Medical Image Orientation in VTK

Release 1.2

D. G. Gobbi[1]

May 21, 2013

[1]Department of Radiology, University of Calgary, Calgary, Canada

Abstract

Although VTK provides all of the tools that are necessary to correctly utilize medical coordinate systems, it can, in practice, be difficult to manage these coordinate systems within a VTK display pipeline. The VTK image reader and display classes behave contrary to what is expected by the DICOM standard, and as a result, these classes require special consideration if they are to be used with DICOM images. Due to misunderstanding of the ways in which VTK handles image coordinate systems, it is often the case that a programmer will write a pipeline that does not use DICOM coordinates but instead uses a different coordinate system. We present a system for describing and applying medical image orientation information that allows DICOM patient coordinates to be used throughout VTK, and argue that non-DICOM coordinate systems should be avoided when processing DICOM images.

Contents

1 Introduction

VTK can be a powerful tool for medical image visualization, but due to its size and complexity it can be difficult to navigate for programmers who are not familiar with it. If the correct steps are not taken when designing a VTK application, it can be hard to achieve the desired results. Working with DICOM images in VTK is an area where a novice VTK programmer without guidance will be certain to run into great difficulty. The problems begin as soon as the DICOM image has been loaded into VTK, because the native DICOM reader that comes with VTK will flip the image into VTK's default bottom-to-top orientation, instead of maintaining DICOM's standard top-to-bottom orientation. The VTK image viewers, as well, provide default view orientations that are counter to what is expected by radiologists. All of these issues can be avoided by using the correct settings for the various VTK classes that access or process the DICOM data, but finding the correct settings requires not only knowledge of VTK and DICOM, but also some mathematics in order to reconcile certain VTK conventions with the DICOM standard.

The purpose of this document is to discuss issues related to medical image orientation in VTK. It is not a proposal for adding an orientation matrix to the image data class as was done for ITK. Instead, what we present here are a set of procedures for managing image orientation with the current VTK release, specifically VTK 5.10. In order to restrict the scope of this document to something manageable, it is assumed that the DICOM images are MRI or CT or from a similar modality, and will be reconstructed into a 3D volume when they are loaded into VTK. Each VTK image will then correspond to a series of DICOM images (temporal data is a special case, since VTK images have no time dimension — each time point will become its own VTK image). The reason for performing the reconstruction is that, without reconstruction, it is impossible to perform volume rendering, 3D contouring, or reformatting for different view orientations. All of the DICOM readers that are typically used with VTK perform reconstruction, but they are not always consistent with how they utilize the metadata to ensure that the reconstruction is performed correctly, nor are they consistent with how they store the metadata following the reconstruction.

We strongly recommend that when DICOM data is brought into VTK, that the integrity of the data is preserved (no flips or permutations) and that the VTK coordinate systems be matched to the DICOM coordinate systems. This means that the VTK image data coordinate system should be matched with the DICOM image plane coordinate system (i.e. without the orientation or image position applied) and the VTK world coordinate system matched with the DICOM patient coordinate system. This greatly simplifies the process of combining data sets in VTK, for example when DICOM dose contours must be superimposed on a CT scan. In the case where separate image scans must be fused through image registration, the patient coordinate system of one scan should be kept as the primary coordinate system, while the patient coordinate system of the other is modified in order to achieve fusion.

Another recommendation is that programmers avoid using the VTK imaging pipeline to reformat the data as part of a visualization pipeline. The VTK rendering classes (meaning the mappers, actors, and widgets) are able to display reformatted views while leaving the data intact. Relying on these rendering classes will greatly simplify data management within an application, and can accelerate the image display since some operations can be offloaded onto the GPU.

At the end of this document, we examine the implementations of some existing VTK classes for importing DICOM images and provide examples of how they can be used with the DICOM data handling procedures that we provide here.

1.1 DICOM Coordinate System

The DICOM standard describes a 3D patient coordinate system in terms of the major axes of the body. In this coordinate system, the $+x$ axis points left, the $+y$ axis points in the posterior direction (towards the back), and the $+z$ axis points in the superior direction (towards the head). These axes are illustrated in Figure 1. The DICOM patient coordinate system is often referred to by the acronym LPS (Left, Posterior, Superior), which refers to the directions of the $+x$, $+y$, and $+z$ axes in this coordinate system.

For MR, CT, and other modalities that provide images of slices within the body, all of the information required to correctly position the DICOM image slice within the patient coordinate system is described in DICOM Part 3 Annex C Section 7.6.2, Image Plane Module[1]. The following table lists the tags that will always be present for this module:

Name	Tag	Parameters	Units
PixelSpacing	(0028,0030)	s_x, s_y	mm
ImageOrientationPatient	(0020,0037)	$u_x, u_y, u_z, v_x, v_y, v_z$	unitless
ImagePositionPatient	(0020,0032)	p_x, p_y, p_z	mm

It is assumed we are in the simple case where only Pixel Spacing (0028,0030) is present, otherwise see DICOM Part 3, Section 10.7.1.1. The parameters in this table, which are illustrated in Figure 2, describe the position of the center of each DICOM pixel (i, j) in patient coordinates according to the following equation:

$$\begin{pmatrix} x_{\text{patient}} \\ y_{\text{patient}} \\ z_{\text{patient}} \end{pmatrix} = \begin{pmatrix} u_x & v_x \\ u_y & v_y \\ u_z & v_z \end{pmatrix} \begin{pmatrix} s_x\, i \\ s_y\, j \end{pmatrix} + \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \tag{1}$$

The DICOM Image Plane Module states that the vectors $\mathbf{u}$ and $\mathbf{v}$ must form an orthonormal pair, i.e.:

$$|\mathbf{u}| = 1, \qquad |\mathbf{v}| = 1, \qquad \mathbf{u} \cdot \mathbf{v} = 0$$

Also, according to DICOM Part 3 Section 10.7.1.3, the pixel spacing s_x, s_y must be non-negative:

$$s_x \geq 0, \qquad s_y \geq 0$$

Either s_x or s_y can be zero if the image consists of a single column or a single row, but otherwise they must both be greater than zero. Equation 1 can be extended to an image volume with multiple slices by adding a slice index k,

$$\begin{pmatrix} x_{\text{patient}} \\ y_{\text{patient}} \\ z_{\text{patient}} \end{pmatrix} = \begin{pmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{pmatrix} \begin{pmatrix} s_x\, i \\ s_y\, j \\ s_z\, k \end{pmatrix} + \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \tag{2}$$

where (p_x, p_y, p_z) is the position of the first slice in the series, i.e. the slice for which $k = 0$. Unlike the previous equation, this volumetric equation is not normative in the DICOM standard and does not come with the same assurances of positive spacing and orthogonality. Under certain circumstances the slice spacing s_z can be negative (e.g. DICOM Part 3 Annex C Section 8.4.15, NM Reconstruction Module), and the orientation vector $\mathbf{w}$ is not always orthogonal to $\mathbf{u}$ and $\mathbf{v}$ (e.g. a tilted CT gantry results in slices that are not perpendicular to the scan direction).

[1] ftp://medical.nema.org/medical/dicom/2011/11_03pu.pdf

A method for converting a series of DICOM slices into an oriented volume in accordance with Equation 2 is described in Section 2.1. The conversion of Equation 2 into a format that can be applied to image orientation within VTK is described in Section 2.2.

1.2 VTK Coordinate Systems

Each portion of the image display process in VTK has a different coordinate system associated with it. The chain of coordinate conversions, as an aggregate, provide the mapping from image pixels (or voxels) to screen pixels within the viewport (in computer graphics, the viewport is the rectangle within the screen or window to which images are rendered).

Coordinate System	Purpose
Structured Coords	Sample (i, j, k) indices used by image data
Data Coords	Used by data objects and data filtering operations
World Coords	Used for 3D rendering operations
View Coords	Coordinates from the viewer's perspective
Display Coords	The (X, Y) screen pixel locations in the viewport

The image display process can be roughly explained as follows:

1. Structured coordinates are scaled by the voxel spacing to give data coordinates.
2. Data coordinates are rotated and translated to give world coordinates.
3. World coordinates undergo a projection to give view coordinates.
4. View coordinates are scaled to the viewport size to give display coordinates.

In VTK, the "world" is composed of multiple data sets, each of which has its own orientation and position within the world. One can also define multiple viewpoints, each of which will be displayed in a different window or viewport on the screen. When using VTK with DICOM images, it is best to think of the world as the patient, i.e. one will have multiple data sets for the patient, each of which will be correctly positioned within the patient coordinate system. The patient will then be viewed from three different viewpoints to provide the standard axial, coronal, and sagittal radiological slice orientations.

Structured Coordinates and Data Coordinates

Each sample in a VTK image can be identified by its indices (i, j, k). These are called the structured coordinates of the image. The indices are generally integers used to uniquely identify a single sample point, but fractional indices may be used to identify locations between the sample points. The structured coordinates are converted to VTK data coordinates via the following equation:

$$\begin{pmatrix} x_{\text{data}} \\ y_{\text{data}} \\ z_{\text{data}} \end{pmatrix} = \begin{pmatrix} s_x i + o_x \\ s_y j + o_y \\ s_z k + o_z \end{pmatrix} \tag{3}$$

This equation utilizes pixel and slice spacings s_x, s_y, s_z with exactly the same definitions as for DICOM, and also includes a offset (o_x, o_y, o_z) called the VTK image origin. It is important to note that this offset is

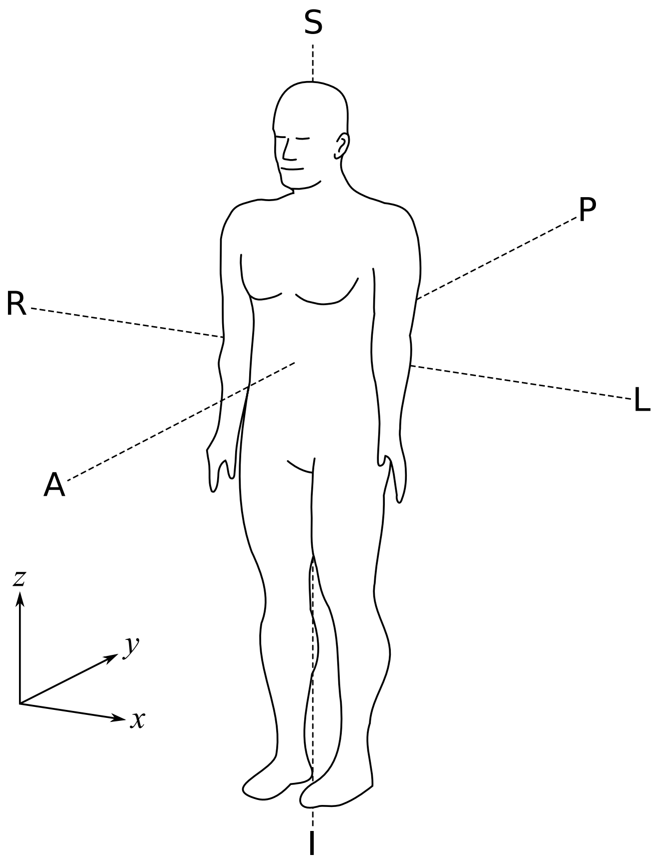

Figure 1: The DICOM patient coordinate system. Also see DICOM Part 17 Annex A.

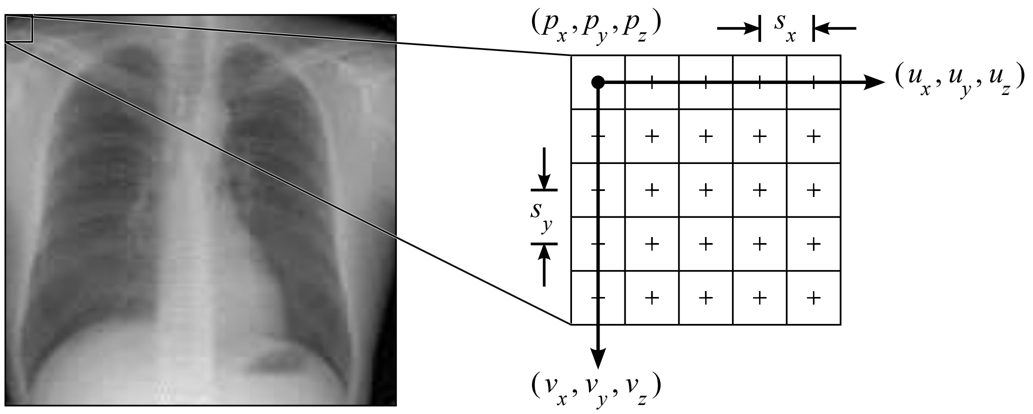

Figure 2: The DICOM image plane.

different from the DICOM slice position (p_x, p_y, p_z). The difference is as follows: the VTK image origin is aligned with the rows and columns of the image, while the DICOM position is aligned with a coordinate system that is *rotated* with respect to the rows and columns of the image. A full description of the relationship between (o_x, o_y, o_z) and (p_x, p_y, p_z), and of the relationship between DICOM coordinates and VTK image data coordinates in general, is provided in Section 2.2.

World Coordinates

The world coordinate system is VTK's primary coordinate system. Its units and orientation can be chosen according to the data that is being visualized. For viewing DICOM images, the world coordinate system can be defined to be the patient coordinate system. This will mean that the VTK camera, the VTK picker, the VTK actors, and any other VTK entities whose positions are expressed in world coordinates will operate within the DICOM patient coordinate system.

All data coordinates are converted to world coordinates when the data is rendered. To facilitate this, each vtkActor, vtkVolume, or vtkImageSlice (all of which are subclasses of vtkProp3D) carries a 4×4 matrix $\mathrm{M}_{\mathrm{prop}}$ that relates data coordinates to world coordinates:

$$\begin{pmatrix} x_{\mathrm{world}} \\ y_{\mathrm{world}} \\ z_{\mathrm{world}} \\ 1 \end{pmatrix} = \mathrm{M}_{\mathrm{prop}} \begin{pmatrix} x_{\mathrm{data}} \\ y_{\mathrm{data}} \\ z_{\mathrm{data}} \\ 1 \end{pmatrix} \tag{4}$$

For DICOM, this 4×4 matrix should usually be chosen to be a matrix that converts from VTK image data coordinates to DICOM patient coordinates, in order to enforce the equality between world coordinates and patient coordinates. This is a flexible rule, however, given that the world coordinate system can be chosen to be whatever works best for the task at hand.

View Coordinates and Display Coordinates

In VTK, in order for data to be rendered it is necessary to place a virtual camera within the VTK world. For example, to obtain an axial view, one would place a camera at the patient's feet looking towards the patient's head. The VTK view coordinates are analogous to the film in the camera: the picture that is captured on the film depends on where the camera is, where it is pointing, and what the zoom factor is set to.

View coordinates range from -1 to $+1$ horizontally and vertically, with $(0,0)$ at the center. The center corresponds to the camera's focal point, i.e. where the camera is looking, while and the height and width correspond to the region of the world that is framed by the camera. In addition, view coordinates have a third component, depth, that increases as the distance from the camera increases.

Display coordinates are a result of mapping the view to the screen. These coordinates are measured from the outer corners of the screen pixels, so the center of the lower, left screen pixel in the viewport is at display position of $(\frac{1}{2}, \frac{1}{2})$ while the center of the upper, right screen pixel is at display position $(w - \frac{1}{2}, h - \frac{1}{2})$, where w and h are the number of screen pixels across the width and height of the viewport.

2 Methods

Managing DICOM image orientation in VTK involves three separate elements:

1. reconstructing the DICOM image slices into a 3D VTK image,
2. building a coordinate transformation to relate the VTK image to DICOM patient coordinates, and
3. adjusting the VTK camera to generate the desired views of the patient.

2.1 Reconstructing a volume from DICOM image slices

For the vast majority of DICOM CT/MR acquisitions, each slice is stored in a separate file with its own DICOM metadata. This restriction is due to the fact the MR Image Storage and CT Image Storage, respectively 1.2.840.10008.5.1.4.1.1.4 and 1.2.840.10008.5.1.4.1.1.2 do not allow anything but 2D image. Newer CT/MR modalities are not using Enhanced MR Image Storage and Enhanced CT Image Storage SOP Classes. Those newer classes allow storing of Multiframe images within a single file. One should pay attention that DICOM Multiframe simply means multiple 2D slice within a single file, this does not imply a particular 3D convention (eg. image orientation can be different in between each slices of the same file). Some other well known exceptions to this are nuclear medicine scans (the NM modality) and 3D ultrasound scans, which store all of the slices within a single file. Since those are the exceptions rather than the rule, this section will focus on reconstruction from slices stored in individual files. The reconstruction is usually performed by a VTK reader class, such as the vtkDICOMImageReader or the vtkGDCMImageReader[2].

As described in Section 1.1, each DICOM slice file has its own ImagePositionPatient (p_x, p_y, p_z) and ImageOrientationPatient $(u_x, u_y, u_z, v_x, v_y, v_z)$ stored in its metadata. Together with the PixelSpacing, these provide all of the information that is necessary to reconstruct the slices into an image volume. If we assume that the slices will have the same orientation and will be evenly spaced, then the reconstructed volume will be described by Equation 2:

$$\begin{pmatrix} x_{\text{patient}} \\ y_{\text{patient}} \\ z_{\text{patient}} \end{pmatrix} = \begin{pmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{pmatrix} \begin{pmatrix} s_x i \\ s_y j \\ s_z k \end{pmatrix} + \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}$$

Performing the reconstruction involves three things: sorting the slices to ensure they are in the correct order, finding the slice spacing s_z, and computing the final column w_x, w_y, w_z of the orientation matrix. All three of these can be accomplished by examining how the ImagePositionPatient (p_x, p_y, p_z) varies from one slice to the next:

$$\mathbf{p}_k = \mathbf{p}_0 + k s_z \mathbf{w} \tag{5}$$

Note that k is not the image InstanceNumber tag (0020,0013), it is a number that we assign to the images ourselves. We define our first slice ($k = 0$) to be at one of the ends of the volume, but the choice of which

[2] http://gdcm.sourceforge.net/html/classvtkGDCMImageReader.html

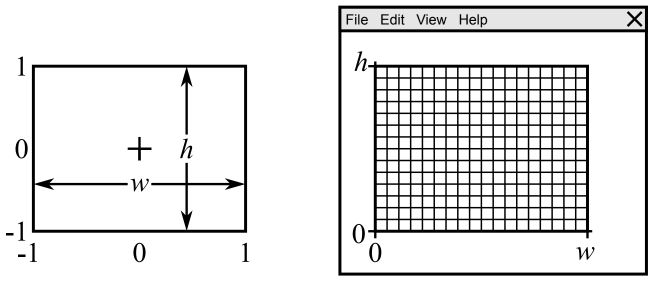

Figure 3: VTK view coordinates (left) and display coordinates (right) within the viewport.

end to use is not arbitrary; we want to chose the ordering such that the spacing s_z is positive, while the direction $\mathbf{w}$ is constrained such that the axes $\mathbf{u}, \mathbf{v}, \mathbf{w}$ form a right-handed coordinate transformation, i.e. the determinant of the 3×3 direction cosines matrix will be positive. Due to these constraints, our numbering of the slices may be opposite to that provided by the InstanceNumber for some acquisitions.

The following steps can be used to assign the ordinal number k to each slice in a DICOM series, and to compute the parameters $\mathbf{w}$ and s_z that are needed for Equation 5, while enforcing that $\mathbf{w}$ and s_z follow the constraints described in the previous paragraph. The only metadata required for this procedure is the ImageOrientationPatient and ImagePositionPatient for each slice:

1. Verify that $\mathbf{u}$ is the same for each image in the series. Do the same for $\mathbf{v}$.
2. Compute a vector perpendicular to the slices $\mathbf{r} = \mathbf{u} \times \mathbf{v}$.
3. For each slice k, compute an offset along the perpendicular $d_k = \mathbf{r} \cdot \mathbf{p}_k$.
4. Sort the slices using the offset d_k, from the smallest d to the largest.
5. Compute the spacing $s_z = |\mathbf{p}_{(n-1)} - \mathbf{p}_0|/(n-1)$ where n is the number of slices.
6. Verify the space between adjacent slices $|\mathbf{p}_{(k+1)} - \mathbf{p}_k| = s_z$ for all k in range $[0, n-2]$.
7. Compute $q_x = \frac{\Delta p_x}{\Delta d}$ via linear regression, do the same for q_y and q_z. Then compute $\mathbf{w} = \mathbf{q}/|\mathbf{q}|$.

It is worth repeating that the slice ordinal k will not necessarily match the InstanceNumber tag (0020,0013). It is therefore wise to keep a mapping between k and the original DICOM file containing that slice, or at minimum between k and the DICOM metadata for that slice.

The checks in Step 1 and Step 6 should include a reasonable tolerance, for example a tolerance of 10^{-4} on the orientation vectors and a tolerance of 0.1 mm on the slice spacings. Higher resolution images should use proportionately smaller tolerances. The reason for this tolerance is that the orientation and position readouts from the scanner might fluctuate due to noise. If the slices were not evenly spaced, then Step 5 was not valid. It may be replaced by a step that computes a separate offset l_k for each slice, using the equation $l_k = \mathbf{w} \cdot (\mathbf{p}_k - \mathbf{p}_0)$. The slice positions will then be given by the following equation, instead of by Equation 5:

$$\mathbf{p}_k = \mathbf{p}_0 + l_k \mathbf{w} \tag{6}$$

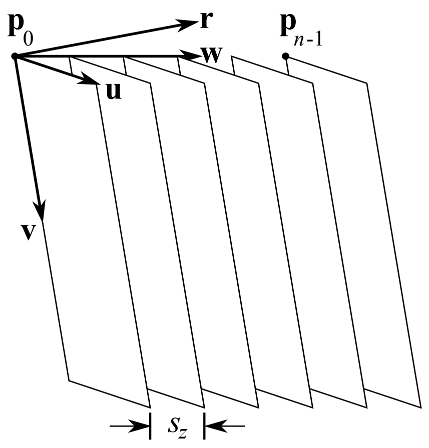

Figure 4: Reconstructing a volume from slices.

The offset l_k is analogous to the optional SliceLocation tag (0020,1041), but is not necessarily equal to it because the reference slice may be different — the DICOM standard does not specify which slice is used as a reference for the SliceLocation. Note that variable spacing is rare, it will only occur if a slice is missing, or if the series is a CT protocol that varied the spacing in order to reduce the radiation dose to the patient.

In Step 7, if the the slice direction vector $\mathbf{w}$ is not equal to the slice perpendicular vector $\mathbf{r}$, then the image stack is sheared. The angle between the vectors is the tilt (or shear) angle, which can be computed from the dot product of the vectors:

$$\cos\theta = \mathbf{r} \cdot \mathbf{w} \tag{7}$$

Sheared volumes are rare for MR, but are common for CT and indicate that the slices were collected with a tilted gantry. If the tilt angle is larger than one or two degrees, it might be necessary to reject the images. If no sheared volumes will be encountered by an application, then $\mathbf{w} = \mathbf{r}$ and Step 7 can be eliminated.

The SpacingBetweenSlices tag

A crucial point of the preceding discussion is that reconstruction of a CT or MR image volume from DICOM slices can, and should, be done using only the information in the PixelSpacing, ImageOrientationPatient, and ImagePositionPatient tags. The use of any other metadata for the reconstruction is at best redundant, and at worst ambiguous or incorrect. The only instances in which other metadata should be used for reconstruction are when the image series includes non-spatial dimensions such as time, in which case the discussion above is still valid but must be extended to account for the extra dimensions.

For MR images, the SpacingBetweenSlices (0018,0088) tag should not be relied upon for volume reconstruction, at best it should just be checked against the value of s_z that was computed with the method described above. In the DICOM standard, this tag is described only briefly in Part 3 C.8.3.1 MR Image Module (Table C.8-4 MR IMAGE MODULE ATTRIBUTES), which states that it is an optional tag equal to the distance between the centers of adjacent slices. The DICOM CT Module does not state anything about this tag whatsoever, and CT slices are often collected with varying spacing in order to minimize dose to the patient. The SliceThickness and SliceLocation tags should also be avoided when doing volumetric reconstructions, for the reasons given in the table below.

Name	Tag	Reason to avoid
SpacingBetweenSlices	(0018,0088)	may not be present, slice order is ambiguous
SliceThickness	(0018,0050)	slice thickness is unrelated to reconstruction
SliceLocation	(0020,1041)	reference point and direction are ambiguous

Multiple slices in a single DICOM file

The one case where the SpacingBetweenSlices value is necessary for a reconstruction is when multiple slices are stored in a single DICOM file, as is for NM (nuclear medicine). In such files, only the first slice has an ImagePositionPatient value and it is therefore impossible to compute s_z and $\mathbf{w}$ with our seven-step sorting method. For these images, $\mathbf{w} = \mathbf{u} \times \mathbf{v}$ and s_z is given by the SpacingBetweenSlices tag (0018,0088), as described in DICOM Part 3 Annex C Section 8.4.15, NM Reconstruction Module. The standard explicitly states that the spacing may be negative, so any application that reads NM images must be equipped to handle negative slice spacing. The previous statement can also be applied for a subset of Enhanced MR Image Storage and Enhanced CT Image Storage instances.

Per-slice intensity mapping

Some modalities, notably CT and PT (positron emission tomography or PET), have RescaleIntercept (0028,1052) and RescaleSlope (0028,1053) in their metadata for use in calibrating the image intensity values. For CT, these values allow conversion of the stored pixel values into Hounsfield units. For PET, the situation is very complex because PET images can use a large variety of units. Furthermore, for some PET scans the RescaleSlope will vary from slice to slice. When this occurs, it is necessary to rescale the intensity of each slice when constructing a volume, otherwise reformatting or volume rendering will not be possible.

A DICOM reader that is not equipped to handle per-slice intensity calibration should not be used with PET images. For more information on intensity calibration and lookup tables, see DICOM Part 3 Annex C Section 11, Look Up Tables and Presentation States. Also see the notes on the RescaleIntercept and RescaleSlope for the individual modalities that are described by the standard.

2.2 Relating VTK image data coordinates to DICOM patient coordinates

The recommended method

Once the DICOM slices have been read from disk and used to construct a VTK image data set, it is necessary to relate the data coordinates to DICOM patient coordinates via a VTK 4×4 matrix transformation. This can be done by simply converting Equation 2 into homogeneous coordinates, resulting in the following equation:

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} u_x & v_x & w_x & p_x \\ u_y & v_y & w_y & p_y \\ u_z & v_z & w_z & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_x i \\ s_y j \\ s_z k \\ 1 \end{pmatrix} \tag{8}$$

Note that (o_x, o_y, o_z) does not appear in this equation. In order to relate VTK data coordinates to DICOM patient coordinates via Equation 8, one *must* set the VTK image origin to zero:

$$o_x = 0, \qquad o_y = 0, \qquad o_z = 0$$

In addition to requiring that the origin of the data from the vtkDICOMImageReader is set to zero (e.g. through the use of vtkImageChangeInformation), it also requires that the reader does not flip the image image into a so-called “VTK friendly” bottom-to-top orientation. Our recommended three-step method for handling DICOM coordinates in VTK are to undo any vertical flip applied to the data by the reader, set the origin to zero, and build the above 4×4 matrix from the DICOM metadata.

Encoding the position in the origin

Setting the origin to zero is not the only option, it is possible to encode the position in the origin instead of including the position in the 4×4 matrix as above. If we do this, then the following equation will convert VTK image data coordinates to DICOM patient coordinates:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{pmatrix} \begin{pmatrix} s_x i + o_x \\ s_y j + o_y \\ s_z k + o_z \end{pmatrix} \tag{9}$$

By inspection of Equation 2, we see that in order for the above equation to be true, the VTK image data origin must be computed from the DICOM image position as follows:

$$\begin{pmatrix} o_x \\ o_y \\ o_z \end{pmatrix} = \begin{pmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{pmatrix}^{-1} \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \tag{10}$$

Unfortunately, it is very common for programmers to assume equality between the origin and position, resulting in an offset to the computed patient coordinates. The reasons that such mistakes often go unnoticed are twofold: 1) the majority of MR and CT scans are done in the axial orientation, where the orientation matrix is identity or very close to identity, 2) most software testing is qualitative and while orientation errors are easily noticed, position offset errors are not. Examples of widely-used VTK classes that erroneously set the origin to the position are the vtkGDCMImageReader and itkVTKImageExport. The latter assumes that the ITK image origin is equivalent to the VTK image origin, while the ITK image origin is in fact analogous to the DICOM position.

A more general equation

Both of the methods described above, the former including the position in the matrix and the latter encoding the position in the origin, are generalizations of a broader relationship between VTK image coordinates and DICOM patient coordinates:

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} u_x & v_x & w_x & q_x \\ u_y & v_y & w_y & q_y \\ u_z & v_z & w_z & q_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_x i + o_x \\ s_y j + o_y \\ s_z k + o_z \\ 1 \end{pmatrix} \tag{11}$$

where

$$\begin{pmatrix} q_x \\ q_y \\ q_z \end{pmatrix} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} - \begin{pmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{pmatrix} \begin{pmatrix} o_x \\ o_y \\ o_z \end{pmatrix} \tag{12}$$

If the VTK image origin is set to zero, then this is equivalent to Equation 8. If (q_x, q_y, q_z) is set to zero, then it becomes equivalent to Equation 9.

Comments about row ordering of images

As discussed in Section 1.2, VTK's Display Coordinates have their origin in the lower left corner of the viewport. In the early days of VTK, the image pixels were drawn directly to screen pixels by the 2D rendering pipeline, and any images that were stored in a top-to-bottom row ordering would be drawn upside down. As a result, the VTK image readers were written so that they would vertically flip any images that were stored with top-to-bottom row ordering on disk, so that they would be drawn in the correct orientation.

Unlike the VTK 2D rendering pipeline, the VTK 3D rendering pipeline can handle either ordering equally well, as will be discussed in the following section. Even with the 2D rendering pipeline, top-to-bottom images can be rendered correctly by applying a vertical flip immediately before the image is rendered. We therefore recommend that one always uses the original DICOM row ordering. Most VTK image reader classes support a method called FileLowerLeftOn() that causes the reader to keep the original top-to-bottom row ordering. For readers like the vtkDICOMImageReader that ignore this method, the vtkImageFlip filter can be used to return the image to its original ordering.

If one does not heed our advice, and allows the reader to flip the image, then certain changes must be done to the DICOM orientation metadata (the ImagePositionPatient and the ImageOrientationPatient) if pixel index $(0,0)$ is no longer at the upper right as shown in Figure 2). These changes amount to computing a new position for the lower left pixel and reversing the direction of the column orientation vector, so they are not very difficult to do. However, we believe that any unnecessary modifications to either the data or the metadata, no matter how trivial or temporary, are unwise.

We should also note that the vtkImageViewer2 class, which is used in many VTK examples, is incapable of providing correct views of top-to-bottom ordered data. This class only supports a very limited number of view orientations and lacks the flexibility necessary for displaying oriented DICOM images in VTK. More suitable methods for displaying oriented images are discussed in the next section.

2.3 Setting view orientations

It is assumed that, by this point, a matrix has been created to convert VTK image data coordinates to DICOM patient coordinates. All of the vtkProp3D classes (which include vtkVolume, vtkImageSlice, and vtkImageActor) have a SetUserMatrix() method that can be used to automatically use this matrix to correctly position the image within VTK's world coordinate system, which we have defined to be equal to DICOM's patient coordinate system. The next step is to instruct the VTK camera to view the world (or patient) from the desired viewpoint, so that when VTK renders to the computer screen, the expected view orientation will appear.

At this point, some readers will be thinking that an important step is missing: at what stage of the rendering process is the image data reformatted (or resliced) to match the desired view orientation? Fortunately, this step is taken care of automatically by the vtkImageResliceMapper class, which can communicate with the vtkCamera, and then extract and render the slice that matches the camera's orientation and that passes through the camera's focal point. Starting from VTK 5.10, the programmer no longer has to write his or her own pipeline to reslice the image prior to rendering it.

Given that the view is purely a function of the camera parameters, it is important to understand those parameters (though a simplified method of setting view orientations will be discussed in the next section). Each of the parameters in the table below is illustrated in Figure 5:

Method Name	Parameters	Description
SetFocalPoint()	f_x, f_y, f_z	The point of interest, it will be centered in the view
SetPosition()	c_x, c_y, c_z	Location of the camera itself
SetViewUp()	$\mathbf{v}_{\text{up}}$	Direction that will be "up" on the screen
SetClippingRange()	$d_{\text{near}}, d_{\text{far}}$	Near and far clipping planes
SetParallelProjection()	1 or 0	Whether to use a parallel projection
SetParallelScale()	h	View height, only for parallel projections
SetViewAngle()	α	Vertical view angle, only for perspective projections

These parameters, and the concept of a "virtual camera," can be counterintuitive when the goal is to display medical images, for example, the near and far clipping planes define a volume rather than a slice. It would be preferable to set up the view in terms of the image orientation, position, field of view, and slice number. Fortunately, it is not difficult to compute the camera parameters from these more intuitive parameters. In Figure 5, the camera has been labeled with orientation vectors $\mathbf{v}_{\text{right}}, \mathbf{v}_{\text{up}}, \mathbf{v}_{\text{out}}$ where "right", "up", and "out" refer to directions with respect to the computer screen on which the images will be shown, as illustrated for

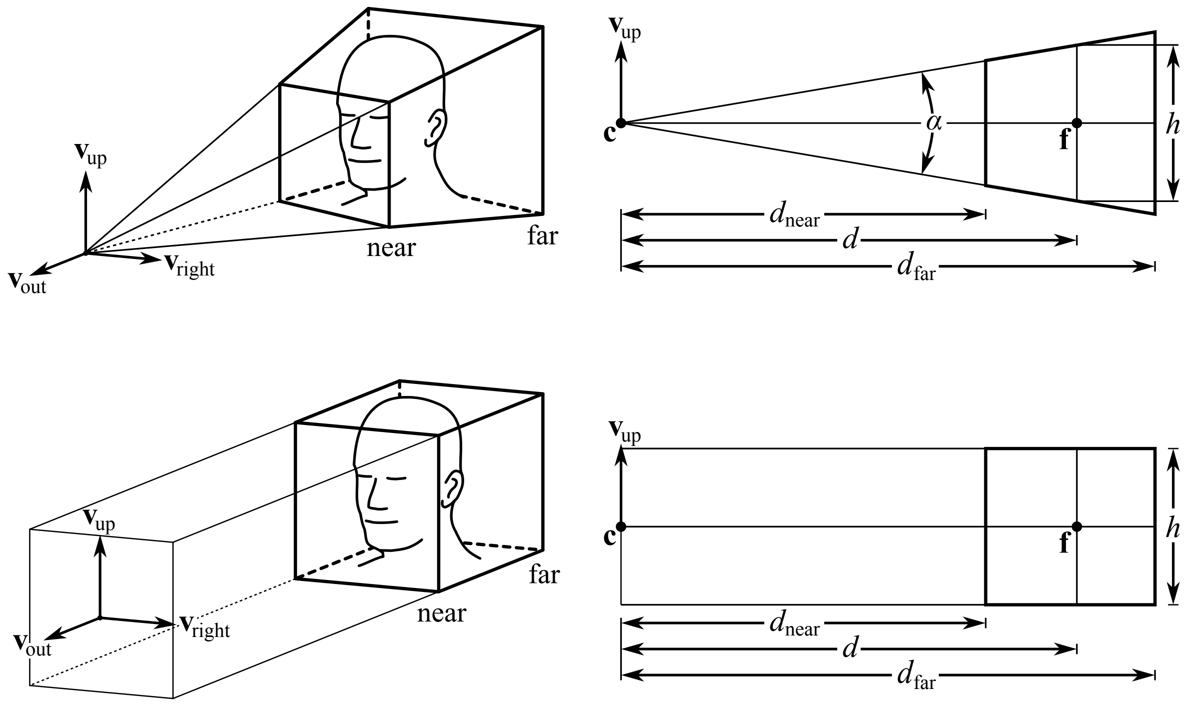

Figure 5: Perspective projection (top) and parallel projection (bottom), together with the related camera parameters (right). The parameter h refers to the height of the viewport at the focal plane, measured in millimeters in the direction of the vector $\mathbf{v}_{\text{up}}$. These figures describe the view projections used in VTK, and are unrelated to the projective or tomographic geometries that were used to acquire the images.

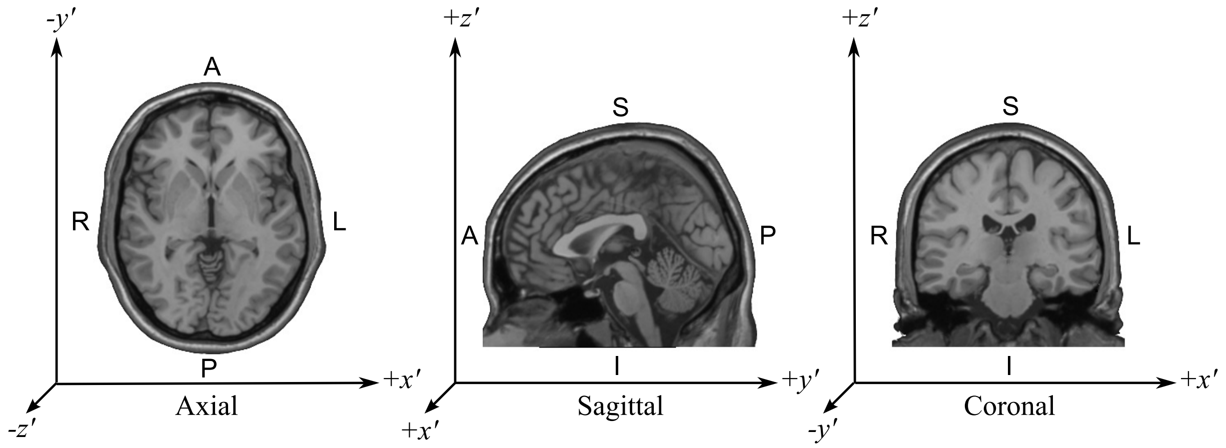

Figure 6: Standard head views used in radiology. The axes are are marked with a prime ($'$) because DICOM slices are often scanned and viewed in an orientation that is slightly oblique to the patient coordinate system.

typical radiological views in in Figure 6. If one simply wants to view an image in its original scan orientation, then one would use the ImageOrientationPatient values $[u_x, u_y, u_z, v_x, v_y, v_z]$, noting that the camera's $\mathbf{v}_{\text{up}}$ is in the opposite direction from the DICOM column vector $\mathbf{v}$:

$$\mathbf{v}_{\text{right}} = \mathbf{u}, \qquad \mathbf{v}_{\text{up}} = -\mathbf{v}, \qquad \mathbf{v}_{\text{out}} = \mathbf{v}_{\text{right}} \times \mathbf{v}_{\text{up}} \tag{13}$$

The displayed slice should be the slice at the focal plane, and the camera focal point $\mathbf{f}$ should be placed at the center of this slice. Calling the ResetCamera() method of the vtkRenderer will automatically place the focal point at the center of the volume, or the desired focal point can be computed manually through the use of Equation 1 or 2. Given the focal point $\mathbf{f}$, the camera position $\mathbf{c}$ that is needed to provide the desired viewpoint can be found by using the following equation:

$$\mathbf{c} = \mathbf{f} + d\,\mathbf{v}_{\text{out}} \tag{14}$$

The camera distance d can also be automatically set by calling ResetCamera(), and the clipping range $[d_{\text{near}}, d_{\text{far}}]$, which should be set so that all of the patient data fits between them, can be automatically set by calling ResetCameraClippingRange(). Alternatively, these can be manually set to reasonable values by referencing Figure 5.

2.4 View orientations for multi-planar reformats

The radiological view orientations for head scans are shown in Figure 6, though orientations can be anatomy- and application-specific. Notwithstanding these exceptions, this section will focus on the standard axial, sagittal, and coronal orientations. The first thing to note is that axial scans do not always lie exactly within the XY plane of the DICOM patient coordinate system, the technologists often choose a scan plane that is slightly oblique in order to better align the slices with the patient's anatomy. The same is true for sagittal and coronal scans. The goal, then, is to align the reformatted views not with the DICOM (x, y, z) axes, but with these slightly oblique scan axes (x', y', z') instead, which can be generated from the DICOM slice orientation axes $\mathbf{u}, \mathbf{v}$ by using the following procedure:

1. Find the component u_i of (u_x, u_y, u_z) that has the largest absolute value.
2. Find the component v_j of (v_x, v_y, v_z) that has the largest absolute value, where $j \neq i$.
3. Set k to the remaining component index, i.e. the index not equal to i or j.
4. Let $\mathbf{a}_x, \mathbf{a}_y, \mathbf{a}_z$ represent the unit vectors for the x', y', z' axes.
5. Two of these vectors are given by $\mathbf{a}_i = \operatorname{sgn}(u_i)\,\mathbf{u}$, and by $\mathbf{a}_j = \operatorname{sgn}(v_j)\,\mathbf{v}$.
6. The third vector is given by $\mathbf{a}_k = \mathbf{a}_i \times \mathbf{a}_j$ if ijk is an even permutation like zxy, or by $\mathbf{a}_k = -\mathbf{a}_i \times \mathbf{a}_j$ if ijk is an odd permutation like xzy.

The computed values for $\mathbf{a}_x, \mathbf{a}_y, \mathbf{a}_z$ can be used to set the view axes with reference to the x', y', z' labels in Figure 6. The result will be three orthogonal view orientations, where one of the three is aligned with the original scan plane. Please note, however, that if the original scan plane was more than just slightly oblique (e.g. more than 20° oblique), then it is not valid to label the generated orientations as axial, sagittal, and coronal. The non-oriented oblique angle can be computed with the following mathematical identity:

$$\cos\theta = \tfrac{1}{2}(a_{xx} + a_{yy} + a_{zz} - 1) \tag{15}$$

The way in which an application should respond to oblique scan orientations will depend on the purpose of the application.

Simplifying camera settings with vtkInteractorStyleImage

The vtkInteractorStyleImage class in VTK 5.10 has several methods that can simplify setting the image orientation. After these methods are called, the renderer's ResetCamera() and ResetCameraClippingRange() methods should be called to center the image within the renderer:

SetCurrentRenderer(*renderer*)	Set the renderer that the orientation is to be set for.
SetImageOrientation($\mathbf{v}_{\text{right}}$, $\mathbf{v}_{\text{up}}$)	Set the orientation from arbitrary $\mathbf{v}_{\text{right}}$, $\mathbf{v}_{\text{up}}$ direction vectors.
SetImageOrientation($\mathbf{u}$, -$\mathbf{v}$)	Use the DICOM slice orientation $\mathbf{u},\mathbf{v}$ as the view orientation.
SetZViewRightVector($\mathbf{a}_x$) SetZViewUpVector(-$\mathbf{a}_y$)	Set the orientation that will be selected when the Z key is pressed on the keyboard to $\mathbf{a}_x$,-$\mathbf{a}_y$ (axial).
SetXViewRightVector($\mathbf{a}_y$) SetXViewUpVector($\mathbf{a}_z$)	Set the orientation that will be selected when the X key is pressed on the keyboard to $\mathbf{a}_y,\mathbf{a}_z$ (sagittal).
SetYViewRightVector($\mathbf{a}_x$) SetYViewUpVector($\mathbf{a}_z$)	Set the orientation that will be selected when the Y key is pressed on the keyboard to $\mathbf{a}_x,\mathbf{a}_z$ (coronal).

3 Discussion

The procedures that we have described in this document, for maintaining the correct image orientation, require certain behavior from the reader and importer classes. Specifically, they require that the image is not flipped into a bottom-to-top orientation to match the VTK display coordinate system, and that the origin of the images is set to zero so that the position can be specified in a separate 4×4 matrix that also includes the orientation. Two image reader classes and one image importer are discussed below, along with recommendations on how they should be used.

For viewing images, we cannot recommend either of the VTK image viewer classes (vtkImageViewer or vtkImageViewer2). These classes are inflexible with respect to setting view orientations. It is better to directly use a vtkRenderer instead, and to either use vtkCamera to set the orientation, or to set the camera's orientation with vtkInteractorStyleImage, as described in Section 2.3.

3.1 The vtkDICOMImageReader

VTK comes with its own DICOM reader, which has unfortunately received very little attention over the last eight years. It has several faults, although most of these could be fixed with only small changes to the source code:

- This reader ignores the FileLowerLeft setting. Instead, its output must be passed through the vtkImageFlip filter, with FilteredAxis set to 1 (i.e. flip vertically), in order to undo the forced flip in the reader code.
- Sorting of the images follows the same scheme that we described in Section 2.1, but without verifying that all images have the same orientation, that all images are spaced the same distance apart, that all images are within the same Frame of Reference, or that the volume has no shear (e.g. tilted CT gantry).
- The only way that files can be specified is by giving a single file name, or by giving a directory name. This reader cannot be given a list of files.

- No support is provided for compressed DICOM images, even though compressed files are very commonly encountered.
- Most importantly the reader assumes the spacing can be read from the Pixel Spacing tag, which is as we saw only correct for MR Image Storage and CT Image Storage.
- Finally this reader does not support the new Enhanced MR Image Storage or Enhanced CT Image Storage.

3.2 The vtkGDCMImageReader

GDCM[3] is the primary DICOM reader for ITK, and it can also be used with VTK. Given the serious problems with the vtkDICOMImageReader, we recommend that people use the vtkGDCMImageReader until such a time that the vtkDICOMImageReader is fixed.

- Sorting of the images is supported by a separate class called gdcm::IPPSorter, but this class is not used by default. The only faults that we found with the sorting were that it did not include any tolerance when verifying that the orientation was the same for all slices, which reduces the robustness of the sort. When this sorter was not used, we found that the reader reversed the Z direction of some of our data sets.
- The image is flipped by default to respect VTK convention and thus allow easy writting to TIFF or PNG. This can be changed by calling the FileLowerLeftOn() method of the reader, which will keep the original top-to-bottom row ordering of the image.
- The origin of the output image is incorrectly set if the orientation matrix is not the identity matrix. This can be fixed by passing the image through vtkImageChangeInformation and using SetOutputOrigin() to assign a correct value to the origin, as per Section 2.2.

3.3 Importing data via ITK

DICOM images can be read and processed with ITK, and then exported to VTK for visualization. The default behavior of the ITK to VTK bridge is incorrect for oriented images, but can be easily fixed:

- The origin of the VTK image is set to the origin of the ITK image, which is incorrect when the ITK image has an orientation. Fix this by using vtkImageChangeInformation with SetOutputOrigin() to set the correct origin, as per Section 2.2.

4 Conclusion

We have described methods for managing medical image orientation in VTK, at both the data level and rendering level. Since the image data class does not provide a method for storing orientation itself, the orientation must be stored in a separate VTK matrix. Our recommendation is that this matrix be attached to the actor (or prop) that is used to associate the data with the VTK renderer, which will result in identity

[3] http://gdcm.sf.net

between the VTK world coordinate system and the DICOM patient coordinate system. This simplifies the fusion of other data sets associated with the DICOM image within the world coordinate system.

Rendering of images in the correct view orientation is a complex process, since VTK's use of a virtual camera is counterintuitive with respect to generating 2D image views. However, we hope that our instructions on setting the camera parameters will allow other programmers to achieve their desired results with a minimum of effort. Through the use of the vtkImageResliceMapper class in VTK 5.10, reformat views can be achieved solely through setting the camera orientation without the need to create a VTK imaging pipeline to reformat the data.

The popular DICOM image readers for VTK differ in behavior from the recommendations that we provide, but in each case there are ways to work around the differences. We hope to be able to resolve the issues with the readers in the near future.

References

[1] Medical Imaging & Technology Alliance. DICOM Base Standard – 2011. Published by the Association of Electrical and Medical Imaging Equipment Manufacturers (NEMA), 2011.