as landmarks. For mosaic images, only the computation of the translation is needed and then can be computed in Fourier Space (see the *Stiching 3D* plugin).

4. SEGMENTATION

Segmentation is the central procedure in image analysis. It permits to give a meaning to an image by extracting the objects. The labelling of pixels to a numbered object (or to background) is one of the most difficult task to achieve automatically with a computer. Therefore, semi-automatic or manual approaches are sometimes more suited than completely automatic ones.


3D Median
9/15; 548x249 pixels; 16-bit; 3.9MB
3D Median
7/15; 548x249 pixels; 16-bit; 3.9MB
Particles 3D Median
7/15; 548x249 pixels; 16-bit; 3.9MB
Particles 3D Median
8/15; 548x249 pixels; 16-bit; 3.9MB
Figure 4. Output of the sample data from fig. 1 with the *ObjectCounter3D* plugin. Objects are segmented in 3D and labelled with an unique index, displayed here with a look-up table. Left : red channel, note that some objects are merged. Right : green channels, note all that fibres form only one object.

Thresholding

If the objects have homogeneous pixel values and if the values are significantly different from the background, a thresholding can be applied. In the case of an inhomogeneous illumination of the image the *Substract Background* procedure should be used beforehand. The *Adjust Threshold* tool helps in determining the threshold. An interval of values is manually selected and the thresholded pixels, corresponding to the objects, are displayed in red colour. An automatic threshold can be computed using various algorithms like *IsoData*, *Maximum of Entropy*, *Otsu*, etc.

Thresholding in 3D obeys the same principles as in 2D. However the object may present intensity variation in the Z direction. This variation can be corrected by a normalisation procedure (see section 3). An automatic calculation of the threshold can also be computed in each individual slices, but due to normal variation of objects intensities in Z this procedure is not well adapted for 3D segmentation.

After thresholding, objects should be labelled, that is to say numbered. The *Analyse Particles* procedure performs this task in 2D, it can be applied to 3D but it will label objects in each slices not taking into account the third dimension. Applied to an image stack, this procedure labels independently each slice so that sections at different levels within a given object will be represented with distinct labels. Furthermore objects that look separated in 2D may be part of the same object in 3D. Hence a true 3D labelling should be performed. The plugin *ObjectCounter3D* will both threshold and label the objects in 3D (see Fig. 4). It also computes geometrical measures like volume and surface.

Manual segmentation

In some cases the objects can not be thresholded in a satisfying way. Mathematical morphology operations (see section 3) may improve the binarized image, but for some complex objects, as ones observed by electron microscopy, a manual segmentation is required. The user have to draw in each slices the contours of the different objects. Some plugins can help the user in this task, such as *TrackEM2* or *SegmentationEditor*. Other non commercial tools exist, like *Free-D* or *IMOD*.

Semi-automatic segmentation : active contours

Active contours models, known as *snakes*, are semi automatic techniques that can help the user automate a manual segmentation. The user draws the initial model close to the object he wants to segment. The algorithm then iteratively deforms this initial shape so that it fits onto the nearest contour in the image. Many algorithms are available as plugins, such as *IVUSnake*, *Snakuscule*, or *ABSnake*.

Snake models, though developed for 2D data, can be used in 3D in a iterative manner. The user draws an initial shape on the first slice of the volume and the algorithm segments the object. This first segmentation can then be used as the initial shape for the next slice, and so on. However this method does not take into account the 3D information and the links that exist between adjacent slices. Furthermore, in the case of noisy data or other problem due to acquisition, an object can be missing or deformed in a particular slice. The slice by slice approach can not alleviate this problem and a real 3D procedure, taking into account constraints from adjacent slices, should be used. The plugin *ABSnake3D* uses such an approach, the model is a set of points lying in the different slices. At each iteration, each point moves towards the contours in the image while being constrained in $X - Y$ by its two neighbouring points and also in Z by the two closest points in the upper and lower slices.

5. ANALYSIS

Once the objects have been segmented the analysis can be performed to compute geometrical and intensity-based measures for each individual objects. These measures must be performed using the physical units of the image. Hence the image should be calibrated in the 3 dimensions thanks to the *Image/Properties* window.

Geometrical features

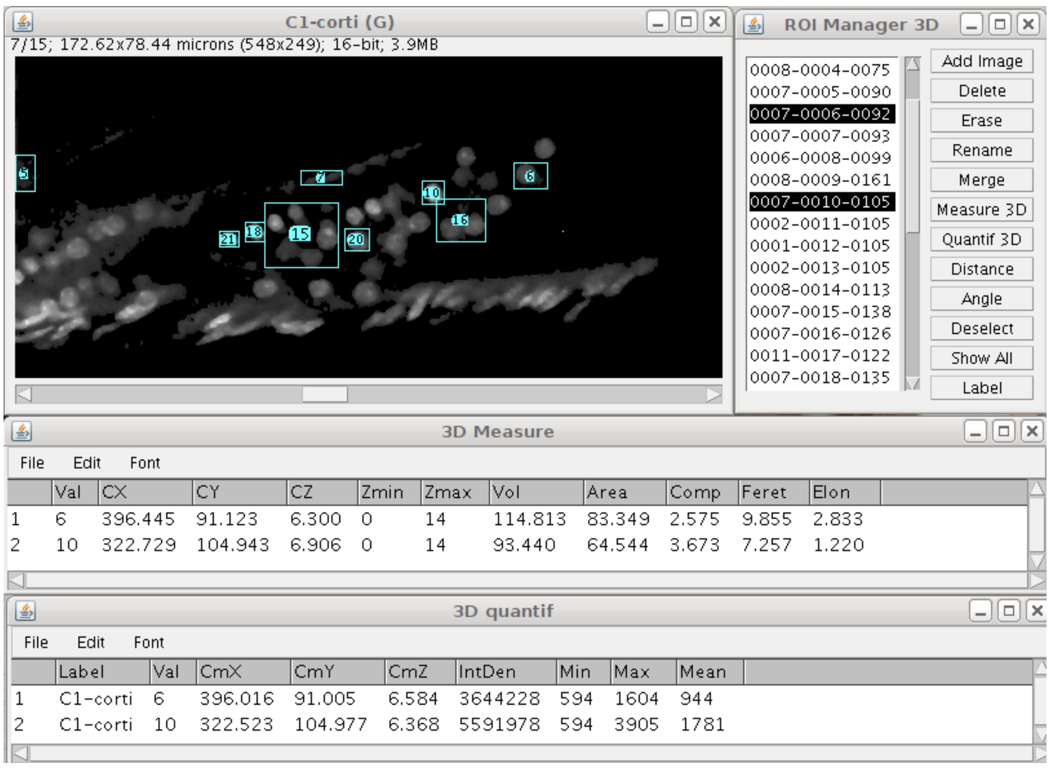
Geometrical features characterize the size and the shape of the objects without taking into account intensity values. Simple measures are volume and area. Volume is the number of voxels belonging to the object multiplied by the real volume of 1 voxel. This volume can also be computed form 2D slices as the sum of 2D areas multiplied by the Z spacing. The area surface of a 3D object may be estimated by the number of voxels at its border. However, as it is the case in 2D for perimeter, this estimation is higlhy sensitive to spatial resolution. Furthermore, object border can be defined in several ways, for instance allowing diagonal voxels or not. Hence, the computation of area may significantly vary from one program to another.

From these basic measures one can compute the 3D sphericity which is an extension of 2D circularity, and is computed from the ratio of volume and area. The sphericity, as well as the circularity, is maximal and equals 1 for a sphere : $Sph = \frac{36.\Pi.V^2}{A^3}$, with Sph being the sphericity, V the volume and A the area (see Fig. 5).

Other measures like Feret diameter or ellipsoid fitting can not be computed from 2D slices, contrary to volume. Feret diameter is the longest distance from two contour points of the 3D object. In order to have an idea of the global orientation of the object the fitting a 3D ellipsoid can be used. Actually the main axes of the 3D shape can be computed as the eigen vectors of the following moments matrix :

$$\begin{pmatrix} sxx & sxy & sxz \\ syx & syy & syz \\ szx & szy & szz \end{pmatrix}$$

with :
$sxx = \sum(x - cx).(x - cx)$,



7/15; 172.62x78.44 microns (548x249); 16-bit; 3.9MB

```
0008-0004-0075
0007-0005-0090
0007-0006-0092
0007-0007-0093
0006-0008-0099
0008-0009-0161
0007-0010-0105
0002-0011-0105
0001-0012-0105
0002-0013-0105
0008-0014-0113
0007-0015-0138
0007-0016-0126
0011-0017-0122
0007-0018-0135
```

	Val	CX	CY	CZ	Zmin	Zmax	Vol	Area	Comp	Feret	Elon
1	6	396.445	91.123	6.300	0	14	114.813	83.349	2.575	9.855	2.833
2	10	322.729	104.943	6.906	0	14	93.440	64.544	3.673	7.257	1.220

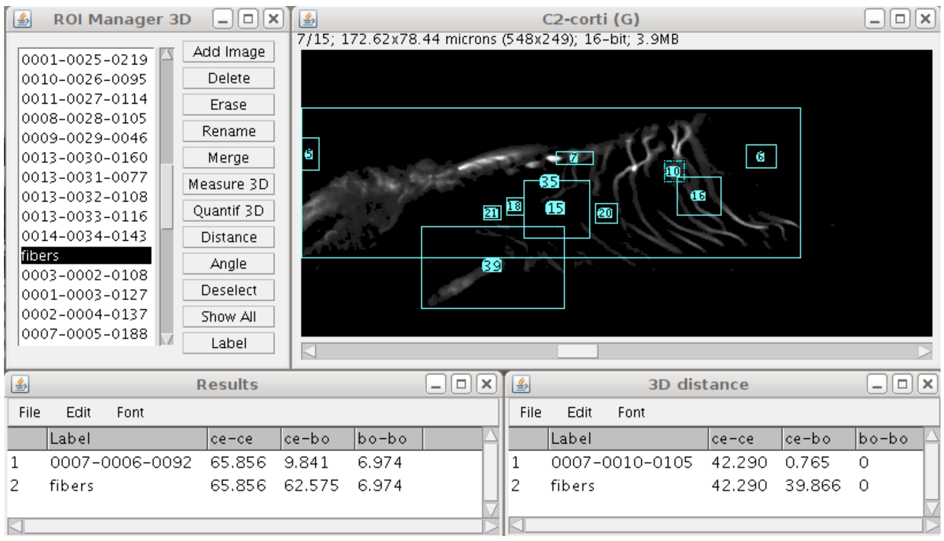
	Label	Val	CmX	CmY	CmZ	IntDen	Min	Max	Mean
1	C1-corti	6	396.016	91.005	6.584	3644228	594	1604	944
2	C1-corti	10	322.523	104.977	6.368	5591978	594	3905	1781

Figure 5. The *3DManager* plugin in action. Objects are added to the manager via the segmented image of fig. 4. The objects 6 and 10 were selected and measured. Geometrical measures appear in the *3D Measure* results window. The different measures are volume, area, 3D compacity (=sphericity), Feret Diameter and elongation of the best fitted ellipsoid. The canal 1 window was selected and intensity measures from this image appear in the *3D Quantif* results window.

$syy = \sum(y - cy).(y - cy)$,
$szz = \sum(z - cz).(z - cz)$,
$sxy = syx = \sum(x - cx).(y - cy)$,
$sxz = szx = \sum(x - cx).(z - cz)$,
$syz = szy = \sum(y - cy).(z - cz)$,
x, y and z being the coordinates of the points belonging to the object, cx, cy, cz being the coordinates of the centroid of the object. An elongation factor can be computed as the ratio between the two eigen values of the two first main axes. The *moments 3D* plugin automatically computes the main principal axes and align the object along the canonical axes.

Other geometrical measures can give valuable information like distances between objects. The distances can be computed from centre to centre, or from border to border giving the minimal distance between objects (see Fig. 6). One may also be interested by the distance from the centre of an object to the border of another, or the distance along a particular direction like the radial distance. One may also be interested in the percentage of inclusion of an object into another, *i.e.* colocalisation.

Finally the shape of the envelope of the object can be locally analysed. In 2D this envelope correspond to the perimeter of the object and can be mathematically analysed as a curve, hence information like curvature can be computed. In 3D such analysis is quite more complex since the border corresponds to a mathematical surface,



0001-0025-0219
0010-0026-0095
0011-0027-0114
0008-0028-0105
0009-0029-0046
0013-0030-0160
0013-0031-0077
0013-0032-0108
0013-0033-0116
0014-0034-0143
fibers
0003-0002-0108
0001-0003-0127
0002-0004-0137
0007-0005-0188

7/15; 172.62x78.44 microns (548x249); 16-bit; 3.9MB

	Label	ce-ce	ce-bo	bo-bo
1	0007-0006-0092	65.856	9.841	6.974
2	fibers	65.856	62.575	6.974

	Label	ce-ce	ce-bo	bo-bo
1	0007-0010-0105	42.290	0.765	0
2	fibers	42.290	39.866	0

Figure 6. Distance computation with the *3DManager* plugin. The objects from red and green channels were added to the manager. The distances between objects 6 and 10 from red canal and object 1 (representing the fibres) from green canal were computed. From border to border distances one can see that object 10 touches the fibres.

and for instance for each point of a surface there are two curvatures instead of one, see the *principal curvature plugin* for details.

Intensity features

While the above geometrical measures quantify size and shape independently of the voxels values, intensity features characterise signal intensity and spatial distribution within objects. For instance *Integrated Density*, defined as the volume multiplied by the mean intensity in the object, corresponds to the sum of intensities inside the object. Mass centre is the barycentre of the voxels weighted by the voxel intensities : $MCx = \dfrac{\sum I_p.x}{\sum I_p}$, where I_p is the intensity at point p belonging to the object and x its x coordinate (similarly for y and z). This mass centre may reflect more accurately the centre of fluorescence spatial distribution.

Main axes can also be computed by weighting all coordinates by voxel values. Note that, in cases where the objects present homogeneous and symmetrical intensities, intensity-based barycentre and axes does not differ much from those computed in the above paragraph, where actually all voxels are weighted by a factor of 1.

Morphological analysis

Mathematical morphology can be used to fill holes inside objects or make them more compact. It can also be used to have a estimate of the sizes of the objects without segmenting them. The technique, called *granulometry*, is based on the idea that bright objects processed by an erosion greater than the radius of the object will disappear. A sequence of filtered images is generated by applying openings (erosion followed by dilatation) of increasing radii. The image that differs the most from the original one corresponds to the disappearing of many objects. The corresponding radius gives an estimate of the radius of objects present in the volume. If objects of different radii are present, as many major changes will be observed in the image sequence. Similarly, series of closing (dilatation followed by erosion) can be applied to analyse the background size distribution, that is to say distances between objects. This procedure is only valid for spherical objects.

Other analyses

The size of an object can be locally analysed by computing for each voxel the sphere that locally fits into the object. This can be done using the *3D Local Thickness* plugin. This measure can be performed on tube-like objects like axons for instance. In order to analyse the branching of neurons, one can first segment them, compute and analyse their skeletons using the *Skeletonize3D* and *AnalyzeSkeleton* plugins.

Spatial analyses

The repartition of objects within the 3D space can be analysed, for example to test whether or not interactions, either attractive or repulsive, exist between different objects. Such analyses largely rely on spatial statistical tools based on distances between objects or between arbitrary space positions and objects.

6. PROGRAMMING

Many, if not all, operations implemented in ImageJ can be performed on 3D data on a slice by slice manner. Hence macro recording should be sufficient to record series of processing performed on a stack by either ImageJ or plugins. For 3D analysis, access to all the slices is often required. This can be done in macros using the slices commands, and the `ImageStack` class in plugins.

Macros

In order to manipulate a stack, the different slices corresponding to different z should be accessed. This can be done using the macro command `setSlice(n)` where n is the slice number between 1 and the total number of slices. The current displayed slice can be retrieved using the command `getSliceNumber()`, `nSlices` is a built-in variable corresponding to the total number of slices. In order to scroll trough the slices a loop is required, a loop is constructed by the `for` command : `for(index=start;index<=end;index=index+step)` where `start` and `end` are the starting and ending value for the variable `index`, `step` is the increment of `index` between each iteration, usually set to 1. Here are an example of a macro that finds the slice where the 3D object has the largest area :

```
// start can be 1 or the actual slice
start=getSliceNumber();
end=nSlices;
areamax=0;
slicemax=start;
run("Set Measurements...", "area redirect=None decimal=3");
for(slice=start;slice<=end;slice=slice+1){
        setSlice(slice);
        run("Analyze Particles...", "size=0-Infinity circularity=0.00-1.00
             show=Nothing display exclude record slice");
        area=getResult("Area");
        if(area>areamax){
                areamax=area;
                slicemax=slice;
        }
}
```

```
write("Max area at slice "+slicemax);
getVoxelSize(sizex,sizey,sizez,unit);
write("Max area at z "+slicemax*sizez+" "+unit);
```

For 4D and 5D data, the macro functions `Stack.` should be used. The various dimensions, that is to say the width, height, number of channels, number of slices and number of times, can be retrieved using the function `Stack.getDimensions(width,height,channels,slices,frames)`. The equivalent of `setSlice(n)` is simply `Stack.setSlice(n)` for HyperStacks, similarly the functions `Stack.setChannel(n)` and `Stack.setFrame(n)` will change the channel and timepoint. The different information can be set using one function `Stack.setPosition(channel,slice,frame)` and retrieved by the function `Stack.getPosition(channel,slice,frame)`. The same macro is rewritten supposing a 4D data :

```
Stack.getPosition(channel,slice,frame);
startTime=frame;
startZ=slice;
Stack.getDimensions(w,h,channels,slices,frames);
endTime=frames;
endZ=slices;
run("Set Measurements...", "area redirect=None decimal=3");
getVoxelSize(sizex,sizey,sizez,unit);
for(time=startTime;time<=endTime;time=time+1){
        Stack.setFrame(time);
        areamax=0;
        slicemax=startZ;
        for(slice=startZ;slice<=endZ;slice=slice+1){
                Stack.setSlice(slice);
                run("Analyze Particles...", "size=0-Infinity circularity=0.00-1.00
                     show=Nothing display exclude record slice");
                area=getResult("Area");
                if(area>areamax){
                        areamax=area;
                        slicemax=slice;
                }
        }
        write("For time "+time+" Max area at slice "+slicemax);
        write("For time "+time+" Max area at z "+slicemax*sizez+" "+unit);
}
```

Plugins

Plugins are more adapted than macros to implement new processing procedures since the pixels access is quite faster. The Java class that stores stacks is called `ImageStack`, the stack can be retrieved from the current `ImagePlus` window :

```
// get current image and get its stack
 ImagePlus plus=IJ.getImage();
 ImageStack stack=plus.getStack();
```

The dimensions of the stack can be obtained using the methods `getWidth()`, `getHeight()` and `getSize()`:

```
 int w=stack.getWidth();
 int h=stack.getHeight();
 int d=stack.getSize();
 IJ.write("dimensions : "+w+" "+h+" "+d);
```

In order to access the pixels values one has to access first the slice of interest and then the pixel inside this slice :

```
 int x,y,z;
 x=w/2; y=h/2; z=d/2;
 ImageProcessor slice=stack.getProcessor(z+1);
 int val=slice.getPixel(x,y);
 IJ.write("The central pixel has value : "+val);
```

Note that the slice number in the `getProcessor` method starts at 1 and not 0. Following is an example of a 3D mean filtering :

```
 int p0,p1,p2,p3,p4,p5,p6,mean;
 ImageStack filtered=new ImageStack(w,h);
 ImageProcessor filteredSlice;
 ImageProcessor slice, sliceUp, sliceDown;
 for(int k=2;k<=d-1;k++){
        slice=stack.getProcessor(k);
        sliceUp=stack.getProcessor(k+1);
        sliceDown=stack.getProcessor(k-1);
        filteredSlice=new ByteProcessor(w,h);
        for(int i=1;i<w-1;i++){
        for(int j=1;j<h-1;j++){
                p0=slice.getPixel(i,j);
                p1=slice.getPixel(i+1,j);
                p2=slice.getPixel(i-1,j);
                p3=slice.getPixel(i,j+1);
                p4=slice.getPixel(i,j-1);
                p5=sliceDown.getPixel(i,j);
                p6=sliceUp.getPixel(i,j);
                mean=(p0+p1+p2+p3+p4+p5+p6)/7;
                filteredSlice.putPixel(i,j,mean);
        } }
        filtered.addSlice("",filteredSlice);
 }
 ImagePlus filteredPlus = new ImagePlus("Filtered", filtered);
 filteredPlus.show();
```

Note that the result slices `filteredSlice` are built one by one and added to the result stack `filtered`.

Using libraries

In order to manipulate volume data more easily, specifically designed classes for 3D data can be used such as *ij3d* for processing and analysis, or classes coming from *ImageJ 3D Viewer* or *VolumeViewer* for visualisation. With the *ij3d* package an 3D image is created from a stack, and pixels can be accessed directly :

```
// build a Image3D for 8-bits or 16-bits data
IntImage3D image3d = new IntImage3D(imp.getStack());
// get the sizes
int w = image3d.getSizex();
int h = image3d.getSizey();
int d = image3d.getSizez();
// get the value of the central pixel
int pix = image3d.getPixel(w/2,h/2,d/2);
```

Classical filters are implemented like median, mean, edges, morphological filters, and so on. The result can be visualised retrieving the underlying `ImageStack` :

```
// define the radii for filtering
IntImage3D filtered = image3d.meanFilter(2, 2, 1);
// dislay the filtered stack
new ImagePlus("3D mean", filtered.getStack()).show();
```

Objects can also be segmented in 3D and analysed using the class `Object3D`, basic 3D calculation can be performed using the class `Vector3D`, basic shapes can be draw in 3D using the class `ObjectCreator3D`, ...

```
// stack is a stack containing segmented objects, each one having a particular grey-level
// for instance coming from ObjectCounter3D
IntImage3D seg = new IntImage3D(stack);
// create the object 1, which corresponds to segmented voxels having values 1
obj = new Object3D(seg, 1);
// compute barycenter
obj.computeCenter();
// compute borders
obj.computeContours();
// get barycenter
float bx = obj.getCenterX();
```

```
 float by = obj.getCenterY();
 float bz = obj.getCenterZ();
 // get the feret's diameter
 double feret = obj.getFeret();
 // construct an sphere with pixel value 255 centered on barycenter and with radius=feret/2
 float rad=feret/2.0;
 ObjectCreator3D sphere = new ObjectCreator3D(w, h, d);
 obj.createEllipsoide(bx, by, bz, rad, rad, rad, 255);
 new ImagePlus("Sphere", sphere.getStack()).show();
```

For visualisation the ImageJ 3D Viewer library can be used, this example is extracted from the ImageJ 3D viewer site http://3dviewer.neurofly.de/ .

```
// Open an image
String path = "/home/bene/PhD/brains/template.tif";
ImagePlus imp = IJ.openImage(path);
new StackConverter(imp).convertToGray8();

// Create a universe and show it
Image3DUniverse univ = new Image3DUniverse();
univ.show();

// Add the image as a volume rendering
Content c = univ.addVoltex(imp);

// Display the image as orthslices
c.displayAs(Content.ORTHO);

// Add an isosurface
c = univ.addMesh(imp);

// Display the Content in purple
c.setColor(new Color3f(0.5f, 0, 0.5f));

// Make it transparent
c.setTransparency(0.5f);

// Change the isovalue (= threshold) of the surface
c.setThreshold(50);

// remove all contents and close the universe
univ.removeAllContents();
univ.close();
```

APPENDIX A. LIST OF PLUGINS

- VolumeViewer : http://rsbweb.nih.gov/ij/plugins/volume-viewer.html
- Volume Slicer : http://doube.org/macros
- Moments 3D : http://doube.org/plugins
- ImageJ3DViewer and 3D Segmentation Editor : http://www.neurofly.de/
- Stitching 3D : http://fly.mpi-cbg.de/˜preibisch/software.html#Stitching
- Image5D : http://rsbweb.nih.gov/ij/plugins/image5d.html
- IJ3D processing and analysis : http://imagejdocu.tudor.lu/ plugin:morphology:3d_binary_morphological_filters:start
- StackAlignment : http://www.med.harvard.edu/JPNM/ij/plugins/Align3TP.html
- 3D_Object_Counter : http://imagejdocu.tudor.lu/plugin:analysis:3d_object_counter:start
- ABSnake : http://imagejdocu.tudor.lu/plugin:segmentation::active_contour:start
- LiveWire and IVUSnake : http://ivussnakes.sourceforge.net/
- Snakuscule : http://bigwww.epfl.ch/thevenaz/snakuscule/
- TrackEM2 : http://www.ini.uzh.ch/ acardona/trakem2.html
- DeconvolutionLab : http://bigwww.epfl.ch/algorithms/deconvolutionlab/
- Parallel Iterative Deconvolution : http://sites.google.com/site/piotrwendykier/ software/deconvolution/paralleliterativedeconvolution
- Principal Curvature Plugin : http://fly.mpi-cbg.de/p̃reibisch/software.html#Curvatures
- 3D Local Thickness : http://www.optinav.com/Local_Thickness.htm
- Skeletonize3D : http://imagejdocu.tudor.lu/plugin:morphology:skeletonize3d:start
- AnalyzeSkeleton : http://imagejdocu.tudor.lu/plugin:analysis:analyzeskeleton:start

APPENDIX B. OTHER PROGRAMS

- IMOD 3D segmentation and visualisation : http://bio3d.colorado.edu/imod/
- FreeD 3D modeling : http://amib.jouy.inra.fr/html/software_en.html
- Voxx2 3D and 4D visualisation : http://www.nephrology.iupui.edu/imaging/voxx/
- TOMOBFLOW 3D adaptive filtering : http://sites.google.com/site/3demimageprocessing/tomobflow