VAST *Lite*
Volume Annotation and Segmentation Tool
User Manual, VAST *Lite* 1.1 RC1

Daniel R. Berger

May 22nd, 2017

Contents

Chapter 1

Introduction

VAST is a utility program for manual annotation and segmentation of large volumetric image (voxel) data sets. It enables users to work with data sets in the Terabyte or even Petabyte range at interactive speeds, to explore them visually and to label structures of interest by voxel painting.

Voxel painting has a number of advantages over other manual segmentation approaches like bread crumbing (placing a number of labeled points inside each object) and skeletonization (placing labeled points and connecting them with edges to form a 'skeleton'). It reveals the true shape of objects and makes visualizations of the data more comprehensive. It also allows measurement of volume and surface shape properties of the labeled objects. When working on a dense segmentation, the fact that voxel painting labels areas rather than single points or lines in the cross-sections of objects makes it a lot easier to spot objects that have not been labeled yet (visual pop-out). It also allows for some additional functionality, for example reliably determining neighbor regions, using voxel overlap of several segmentations for merging regions or determining synaptic connectivity, and conditional painting. Such functions are also difficult to implement when using outline labeling (drawing vectorized lines around process cross-sections in 2D slices). In addition, outline labeling can introduce problems in identifying corresponding outlines in different slices and can cause region overlap. Also, many machine learning algorithms for automatic segmentation rely on voxelized labelings for training, and the output of many such algorithms are labeled voxelized regions. VAST can be used to generate volumetric training data sets in their native format, and can to some extent also be used for importing, proof-reading and correcting results of segmentation algorithms.

The disadvantages of voxel painting which are mentioned most often are that it is slower than for example skeletonization or bread crumbing, and that it needs more storage space than alternative methods, especially when working with very large data sets. VAST tries to alleviate both disadvantages. For voxel painting, the time needed to label an object is largely determined by the number of outlines that have to be drawn. VAST can do automatic convex Z-filling to reduce the number of outlines by a factor of up to 8 if accurate boundary tracing is not needed. Painting is also optimized to be highly responsive, and the user interface provides quick access to functions which are used often (navigating through the image stack, changing the tooltip size, color picking, and switching between paint and delete mode). To reduce the amount of required memory and to enable interactive painting speed with any tooltip size, VAST implements multi-scale painting.

Considering that alternative labeling methods are likely to be more error-prone and will probably require more time until an acceptably low error rate is reached, voxel painting might be the overall faster alternative for fully manual labeling. If an object skeleton is needed (for example to compute length of a dendrite or number of spines etc.), it can be computed from the voxelized segmentation, whereas the inverse operation (computing accurate volumes from skeletons) is much harder. In a preliminary test we found that using VAST, a very experienced user can produce a dense segmentation of well-stained and well-aligned cortex neuropil data (6x6x30 nm voxel size) at a speed of about 4

cubic microns per hour. If cross-sections are labeled with a color dot in the middle rather than accurately outlining and filling the cross-sections, the labeling speed can be increased by a factor of 3 at the expense of accuracy (12 cubic microns per hour).

VAST is also very portable and light-weight. It consists of a single Windows executable file which is independent of third-party libraries. It does not require to be installed and can be easily copied and for example run from a memory stick.

Many tools for labeling voxel data exist, but such tools usually have drawbacks when it comes to painting in large data sets. Most tools do not support voxel painting but instead do bread crumbing, skeletonizing or storing object outlines as vector graphics (for example splines). Also most of them require the data to be loaded in RAM completely. This is not possible for the data sets in the Terabyte (and soon Petabyte) range which are currently produced by serial-section electron microscopy of biological samples. Many of these tools are also developed as cross-platform applications, so that they can be run on Windows, Mac and Linux systems. This usually means that a non-native GUI system has to be used (e.g. Qt) which makes the program a lot more bulky and more difficult to install and maintain. VAST is a Windows-only program and uses native Windows GUI and graphics functions.

VAST is currently being used in several scientific projects to label cells in electron-microscopic and light microscopic image stacks. It was also used for the recently published studies [3], [4], [5] and [6].

The key concepts of VAST are:

- Image stacks are imported into VASTs .vsv file format, where they are stored as dices. This, together with pre-computation of mip maps, allows for fast panning and zooming through the data when it is opened in VAST.
- .vsv files support lossless and lossy compression to reduce the resulting file size
- Image data and segmentations are stored in single files, which makes it easy to copy them from one place to another
- Support for loading EM stacks from a web server over HTTP (openconnecto.me, neurodata.io and butterfly formats)
- Dynamic multi-threaded cacheing in RAM with pre-loading for low-latency display update
- Layering: Several image and segmentation stacks can be opened and displayed together with a number of blending and tinting options
- Multi-scale painting (in VAST, painting always happens at the currently displayed resolution (mip level))
- Automatic convex Z-filling during painting to speed up coarse labeling
- Automatic 2D-filling of closed contours
- Label color patterns create a larger number of distinguishable label colors
- Label hierarchies allow for fast and reversible grouping of labeled segments in a tree structure
- Segments have anchor points to quickly find them in the volume
- Exporting of segmentations, EM stacks and mixed image stacks ('screenshots') in multiple formats
- Importing of segmentations from image stacks

- File-modification free editing. Image files are not changed (except if you explicitly update information in them). Segmentation files are only changed it you save the segmentation back to the same file.
- TCP/IP-based API through which external programs can directly communicate with VAST, and supporting Matlab script 'VastTools' which provides supplementary functions like measuring labeled segments and exporting of 3D surface models.

Current limitations:

- Currently only 16-bit segmentations are supported
- There is no 'Undo' function
- Exporting of 3D models of segmented objects is currently only supported externally (using the VastTools Matlab script)
- No 3D model display
- No multi-user support
- Not an open-source project, but the VAST *Lite* executable can be copied and used freely (see section A.6 for details)

Please note that VAST is still under development, and is subject to change as new features are added and bugs are fixed. For bug reports or helpful suggestions, please contact me at: `danielberger@fas.harvard.edu`.

Chapter 2

Getting Started

2.1 System Requirements

VAST currently runs on 32- and 64-bit Windows computers that support DirectX 11. These are Windows Vista, 7, 8 and 10, with DirectX 11 or later available. Windows XP and older versions will not work. 64 bit versions of these operating systems are recommended, because 32 bit programs are limited in the amount of RAM they can handle (4 GB max. theoretically, less realistically). The computer also has to have a DirectX 11 compatible graphics card. In most modern computers even the on-board graphics chips support DirectX 11.

Recommended system configuration:

- Windows PC with 64 bit Windows 7 or later
- 16 GB of RAM (the more the better)
- DirectX 11 compatible graphics card
- 2 TB of disk space (depends on the size of the data you work with)
- Wacom Cintiq 13HD or other pen touch screen with configurable two-button pen

Minimal system configuration:

- Windows PC with 32 bit Windows 7
- 2 GB of RAM (at least 4 GB recommended)
- DirectX 11 compatible on-board graphics card
- Standard screen and mouse

2.2 Program Setup

To use VAST, simply copy the executable program into a folder where you have read/write access, and set up links on your desktop, start menu and/or taskbar if desired. It is important that VAST has read and write access to the folder where the executable is because it will write a configuration file (`vast_preferences.dat`) into the same folder to store your settings.

Start the executable.

2.2.1 Try It Out!

The quickest way to try out VAST is to use an online data set. Several online data sets are included in the .ZIP package of supplementary files as .VSVR files. You can save this package from the executable by clicking 'Yes' in the 'First Start' pop-up window when you start VAST for the first time, or by choosing 'Save Documentation .ZIP To Disk...' from the 'Info' menu.

Unzip the package (or drag&drop the contents to a folder outside the .ZIP using Windows Explorer for example). Then, in VAST, go to 'Open EM...' and select one of the .VSVR files in the `VAST_package/Online Datasets/` folder, for example '`neurodata_kasthuri11.vsvr`'. This will load images of a big EM stack from the Johns Hopkins `neurodata.io` server. Your computer has to have internet access for this to work.[1]

Click and drag the EM slice to pan. Use the mouse wheel to zoom. Use UP/DOWN arrow keys or A/Z to scroll through the stack.

Click on the little pencil icon in the toolbar to switch to 'Paint' mode. Choose 'Yes' in the popup window. Click and drag over the image to paint. You can select different paint colors in the 'Segment Colors' tool window. To erase, hold down the 'Delete' key while painting (or click and hold the right and left mouse buttons together). Select 'Keyboard Shortcuts' from the 'Window' menu for a list of available keyboard functions.

2.2.2 Preferences

In the main menu of VAST, go to '`File / Preferences ...`' to open the Preferences dialog window. Here you can set the parameters for data cacheing and display. VAST will set up the preferences for you when you run it for the first time on a computer (whenever it cannot find the preferences file). You can edit these preferences if you want to. You should at least check once whether the folder in which VAST puts its temporary disk cache is on a hard drive with lots of free space. Depending on what you do, temporary disk cache files can get similarly large as the segmentation files you are working with, in particular if you use global segmentation editing functions like merging, segment deleting, or segment welding. Each VAST instance will write a cache file (`vastsegcache*.vss`) to this folder and delete it when VAST is closed. In case VAST crashed (which should not happen) or power was lost while VAST was running, a cache file may be left there which takes up disk space, so you may want to check for and delete old cache files once in a while if VAST was terminated anormally.

Memory and Cacheing

On the left side of the Preferences dialog you can set how much cache memory VAST will use maximally for voxel images and for segmentations. VAST chooses initial values which are reasonable for your system. The rule of thumb is: If you can afford it, leave 1-2 GB for the system, and split the rest 1/3 each for image cache, segmentation cache, and general usage of VAST (don't assign). On a system with 8 GB RAM, this means to give 2000 MB to the image RAM cache and 2000 MB to the segmentation RAM cache. If you are only viewing images and not using segmentations, you can increase the size of the image cache and reduce the size of the segmentation cache accordingly. If you plan to use other programs at the same time or run two instances of VAST, please reduce these values as needed. VAST will not immediately use all of the allotted memory, but it will stop reserving new memory for cache blocks and re-use old blocks when it reaches the limit.

In general, do not allow VAST to allocate more memory than the system has. This can result in severe performance issues. There is a memory usage indicator in the upper right corner of the VAST window which shows you how much memory is currently used. The blue frame indicates the maximum

[1] Remote EM images are loaded from a server using a cutout service over HTTP. The VSVR file is just a text file which defines the web address of the dataset to load and its dimensions.

amount of RAM which VAST uses for image and segmentation cacheing. If the memory indicator becomes red and your system slows down, try to REDUCE the cache limits to allow Windows and VAST to use more RAM for other data.

Some of the segmentation cache is used for holding the currently displayed part of the segmentation in memory. When you exit the preferences dialog, VAST will tell you how much of the segmentation cache it needs for the current display settings and whether the cache size is sufficient.

'Disk Cache Directory': Here you can specify the folder where VAST stores its temporary disk cache. Click the '[...]' button to browse. Set this to a folder where you have lots of free space (more than the size of the largest segmentation file you will be working with, since for certain functions VAST has to duplicate the segmentation data).

Painting

The segmentation bit depth is currently fixed at 16 bits. This allows for a maximum of 65535 labels; however, since each segment is represented in the tree view of the 'Segment Colors' window in VAST, memory limitations in the Windows system might prevent VAST from using that many labels.[2]

'Tablet Mode (Pen Paints, Finger Moves)': On some pen-enabled tablet computers VAST can distinguish finger and pen input. If this mode is enabled, the pen should paint and the finger should move the view when in Paint Mode. This may not work on some newer systems.

'Generate New Colors From:': Allows you to select which color space to use for assigning random colors to new segments. Segment colors can of course also be changed manually (see section 3.4.11).

Display Properties

'Maximum Window Width' specifies the width (or height, whichever is greater) of the largest window you will be using, in pixels. This value is used to determine how many textured tiles are needed to fill the entire window at all zoom levels. Setting this value smaller reduces memory consumption and increases cacheing speed, but if the value is too small, the image texture might not reach all the way to the sides of the window at all zoom levels.

'Target Resolution Smaller Than' lets you specify the effective resolution of the displayed textures on the screen in screen pixels per texture pixel. This affects at what zoom levels which mipmaps are used. '2' is a good general setting; '1' makes it more detailed but slower (and more memory-consuming), and '4' makes it faster but blurry.

'Texture size m (texture is m^2):' defines how large the texture tiles will be which are used for displaying image and segmentation textures. Depending on the graphics card some texture sizes might be faster than others. I recommend to leave this setting at 128.

'Texture Smoothing': You can set here whether you want to use texture interpolation. This reduces aliasing effects but can result in a slightly blurred appearance of the textures. The most natural setting of this is, in my opinion, 'All except Mip 0', which will show pixels with sharp boundaries only if you zoom in more than the native resolution of the image data.

'Link Tool Windows': When enabled, VAST's floating tool windows will automatically move to their default locations with respect of the main window when the main window is moved or rescaled.

The remaining options are self-explanatory. Opacity values have to be set between 0 (fully transparent) and 255 (fully opaque).

Other

'Enable API Remote Connections at Startup': This can be useful to enable if you often use the VAST API to link to external programs. The API remote connection functionality can also be

[2] It runs fine with more than 36000 labels in one of our data sets.

enabled and disabled in the 'Remote Control API Server' dialog under 'Window' in the VAST main menu.

Press OK after you're done configuring the preferences.

2.3 Additional Files Included With VAST

Under 'Info / Save Documentation .ZIP To Disk ...' in the main menu you can save out additional files which are packaged into the VAST executable, as a ZIP file. Select a target location and save, then unzip the ZIP file.

Currently this includes a set of .vsvr files to access some large EM data sets remotely (see section 3.2.1), some Matlab scripts which can be useful for analyzing VAST data in Matlab, including the VastTools toolbox which can communicate directly with VAST through the API and provides additional functionality (see chapter 4), and this documentation as a PDF file.

Chapter 3

Working with VAST

VAST uses its own file format `.VSV`[1] to store image data. It can open `.VSV` files immediately and navigate in them quickly. If your data is a stack of for example `.PNG` images, you will have to import it into VAST before you can use it. During importing the data will be saved into a VAST-specific `.VSV` data file, which allows quick access to arbitrary parts of the data. After opening a `.VSV` image file, you can create a segmentation by painting on top of the images, or you can open an associated segmentation file (`.VSS`)[2] and view image and segmentation together. `VSS` files tend to get big quickly, but can be packed efficiently, for example in a `ZIP` file. You can view segmentations, modify and save them. You can export segmentations as image stacks for using them in other analysis programs or to render the segmented objects in a 3D animation program like 3D Studio MAX. You can also import segmentation image stacks that were generated externally.

3.1 Image Stack Importing

Typically volumetric image data is stored as a series of 2D images, or as a serial 3D block of data, which is not suitable for fast interactive viewing. When you import such a data set into VAST, it puts the images into a single file containing a diced data structure, and computes and includes mipmaps for the images.[3] Using diced data does not only speed up loading of parts of images, but will in the future also enable fast loading of volumetric sub-regions or 2D sections at other orientations through the image data.

VAST does currently not include image alignment and stitching functions. If you are starting with an unaligned stack of images, you will first have to align the images with a different program (Fiji or Photoshop, for example) and then save a stack of aligned images which all have the same dimensions and are named and numbered in a consistent way (for example `img000.png, img001.png, ...`). Put all images into the same folder.

VAST can import single-tile image stacks, multi-tile image stacks, and 3D volume files. In a single-tile image stack, each slice of the stack consists of a single image file. In a multi-tile image stack, each slice is composed of several tiles in a XY grid, and each tile is stored in a separate image file. A 3D volume file stores all slice images in a single file. Currently the only 3D volume file format that VAST supports is NIfTI (.nii). VAST will convert image data to either 8-bit graylevel or 24-bit RGB when importing.

[1] .VSVOL can be used instead, and may be preferable, because Microsoft thinks .VSV is a 'Microsoft Visio' file.

[2] You can use .VSSEG instead of .VSS as an alternative file name extension for VAST segmentation files.

[3] A *mipmap* is a downsampled version of an image. VAST uses power-of-two (2D, XY) mipmaps. For example, for an original image of 1024x1024 pixels, it will compute mipmaps of 512x512 and 256x256 pixels. It does this for every slice image in a stack.

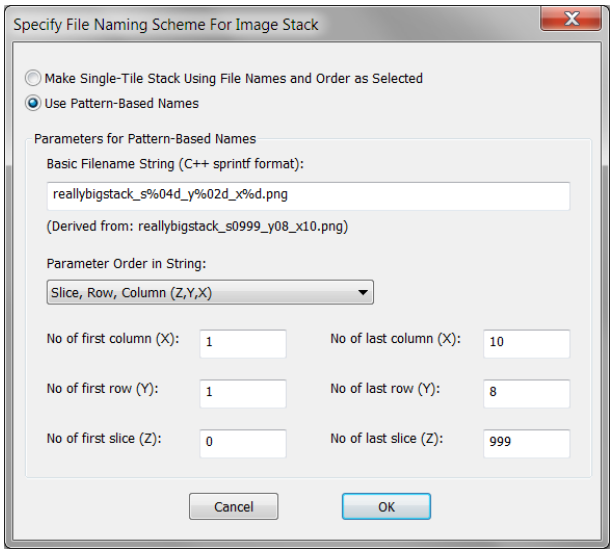

Figure 3.1: First dialog for importing EM image stacks: Specification of pattern-based names

For importing and dicing, VAST will use the RAM cache which is normally used for cacheing EM image data during viewing and painting. Having lots of cache memory available will make importing somewhat faster, because images have to be re-loaded less often. You can set the size of the EM image cache in the Preferences (see section 2.2.2).

3.1.1 Importing image stacks: Pattern-based names

In the main menu of VAST, go to '`File / Import EM Stack from Images ...`'. VAST will show a file browser dialog in which you can select one or several image files. For importing 3D NIfTI files, please select only one file. If you import a single-tile stack and do not want to use pattern-based names, select all slice images in the correct order, because images will be stacked in the same order in which they appear in the system's list of selected files. The order is usually correct if you select the last image first, then shift-click (hold the `SHIFT` key down and click left with the mouse) the first image to select the whole range. You can also try 'Select All' by pressing `CTRL-A` if the folder only contains the image files you want to import. If you are worried about the order of the images and want more precise control, you can use pattern-based names. If you make use of pattern-based names to import single- or multi-tile image stacks, it is sufficient to select one file, but even better to select the first and the last file in your set of images. Then click '`Open`'.

After selecting one or more image files (not `.nii`), VAST will display the dialog shown in Figure 3.1. To import without pattern-based names, select 'Make Single-Tile Stack Using File Names and Order as Selected' and press `OK`.

If you select the second option, '`Use Pattern-Based Names`', the parameters in the lower part of the dialog window will be enabled. With pattern-based names, you specify a template string for the file names which contains placeholders for numbers, and ranges for these numbers. With this you can also import image stacks in which each slice is stored in several image files (multi-tile image stacks). This is useful for data sets in which a single slice is so large that it can not be stored in a single image file, but is stored in a set of tiles which form a regular grid. Please note that these tiles should *not* be unstitched image tiles as they come off a microscope, but they have to fit seamlessly. If you have a set of raw microscopic images which are not yet stitched and aligned, please use an external program to generate a stitched and aligned image stack first, and store each slice as a single

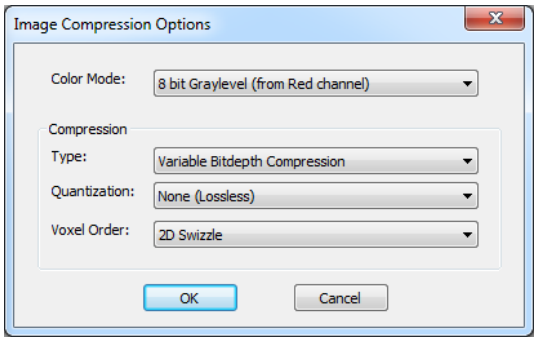

Figure 3.2: Second dialog for importing EM image stacks: Image compression options

image or a set of image tiles. You can then import those images into VAST.

VAST will use the file(s) you selected in the previous dialog to determine the source directory where the image files are and to generate a basic template for the file name. It assumes that all images of the stack are in the same folder (the one you picked an image from), and are named consistently with numbers for slices, rows and columns. It also assumes that the set of images is complete, which means that there's an image for every slice/row/column combination in the range you give. In this dialog, you specify these ranges as well as a schema to derive the filename for a given slice/row/column coordinate.

Let's say, for example, you have a data set called 'bigstack', which has 1000 slices, numbered from 0 to 999, and each slice has 10x8 tiles, numbered from 1 to 10 and 1 to 8. Assume the image in the upper left corner of the first slice is called 'bigstack_s0000_x01_y01.png', the image tile right of it is called 'bigstack_s0000_x02_y01.png', and so on. The last image in the lower right corner of the last slice would be called 'bigstack_s0999_x10_y8.png'.

First, make sure that the file name in the edit box at the top contains the correct *C++ format string* (as it is used by `printf()`). In general, numbers which specify the slice, column and row coordinates have to be replaced by codes like '%d' (integer number) or '%04d' (integer number with zero-padding to 4 digits). VAST will then fill in those numbers for each image. For more information about format strings, please refer to a C++ manual or ask the internet.

In the combo box below, select which coordinates are used in the file names, and in which order they appear. The edit boxes below let you specify the range of (integer) numbers for the three coordinates. After you entered all parameters, press `OK`.

3.1.2 Lossless and lossy compression

Next, VAST will show a dialog where you can specify the color mode and image compression options (Figure 3.2).

Under 'Color Mode', please select if you want to import the images as 8-bit graylevel or 24-bit color images, and for graylevel which source color channel to use. When importing from graylevel images, please select the first option ('from RED channel').

Under 'Compression' you can specify compression options. Under 'Type' you can select the compression method - Uncompressed, Variable Bitdepth Compression, zlib Compression, and Spectral Compression. The three different compression algorithms are by themselves lossless, but might produce slightly smaller or larger file size depending on your data. Variable Bitdepth Compression should be fastest when reading from the compressed files.

'Quantization' specifies whether the compression should be lossy or lossless. Lossyness is achieved by quantizing, meaning throwing away bits. For example, if you set Quantization to -2 bits, graylevel images will have only 6 bits resolution (64 different gray levels) rather than 8 bits (256 different gray

levels). Throwing bits away reduces both file size and image quality.

'Voxel Order' defines in which order the pixels in the images will be stored. This can have an effect on compressed file size. '2D Swizzle' stores pixels in 2D Z-order. I usually get best results using '2D Swizzle', but '3D Swizzle' might be superior for very well aligned data (e.g. FIB-SEM).

Next VAST will ask you to specify a target location and file name for the resulting `.vsv` image volume file. Use '`.vsv`' as extension for the file name. Choose a location where you have enough storage space for the file. The file will not only contain the original image data, but also the mip maps. For example, if you import 1024 images of 1024x1024 pixels each and store uncompressed, the `.vsv` file will be approximately $1024 \cdot 1024 \cdot 1024 + 512 \cdot 512 \cdot 1024 + 256 \cdot 256 \cdot 1024 = 1409286144$ Bytes (≈ 1.3 Gigabytes) large. Lossless compression will reduce the file size, and lossy compression even more, but by how much depends strongly on your data and the compression method used.

Then, the images will be read, diced, and put into the target file. After that, VAST will compute the mipmaps and put those in the target file too. Depending on the size of the data, this process can take several hours. For example, a big data set of 350 GB takes about 5 hours to import on a recent desktop machine. The limiting factor is the speed of hard drive access.

You can cancel the importing, but the target file will then be incomplete / corrupted and can not be used with VAST.

3.1.3 Importing 3D volume files

Importing a 3D volume file is easier than importing an image stack. The only format currently supported by VAST is Nifti. VAST will ask you to specify the name of the source (`.nii`) file and the name of the target `.vsv` file. Currently VAST requires the whole Nifti file to be loaded at once into RAM, so this only works for smaller volumes. Also, the data in the Nifti file currently has to be 8 bit per pixel.

3.1.4 Image scale and description

After importing, VAST will show a dialog where you can set the voxel size in nanometers of your data in the file.[4] Press 'Save to file' in the dialog to save the information you entered to the VSV file. The voxel size entered here is used for the scale bar which you can enable in the main menu under '`Info / View Scale Bar`', and for scaling models and measurements in VastTools. You can access and change these numbers for the selected image layer under '`Info / Volume properties ...`' in the main menu. This dialog also displays how large your image stack is in voxels.

Under '`Info / EM File Information ...`' in the main menu you can enter and view text which will be stored in your VSV file as well. This can contain a description of the data, copyright information, or other.

3.2 Viewing and Navigating an Image Stack

After you imported a stack of images, you can view them interactively. After you closed the program, you can re-open a previously diced data set by using '`File / Open EM Stack ...`' from the main menu. VSV files you open will be added to a list under '`File / Open Recent EM Stack`', from where you can quickly access them again. The list contains the 16 most recent VSV files.

You can also open VSV (and VSS) files by drag-and-drop from a file browser (Windows Explorer) onto the VAST window.

VAST currently has a 'Move mode', a 'Paint mode', a 'Collect Mode' and an 'Eyedropper Mode', which you can set by clicking the tool buttons in the toolbar. The cross of arrows icon selects 'Move

[4] You can access the same dialog at any time under 'Info / Volume properties ...' in the main menu.

mode' and the little pencil selects 'Paint mode'. In this section we will explain how to use the 'Move mode'. For an explanation of the other modes please refer to section 3.3.

The easiest way to navigate in the image stack is by using the mouse in 'Move mode'. You can pan (move the image sideways) by left clicking and dragging it. You can use the mouse wheel to zoom in and out. Alternatively, you can zoom using the N and M buttons, or the sidebar (see below). Use the UP and DOWN arrow keys or A and Z to scroll through the slices of the stack, or the sidebar to scroll more quickly.

3.2.1 Remote image stacks

In addition to using an image stack in VAST which has been imported into a local `.VSV` file, you can also open and access image stacks which are hosted online. VAST supports the *'Open Connectome Project Cutout Service'* from `http://www.openconnecto.me`, the newer `neurodata.io` data servers and Harvard's *Butterfly* cutout service.[5] To access a remote image stack you need a `.VSVR` file which specifies the parameters of the data set. `.VSVR` files are text files in a JSON-like format; here is the content of the `openconnectome_kasthuri11.vsvr` file:

```
{
  "Comment": "Source: http://openconnecto.me/ocp/ca/kasthuri11/info/",
  "ServerType": "openconnectome",
  "ServerName": "openconnecto.me",
  "ServerFolder": "/ocp/ca/kasthuri11",
  "SourceDataSizeX": 21504,
  "SourceDataSizeY": 26624,
  "SourceDataSizeZ": 1850,
  "TargetDataSizeX": 10747,
  "TargetDataSizeY": 12895,
  "TargetDataSizeZ": 1850,
  "OffsetX": 0,
  "OffsetY": 0,
  "OffsetZ": 0,
  "OffsetMip": 1,
  "TargetVoxelSizeXnm": 6,
  "TargetVoxelSizeYnm": 6,
  "TargetVoxelSizeZnm": 30,
  "TargetLayerName": "Kasthuri11@OpenConnectome"
}
```

VAST comes with several pre-defined `.vsvr` files which you can use to open and view some example data sets.

3.2.2 The sidebar

VAST provides a sidebar for zooming and moving through the stack. The sidebar is a region close to the left and the right edge of the main window. When you move the mouse cursor to the left or right edge of the window you will see it appear as a transparent white overlay strip.[6] Clicking into the sidebar and dragging the mouse up or down will scroll through the slices of the stack (left mouse button) or zoom (right mouse button). If you move the mouse cursor too far away from the side of the window, the view will 'jump back' to the previous view. If you move the mouse cursor very close to the top or bottom of the window while scrolling (not zooming), VAST will start to scroll continuously, with a speed depending on mouse cursor position. You can use this function to quickly scroll through a very large image stack.

[5] See for example `https://arxiv.org/abs/1306.3543`. Essentially VAST requests zipped [128x128x16] pixel blocks of the data set from the data server with URLs which specify the requested region, like: *http://openconnecto.me/ocp/ca/kasthuri11/zip/6/1,129/1,129/1,17/*. The received file is then unzipped to extract the image data.

[6] You can set the opacity of the sidebar in the Preferences, under 'Side Bar Opacity'

3.2.3 The RAM usage indicator

At the right side of the toolbar you can see a little field named 'RAM:' which shows the current RAM usage in your computer. The blue frame indicates how much RAM VAST will use maximally for segmentation- and image cache combined. Make sure that this frame is not dedicating more than 2/3 of your total RAM (you can adjust these settings in the Preferences, see section 2.2.2). The solid blue block shows how much RAM VAST has currently allocated for segmentation and image cache. The light blue area shows how much memory is allocated by VAST for other purposes. The green area shows how much RAM the Windows system and other programs are using. The colors will change to yellow if the total memory usage goes above 90%, and red if they go above 96%. Running out of available RAM can slow down your system significantly. However, in some cases Windows uses large amounts of the available RAM for disk caching and can free those instantly if more RAM is needed by programs without affecting the system performance.

3.2.4 Getting and setting coordinates

VAST uses a coordinate system with a zero point in the upper left corner of the first slice, with positive X to the right and positive Y down in the slice, and Z marking the slice number. Coordinates are given in pixels at full resolution (the coordinates are independent of the mip map displayed). The coordinates displayed in the upper left corner of the main window show the current location of the center of the main window. You can switch the displayed coordinates on and off by using 'Info / View Coordinates' from the main menu. Zooming in or out will not move the center point of the window and therefore also not change its coordinates. Getting or setting coordinates will also use the coordinates of the center of the screen, as do the 'anchor points' of segments (see section 3.3). While you drag the slice with the mouse VAST displays a transparent cross which indicates the location of the center.[7]

Once you load an image stack, a tool window labeled 'Coordinates' will appear in the upper right corner of the main window. If the tool window is not displayed you can open it using 'Window / Coordinates' from the main menu. It shows you the current center coordinates and allows you to read and set these values. The edit field in the tool window is updated as you navigate through the stack. To save the current location, simply copy the coordinates from that text field (mark with the mouse and press CTRL-C), then paste it into the text editor of your choice. You can also set the coordinates by entering or pasting numbers here and pressing Enter. VAST will then jump to the new coordinates. The exact format of the string does not matter; VAST simply looks for the first three numbers in the string. VAST does not mind whether there are commas or brackets or other non-numerical characters.

This function is quite useful if you want to store coordinates of interesting points in an external text file or spread sheet. Please keep in mind that the coordinate denotes the center of the current view. The center is indicated by transparent crosshairs when you pan the view. You can also center any point by right-clicking that point with the mouse in 'Move' mode and selecting 'Center' from the context menu.

The dropdown-listbox in the Coordinates tool window lists the up to 64 most recent locations you visited. A new entry is added every time you pan the view (but currently not if you scroll through Z). You can go back to previous locations by selecting the coordinates from this list.

3.2.5 Layers

VAST can open several image and segmentation stacks at the same time, provided that they have the same stack size. Each stack is listed as a 'Layer' in the 'Layers' tool window. The order in the list defines the order of the layers. Layers BELOW in the list are 'in front'. The segmentation layers

[7] You can set the opacity of the center cross in the Preferences, under 'Center Cross Opacity'

are always rendered on top of all image stack layers. You can change the order of the layers by drag-and-drop in the list. If you can not see all layers in the list, increase the size of the tool window by dragging a corner.

One image layer, and, if available, one segmentation layer is the selected layer of its kind, and its name is displayed in bold text in the 'Layers' tool window. One of these, the one that was clicked last, is also highlighted, and is the layer for which the 'Layer Properties' are shown in the 'Layers' tool window, below the list of layers:

- 'Solo': If this function is enabled, only the currently selected layer will be displayed.
- 'Editable': Uncheck this for segmentation layers you do not want to accidentally paint into. Disabled for image layers
- 'Visible' (image stacks only): Transparency value for this layer. Switch off to hide layer.
- 'Bright' (image stacks only): Image Brightness; switch on to enable brightness control with the slider
- 'Contrast' (image stacks only): Image Contrast; switch on to enable contrast control with the slider
- 'Alpha' (segmentation stacks only): Opacity value for this layer. Switch off to hide layer. Duplicated in toolbar.
- 'SelAlpha' (segmentation stacks only): Opacity value for selected segments. Duplicated in toolbar.
- 'Pattern' (segmentation stacks only): Contrast of segment patterns. Duplicated in toolbar.

VAST can blend image layers with different transparency modes. Click on the button 'Menu' to access more layer options. Under 'Blend Mode', you can select either 'Blend' for alpha-blending or 'Additive' for additive blending. The different settings for the transparency computation are:

- 'Flat': All pixels in the image share the same transparency [Default]
- 'Dark Transparent': The darker a pixel ((R+G+B)/3), the more transparent it is
- 'Bright Transparent': The brighter a pixel ((R+G+B)/3), the more transparent it is
- 'Max(RGB) Dark Transparent': The darker a pixel (Max(R,G,B)), the more transparent it is
- 'Max(RGB) Bright Transparent': The brighter a pixel (Max(R,G,B)), the more transparent it is

For RGB image stacks, 'RED Channel Target Color', 'GREEN Channel Target Color' and 'BLUE Channel Target Color' let you swap and disable different color channels individually, and even assign an arbitrary mixed color to each channel for more sophisticated color space transformations. This is particularly useful for mapping colors in fluorescence microscopy image stacks.

'Color Filter ...' will open a color selection dialog where you can choose a color by which the layer images should be (subtractively) filtered for display. To not filter the images, choose white (255,255,255) [Default].

3.3 Painting

The main function currently provided by VAST is painting of segmentations as a colored overlay of the image data. When a stack of EM images is loaded, you can enter 'Paint Mode' by clicking the little pencil icon in the toolbar. When you start a new segmentation like this, VAST will ask you if you want to add 16 segments (label colors) to your segment list. Also, two floating tool windows will appear at the right side. The upper one, 'Drawing Properties' (Figure 3.3), provides options for drawing, whereas the lower, 'Segment Colors', lets you select and organize the segment labels and their colors in the segmentation.

To add a new separate segmentation layer, click 'Menu' in the 'Layers' tool window and choose 'Add New Segmentation Layer'. Each segmentation layer holds a separate set of segment colors and is (or will be, when saved) associated with its own segmentation file.

When in paint mode, you can paint on top of the currently displayed EM image. Select a color (label number) from the 'Segment Colors' window at the right by clicking on it. Then click the left mouse button where you want to paint in the image.[8] You will see the outline of your current tooltip as a circle. By clicking and dragging the mouse you can paint larger regions. All painting happens in an overlay plane, the segmentation layer, which is blended over the EM image (the EM image itself will not be changed). You can use the 'Alpha:' checkbox in the toolbar to switch the selected segmentation layer on and off, and the slider right of it to set its opacity. Most colors are not solid colors, but have patterns. Use the 'Pattern:' checkbox to switch patterns on and off, and use the slider right of it to manipulate the contrast of the patterns. If you enable the 'SelAlpha:' checkbox, the opacity of the selected segment and its children will be controlled separately by the SelAlpha slider. You can use this to highlight a particular segment or set of segments. You can also switch the EM image layer on and off, by clicking on the 'EM' checkbox in the toolbar. This is sometimes useful if you want to inspect just the segmentation. These controls are duplicated in the 'Layers' tool window for the selected layer.

You can change the size of the pen tooltip. The easiest way is, if you are using a pen tablet and VAST is properly configured, to hold down one of the pen buttons and to move the pen up or down on the screen. You can also use the - and + keys on the keyboard. The current pen diameter (in pixels) is displayed in the Drawing Properties tool window. The third way of changing the tooltip size is to edit the 'Pen Diam.' text field in the tool window. You can also lock the current tooltip size if you don't want it to be changed accidentally, for example if the size of the tooltip is important for your data analysis, by switching on 'Lock'.

The checkbox 'Fill' next to it switches automatic filling of closed contours on and off. If enabled, after each paint stroke VAST checks a rectangular area with the approximate extent of the stroke for empty closed contours of paint color and fills them with the paint color.

Below you can choose from 'Paint All', 'Background' and 'Parent'. This determines which voxels in the current segmentation are paintable. If you select 'Paint All', you will paint over or delete anything, no matter if it was painted before or not. If it is set to 'Background', previous paints will not change, but your paint will only be applied to voxels which have not yet been painted to. If you erase, only the current paint color will be erased to empty (background). This is the most useful painting mode.

Instead of only affecting background pixels, 'Parent' mode will affect only pixels which have the color of the immediate parent of the current paint color (see below for a description of segment hierarchies). When erasing, voxels with the current paint color will be changed back to the parent color. This mode is only useful in special cases, in particular when re-labeling a previously painted area to a new color.

[8] Even though you can use VAST with a mouse, it is designed to be used with a pen tablet.

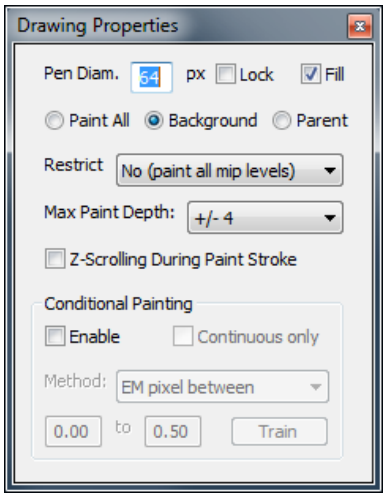

Figure 3.3: The Drawing Properties tool window

3.3.1 Multi-scale painting

A specialty of VAST is that it allows you to paint at different resolutions. In fact, VAST limits you to always paint at the currently displayed resolution. The advantage of this is that the amount of data that has to be manipulated when you paint a stroke is limited by the window size and screen resolution. Otherwise, for very large volumes one could easily get into a situation in which the amount of data that has to be written for a paint stroke is much larger than what can be loaded in RAM at one time, which would cause all sorts of problems, including very slow painting. Also it does not make sense to paint at a resolution which is much higher than the screen resolution because mouse (or pen) precision is also limited. Finally, allowing low-resolution painting can save a lot of memory, if large objects are painted coarsely.

In VAST, images are stored as a pyramid of mipmaps with reduced resolution using powers-of-two factors. Painting always happens at the resolution of the currently displayed mipmap. This means that you can change the resolution at which you are painting by zooming. A single segmentation can be composed of parts at different resolutions. For example it is possible to draw a rough outline of an object at a low resolution, and then to zoom in and correct the object's shape at a high resolution. VAST will automatically upscale and downscale the displayed segmentation as you zoom, but zooming will not change the painted segmentation. The segmentation is stored at the resolution at which it was painted. If you paint at a low resolution first and then correct at a high resolution, part of the low-resolution segmentation will be replaced by a high-resolution version. If you paint at a high resolution first and then correct at a low resolution, part of the high-resolution segmentation will be replaced by a low-resolution version, including pixels in the vicinity.[9]

Sometimes you might want to make sure that a painted segmentation has a certain resolution. You can enforce painting at only one resolution by restricting painting to a particular mipmap ('`Restrict`' in the `Drawing Properties` tool window). VAST will then enable painting only when the image stack is zoomed to display the selected mipmap.

[9] I have to do this because at the time of painting at a low resolution, not all higher-resolution images may be available in RAM (they may even be too large to be loaded in RAM).

3.3.2 Automatic Z-filling

The time that has to be spent to manually paint a segment in VAST depends largely on the number of 2D outlines that have to be drawn. Especially if you follow a process that runs vertically through the volume, you have to paint (almost) the same outline over and over again, for every slice. If you want to just get a rough outline of an object and you're not interested in a high precision of the boundary, you could increase the painting speed by a factor of n if you paint the outline only in every n-th slice, or paint n slices at a time. In the first case, you get gaps of $n-1$ slices between the painted outlines, in the second case it is hard to determine what you are actually painting because you can't see where your color goes in most of the slices.

VAST uses a third method. It supports automatic Z-filling of intermediate slices where the regions of the lower and the upper painted region overlap. It turns out that in most cases neuronal objects are *locally convex*. Exceptions are branches, for example when a spine neck runs very close to the dendritic shaft. Automatic z-filling will only fill in the volumes of the overlap between the specified painted regions, and in most cases (for convex objects) the filled regions will stay inside the segmented object.

Z-filling makes sense across a few slices only, because there will be no overlap if your object moves too much from slice to slice (runs at an oblique angle). Also, VAST has to load multiple slices in RAM to be able to fill in those slices. The maximal distance across which VAST lets you fill in depends on the size of the image cubes used. Currently the cubes are set to be 16^3 voxels large, and VAST allows you to fill in up to +-8 slices (because it loads two layers of cubes at a time). In the data sets we are using this is approximately as far as z-filling makes sense, and it speeds up painting by a factor of 8.

You can set how far the z-filling will reach by setting '`Max Paint Depth`' in the 'Drawing Properties' tool window. This value controls both the distance at which Z-filling occurs, and the stepping distance for navigating with S, X or PageUp, PageDown keys, to ensure gap-free painting.

Automatic Z-filling is only applied while painting, not when erasing. This makes it easier to correct what has been filled in in the case of non-convex neighborhoods. This also means that the best strategy to draw an object coarse-to-fine is to try to paint conservatively (try to stay within the object boundaries), and correct by adding paint rather than removing paint, because deleting has to be done in every slice individually.

'`Z-Scrolling During Paint Stroke`' is by default disabled to prevent painting errors when accidentally switching to the next slice before the paint stroke is finished. However, if you enable it, you can very quickly coarsely label a long neurite running through your stack vertically by scrolling through the stack while following the neurite with your pen – provided that loading of the image stack keeps up with the update rate of the screen.

3.3.3 Using conditional painting

The last section of the `Drawing Properties` tool window handles the settings for 'conditional painting'. If you switch on conditional painting by clicking the check box `Enable`, only pixels will be painted for which the EM image fulfills certain criteria. You can choose from several methods which determine the paintable pixels depending on whether the brightness of the image pixel (of the selected image layer) is in a certain range, which you can set. The value range for minimum and maximum brightness is [0..1]. If you are using several image layers and conditional painting does not seem to work, please make sure that the correct image layer is selected in the 'Layers' window.

The last method in the list, 'Use Trained Color', lets you interactively set which color or brightness range to paint from examples. If you click the 'Train' button, a small tool window will appear which lets you set and clear positive and negative examples from painted segments. This currently only uses the brightness / color statistics of single pixels and can not learn patterns or textures. To use this function, paint over some image area which you want to have labeled in one segment color,

select that segment color and click 'Add Positive from Selected Segment'. Then use a different segment color to label some image area which should be excluded from painting, make sure that segment color is selected, and click 'Add Negative from Selected Segment'.[10] Then set the 'Method' to 'Use Trained Color'. Subsequently VAST will only paint over pixels which have brightness or color similar to the positive example pixels. If you add positive or negative examples several times, the new examples will be added to the set of already defined examples, until you clear the training data.[11]

3.4 Segments

3.4.1 Picking segments

You can select the segment color to paint with in the 'Segment Colors' tool window by clicking on it in the tree view. You can also pick any color you see in the segmentation layer by using the pipette tool. To do this hold down the SHIFT key and click on the segment you wish to select. This makes is very easy to switch between segment colors while painting. Alternatively you can use the Pipette mode which you can select in the main toolbar. If you have several segmentation layers, you can pick colors in a different segmentation layer as well, and VAST will automatically make that segmentation layer active. If you hover over segment colors in the main window when in picking mode, VAST will display the name of the segment as a tooltip. If the segment is in a different segmentation layer, the name will be shown in brackets together with the layer name.

3.4.2 The segment hierarchy

VAST can arrange segments in a tree-like hierarchy. This means that each segment can have other segments as children, which can themselves have children, and so on. VAST also allows you to collapse and expand parts of the tree dynamically, so that you can quickly switch between a visualization which shows a whole branch of the tree in the same color or individual sub-branches in individual colors. For example, if all spines of a spiny dendrite are labeled as sub-objects (children) of the dendritic shaft, one can instantly flip between a display in which the whole dendrite has the same color, or each spine has a different color, by opening and closing the dendritic shaft folder. Segments can also be used as folders to group segments, for example to classify labeled objects. You can use tags to designate certain segments as folders, to help external analysis (see section 3.4.10). The grouping can also be applied when segmentations are exported.

Segment hierarchies are visualized and edited in the 'Segment Colors' tool window. This tool window uses a 'tree view control', similar to the navigation pane of a windows explorer window, which makes usage very intuitive. Most advanced functions can be found in the tool window's menu, which opens either by clicking the 'Menu' button or right-clicking into the tool window.

3.4.3 Re-ordering and moving segments in the tree

To re-order the segments, simply drag and drop them with the mouse. You can only select and drag one segment at a time, but if the segment has children the whole branch will be moved (including all children). Please note that to make a segment the first child of another segment, you have to drag it to the right side of the tool window, right of the new parent segment. The new parent segment will then be highlighted in blue, instead of the black line indicating the target space between two segments. You can move any segment, with exception of the 'Background' segment. The Background segment can also not have any children.

[10] VAST only scans the pixels in the selected segment color which are in the currently displayed slice and region.
[11] Internally VAST uses a 64^3 HSV cube for example storage and color space mip maps for generalization.

Because it can be cumbersome to move hundreds of items from one folder to another one by one, VAST currently supports two functions 'Make all siblings children' and 'Make all children siblings' which can help in certain situations (see section 3.4.7).

3.4.4 Collapsing and expanding tree branches

You can collapse and expand tree branches, which are displayed in the same way as folders and subfolders are in the Windows explorer, by clicking on the little '+' or '-' sign left of parent segments. When you collapse a folder, all its children will be displayed in the same color as the parent. If you pick a segment color from the segmentation layer by shift-clicking, and the selected segment is in a collapsed folder, the folder will be automatically expanded to show the native color of the segment you selected. In addition the context menu has two functions to set the collapse state of all subfolders of the selected segment at once, under 'Expand / Collapse Child Folders'. Also see section 3.4.10.

3.4.5 Using anchor points

Each segment has an 'Anchor Point' stored with it. This is an XYZ coordinate vector which indicates the location of the segment in the image stack. Initially the anchor point is set to the location at which the segment is painted first. You can jump to the anchor point by right-clicking on a used segment in the 'Segment Colors' tool window and selecting 'Go To Anchor Point' from the context menu. You can quickly jump to the anchor point of the selected segment by pressing the 'Home' key or 'G' (for 'go to').[12] You can also set the anchor point of the selected segment to the current view location (as indicated by the center cross) by selecting 'Set Anchor Point' from the context menu. You will have to confirm this action in a pop-up window to prevent accidental setting of anchor points.

3.4.6 Adding new segments

The context menu of the 'Segment Colors' tool window provides several functions to add more segments to the selected segment layer. You can add a segment as next sibling or as a child of the selected segment. 'Add 10 Segments' will add 10 segments immediately after the selected segment. 'Add Skeleton Segments' adds a set of child segments to the selected segment which can be used for rudimentary skeletonization. I have not found this function particularly useful though.

The most sophisticated way to add segments is 'Add Named Segments ...', which lets you specify a naming scheme and add multiple named segments at the desired target location in the segment tree. VAST will attempt to guess a naming scheme from the name of the currently selected segment.

You can change the name of any segment in the same way as file names are changed in the Windows Explorer – click a selected segment name a second time, then rename it.

3.4.7 Helper functions for arranging segments

Under 'Arrange' in the context menu you can find two useful functions to move many segments at once. 'Make All Siblings Children' will move all siblings of the selected segment into its folder (make them children of the selected segment). 'Make All Children Siblings' moves all children of the selected segment out of its folder and makes them siblings. Be careful with these functions because currently there is no 'Undo'.

[12] If you pressed the 'Home' key accidentally and want to go back to where you were, you can select the previous location from the drop-down menu in the 'Coordinates' tool window.

3.4.8 Select recently selected segments

Under 'Select Recently Selected' in the 'Segment Colors' tool window's context menu you can find a list of the segments you had recently selected. You can click on one of the listed segments to select it again.

3.4.9 Global operations: Deleting and welding segment subtrees

The context menu of the 'Segment Colors' tool window provides two functions which change segment numbers in the whole segmentation layer, one to delete segments from the segmentation layer ('Delete Segment + Subtree') and one to weld all children in a subtree to its parent (making them all the parent color, while removing the child segment numbers), 'Weld Segment Subtree'.

Deleting and welding segments is actually more difficult than it seems because it involves traversing the whole segmentation data set and inspecting every single painted voxel. When you choose to delete the selected segment and its children, VAST will actually have to not only set all voxels with those segment numbers to 0, but also renumber all the other voxels so that the used segment numbers will have no 'gaps' (all segment numbers between 0 and n are used).

'Welding' will merge a whole segment subtree into a single segment (the parent). For this VAST needs to renumber all voxels with segment numbers of children of the selected segment to the segment number of the selected segment, and also, similar to when deleting, renumber the other segment voxels so that the resulting segmentation is free of segment number gaps.

Because these functions change almost the complete segmentation and VAST is designed so that the opened segmentation file is not changed, it basically has to copy almost the complete segmentation data to the temporary cache file. Depending on what segmentation you are working on this can take a lot of time, RAM and disk space. Also it will change the internal ID numbers of the segments, so please think twice before using these functions in case you are relying on absolute segment IDs in your analysis. These functions are also not well tested; please report any bugs you might encounter.

3.4.10 Segment tags

Each segment can have a 'Tag' which you can select in the 'Segment Colors' context menu under 'Tags'. A tag is a number between 0 and 15 which can indicate the type of the segment. The tag value is exported with the Segment Colors text file and can be used to help external analysis. By default the tag of all segments is 0. VAST uses tag 1 to indicate that the segment is a 'Folder Segment', which is not a labeled object by itself but rather a folder which contains other folders and objects. This information can be used to collapse all objects, but expand all folders – select 'Expand Only Segments Tagged as Folder' under 'Expand / Collapse Child Folders' in the 'Segment Colors' tool window context menu.

Segments which have a tag that is not 0 will have an icon with a colored frame in the Segment Colors tree window. Folder segments have a gray frame.

3.4.11 Editing the color of a segment

The color and pattern of any segment can be changed. Right-click on a segment and go to the submenu 'Colors' of the context menu, then choose the desired option. Each segment has a primary color, a secondary color, and a pattern that is used to blend between them. You can also randomize the colors of all segments of the selected subtree (the selected segment and all its children), or set the primary or secondary color or pattern of all segments of the selected subtree the same.[13]

[13] If the Background segment is selected, these functions will apply to all segments in the segment layer (except the Background segment).

Please remember that by collapsing segment folders you can quickly and reversibly switch the displayed color of a segment to the color of the collapsed parent, without the need to change the color of all children individually.

3.4.12 Exporting segment metadata

The entry 'Save Segment Colors...' in the context menu lets you export the metadata of the segments to a text file, which you can then for example parse with MATLAB to extract colors, hierarchies, names, anchor points etc. of the segments for analysis. A Matlab script which can parse these files is included in the supplementary package (scanvastcolorfile.m). The format of the text file is described at the beginning of the text file itself:

1	ID number of the segment
2	Flags field of the segment as 32-bit value
3-6	Primary color as red, green, blue, pattern1
7-10	Secondary color as red, green, blue, pattern2 (unused)
11-13	XYZ coordinates of the segment's anchor point (in voxels)
14-17	IDs of parent, child, previous and next segment (0 if none)
18	If the segment is collapsed into a folder, this is the folder ID
19-24	Segment bounding box (may be incorrect if voxels were deleted)
25	Segment name as text in " "

Table 3.1: Columns of the segment metadata text file

The 'Flags' field uses the following bits: 0: isused, 8: isexpanded, 16,17: isselected (1: selected, 2: in child group of selected)

3.4.13 Segment information

'Segment Information ...' in the context menu opens a window which shows you the internal information associated with the selected segment. You can use this information to count children of the segment, get its internal ID or other parameters. Getting the segment info of the 'Background' segment returns statistics for the whole segmentation layer. The text can be copy-pasted if needed.

3.4.14 Searching for a segment with a given name or ID

At the top of the 'Segment Colors' tool window there is an edit field which can be used to find segments. As you type or paste a text string into this field, VAST will select the next segment (after the selected segment in depth-first search order) the name of which contains the typed sub-string. The edit field is case-sensitive. If there is more than one segment which contain this sub-string, you can click on the magnifying glass right of the text edit field to go to the next segment that matches your text. The F3 key has the same function, provided that the segment tree sub-window of the 'Segment Colors' tool window is active.

You can also search for a segment with a particular ID. To do this, type an opening bracket [into the find edit field, followed by the ID number (internal segment number) you are looking for.

3.4.15 The 'Collect' tool

The 'Collect' tool ('Collect Segment Mode') can be selected in the toolbar by clicking the icon with an arrow pointing at a folder. When in Collect mode, segments you click will be moved in the segment tree to become children of the currently selected segment. This can be useful to quickly

classify different objects into different types. Simply make a folder segment for each type, select it, and click on the objects in the image that are of that type with the 'Collect' tool.

Using this tool is a bit dangerous because there is no Undo. If you click the wrong object it might be difficult to remember where it came from and there is currently no easy way to 'move it back'. Secondly, when an segment is moved, all its child segments are moved with it, but not its parent(s). So if you are dealing with objects which consist of several parts, make sure that you move the parent segment of the object and not only a side branch of its tree.

3.5 Saving Segmentations

Important: VAST DOES NOT SAVE AUTOMATICALLY while you paint. Your tracings will be held in RAM and/or a cache file on disk until you explicitly save them. If you open a segmentation from a `.VSS` file and work on it, the file will not be changed unless you explicitly tell VAST to save the changes you made to the opened file by selecting '`Save Segmentation`' from the main menu. If you want to keep the previous version and save to a new file, use '`Save Segmentation As ...`' instead. VAST will then take all data from the opened file, the RAM cache, and the segmentation cache file, and combine them into a new file on disk.

We have had cases in which people had VAST open for several days without saving and lost a lot of work when the computer crashed. Please save your work once in a while.

3.5.1 Save Segmentation As Special

'`Save Segmentation As Special ...`' provides you with two functions to save your current segmentation to a new file in a modified way. First, you can choose to save only the selected segment or subtree of the current segmentation to a new file. Alternatively, you can change the resolution of your segmentation and adjust the canvas size on which the saved segmentation will open (this is currently only supported when saving all segments).

To save only the selected segment or the selected segment and its child tree to a new segmentation, select that option from the drop-down list and press 'Save'. Please be aware that the internal IDs of the segments in the new segmentation file will be changed to maintain a gap-free list of IDs from 1 to n for n segments.

The settings in the lower part of the dialog provide limited functionality to adjust the resolution of the saved segmentation and target canvas size.

Normally VSS files only open on top of an image stack of the same size in voxels. You can use this function to save an existing segmentation so that you can open it on a stack which has slightly different size and/or is scaled up or down by a power of two.

Currently the saved segmentation will stay aligned to the upper, left, top corner of the stack. Also, only powers-of-two scaling is possible. If you want to translate the segmentation to a different location or scale in the target stack, you'll have to export the segmentation as an image stack, modify the images accordingly, and re-import.

3.6 Segmentation Merging

If you have two or more (for example partial) segmentations (`.VSS` files) of the same image stack you can merge them into a single segmentation. For this, first open one of the segmentations you want to merge in VAST, and then select '`File / Merge Segmentations in ...`' from the main menu, and choose the `.VSS` file(s) you would like to merge in with the opened segmentation. During merging, VAST will add the selected `.VSS` files to the current segmentation. VAST lets you choose whether you want to change segment IDs of the merged-in data so that they do not overlap with existing segments, or keep segment IDs of the imported file (then only segments with new IDs will

be added to the segment list, and segments with existing IDs will extend pre-existing regions). You can also choose whether to put new segments into a separate folder, and whether or not to overwrite pre-existing voxels or only write into empty voxels. You can select several `.VSS` files at once for merging, which will then be added one-by-one.

VAST does not save the merged segmentation automatically nor does it change any of the segmentation files. It will generate the merged segmentation in the segmentation cache, and if it does not fit in RAM it will end up in the temporary segmentation file. You will have to save the resulting merged segmentation if you want to keep it, for example via '`File / Save Segmentation As ...`' from the main menu. Please make sure that there is enough space on the drive used for the disk cache.

3.7 Importing Segmentations From Image Stacks

VAST can import segmentations from image stacks, either generating a new segmentation or merging with a previously loaded segmentation.

Just like the image files generated during segmentation exporting (see section 3.8 below), the RGB values of the image pixels should encode the segment numbers (the least significant 8 bits (0..7) are in the blue channel, bits 8..15 in the green channel, and bits 16..23 in the red channel). Please keep in mind that VAST can currently only handle 16-bit segmentations, so the red channel should always be 0. However, because of the requirement to have gap-free segment numbers in segmentations, VAST will scan all images for existing segment numbers and renumber them as necessary if it finds gaps or channels are used differently than expected.

To import, select '`File / Import Segmentation from Images ...`' from the main menu. VAST will ask you to select one or several image files of the image stack you want to import. Then it will open a dialog where you can specify the parameters for segmentation importing.

File name(s):
You have to specify a name template for all image files in the stack. This has to be a *C++ format string* (as it is used by `printf()`). This means that numbers which specify the slice, column and row coordinates in the file name have to be replaced by codes like '`%d`' (integer number) or '`%04d`' (integer number with zero-padding to 4 digits). VAST will then fill in those numbers for each image. For more information about format strings, please refer to a C++ manual or ask the internet. You also have to specify the order and the value limits for the parameters of your file name template, to define the range of names of all the images or image tiles in your stack.

Image Parameters:
Here you can rotate and flip the images if necessary, and tell VAST where to put them into the currently opened volume. The 'Tile Size' is extracted from the image file you selected, but you can adjust it here too. The 'Start coordinates' are currently not allowed to be negative. Please crop your images prior to importing if necessary.

Segmentation Image Preprocessing:
The option '`Force 8 bit Graylevel`' tells VAST to interpret the images as single-channel 8 bit images, by only using the least-significant 8 bits of the image data. This can help in cases where the loaded images are supposed to be 8 bit graylevel images but are encoded or interpreted as RGB or RGBA color images.

Segment Label Parameters:
If you have a segment metadata file for the imported segmentation stack (same format as the file written in section 3.4.12) you can provide it here. The options below specify how VAST should deal

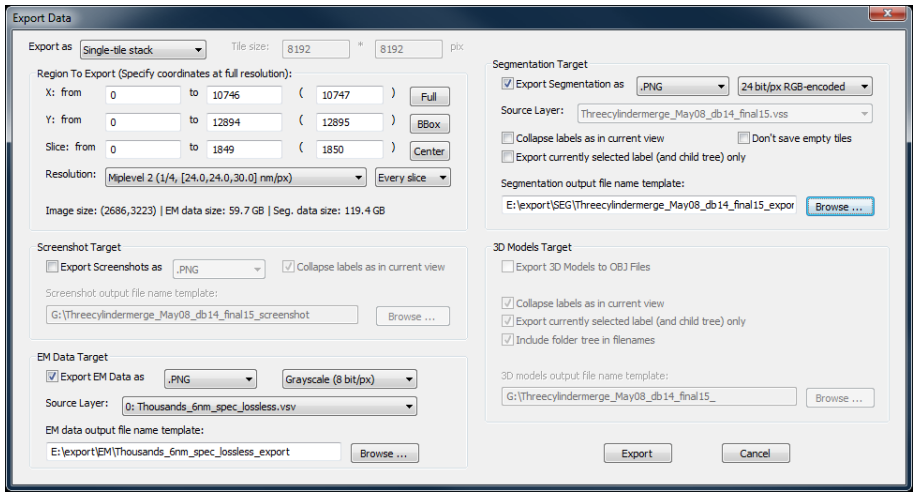

Figure 3.4: Export dialog

with the segments in the imported stack. If you are importing into a previously loaded segmentation, you can either keep the imported segments separate by renumbering and appending them to the existing segments, or merge the imported segments which have already existing segment IDs with the existing ones. New segments will be appended. If you use the option 'Merge with existing segments; Renumber unknown segments' together with a metadata file, the hierarchy in the metadata file will be ignored.

Voxel Combination Parameters:
If 'Overwrite existing non-zero voxels' is checked, imported non-zero voxels will overwrite existing non-zero voxels. If it is not checked, VAST will keep existing non-zero voxels and ignore imported non-zero voxels at the same place.

Similar to segmentation merging, VAST will not save the segmentation after importing automatically. You can do that yourself after importing, using `File / Save Segmentation As ...`' from the main menu.

3.8 Exporting Image Stacks

Exporting EM volumes and/or segmentations as image stacks can be useful to generate slice animations, to transfer data to other programs like the Fiji 3D viewer, or to process your results further externally.

To export (parts of) the image stack and/or segmentation stack as a stack of image files, select '`File / Export ...`' in the main menu. The dialog shown in Figure 3.4 will pop up. You can export EM image stacks, segmentation image stacks, and screenshot image stacks. You can specify a region of the data set to be exported, and a resolution (mip level) for the images. You can also export a data region which is too large for storing the whole slice in a single image as a tiled set of images.

Export as:
Choose here if you want to export an image stack as a single-tile stack (one tile per slice) or a multi-tile stack (a grid of image tiles per slice). For the second option you can define the tile size to be used.

Region to export (Specify coordinates at full resolution):
This defines the region of the stack that should be exported, for all three targets (Screenshot, EM (image) data, Segmentation). By default it is set to the full stack. You can restrict the export region here. X is the horizontal axis in the slice, pointing rightwards; Y is the vertical axis in the slice, pointing downwards on the screen, and Slice (Z) defines the range of slices that should be exported. The first and second columns of the edit fields let you define start and end of the region for each axis, the third column defines the size of the exported region (edit fields change each other to stay consistent).

The 'Full' button will set the values back to the full extent of your stack.

The 'BBox' button sets the region to the bounding box of the currently selected segment. You can use this function to define a cut-out region from a painted segment[14], or use a new segment and just paint the upper left top and lower right bottom corners of the cutout region, select it and use the 'BBox' button to set the export region.

The 'Center' button will center the exported region to the current coordinates in the main window. You can for example use this in the following way: First move the image in the main window to the location you want to export. Then open the export dialog and specify the size of the image you want to export in pixels, in the third column of the 'Region to Export' section, for example to 1024, 1024, 1 for one image with size (1024,1024). Then click 'Center' to place the export region at the currently viewed location.

Under 'Resolution:' you can select at which mip level you want to export. VAST does not support arbitrary scaling, but can export image stacks at its native mipmap scales (which are powers of two). You can also subsample the stack by slices (every nth slice).

Below you can see the image size resulting from your settings and an estimate of the (raw) data size that will be exported. Compressed image formats like .PNG can however produce much smaller file sizes, depending on the image content.

Screenshot target:
If you want to export a stack of 'screenshots' how the images look in the main window of VAST, enable the checkbox 'Export Screenshot as'. VAST will reproduce the pattern, blending and tinting settings as they are currently set in the main window in the exported 'screenshots'. Select a target image format and filename prefix / location.

EM data target:
This saves a stack of (EM) image data from the selected layer. You can specify the target format, filename prefix and location.

Segmentation target:
This saves the segment IDs of the currently selected segmentation layer or part of it as an image stack. The segment ID of each pixel (a 16-bit number) will be encoded in the color of the pixel in the exported image. Bits 0-7 will end up in the blue channel and bits 8-15 will end up in the green channel. The red channel will currently stay 0.

3D models target:
This function is currently disabled because it is not fully implemented yet. For now please use VastTools to export 3D models of segmentations (see section 4.2).

When you are done setting up the parameters for the export, press 'Export' and VAST will start exporting the image stack or stacks (VAST can export more than one target at the same time).

[14] Note that the bounding box is not always correct, in particular if you delete parts of what was painted before, the bounding box will not shrink.

3.9 The 3D Viewer

VAST now contains a simple 3D viewer which lets you view a region of your image stack in 3D. To open the 3D viewer, select 'Window / 3D Viewer' in the main menu of VAST.

To use this you first have to set up the 2D view so that it shows what you want to see in 3D, and then select 'View / Update' in the menu of the 3D Viewer window to load the data. The block shown will be more or less centered around where the current 2D view is located (the center cross location), and use the current 2D resolution (mip level). The transparency in the 3D view is derived from the pixel brightness. You can choose to either 'Show bright', 'Show dark' or apply 'Constant' opacity by selecting the corresponding option under 'Transparency' in the window menu. To see the effect of the selection, update the data by selecting 'View / Update' in the menu of the 3D Viewer window.

Assume for example you have an EM stack and a segmentation loaded in VAST, and you want to see just one segmented object with its children in 3D. For this, adjust the 2D view so that only that segmented object is shown and everything else is black: Switch off the EM layer and 'Alpha', enable and maximize 'SelAlpha', and select the parent segment of the segments you want to show up in 3D. Center the 2D view to the segment of interest and zoom to the desired resolution. Then, in the 3D Viewer window, make sure that 'Show bright' is selected under 'Transparency' in the window menu, and select 'View / Update'. After the data is loaded, just your segmented object(s) should show up in the 3D viewer.

You can zoom the 3D view in and out with the mouse wheel and rotate it in all three axes by dragging with the mouse (the mouse cursor shows which rotation axes will be affected, depending on where the mouse cursor is in the window).

Further options include changing the background color, displaying a line cube around the data block, and choosing from two different block sizes.

Chapter 4

The VastTools Matlab Toolbox

VAST includes an API through which it can communicate directly with client programs, using a TCP/IP connection. The Matlab script VastTools, which is included with VAST Lite, uses this API to provide users with a number of additional tools for VAST, including target lists, 3D surface exporting and measurement functions. It can be found in the `vast_package.zip` file. Since VastTools is a Matlab script, users can extend the interface with their own functions. Documentation of the VAST API is provided in Appendix B.3.

4.1 Getting started with the VastTools Matlab Toolbox

To start the toolbox, open `vasttools.m` in Matlab and run it. If Matlab asks, change the current directory. A small window should pop up with a menu, message field and cancel button.

Before you can use any of the functions of the toolbox you have to connect VastTools to an instance of VAST which is currently running. First, in VAST, you must enable the Remote Control API Server. Select 'Window / Remote Control API Server' in the main menu of VAST. In the tool window which then opens, enable a TCP/IP port for communication by clicking the 'Enable' check box in the upper left corner. If you are running VAST and VastTools on the same computer, you can use the standard settings (IP `127.0.0.1` and port 22081 on both the VAST and Matlab side). If you are using VastTools a lot and do not want to enable the API every time manually, you can set a flag in the Preferences to have it enabled automatically when VAST starts (see section 2.2.2).

Then, in VastTools, connect to VAST by selecting 'Connect / Connect to VAST' in the main menu of the VastTools window. If the menu item 'Connect / Connect to VAST' changes to 'Disconnect / Disconnect from VAST' you are connected. The message log in the VAST Remote Control Server window will also show when a remote connection has been accepted. In case you are using a different IP or port, for example because VAST and VastTools run on separate computers, you can set the IP address and port to use under 'Connect / Connection Options' in the VastTools main menu.

Once connected, you can use the different functions of the VastTools toolbox by selecting from the main menu of the VastTools window in Matlab.

4.2 Exporting 3D Models

3D model exporting generates surface meshes of the painted voxelized segmentation in VAST, and saves the resulting meshes in Wavefront OBJ files which are widely supported by 3D rendering and animation programs. VastTools uses Matlab's `isosurface` function to generate the meshes. Typically, it will generate surface meshes in parts of the volume to limit memory usage, and then glue together the mesh pieces to generate the final objects. The exporter also saves a .MTL file for each

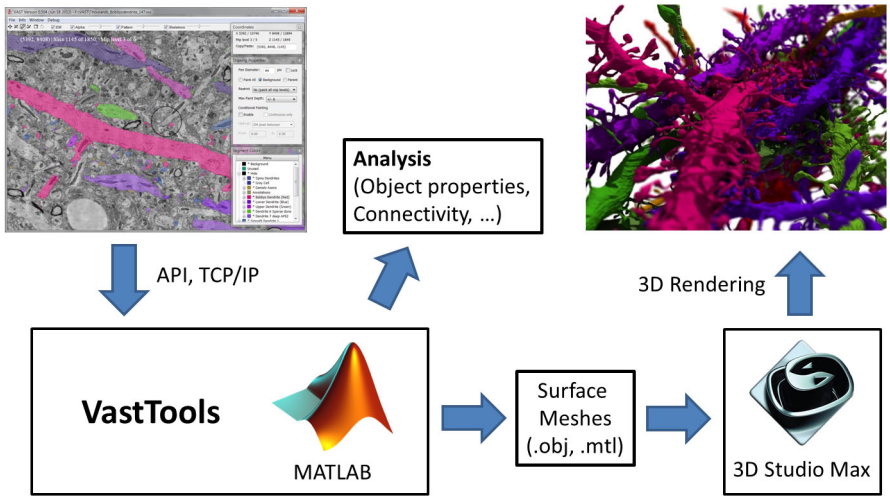

Figure 4.1: Exporting from VAST using VastTools

OBJ file which defines the material properties (color) as defined in VAST. You can also measure the surface area of all exported objects and save the results to a text file.

If you select 'Export / Export 3D Surfaces as OBJ Files ...' from the main menu in VastTools, a dialog window will pop up in which you can specify all the parameters for the export. Once you click 'OK' the exporting will start and run until finished or until you press 'Cancel' in the main VastTools window. This will export objects of the *selected segmentation layer*.

The following parameters for the export can be set:

Render at resolution:
Here you can select the mip level at which the surfaces should be generated. Lower mip levels will result in more detailed models, but also generate larger OBJ files with more triangles and take longer to process. Models and surface computations will be automatically scaled to compensate.

Use every nth slice:
If you want to reduce the resolution in Z you can do so by skipping slices. If you set this value to 2, every second slice will be used, if you set it to 3, every third, and so on. Models and surface computations will be automatically scaled to compensate.

Render from area:
This specifies the region of the current volume in which you want to generate surface models. By default the region is set to cover the whole volume, but you can change the values to cover a smaller region if needed. These values are always specified at full resolution, no matter which mip level is set in the 'Render at resolution' selector.

'Set to full':
Push this button to set the area boundaries to the full extent of the data set opened in VAST.

'Set to selected bbox':
This sets the render area to the combined bounding box of the segment selected in VAST and all its

children. CAUTION: The segment bounding boxes are currently not always correct. For example, if you erase voxels you previously painted in VAST, the bounding box will not shrink.

'Set to current voxel':
This button sets the render area to a single voxel, the current center voxel set in VAST. Use together with 'Extend to current voxel' to define a target area by its corner points.

'Extend to current voxel':
This button extends the boundaries of the render area to include the current center voxel set in VAST. Use together with 'Set to current voxel' to define a target area by its corner points.

Voxel size (full res):
These fields will be filled in automatically by the values provided by VAST. The voxel size is used to scale your models correctly. You can override these values manually by entering different values into the edit fields.

Scale models by:
Additional scaling factors for your models. By default these factors are set to 0.001 (1/1000) in all directions to convert the units from nanometers to micrometers.

Model output offset:
By default the models will be placed so that the upper left corner of the exported region is at the origin (0,0,0). You can provide a constant offset here if you want to move the models somewhere else.

Processing block size:
As mentioned above, models are exported in smaller blocks and then glued together; this specifies the block size. A smaller block size will reduce memory consumption and may increase the speed of isosurface computation, but will increase the processing time needed for glueing model parts together. Changing the block size should only have an effect on memory consumption and processing speed, not on the exported models.

Export what:
Here you can select whether you want to export models of all segments, or of a selected branch, and you can choose whether to export all segments individually or glue them together as they are currently displayed in VAST (by selecting a parent segment, and collapsing and expanding folders in VAST). This information will be read from VAST when you press OK, so you can make adjustments in VAST while the VastTools export dialog is open.

File name prefix:
All OBJ file names, and also the names of the exported objects, will start with this prefix string.

Object Colors:
Here you can select between 'Object colors from VAST' (the default) and 'Object volumes as JET colormap'. The latter only works if you previously computed object volumes (under 'Measure / Measure Segment Volumes ...').

Target folder:
All generated OBJ and MTL files will be stored to this target folder. Use the 'Browse...' button to select a different folder.

Include Vast folder names in file names:
If enabled, the VAST folder names of the segment hierarchy will be added to all OBJ file names, and also the names of the exported objects. This makes it easy to select and process subsets of the objects based on their names. However, please keep in mind that the length of file names can be limited, depending on your file system, and names which are too long can lead to problems.

Invert Z axis:
In enabled, the models will be mirrored so that they lie below the $z = 0$ plane. This reflects the actual shape of the objects in the tissue, if slices are counted up as the are cut off the surface of a block.

Write 3dsMax bulk loader script to folder:
If enabled, a small file called '`loadallobj_here.ms`' will be saved to the target directory. This is a 3dsMax script which, if executed in 3dsMax, will batch-load all objects in the same directory. This is very useful if you are working with many object files.

Close surface sides:
If enabled, meshes will be closed at the sides where they exit the extraction region.

Skip model file generation:
Enable this option if you just want to measure model surface area and not generate OBJ files.

Save surface statistics to file:
If this is enabled, the export script will also compute the surface area of each exported object (by summing up the triangle surface areas) and save the result to the text file with the provided name. This file will always be stored in the same folder as the OBJ files (the target folder), so please give only a file name in the text field, not a file name with target path. Computing the surface area will take additional time, so disable it if you don't need it.

4.3 Exporting Projection Images

The 'Export Projection Image' tool allows you to generate 2D projection images in which your microscopic image stacks and/or segmentations are projected along a cardinal axis. This is for example useful to generate 'renderings' of your segmentation or a Z-projection of part of a confocal light microscopic image stack. You can also use these projection images as 'Simple Navigator Images', see section 4.5. The following parameters can be set in the export dialog:

Render at resolution:
Sets the mip level at which the projection image is rendered. This affects the resolution of the source images in X and Y, but not Z.

Use every nth slice:
To reduce the resolution in Z (and speed up rendering of very large stacks) you can skip slices if you set this to an integer value larger than 1.

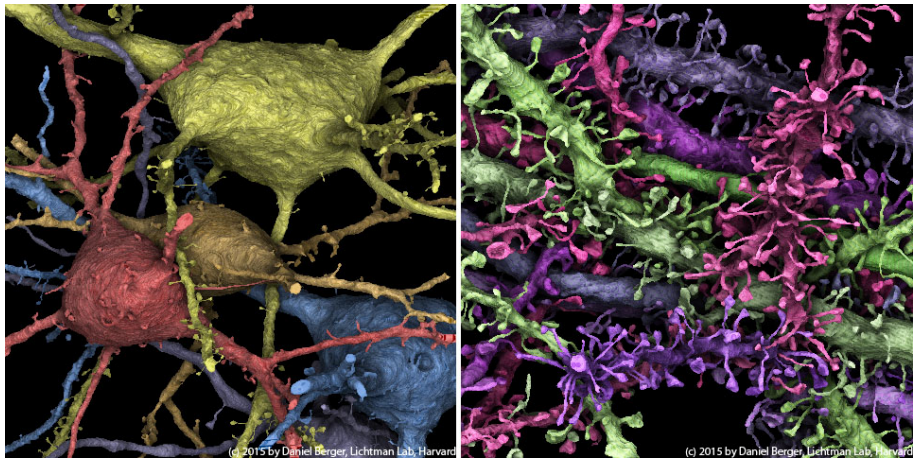

Figure 4.2: Example stack projection images with simulated shadows, generated with VastTools in Matlab

Render from area:
This determines the XYZ block from which the image will be generated. You can use the buttons 'Set to current voxel' and 'Extend to current voxel' together with moving the center cross in VAST to different locations to help you set up a source area. 'Set to full' sets the area to the complete volume, and 'Set to selected bbox' sets it to the combined bounding box of the selected segment and its children. CAUTION: The bounding boxes associated with each segment in VAST are currently not always correct. For example, if you erase voxels you previously painted in VAST, the bounding box will not shrink.

Projection axis, stretching:
Select here along which cardinal axis you would like to project and from what side (which side should be in front in the image), and whether the resulting image should be stretched in Z (for projections along the X or Y axis) if the slice thickness is different from the pixel size. The stretching will be determined by the voxel size of your image stack as set in VAST (see section 3.1.4).

File name, Target folder:
Enter the target file name and location here. The projection image will be saved there after generation if 'Save to file' is enabled.

Segmentation preprocessing:
Select here how you want to use the segmentation for the projection image. This determines which parts of the segmentation are used, and for the 'Screenshots' image source whether the segments should be colored as displayed in VAST (collapsed) or in their individual native colors (uncollapsed).

Expand segments by n pixels:
Makes your segmented regions larger. Useful when using the segmentation as a mask.

Blur edges by n pixels:
Blurs the edges of the segmentation to smooth out the projection image.

Image source:
Select here what you want to use as source images for the projection image. Select 'Screenshots' if you want to use images as they are shown in VAST (using all brightness and contrast settings, layer blending and tinting, segmentation transparency, patterns, etc). A faster option is to just use the primary colors of either the selected or all segmentation layers.

Background color:
Color shown behind the image stack where the stack is transparent

Opacity source:
Determines which parts of each image plane should be rendered opaque for the projection image. Choose to render only segmented areas, only unsegmented areas, or all (Constant). You can also use the pixel brightness of screenshots to define opacity, and either render bright or dark pixels opaque.

Object opacity [0..1]:
Multiplier for the opacity of areas selected in 'Opacity source'. Use a value between 0 (transparent) to 1 (fully opaque).

Blending mode:
Select here whether you want to alpha-blend images with the selected opacity map, add all images, or do a maximum or minimum value projection.

Use shadows, Shadow cone angle:
Enable the check box if you want to include (fake) shadows in the rendering. The shadow cone angle determines how much the shadow spreads out (by means of an image blur filter) from slice to slice.

Depth attenuation (far brightness) [0..1]:
If you set this to values below 1, slices which are further away in the stack will be darkened. This can support the impression of depth in the resulting projection image.

Normalize projection image:
If enabled, the brightness of the final projection image will be adapted so that it uses the whole brightness range (without changing color hue).

4.4 Measuring

VastTools provides several functions to measure objects labeled in VAST.

4.4.1 Measure Segment Volumes

This function can count the number of voxels of different segmented objects in a source area, and save the results to a text file. Parameters in the setup dialog of this function are similar to those of exporting 3D models and measuring surface sizes (see section 4.2 above). Please make sure that the voxel size you use is correct and that you cover the complete area you want to measure to get correct results.

4.4.2 Measure Segment Lengths

This function is not yet implemented.

4.4.3 Measure Segment Surface Area

Since this function relies on a surface mesh to estimate the surface area, it is part of the 'Export 3D Surfaces as OBJ Files' function (see section 4.2). You do not have to generate model files if you just want to measure surface area (check 'Skip model file generation' in the parameter setup dialog). Please make sure that the voxel size you use is correct and that you cover the complete area you want to measure to get correct results.

4.4.4 Euclidian Distance Measurement Tool

This will open a dialog in which you can make simple 3D distance measurements between points in VAST.

First, go to a location in VAST (use the center cross). Then, in VastTools, click the 'Get' button next to the first coordinate to read the current location from VAST. Then go to the other location and click 'Get' next to the second coordinate. The distance between the two locations will show up as a distance in voxels, and in nanometers.

You can easily jump back to the first and second coordinate in VAST by pressing the 'GO!' button.

Make sure you are using the right voxel size to get accurate nanometer measurements. Click 'Update' to read the current voxel size from VAST. To change the voxel size, please set the voxel size in the .vsv file used in VAST. To do this go to 'Info / Volume Properties ...' in VAST and change the voxel size (see also section 3.1.4). Then click 'Update' in the VastTools Euclidian Distance Measurement Tool.

You can copy the values in the edit fields in the Euclidian Distance Measurement Tool to paste them into different programs, but editing these values directly will currently not work.

4.5 Simple Navigator Images

Simple Navigator Images provide a means to navigate in an image stack using a projection image. After you rendered a projection image in VastTools using 'Export / Export Projection Image ...' (see section 4.3), you can generate a clickable Simple Navigator Image from it. Select 'Navigate / New Simple Navigator Image From Last Projection Image ...'. A new window should pop up which shows the last projection image together with a menu and toolbar. Use the arrow tool to click on a segment anywhere in the projection image to have VAST move to that location in your data set. The magnifying glass tool and the hand tool can be used to zoom and pan the image respectively.

Use 'File / Save Simple Navigator Image ...' to save this Simple Navigator Image to a file which you can open again later. You can have several Simple Navigator Images open at the same time, for example projection images along several axes, and navigate using all of them.

4.6 Target Lists

Target lists can store the current view coordinates, zoom level and selected segment together with comments. You can add the current view in VAST to the target list by clicking the 'Add current VAST location' button. To move the view in VAST back to a stored location, click on the 'GO!' field of the row in the table. You then have to make VAST the active window to see the change.

Functions to delete, rearrange and add separator lines to the list are provided in the Edit menu. You can save target lists to a file and load them back later. You can have several target lists open at once and cut/copy/paste between them. You can also select the list, or part of the list, and copy (Ctrl-C) and paste (Ctrl-V) the contents into other programs, like a text editor or Excel.

Target lists are stored as simple .mat Matlab files, so you can generate them yourself if you fill a .mat file with the appropriate variables. Simply load a target list file into Matlab to see which variables it contains.

Appendix A

FAQ and Trouble Shooting

A.1 Frequently Asked Questions

My image stack is not aligned. How do I get it aligned into VAST?
VAST does not have any stack alignment (not stitching) functionality. You'll have to use other programs to render an aligned image stack (for example Adobe Photoshop or plugins in Fiji [1], [2]), and then import that aligned stack into VAST; or you'll have to work on an unaligned image stack.

Can I analyze multi-channel optical image stacks in VAST?
Yes. You can load several image stacks at once, provided they are the aligned and the same size, and each one can be RGB or graylevel. You can tint different image stacks in different colors to distinguish different channels and blend them together in different ways (see section 3.2.5).

Does VAST support 4-dimensional data sets (for example time-lapse data of a 3D structure)?

No.

How do I open a .VSS file without a matching .VSV file in VAST?
This is possible with a work-around. If you load any `.VSV` and then open the `.VSS` in question, and the `.VSS` file has a different size, VAST will tell you its dimensions (size in pixels) in the error message window. Note down these numbers. Now create a dummy `.VSVR` with these dimensions (`VSVR` files are just text (.txt) files with a different file name extension). Set the field 'ServerName' to an empty string to make it a dummy layer. Then restart VAST, open the dummy `.VSVR` as image stack, and open your `.VSS` (which should have the same size) on top. Here is an 'empty dummy' `.VSVR` example:

```
{
  "Comment": "Empty Dummy Layer",
  "ServerType": "openconnectome",
  "ServerName": "",
  "ServerFolder": "",
  "SourceDataSizeX": 49152,
  "SourceDataSizeY": 32768,
  "SourceDataSizeZ": 255,
  "TargetDataSizeX": 49152,
  "TargetDataSizeY": 32768,
  "TargetDataSizeZ": 255,
  "OffsetX": 0,
  "OffsetY": 0,
```

```
  "OffsetZ": 0,
  "OffsetMip": 0,
  "TargetVoxelSizeXnm": 6,
  "TargetVoxelSizeYnm": 6,
  "TargetVoxelSizeZnm": 30,
  "TargetLayerName": "Empty Dummy Layer"
}
```

An empty dummy `.VSVR` file (`emptydummy.vsvr`) for you to edit is also provided in the support package in the folder '`Online Datasets/`' (see section 2.3).

How do I deal with self-touching objects?
If you need to be able to recover the true shape of an object, for example for correct skeletonization or computation of the surface area, places where there are self-touches (for example, a dendritic spine touching the dendritic shaft) can be problematic. The easiest solution is to leave a gap between the two sides, but that is not always feasible, in particular in the Z direction. One way to get around this is by using sub-objects and glue. Just like a plastic model which is constructed from parts, you would make the spine a child of the parent and add 'glue' – a different segment which you treat specially in the analysis – to the interface where parent and child object are actually connected.

How do I make shiny 3D pictures and animations from the segmentations I painted in VAST?
VAST does currently not have 3D rendering capabilities. I use the Matlab-based script `VastTools` to extract `.OBJ` (Wavefront OBJ) model files of objects segmented in VAST (see section 4.2). These models can then be loaded into 3D rendering programs (I use Autodesk 3dsMax).

I accidentally pressed 'Home'. How do I get back to where I was just painting? The tool window 'Coordinates' in the upper right corner of the VAST window keeps a temporally ordered list of recently visited places. Select the second entry from the bottom in the drop-down menu list to jump back.

Suddenly all internal segment IDs changed – What happened?
VAST currently keeps a continuous list of segment IDs between 1 and n for n segments at all times. This means that if segments are removed from a segmentation, the other segments will 'move up' to keep the list continuous.
This happens when you use the functions '`Delete Segment + Subtree`',
'`Weld Segment Subtree`' or save a new `.vss` file with a subset of the current segments using the '`Save Segmentation As Special ...` function. If you are using the internal IDs to identify particular segments and do not want them to change, avoid those functions. For example you can put deleted segments into a 'Deleted' folder and/or re-use them instead of actually deleting them. If you use '`Merge Segmentations in...`' or '`Import Segmentation ...`' with certain settings, the internal segment IDs of the imported segmentation will also be changed so that they don't overlap with existing IDs. You can specify that in the import dialog.

Why is it called 'VAST Lite' and not just 'VAST'?
The 'Lite' in the name emphasizes that this is not supposed to be the final version of the software. It is currently a usable tool with a limited set of capabilities, which is provided to the scientific community without restrictions. Development of VAST continues, and there may at some point be a released version with more features. Also, 'VAST *Lite*' is a better name for Google searches than 'VAST'.

A.2 Typical Use Cases

Analyzing synaptic connectivity in a cortex EM stack
It is possible to define the location of synapses and their synaptic partners by painting the synaptic membrane. We do this in two steps. First, we paint the axons and dendrites with individual colors. Then we generate a second data set (a second segmentation layer) in which just the synapse membranes are labeled, using a pen with fixed size (for example 16 pixels diameter) which is wide enough to overlap with pre- and postsynaptic labels. Make sure that the 3D region painted for each synapse is a single connected component and that different synapses are separate connected components. We then export both data sets as image stacks and use a Matlab script to find each synapse by doing connected-component analysis. For each synapse we then find the axon and dendrite which occupy the same voxels as the synapse in the other data set, which gives us the connectivity information. If axons and dendrites are classified as such either via name or segment hierarchy, we can also extract which side is presynaptic and which side is postsynaptic.

Counting and classifying objects by painting
Just as for the synapses in the previous example, you can use connected-component analysis in Matlab to count other objects in the stack, for example neuron cell bodies. If you paint all neurons of each type in the same color and use different colors for different types, connected-component analysis can be used to separate the different cell bodies for each type, given that they are separated in space. The connected-component analysis will also give you the number of objects of each type, and their volume, if you count the painted voxels for each connected component.

Segmenting out a single cell from a confocal light microscopy image stack
VAST can also load image stacks acquired in light microscopy. If a subset of cells with overlapping branches is fluorescently labeled in such an image stack, VAST can be used to generate a Z-projection image which shows only one cell. For this, first paint over all parts of the cell you want to show. Switch off 'Alpha' for the segmentation layer in VAST so the segmentation is invisible. Then use the VastTools function 'Export / Export Projection Image' to generate a Z-projection image of only the segmented regions of the image stack. For this use 'Screenshots' as image source, 'Segmented areas' as opacity source, and 'Additive' as blending mode. You can optimize brightness/contrast and blending of several layers in VAST to tune the Z-projection image.

Tracking objects in a video
If you translate a video into a sequence of images, you can of course import this image sequence as a stack into VAST (even in color). In the same way as you can label three-dimensional structures in VAST, you can label objects or regions or fiducial points as they move through the video. You can then export the labelings as an image stack and analyze locations in the image and movement.

Defining fiducial points in an unaligned image stack for manually aided alignment
Some EM image stacks are difficult to align with automatic methods, for example if the image quality is bad, there is high-contrast background, or the tissue slices have folds. Manually defined fiducial points which should end up in the same place from slice to slice can help improve the alignment. You can load an unaligned stack into VAST and use manual painting to define fiducial points. Use a different color for each feature you are tracking through the stack, so that in your alignment script you can tell which points belong together.

A.3 Some Performance Tips

- If file access is very slow (when you move through the stack and it takes time until the images appear) consider storing the data locally and/or on SSD drives. In particular, especially when using non-SSD drives, put files which you use together on physically separate drives. The problem is often that two files (for example an EM layer and the segmentation layer) are loaded at the same time from the same non-SSD hard drive, which causes the read/write head to jump forth and back between two locations at high speed. This slows down file access a lot.

- If you experience a low frame rate (the mouse cursor is jumping rather than moving smoothly), try to reduce the size of the 'Maximal Window Width' in the Preferences to something like 1280. On very large screens you can set the 'Target resolution smaller than' to 4 (Default: 2) to help.

- If VAST slows down considerably after using it for a while, check if the RAM of your computer is full (check the RAM usage indicator in the upper right corner of the VAST main window, see section 3.2.3). Once memory is full, Windows might swap parts of the data to the hard drive, which can slow down processing a lot. To fix that problem tell VAST to use less memory by REDUCING the maximal RAM cache memory sizes in the Preferences dialog. This should not cause problems even if you are working with large image stacks since VAST does not need to load all data in RAM at once. The only effect which reducing the cache memory size should have is that VAST may have to load parts of your data set from disk more often, which is slower than reading from RAM.

A.4 Setting up VAST with a Wacom screen

When it comes to fast and accurate manual painting on a computer, tablets and in particular tablet screens can improve productivity significantly. We are using various Wacom Cintiq screen tablets. While the larger Cintiqs have a better screen, they can be expensive and bulky. I personally prefer the Cintiq 13HD, which can be laid flat onto a desk and close to the user.

An alternative could be tablet laptops, if they fulfill the system requirements for VAST. We tried the Asus EEE Slate EP121; it works but it is a bit slow, and pen button presses tend to be unreliable. Not all tablets are suitable for VAST, for example if pen input is unreliable or if the pen has no buttons or the buttons cannot be customized.

Wacom tablets come with driver software which lets you configure the pen buttons for each program. For optimal workflow in VAST I find it most useful to have 'erase' on one pen button and 'change tooltip size' on the other. For this it is easiest to set one pen button to 'right click' and the other one to 'middle click'. If that does not work for your system (sometimes Windows uses the Right Click event in a special way, for example), you can configure the pen buttons to simulate equivalent key presses (see Appendix A.5).

If you see brief circular animations when you use the pen and drawing is delayed, you should go to 'Pen and Touch' in the Windows Control Panel and switch off 'Flicks'. On the tab 'Flicks', uncheck 'Use flicks to perform common actions quickly and easily' and click 'Apply'.

A.5 Keyboard Shortcuts in VAST

You can open a window which lists all the keyboard shortcuts in VAST from the main menu under `Window / Keyboard Shortcuts`. Here is a summary.

`SHIFT or ENTER`	Pick segment color by mouse click
`CTRL or INSERT`	Temporarily go to 'move' mode
`TAB or |`	Move mode: Click left and move pen up/down to zoom;
	Paint mode: Click left and move pen up/down to change pen diameter
`‘~ or DELETE`	Paint mode: Erase mode
`H or L`	Hides segmentation while held down

Table A.1: Mode Modifiers (hold down)

`UP or A`	One slice up
`DOWN or Z`	One slice down
`PAGEUP or S`	`MAXPAINTDEPTH` slices up
`PAGEDOWN or X`	`MAXPAINTDEPTH` slices down

Table A.2: Slice Navigation

`MAXPAINTDEPTH` is the value set under 'Max Paint Depth' in the 'Drawing Properties' dialog.

`-_`	Decrease tooltip diameter
`=+`	Increase tooltip diameter
`N or Keypad /`	Zoom out
`M or Keypad *`	Zoom in
`F`	Flash selected segment
`I, O, P`	Set paint mode to Paint all, Background, Parent
`HOME or G`	Go to anchor point of selected segment
`,< / .>`	Select previous / next segment in recently selected list

Table A.3: Other Controls

A.6 Terms of Usage and Privacy Statement

This version of VAST ('the software') is free of charge and may be distributed freely, but not sold. Commercial usage is allowed.

You are using this software at your own risk. Even though it has been tested extensively, it is not free of bugs. Please keep backup copies of your data.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

VAST does not collect usage statistics or other data. In particular, it does not transmit any of your image or segmentation data anywhere.[1]

This software uses `easyzlib.c`, which is based on the `zlib` library by Jean-loup Gailly and Mark Adler, and `libtiff` with Copyright (c) 1988-1997 Sam Leffler and Copyright (c) 1991-1997 Silicon Graphics, Inc. VastTools uses `jtcp.m` by Kevin Bartlett.

[1] Except if you set that up explicitly using the Remote Control API Server of course. You can make it transmit segmentation data through the API to VastTools for example, but that is under your control.

Appendix B

Technical Information

B.1 Size limitations

Maximal file size for EM images and segmentations: Limited by maximal file size on disk; theoretical maximum 2^{64} bytes.
Largest EM stack that has been imported into VAST so far: $\sim$ 5 Terabytes

Maximal number of labels supported: currently $2^{16} - 1$ (but needs lots of RAM; 600 bytes per label).

Largest image supported (at full resolution): $(2^{31} - 1) \cdot (2^{31} - 1)$

B.2 Supported file formats for importing / exporting

Importing of EM images

Currently 8 bit graylevel and 24-bit RGB image stacks are supported.
Stacks and tiled stacks: .JPG, .PNG, .TIF, .BMP
3D Volume files: .NII (NIfTI); will be converted to 8 bits when imported
Versions 1.1 and later of VAST Lite use `libtiff` for importing from .TIF images, which solves the previous problems caused by the Windows GDI+ TIFF routines.

Exporting of EM images

Stacks and tiled stacks: .PNG 8-bit indexed, .TIF uncompressed, .RAW

Importing of segmentations

Segmentations can be imported from RGB .TIF and .PNG image stacks. The segment number for each pixel is encoded in the RGB value of the image as follows: Bits 0-7 of the label number are expected in the blue channel, bits 8-15 in the green channel, and bits 16-23 in the red channel. This is the same format used for exporting segmentations to image stacks (see below). Please be aware that VAST can currently only handle 16-bit segmentations.

Exporting of segmentations

When exporting segmentations, the available file formats depend on the range of segment numbers used. For example, if the highest segment number is greater than 255, 8 bit indexed file formats

will not be available. In that case the label numbers will be encoded into the color channels (for example for RGB, bits 0-7 of the label number will be put into the blue channel, bits 8-15 into the green channel, and bits 16-23 into the red channel).

Stacks and tiled stacks:	.PNG, .TIF uncompressed, .RAW
3D Volume files:	.NII (NIfTI) 8 bit only (currently unsupported)

Exporting of screen shots

Stacks and tiled stacks:	.PNG, .TIF uncompressed, .RAW (all 24 bit RGB)
3D Volume files:	.NII (Nifti) 8 bit only (currently unsupported)

B.3 API Function Reference

VAST includes an API through which external programs can communicate with VAST. The communication is done through a TCP/IP connection. This makes it possible to write client programs in any programming language which supports the TCP/IP protocol. It also allows client programs to run on separate computers and communicate through the network.

VAST comes with a client-side implementation in Matlab, which is used by the VastTools toolbox and included in the `vast_package.zip` file (see section 2.3).

In Matlab the VAST API is implemented as a class, `VASTControlClass.m`. Internally it uses the `jtcp` library for TCP/IP communication in Matlab. The different API functions are implemented as class methods.

Using the VAST API is very simple. With `VASTControlClass.m` and its helper functions available (in the path), simply make an instance of the class and use its functions. Here is a short example which connects to VAST, reads out some basic information, and disconnects:

```
vast=VASTControlClass();
res=vast.connect('127.0.0.1',22081,1000);
if (res==0)
  warndlg('Connecting to VAST at 127.0.0.1, port 22081 failed.','Error');
else
  vinfo=vast.getinfo();
  disp(vinfo);
  vast.disconnect();
end;
```

You can also use VastTools to handle the connection. Simply run VastTools and connect, then run your own script. VastTools uses a global variable 'vdata'. To access the API functions, simply enable access to the global variable, then use the VAST API like so:

```
global vdata;
vinfo=vdata.vast.getinfo();
disp(vinfo);
```

The following API functions are currently available (Version 2, as returned by `getapiversion`):

res = connect(host, port, timeout)

Tries to connect to VAST via TCP/IP. 'host' is given as a text string, 'port' as a number and 'timeout' as a number (in milliseconds). Returns 1 if connected, 0 if connection failed.

res = disconnect()

Disconnects the TCP/IP connection to VAST. Always returns 1. If the request fails, a Java Error will stop the script, so a return of 0 is currently not possible.

errornumber = getlasterror()

Retrieves the error code of the last call of an API function. Error codes have the following meaning:

0	No error
1	Unknown error
2	Unexpected data received from VAST - API mismatch?
3	VAST received invalid data. Command ignored.
4	VAST internal data read failure
5	Internal VAST Error
6	Could not complete command because modifying the view in VAST is disabled
7	Could not complete command because modifying the segmentation in VAST is disabled
10	Coordinates out of bounds
11	Mip level out of bounds
12	Data size overflow
13	Data size mismatch - Specified coordinates and data block size do not match
20	Segment number out of bounds
21	No segmentation available
22	RLE overflow - RLE-encoding makes the data larger than raw; please request as raw
30	Layer number out of bounds
31	Invalid layer type
50	`sourcearray` and `targetarray` must have the same length

Table B.1: VAST API Error Codes as returned by getlasterror()

[apiversion, res] = getapiversion()

Returns the version of the API provided by the currently connected VAST executable. The API version described here is version 2.
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

[version, res] = getcontrolclassversion()

Returns the version of the `VASTControlClass.m` script (not part of the API). The current version is 2, introduced in VAST 1.02.

[info, res] = getinfo()

Reads out general information from VAST.
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.
Returns a struct with the following fields if successful, or an empty struct `[]` if failed:

info.datasizex	X (horizontal) size of the data volume in voxels at full resolution
info.datasizey	Y (vertical) size of the data volume in voxels at full resolution
info.datasizez	Z (number of slices) size of the data volume in voxels
info.voxelsizex	X size of one voxel (in nm)
info.voxelsizey	Y size of one voxel (in nm)
info.voxelsizez	Z size of one voxel (in nm)
info.cubesizex	X size of the internal cubes used in VAST in voxels; always 16
info.cubesizey	Y size of the internal cubes used in VAST in voxels; always 16
info.cubesizez	Z size of the internal cubes used in VAST in voxels; always 16
info.currentviewx	Current view X coord in VAST in voxels at full res (window center)
info.currentviewy	Current view Y coord in VAST in voxels at full res (window center)
info.currentviewz	Current view Z coord in VAST in voxels (slice number)
info.nrofmiplevels	Number of mip levels of the current data set

Table B.2: getinfo() struct

[nr, res] = getnumberofsegments()

Returns the number of segments of the active segmentation layer or [] if failed.
If the call succeeded, res will be 1, otherwise 0. Use getlasterror() to retrieve the error code.

[data, res] = getsegmentdata(id)

Reads out information of the segment with number id in the active segmentation layer. This is the same information that gets written to a text file by 'Save Segment Colors ...' in VAST (see section 3.4.12).
If the call succeeded, res will be 1, otherwise 0. Use getlasterror() to retrieve the error code.
Returns a struct with the following fields if successful, or an empty struct [] if failed:

data.id	ID of the requested segment
data.flags	Flags field of the segment as 32-bit value
data.col1	Primary color as 32-bit value
data.col2	Secondary color as 32-bit value
data.anchorpoint	XYZ coordinates of the segment's anchor point (in voxels)
data.hierarchy	IDs of parent, child, previous and next segment (0 if none)
data.collapsednr	If the segment is collapsed into a folder, this is the folder ID
data.boundingbox	Segment bounding box (may be incorrect if voxels were deleted)

Table B.3: getsegmentdata() struct

[name, res] = getsegmentname(id)

Returns the name of the segment with the given ID in the active segmentation layer. Returns [] if the ID is out of range.
If the call succeeded, res will be 1, otherwise 0. Use getlasterror() to retrieve the error code.

res = setanchorpoint(id, x, y, z)

Sets the anchor point of the segment with the given ID in the active segmentation layer to the given coordinates (in voxels at full resolution). Only non-negative values are allowed for x, y and z.

If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

res = setsegmentname(id, name)

Sets the name of the segment with the given ID in the active segmentation layer.
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

res = setsegmentcolor8(id, r1, g1, b1, p1, r2, g2, b2, p2)

Sets the primary and secondary colors (and pattern) of the segment with the given ID in the active segmentation layer. Values for r1, g1, b1, r2, g2 and b2 have to be between 0 and 255. p1 defines the pattern and has to be between 0 and 15. p2 is currently unused.
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

res = setsegmentcolor32(id, col1, col2)

Same as setsegmentcolor8, but the two colors for the segment are given as two 32-bit values.
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

[x, y, z, res] = getviewcoordinates()

Returns the current view coordinates in VAST in voxels at full resolution.
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

[zoom, res] = getviewzoom()

Returns the current zoom value in VAST.
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

res = setviewcoordinates(x, y, z)

Sets the current x, y, z coordinates (in voxels at full resolution) of the view (center of window) in VAST.
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

res = setviewzoom(zoom)

Sets the current zoom value. Zoom values are currently integer values and can be negative. The higher the value, the more magnified the view. A zoom value of 0 sets the pixel size of the full resolution image to exactly the 'Target Resolution Smaller Than' value specified in the VAST Preferences. A value of -8 zooms out by a factor 2 (and shows the next lower mip level texture).
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

[nroflayers, res] = getnroflayers()

Returns the number of layers currently loaded in VAST.
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

[linfo, res] = getlayerinfo(layernr)

Returns information about the layer with the given number. layernr can be a number between 0 and getnroflayers-1. If the function succeeds, it will return a struct `layerinfo` with fields as described below and res will be 1. If it fails it will return an empty struct and res will be 0. This returns the values which are also visible in the VAST 'Layers' tool window.
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

`linfo.type`	0: EM layer (VSV); 1: Segmentation layer; 3: remote EM layer
`linfo.editable`	1 if the layer is editable, 0 else
`linfo.visible`	1 if the layer is visible, 0 else ('Visible' checkbox)
`linfo.brightness`	'Bright' (brightness) or 'Pattern' checkbox
`linfo.contrast`	'Contrast' or 'Sel Alpha' checkbox
`linfo.opacitylevel`	Slider next to 'Visible', which defines layer opacity
`linfo.brightnesslevel`	Slider next to 'Bright' or 'Pattern'
`linfo.contrastlevel`	Slider next to 'Contrast' or 'Sel Alpha'
`linfo.blendmode`	The current blend mode (see below)
`linfo.blendoradd`	0: alpha blending, 1: additive blending
`linfo.tintcolor`	Tint color as 32-bit RGBA value. Default: white (`0xffffffff`)
`linfo.name`	Name of the layer
`linfo.redtargetcolor`	Red channel target color (RGBA). Default: red (`0xff0000ff`)
`linfo.greentargetcolor`	Green channel target color (RGBA). Default: green (`0x00ff00ff`)
`linfo.bluetargetcolor`	Blue channel target color (RGBA). Default: blue (`0x0000ffff`)

Table B.4: getlayerinfo() struct

Blend modes are: 0: no transparency ('Flat'), 1: the darker `mean(r,g,b)` the more transparent, 2: the brighter `mean(r,g,b)` the more transparent, 3: the darker `max(r,g,b)`, the more transparent, 4: the brighter `max(r,g,b)`, the more transparent.

[segdata, res] = getallsegmentdata()

This function is similar to `data = getsegmentdata(id)` above, but retrieves the data for all segments in the active segmentation layer at once and returns a cell array of structs instead of a single struct.
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

[segdatamatrix, res] = getallsegmentdatamatrix()

Same as `data = getallsegmentdata(id)` above, but returns the data as a matrix with one row per segment and one column per data value. The columns are the same as when exporting segment metadata to a file (see section 3.4.12), with the exception of the segment names.
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

[segname, res] = getallsegmentnames()

This function is similar to `name = getsegmentname(id)` above, but retrieves the names of all segments in the active segmentation layer at once and returns them in a cell array. If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

res = setselectedsegmentnr(segmentnr)

Sets the selected segment in the active segmentation layer in VAST to the segment with the given ID number. If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

[selectedsegmentnr, res] = getselectedsegmentnr()

Returns the ID number of the currently selected segment in the active segmentation layer, or -1 if an error occurred. If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

res = setselectedlayernr(obj)

Sets the selected layer in VAST to the layer with the given ID number (counting from 0 as the first layer in the list). If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

[selectedlayernr, selectedemlayernr, selectedsegmentlayernr, res] = getselectedlayernr()

Returns the numbers of the currently selected layer, EM layer, and segment layer, or -1 if there is no such layer. The 'selected layer' will always be either the 'selected EM layer' or the 'selected segment layer', whichever is actually highlighted in the 'Layers' tool window in VAST. The other number indicates the most recently selected layer of the other type.
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

[segimage, res] = getsegimageraw(miplevel, minx, maxx, miny, maxy, minz, maxz, flipflag)

Reads out the segmentation as a voxel image for a given mip level (resolution) and area, as defined by minimum and maximum values for the ranges in x, y and z. This function transmits the segmentation image as raw data (a one-dimensional array of bytes); consider using `getsegimageRLEdecoded()` below for a version with faster transmission. If `flipflag` is 1, X and Y axis will be swapped to make the image appear the same as in VAST. If `flipflag` is unspecified or 0, segimage will appear with X and Y axis swapped.
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

minx, maxx, miny, maxy, minz and maxz are given in voxels *at the resolution defined by* `miplevel`. You can retrieve the size of the volume at full resolution (mip level 0) using the function getinfo(). To compute the size of the volume in X and Y at a different mip level `miplevel`, use the following code:

```
xmin=bitshift(xmin,-miplevel);
xmax=bitshift(xmax,-miplevel)-1;
ymin=bitshift(ymin,-miplevel);
ymax=bitshift(ymax,-miplevel)-1;
```

Note that the size of the volume in Z does not change with mip level (this is the number of slices).

[segimageRLE, res] = getsegimageRLE(miplevel, minx, maxx, miny, maxy, minz, maxz, surfonlyflag)

Reads the requested volume region and returns it as an runlength-encoded (RLE) string. This function is useful if you want to directly process the RLE string; otherwise use one of the functions below.
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.
See `getsegimage()` above for information on volume coordinates.

The runlength-encoded image is an array of pairs of unsigned 16 bit values. The first value in a pair is the segment ID which follows, and the second value is the number of voxels with that ID to follow. The encoding order of voxels in the volume is row by row (X from left to right) concatenated for one plane (Y from top to bottom), and planes are concatenated from the first to the last slice of the volume. Please note that sequences of the same value defined by one pair can loop around at the end of a row and/or extend into the next plane. Run lengths longer than $2^{16} - 1$ (65535) are encoded in several pairs.

[segimage, res] = getsegimageRLEdecoded(miplevel, minx, maxx, miny, maxy, minz, maxz, surfonlyflag, flipflag)

Same as `getsegimage()` above, except that the data is transmitted run-length encoded (RLE) from VAST to Matlab, and then decoded on the Matlab side. This can improve speed because it reduces the size of the transmitted data. If `surfonlyflag` is 1, all interior voxels (any nonzero voxels for which all six direct neighbors have the same ID) are set to 0. If `flipflag` is 1, X and Y axis will be swapped to make the image appear the same as in VAST. If `flipflag` is unspecified or 0, segimage will appear with X and Y axis swapped. See `getsegimageraw()` above for information on volume coordinates.
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

[values, numbers, res] = getRLEcountunique(miplevel, minx, maxx, miny, maxy, minz, maxz, surfonlyflag)

This function returns the segment IDs (values) and number of voxels of each ID (numbers) in the requested subvolume. See `getsegimageraw()` above for information on volume coordinates.
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

[segimage, values, numbers, res] = getsegimageRLEdecodedcountunique(miplevel, minx, maxx, miny, maxy, minz, maxz, surfonlyflag, flipflag)

Combines the functions `getsegimageRLEdecoded()` and `getRLEcountunique()` above, and returns both the decoded subvolume and a list of segment IDs in the volume and number of voxels for each ID. If `flipflag` is 1, X and Y axis will be swapped to make the image appear the same as in VAST. If `flipflag` is unspecified or 0, segimage will appear with X and Y axis swapped. See `getsegimageraw()` above for information on volume coordinates.

If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

[segimage, values, numbers, bboxes, res] = getsegimageRLEdecodedbboxes(miplevel, minx, maxx, miny, maxy, minz, maxz, surfonlyflag, flipflag)

Same as `getsegimageRLEdecodedcountunique()` above, but also returns the bounding boxes for all IDs. The coordinate order for the bounding boxes in `bboxes` is: `[minx,miny,minz,maxx,maxy,maxz]`, one row per segment.[1] `bboxes` is not changed by `flipflag`. See `getsegimageraw()` above for information on volume coordinates.
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

res = setsegtranslation(sourcearray, targetarray)

Tells VAST how to translate the segmentation volume before transmitting it through `getsegimage*` and `getRLE*` functions. `sourcearray` and `targetarray` are arrays of unsigned (positive) 16-bit values and have to have the same length. VAST will convert all voxels with IDs in `sourcearray` to the corresponding values in `targetarray`. Voxels with other IDs will be removed (set to 0). To disable the segmentation translation, call `setsegtranslation([],[])`.
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

[emimage, res] = getemimageraw(layernr, miplevel, minx, maxx, miny, maxy, minz, maxz)

Reads out the EM image of the specified layer as a voxel image for a given mip level (resolution) and area. The area is defined by minimum and maximum values for the ranges in x, y and z. This function transmits the segmentation image as raw data (a one-dimensional array of bytes) with either one or three bytes per voxel, depending on whether the layer has one or three color channels. Use `getemimage` to request a correctly reshaped two-, three- or four-dimensional image.
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

[emimage,res] = getemimage(layernr, miplevel, minx, maxx, miny, maxy, minz, maxz)

Same as `getemimageraw`, but reshapes the one-dimensional array to a matrix of the requested dimensions. The order of dimensions in the matrix is Y,X,Z,C (C for color, R G B, if the layer has three color channels).
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

[screenshotimage,res] = getscreenshotimageraw(miplevel, minx, maxx, miny, maxy, minz, maxz, collapseseg)

Reads out the EM and segmentation stack as currently displayed in VAST as a voxel image for a given mip level (resolution) and area. The area is defined by minimum and maximum values for the ranges in x, y and z. If `collapseseg` is 1, the color of the segmentation will appear as displayed in VAST, if it is 0, each segment will be colored in its native (uncollapsed) color. This function transmits the segmentation image as raw data (a one-dimensional array of bytes). Use `getscreenshotimage` to request a correctly reshaped two-, three- or four-dimensional image. Screenshot images are always returned in RGB format (three bytes per voxel).

[1] Sorry that the order is different than in most other places.

If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

[screenshotimage,res] = getscreenshotimage(miplevel, minx, maxx, miny, maxy, minz, maxz, collapseseg)

Same as `getscreenshotimageraw`, but reshapes the one-dimensional array to a matrix of the requested dimensions. The order of dimensions in the matrix is Y,X,Z,C (C for color, R G B).
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code.

res = setsegimageraw(miplevel, minx, maxx, miny, maxy, minz, maxz, segimage)

Writes the (2D or 3D) image in `segimage` to the active segmentation layer in VAST, at the mip level `miplevel` and target coordinates given in minx, maxx, miny, maxy, minz, maxz. These coordinates have to be specified at the resolution of `miplevel` (see explanation in section `getsegimageraw` above).
VAST will not accept segmentation images which contain segment numbers which exceed the segment list. For existing segments, the segment metadata will be updated to include the imported segmentation image.
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code. Introduced in API version 2.

res = setsegimageRLE(miplevel, minx, maxx, miny, maxy, minz, maxz, segimage)

Same as `setsegimageraw`, but this version encodes the segmentation image using RLE and transmits the RLE-encoded data to VAST. This reduces the amount of data that has to be transmitted, but RLE encoding in Matlab is slow, so `setsegimageraw` may be the faster alternative for Matlab.
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code. Introduced in API version 2.

res = setsegmentbbox(id, minx, maxx, miny, maxy, minz, maxz)

Sets the bounding box of the segment with number `id` in the active segmentation layer.
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code. Introduced in API version 2.

[firstsegmentnr, res]=getfirstsegmentnr()

Returns the segment number (id) of the first segment (after Background) in the active segmentation layer.
If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code. Introduced in API version 2.

[info, res] = gethardwareinfo()

Returns the hardware properties of the computer on which VAST is running in the struct `info`. The fields of the struct are as follows:

info.computername	text	Name of the computer
info.processorname	text	Name of the processor
info.processorspeed_ghz	double	Clock speed of the processor in GHz
info.nrofprocessorcores	integer	Number of processor cores
info.tickspeedmhz	double	Tick speed of the VAST computer in MHz
info.mmxssecapabilities	text	Text describing the processor capabilities
info.totalmemorygb	double	Memory installed, in GB
info.freememorygb	double	Free memory, in GB
info.graphicscardname	text	Graphics card name
info.graphicsdedicatedvideomemgb	double	Dedicated video memory in GB
info.graphicsdedicatedsysmemgb	double	System memory dedicated for graphics, in GB
info.graphicssharedsysmemgb	double	System memory shared for graphics, in GB
info.graphicsrasterizerused	text	'Undefined', 'Hardware', 'Warp' or 'Reference'

Table B.5: gethardwareinfo() struct

If the call succeeded, `res` will be 1, otherwise 0. Use `getlasterror()` to retrieve the error code. Introduced in API version 2.

Bibliography

[1] Cardona A., Saalfeld S., Schindelin J., Arganda-Carreras I., Preibisch S., et al.: *TrakEM2 Software for Neural Circuit Reconstruction*, PLoS ONE, **7**(6):e38011, (2012), doi:10.1371/journal.pone.0038011.

[2] Saalfeld, S., Fetter, R., Cardona, A., and Tomancak, P.: *Elastic volume reconstruction from series of ultra-thin microscopy sections*, Nature Methods, **9**(7), (2012), 717-720.

[3] Srubek Tomassy, G., Berger, D., Chen, H., Kasthuri, N., Hayworth, K., Vercelli, A., Seung, S., Lichtman, J., and Arlotta, P.: *Distinct Profiles of Myelin Distribution Along Single Axons of Pyramidal Neurons in the Neocortex*, Science, **344**(6181), (2014) 319-324, doi:10.1126/science.1249766.

[4] Kasthuri, N., Hayworth, K., Berger, D., Schalek, R., Conchello, J., Knowles-Barley, S., Lee, D., Vazquez-Reina, A., Kaynig, V., Jones, T., Roberts, M., Morgan, J., Tapia, J., Seung, H.S., Gray Roncal, W., Vogelstein, J., Burns, R., Sussman, D., Priebe, C., Pfister, H., and Lichtman J.: *Saturated Reconstruction Of A Volume Of Neocortex*, Cell, **162**(3), (2015) 648-661.

[5] Morgan, J., Berger, D., and Lichtman J.: *The Fuzzy Logic of Network Connectivity in Mouse Visual Thalamus*, Cell, **165**, (2016) 192-206.

[6] Quadrato, G., Nguyen, T., Macosko, E., Sherwood, J., Yang, S., Berger, D., Maria, N., Scholvin, J., Goldman, M., Kinney, J., Boyden, E., Lichtman, J., Williams, Z., McCarroll, S., and Arlotta, P.: *Cell diversity and network dynamics in photosensitive human brain organoids*, Nature, **545**, (2017), 48-53, doi:10.1038/nature22047.