nifti converter plugin for brainvoyager qx

Hester Breman
Brain Innovation B.V.

October 16, 2007

Contents

Chapter 1

Introduction

1.1 Description

1.1.1 The NIfTI-1 data format

This is a description of the nifti-converter for BrainVoyager QX [2]. NIfTI-1 is the new data standard defined by the Data Format Working Group (DFWG). For more information about the data format, see `http://nifti.nimh.nih.gov` and [1]. It is also downloadable in *.chm and FlashHelp format from the BrainVoyager wiki at `http://wiki.brainvoyager.com/BVQX_Plugins`. For other implementations, see `http://niftilib.sourceforge.net`.

1.1.2 Changes in the NIfTI converter for BrainVoyager QX

The nifti-converter has been completely refurbished because of portability limitations and code maintenance reasons. Version 1.04 was only for Windows. The new version (from 1.05) is cross-platform. Furthermore, the BrainVoyager QX plugin access functions have been used to obtain access to the files instead of reading and writing directly from and to disk. Finally, the implementation is based on C++ instead of C to a much larger extent, using inheritance and templates.

Document version: 0.3

Applicable to BrainVoyager QX version: 1.9

Applicable to plugin version: 1.05

Table 1.1: Byte order conversions

Platform (byte order)	*Import to BVQX*	*Export to NIfTI-1*
Windows XP	`use nifti1_io.h/c`	n/a
Mac PowerPC (Big Endian)	use CoreFoundation.h	n/a
Mac Intel (Little Endian)	n/a	n/a
Linux SUSE 10 (Little Endian)	`use nifti1_io.h/c`	n/a

1.2 Availability

During development, the intermediate plugins are placed on the BV wiki (`http://wiki.brainvoyager.com/BVQX_plugins`) as fileheader-viewers. Conversion functions will be added regularly and the updated versions will be placed on the wiki. From October, 2007, the plugin and source code will be available from the NITRC website at `http://www.nitrc.org/`.

Chapter 2

Structural design

2.0.1 Classes

The new structure will be the following (see figure 2.1):

- superclass neuroimagefile, with generic methods for reading and transforming data; these are polymorphic classes
  - subclass brainvoyagerfile
    - subclasses bvfiles (vmr, fmr, vtc, vmp, ...)
      - subsubclasses bvfiles (vmpar, vmpnr, ...)
  - subclass niftifile
    - subclasses niftifiles (nifti1, gifti, sifti, ...)
- classes for tools (datahandler, utilities, displaybuilder, matrixmaster, transformer, ...)
- a headerfile with constants

The logic behind the converter is that all procedures are performed at the highest level possible, to make the code as efficient as possible c.q. avoid duplicate routines.
So all filetype-specific information that a file has in common with other filetypes, is transferred to the level 'neuroimagefile', for example the size of the file on x-axis, y-axis and z-axis. This makes it possible to read the data via the generic template class 'datahandler'.
The main course of the conversion is performed in the file 'nifticonverter.cpp', where 'neuroimagefile' class functions are invoked to read, convert and write. The subclasses should be independent of each other, so that it is easy to isolate or insert a class of files.

Superclasslevel
All data transformations, are performed on the level of the superclass, the 'neuroimagefile' class. When a subclass file is read, its data will be loaded before conversion to the 'neuroimagefile' class field 'fdata'. These float data are then transformed using the 'matrixmaster', 'datahandler' and 'transformer' classes.
The conversion of the headers also takes place on the level of the 'neuroimagefile' class. When a file is created, its default values on neuroimageclass level are filled, for example for VTC files that the time runs fastest. This makes it possible to convert header and data on the highest level while the converter is in fact ignorant

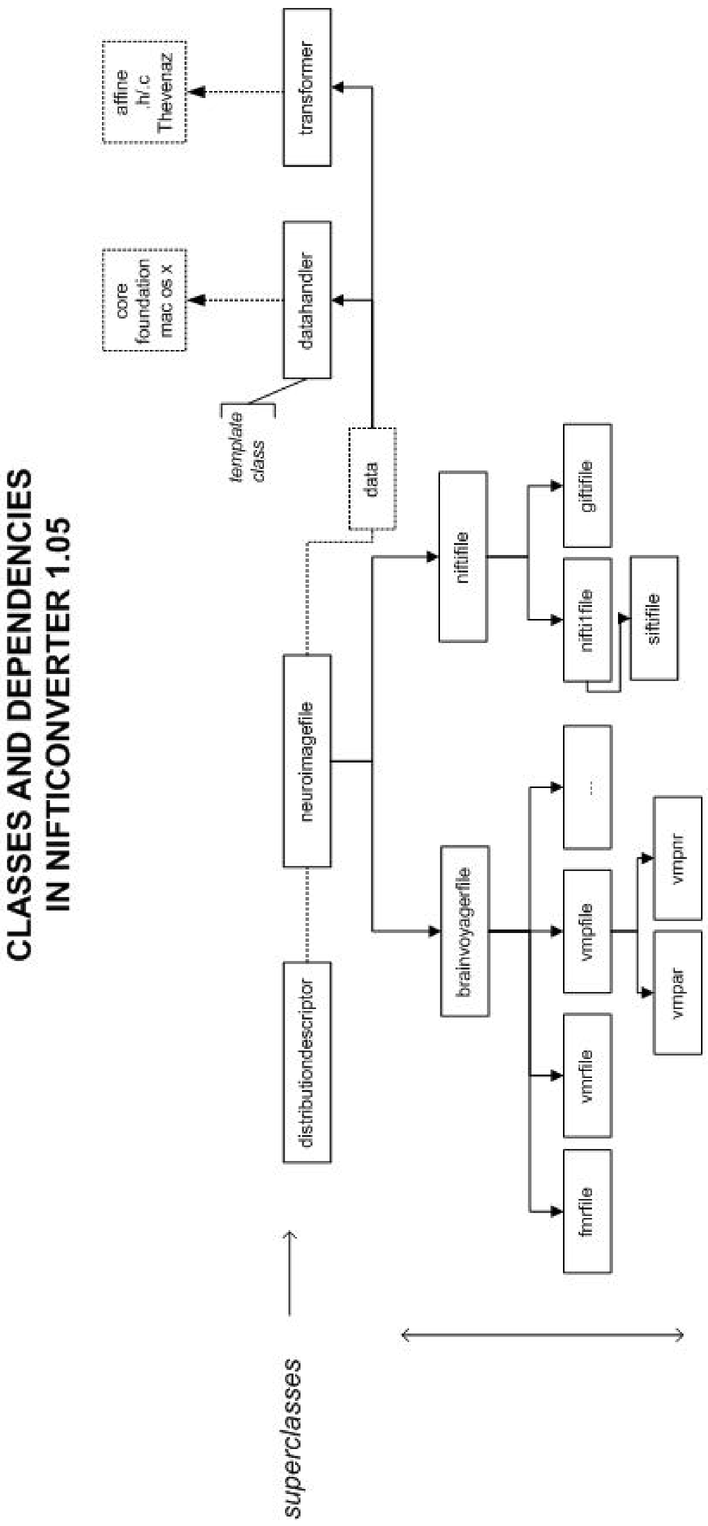


Figure 2.1: NIfTI converter classes

about the source file class or destination file class.
The `neuroimagefile` has coordinate system DICOM (in field `coordsys`) where the position of the image is described via the $4x4$ matrix `position`, while the subclasses and subsubclasses can contain positioning and transformation information in the fields `localposition` according to the local coordinate system contained in the field `localcoordsys`.

Subclass level

On the subclass level, which are the 'niftifile' and 'brainvoyagerfile' classes, the flags for the coordinate system and orientation are set by default.

Subsubclass level

The implementation for the read and write routines is specific to the subsubclass of the neuroimagefile class that the file belongs to. When reading data via function `load_data()`, the native data are converted to a 3D or 4D float volume.
When writing the converted file to disk, the float data are cast to the native data type of the destination file class. For reading and writing BrainVoyager file classes, the routines used are where possible the BVQX plugin access functions, which rely on file currently opened in the BrainVoyager main window. If the file is not available, the converter will try to read from disk. For NIfTI-1 files, the routines offered in the `nifti1_io.h/c` files by Robert W. Cox and Rick Reynolds are used.

2.0.2 Swapping

When a NIfTI image is read, the image is swapped automatically to system byte order by the `nifti1_io.h/c` code. For BrainVoyager files, the flag `databyteorder` is set during construction of the object `brainvoyagerfile` to `BYTEORDER_IEEE_LE`. When the BrainVoyager file is written, the flags `databyteorder` and `sysbyteorder` are compared. In case of incongruence, swapping will be applied in the function `write()`. Here the subsubheader can be swapped while the superclass metadata stay in congruence with the system byte order. This makes it possible to easily write without swapping back and forth in order to figure out the dimensions of the image.

2.0.3 File naming

The name of the original file is used where possible. If the file has no name, like in the case of a BrainVoyager contrast map created from a GLM file via the GLM Options dialog, the plugin tries to obtain the name from the underlying VTC file to create a name `<vtcname>_statmap.nii`.

2.0.4 Positioning

In NIfTI-1, images are positioned with respect to the NIfTI-1 coordinate system. In BrainVoyager QX, however, different coordinate systems are used (see appendix A. Also, images can be a subspace of an anatomical image; this means that it has coordinates with respect to the file of which the image is a subspace. The calculation of positioning information is being performed in the class `posfile`. The calculation principle is depicted in figure 2.2.

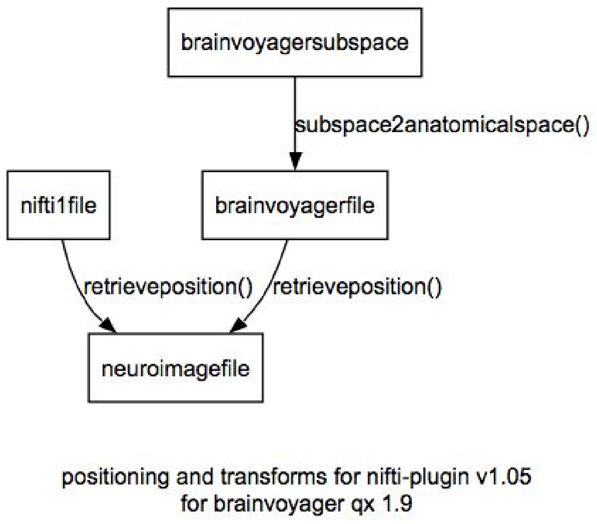


Figure 2.2: Positioning

2.0.5 Statistical data

Statistical data are described internally via the `distributiondescriptor` class, which is a member of the `neuroimagefile` class. The advantage of not directly writing conversion code separately for the statistical data files is that it is transparent what happens to the values. Also, extension and maintenance of the code is easier.

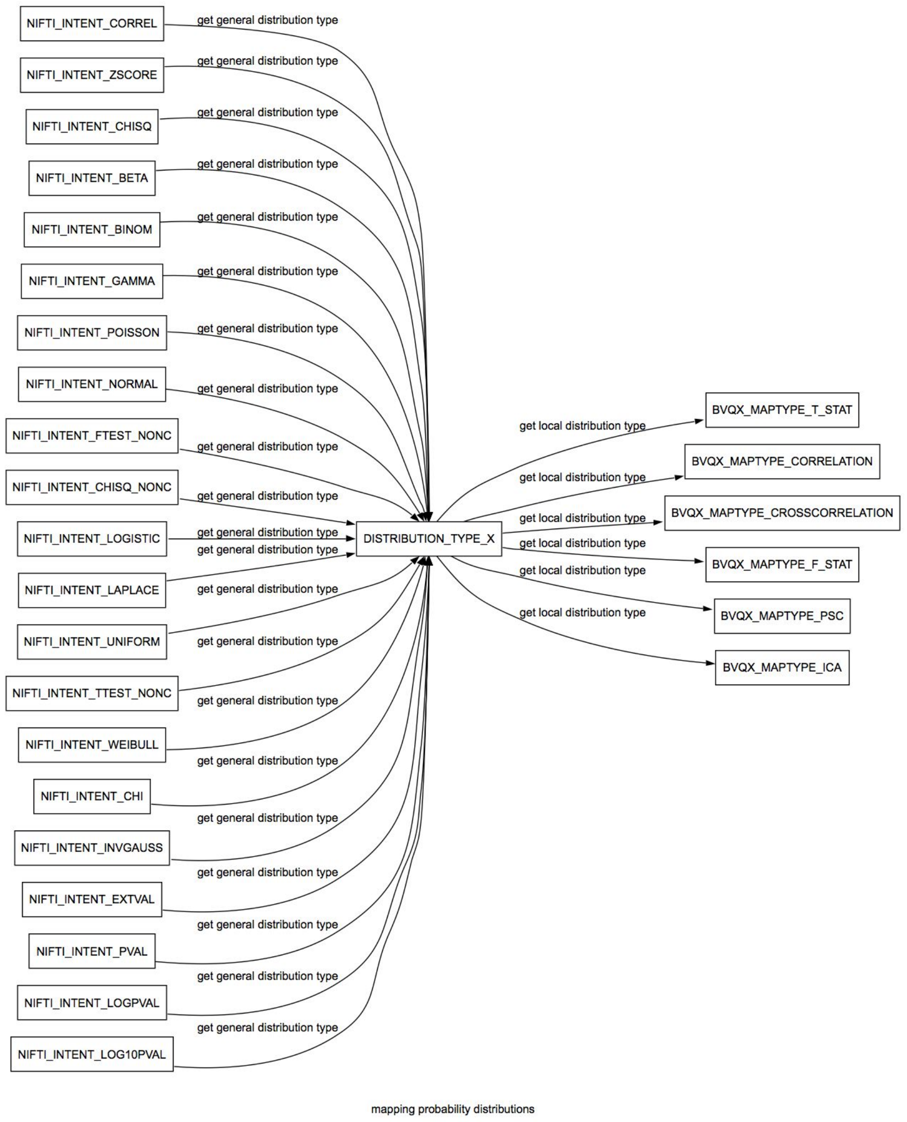


Figure 2.3: Matching the probability distribution types in NIfTI-1 and BrainVoyager QX

Chapter 3

Functional design: the conversion procedure

3.1 Introduction

Because of the C++ inheritance mechanisms, it was possible to invoke the conversion procedure in the main file `nifticonverter.cpp`, which enhances the transparency of the code. Also, the procedure looks simple, because the conversion can now be done in just a few lines (see figure 3.2); the naming of the functions is dictated by the superlevel neuroimagefile class, and the implementation can be found in subclasses.

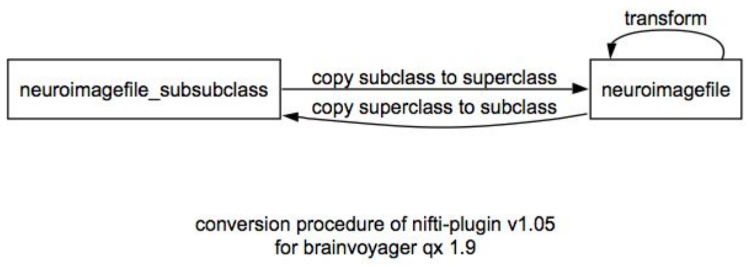

transform
neuroimagefile_subsubclass
copy subclass to superclass
copy superclass to subclass
neuroimagefile
conversion procedure of nifti-plugin v1.05
for brainvoyager qx 1.9

Figure 3.1: Conversion procedure

The values in the superclass and subclass are initially set to 0 or false in the respective constructor classes. The structural filetype specific superclass values are set in the constructor function of the subclass, for example 'extension' or 'datatype'. The varying values, like for dimensions or name of the image, are copied on demand later, via the `copyheader()` function.

3.2 Process flow

3.2.1 reading

For reading files, the converter will first check if a file of that type is opened in BrainVoyager QX, via the method `getcurrentheader()`. BrainVoyager QX plugin access methods are used to retrieve the header and data of BrainVoyager files.

If the file is not currently opened, or it is a NIfTI file, the function `readheader()` will be invoked. In these cases the file will be read from disk; in the case of NIfTI-1 files, the functions in `nifti1_io.h/c` are used to read the file.

3.2.2 conversion

Metadata are copied from the filespecific header to the superclass header via `copyheader(SUBCLASS2SUPERCLASS)`; after having retrieved the positioning and transformation information via the `retrieveposition()` and `retrievetransformations()` functions, this also entails a transformation of the local coordinates in `localposition` to fill the `neuroimagefile::position` field, which is the image position with respect to the DICOM coordinate system. For BrainVoyager files, it will create a positioning matrix from the current VMR, FMR or DMR header. For NIfTI files, this will retrieve the `sto_xyz` or `qto_xyz` fields. The positioning matrix will be saved in the local header. Then, the data are loaded to the original file via `loaddata()`.

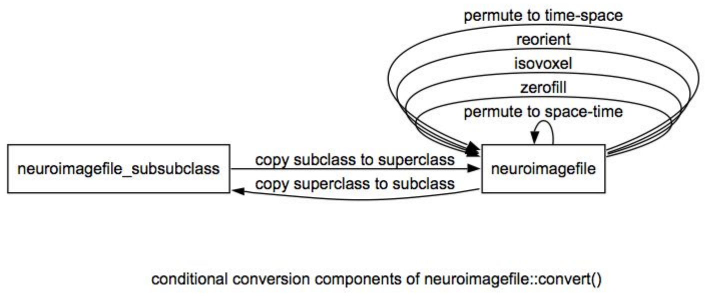

permute to time-space
reorient
isovoxel
zerofill
permute to space-time
neuroimagefile_subsubclass
copy subclass to superclass
copy superclass to subclass
neuroimagefile
conditional conversion components of neuroimagefile::convert()

Figure 3.2: neuroimagefile::convert()

The `loaddata()` function first tries to obtain the data via `getcurrentdata()`, which uses the native BrainVoyager QX plugin access functions and the `nifti1_io` function. Otherwise, in some cases, it will try to read from disk. In the function `loaddata()` the data are cast to float data type via the datahandler routine `cast2float()` and linked to the superclass field `fdata`.
Transfer of the data takes place on highest level in the hierarchy, in the `convert(imgfile)` function of the `neuroimagefile` class; metadata are then adapted if necessary, for example when the resolution changed (when importing to BrainVoyager). The data preserved at neuroimagefile class level are converted by the transformer and datahandler classes. For export to NIfTI, it will be possible to choose whether the image will be transformed or that just a positioning matrix will be attached (in SPM5 this is sufficient for proper display). In `transformer::reorient()` the flag `flag_transformimage` determines whether the image data will be transformed or not. If the flag `flag_transformimage` is false, the `copy(superclass2subclass)` method of the target file should take care of incorporating the reorientation matrix saved in the field `neuroimagefile::transform`.
The transformer class uses the code of Philippe Thevenaz with kind permission [3]. Then, the header and data are transferred from neuroimagefile class level to file specific level via `copyheader(SUPERCLASS2SUBCLASS)`.

3.2.3 writing

The file is written to disk via the `write()` function, a function which is implemented on the lowest level. For writing BrainVoyager files, the converter will check if a BrainVoyager QX plugin access method is available. In case of NIfTI files, the nifti-1 API is used.

Chapter 4

Platforms and differences

Platform differences for compiling:

Linux
The differences in code are:

1. Static BVQX plugin access headers
2. conccfiles shell script for assembling *.cpp files in one *.cpp file and compilation
3. modified `nifti1_io.h/c` files.

Because of templates, put datahandler.h lowest in concatenated.cpp and/or put all functions of template class in *.h file after class declaration (not IN class declaration). Try g++ if gcc does not work.
Compilation: once for each number of files: in shell, type `$ ./conccfiles <name>.so`
Recompilation: in shell, type `$ ./compile_linux`
Start bvqx: on linuxbi, in bin `./bvqx`

Mac
Makefile for compilation Open the `*.xproj` file in the application XCode and press 'Build'.

Windows
Visual Studio C++ 2005.
Compilation: 'Build' or 'Rebuild all'.

Appendix A

Coordinate systems

Summary of axes systems in BrainVoyager QX:

1. Internal coordinates. Origin at voxel (0, 0, 0).
   XBV: anterior → posterior
   YBV: superior → inferior
   ZBV: right → left

2. System coordinates. Origin, directions/values are defined the same as the internal coordinate system but axes names follow Talairach standard:
   XSYS: right → left
   YSYS: anterior → posterior
   ZSYS: superior → left

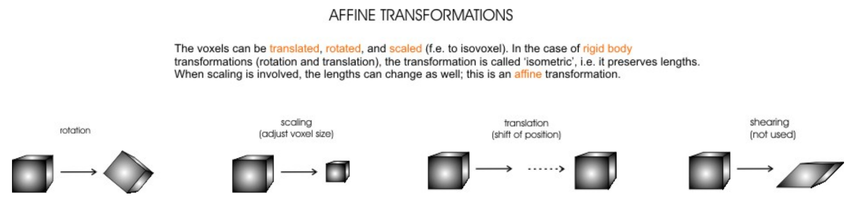
3. Talairach coordinates. Axes names like in system coordinates but opposite directions, origin in AC (128,128,128), values defined according to 8 landmarks (AC, PC, LP, RP, SP, IP, AP, PP).
   XTAL: left → right
   YTAL: posterior → anterior
   ZSYS: superior → left

4. OpenGL coordinates. Like internal (but also shown as system coordinates to the user, except small axes cross in left lower corner of OpenGL (surface) window.

Appendix B

Transformations

In this chapter a graphical representation of image transformations has been provided. For affine transformations, 4 types of parameters are possible, which are rotation, translation, scaling and shearing (see figure B).


AFFINE TRANSFORMATIONS

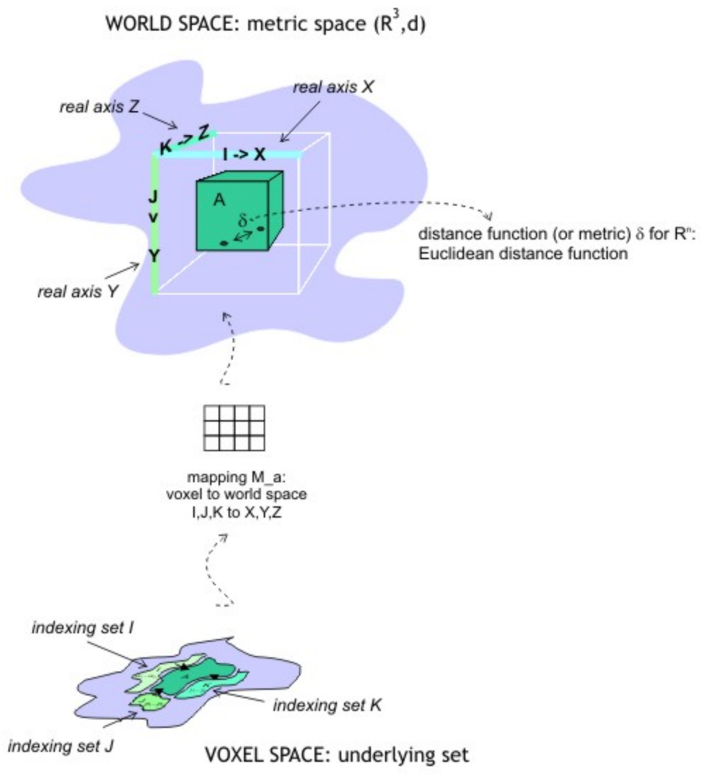
The voxels can be translated, rotated, and scaled (f.e. to isovoxel). In the case of rigid body transformations (rotation and translation), the transformation is called 'isometric', i.e. it preserves lengths. When scaling is involved, the lengths can change as well; this is an affine transformation.

Figure B.1: The affine transformations. Note: scaling operations are sometimes referred to as zooms.

Affine transformations can be concisely represented in a $4x4$ transformation matrix (see figure B), which computes a new position vector for each voxel. The transformation is linear, that is, the same transformation is used for all voxels in a volume.

Voxel space to world space

In an indexed set of voxels, the only order in the set is caused by the indexing sets providing dimensions to the set.
This set of numbered elements is also called voxel space. The number of dimensions is indicated by the number of indexing sets.
So if there are 3 sets, f.e. I,J and K, the set has 3 dimensions.
The size of the dimensions is indicated by the number of elements in each indexing set, f.e. 0...255 or 1..256.
Each element in the indexed family of subsets possesses the indexes, therefore total number of elements in the set is a multiplication of the sizes of the indexes of each indexing set (f.e. if there are elements $a_{\alpha 1,\beta 1,\gamma 1}$.. $a_{\alpha 256,\beta 256,\gamma 256}$, the total size of the voxel set is 256 x 256 x 256 = 16.777.216 elements).

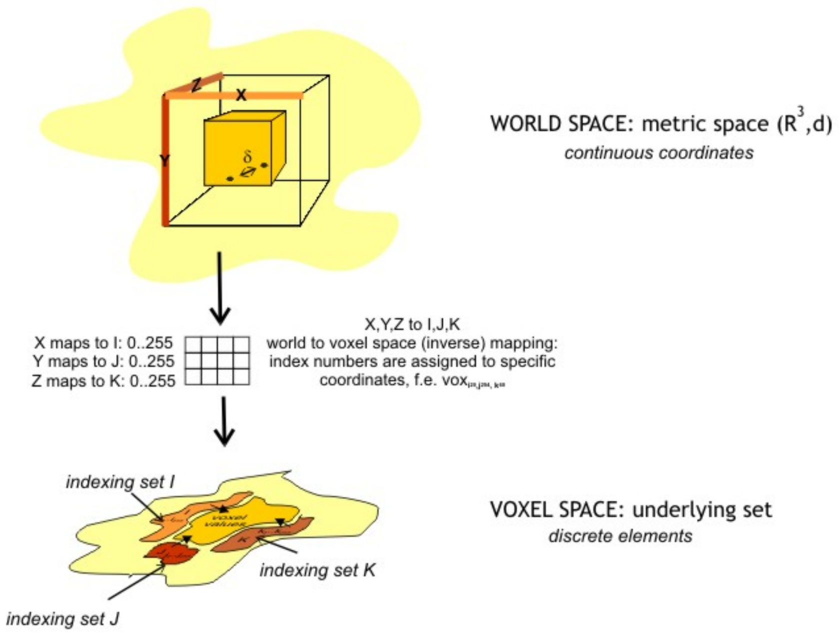


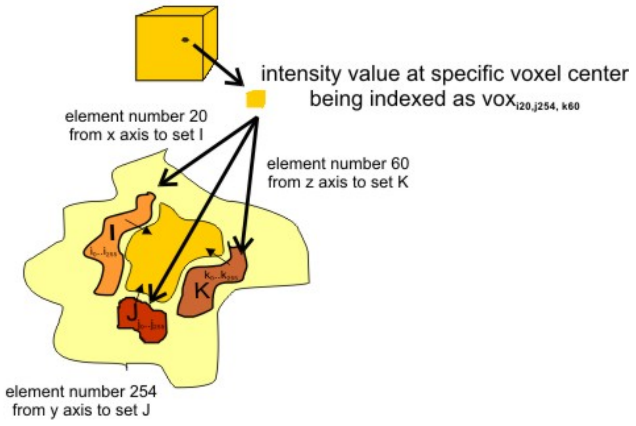
When the indexed family of subsets is projected onto a metric space, the indexed family does not only have a certain number of elements in each dimension but also the coordinates of each element in each dimension are provided.

For MRI images, the term world space is often used. This world space is an Euclidean metric space in 3 dimensions (R^3).

The formal notation of this metric space is (R^3,*d*), where *d* is the distance function.
A transformation function provides the coordinates that each element from the voxel set should have in each dimension.
So the transformation function maps the indexes I,J,K to the real X, Y and Z axes from 0..255.
For mapping back from a real world space to the set, the inverse of the transformation is used to end up with units of 1 so that the inverse mapping provides the indexing numbers.

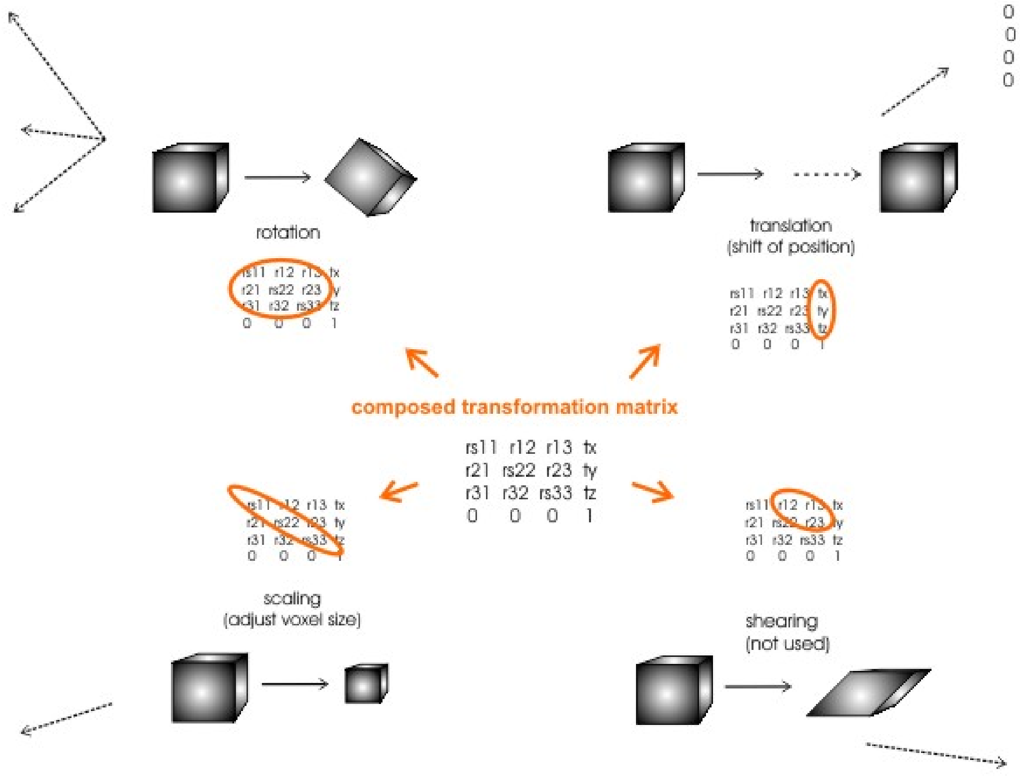
World space to voxel space


Z
X
Y
δ
WORLD SPACE: metric space (R^3,d)
continuous coordinates
X maps to I: 0..255
Y maps to J: 0..255
Z maps to K: 0..255
X,Y,Z to I,J,K
world to voxel space (inverse) mapping:
index numbers are assigned to specific
coordinates, f.e. $vox_{i20,j254,k60}$
indexing set I
voxel values
indexing set K
indexing set J
VOXEL SPACE: underlying set
discrete elements



AFFINE TRANSFORMATION MATRICES

The mapping, transforming points in one dataset to points in another dataset, involve translations, rotations and scaling. These transformations can be expressed in a single composed transformation matrix. This is a product of the respective translation, rotation, (shearing) and scaling matrices.



rotation matrices

rotation about x-axis

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\Theta) & \sin(\Theta) & 0 \\ 0 & -\sin(\Theta) & \cos(\Theta) & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

rotation about y-axis

$$\begin{matrix} \cos(\Theta) & 0 & \sin(\Theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\Theta) & 0 & \cos(\Theta) & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

rotation about z-axis

$$\begin{matrix} \cos(\Theta) & -\sin(\Theta) & 0 & 0 \\ \sin(\Theta) & \cos(\Theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

translation matrix

$$\begin{array}{ccccl} 0 & 0 & 0 & 1 & \text{translation x} \\ 0 & 0 & 0 & 1 & \text{translation y} \\ 0 & 0 & 0 & 1 & \text{translation z} \\ 0 & 0 & 0 & 1 & \end{array}$$

rotation

$$\begin{matrix} rs11 & r12 & r13 & tx \\ r21 & rs22 & r23 & ty \\ r31 & r32 & rs33 & tz \\ 0 & 0 & 0 & 1 \end{matrix}$$

translation (shift of position)

$$\begin{matrix} rs11 & r12 & r13 & tx \\ r21 & rs22 & r23 & ty \\ r31 & r32 & rs33 & tz \\ 0 & 0 & 0 & 1 \end{matrix}$$

composed transformation matrix

$$\begin{matrix} rs11 & r12 & r13 & tx \\ r21 & rs22 & r23 & ty \\ r31 & r32 & rs33 & tz \\ 0 & 0 & 0 & 1 \end{matrix}$$

scaling (adjust voxel size)

$$\begin{matrix} rs11 & r12 & r13 & tx \\ r21 & rs22 & r23 & ty \\ r31 & r32 & rs33 & tz \\ 0 & 0 & 0 & 1 \end{matrix}$$

shearing (not used)

$$\begin{matrix} rs11 & r12 & r13 & tx \\ r21 & rs22 & r23 & ty \\ r31 & r32 & rs33 & tz \\ 0 & 0 & 0 & 1 \end{matrix}$$

scaling matrix

$$\begin{array}{ccccl} 1 & 0 & 0 & 0 & \text{scaling x} \\ 0 & 1 & 0 & 0 & \text{scaling y} \\ 0 & 0 & 1 & 0 & \text{scaling z} \\ 0 & 0 & 0 & 1 & \end{array}$$

shearing matrix

$$\begin{matrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

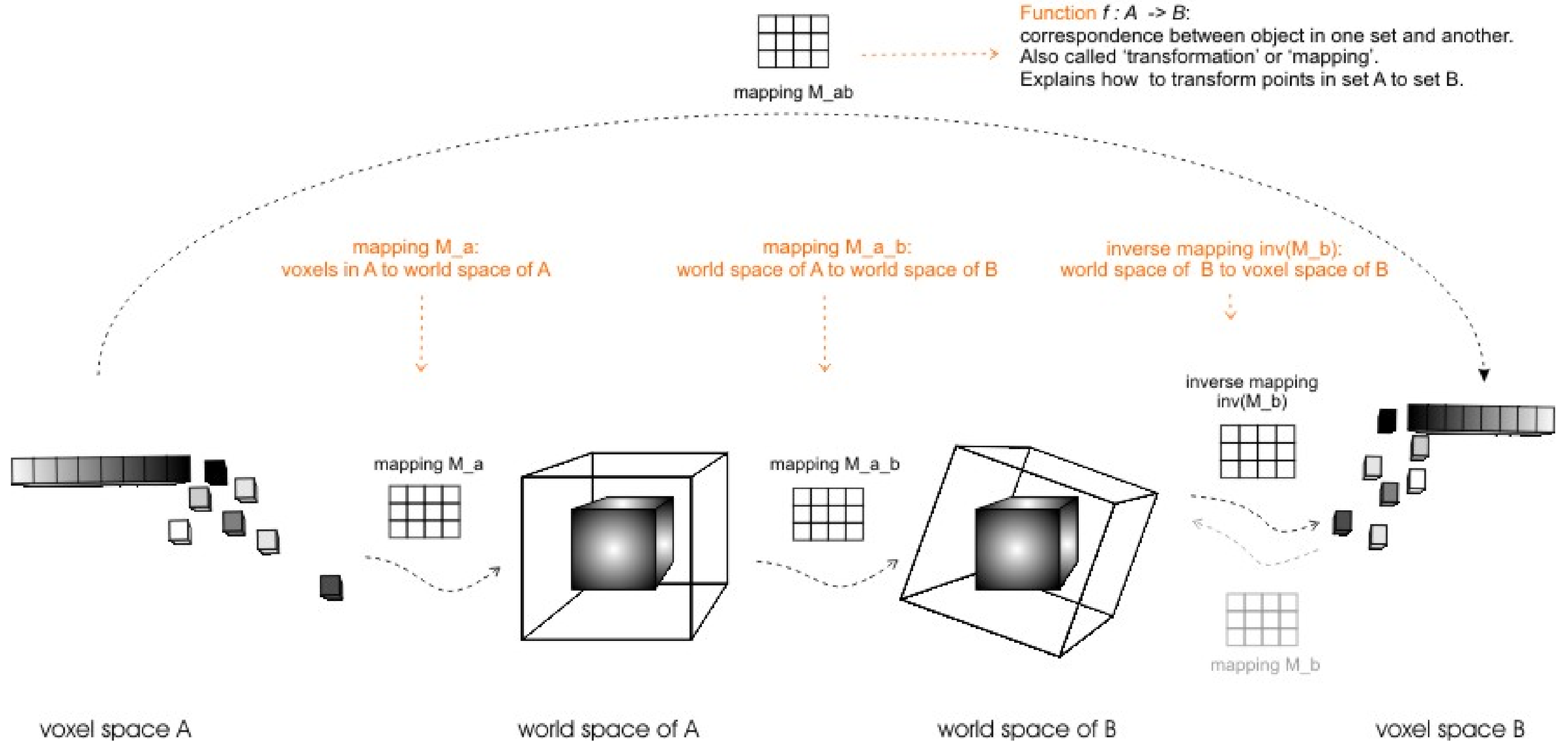
voxel space: position of voxel coordinates i,j,k in index of 1D array
world space: position of voxel coordinates x,y,z in units (f.e. millimeters)

forward mapping: from voxel space to world space
inverse mapping: from world space to voxel space

MAPPINGS BETWEEN COORDINATE SPACES

Mapping from voxels in image A to voxels in image B İnvolves multiplying by transformation matrices:
mapping M_ab = M_b \ M_a_b * M_a = inv (M_b) * (M_a_b * M_a)

Function *f : A -> B*:
correspondence between object in one set and another.
Also called 'transformation' or 'mapping'.
Explains how to transform points in set A to set B.



Bibliography

[1] R.W. Cox et al. A (sort of) new image data format standard: Nifti-1. *Human Brain Mapping*, 25(x):xxx, 2004.

[2] R. Goebel, F. Esposito, and E. Formisano. Analysis of fiac data with brainvoyager qx: From single-subject to cortically aligned group glm analysis and self-organizing group ica. *Human Brain Mapping*, 27(5):392–401, 2006.

[3] P. Thevenaz, T. Blu, and M. Unser. Interpolation revisited. *IEEE Transactions on Medical Imaging*, 19(7):739–758, 2000.