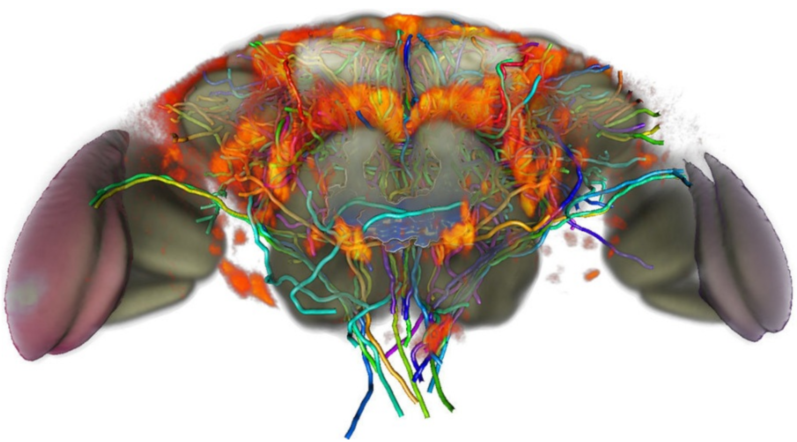
Vaa3D: an extendible and versatile open-source tool for 3D visualization-assisted analysis of large-scale bioimages

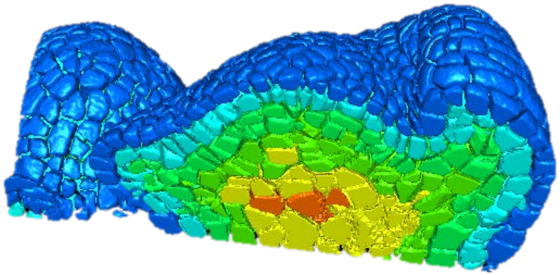
8th Advanced School on Scientific Visualization @CINECA
14 – 18 October 2013 CINECA - Bologna

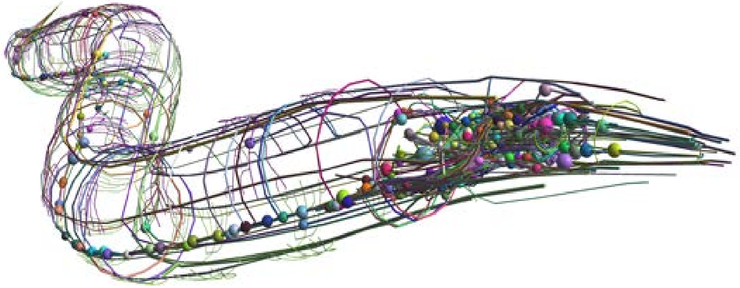
speaker
Ing. Alessandro Bria
PhD student at University of Cassino and L.M., Cassino, Italy
Collaborator at University Campus Bio-Medico of Rome, Rome, Italy
Collaborator at Allen Institute for Brain Science, Seattle, WA, USA
Collaborator at Radboud University Nijmegen, The Netherlands
Member of the Projectome project at ICON foundation, Florence, Italy

The context: bioimage informatics **(1/2)**

- using **computational techniques** to **analyze** (= extract useful information from) multi-dimensional **bioimages** at molecular, cellular or systemic scale, e.g. :
  - high-content screening (or *visual screening*) for drug discovery
  - cells segmentation
  - **mapping brain circuits**





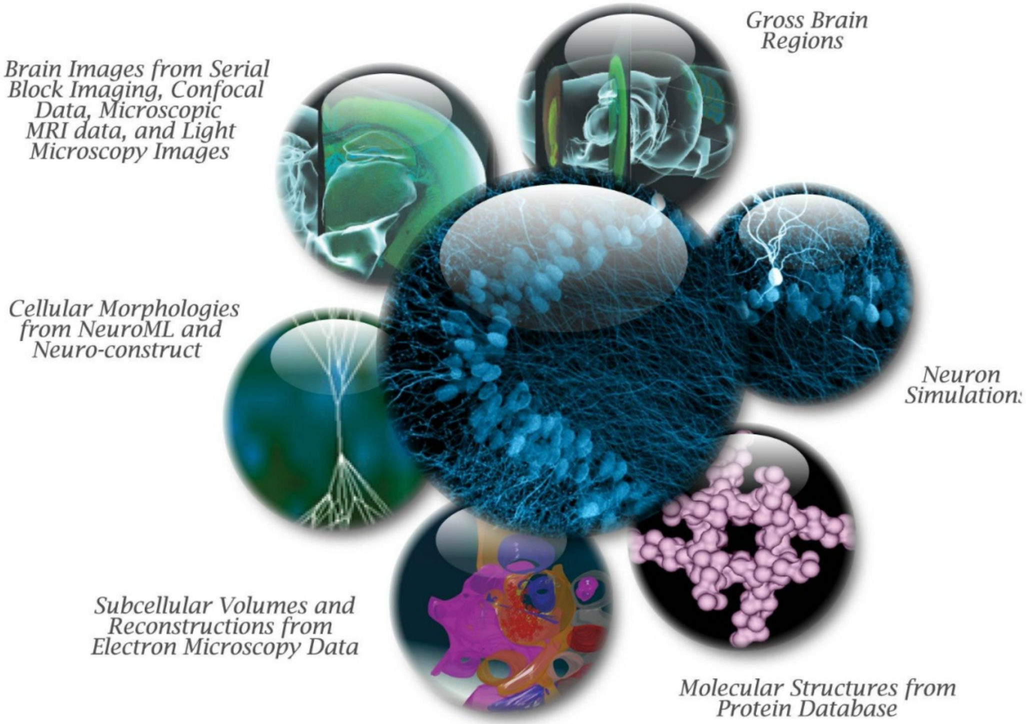


The context: bioimage informatics (2/2)

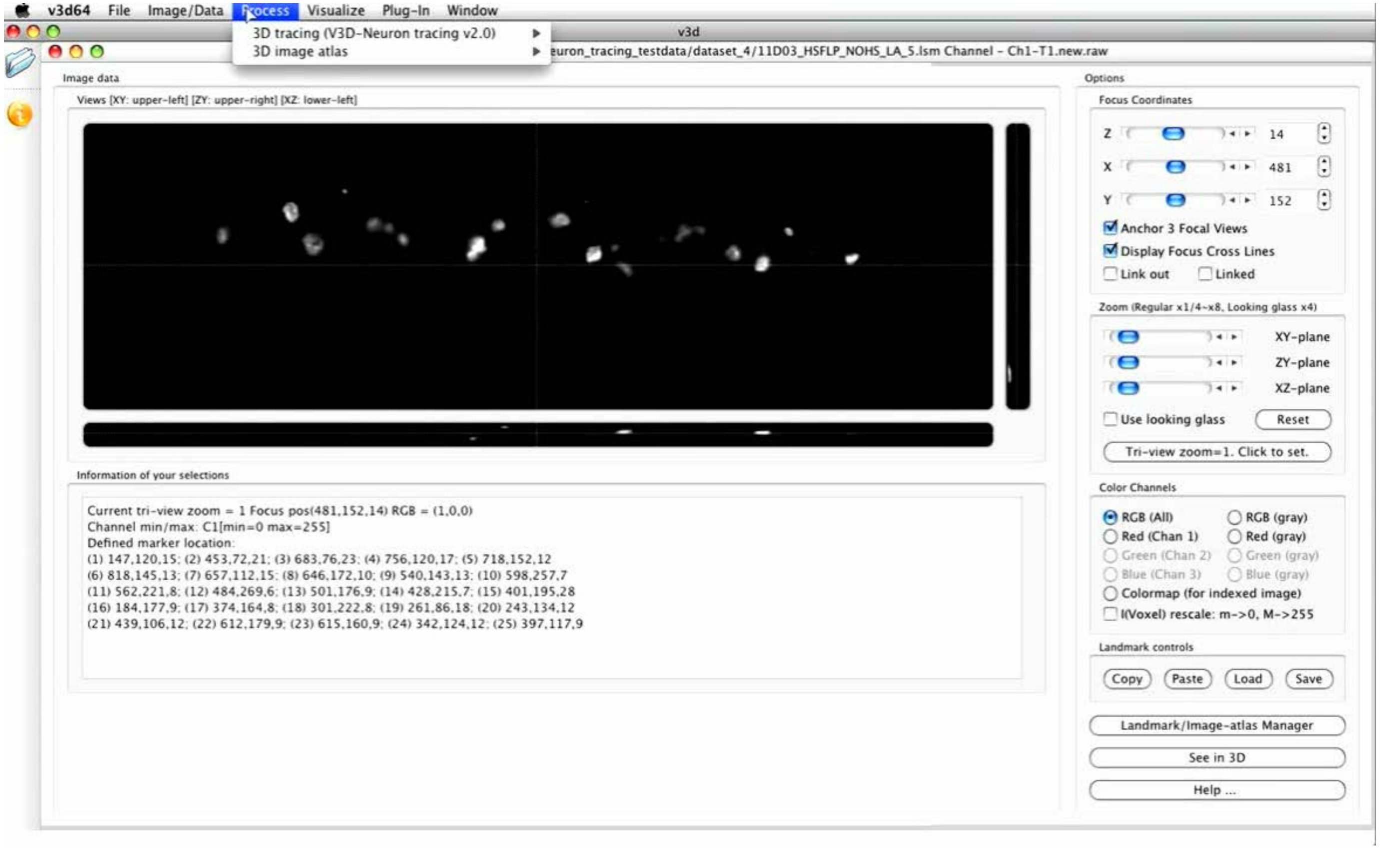
- automated microscopes and the increase in resolution has led to **bioimage data explosion**
  - **terascale** has become a reality
- need of automatic processing
  - fully- or semi-automatic?
  - **human intervention** might be needed
    - post-processing proofreading
    - semi-automatic analysis

⇩

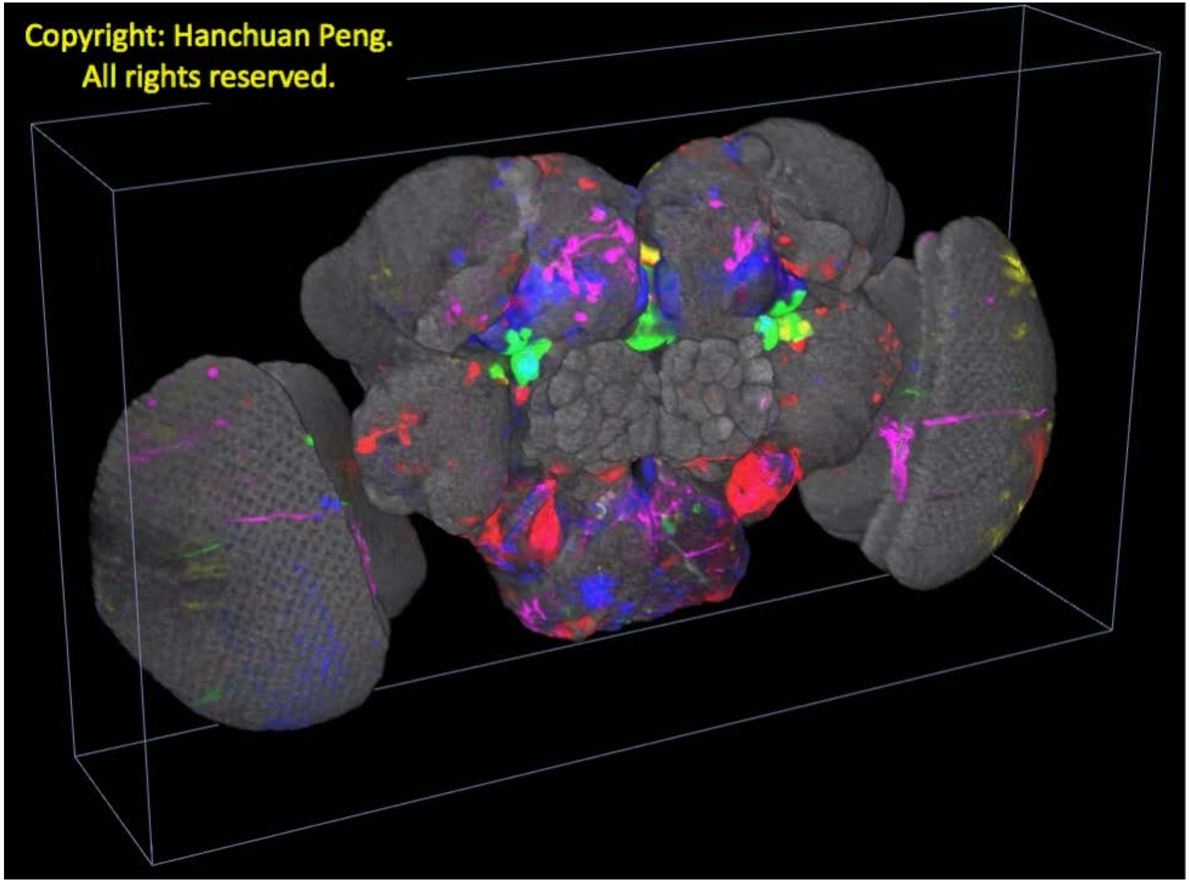
2-5D visualization-assisted analysis



The goal: visualization-assisted analysis of large bioimages **(1/2)**



The goal: visualization-assisted analysis of large bioimages **(2/2)**



The state of the art

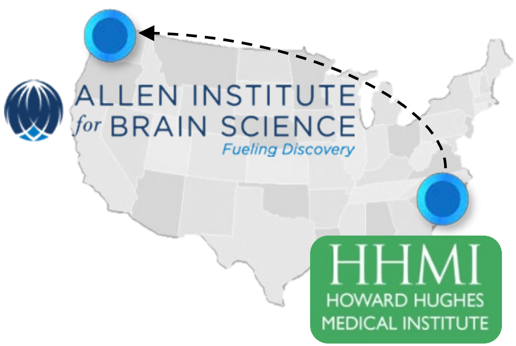
- free and/or open-source visualization tools
  - Voxx, OME, **ImageJ**, Icy, ilastik, CellProfiler, CellOrganizer, CellExplorer, FARSIGHT, Bisque, BrainExplorer, BrainAligner, **3D Slicer**, **ParaView**
- commercial tools
  - Amira (VSG), Imaris (Bitplane), ImagePro (MediaCybernetics), Neurolucida (MBF Bioscience)
- standalone 3-5D visualization-assisted analysis of large images not feasible with <u>any</u> of these tools at present
  - ⛔ large ≠ terascale
  - ⛔ missing 3-5D visualization
  - ⚠ low versatility: supported image formats, cross-platform, etc.
  - ⚠ low extendibility: how many available plugins? Are they easy-to-write?
  - ⛔ high memory requirements: both system RAM and GPU RAM

Vaa3D[1]: enjoying working with 3D image data!

"Vaa3D is designed expressly for working with 3D volumetric data and is built on an efficient 3D renderer that allows real-time visualization and manipulation of multigigabyte–sized data on a standard computer. […] it may not be as ***fun*** *as 3D gaming but Vaa3D promises to make working with 3D image data in the lab much more* ***enjoyable****"*

Daniel Evanko, "*Connecting the dots in 3D*", Nature Methods highlights, 2010

- developed and under development at Peng Lab



Hanchuan Peng

members that left

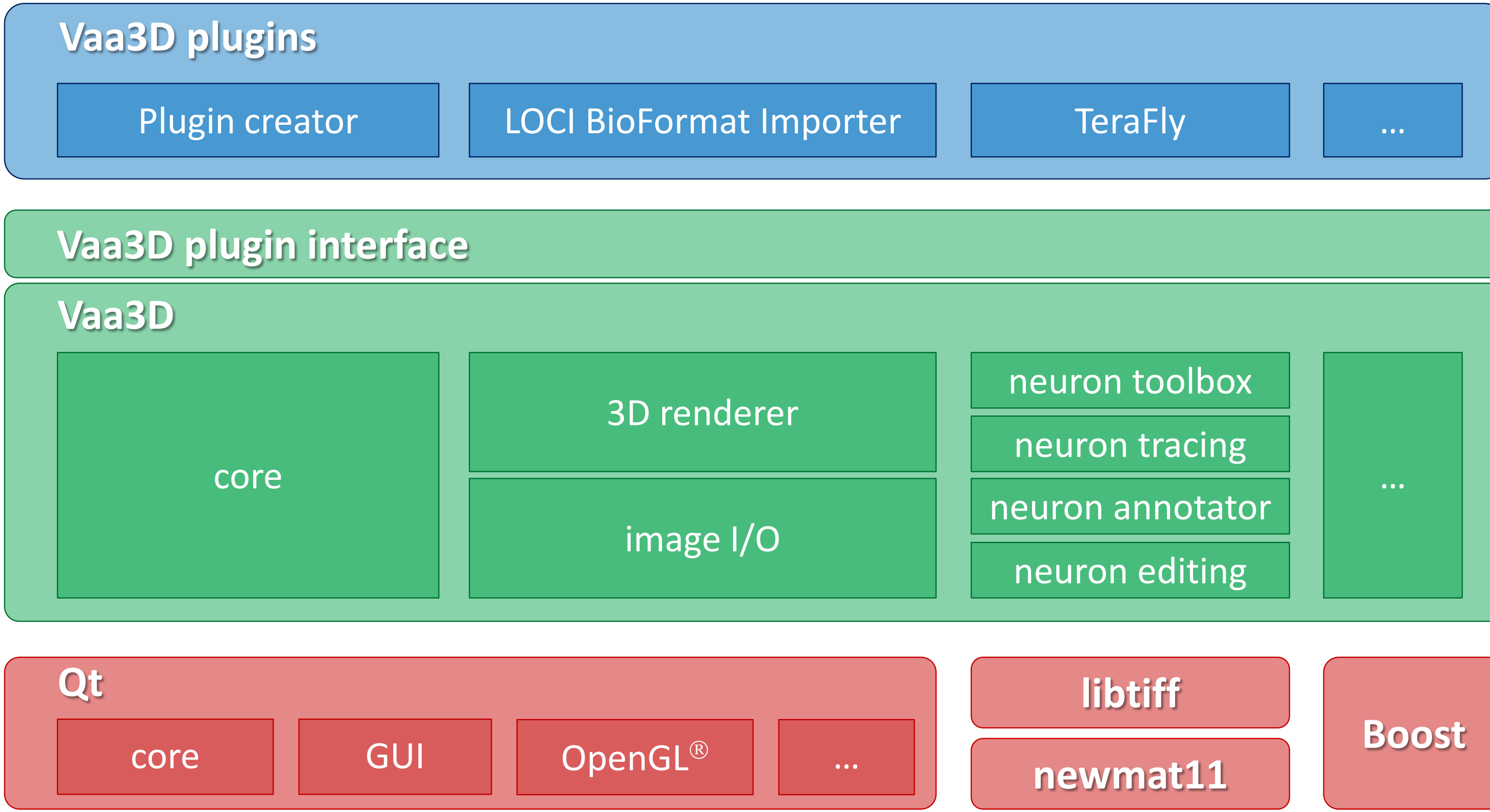
Zhi Zhou

Alessandro Bria

recruiting ⚠

[1] Peng, H. et al, "*V3D enables real-time 3D visualization and quantitative analysis of large-scale biological image data sets*", Nature Biotechnology 28, 348-353, 2010

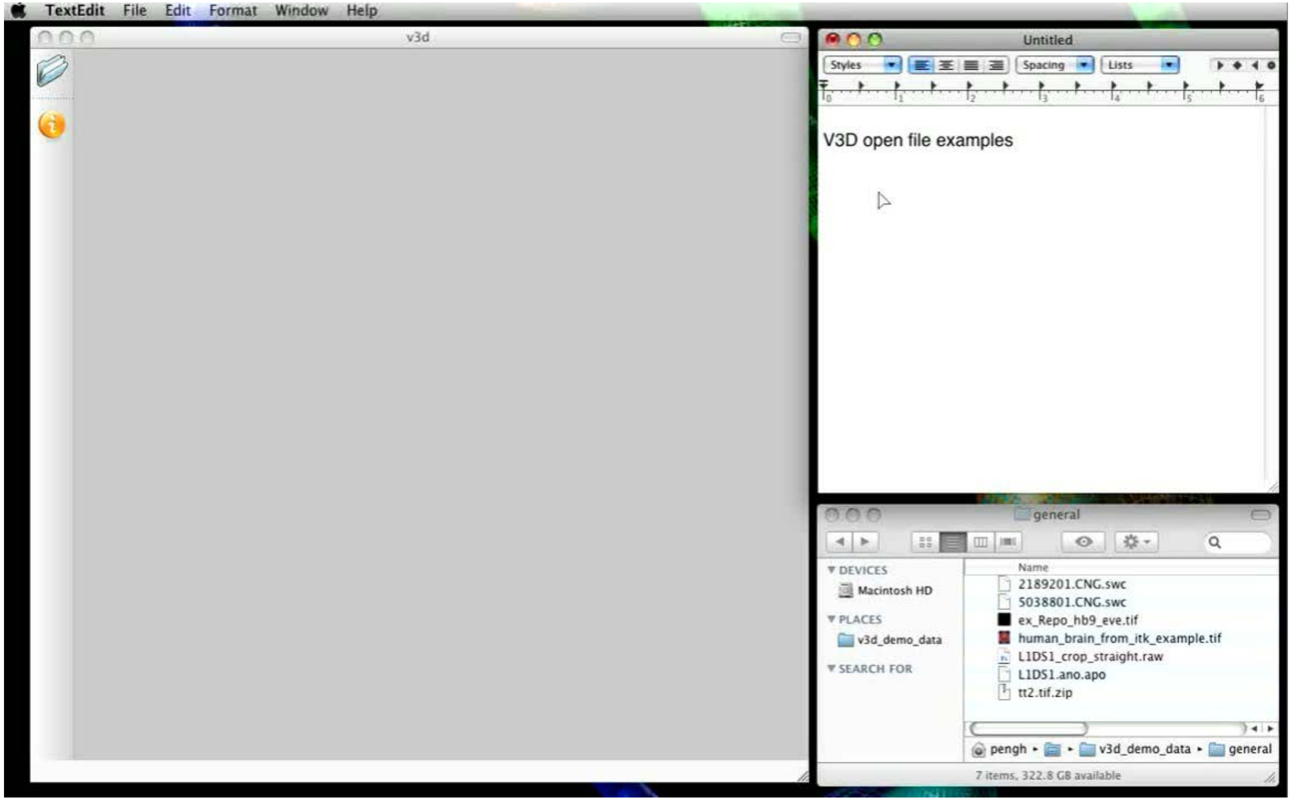
Vaa3D: architecture



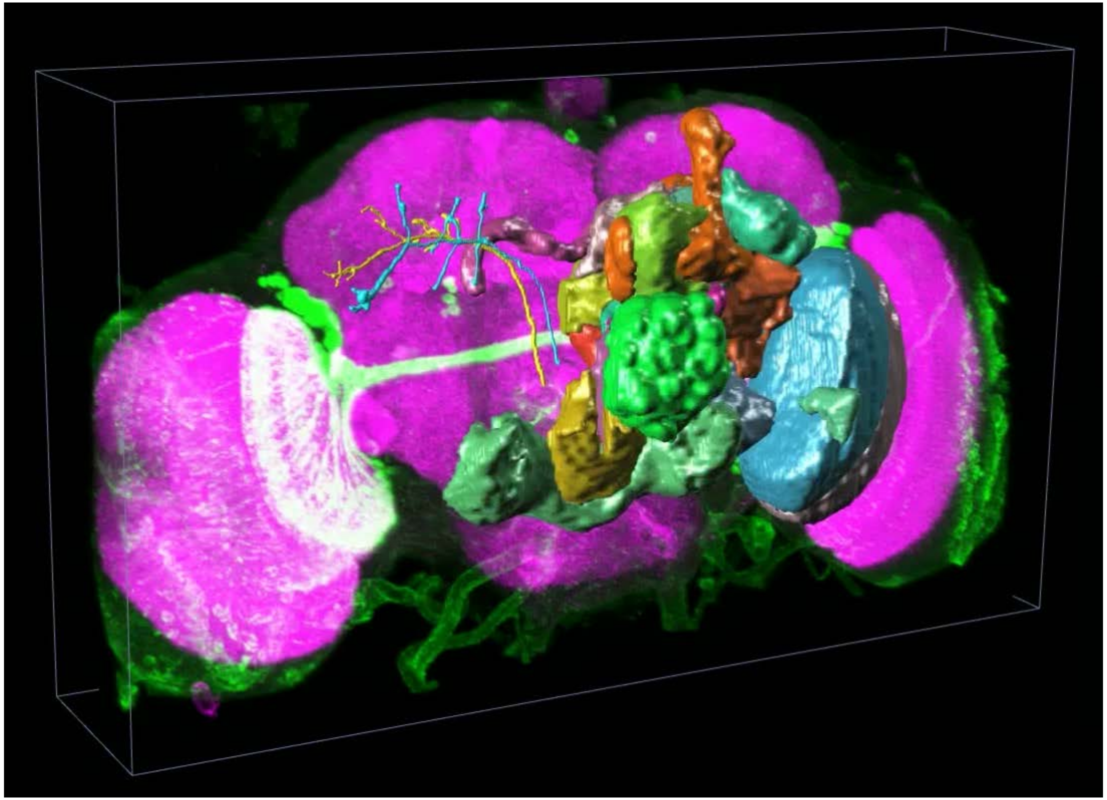
Vaa3D: supported bioimage formats

- *3D color image stacks*
  - Tiff stack (.tif, .tiff), Zeiss LSM (.lsm), MRC (used for electron microscopy images) (.mrc), Vaa3D's raw file (.v3draw, .raw)
  - **any bioimage format** supported by **LOCI Bioinformats Java library** (using the Vaa3D-bioformats plugin)
- *5D time series of color image stacks*
  - each time point saved as a separate file (end with suffix like 000.tif, 001.tif, ...)
  - each time point saved as a single slice of a 3D image stack of whatever formats Vaa3D supports (e.g. tiff, or Vaa3D's raw)
- *3D irregular shaped surfaces*: Wavefront .OBJ files, Vaa3D’s surface format (.v3ds)
- *3D point cloud*: .apo file (a simple CSV format with fixed number of columns)
- *3D landmarks*: .marker (indeed a simple CSV format), .csv

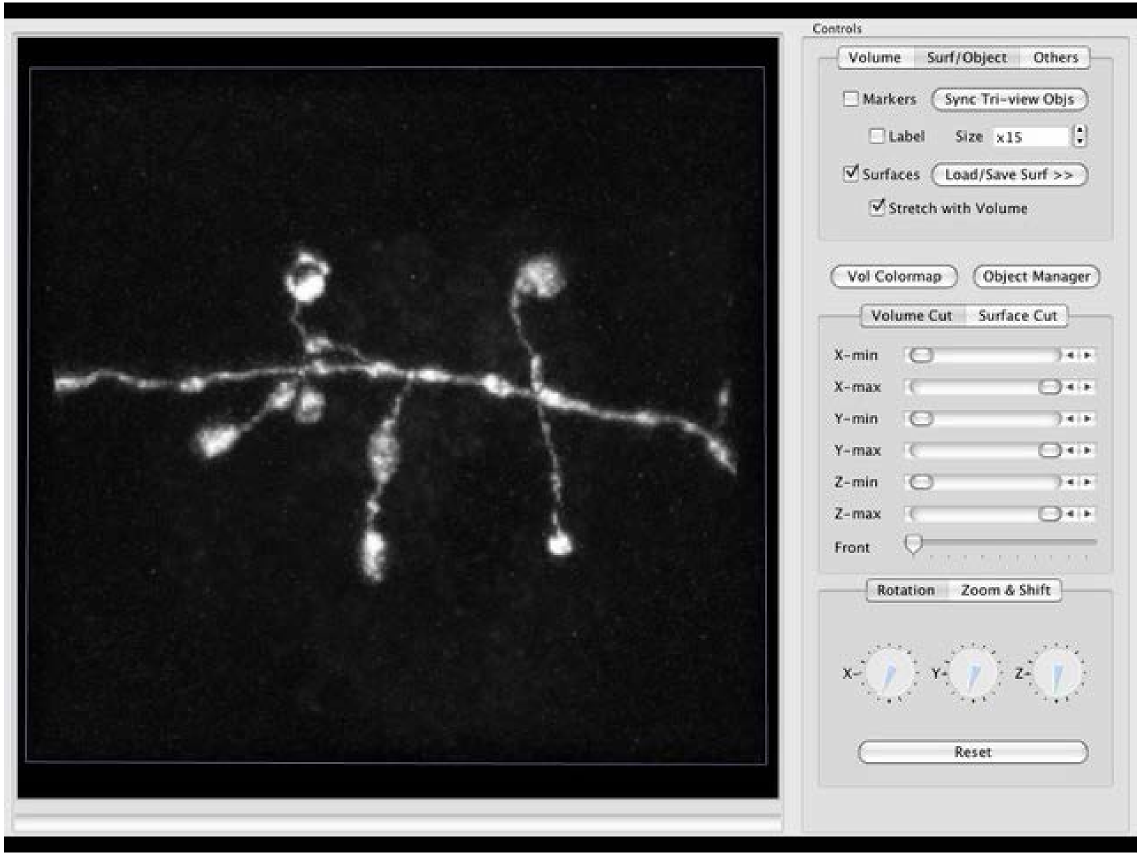
Vaa3D: basic use



Vaa3D: surface rendering and creation



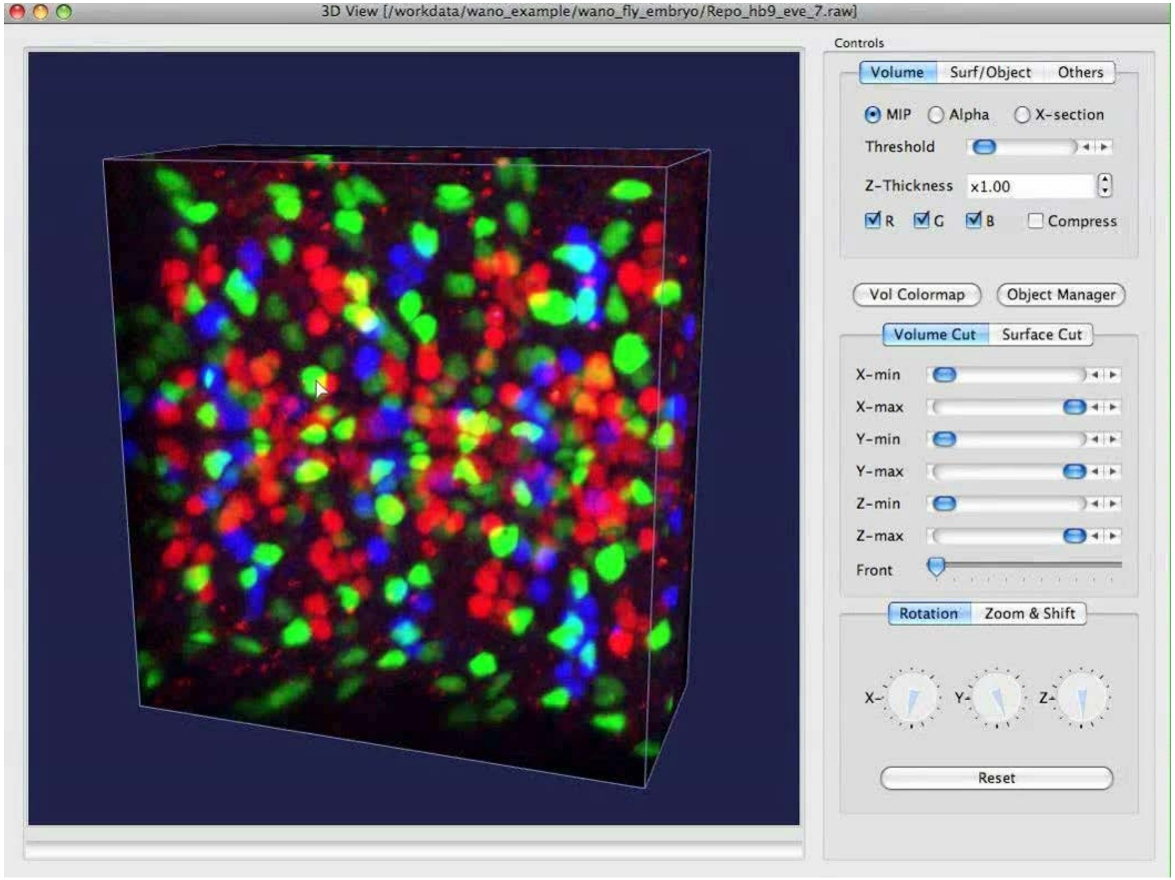
Vaa3D: neuron editing



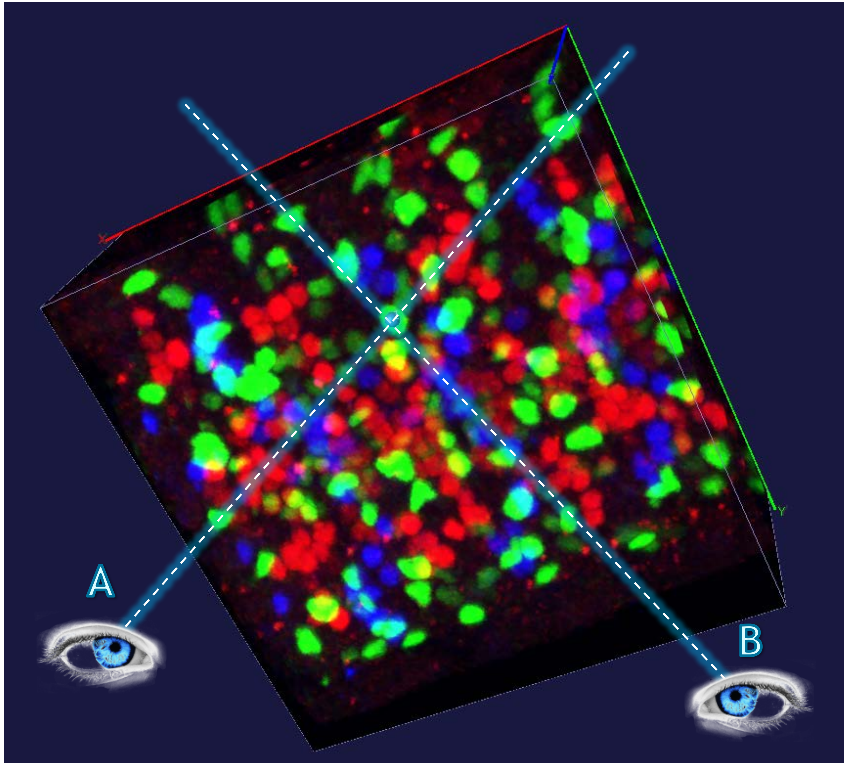
Vaa3D: visualizing 5D data

- On-air demostration

Vaa3D: 2-mouse click 3D pinpointing (1/2)

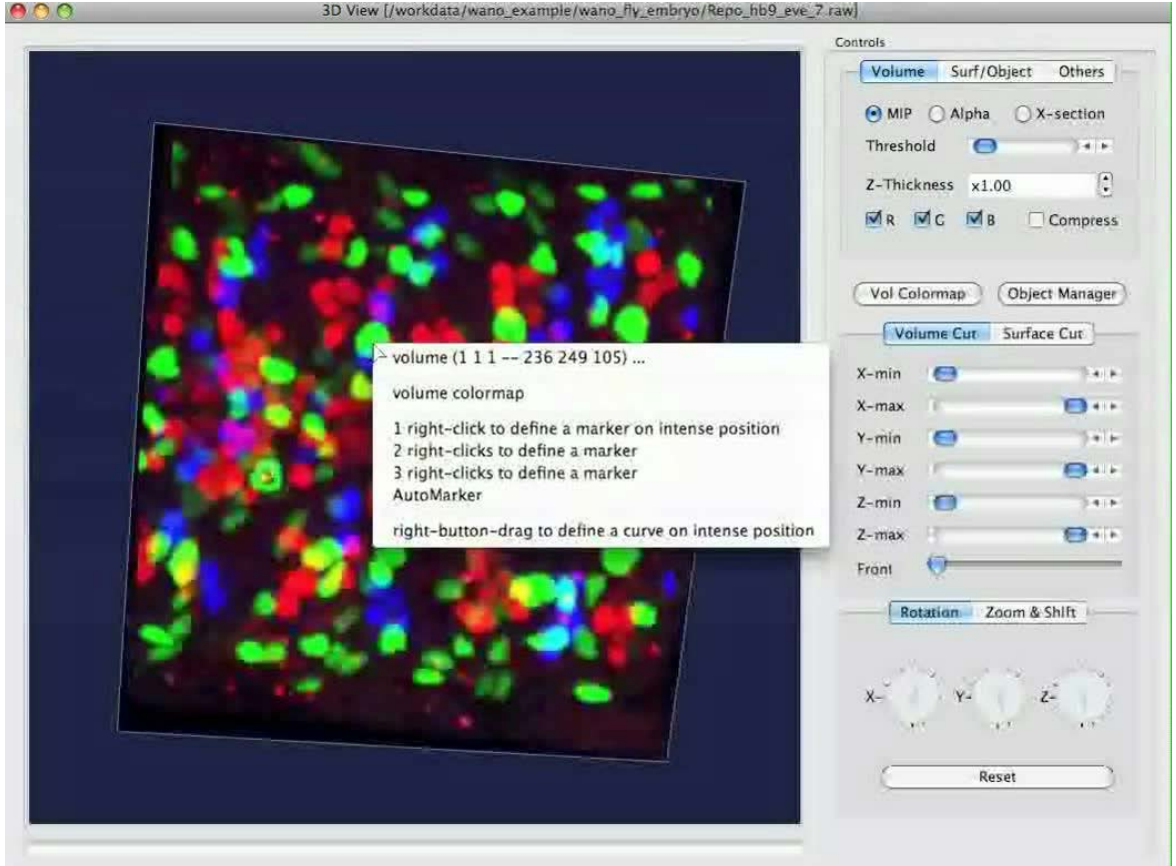


Vaa3D: 2-mouse click 3D pinpointing (2/2)

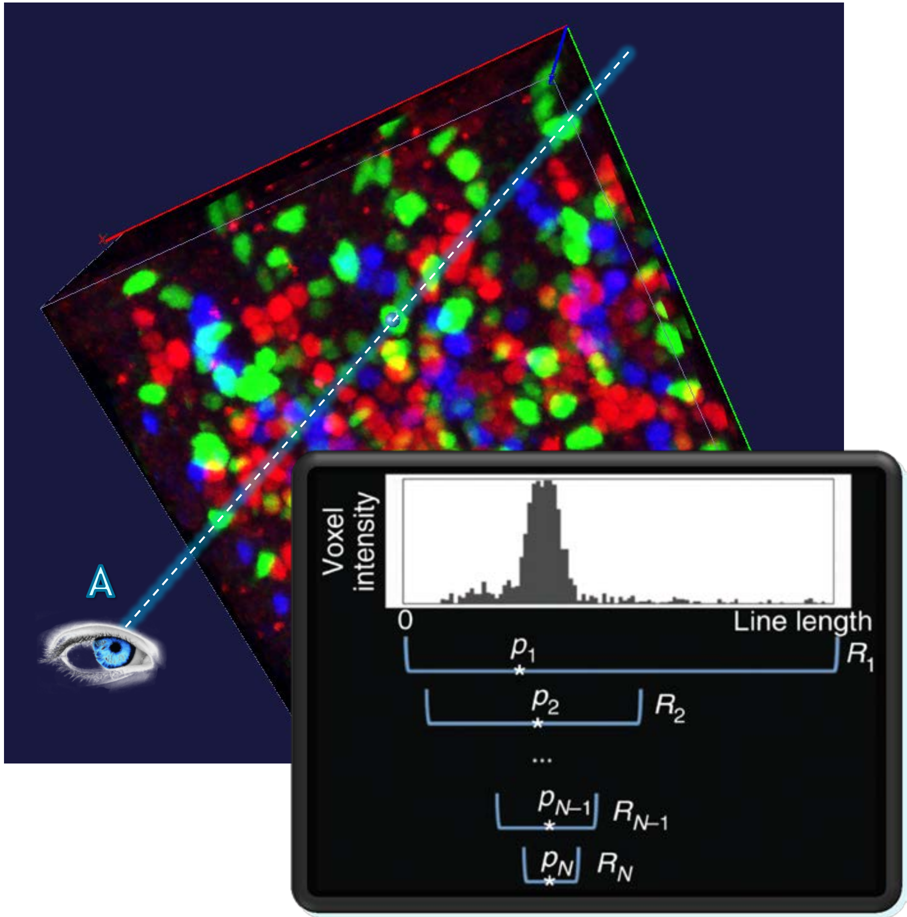


- be A and B the two non-parallel rays generated at two viewing angles, corresponding to 2-mouse clicks
  - each click defines a ray through the current cursor location orthogonal to the screen
- a marker is created at the point in space for which the sum of its Euclidean distance to A and B is minimal
  - robust to inaccuracy in the user's 2D clicks

Vaa3D: 1-mouse click 3D pinpointing **(1/2)**

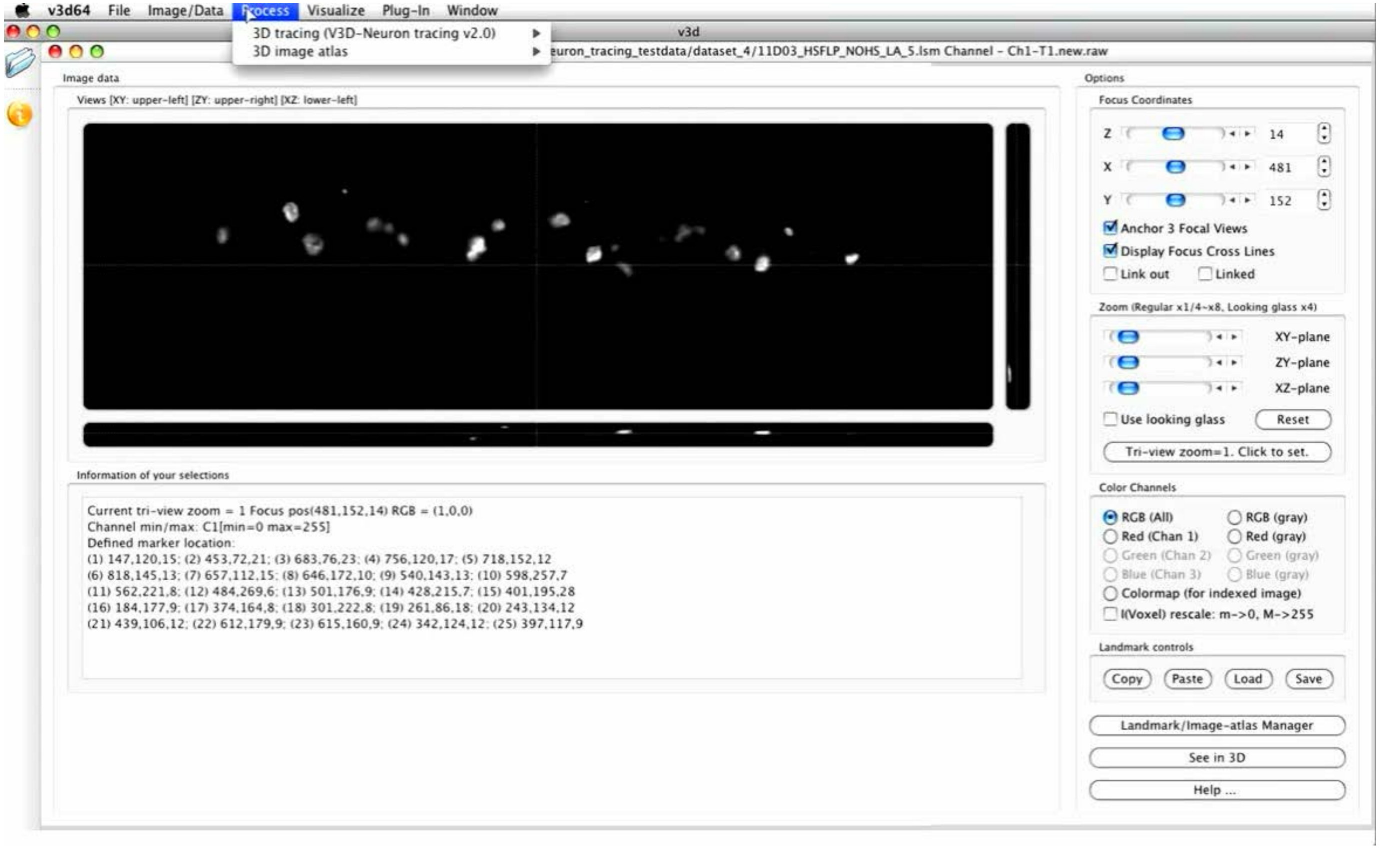


Vaa3D: 1-mouse click 3D pinpointing (2/2)

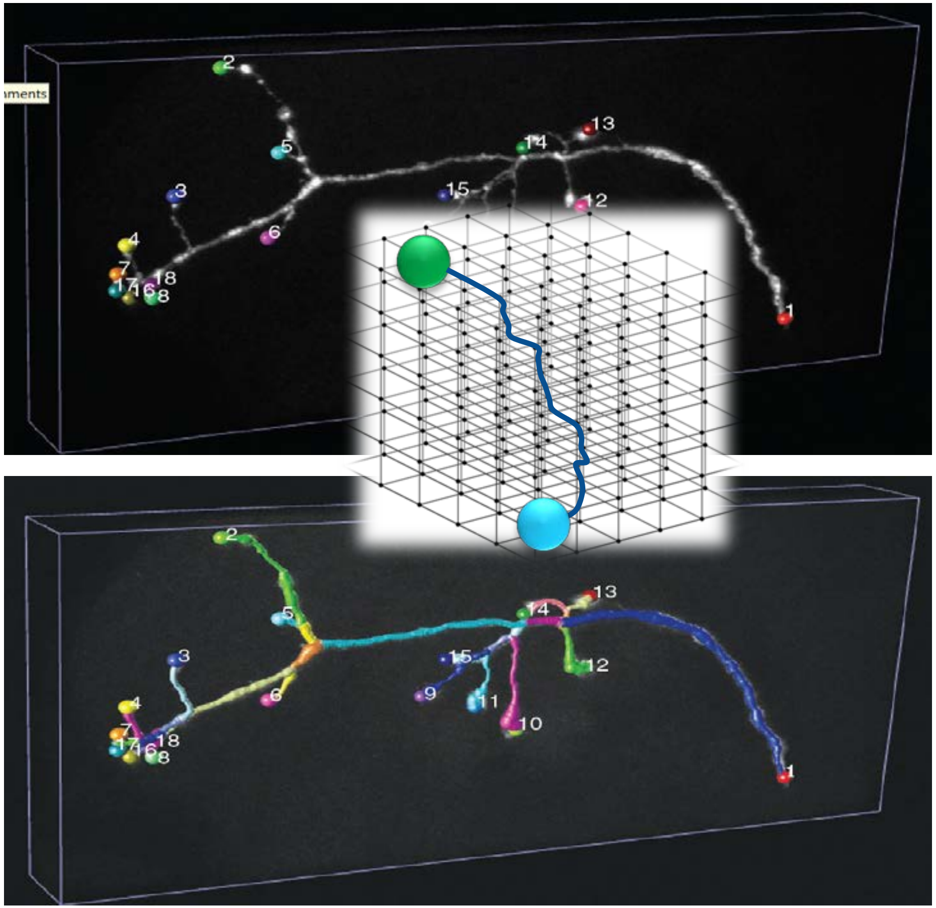


- the most probable location on A is estimated by applying the **mean-shift** algorithm on the intensity distribution
  - the initial center of mass (CoM) p_1 is computed along the whole ray $[0, R_1]$ intersecting the volume
  - the CoM is repeatedly reestimated by using progressively smaller intervals around the proceeding CoM until convergence
- can be used for quick manual cell-counting or for quantitatively profiling the voxel intensity along the straight line segment connecting two markers

Vaa3D: semi-automatic neuron tracing (1/2)

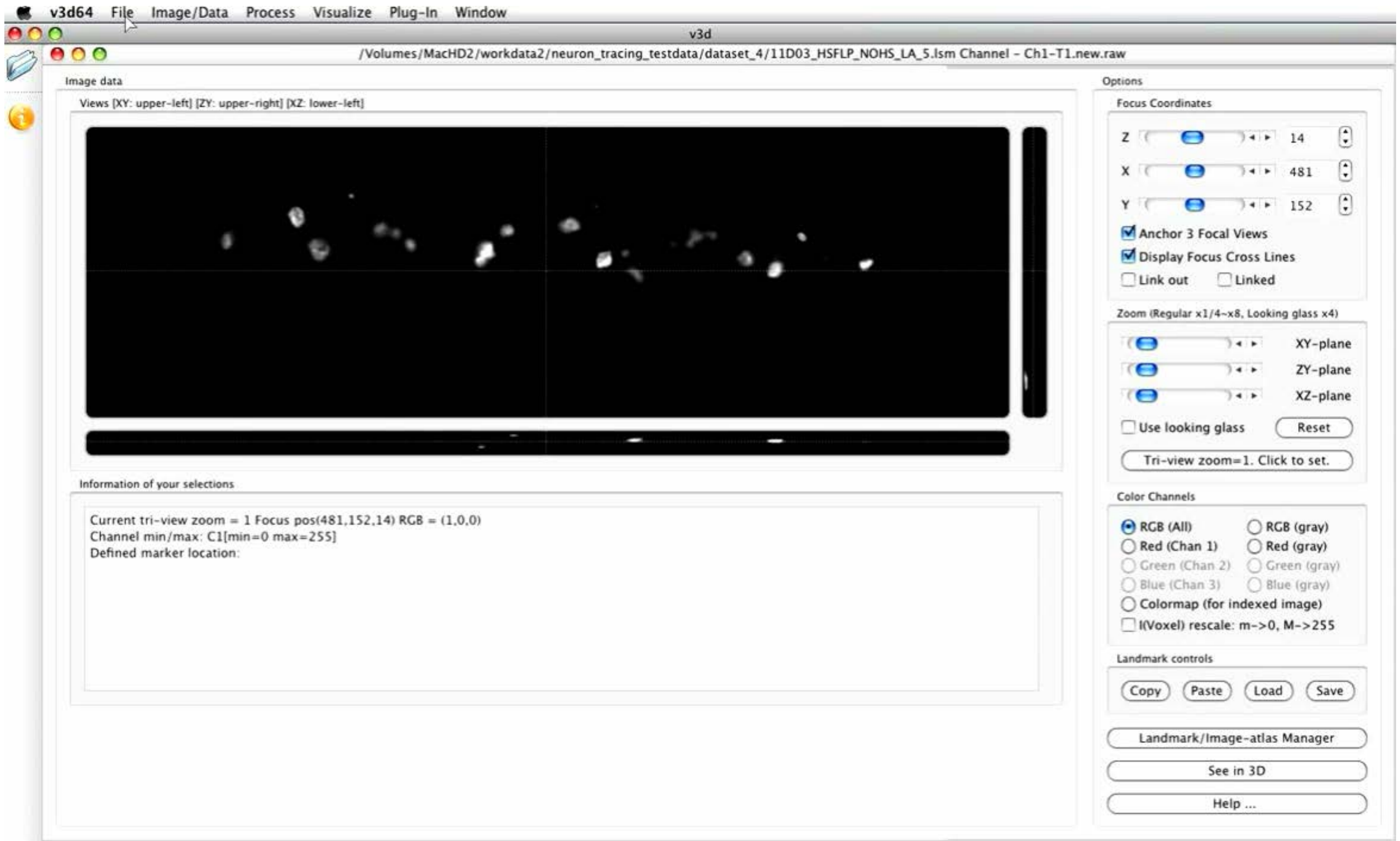


Vaa3D: semi-automatic neuron tracing (2/2)



- searching the "optimal path" connecting a set of markers
  - voxels are considered as graph vertexes
  - edges connect each pair of adjacent voxels
  - edge weight is the inverse of the average intensity of the two voxels
  - Dijkstra's algorithm is used to find the least-cost path between pairs of markers
- individual segments are defined as paths between markers and branching points

Vaa3D: 1-stroke 3D curving



Vaa3D: built-in plugins

- Vaa3D plugin creator
  - Vaa3D plugin creator
- celegans
  - atlasguided seganno
  - celegans straighten
- data IO
  - bioimageIO Bioformat
- data type
  - 5D stack Converter
  - datatype converter
  - Convert Image to AtlasViewMode
- image blending
  - blend multiscanstcks
- image edge detection
  - edge of maskimg
- image filters
  - Distance Transform
  - Gaussian Filter
  - Greyscale Distance Transform
  - minMaxfilter
- image geometry
  - montage image sections
  - recenterimage
  - Rotate Image
- image registration
  - ssd registration
- image resolution
  - XYZ Resolution
- image ROI
  - ROI Editor
- image stitching
  - ifusion
  - istitch
  - Map View
- image thresholding
  - Simple adaptive thresholding
- linker file
  - Linker File Generator
- movies
  - Simple Movie Maker
- neuron utilities
  - Enhanced SWC Format Converter
  - Global Neuron Feature
  - Resample SWC
  - Sort SWC
  - SWC to Maskimage
- pixel intensity
  - Canvas Eraser
  - Change Single Pixel Value
- principal skeleton detection
  - **Principal Skeleton Detection**
- reference extract
  - refextract
- Vaa3D PluginInterface Demos
  - 3D viewer data push and display
  - Plugin Call Each Other
  - Mouse Event Monitor
  - Multi Image Interface
  - Single Image interface

a plugin to create plugins!

importing ANY bioimage format

image filters

image registration

image stitching

movie maker

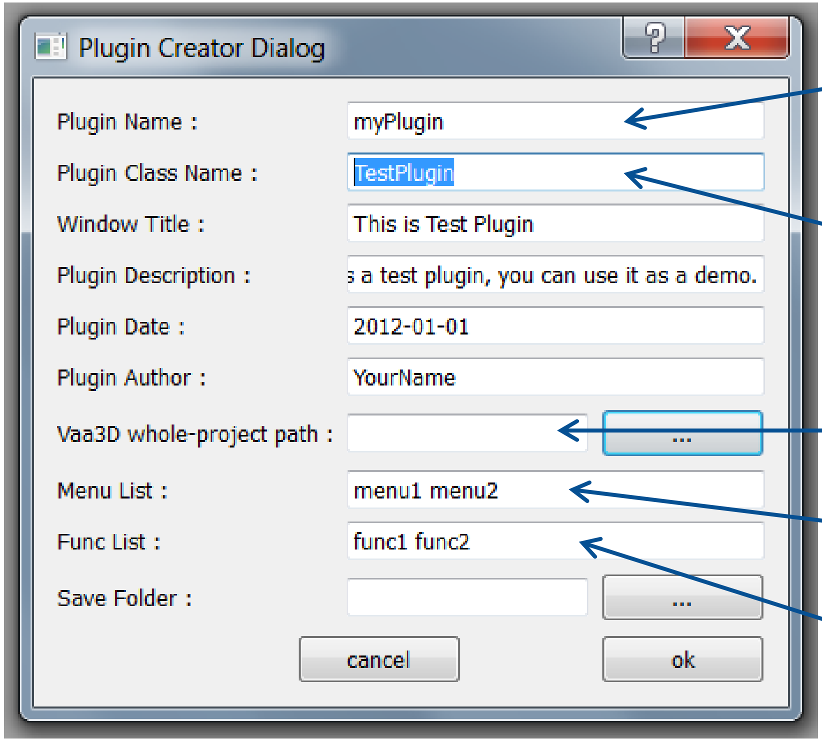
Vaa3D: a tutorial for developing a plugin (1/8)

- Setting up the development environment under Linux / MacOS
  - get and build (or install, if available for your platform) Qt 4.7.2 (or .3, .4)
  - get and build Vaa3D source code by following instructions at https://code.google.com/p/vaa3d/
  - get and install Qt Creator (optional)

- Setting up the development environment under Windows
  - download the precompiled binaries Qt 4.7.2 for Visual Studio 2008
  - check the prerequisites at https://code.google.com/p/vaa3d/
  - get and build Vaa3D source code by following the build instructions for CMake at the Vaa3D's google code webpage. Use Visual Studio 2008 as both project generator and compiler.
  - get and install Qt Creator (optional)

Vaa3D: a tutorial for developing a plugin (2/8)

- Using *Vaa3D plugin creator* for creating a Qt project of a new plugin



The name of the plugin that will appear under the Vaa3D plugins menu

The name of the C++ class interfacing with the Vaa3D plugin loader

Path of the “v3d_external” folder

In-Vaa3D functions (will appear in the menu)

Command-line functions

Vaa3D: a tutorial for developing a plugin (3/8)

- myplugin.pro

```
TEMPLATE        = lib
CONFIG          += qt plugin warn_off
VAA3DPATH       = D:\Vaa3D\v3d_external
INCLUDEPATH     += $$VAA3DPATH/v3d_main/basic_c_fun

HEADERS         += myPlugin_plugin.h
SOURCES         += myPlugin_plugin.cpp
SOURCES         +=
$$VAA3DPATH/v3d_main/basic_c_fun/v3d_message.cpp

TARGET          = $$qtLibraryTarget(myPlugin)
DESTDIR         = $$VAA3DPATH/bin/plugins/myPlugin/
```

configuring the current project as a Qt plugin

path of the “v3d_external” folder

plugin headers and source files

Vaa3D core functions, I/O, etc.

Vaa3D: a tutorial for developing a plugin (4/8)

- myplugin_plugin.h

```
#ifndef __MYPLUGIN_PLUGIN_H__
#define __MYPLUGIN_PLUGIN_H__

#include <QtGui>
#include <v3d_interface.h>

class CPlugin : public QObject, V3DPluginInterface2_1
{
   Q_OBJECT
   Q_INTERFACES(V3DPluginInterface2_1);

public:
   float getPluginVersion() const {return 1.1f;}

   QStringList menulist() const;
   void domenu(const QString &menu_name,
    V3DPluginCallback2 &callback, QWidget *parent);

   QStringList funclist() const ;
   bool dofunc(const QString &func_name, const
    V3DPluginArgList &input, V3DPluginArgList &output,
    V3DPluginCallback2 &callback, QWidget *parent);
};

#endif
```

configuring the current class as a Qt plugin

a Qt class MUST start with Q_OBJECT

generates the plugin menu in Vaa3D

processes the user’s menu selection

command-line interaction

Vaa3D: a tutorial for developing a plugin (5/8)

- `myplugin_plugin.cpp`

```
Q_EXPORT_PLUGIN2(myPlugin, CPlugin);

QStringList CPlugin::menulist() const
{
    return QStringList()
           <<tr("thresholding")
           <<tr("about");
}

void CPlugin::domenu(const QString &menu_name,
                     V3DPluginCallback2 &callback,
                     QWidget *parent)
{
    if (menu_name == tr("thresholding"))
    {
        v3d_msg("To be implemented.");
    }
    else
    {
        v3d_msg(tr("This is a test plugin, you can
use it as a demo..Developed by Alessandro
Bria, 2012-01-01"));
    }
}
```

- `Q_EXPORT_PLUGIN2`: configuring the current class as a Qt plugin named *myPlugin.* The name must match the TARGET parameter in the .pro
- `menulist()`: generates the plugin menu in Vaa3D
- `domenu()`: processes the user's menu selection
- `v3d_msg("To be implemented.")`: displays a message to the user when selecting "thresholding" from the plugin menu
- `v3d_msg(tr("This is a test plugin…"))`: Displaying an "About" message when the user selects "about" from the plugin menu

Vaa3D: a tutorial for developing a plugin (6/8)

- V3DPluginCallback (1/3)

```
////invoke a Vaa3D plugin function
        virtual bool callPluginFunc(const QString & plugin_name, const QString & func_name,
                                    const V3DPluginArgList & input, V3DPluginArgList & output) = 0;

////get opened images

        //obtain a list of all currently opened images
        virtual v3dhandleList getImageWindowList() const = 0;

        //obtain the *current* selected image window, defined as the tri-view window currently selected in Vaa3D main window
        virtual v3dhandle currentImageWindow() = 0;

        //obtain the *current* selected image window, defined as the currently being operated 3D viewer
        //curHiddenSelectedWindow may not be the *currentImageWindow* if the selection is done from a 3d viewer
        virtual v3dhandle curHiddenSelectedWindow() = 0;

////set computed image content/result

        //create a new image window for returning some computed image content
        virtual v3dhandle newImageWindow(QString name="new_image") = 0;

        //directly output computed image content to an existing image window. The size of the window will be changed
        //automatically.
        virtual void updateImageWindow(v3dhandle image_window) = 0;
```

Vaa3D: a tutorial for developing a plugin (7/8)

- V3DPluginCallback (2/3)

```
////manipulate image names

        virtual QString getImageName(v3dhandle image_window) const = 0;
        virtual void setImageName(v3dhandle image_window, QString name) = 0;

////access the actual 4D image data structure

        virtual Image4DSimple * getImage(v3dhandle image_window) = 0;
        virtual bool setImage(v3dhandle image_window, Image4DSimple * image) = 0;

////access the 3D landmark list defined for an image

        virtual LandmarkList  getLandmark(v3dhandle image_window) = 0;
        virtual bool setLandmark(v3dhandle image_window, LandmarkList & landmark_list) = 0;

////access the 3D region of interest (ROI) defined for an image

        virtual ROIList getROI(v3dhandle image_window) = 0;
        virtual bool setROI(v3dhandle image_window, ROIList & roi_list) = 0;

////access the 3D curve, 3D curve set, and 3D reconstructed neuron structure for an image

        virtual NeuronTree getSWC(v3dhandle image_window) = 0;
        virtual bool setSWC(v3dhandle image_window, NeuronTree & nt) = 0;
```

Vaa3D: a tutorial for developing a plugin (8/8)

- V3DPluginCallback (3/3)

```
////operating (open/close) the rendering windows

        //open and close a global 3D viewer
        virtual void open3DWindow(v3dhandle image_window) = 0;
        virtual void close3DWindow(v3dhandle image_window) = 0;

        //open and close a local 3D viewer
        virtual void openROI3DWindow(v3dhandle image_window) = 0;
        virtual void closeROI3DWindow(v3dhandle image_window) = 0;

////Data pushing functions

        //update the surface objects currently being displayed in a 3D viewer
        virtual void pushObjectIn3DWindow(v3dhandle image_window) = 0;

        //update the content in a 3D viewer directly
        virtual void pushImageIn3DWindow(v3dhandle image_window) = 0;

        //update the time point of a 3D viewer if it is displaying 5D (temporal) data
        virtual int pushTimepointIn3DWindow(v3dhandle image_window, int timepoint) = 0;

////direct pointers to Vaa3D internal data structure

        virtual View3DControl * getView3DControl(v3dhandle image_window) = 0;
        virtual View3DControl * getLocalView3DControl(v3dhandle image_window) = 0;
        virtual TriviewControl * getTriviewControl(v3dhandle image_window) = 0;
```

Direct access to Vaa3D core!
To be used with care.

Vaa3D: implementing an example plugin

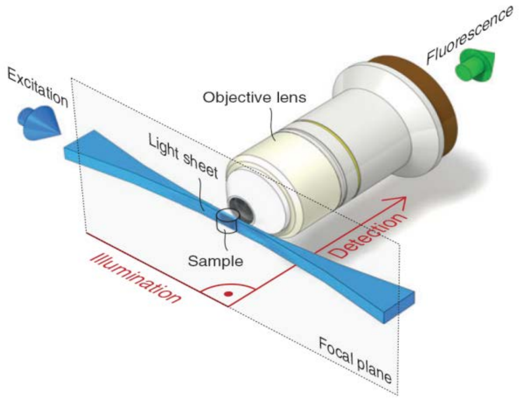
- On-air demostration

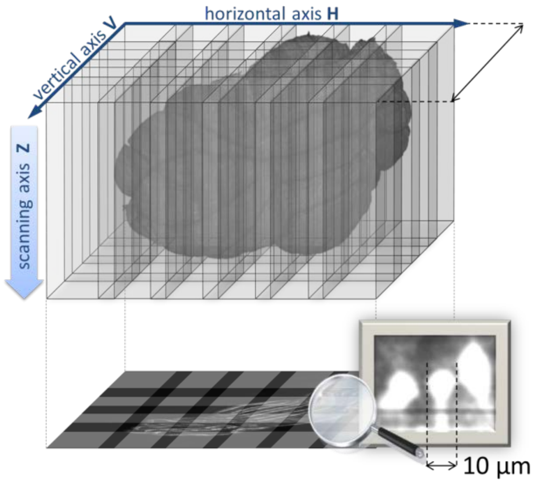
Moving towards terascale bioimages: The Projectome Project

- High Performance Computational Infrastructure for processing and visualizing neuro-anatomical information obtained using CLSM
- Whole mouse brain 3D imaging with Confocal Light-Sheet Microscopy:
  - micrometer resolution
  - cm-sized specimen
  - TeraVoxel-sized dataset

International Center of Computational Neurophotonics

Moving towards terascale bioimages: CLSM microscopy

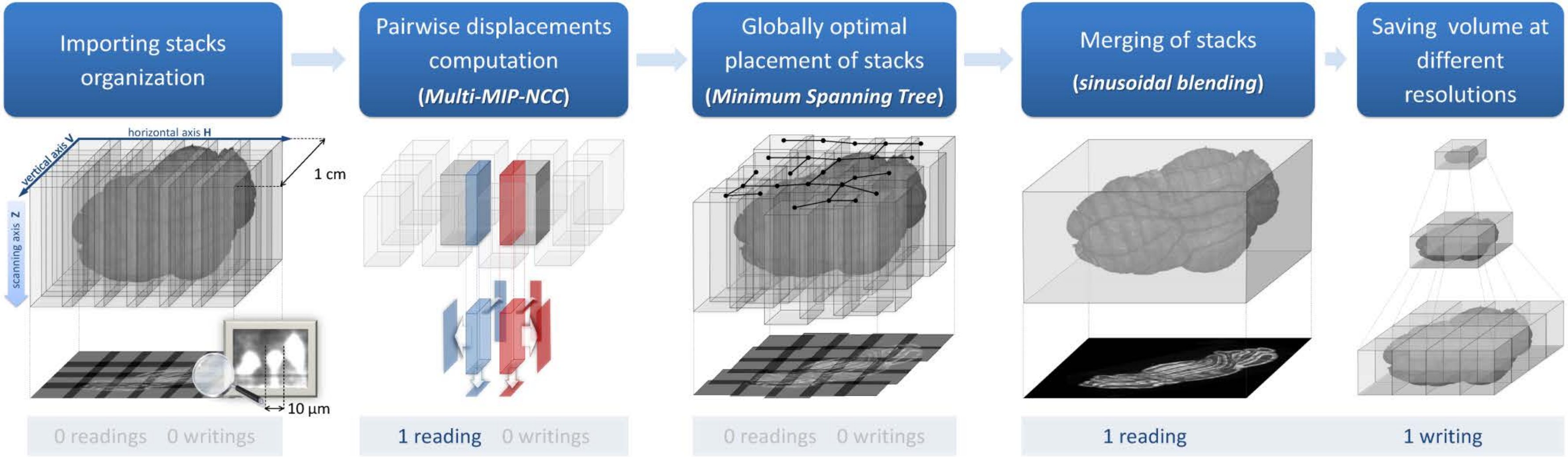




- neurons and other structures are selectively labeled with a fluorescent protein
- the specimen is optically cleared and fixed
- the specimen is illuminated by a thin sheet of light and the fluorescence emission is observed from the scanning axis perpendicular to the illumination plane
- translations of the light sheet along the scanning axis produce a stack of 2D slices
- the field of view of the confocal microscope is limited, so translations of the system along V, H axes are needed to produce different overlapping stacks

Moving towards terascale bioimages: TeraStitcher (1/3)

- fast 2D approach to align adjacent stacks
- efficient use of memory resources (< 2GB of memory peak)
- the stitched volume is saved into a multiresolution representation suited for further processing

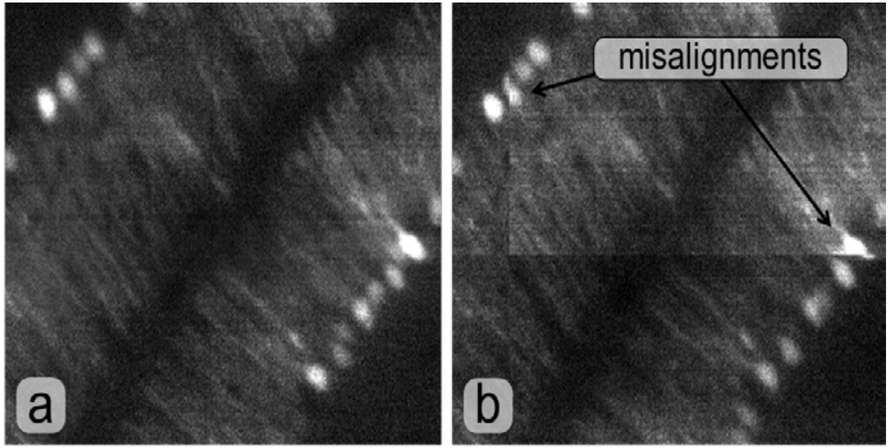


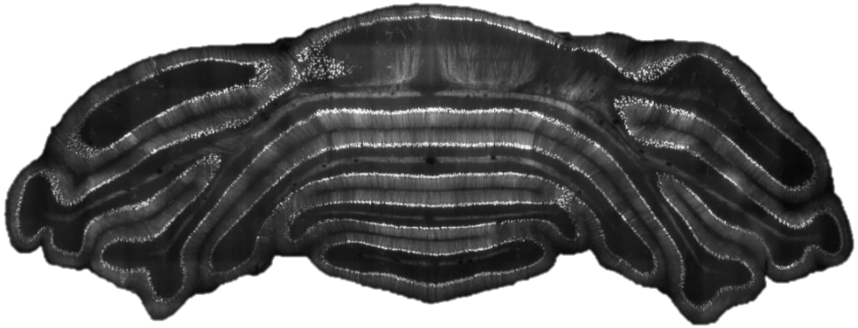
- results

Tool	Displ. comp.	Merging	Total	Memory occ.
Fiji	5.4	8	13.4	800 MB
XuvStitch	3.5	11	14.5	320 MB
iStitch	9	2	11	600 MB
Our tool	2.2	1.7	3.9	200 MB

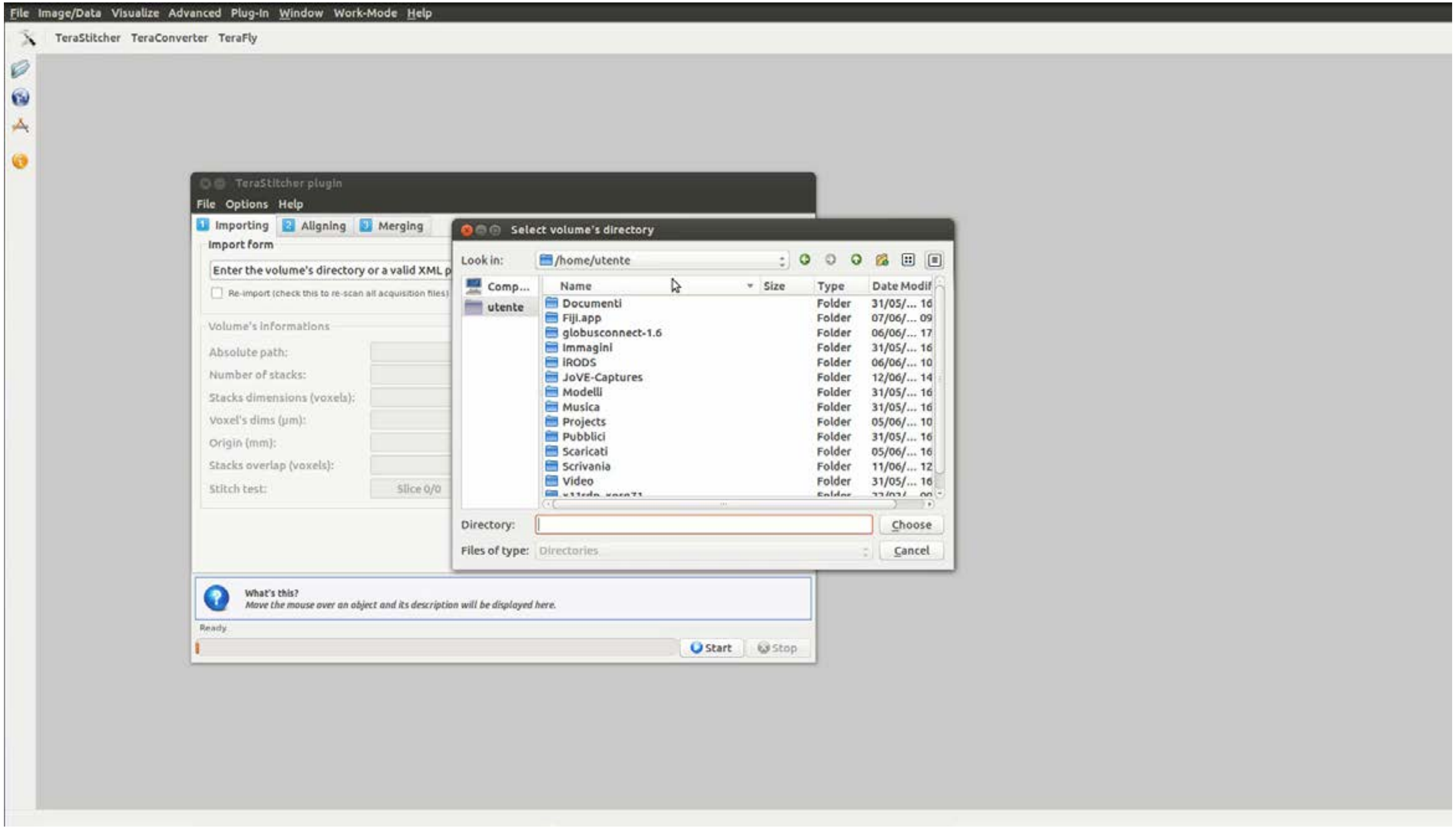
the Vaa3D's built-in stitching plugin

Gvoxel	Displ. comp.	Merging	I/O	Total	Memory occ.
63,25	23	37	157	217	821 MB
126,50	68	77	318	433	1132 MB
198,78	109	123	539	771	1132 MB
1315,04	n.a.	n. a.	n. a.	6050	2450 MB





Moving towards terascale bioimages: TeraStitcher (3/3)



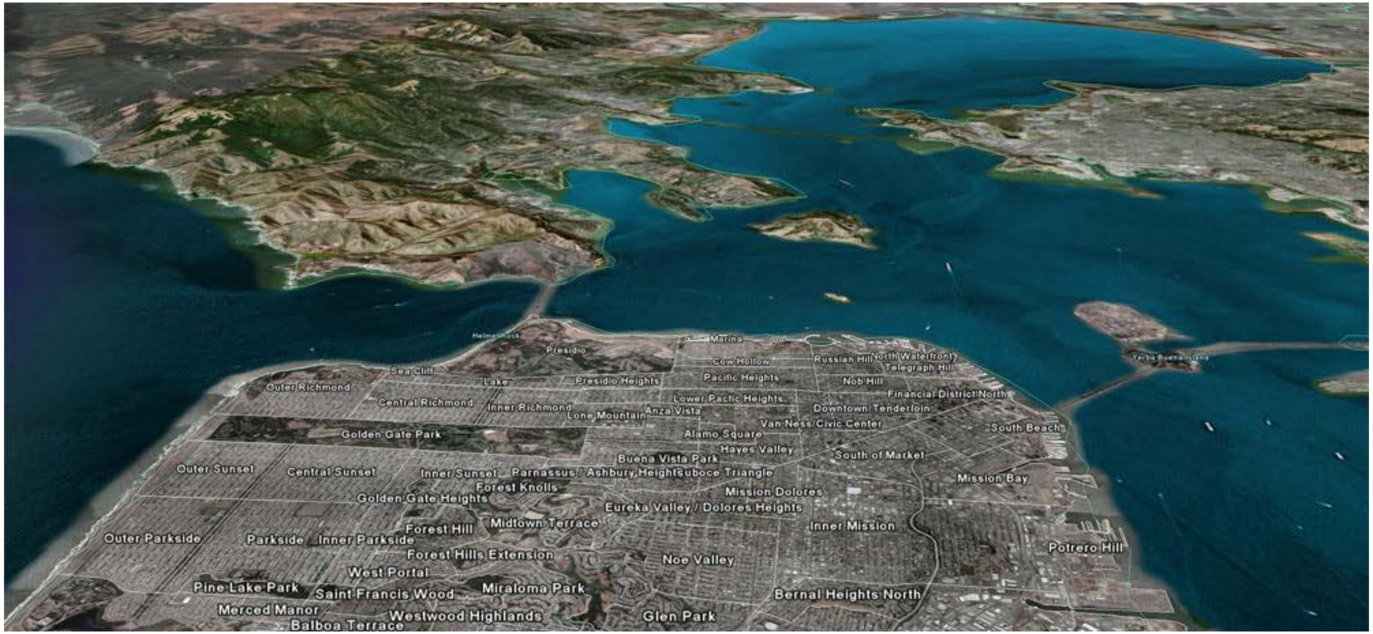
TeraFly: overview (1/3)

- Vaa3D's 3D rendering cannot handle **very large 3D images**
  - e.g. 1 GigaVoxels images require *at least* a video card with 1 GB of dedicated memory

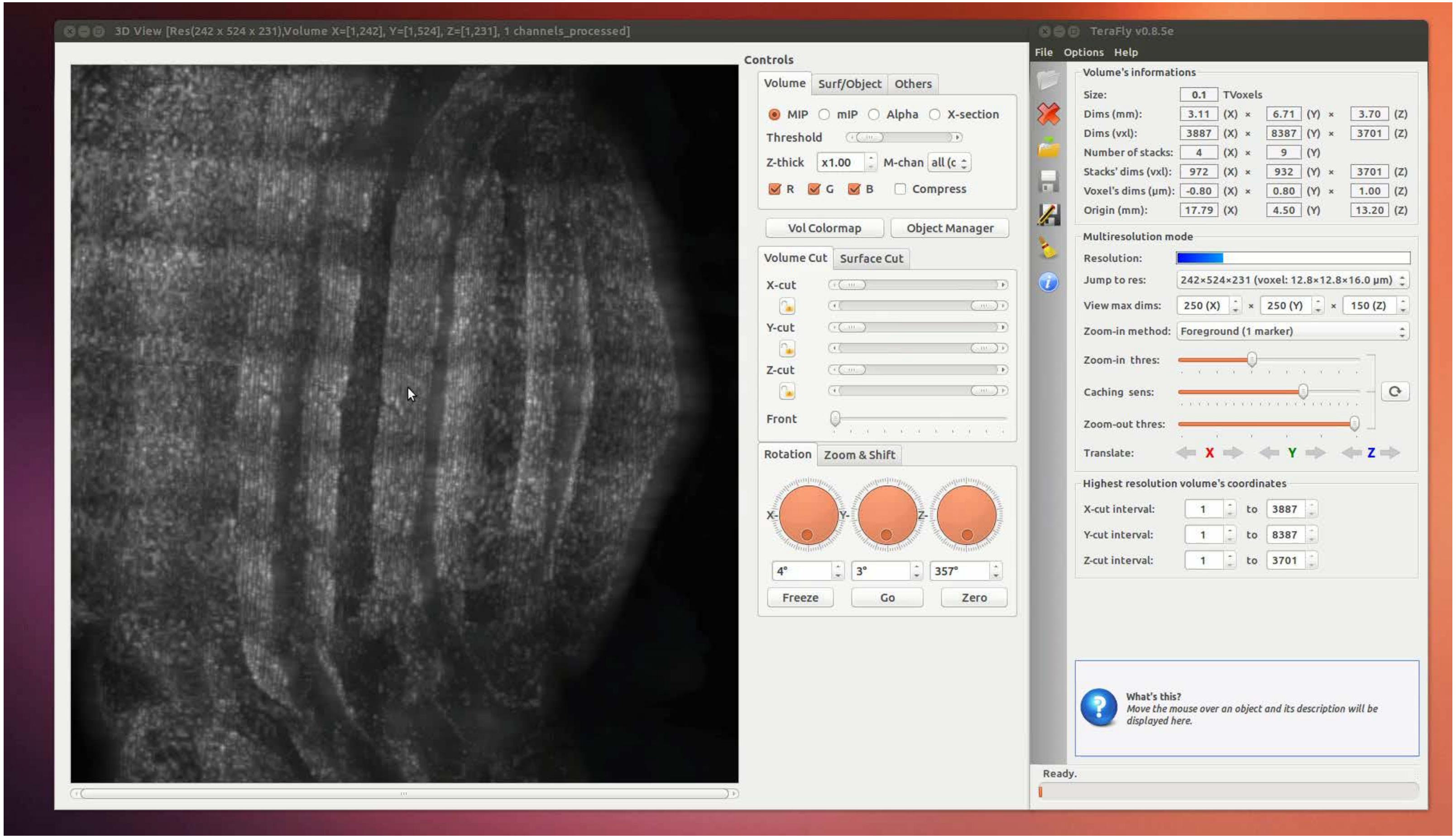
- **TeraFly** extends the Vaa3D software to cope with (potentially) **unlimited** sized bioimages even on laptops with a limited amount of system memory (≤ 4 GB) and video card memory (≤ 1 GB)
  - easy zoom-in/out with mouse-scroll
  - 4D supported (5D support work in progress)
  - automatic scaling of 3D markers and 3D curves throughout 3D navigation
  - annotation of 3D objects
  - basic caching (advanced caching work in progress)
  - separate translations along X, Y, Z
  - separate threads for GPU and I/O
  - fast zoom-in by interpolation + subsequent refinement by image slicing (web-like)

TeraFly: overview (2/3)

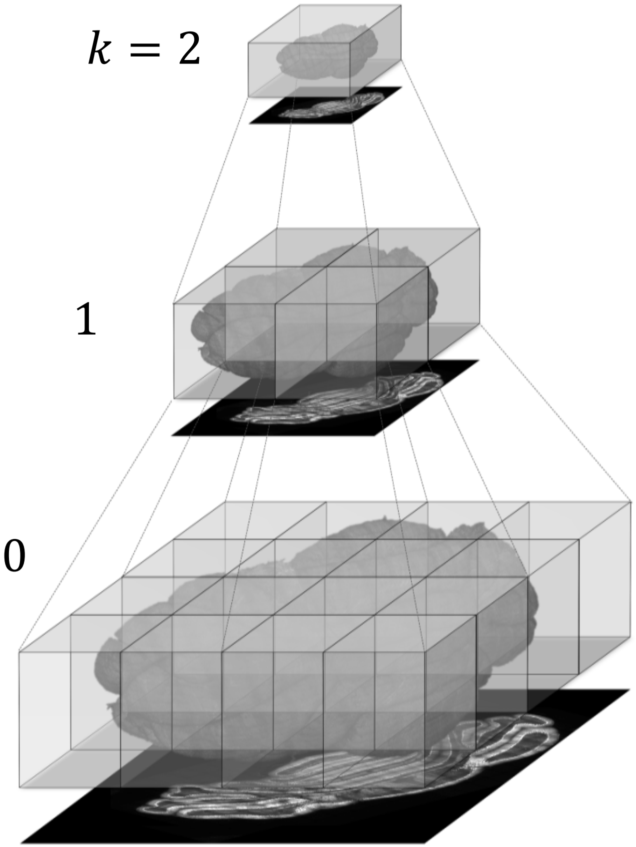
- the underlying idea is to mimic the behavior of **Google Earth**
  - what you see is what you need (WYSIWYN)
  - multiresolution representation
  - mouse scroll for zoom-in/out



TeraFly: overview (3/3)

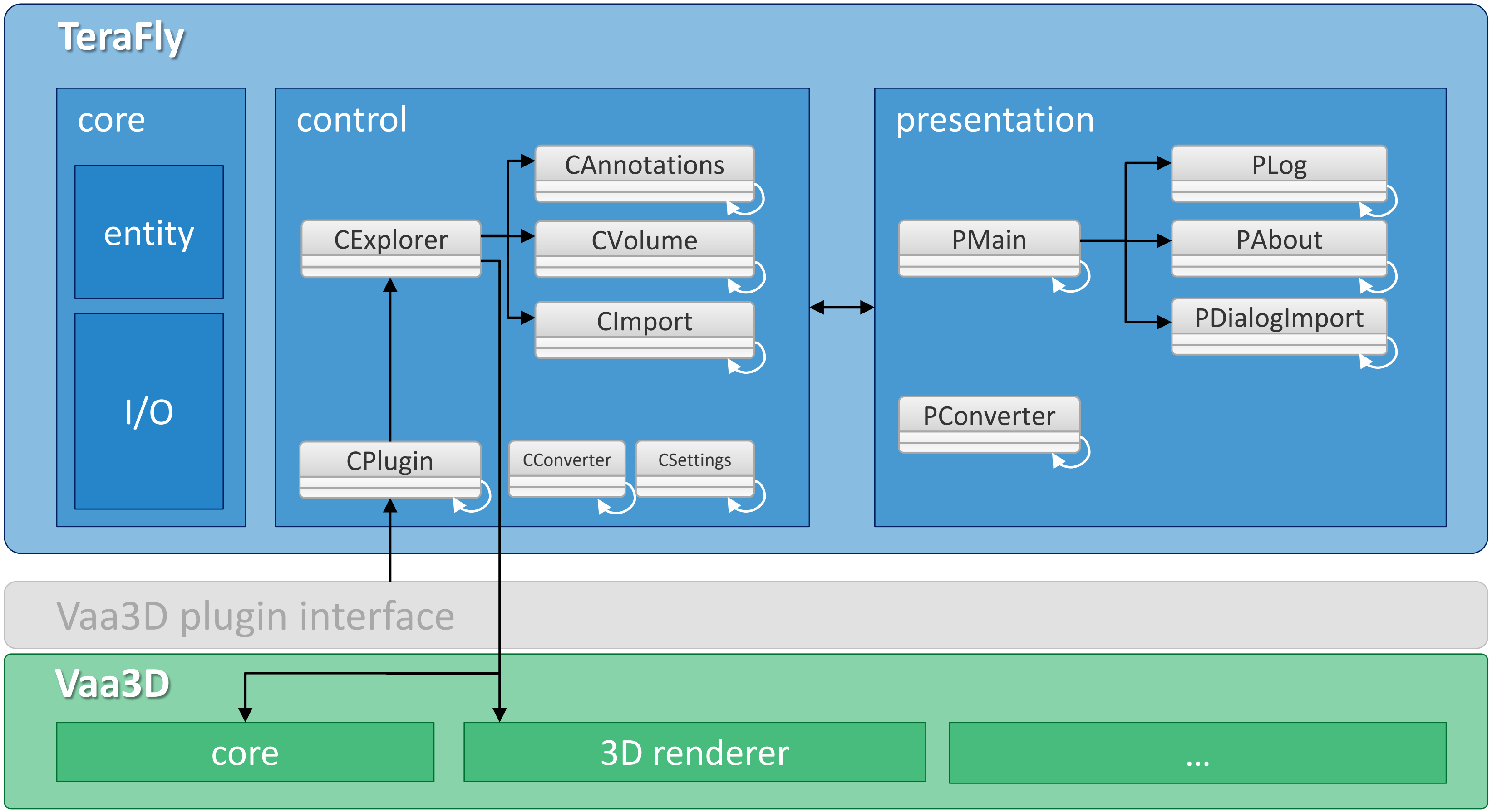


TeraFly: multiresolution representation

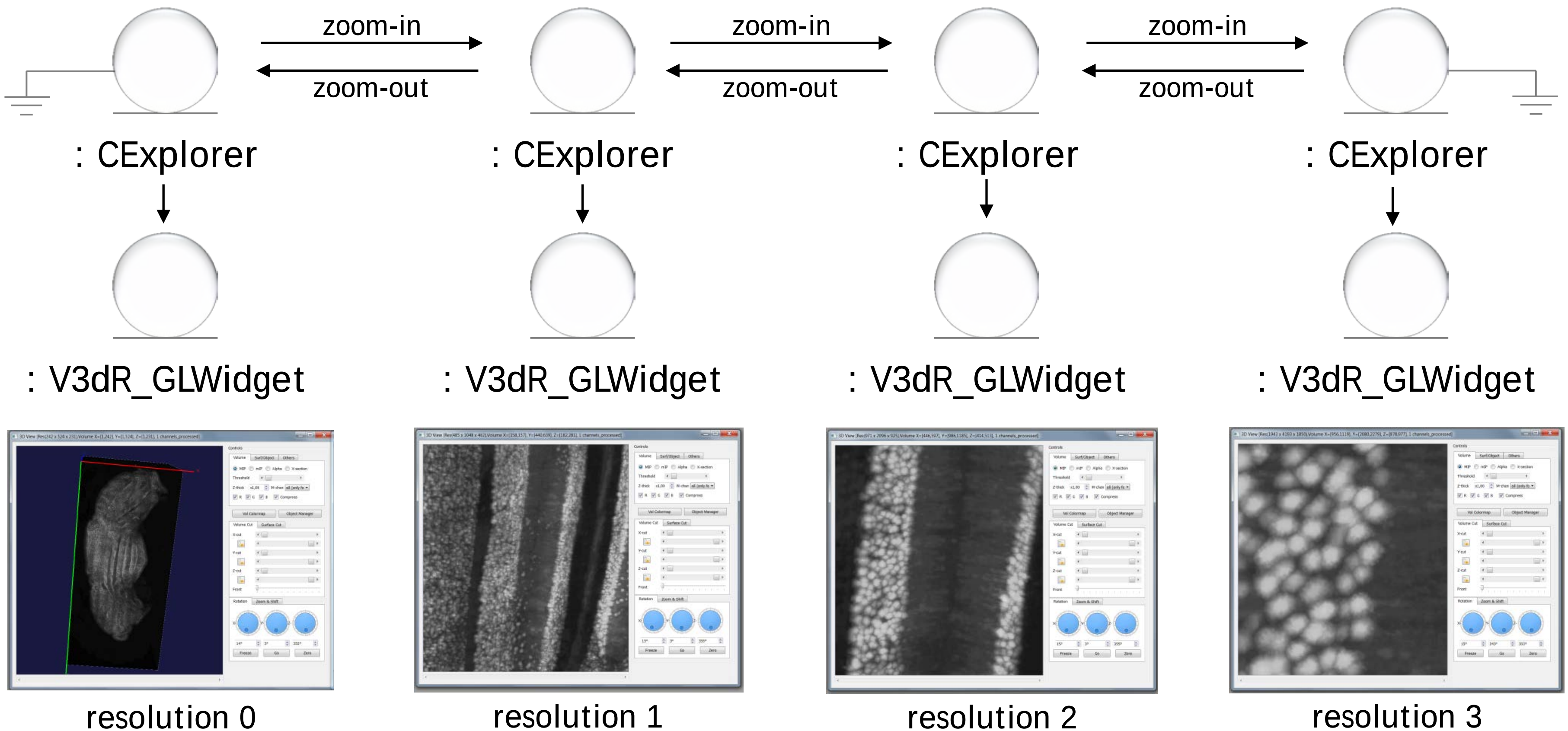


- the volume is saved in tiled format at different resolutions $i = 0, \dots, k$
  - *i*-th resolution is obtained by dividing *i-1*-th resolution by 2, that is equivalent to divide the original resolution by 2^i
  - k is chosen so as the *k*-th resolution has size <100 MegaVoxels, thus it can be easily handled by the Vaa3D renderer
  - e.g. for a ~1 TeraVoxels volume whose size is $10.000 \times 10.000 \times 10.000$
  - $k = 5$ and the 5-th resolution has size ~29 MegaVoxels
- tiles dimensions is typically in [256, 512]

TeraFly: architecture (1/2)



TeraFly: architecture (2/2)

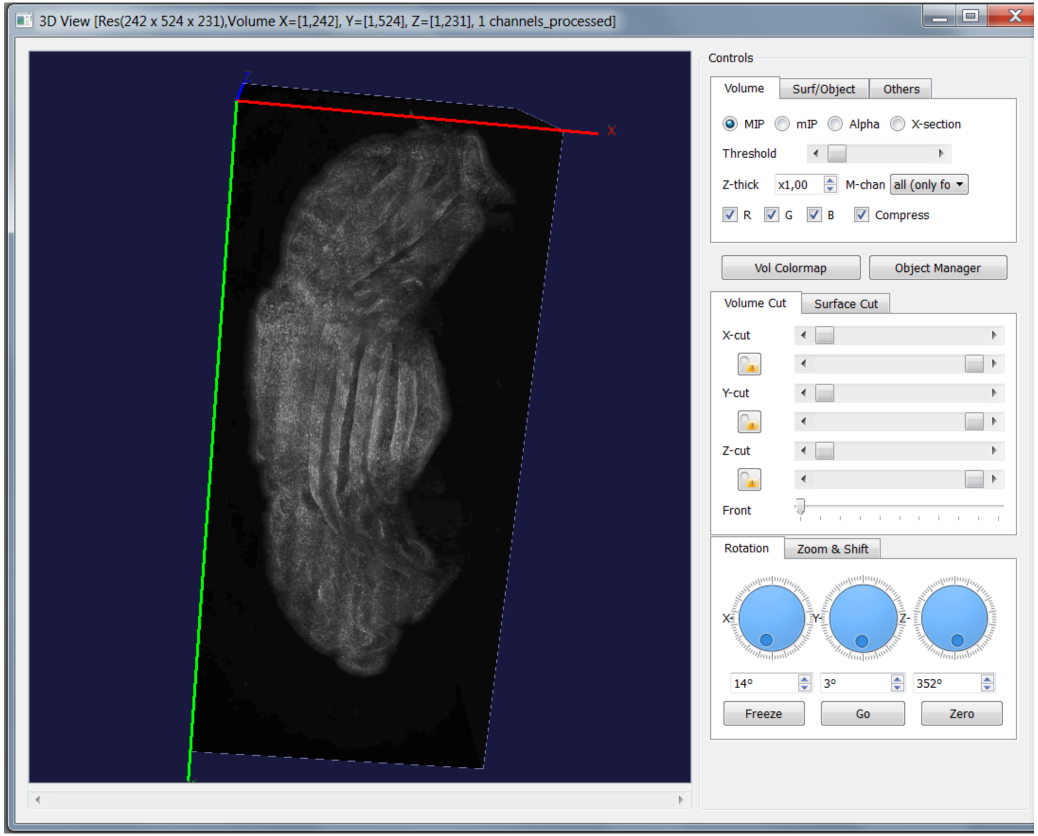


TeraFly: flying through the brain on a laptop

- On-air demostration

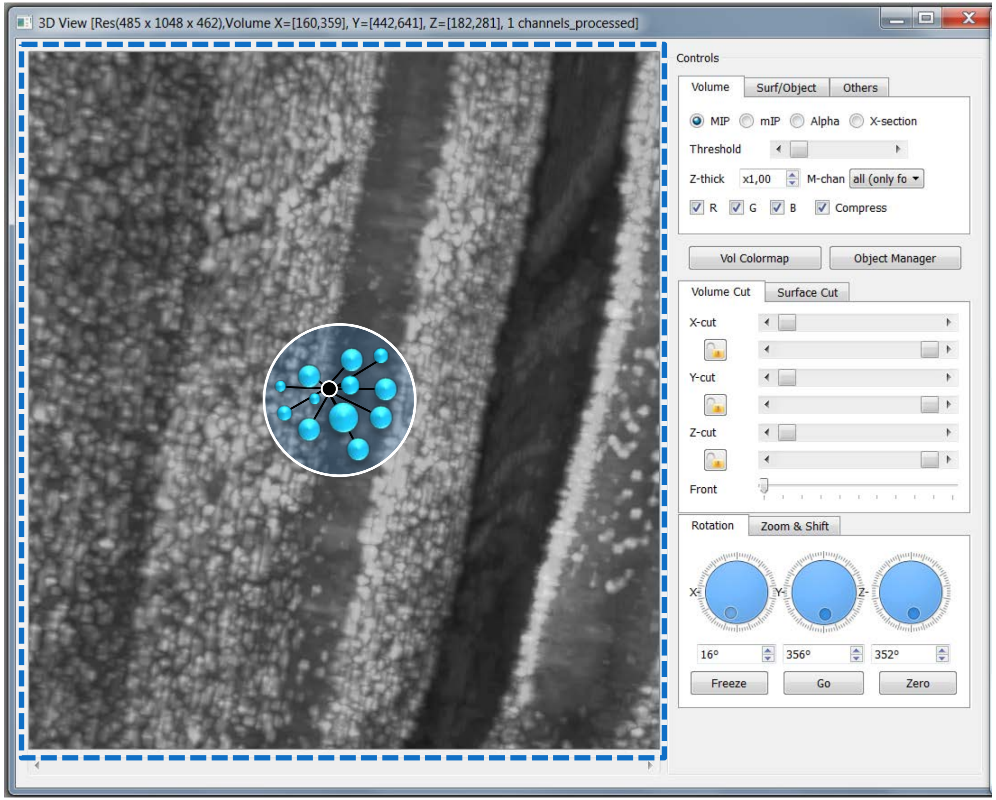
  - easy zoom-in/out with mouse scroll
  - fast zoom-in by interpolation + 1-step refinement
  - separate threads for GPU and I/O
  - basic caching

TeraFly: zoom-in method (1/3)



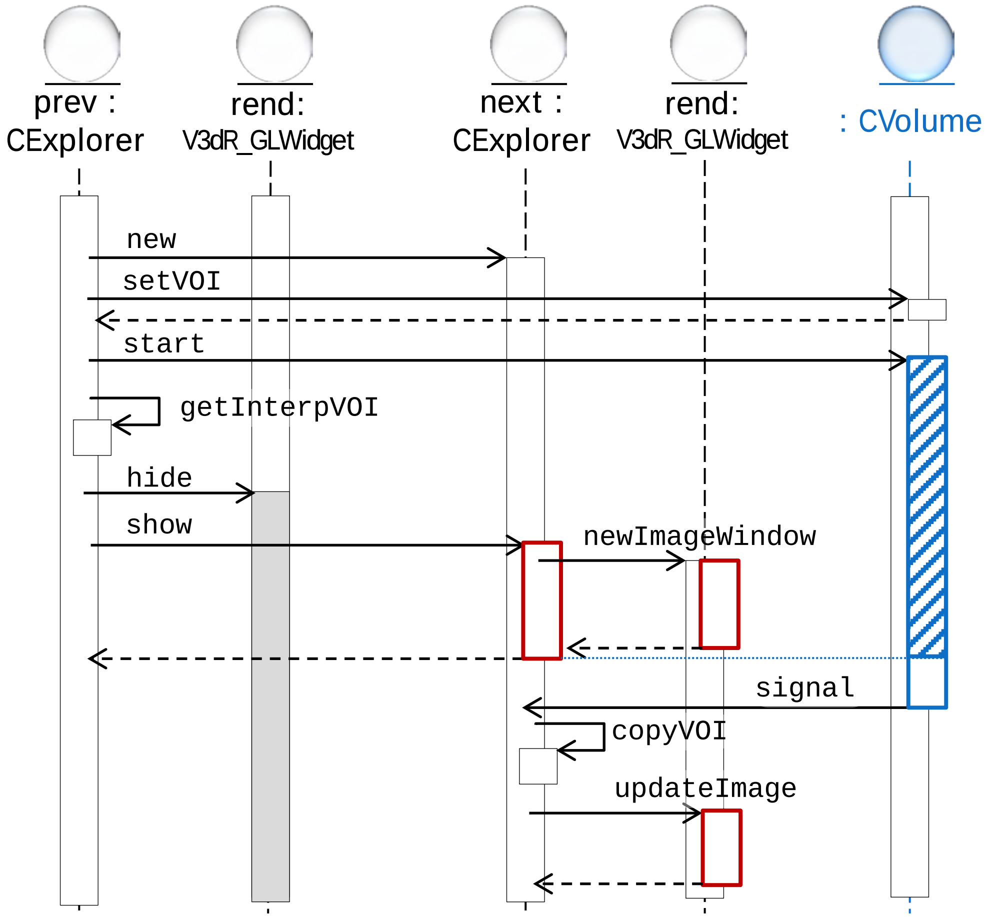
- 3D exploration starts with a pre-computed 3D image of the whole volume at low resolution
  - the first time a multiresolution volume is imported into TeraFly, the resolution that is best suited for the computer hardware capabilities is chosen and saved in a fast-to-load format (`vmap.bin`)
  - the low-res volume map so obtained will be used for starting the 3D exploration every time the user will open the volume
  - usually has size < 100 MegaVoxels

TeraFly: zoom-in method (2/3)



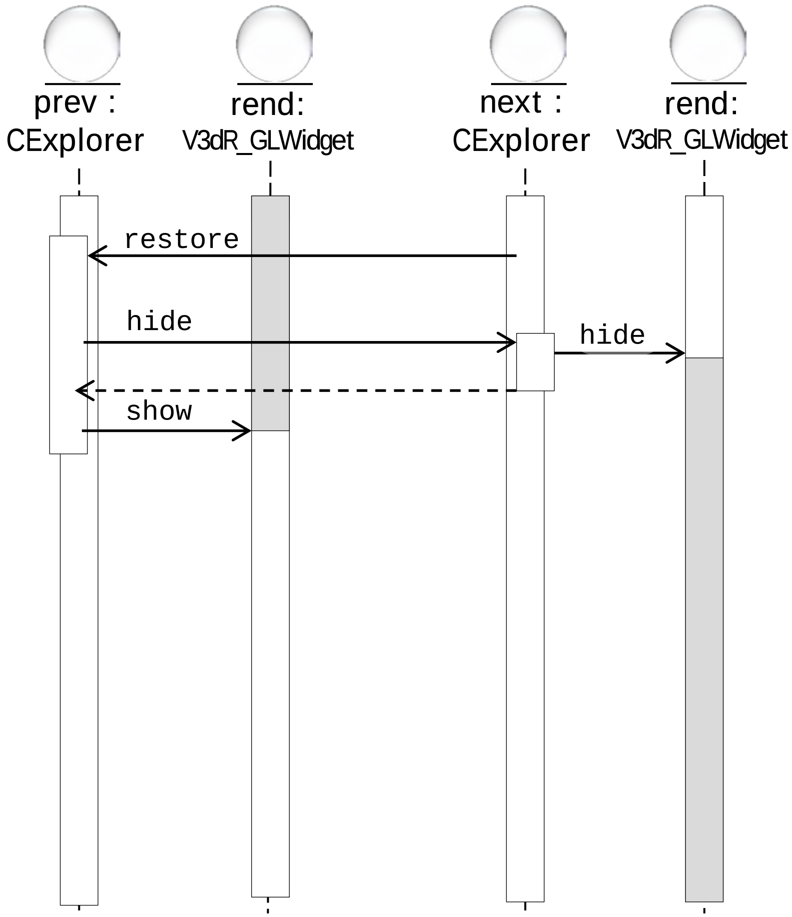
- zoom-in is triggered when mouse scroll exceeds a fixed threshold
- since the Vaa3D renderer zoom-in is center-based, we look at the center of the viewport
  - random 1-click pinpointing actions are triggered around the center of the viewport
  - the majority of markers is created on the foreground tissue/cells
  - we take the centroid as the center of the next higher-res view
  - the VOI is defined using the view size (can be set by the GUI)

TeraFly: zoom-in method (3/3)



- data I/O starts asap in a different thread
- meanwhile, VOI is extracted from the current view by interpolation and passed to the next view
- the interpolated VOI is shown in Vaa3D (GPU thread)
- meanwhile, data I/O ends and a signal is emitted
- the GPU thread catches the signal and triggers an update in the current view with the high res data just loaded

TeraFly: zoom-out method and basic caching



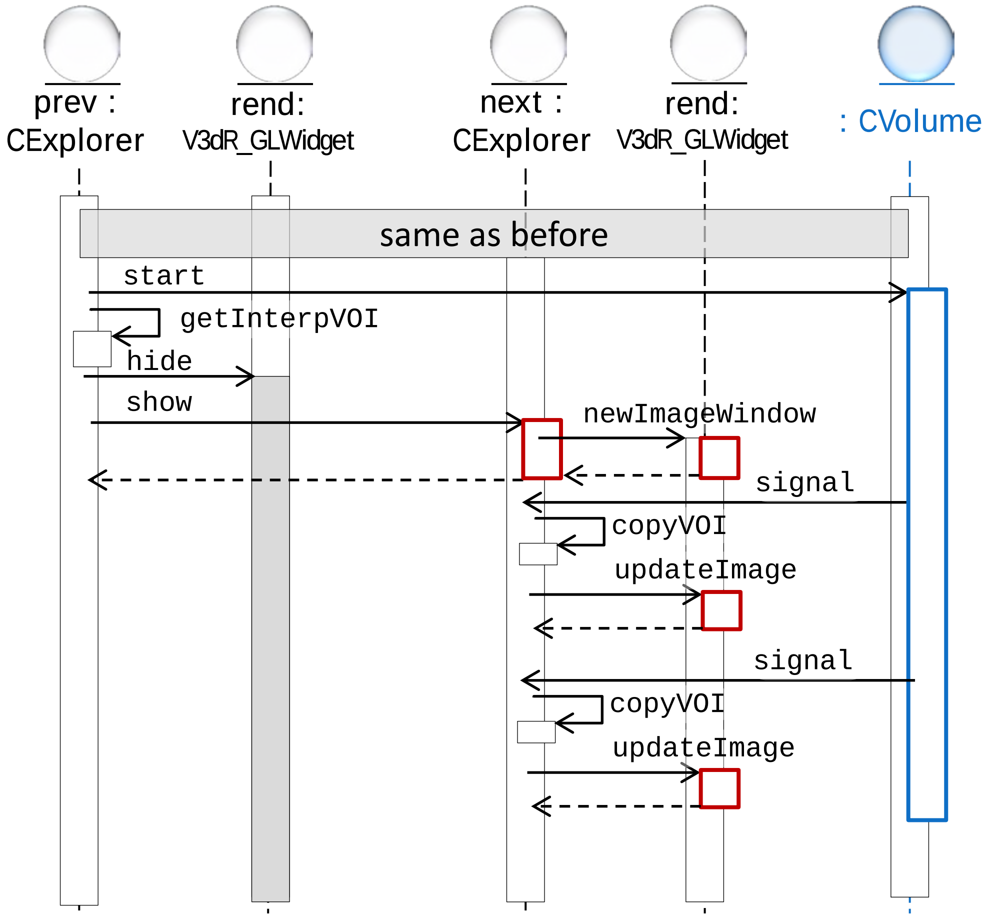
- zoom-out is triggered when the mouse scroll down exceeds a fixed threshold (can be tuned in the GUI)
- the current view is hidden and the previous view is restored
- the higher res view just hidden is maintained in memory for basic caching
  - when zooming-in again, the cached view is simply restored if the overlap between its VOI and the requested VOI is above a certain percentage (can be tuned in the GUI)

TeraFly: image slicing (1/2)

- On-air demostration

  fast zoom-in by interpolation + n-step refinement (by *image slicing*)

TeraFly: image slicing (2/2)



- depending on hardware speed, it might be convenient to use *image slicing* so as to load the first chunk of high res data and display it asap
- convenient when image updates are very fast and I/O is quite slow (*tradeoff*)
- the optimal number of steps should be automatically detected given the hardware specs

TeraFly: annotating 3D objects **(1/4)**

- On-air demostration

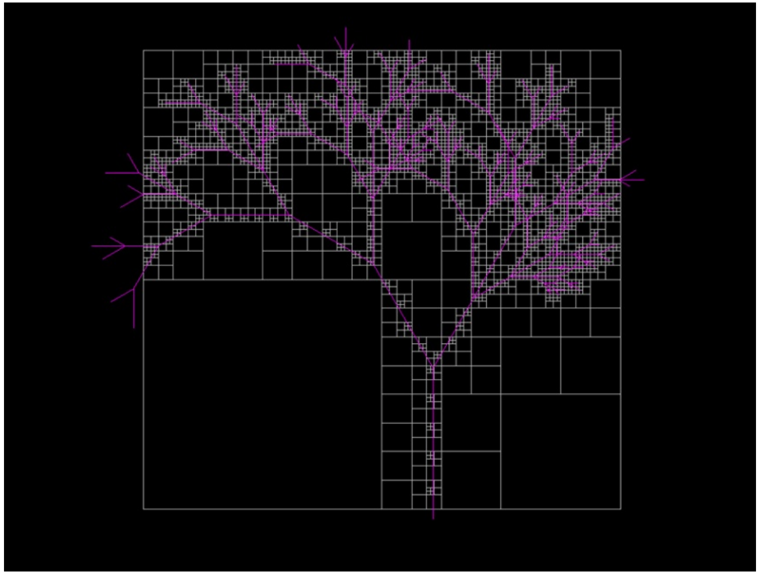
automatic scaling of 3D markers and 3D curves throughout 3D navigation

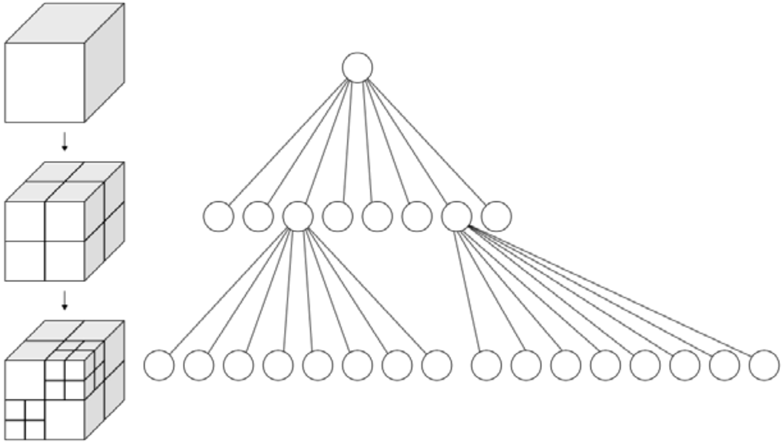
annotation of 3D objects

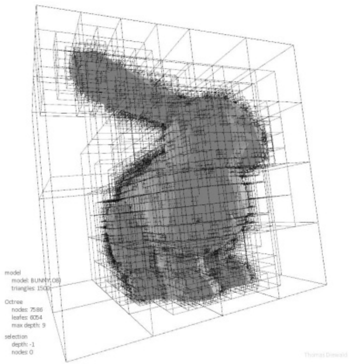
separate translations along X, Y, Z

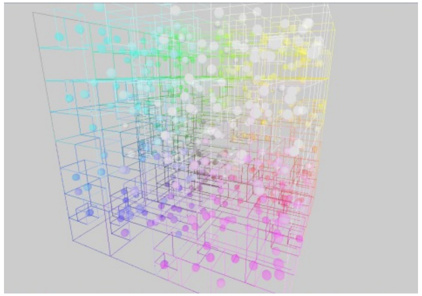
TeraFly: annotating 3D objects (2/4)

- Octrees

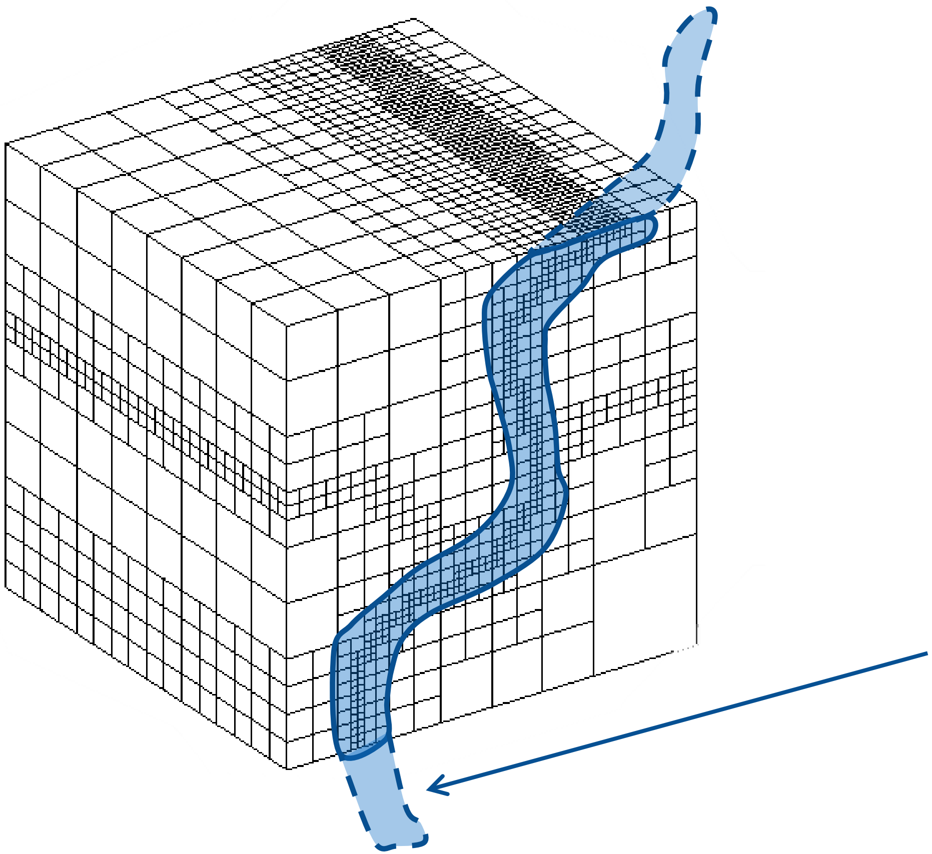






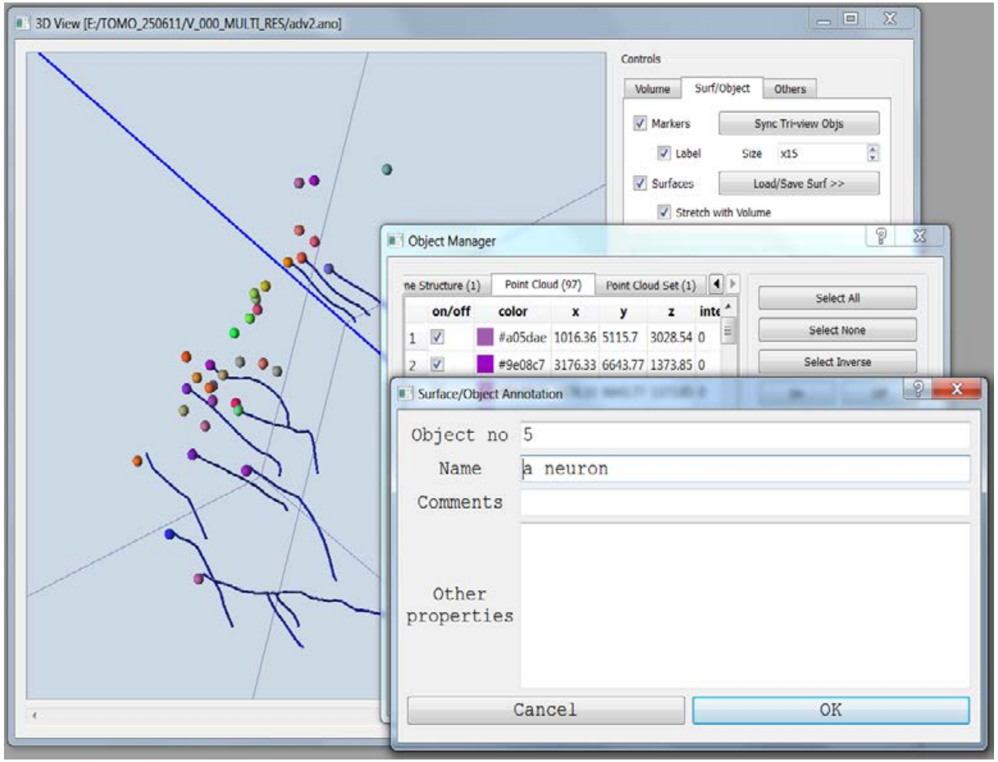


TeraFly: annotating 3D objects **(3/4)**



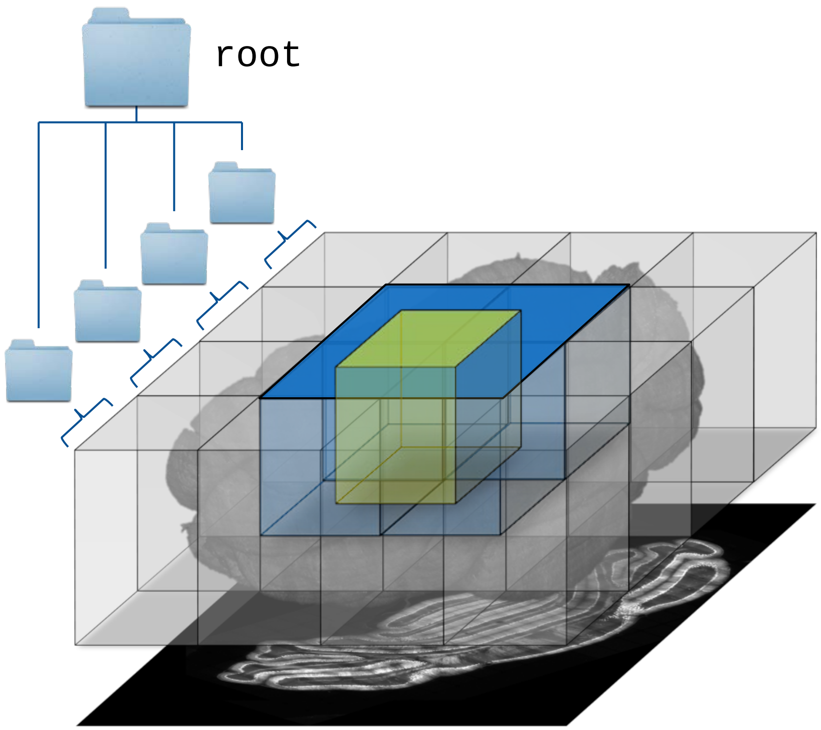
- we use a point region (PR) octree for storing 3D markers and 3D curves
  - the node stores an explicit 3-dimensional point, which is the "center" of the subdivision for that node
- compact representation
- **fast search**
- for 3D curves, additional linking between nodes is introduced for loading whole segments

TeraFly: annotating 3D objects (4/4)



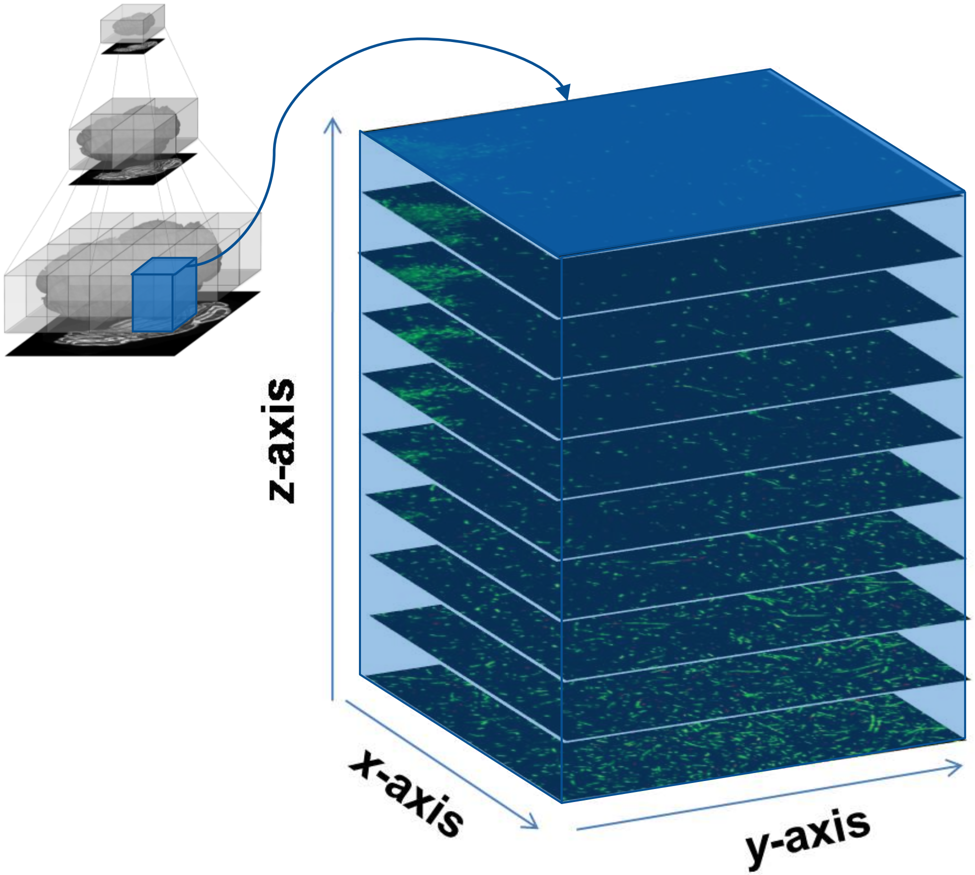
- markers are saved into 3D point cloud files (.apo)
- curves are saved into SWC files (.swc)
- a link file (.ano) is automatically generated for grouping heterogeneous 3D objects and annotations
- .ano files can be simply drag-and-dropped into Vaa3D or loaded by TeraFly on the image

TeraFly: I/O and the importance of the file format **(1/3)**



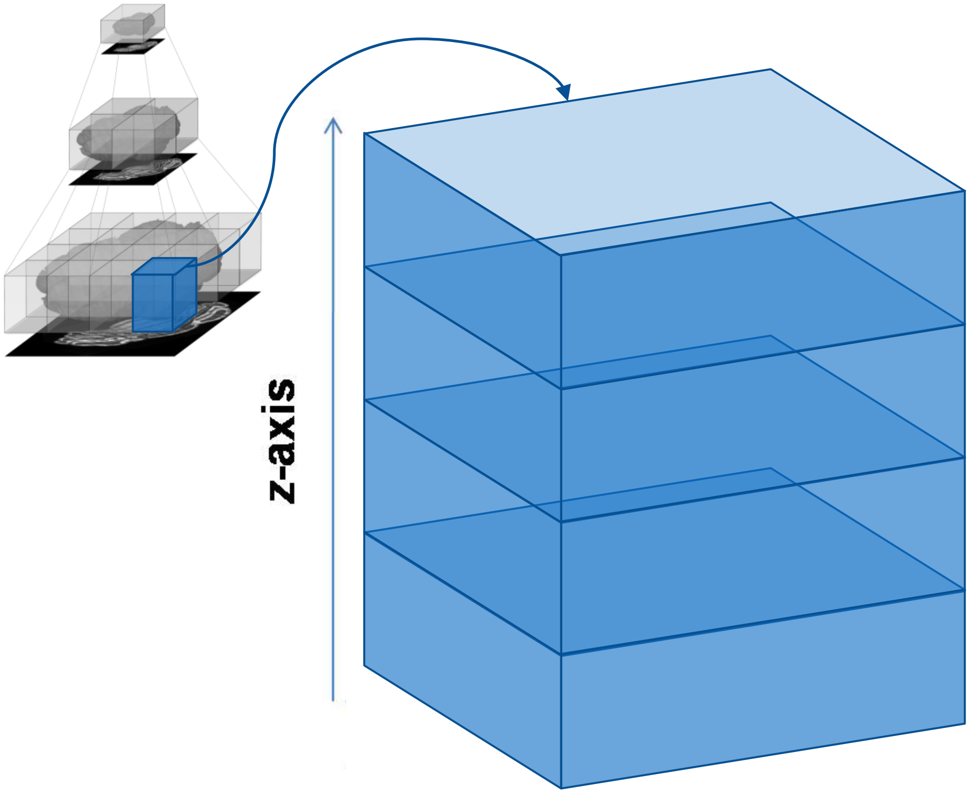
- tiled format pros:
  - small files
  - can copy a subvolume (*volume slicing*) by simply copying directories
  - when a VOI is requested, only the tiles intersecting the VOI are involved

TeraFly: I/O and the importance of the file format **(2/3)**



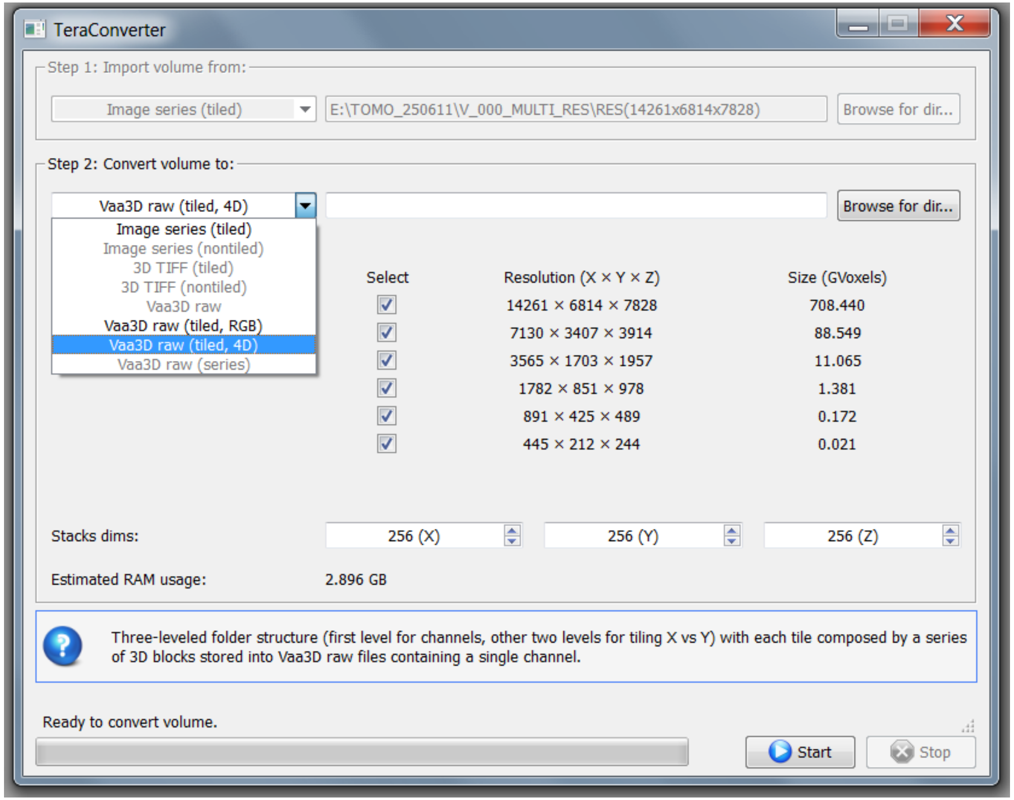
- in the beginning, TeraFly could handle only the *"Image series (tiled)"* format
- for very large volumes along X and Y, this led to a huge number of slice files
  - 10.000 slices × 50 × 50 stacks = 25 millions of files!
  - almost impossible to move
  - slow to access data: need to open thousands of files

TeraFly: I/O and the importance of the file format **(3/3)**



- TeraFly now uses the "*Vaa3D raw (tiled, 4D)*" format
- blocks along Z instead of slices!
  - 10.000 slices × 50 × 50 stacks = 50.000 files with blocks containing each 500 slices
  - each block is a 3D single-channel Vaa3D raw (random access, "*almost 10 times faster than TIFF*")
  - one volume per channel (i.e. channels is the 4th dimension)

TeraConverter



- for the conversion of terascale volumes from one format to another
  - Image series (tiled / nontiled): **any** image format is supported (tiff, png, jpeg, bmp, etc.)
  - 3D tiff
  - Vaa3D raw (single file / tiled with blocks / series), RGB or 4D
- RAM usage estimation
- selection of the resolutions to be produced

Summary

- Vaa3D is a free, open-source, cross-platform, extendible and versatile tool for visualizing and analyzing 3-5D bioimages on workstations and even on laptops
- the existing standalone tools, both free and commercial, still cannot deal with terascale images and/or do not embed such a powerful and user-friendly 3D-visualization-assisted analysis of bioimages
- TeraFly enables Vaa3D to handle terascale 4D images, thus making it possible to *fly through* terabytes of images almost instantly and even on laptops
- thanks to TeraConverter, TeraFly is also independent from the file format


THANKS!