Region of Interest Drawing and Usage in AFNI

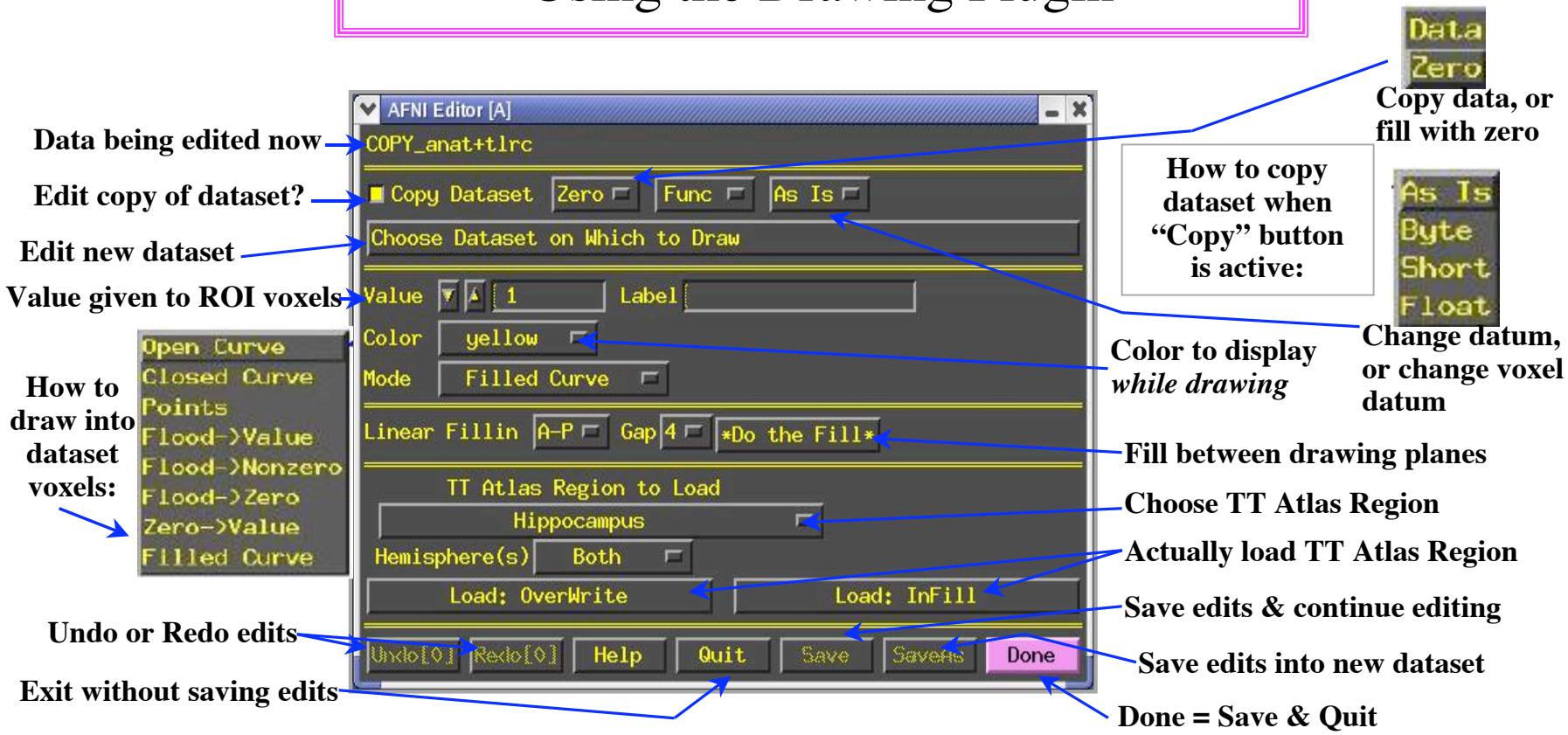
- Method 1: Manually select Regions of Interest (ROIs) based on anatomical structures, then analyze functional datasets within these regions
  - ★ E.g., Someone doing a study on emotion may want to draw an ROI mask that includes the amygdala only in order to analyze the voxels that fall within that brain region
  - ★ This method relies on 'a priori' assumptions about localization of brain function
- Method 2: Analyze functional dataset for *entire* brain first, then focus on geometrically connected clusters of 'activity' (supra-threshold voxels in some functional statistical map)
  - ★ i.e., analyze the entire brain first and *then* pinpoint interesting areas of activity and do further analyses on those areas (**`3dclust`** or **`3dmerge`** can assist you in finding those larger "blobs" or clusters of activity)
    - ➥ Even with this method, you might want to manually adjust the ROIs (i.e., add or delete some voxels in the ROI mask)
- Method 3: Use atlases to select your ROIs regions.
  - ★ By using whereami program or symbolic notation to create masks on the command line.
- ROIs are stored as regular AFNI datasets (i.e., with a .HEAD and .BRIK file); it is only the user (you) that decides whether a particular dataset is a ROI mask
  - ★ Nonzero voxels are "in" the mask; zero voxels are "outside"

Method 1- Handmade ROI's from Anatomical Volumes

- **Quick outline of procedure:**
  1. On the main AFNI control panel, set the anatomical underlay dataset (with **[UnderLay]**) to be what you want to draw on -- usually a SPGR or MP-RAGE type of dataset
     - i.e., the anatomical underlay will serve as our guide or template for drawing the ROI mask
  2. Start the **Draw Dataset** plugin (this is our ROI-creating plugin):
     - **[Define Datamode]** → **[Plugins]** → **[Draw Dataset]**
  3. Create an *all zero* anatomical overlay dataset with the **Draw Dataset** plugin. This is the beginning of our ROI mask. At this point, the anatomical overlay is empty - i.e., all the voxel values are zero
     - This anatomical overlay *must* have the same geometry as the anatomical underlay, i.e., it has the same voxel size as the underlay, the same *xyz*-grid spacing, etc…, since drawing/editing is done on a voxel-by-voxel basis
       - Think of the anat overlay as a blank piece of tracing paper that is the same size as the template underneath. The blank overlay will be used to trace portions of the underlay. Voxels *inside* the traced portion will make up the ROI mask. Values *outside* the traced region are irrelevant (zeros)
     - Note: You could also edit an already-existing ROI mask (that you created earlier) at this step

4. To view and begin drawing an ROI mask (or several ROIs) on this blank anatomical overlay dataset, go to the main AFNI interface and **[Switch Overlay]** to be the empty anatomical dataset you will be editing. Also turn the overlay ON with **[See OverLay]**
5. Start drawing the ROI mask into this blank anatomical overlay dataset. Voxels inside the ROI mask will receive a non-zero value (you decide what value to give them). Values outside the ROI mask will remain zero
   - ➥ Be sure to save the results by pressing **[Save]**, **[SaveAs]** or **[DONE]** in the ROI plugin GUI (**[Quit]** will exit the ROI plugin without saving your work)
6. Convert the anatomical-resolution ROI dataset into a dataset at the resolution of the functional (statistical) datasets you want to analyze with the ROI
   - ➥ Note: The ROI and functional datasets may already be at the same resolution, if you are operating in `+tlrc` coordinates
   - ➥ Resolution conversion of masks is done with program **3dfractionize**
7. Use programs **3dmaskave**, **3dmaskdump**, and **3dROIstats** to extract ROI-based information about functional datasets
   - ➥ Also can use the **ROI Average** plugin to extract interactively the average of a dataset over a ROI (does the same thing as 3dmaskave)

Using the Drawing Plugin



- **Critical things to remember:**
  - ★ You should have [See OverLay] turned on, and be viewing the same overlay dataset in AFNI as you are editing
    - ➥ Otherwise, you won't see anything when you edit!
  - ★ When drawing, you are putting numbers into a dataset brick
    - ➥ These numbers are written to disk only when you do [Save], [SaveAs] or [Done]; before then, you can [Quit] (or exit AFNI) to get the unedited dataset back

Keys 'o' and 'u' and scroll wheel come in handy here

- **Step 1: Load a dataset to be edited (for ROI creation):**
  - ★ [**Choose Dataset**] button gives you a list of datasets that
    (a) Actually have brick data with only one sub-brick;
    (b) Are at the same voxel dimension, grid size, etc., as current anat underlay
  - ★ When you are starting, you probably don't want to edit an existing dataset -- i.e., you don't want to write on the underlay itself; you just want to use it as a template and draw on a blank overlay that shares the same geometry as that existing underlay dataset
    - ➥ To do this, you must create an all-zero *copy* of the anatomical underlay (by "copy" we mean the all-zero dataset shares the same geometry as the underlay, not the same voxel data values)
    - ➥ To create an all-zero copy, click the [**Copy**] button on (from the **Draw Dataset** plugin GUI) and set the controls to its right to [**Zero**] and [**As Is**] (the **[Func]** button is a relic of the program and is now obsolete -- ignore)
      - ➪ [**Data**] would make a copy of the underlay dataset with the actual voxel data values. [**Zero**] copies the geometry of underlay, but gives each voxel a data value of zero (this latter option is usually want you want when starting out)

- [As Is] keeps the voxel values in the copy as the same type as in the original underlay; you can also change the voxel values to be stored as:
  - [Byte] (integer values: 0..255) ⇒ 1 byte each
  - [Short] (integer values: -32767…32767) ⇒ 2 bytes each
  - [Float] (fractional values) ⇒ 4 bytes each
  - Bytes and Shorts make the most sense for ROI masks, where you are essentially attaching labels to voxels
- Click on [Choose Dataset], select the dataset you want a copy of (e.g., the anatomical underlay), and then press [Set]

- **Step 2: Drawing the ROI (or ROIs):**
  - *Choose the value to draw into the anatomical overlay dataset* (recall that all values in the copied/blank overlay dataset are zero at this point)
    - If you drawing only one ROI, then the default value of 1 is good
    - If you are drawing multiple ROIs, then you should choose a different numerical value for each so that they can be distinguished later
      - Pencil and paper are our friends -- write down which number corresponds with which ROI for later recall!
  - You could use ROI color maps:
    - Choose the '**' map. Set 'pos' on.
    - Right click on colormap --> Choose Colorscale --> ROI_32
    - Set 'autoRange' off. Set range to 32
  - *Choose the drawing color*
    - This is the color that is shown *while you are drawing*
    - Color choice is only important to give a good contrast with underlay, so don't obsess too much over this

➥ After you finish a drawing motion, the voxels you drew will be filled with the drawing value, the image will be redisplayed, and the colors will be determined by the **[Define OverLay]** control panel

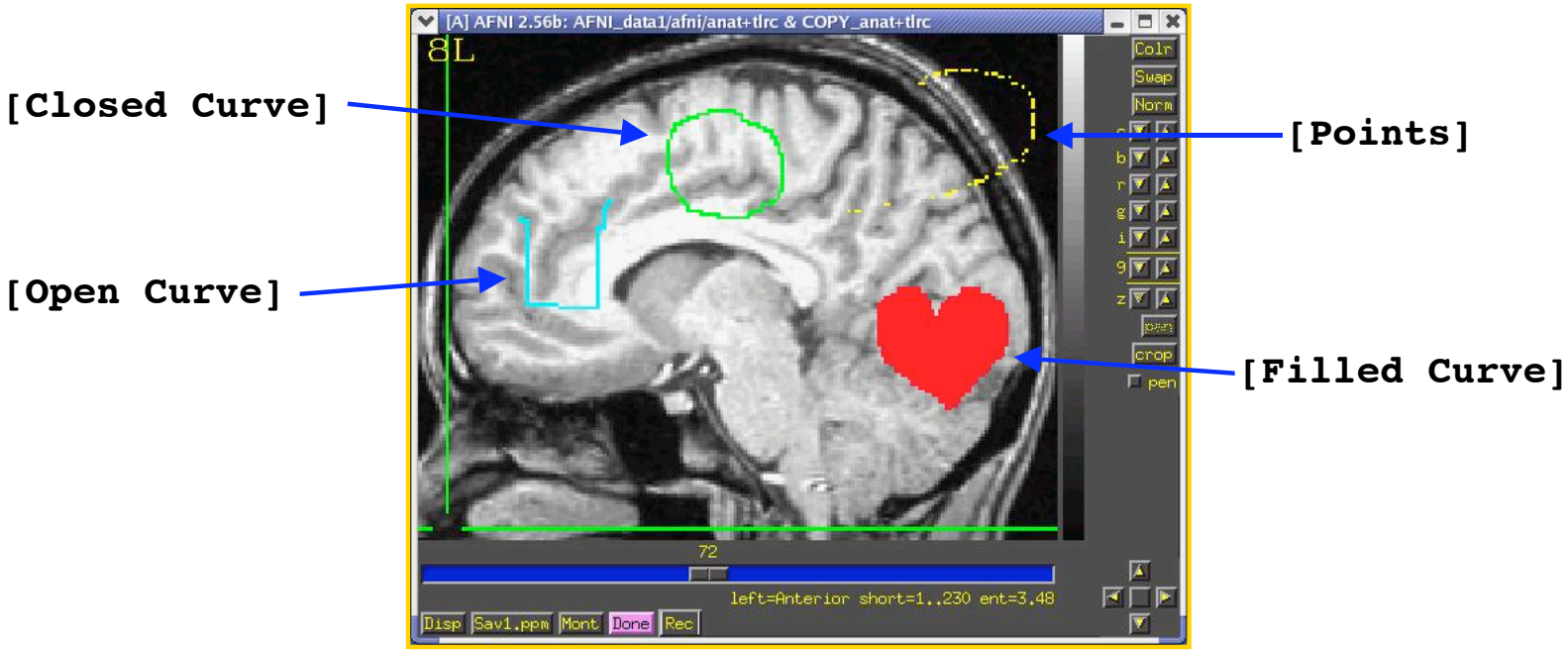
★ *Choose the drawing mode*

➥ **[Filled Curve]**
Drawing action produces a continuous closed-ended curve (default setting)

➥ **[Open Curve]**
Drawing action produces a continuous open-ended curve

➥ **[Closed Curve]**
Drawing action produces a continuous closed-ended curce

➥ **[Points]**
Only points actually drawn over are filled (used to “touch up” and ROI)



- ➥ [**Flood→Value**]

  Flood fills space outward from the drawing point, stopping when the flood hits the current drawing value (used to fill a closed curve)

- ➥ [**Flood Nonzero**]

  Drawing action produces a continuous closed-ended curve (filled inside)

- ➥ [**Zero→Value**]

  Floods voxels with zero until the flood hits nonzero voxels (you can also do this more easily with Filled Curve, value=0)

- ➥ [**Flood→Nonzero**]

  Flood fills outwards from drawn point, stopping when the flood hits any nonzero voxel (used to fill between regions):

★ Important Note:

- ➥ A ROI is defined by the values stored in voxels of the ‘mask’ dataset
- ➥ Contiguity of voxels has no meaning to the ROI software described below
- ➥ Two voxels are in the same ROI if they have the same value in the mask dataset (i.e., it doesn’t matter where they are located in the volume)

- ★ *Actually draw something*
  - ➥ Drawing is done with mouse Button 2 (“middle” button) in 2D slice image
  - ➥ Hold the button down in the image window during a single drawing action
  - ➥ While the drawing action is happening, the chosen drawing color will trace the screen pixels you draw over
  - ➥ When you release the button, these pixels are converted to voxels, and the dataset is actually changed, using the drawing value and drawing mode you selected
  - ➥ At this point, the color of the drawn region will change to reflect the drawing value and the setup of the **[Define OverLay]** control panel
- ★ **[Undo]** button will let you take back the last drawing action (you can go “undo” many levels back, i.e., multiple undo function)
- ★ You can draw on one 2D slice image at a time
  - ➥ If you draw on a montage display, only screen pixels overlaying the first image you Button 2 click in will count
  - ➥ While drawing, if you cross over between sub-images in the montage, unexpected effects will result
    - ➪ But there is always **[Undo]** to the rescue!

- **Step 3: Save your results:**
  - ★ **[Save]** will write the current dataset values to disk (overwriting any existing .BRIK file, i.e., if you had edited this ROI earlier, the new changes would overwrite the old file)
    - ➥ You could also then choose another dataset to edit
  - ★ **[Save As]** will let you write the current dataset to disk under a new name, creating a new dataset, then continue editing the new dataset
  - ★ **[Quit]** exits editing and closes the plugin window, without saving to disk any changes since the last [Save]
    - ➥ Exiting AFNI has the same effect
  - ★ **[Done]** is equivalent to **[Save]** then **[Quit]**
- **Optional Drawing Steps:**
  - ★ **[Linear Fillin]** lets you draw a 3D ROI not in every slice, but in every third slice (say), and then go back and fill in the gaps
    - ➥ For example, if you draw in coronal slices, then you want to fill in the **[A-P]** direction (the default)
    - ➥ If you draw every nth slice, then you want to set the Gap to n-1
    - ➥ Line segments of voxels in the fillin direction that have a current drawing value at each end, and have no more than [Gap] zero voxels in between, will get their gap voxels filled with the drawing value
      - ➪ After you try this, you will probably have to touch up the dataset manually

- ➥ This operation can also be done with program **3dRowFillin**, which creates a new dataset

★ [**TT Atlas Region to Load**] lets you load regions from the Talairach Daemon database into the dataset voxels

- ➥ Requires that you be drawing in **+tlrc** coordinates, or at least have a transformation from **+orig** → **+tlrc** computed in the current directory
- ➥ Choose a region to draw into the dataset (e.g., Hippocampus)
- ➥ [**Load: Overwrite**] will fill all voxels in the region with the drawing value
- ➥ [**Load: Infill**] will fill only voxels in the region that are currently zero
  - ➪ You probably want to edit the results manually to fit the specific subject

- **Drawing and Volume Rendering at the Same Time** (totally fun, and maybe useful)

★ You cannot draw into the rendering plugin, but you can use it to see in 3D what you are drawing in 2D

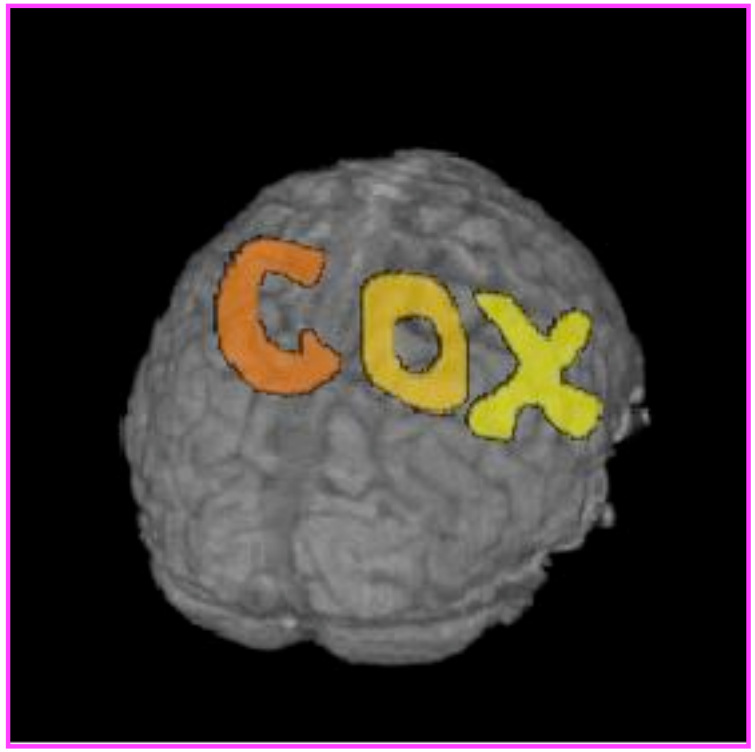
- ➥ If you meet the criteria for rendering (usually in `+tlrc` coordinates)

★ How to set up the renderer:

- ➥ Choose the underlay to be the current anatomical dataset (or a "scalped" version, from **3dSkullStrip** or **3dIntracranial**)
- ➥ Choose the overlay dataset to be the dataset you are editing
- ➥ Turn on [**See Overlay**]
- ➥ Set [**Color Opacity**] to [**ShowThru**] (or **ST+Dcue**)

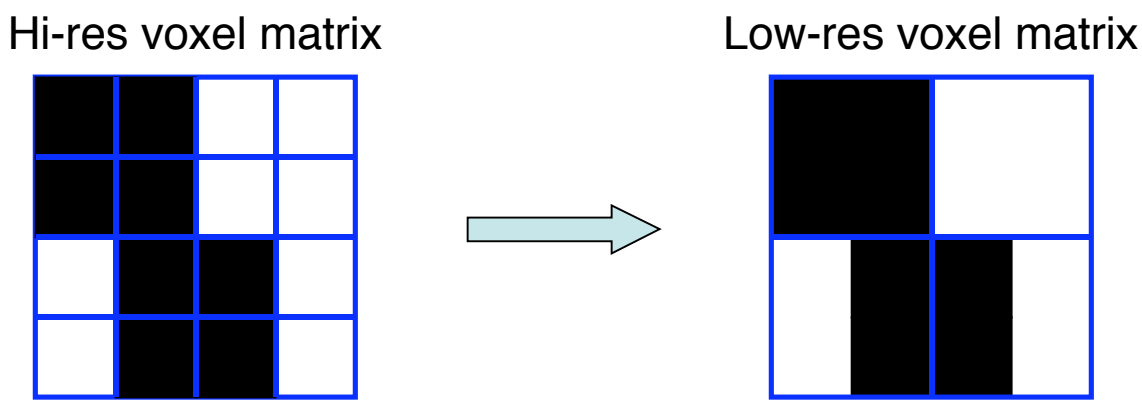
- Turn on [DynaDraw]
- Drawing in a 2D image window immediately triggers a redraw in the rendering window

  (if the 2D and 3D overlay datasets are the same)
- This is only useful if your computer is fast enough to render quickly (<1 sec per frame)



Things to Do with ROI Datasets
(no matter how you create them)

- ROIs are used on a voxel-by-voxel basis to select parts of datasets (usually functional datasets)
- If you draw at the anatomical resolution and want to use the ROI dataset at the functional resolution, you probably want to convert the high-resolution ROI dataset to a low-resolution dataset (unless you're working in `+tlrc` coordinates)
  - ★ E.g., hi-res anatomical ROI resampled to low-res functional dataset:



  - ★ Each voxel inside the ROI is given a nonzero value (e.g., 4; values outside the ROI are zeros. When the resolution is changed, what do you do with a voxel that's only partially filled by the ROI?

- **3dfractionize** does this resolution conversion:

```
3dfractionize  -template low_res_dset+orig          \
               -input ROI_high_res+orig             \
               -clip 0.5 -preserve -prefix ROI_low_res
```

  - ⋆ **-template** → The destination grid you want your ROI grid to be resampled to (we're going from high to low resolution here). Our output dataset **ROI_low_res+orig** will be written at the resolution of **func+orig**
  - ⋆ **-input** → Defines the input high-resolution dataset (that needs to be converted from high resolution to low resolution)
  - ⋆ **-clip 0.5** → Output voxels will only get a nonzero value if they are at least 50% filled by nonzero input voxels (you decide the percentage here). E.g., when going from high to low res, keep a label a voxel as part of the ROI if it is filled with at least 50% (or more) of the voxel value. For example:

This voxel is 80% filled with the ROI value -- keep it

This voxel is 30% filled with the ROI value -- lose it

★ **-preserve** → once it has been determined that the output voxel will be part of the ROI, preserve the original ROI value of that voxel (and not some fraction of that value). For example, if our ROI mask has values of "4":

This voxel is 80% filled with the ROI value -- keep it.

Without the -preserve option, this voxel would be given a value of "3.2" (i.e., 80% of "4").

With -preserve, it is labeled as "4"

- **3dresample** does conversion too but you have less controls for handling partial overlaps:

```
3dresample -master low_res_dset+orig      \
           -prefix ROI_low_res            \
           -inset ROI_high_res+orig       \
           -rmode NN
```

★ **-master**: the destination grid we want our ROI mask resampled to

★ **-prefix**: The output from **3dresample** -- in this example, a low resolution ROI mask that corresponds with the voxel resolution of our master dataset

★ **-inset**: The ROI mask dataset that is being resampled from high to low resolution

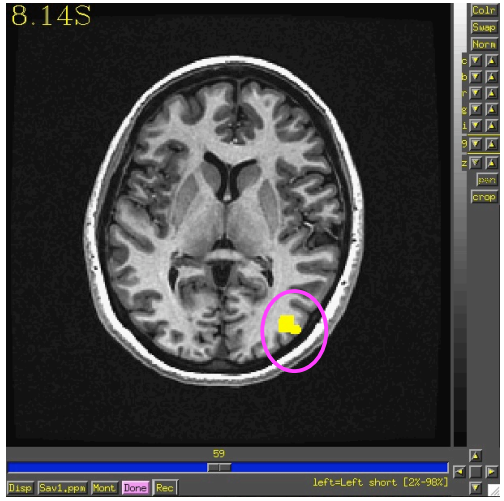
★ **-rmode NN**: If a voxel's "neighbor" is included in the ROI mask, include the voxel in question as well

- Let's do a class example of **3dresample**:

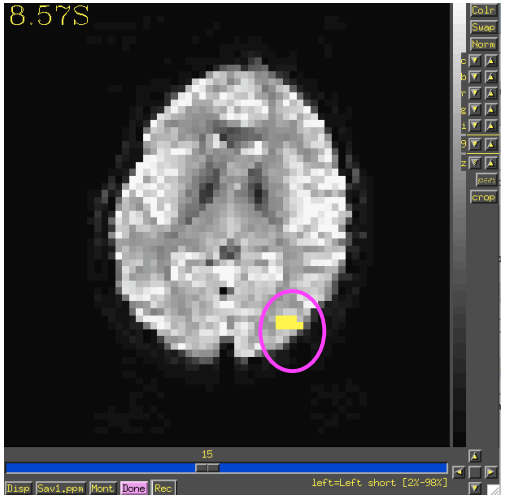
```
cd AFNI_data3/roi_demo

3dresample -master epi_9runs+orig      \
           -prefix anat_roi_resam      \
           -inset anat_roi+orig        \
           -rmode NN
```

★ In this class example, we want to take our ROI mask, which has a high voxel resolution of 0.9 x 0.9 x 1.2 mm, and resample to it the lower resolution of the time-series dataset, **epi_9runs+orig** (3.75 x 3.75 x 3.5mm).



Before, overlay ROI is:
anat_roi+orig
0.9x0.9x1.2 mm voxel grid

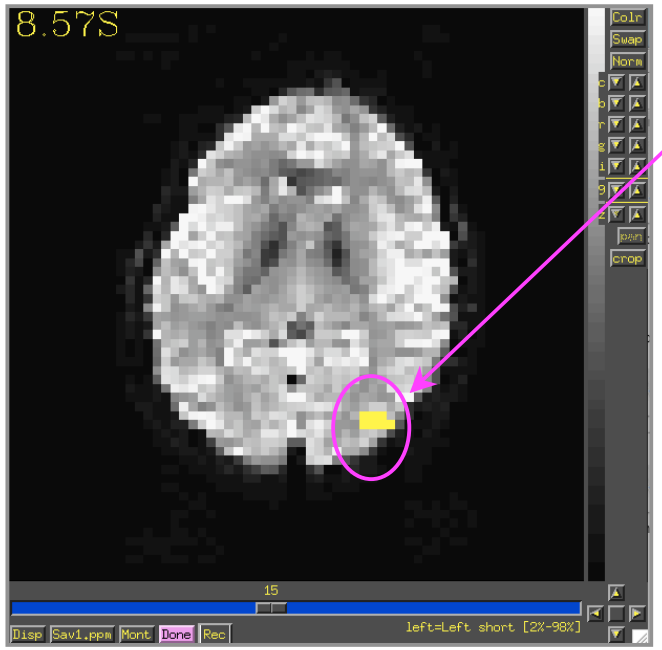


After, overlay ROI is:
anat_roi_resam+orig
3.75x3.75x3.5 voxel grid

- **3dmaskave**
  - ★ This ROI program computes the average of voxels (usu.from a functional or time-series dataset), that are selected from an ROI mask
    - ➥ (interactive version: **ROI Average** plugin)

  - ★ Class Example:

```
3dmaskave -mask anat_roi_resam+orig -q \
          epi_9runs+orig > epi_avg.1D
```



The above command takes the voxels that fall within the ROI mask, and computes a mean for every time point/volume.

In this example, there are 576 time-points in this dataset (9 runs x 64 volumes per run = 576), so the output will be a column of 576 means.

-q : Suppresses the voxel-count output (e.g., "[7 voxels] make up the ROI mask") from appearing next to each mean.

Alternatively, instead of having the results of **3dmaskave** spewed into the shell, you can redirect (**>**) the results into a text file (**epi_avg.1D**) and save them for later use.

★ Output will look like this (576 means in the column):

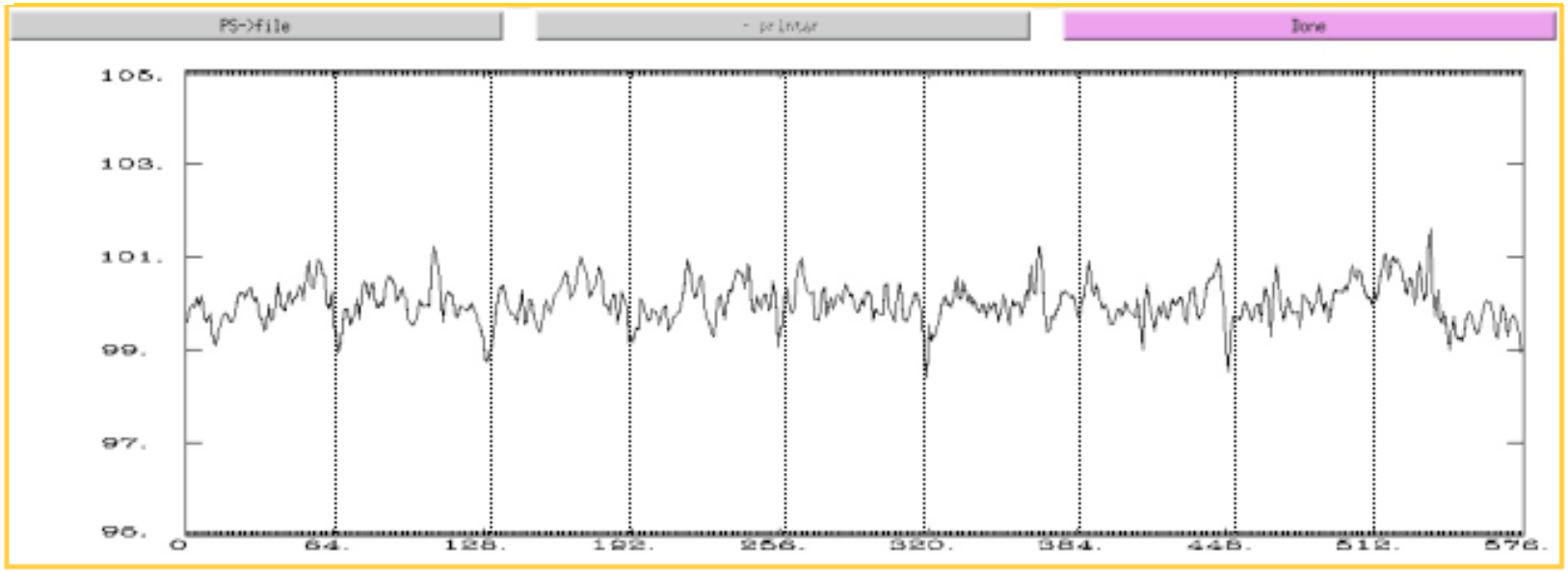
```
more epi_avg.1D
```

```
99.5828
99.606
99.9108
99.9582
...
98.9149
```

★ Data can also be plotted out using **`1dplot`**:

```
1dploto epi_avg.1D or…
1dplot -yaxis 95:105:5:1 -xaxis 0:576:9:64 epi_avg.1D
```

Mean voxel intensity for voxels falling within the ROI mask (at each timepoint).

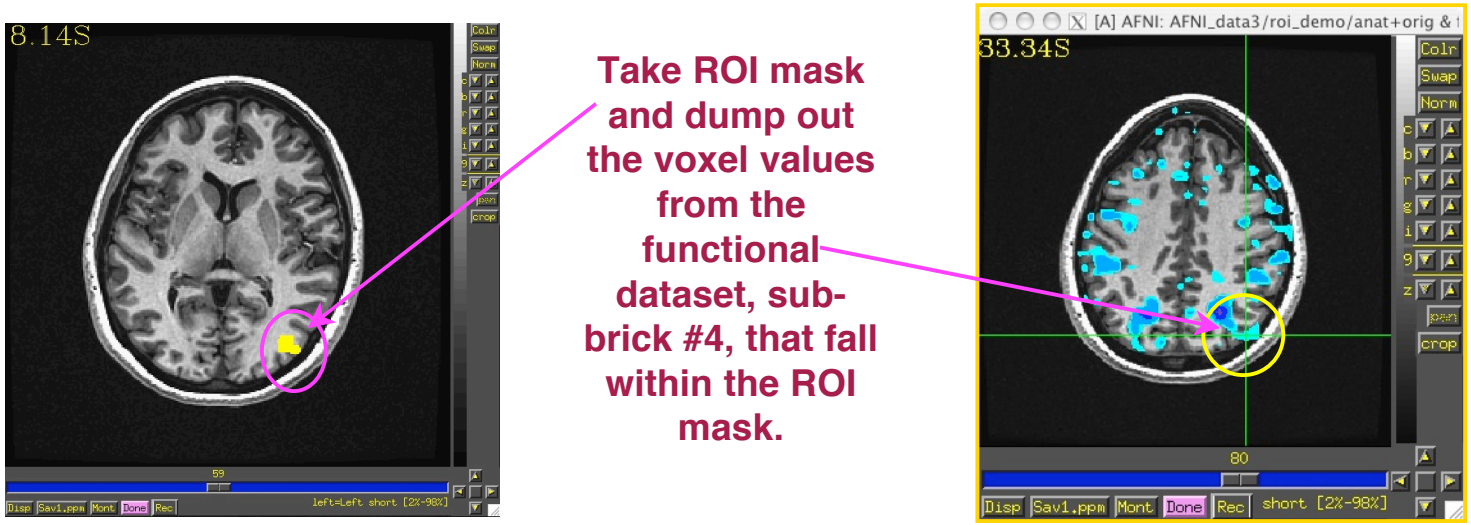


9 concatenated EPI runs, Timepoints (0-575)

- **3dmaskdump**
  - ★ Program that dumps out all voxel values in a dataset that fall within the ROI of a given mask dataset
  - ★ Class example:

```
3dmaskdump -noijk -mask anat_roi_resam+orig          \
           'func_slim+orig[4]' > FaceNeg-tstats.txt
```

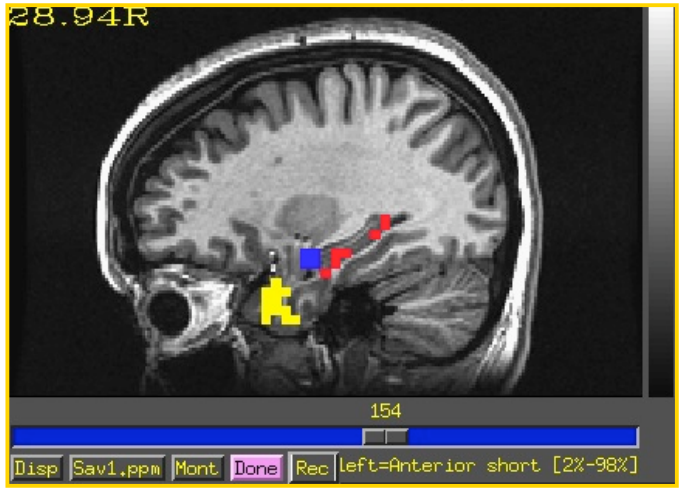
➥ The output appears in the shell (unless you redirect it (>) into a text file). This example shows one column of numbers, representing the voxel values for functional sub-brick #4 ('Negative Face-to-Face interaction' t-values) that fall within the ROI mask:



- More than one sub-brick can be chosen at a time (e.g., func_slim+orig'[2,4-6])
- Main application of **3dmaskdump** is to dump out the data or functional values that match an ROI so they can be processed in some other program (e.g., Microsoft Excel)
- If **-noijk** option is omitted, each output line starts with ijk-indexes (i.e., location) of the voxel
  - Program **3dUndump** can be used to create a dataset from a text file with ijk-indexes and dataset values

- **3dROIstats**
  - Program to compute separate statistics for each ROI in a dataset
    - E.g. Mean can be computed for *several* ROIs separately and simultaneously
    - This differs from **3dmaskave** because the ROIs within a single mask are not collapsed and then averaged. Here the averages are done *separately* for each ROI within the mask dataset
  - Averaging is done over each region defined by a distinct numerical value in the ROI dataset

3rois+orig



ROI 1: hippocampus
ROI 2: amygdala
ROI 3: superior temporal gyrus

- Example:

```
3dROIstats -mask 3rois+orig func_slim+orig'[0]'
```

Output shown in the shell (use **>** command to save into to a text file):

```
File              Sub-brick   Mean_1   Mean_2   Mean_3
func_slim+orig        0       2.006    2.395    4.044
```

➥ The **Mean_1** column is the average over the ROI whose mask value is "1". The average is calculated for voxels from our functional dataset **func_slim+orig**, that fall within the ROI. Averages are computed at sub-brick #0

➪ Means have also been computed for ROIs whose mask value is "2" (Mean_2) and "3" (Mean_3)

- Very useful if you create ROI masks for a number of subjects, using the same number codes for the same anatomical regions (e.g., 1=hippocampus, 2=amygdala, 3=superior temporal gyrus, etc.)
- You can load the output of **3dROIstats** into a spreadsheet for further analysis (e.g., statistics with other subjects' data)

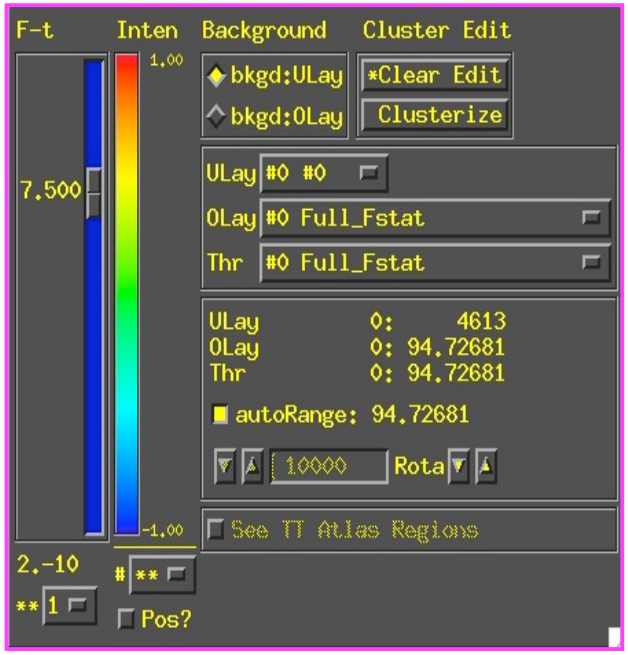
Method 2: Creating ROI datasets from Activation Maps

- The program **3dmerge** can find contiguous supra (above) threshold voxel clusters in an activation (functional) map and then convert each cluster into a ROI with a separate data value
  - ★ These ROIs can then be used as starting points for some analysis

- Example:

  cd AFNI_data3/roi_demo and launch **afni &**

- Let's pick select some criteria that will determine how big our voxel clusters will be. Any voxels that survive these criteria will be part of our ROI mask(s)



Select UnderLay: **anat+orig**

Select OverLay: **func_slim+orig**

--> Define OLay & Threshold: **Sub-brick 0 (Full-F)**

--> Set Threshold to **F ≥ 7.5**

--> To be part of a cluster, voxels must be right next to each other, **rmm = 3.5**

--> Clusters must be 50+ voxels in size

3.75 x 3.75 x 3.5 = 49.22 x 50 = **vmul = 2461**

```
3dmerge -prefix func_roi -1clip 7.5   \
        -1clust_order 3.5 2461 func_slim+orig.'[0]'
```

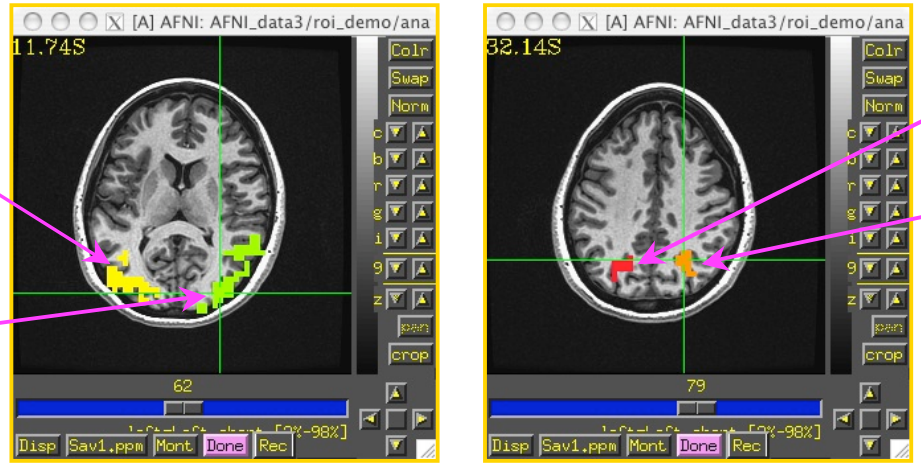
- ★ **-1clip 7.5**: Ignore voxels that don't survive your threshold (e.g., F=7.5) -- the threshold could be any stat you have set, a t-test, an F-value, a correlation coefficient, a mean, etc..
- ★ **-1clust_order 3.5 2461**: Here we've told the program to include voxels as part of a "cluster" if they are no more than 3.5mm apart (i.e., right next to each other in this case). The 2461 indicates the minimum volume (in *microliters* (vmul)) required to form a cluster. In this example, a cluster is defined by 50 or more voxels grouped together (voxel size: 3.75 x 3.75 x **3.5** x 50 voxels = 2461 vmul)
  - ➥ or try this command: It's the same as above but now we're pretending the voxel dimensions are 1x1x1 (it becomes much easier to figure out the vmul).

```
3dmerge -prefix func_roi -dxyz=1 -1clip 7.5   \
        -1clust_order 1.00 50 func_slim+orig.'[0]'
```

- <u>The result</u>: 4 clusters survived our criteria:

ULay: `anat+orig`

OLay: `func_roi+orig`



- The program **3dclust** looks for clusters of activity that fit the criteria set on the command line, and prints out a report about the active voxels that make up the ROI cluster(s) -- similar to 3dmerge, but creates a report instead of a new dataset

- Example:

```
3dclust  -1clip 7.5 3.5 2461 func_slim+orig.'[0]'
```

or…

```
3dclust  -dxyz=1 -1clip 7.5 1 50 func_slim+orig.'[0]'
```

  - ★ The above command tells **3dclust** to find potential cluster volumes for dataset **func_slim+orig**, sub-brick #0, where the threshold has been set to 7.5 (i.e., ignore voxels with an activation threshold <7.5). Voxels must be no more than 3.5mm apart, and cluster volume must be at least 2461 micro-liters in size (if using the -dxyz=1 option, voxels will be no more than 1mm apart and vmul will be 50).

  - ★ Once these ROI clusters have been identified, a report will be printed out:

#Volume	CM RL	CM AP	CM IS	minRL	maxRL	minAP	maxAP	minIS	maxIS	Mean	SEM	Max Int	MI RL	MI AP	MI IS
387	-37.2	29.1	1.4	-61.4	-16.4	-1.3	58.7	-19.4	26.1	16.064	0.5	94.727	-38.9	28.7	-1.9
348	34.2	31.3	3.1	9.8	47.3	-1.3	58.7	-12.4	26.1	16.24	0.5369	80.252	32.3	28.7	1.6
75	-26.6	24.8	37.8	-42.7	-20.2	17.5	32.5	26.1	54.1	12.647	0.5967	29.788	-23.9	28.7	40.1
66	16.8	26.3	40.9	9.8	24.8	17.5	36.2	29.6	54.1	11.69	0.4156	22.931	17.3	21.2	40.1
# 876	-3.7	29.6	6.9							15.512	0.3164				

- In this example, 4 ROI clusters were found that fit the criteria designated by the **`3dclust`** command. Below is an explanation of the output:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
#Volume	CM RL	CM AP	CM IS	minRL	maxRL	minAP	maxAP	minIS	maxIS	Mean	SEM	Max Int	MI RL	MI AP	MI IS
#------	-----	-----	-----	-----	-----	-----	-----	-----	-----	-------	-------	-------	-----	-----	-----
387	-37.2	29.1	1.4	-61.4	-16.4	-1.3	58.7	-19.4	26.1	16.064	0.5	94.727	-38.9	28.7	-1.9
348	34.2	31.3	3.1	9.8	47.3	-1.3	58.7	-12.4	26.1	16.24	0.5369	80.252	32.3	28.7	1.6
75	-26.6	24.8	37.8	-42.7	-20.2	17.5	32.5	26.1	54.1	12.647	0.5967	29.788	-23.9	28.7	40.1
66	16.8	26.3	40.9	9.8	24.8	17.5	36.2	29.6	54.1	11.69	0.4156	22.931	17.3	21.2	40.1
#------	-----	-----	-----	-----	-----	-----	-----	-----	-----	-------	-------	-------	-----	-----	-----
# 876	-3.7	29.6	6.9							15.512	0.3164				

- ★ Volume: Size of each cluster volume
- ★ CM RL: Center of mass (CM) for each cluster in the Right-Left direction
- ★ CM AP: Center of mass for each cluster in the Anterior-Posterior direction
- ★ CM IS: Center of mass for each cluster in the Inferior-Superior direction
- ★ minRL,maxRL: Bounding box for cluster, min & max coordinates in R-L direction
- ★ minAP,maxAP: Bounding box for cluster, min & max coordinates in A-P direction
- ★ minIS, maxIS: Bounding box for cluster, min & max coordinates in I-S direction
- ★ Mean: Mean value for each volume cluster
- ★ SEM: Standard error of the mean for the volume cluster
- ★ Max Int: Maximum Intensity value for each volume cluster
- ★ MI RL: Maximum Intensity value in the R-L direction of each volume cluster
- ★ MI AP: Maximum intensity value in the A-P direction of each volume cluster
- ★ MI IS: Maximum intensity value in the I-S direction of each volume cluster

Method 3: Creating ROI datasets from Atlases: AFNI program `whereami`

- AFNI comes with a collection of atlas datasets
  - ★ Stored in same directory with binaries, user face files, and so on
    - Have you tried example #2 from `imcat -help` ?
  - ★ Atlas dataset names are of the form `TT_something+tlrc.HEAD / .BRIK`
- The `whereami` command line program can create a mask dataset using an atlas dataset and a name of a region stored inside the atlas dataset
- Example:

```
whereami -mask_atlas_region TT_Daemon:left:hippocampus \
         -prefix Lhip
```

  - ★ Produces a mask dataset named `Lhip+tlrc.HEAD / .BRIK` containing the voxels defined in the San Antonio Talairach Daemon as being in the left hippocampus
  - ★ To see ALL the regions available in all the atlases, type the command

```
whereami -show_atlas_code | less
```

  - ★ You could create multiple masks this way and then combine them into a multi-region ROI mask using `3dcalc`
  - ★ You may want to use such automatic atlas-generated masks as a starting point for custom editing of the mask for each subject (using `Draw Dataset` plugin)

- **whereami** can provide the user with more detailed information regarding the output of **3dclust**
  - ★ For instance, say you want more information regarding the center of mass voxels from each cluster (from the 3dclust output). I.e., where do they fall approximately within the atlases?

```
3dclust -dxyz=1 -1clip 9.5  1 1000 func_FullF+tlrc > clusts.1D
whereami -coord_file clusts.1D'[1,2,3]' -tab | less
```

Center of mass output, columns 1,2,3, from 3dclust output (see pg. 25).

```
++ Input coordinates orientation set by default rules to RAI
++ Input coordinates space set by default rules to TLRC
+++++++ nearby Atlas structures +++++++

Focus point (LPI)                          Coord.Space
  42 mm [R], -61 mm [P],  -3 mm [I]        {T-T Atlas}
  43 mm [R], -63 mm [P],  -7 mm [I]        {MNI Brain}
  45 mm [R], -69 mm [P],   2 mm [I]        {MNI Anat.}
Atlas           Within  Label                                Prob.   Code
TT_Daemon       3.0     Right Middle Occipital Gyrus          ---     33
TT_Daemon       3.0     Right Brodmann area 37                ---     113
TT_Daemon       4.0     Right Inferior Temporal Gyrus         ---     29
TT_Daemon       4.0     Right Middle Temporal Gyrus           ---     35
TT_Daemon       6.0     Right Inferior Occipital Gyrus        ---     28
TT_Daemon       7.0     Right Brodmann area 19                ---     96
CA_N27_MPM      4.0     hOC5 (V5 / MT+)                       MPM     110
CA_N27_ML       0.0     Right Inferior Temporal Gyrus         ---     90
CA_N27_ML       1.0     Right Middle Temporal Gyrus           ---     86
CA_N27_ML       5.0     Right Inferior Occipital Gyrus        ---     54
CA_N27_ML       6.0     Right Middle Occipital Gyrus          ---     52

******** Please use results with caution! ********
******** Brain anatomy is quite variable! ********
******** The database may contain errors! ********
```

Shown: Cluster #1's coordinates according to various atlases (TT, MNI, etc), as well as the name of the anatomical structure that is located at or near these coordinates (which may vary by atlas)

etc…

- **whereami** can extract ROIs for various atlas regions using symbolic notation

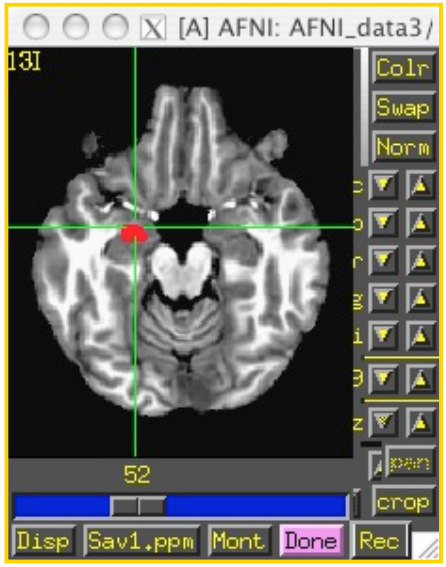
  whereami -mask_atlas_region TT_Daemon:left:amy

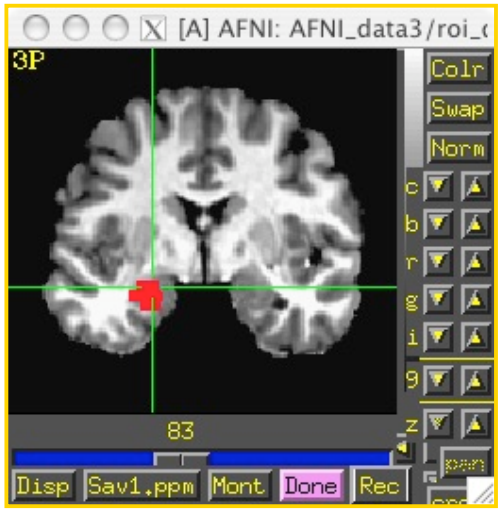
  ★ The above command will use the Talairach-Tourneaux atlas (**TT_Daemon:**) to create an ROI of the left amygdala (**left:amy**).

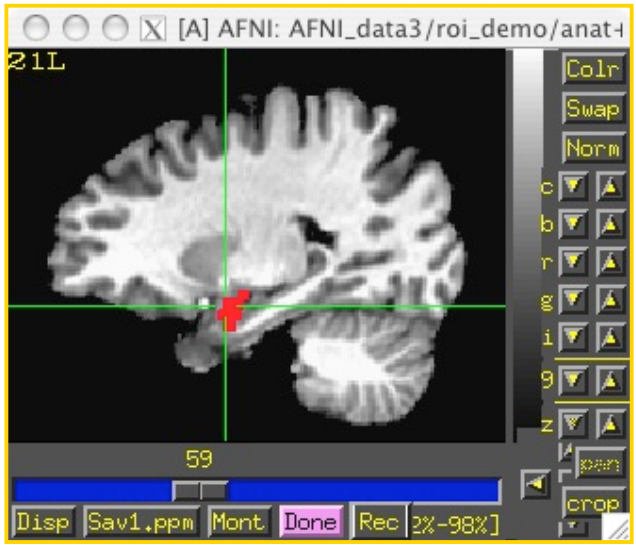
  ➥ See **whereami -help** for more details regarding available atlases

  ★ Result: An ROI dataset called **TT_Daemon.amy.l+tlrc.HEAD/BRIK**:

ULay: **anat+tlrc**
OLay: **TT_Daemon.amy.l.tlrc**







- **whereami** can report on the overlap of ROIs with atlas-defined regions

```
whereami -omask anat_roi+tlrc
```

```
++ Input coordinates orientation set by default rules to RAI
++ Input coordinates space set by default rules to TLRC
++ In ordered mode ...
++ Have 2 unique values of:
   0   1
++ Skipping unique value of 0
++ Processing unique value of 1
++    195 voxels in ROI
++    195 voxels in atlas-resampled mask
Intersection of ROI (valued 1) with atlas TT_Daemon (sb0):
   89.2 % overlap with Middle Occipital Gyrus, code 33
   6.7  % overlap with Middle Temporal Gyrus, code 35
   -----
   95.9 % of cluster accounted for.

Intersection of ROI (valued 1) with atlas TT_Daemon (sb1):
   19.5 % overlap with Brodmann area 37, code 113
   1.5  % overlap with Brodmann area 19, code 96
   -----
   21.0 % of cluster accounted for.

++    195 voxels in atlas-resampled mask
Intersection of ROI (valued 1) with atlas CA_N27_MPM (sb0):
   1.5  % overlap with hOC5 (V5 / MT+), code 110
   -----
   1.5  % of cluster accounted for.

++    195 voxels in atlas-resampled mask
Intersection of ROI (valued 1) with atlas CA_N27_ML (sb0):
   61.0 % overlap with Right Middle Occipital Gyrus, code 52
   20.0 % overlap with Right Middle Temporal Gyrus, code 86
   -----
   81.0 % of cluster accounted for.
```

- You can also specify atlas-based ROI masks directly like this:

```
3dmaskave -mask TT_Daemon:left:hippocampus func_FullF+tlrc
```

  - ★ In the above single command, 2 things will occur:
    1. The Talairach-Tourneaux atlas will be used to create an ROI mask of the left hippocampus, which will be applied to the dataset **func_FullF+tlrc** (note: this dataset has one sub-brick in it - small and simple, for demo purposes).
    2. A mean will be calculated for the voxels in dataset **func_FullF+tlrc** that fall within the ROI mask.
  - ★ Output from above command shows up on the terminal like this:

```
++ 3dmaskave: AFNI version=AFNI_2007_05_29_1644 (Jul 10 2007) [32-bit]
+++ 1112 voxels survive the mask
3.59374 [1112 voxels]
```

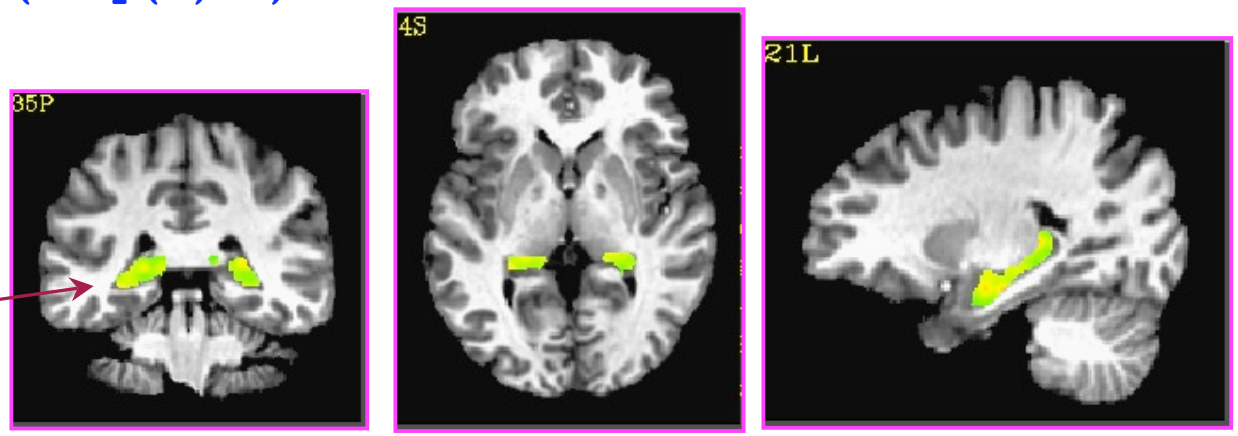
Or

```
3dcalc  -prefix nice_roi                                  \
        -a 'CA_N27_ML::hippo' -b 'func_FullF+tlrc'        \
        -expr '(step(a)*b)'
```

  - ★ Result:

ULay: **anat+tlrc**
OLay: **nice_roi+tlrc**

F-stat voxels in func dataset that fall within the TT left hippocampus ROI



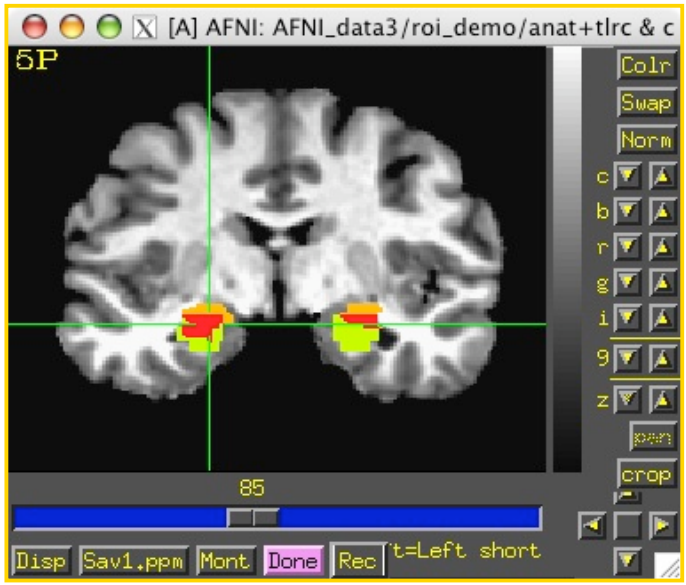
- Above examples (from pg.30) are using a new feature of the AFNI software package:
  - ➥ Creation of a 0-or-1 mask dataset *directly* on the command line using a dataset name of the form **`Atlas_name:Hemisphere:Region_name`**
  - ★ Using this feature, you don't have to create the mask using the **`whereami`** program and then use it later— you can create it and use it at the same time

- Example 9 from **`3dcalc -help`**:

  Compare the left and right amygdala between the Talairach atlas, and the CA_N27_ML atlas. The result will be **1** if a voxel is marked as amygdala in the **TT_Daemon** only, **2** if it is marked as amygdala in the **CA_N27_ML** only, and **3** where they overlap.

```
3dcalc  -a 'TT_Daemon::amygdala' \
        -b 'CA_N27_ML::amygdala' \
        -expr 'step(a)+2*step(b)'\
        -prefix compare.maps
```

Note: **`compare.maps+tlrc`** displays the TT amygdala (value=1) in green, the N27 amygdala (value=2) in orange, and the overlap between the two atlases (value=3) in red.



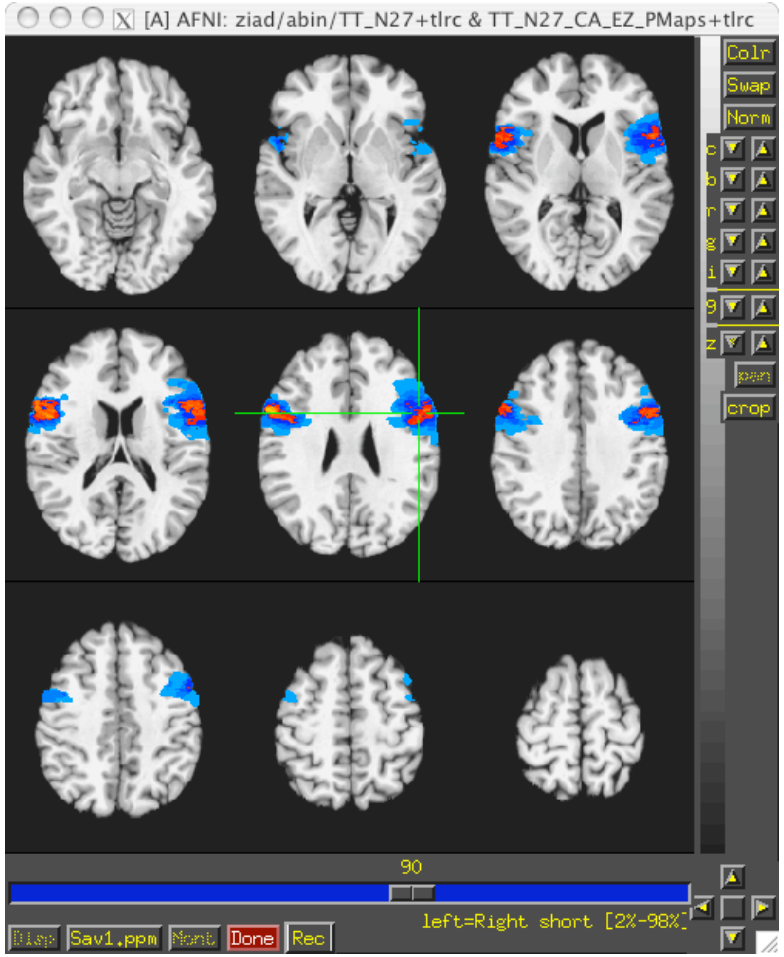
- For more information about **`3dcalc`** , see the AFNI Utilities presentation

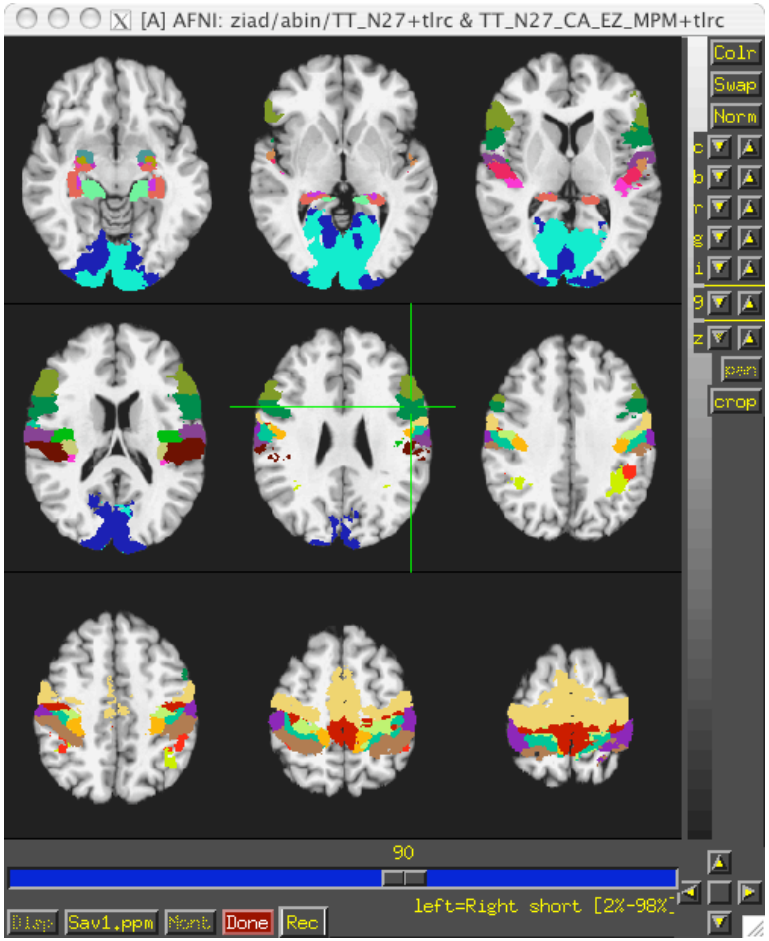
Atlases Distributed With AFNI
TT_Daemon

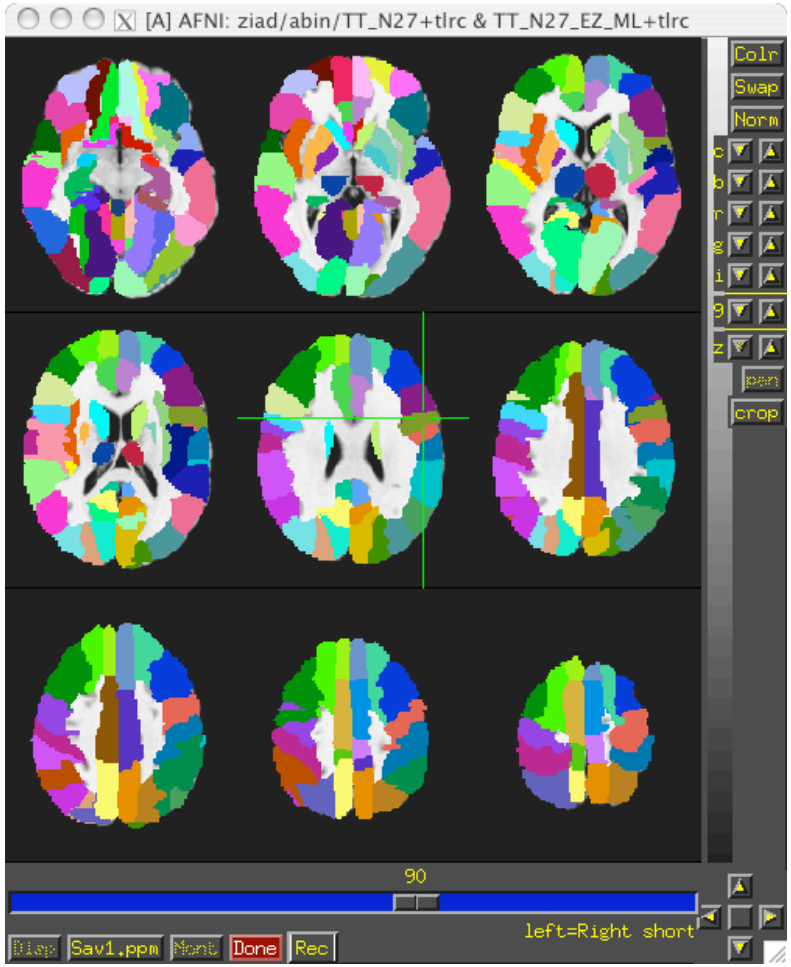
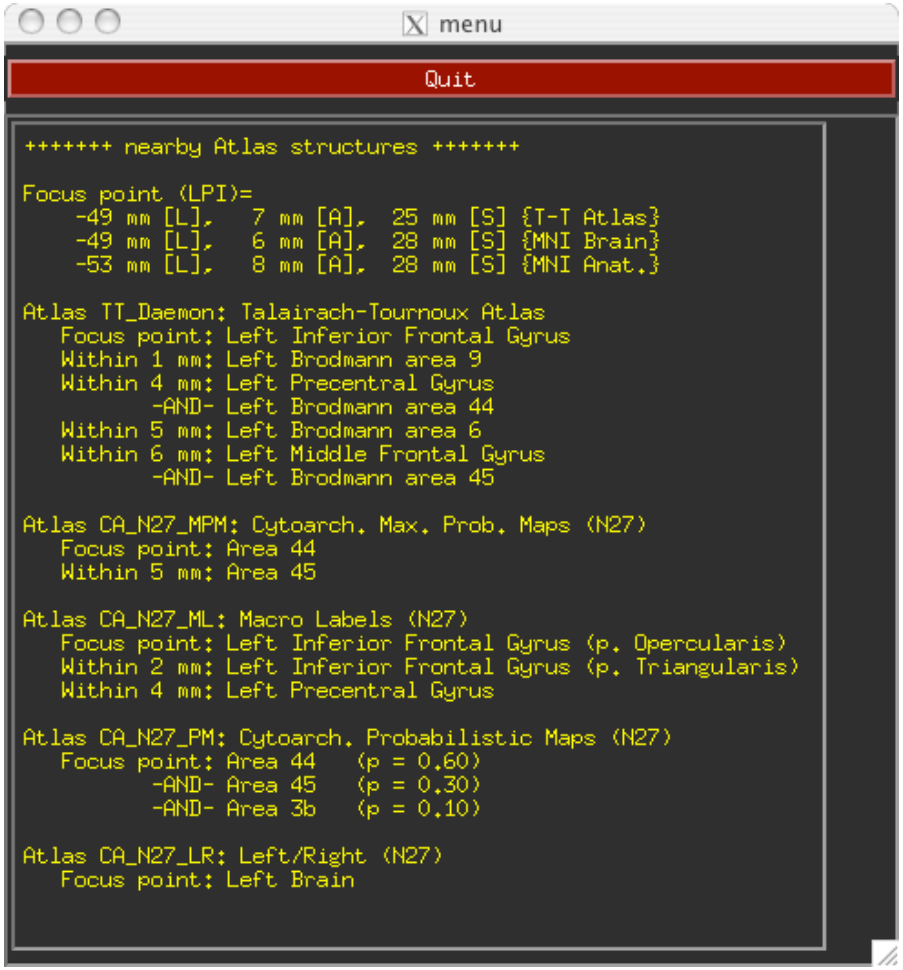
- **TT_Daemon** : Created by tracing Talairach and Tournoux brain illustrations.
  - ★ Generously contributed by Jack Lancaster and Peter Fox of RIC UTHSCSA)

Atlases Distributed With AFNI
Anatomy Toolbox: Prob. Maps, Max. Prob. Maps

- **CA_N27_MPM**, **CA_N27_ML**, **CA_N27_PM**: Anatomy Toolbox's atlases with some created from cytoarchitectonic studies of 10 human post-mortem brains
  - ★ Generously contributed by Simon Eickhoff, Katrin Amunts and Karl Zilles of IME, Julich, Germany





Atlases Distributed With AFNI: Anatomy Toolbox: MacroLabels

- CA_N27_MPM, CA_N27_ML, CA_N27_PM: Anatomy Toolbox's atlases with some created from cytoarchitectonic studies of 10 human post-mortem brains
  - ★ Generously contributed by Simon Eickhoff, Katrin Amunts and Karl Zilles of IME, Julich, Germany