A Definition of Spatial Orientation for the Insight Toolkit

Kent Williams

ITK was conceived by design to be agnostic about image orientation. In other words, the decisions about the relationship between the physical objects depicted in an image and the data representation should be defined by the application, and not by ITK itself. An application that uses ITK has the flexibility to manage the spatial orientation of images in any way that is natural to the problem domain.

But...

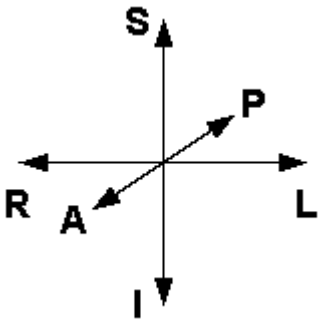
Medical Imaging is an important application of ITK. In this problem domain, it is almost always the case that images represent human anatomy. An image type with no concept of the orientation of the imaging subject – hereafter known as the 'patient,' is essential.

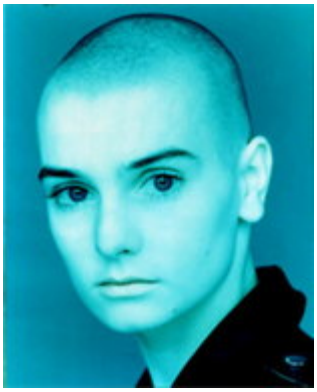
The ITK developers spent a couple of months thinking about how to implement the concept of spatial orientation, and came up with a unambiguous definition and representation, that I will attempt to document here. As a developer at the University of Iowa Department of Psychiatry, my focus is on the spatial orientation of brain anatomy, so that will be the example application. Imaging software that has other subjects as its focus need only define an unambiguous mapping of subject orientation to the definition presented here.

Patient Position

Spatial Orientation in ITK is defined to be consistent with the definition provided by the DICOM imaging standard. This is defined and discussed in the WIKI article "Proposals:Orientation" [1]

Spatial Orientation is defined in terms of the position of a patient. The three spatial dimensions are defined as Patient Left/Patient Right, Inferior (Patient's Feet)/Superior (Patient's Head), and Anterior(Patient's Front)/Posterior(Patient's Back).





These images were borrowed from Graham Wideman's useful web article “Orientation and Voxel-Order Terminology: RAS, LAS, LPI, RPI, XYZ and All That”[2]

Direction Cosines

An ITK Image stores it's spatial orientation in a NxN matrix, where N is the number of dimension in the image. Each column of this matrix is the Direction Cosine for that axis, which is a vector collinear with the axis. This relates to the actual image organization of the image data thusly: each Direction Cosine corresponds to an image dimension, ordered from fastest moving to slowest moving.

This is a generalization from DICOM which stores two direction cosines for the first two dimensions in the file as “Image Orientation (Patient)” (0020,0037). The next dimension is defined as the cross product of the first two. The DICOM standard says that the direction cosines should be orthogonal; when this is the case, the cross product defines the third axis, which aligns with the direction of scanning.

Direction Cosines do not need to be aligned with the unit axis system; they can define an a rotation of the patient's reference frame from the scanner. For example, a patient who is propped so that the patient's body is rotated with respect to the scanner.

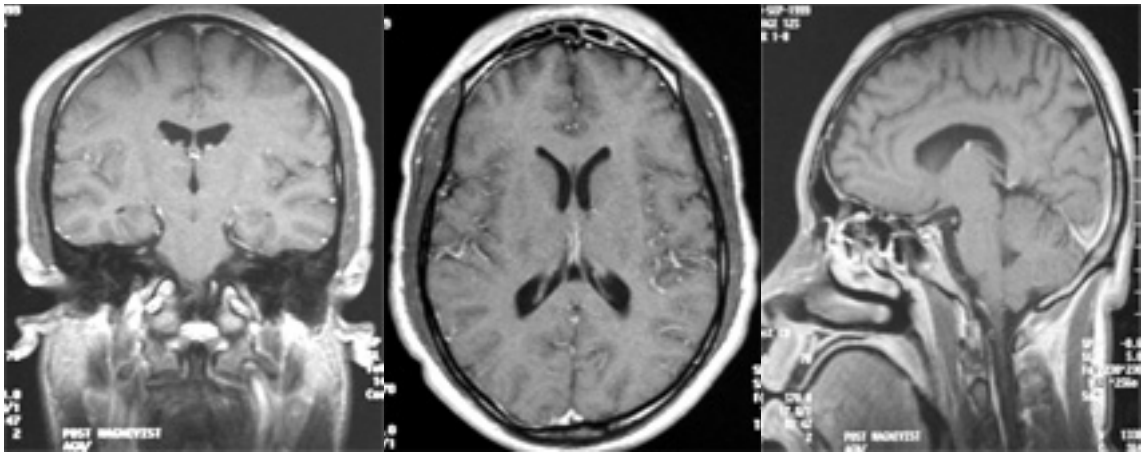
itk::SpatialOrientation

The itk::SpatialOrientation class is a compact shorthand labeling of a subset of all possible spatial orientations, such that each axes of the patient's reference frame is parallel to an axis of the scanner reference frame. This is represented as a three-letter code, indicating the patient orientation in terms of Right/Left, Anterior/Superior, Inferior/Superior. These direction labels refer to the location of the origin with respect to the patient, as opposed to labeling the direction of axes.

For instance, a code of RIP says that the pixel at index [0][0][0] is at the Patient's Right, Inferior, Posterior, and that voxels within the image volume vary fastest from right to left, next fastest from inferior to superior, and slowest from Posterior to Anterior. The

slowest moving index is traditionally thought of as the acquisition plane. In the case of RIP, the slices in an image are acquired by scanning from Posterior to Anterior.

There is no more information in a SpatialOrientation code than in the DICOM-style direction cosines, in fact there is less. SpatialOrientation codes do not indicate any rotation of the patient's orientation with respect to the scanner coordinate system. But itk::SpatialOrientation does map directly onto the orientation information provided by many image volume file formats. For example, Analyze 7.5 files commonly come in three possible orientations, labeled Coronal, Axial, and Sagittal. These correspond to the SpatialOrientation codes RIP, RPI, and PIR, respectively.



Coronal Axial Sagittal

Images borrowed from Wanye State's tutorial web page "Radiologic Anatomy: Brain Module" [3]

Orientation Adapters

itk::SpatialOrientation's codes describe 48 possible data organizations for an Image 3D Volume with respect to the patient orientation. Each of these orientation codes has an equivalent set of direction cosines. For example RAI (indicating Right to Left, Anterior to Posterior, Inferior to Superior) has the equivalent direction cosines [1, 0, 0], [0, 1, 0] , [0, 0, 1].

I wrote the itk::SpatialOrientationAdapter template class to allow conversion of SpatialOrientation codes to and from Direction Cosine matrices. This adapter class is derived from the itk::OrientationAdapterBase class, which defines an interface with two member functions ToDirectionCosines and FromDirectionCosines. Other Orientation Adapter implementation are possible, and probably necessary. For example an adapter that converts quaternions to and from direction cosines would be useful for the NIfTI file I/O routines.

itk::OrientImageFilter

The OrientImageFilter class is contingent on itk::SpatialOrientation, as you specify a given coordinate system for an Image (or specify that the Direction Cosines should be used), and OrientImageFilter will transform the image data into the desired coordinate system. OrientImageFilter is very useful in applications because it allows you to put all images into a common reference frame, with anatomy roughly aligned the same.

Orientation and the NIfTI File Format

As stated above, ITK conforms to DICOM's concept of orientation. NIfTI also has a concise definition of Spatial Orientation, that differs in that you need to reverse the sign of the first two axes in order to describe the same coordinate system. In other words, given the Dicom direction cosines A and B, and the Image Position (Patient) R,

$C = A \, x \, B$ (vector cross product)

$$\text{NIfTI Rotation Matrix} = \begin{bmatrix} -A[0] & -B[0] & -C[0] & -R[0] \\ -A[1] & -B[1] & -C[1] & -R[1] \\ A[2] & B[2] & C[2] & R[2] \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

NIfTI has 2 different methods for describing spatial orientation: The first method, conveniently labeled “Method 1” in the documentation, uses a quaternion to represent orientation, and “Method 2” uses an affine transform. The main difference between the two is that scaling – i.e. voxel spacing – is embedded in the S vectors that represent the transform from pixel space $[i \quad j \quad k]$ to voxel space $[x \quad y \quad z]$

Orientation and the DICOM File Format

Since ITK follows DICOM, the description above should suffice with respect to rotation of the image volume. One difference between DICOM and ITK comes with the definition of the Image Origin. Quoting from comments in the GDCMImageIO source code: DICOM specifies its origin in LPS coordinate, regardless of how the data is acquired.

ITK's origin must be in the same coordinate system as the data. The ITK origin is computed by multiplying the inverse of the direction cosine times the DICOM origin.

Orientation and the Analyze File Format

The Analyze file format[4] is a precursor to the NIfTI file format. The reference implementation NifTI file reader can read and write Analyze format files. Analyze only specifies 3 possible orientations – Coronal, Sagittal, and Axial/Transverse. This poses no problem when reading an image, but when you write an Analyze image, you must reorient the image data into one of those three orientations in order for the file data to match the orientation reported in the header.

Other File Formats

As part of my ongoing work on ITK, I have written several Image I/O classes for other file formats, including GE4X, GE5X, GE/Adw. These were written before the Direction Cosine representation of orientation had been finalized, and currently do not set the orientation in the output image in a conforming manner. This is something I intend to work on in the near future.

Using OrientImageFilter with ImageIO

An application has to have a consistent concept of orientation for any useful process – visualization, registration, and any other operation involving more than one image. Assuming that all file format readers consistently report orientation by way of setting the direction cosines, the OrientImageFilter can be used to ensure that all read images are in a common orientation.

```
template <typename T, unsigned Dimension=3>
typename itk::Image<T,Dimension>::Pointer
DoReadOneITKImage(const std::string &path)
{
typedef typename itk::Image<T,3> ImageType;
typedef typename itk::ImageFileReader<ImageType> ReaderType;
typename ReaderType::Pointer reader = ReaderType::New();

typename ImageType::Pointer theImage;

typedef itk::OrientImageFilter<ImageType,ImageType>
      OrientImageFilterType;

reader->SetFileName(path.c_str());

//
// force image into known orientation
typename OrientImageFilterType::Pointer orienter =
      OrientImageFilterType::New();

orienter->SetInput(reader->GetOutput());
orienter->SetUseImageDirection(true);
```

```
orienter->SetDesiredCoordinateOrientatio
      (itk::SpatialOrientation::ITK_COORDINATE_ORIENTATION_RIP);
//
// read the file in.
try
      {
      orienter->Update();
      theImage = orienter->GetOutput();
      }
catch(itk::ExceptionObject &excp)
      {
      return typename ImageType::Pointer();
      }
return theImage;
}
```

There is a potential problem with this approach: There is an implicit conversion of the direction cosines, which can describe rotations from the scanner coordinate system, to a SpatialOrientation code, which does not. This conversion works by figuring out the dominant direction of each direction cosine, and transforming it to a unit vector. As an example, if the direction cosines describe a rotation, eg

$$\begin{matrix} DirCos_0 = [0.866 \quad -0.5 \quad 0] \\ DirCos_1 = [0.5 \quad .866 \quad 0] \end{matrix} \text{ will be converted to } \begin{matrix} DirCos_0 = [1 \quad 0 \quad 0] \\ DirCose_1 = [0 \quad 1 \quad 0] \end{matrix}$$

In other words, the SpatialOrientation code picks which axis a Direction Cosine is closest to as a function of which axis component's absolute value is largest. The rotation (in this case around the Z axis) of the original direction cosines is lost.

This isn't always a bad thing for some applications. But often, as in the case of registration algorithms, the rotation component of the direction cosines can be used to do a first order, rough alignment of two images. In that case, it would be advantageous to actually rotate the images before applying a more exacting registration technique.

Remaining Work

To fully support orientation in ITK, there remains some work to be done in the ImageIO classes, as they do not all support setting the Direction Cosines properly when reading a file, or use them properly to set the file format's native orientation information when writing a file.

Aside from File I/O, orientation needs to be considered when writing filters. The ITK filters that explicitly re-orient image data – OrientImageFilter, FlipAxesFilter, and PermuteAxesFilter – do set the direction cosine for output images properly. But how each ITK filter affects the image orientation should be reviewed. It isn't always obvious whether the filter should modify the direction cosines of the output image.

A Data Set for Testing Orientation

As an adjunct to this article, I have created a set of NifTI images which all contain the same image data. Each file is in one of the 48 possible orientations for an image volume. The original image was downsampled from an example image volume, in which the letters 'L' and 'R' have been embedded, to show which side of the brain image is patient left and patient right.

1 ITK Wiki - "Proposals:Orientation"
http://www.med.wayne.edu/diagRadiology/Anatomy_Modules/brain/brain.htm – ongoing discussion about handling Orientation in the Insight Toolkit.

2Graham Wideman' “Orientation and Voxel-Order Terminology: RAS, LAS, LPI, RPI, XYZ and All That” http://www.grahamwideman.com/gw/brain/orientation/orientterms.htm – A nice primer on orientation in medical imaging; source of the figure featuring Sinead O'Connor.

3 Jeffrey E. Zapawa and Anthony L. Alcantra, MD "Radiologic Anatomy: Brain Module"
http://www.med.wayne.edu/diagRadiology/Anatomy_Modules/brain/brain.html – Tutorial on brain anatomy, with brain regions defined and labeled.

4 Graham Wideman “Mayo/SPM “Analyze” Format Spec Compilation” http://wideman-one.com/gw/brain/analyze/formatdoc.htm – a reference guide for the Analyze7.5 file format.