DICOM Spatial Transform IO in the Insight Toolkit

Release 1.00

Matthew McCormick[1], Kevin Wang[2], Andras Lasso[3], Gregory Sharp[4], and Steve Pieper[4]

September 11, 2014

[1]Kitware, Inc., Cary, NC, USA
[2]Princess Margaret Hospital, Toronto, Ontario, CA
[3]Queen's University, Kingston, Ontario, CA
[4]Massachusetts General Hospital, Boston, MA, USA
[5]Isomics, Boston, MA, USA

Abstract

This document describes a module that extends the Insight Toolkit, ITK www.itk.org, which reads DICOM Spatial Registration Object files into itk::Transform's. Currently, DICOM files are read by applying the DCMTK library as a backend. An itk::DCMTKTransformIO class can be registered with the IO factory mechanism so itk::TransformFileReaderTemplate will recognize and read these files.

This paper is accompanied with the source code, input data, parameters and output data that the authors used for validating the algorithm described in this paper. This adheres to the fundamental principle that scientific publications must facilitate reproducibility of the reported results.

Latest version available at the Insight Journal [http://hdl.handle.net/10380/3468]

Contents

DICOM, Digital Imaging and Communications in Medicine, is the standard for medical images and related information [3]. As image registration computational research has been adopted into clinical practice, DI-

COM has been extended to support registration information. Registration is applied in medical practices such as radiotherapy [8].

The Insight Segmentation and Registration Toolkit, ITK, is a resource for medical image analysis applied in both a research and clinical context. This article describes a module [2] for ITK that makes it possible to read affine transforms stored as DICOM files.

First, a brief background of how spatial transforms are stored in DICOM is presented. Next, an overview of transforms are handled in the proposed module code is described. A concrete example showing how the module is applied follows. Finally, an overview describing next steps is given.

1 DICOM Spatial Transform Background

Regarding spatial transforms in DICOM, there are three definitions to primarily be concerned with, the *Frame of Reference* [5], the *Spatial Registration Information Object Definition IOD* [6, 7], and the *Deformable Spatial Registration Information Object Definition IOD* [6, 4].

A *Frame of Reference* identifies the coordinate system associated with an image. The Frame of Reference Unique Identification Number (UID), tag `(0020,0052)`, is associated with an image series. Frame of Reference UID's are also used to identify the coordinate systems that a spatial transform maps to and from.

The *Spatial Registration IOD* describes DICOM files that store affine spatial transforms between Frames of Reference. A single Spatial Registration Object (SRO) can contain multiple transforms. All spatial transforms in a SRO are with respect to a Reference Coordinate System, which is also identified with a Frame of Reference UID.

The Reference Coordinate System could be the Frame of Reference UID for a reference image that the other images were registered against. In this case, one of the transforms stored in the SRO, which correponds to the transform for the reference image, will be an identity transform. However, the Reference Coordinate System could also be an independent coordinate system, such as when the images are registered against an atlas.

The transform for each image series in a SRO can consist of a chain of rigid or affine transforms.

The *Deformable Spatial Registration IOD* is similar to the Spatial Registration IOD, but displacement field transforms are also possible in the chain of transforms.

2 ITK Interface to DICOM Transforms

ITK uses an object factory mechanisms to support reading multiple file formats. For images, classes that read and write specific file formats inherit from the `itk::ImageIOBase`, and the `itk::ImageFileReader` uses the objects registered to the factory to attempt to read, then read, a given file. Similarly, the classes that inherit `itk::TransformIOBaseTemplate` to read specific transform file formats are registered in the factory, and `itk::TransformFileReaderTemplate` uses this factory to read a given transform file name.

We wrote an `itk::DCMTKTransformIO` class that inherits from `itk::TransformIOBaseTemplate`, and can read DICOM SRO files. Internally, this class uses the DCMTK library [1] to read the file.

Since a sequence of transforms is possible, `itk::DCMTKTransformIO` generates a `itk::CompositeTransform`. The ITK transforms that correspond to the various DICOM transform

DICOM Type	ITK Type
RIGID_SCALE	itk::ScaleTransform
RIGID	itk::Euler3DTransform
AFFINE	itk::AffineTransform

Table 1: DICOM transform types and their corresponding ITK transform type.

types are listed in Table 2.

The default output of the DCMTKTransformIO is the concatenation of all transforms found in the SRO in the resulting itk::CompositeTransform. To request only the transforms corresponding to a specific Frame of Reference, call SetFrameOfReferenceUID.

To find the spatial transform between the Frame of Reference for a Fixed Image and the Frame of Reference for a Moving Image, first obtain the transform for the Fixed Image,

$$T_f = T_{f1} \circ T_{f2} \circ ... T_{fN} = T_{f1}(T_{f2}(...T_{fN})) \tag{1}$$

where T_f is the Fixed Image composite transform to the Reference Coordinate System. The component transforms, $T_{f1}, T_{f2}, ...$ are a sequence of rigid, scale, or affine transforms.

If the Frame of Reference is specified for the Moving Image composite transform to the Reference Coordinate System, T_m,

$$T_m = T_{m1} \circ T_{m2} \circ ... T_{mN} = T_{m1}(T_{m2}(...T_{mN})) \tag{2}$$

then the transform from the Fixed Image Frame of Reference to the Moving Image Frame of Reference is

$$T_{fm} = T_m^{-1} \circ T_f = T_m^{-1}(T_f) \tag{3}$$

where T_m^{-1} is the inverse of the moving transform.

3 Example

The format of this LaTeX file, allows authors to include code snippets, like the following

```
/*=========================================================================
 *
 *  Copyright Insight Software Consortium
 *
 *  Licensed under the Apache License, Version 2.0 (the "License");
 *  you may not use this file except in compliance with the License.
 *  You may obtain a copy of the License at
 *
 *         http://www.apache.org/licenses/LICENSE-2.0.txt
 *
 *  Unless required by applicable law or agreed to in writing, software
 *  distributed under the License is distributed on an "AS IS" BASIS,
 *  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
```

```
 *  See the License for the specific language governing permissions and
 *  limitations under the License.
 *
 *=========================================================================*/

#include "itkDCMTKTransformIO.h"
#include "itkDCMTKTransformIOFactory.h"
#include "itkTransformFileReader.h"
#include "itkImageSeriesReader.h"
#include "itkDCMTKImageIO.h"
#include "itkDCMTKSeriesFileNames.h"
#include "itkGDCMImageIO.h"
#include "itkGDCMSeriesFileNames.h"
#include "itkCompositeTransform.h"
#include "itkImageFileWriter.h"
#include "itkMetaDataObject.h"
#include "itkResampleImageFilter.h"


int ReadDicomTransformAndResampleExample( int argc, char* argv[] )
{
  // Parse command line arguments
  if( argc < 5 )
    {
    std::cerr << "Usage: " << argv[0]
              << " fixedSeriesDirectory movingSeriesDirectory"
              << " transform fixedImageOutput resampledMovingOutput"
              << std::endl;
    return EXIT_FAILURE;
    }
  const char * fixedSeriesDirectory = argv[1];
  const char * movingSeriesDirectory = argv[2];
  const char * transformFileName = argv[3];
  const char * fixedImageOutputFileName = argv[4];
  const char * resampledMovingOutputFileName = argv[5];


  // Basic types
  const unsigned int Dimension = 3;
  typedef short                                PixelType;
  typedef itk::Image< PixelType, Dimension > ImageType;


  // Read the fixed and moving image
  typedef itk::ImageSeriesReader< ImageType > ReaderType;
  ReaderType::Pointer fixedReader = ReaderType::New();

  // DCMTKImageIO does not populate the MetaDataDictionary yet
  //typedef itk::DCMTKImageIO ImageIOType;
  typedef itk::GDCMImageIO ImageIOType;
  ImageIOType::Pointer fixedIO = ImageIOType::New();
  fixedReader->SetImageIO( fixedIO );

  //typedef itk::DCMTKSeriesFileNames SeriesFileNamesType;
  typedef itk::GDCMSeriesFileNames SeriesFileNamesType;
  SeriesFileNamesType::Pointer fixedSeriesFileNames =
    SeriesFileNamesType::New();
  fixedSeriesFileNames->SetInputDirectory( fixedSeriesDirectory );
  typedef SeriesFileNamesType::FileNamesContainerType FileNamesContainerType;
```

```
const FileNamesContainerType & fixedFileNames =
  fixedSeriesFileNames->GetInputFileNames();
std::cout << "There are "
          << fixedFileNames.size()
          << " fixed image slices."
          << std::endl;
std::cout << "First fixed images series UID: "
          << fixedSeriesFileNames->GetSeriesUIDs()[0]
          << "\n" << std::endl;
fixedReader->SetFileNames( fixedFileNames );

ReaderType::Pointer movingReader = ReaderType::New();
ImageIOType::Pointer movingIO = ImageIOType::New();
movingReader->SetImageIO( movingIO );

SeriesFileNamesType::Pointer movingSeriesFileNames =
  SeriesFileNamesType::New();
movingSeriesFileNames->SetInputDirectory( movingSeriesDirectory );
const FileNamesContainerType & movingFileNames =
  movingSeriesFileNames->GetInputFileNames();
std::cout << "There are "
          << movingFileNames.size()
          << " moving image slices."
          << std::endl;
std::cout << "First moving images series UID: "
          << movingSeriesFileNames->GetSeriesUIDs()[0]
          << "\n" << std::endl;
movingReader->SetFileNames( movingFileNames );

try
  {
  fixedReader->Update();
  movingReader->Update();
  }
catch( itk::ExceptionObject & error )
  {
  std::cerr << "Error: " << error << std::endl;
  return EXIT_FAILURE;
  }


// Create a DICOM transform reader
typedef float ScalarType;

itk::DCMTKTransformIOFactory::Pointer dcmtkTransformIOFactory =
  itk::DCMTKTransformIOFactory::New();
itk::ObjectFactoryBase::RegisterFactory( dcmtkTransformIOFactory );

typedef itk::TransformFileReaderTemplate< ScalarType > TransformReaderType;
TransformReaderType::Pointer transformReader = TransformReaderType::New();
transformReader->SetFileName( transformFileName );

typedef itk::DCMTKTransformIO< ScalarType > TransformIOType;
TransformIOType::Pointer transformIO = TransformIOType::New();
transformReader->SetTransformIO( transformIO );


// Read in the fixed image transform
const ReaderType::DictionaryType & fixedMetaDataDict =
```

```
  fixedIO->GetMetaDataDictionary();
std::string fixedFrameOfReferenceUID;
if( ! itk::ExposeMetaData< std::string >( fixedMetaDataDict,
                                          "0020|0052",
                                          fixedFrameOfReferenceUID ) )
  {
  std::cerr << "Could not find the fixed image frame of reference UID." << std::endl;
  return EXIT_FAILURE;
  }
std::cout << "Fixed image frame of reference UID: "
          << fixedFrameOfReferenceUID << std::endl;
transformIO->SetFrameOfReferenceUID( fixedFrameOfReferenceUID );

try
  {
  transformReader->Update();
  }
catch( itk::ExceptionObject & error )
  {
  std::cerr << "Error: " << error << std::endl;
  return EXIT_FAILURE;
  }
typedef TransformReaderType::TransformListType TransformListType;
TransformListType * transformList = transformReader->GetTransformList();

typedef itk::CompositeTransform< ScalarType, Dimension > ReadTransformType;
TransformListType::const_iterator transformIt = transformList->begin();
ReadTransformType::Pointer fixedTransform =
  dynamic_cast< ReadTransformType * >( (*transformIt).GetPointer() );
if( fixedTransform.IsNull() )
  {
  std::cerr << "Did not get the expected transform out." << std::endl;
  return EXIT_FAILURE;
  }
std::cout << "Fixed transform: " << fixedTransform << std::endl;


// Read in the moving image transform
const ReaderType::DictionaryType & movingMetaDataDict =
  movingIO->GetMetaDataDictionary();
std::string movingFrameOfReferenceUID;
if( ! itk::ExposeMetaData< std::string >( movingMetaDataDict,
                                          "0020|0052",
                                          movingFrameOfReferenceUID ) )
  {
  std::cerr << "Could not find the moving image frame of reference UID." << std::endl;
  return EXIT_FAILURE;
  }
std::cout << "Moving image frame of reference UID: "
          << movingFrameOfReferenceUID << std::endl;
transformIO->SetFrameOfReferenceUID( movingFrameOfReferenceUID );

try
  {
  transformReader->Update();
  }
catch( itk::ExceptionObject & error )
  {
  std::cerr << "Error: " << error << std::endl;
```

```
  return EXIT_FAILURE;
  }

transformList = transformReader->GetTransformList();
transformIt = transformList->begin();
ReadTransformType::Pointer movingTransform =
  dynamic_cast< ReadTransformType * >( (*transformIt).GetPointer() );
if( movingTransform.IsNull() )
  {
  std::cerr << "Did not get the expected transform out." << std::endl;
  return EXIT_FAILURE;
  }
std::cout << "Moving transform: " << movingTransform << std::endl;


// Compose the transform from the fixed to the moving image
ReadTransformType::Pointer movingTransformInverse = ReadTransformType::New();
movingTransform->GetInverse( movingTransformInverse );

ReadTransformType::Pointer fixedToMovingTransform = ReadTransformType::New();
fixedToMovingTransform->AddTransform( fixedTransform );
fixedToMovingTransform->AddTransform( movingTransformInverse );
// Flatten out the two component CompositeTransforms.
fixedToMovingTransform->FlattenTransformQueue();

typedef itk::ResampleImageFilter< ImageType, ImageType, ScalarType, ScalarType >
  ResamplerType;
ResamplerType::Pointer resampler = ResamplerType::New();
resampler->SetInput( movingReader->GetOutput() );
resampler->SetUseReferenceImage( true );
resampler->SetReferenceImage( fixedReader->GetOutput() );
resampler->SetTransform( fixedToMovingTransform );
resampler->SetDefaultPixelValue( -1000 );


// Write the fixed image and resampled moving image (should look similar)
typedef itk::ImageFileWriter< ImageType > WriterType;
WriterType::Pointer writer = WriterType::New();
writer->SetFileName( fixedImageOutputFileName );
writer->SetInput( fixedReader->GetOutput() );
try
  {
  writer->Update();
  }
catch( itk::ExceptionObject & error )
  {
  std::cerr << "Error: " << error << std::endl;
  return EXIT_FAILURE;
  }

writer->SetInput( resampler->GetOutput() );
writer->SetFileName( resampledMovingOutputFileName );
try
  {
  writer->Update();
  }
catch( itk::ExceptionObject & error )
  {
  std::cerr << "Error: " << error << std::endl;
```

```
    return EXIT_FAILURE;
    }


  return EXIT_SUCCESS;
}
```

4 Future Work

While reading the Spatial Registration IOD is now possible, future work could entail adding write support. Support for Deformable Registration IOD's could also be added, which correspond to an `itk::CompositeTransform` of affine transforms and the `itk::DisplacementFieldTransform`. While the low-level reading of the DICOM files is currently achieved with the DCMTK library, similar calls could be performed to create a GDCM library-based implementation.

References

[1] M. Eichelberg, M. Onken, and A. Thiel. The dicom toolkit. http://dicom.offis.de/. Accessed: 2014-09-04. 2

[2] Xiaoxiao Liu, Brian Helba, Luis Ibanez, Brad King, and Matt McCormick. Advance itk with modules. http://www.kitware.com/blog/home/post/557. Accessed: 2014-09-03. (document)

[3] NEMA. The dicom standard. Technical report, NEMA, http://dicom.nema.org/, 2014. (document)

[4] NEMA. The dicom standard: Deformable spatial registration storage sop classes. Technical Report Supp 112, NEMA, ftp://medical.nema.org/medical/dicom/final/sup112_ft.pdf, 2014. 1

[5] NEMA. The dicom standard: Frame of reference module. Technical Report Part 3, Section C.7.4.1, NEMA, http://medical.nema.org/medical/dicom/2014a/output/chtml/part03/sect_C.7.html, 2014. 1

[6] NEMA. The dicom standard: Spatial registration iod. Technical Report Part 3, Section A.39, NEMA, http://medical.nema.org/medical/dicom/2014a/output/chtml/part03/sect_A.39.html, 2014. 1

[7] NEMA. The dicom standard: Spatial registration storage sop classes. Technical Report Supp 73, NEMA, ftp://medical.nema.org/medical/dicom/final/sup73_ft4.pdf, 2014. 1

[8] Csaba Pinter, Andras Lasso, An Wang, David Jaffray, and Gabor Fichtinger. Slicerrt – radiation therapy research toolkit for 3d slicer. *Medical Physics*, 39:6332/7, 10/2012 2012. (document)