1 Using the 3D: Input File Specification

AFNI and its associated programs can read files in many different formats. Rather than encode all different image formats in the software (which would require one format for every programmer that ever lived), the "generic" image file format is an uncompressed contiguous 2D or 3D array of 8 bit unsigned bytes, 16 bit signed shorts, 32 bit signed ints, or 32 bit floats.

The burden is on the user to specify exactly WHERE in the file the image data is stored. This is done with the rather clumsy mechanism of prepending the necessary information to each input filename.

An example:

3D:80:0:64:64:1:fred.001

Piece	Meaning
3D:	Indicates that this is a file of shorts.
80:	Indicates to skip 80 bytes at the start of the file.
0:	Indicates to skip 0 bytes at the start of each 2D image within the file.
64:	Indicates that the "x" dimension of each 2D image is 64.
64:	Indicates that the "y" dimension of each 2D image is 64.
1:	Indicates that there is only 1 2D image in this file.
fred.001	Indicates that the actual filename is "fred.001".

If there were 10 such files, you could put them into a 3D block with the Unix command

cat fred.* > fred3d

This file could be read into to3d with the specification

3D:0:80:64:64:10:fred3d

Notice that the "80" has moved to the "per image header" slot, since each image has the 80 bytes of header information attached. If there were only one 80 byte header for all the data in this big file, then the appropriate specification would be

3D:80:0:64:64:10:fred3d

In general, the format for images of shorts is

3D:hglobal:himage:nx:ny:nz:fname	for 16 bit input
3Ds:hglobal:himage:nx:ny:nz:fname	for 16 bit input, swapped bytes
3Db:hglobal:himage:nx:ny:nz:fname	for 8 bit input
3Di:hglobal:himage:nx:ny:nz:fname	for 32 bit input
3Df:hglobal:himage:nx:ny:nz:fname	for floating point input

where

hglobal	= number of bytes to skip at start of whole file
himage	= number of bytes to skip at start of each image
nx	= x dimension of each image
ny	= y dimension of each image
nz	= number of images (at least 1)
fname	= actual filename on disk to read

The ':' separators are required. The k-th image starts at BYTE offset

hglobal + (k+1)*himage + vs*k*nx*ny

in file 'fname' for k=0,1,...,nz-1. Here, vs=voxel length=1 for bytes, 2 for shorts, 4 for ints and floats.

As a special case, hglobal = -1 means read data starting at offset len-nz*(vs*nx*ny+himage), where len=file size in bytes. In particular, this means that if you have a 64x64 image of shorts in a file, you can read it with

3D:-1:0:64:64:1:filename

1.1 Reading a Number of Files, Method 1

This is clearly a rather clumsy method to use if you have a bunch of files in a funny format. The `count` program (part of the *AFNI* package) and the C-shell backquote operator can be combined to remove a little of the pain.

The `count` program simply generates strings of numbers. For example, the output of "count -digits 3 5 17" is

005 006 007 008 009 010 011 012 013 014 015 016 017

"-digits 3" means to use 3 digits (at least); "5 17" means to count from 5 to 17. A more complex use of count can attach a root to the beginning of each number: "count -digits 3 -root fred. 5 17" produces

fred.005 fred.006 fred.007 fred.008 fred.009 fred.010 fred.011 fred.012 fred.013 fred.014 fred.015 fred.016 fred.017

So, to read in these files, we would use the count command

count -digits 3 -root 3D:-1:0:64:64:1:fred. 5 17

which produces the output
3D:-1:0:64:64:1:fred.005 3D:-1:0:64:64:1:fred.006 3D:-1:0:64:64:1:fred.007
3D:-1:0:64:64:1:fred.008 3D:-1:0:64:64:1:fred.009 3D:-1:0:64:64:1:fred.010
3D:-1:0:64:64:1:fred.011 3D:-1:0:64:64:1:fred.012 3D:-1:0:64:64:1:fred.013
3D:-1:0:64:64:1:fred.014 3D:-1:0:64:64:1:fred.015 3D:-1:0:64:64:1:fred.016
3D:-1:0:64:64:1:fred.017

Finally, we need to get this string onto a command line. This is done by putting the whole count command inside backquotes. In the Unix C-shell, this means to execute the command that is backquoted, then put its output into the whole command line about to be executed. Thus, we could do

to3d -fim ‘count -digits 3 -root 3D:-1:0:64:64:1:fred. 5 17‘

This is clumsy, but not as bad as typing all the 3D: prefixes for each file.

Finally, note that the program FD2 cannot read in 3D blocks of data. Each input file must contain only 1 image. It can, however, use the 3D: prefix to properly skip arbitrary header information before the actual image data is read. The F64 and F128 scripts in the AFNI 1.04a distribution show how this can be done to trick the program into reading slices out of a 3D block.

1.2 Reading a Number of Files, Method 2

If you have AFNI version 1.04a (or higher), then there is an easier way to customize the input formats. It still uses the 3D: specifiers, so you need to understand those. The method is to define a Unix environment variable to associate a 3D: specifier with a given file size. For example,

setenv MCW_IMSIZE_1 8272=3D:-1:0:64:64:1:

means that a file whose length is 8272 bytes should be read with the string “3D:-1:0:64:64:1:” attached before its filename. This will allow you to tell the AFNI programs how to deal with your particular filesizes. You can use the environment variables MCW_IMSIZE_1 up to MCW_IMSIZE_99. If this is done, then the command “FD2 fred.*” would work to read the “fred.001” ... files described earlier.

The following shows the second form of an allowed MCW_IMSIZE environment variable:

setenv MCW_IMSIZE_2 %16096+80=3D:80:7904:64:64:

The “%” that starts the value signals that this is a variable length file type that can contain more than one image. In this case, if the file length is of the form 16096*N+80 bytes, where N=1,2,..., then the string “3D:80:7904:64:64:N:” will be put before the filename. This example allows the input of a 3D brick of images, each 2D image with a 7904 byte header, and the whole collection with an 80 byte header at the very beginning.