SimpleITK Documentation

Release 1.2.0.dev

Insight Software Consortium

Feb 12, 2019

Table of Contents

SimpleITK is a simplified, open source, interface to the National Library of Medicine's Insight Segmentation and Registration Toolkit (ITK), a C++ open source image analysis toolkit which is widely used in academia and industry. SimpleITK is available for eight programming languages including C++, Python, R, Java, C#, Lua, Ruby, and TCL. Binary distributions of SimpleITK are currently available for all three major operating systems (Linux, OS X, and Windows).

CHAPTER 1

Installation

SimpleITK provides a simplified interface to ITK in a variety of languages. You can either download binaries, if they are available for your platform and prefered language, or you can *build SimpleITK* yourself from the source code.

Additionally, there are several recommended third-party software packages.

After you have installed SimpleITK, please look to the Tutorial or the Doxygen pages for more information.

1.1 Downloading the binaries

One of the great advantages of SimpleITK is that (typically) you do not have to build it — you can simply download the binaries and get started right away!

Currently, **Python** binaries are available on Microsoft Windows, GNU Linux and Apple OS X. **C# and Java** binaries are available for Windows. We are also working towards supporting **R** packaging.

1.1.1 Python binary files

There are currently two Python binary package choices: Python Wheels, and Anaconda packages for the Anaconda Python distribution. We recommend the use of a *virtual environment* for installation of SimpleITK.

Wheels for Generic Python Distribution

From the command line use the pip program to install a binary wheel:

```
pip install SimpleITK
```

This installation requires a recent version of pip (>=9.0), to properly detect compatibility with the PEP 427 tags in the wheel filenames. You can update your version of pip using `pip install -U pip`. Also your Python environment must be compatible with one of the pre-compiled binary wheels.

Alternatively, the wheels can be manually downloaded from sourceforge or PyPI, and then installed with pip.

1.1.2 Conda-based distributions (Anaconda, Miniconda)

From the command line prompt, execute:

```
conda install -c https://conda.anaconda.org/simpleitk SimpleITK
```

Beta and release candidate packages are also available on Anaconda cloud under the dev label:

```
conda install -c https://conda.anaconda.org/simpleitk/label/dev SimpleITK
```

1.1.3 C# binary files

Binaries for select C# platforms can be found on SimpleITK's SourceForge page under the version directory then the CSharp sub-directory. Installing the library should only involve importing the unzipped files into you C# environment. The files have the following naming convention:

`SimpleITK-version-CSharp-buildplatform-targetplatform.zip`

eg.

`SimpleITK-1.2.0-CSharp-win32-x86.zip`
`SimpleITK-1.2.0-CSharp-win64-x64.zip`

Details about how to set up a C# Visual Studio project with SimpleITK can be found in the Visual Guide to SimpleITK with CSharp.

More information about getting started with a sample C# program can be found in A visual guide to building SimpleITK on Linux

1.1.4 Java binary files

Binaries for select Java platforms can be found on SimpleITK's SourceForge page under the version directory then the CSharp sub-directory. Installation instructions are available at a visual guide to SimpleITK in Java.

1.1.5 Nightly binaries

The **latest binaries** for the current development version of SimpleITK are also generally available. Binary packages are built as part of the nightly regression testing system. The download links are available from the CDash dashboard in the "Nightly Packages" section.

Each row on the dashboard is a SimpleITK build on a particular system, and if the build was successful there will be a **package icon**: https://open.cdash.org/img/package.png which links to the packages build by the system. A user may directly download the built package from such a link.

1.2 Recommended Software

1.2.1 Fiji (Fiji is Just ImageJ)

SimpleITK has a built in function itk::simple::Show() which can be used for viewing images in an interactive session. By default this Show function searches for an installed Fiji to display images. If Fiji is not found, then it searches for

ImageJ. Fiji/ImageJ were chosen because they can handle all the image types that SimpleITK supports, including 3D vector images with n components.

The Show function first searches the "PATH" environment variable, then additional standard locations are examined, if problems are encountered the correct path can be added to this environment variable and the **"debugOn**" option to **"Show**" flag set.

The Show function can also be configured to invoke some other viewing application using the SITK_SHOW_COMMAND environment variable.

1.2.2 IPython and Jupyter

If you are using python, IPython with Jupyter is terrific environment to perform interactive computing for image processing. With the addition of numpy and scipy, you will have a powerful interactive environment.

We have instructional SimpleITK Jupyter Notebooks which can help you get started.

CHAPTER 2

Fundamental Concepts

The two basic elements which are at the heart of SimpleITK are images and spatial transformations. These follow the same conventions as the ITK components which they represent. The fundamental underlying concepts are described below.

2.1 Images

The fundamental tenet of an image in ITK and consequentially in SimpleITK is that an image is defined by a set of points on a grid occupying a **physical region in space**. This significantly differs from many other image analysis libraries that treat an image as an array which has two implications: (1) pixel/voxel spacing is assumed to be isotropic and (2) there is no notion of an image's location in physical space.

SimpleITK images are either 2D, 3D, or 4D and can have an arbitrary number of channels with a scalar or complex value in each channel. The physical region in space which an image occupies is defined by the image's:

1. Origin (vector like type) - location in the world coordinate system of the voxel with all zero indexes.
2. Spacing (vector like type) - distance between pixels along each of the dimensions.
3. Size (vector like type) - number of pixels in each dimension.
4. Direction cosine matrix (vector like type representing matrix in row major order) - direction of each of the axes corresponding to the matrix columns.

The meaning of each of these meta-data elements is visually illustrated in *this figure*.

Fig. 1: An image in SimpleITK occupies a region in physical space which is defined by its meta-data (origin, size, spacing, and direction cosine matrix). Note that the image's physical extent starts half a voxel before the origin and ends half a voxel beyond the last voxel.

In SimpleITK, when we construct an image we specify its dimensionality, size and pixel type, all other components are set to **reasonable default values**:

1. origin - all zeros.

2. spacing - all ones.
3. direction - identity.
4. intensities in all channels - all zero.

In the following Python code snippet we illustrate how to create a 2D image with five float valued channels per pixel, origin set to (3, 14) and a spacing of (0.5, 2). Note that the metric units associated with the location of the image origin in the world coordinate system and the spacing between pixels are unknown (km, m, cm, mm,...). It is up to you the developer to be consistent. More about that *below*.

```
image = sitk.Image([10,10], sitk.sitkVectorFloat32, 5)
image.SetOrigin((3.0, 14.0))
image.SetSpacing((0.5, 2))
```

The tenet that images occupy a spatial location in the physical world has to do with the original application domain of ITK and SimpleITK, medical imaging. In that domain images represent anatomical structures with metric sizes and spatial locations. Additionally, the spacing between voxels is often non-isotropic (most commonly the spacing along the inferior-superior/foot-to-head direction is larger). Viewers that treat images as an array will display a distorted image as shown in *this figure*.

Fig. 2: The same image displayed with a viewer that is not aware of spatial meta-data (left image) and one that is aware (right image). The image's pixel spacing is (0.97656, 2.0)mm.

As an image is also defined by its spatial location, two images with the same pixel data and spacing may not be considered equivalent. Think of two CT scans of the same patient acquired at different sites. *This figure* illustrates the notion of spatial location in the physical world, the two images are considered different even though the intensity values and pixel spacing are the same.

Fig. 3: Two images with exactly the same pixel data, positioned in the world coordinate system. In SimpleITK these are not considered the same image, because they occupy different spatial locations. The image on the left has its origin at (-136.3, -20.5) with a direction cosine matrix, in row major order, of (0.7, -0.7, 0.7, 0.7). The image on the right's origin is (16.9, 21.4) with a direction cosine matrix of (1,0,0,1).

As SimpleITK images occupy a physical region in space, the quantities defining this region have metric units (cm, mm, etc.). In general SimpleITK assume units are in millimeters (historical reasons, due to DICOM standard). In practice SimpleITK is not aware of the specific units associated with each image, it just assumes that they are consistent. Thus, it is up to you the developer to ensure that all of the images you read and created are using the same units. Mixing units and using wrong units has not ended well in the past.

Finally, having convinced you to think of images as objects occupying a physical region in space, we need to answer two questions:

1. How do you access the pixel values in an image:

   ```
   image.GetPixel((0,0))
   ```

   SimpleITK functions use a zero based indexing scheme. The toolkit also includes syntactic sugar that allows one to use the bracket operator in combination with the native zero/one based indexing scheme (e.g. a one based indexing in R vs. the zero based indexing in Python).

2. How do you determine the physical location of a pixel:

   ```
   image.TransformIndexToPhysicalPoint((0,0))
   ```

   This computation can also be done manually using the meta-data defining the image's spatial location, but we highly recommend that you do not do so as it is error prone.

2.1.1 Channels

As stated above, a SimpleITK image can have an arbitrary number of channels with the content of the channels being a scalar or complex value. This is determined when an image is created.

In the medical domain, many image types have a single scalar channel (e.g. CT, US). Another common image type is a three channel image where each channel has scalar values in [0,255], often people refer to such an image as an RGB image. This terminology implies that the three channels should be interpreted using the RGB color space. In some cases you can have the same image type, but the channel values represent another color space, such as HSV (it decouples the color and intensity information and is a bit more invariant to illumination changes). SimpleITK has no concept of color space, thus in both cases it will simply view a pixel value as a 3-tuple.

Word of caution: In some cases looks may be deceiving. Gray scale images are not always stored as a single channel image. In some cases an image that looks like a gray scale image is actually a three channel image with the intensity values repeated in each of the channels. Even worse, some gray scale images can be four channel images with the channels representing RGBA and the alpha channel set to all 255. This can result in a significant waste of memory and computation time. Always become familiar with your data.

2.1.2 Additional Resources

1. The API for the SimpleITK Image class in Doxygen format.
2. To really understand the structure of SimpleITK images and how to work with them, we recommend some hands-on interaction using the SimpleITK Jupyter notebooks (Python and R only).

2.2 Transforms

SimpleITK supports two types of spatial transforms, ones with a global (unbounded) spatial domain and ones with a bounded spatial domain. Points in SimpleITK are mapped by the transform using the *TransformPoint* method.

All **global domain transforms** are of the form:

$$T(\mathbf{x}) = A(\mathbf{x} - \mathbf{c}) + \mathbf{t} + \mathbf{c}$$

The nomenclature used in the documentation refers to the components of the transformations as follows:

- Matrix - the matrix A.
- Center - the point $\mathbf{c}$.
- Translation - the vector $\mathbf{t}$.
- Offset - the expression $\mathbf{t} + \mathbf{c} - A\mathbf{c}$.

A variety of global 2D and 3D transformations are available (translation, rotation, rigid, similarity, affine…). Some of these transformations are available with various parameterizations which are useful for registration purposes.

The second type of spatial transformation, **bounded domain transformations**, are defined to be identity outside their domain. These include the B-spline deformable transformation, often referred to as Free-Form Deformation, and the displacement field transformation.

The B-spline transform uses a grid of control points to represent a spline based transformation. To specify the transformation the user defines the number of control points and the spatial region which they overlap. The spline order can also be set, though the default of cubic is appropriate in most cases. The displacement field transformation uses a dense set of vectors representing displacement in a bounded spatial domain. It has no implicit constraints on transformation continuity or smoothness.

Finally, SimpleITK supports a **composite transformation** with either a bounded or global domain. This transformation represents multiple transformations applied one after the other $T_0(T_1(T_2(...T_n(p)...)))$. The semantics are stack based, that is, first in last applied:

```
composite_transform <- Transform(T0)
composite_transform$AddTransform(T1)
```

In the context of registration, if you use a composite transform as the transformation that is optimized, only the parameters of the last transformation T_n will be optimized over.

2.2.1 Additional Resources

1. The API for the SimpleITK transformation classes is available in Doxygen format:
   - 2D or 3D translation.
   - VersorTransform.
   - Euler2DTransform and Euler3DTransform.
   - Similarity2DTransform and Similarity3DTransform.
   - 2D or 3D ScaleTransform.
   - ScaleVersor3DTransform.
   - ScaleSkewVersor3DTransform.
   - 2D or 3D AffineTransform.
   - 2D or 3D BSplineTransform.
   - 2D or 3D DisplacementFieldTransform.
   - Transform.
2. To really understand the structure of SimpleITK transforms and how to work with them, we recommend some hands-on interaction using the SimpleITK Jupyter notebooks (Python and R only).

2.3 Resampling

Resampling, as the verb implies, is the action of sampling an image, which itself is a sampling of an original continuous signal.

Generally speaking, resampling in SimpleITK involves four components:

1. Image - the image we resample, given in coordinate system m.
2. Resampling grid - a regular grid of points given in coordinate system f which will be mapped to coordinate system m.
3. Transformation T_f^m - maps points from coordinate system f to coordinate system m, ${}^mp = T_f^m({}^fp)$.
4. Interpolator - method for obtaining the intensity values at arbitrary points in coordinate system m from the values of the points defined by the Image.

While SimpleITK provides a large number of interpolation methods, the two most commonly used are sitkLinear and sitkNearestNeighbor. The former is used for most interpolation tasks and is a compromise between accuracy and computational efficiency. The later is used to interpolate labeled images representing a segmentation. It is the only interpolation approach which will not introduce new labels into the result.

The SimpleITK interface includes three variants for specifying the resampling grid:

1. Use the same grid as defined by the resampled image.
2. Provide a second, reference, image which defines the grid.
3. Specify the grid using: size, origin, spacing, and direction cosine matrix.

Points that are mapped outside of the resampled image's spatial extent in physical space are set to a constant pixel value which you provide (default is zero).

2.3.1 Common Errors

It is not uncommon to end up with an empty (all black) image after resampling. This is due to:

1. Using wrong settings for the resampling grid (not too common, but does happen).
2. Using the inverse of the transformation T_f^m. This is a relatively common error, which is readily addressed by invoking the transformation's *GetInverse* method.

2.3.2 Additional Resources

1. The API for the SimpleITK ResampleImageFilter class in Doxygen format. The procedural interface for this class supports the three variations for specifying the resampling grid described above.
2. To really understand the structure of SimpleITK images and how to work with them we recommend some hands-on interaction using the SimpleITK Jupyter notebooks (Python and R only).

CHAPTER 3

Registration Overview

The goal of registration is to estimate the transformation which maps points from one image to the corresponding points in another image. The transformation estimated via registration is said to map points from the **fixed image** coordinate system to the **moving image** coordinate system.

SimpleITK provides a configurable multi-resolution registration framework, implemented in the ImageRegistrationMethod class. In addition, a number of variations of the Demons registration algorithm are implemented independently from this class as they do not fit into the framework.

3.1 Actual Code

Code illustrating various aspects of the registration framework can be found in the set of *examples* which are part of the SimpleITK distribution and in the SimpleITK Jupyter notebook repository.

3.2 ImageRegistrationMethod

To create a specific registration instance using the ImageRegistrationMethod you need to select several components which together define the registration instance:

1. Transformation.
2. Similarity metric.
3. Optimizer.
4. Interpolator.

3.2.1 Transform

The type of transformation defines the mapping between the two images. SimpleITK supports a variety of global and local transformations. The available transformations include:

- TranslationTransform.
- VersorTransform.
- VersorRigid3DTransform.
- Euler2DTransform.
- Euler3DTransform.
- Similarity2DTransform.
- Similarity3DTransform.
- ScaleTransform.
- ScaleVersor3DTransform.
- ScaleSkewVersor3DTransform.
- AffineTransform.
- BSplineTransform.
- DisplacementFieldTransform.
- Composite Transform.

The parameters modified by the registration framework are those returned by the transforms **GetParameters()** method. This requires special attention when the using a composite transform, as the specific parameters vary based on the content of your composite transformation.

3.2.2 Similarity Metric

The similarity metric reflects the relationship between the intensities of the images (identity, affine, stochastic. . .). The available metrics include:

- MeanSquares .
- Demons.
- Correlation.
- ANTSNeighborhoodCorrelation.
- JointHistogramMutualInformation.
- MattesMutualInformation.

In the ITKv4 and consequentially in SimpleITK all similarity metrics are minimized. For metrics whose optimum corresponds to a maximum, such as mutual information, the metric value is negated internally. The selection of a similarity metric is done using the ImageRegistrationMethod's **SetMetricAsX()** methods.

3.2.3 Optimizer

The optimizer is selected using the **SetOptimizerAsX()** methods. When selecting the optimizer you will also need to configure it (e.g. set the number of iterations). The available optimizers include:

- Gradient free
  - Exhaustive.
  - Nelder-Mead downhill simplex (Amoeba).
  - Powell.

  - 1+1 evolutionary optimizer.

- Gradient based:
  - Gradient Descent.
  - Gradient Descent Line Search.
  - Regular Step Gradient Descent.
  - Conjugate Gradient Line Search.
  - L-BFGS-B. Limited memory Broyden, Fletcher, Goldfarb, Shannon, Bound Constrained (supports the use of simple constraints).

3.2.4 Interpolator

SimpleITK has a large number of interpolators. In most cases linear interpolation, the default setting, is sufficient. Unlike the similarity metric and optimizer, the interpolator is set using the **SetInterpolator** method which receives a parameter indicating the interpolator type.

3.2.5 Features of Interest

Transforms and image spaces

While the goal of registration, as defined above, refers to a single transformation and two images, the ITKv4 registration and the SimpleITK ImageRegistrationMethod provide additional flexibility in registration configuration.

From a coordinate system standpoint ITKv4 introduced the **virtual image domain**, making registration a symmetric process so that both images are treated similarly. As a consequence the ImageRegistrationMethod has methods for setting **three transformations**:

1. SetInitialTransform T_o - composed with the moving initial transform, maps points from the virtual image domain to the moving image domain, modified during optimization.

2. SetFixedInitialTransform T_f - maps points from the virtual image domain to the fixed image domain, never modified.

3. SetMovingInitialTransform T_m- maps points from the virtual image domain to the moving image domain, never modified.

The transformation that maps points from the fixed to moving image domains is thus:

$$p_{moving} = T_o(T_m(T_f^{-1}(p_{fixed})))$$

Multi Resolution Framework

The ImageRegistrationMethod supports multi-resolution, pyramid, registration via two methods SetShrinkFactorsPerLevel and SetSmoothingSigmasPerLevel. The former receives the shrink factors to apply when moving from one level of the pyramid to the next and the later receives the sigmas to use for smoothing when moving from level to level. Sigmas can be specified either in voxel units or physical units (default) using SetSmoothingSigmasAreSpecifiedInPhysicalUnits.

Sampling

For many registration tasks one can use a fraction of the image voxels to estimate the similarity measure. Aggressive sampling can significantly reduce the registration runtime. The ImageRegistration method allows you to specify how/if to sample the voxels, SetMetricSamplingStrategy, and if using a sampling, what percentage, SetMetricSamplingPercentage.

Scaling in Parameter Space

The ITKv4 framework introduced automated methods for estimating scaling factors for non-commensurate parameter units. These change the step size per parameter so that the effect of a unit of change has similar effects in physical space (think rotation of 1 radian and translation of 1 millimeter). The relevant methods are SetOptimizerScalesFromPhysicalShift, SetOptimizerScalesFromIndexShift and SetOptimizerScalesFromJacobian. In many cases this scaling is what determines if the the optimization converges to the correct optimum.

Observing Registration Progress

The ImageRegistrationMethod enables you to observe the registration process as it progresses. This is done using the Command-Observer pattern, associating callbacks with specific events. To associate a callback with a specific event use the AddCommand method.

CHAPTER 4

Common Conventions

This document contains common conventions that SimpleITK filters, and objects follow. It is intended to describe the interfaces that users should use and developers should implement. If a method or class does not specify different behavior or default values then those described here, it should be assumed that it follows the following conventions.

4.1 Dimensional Vectors

Dimensional Vectors must contain at least the number of elements as the dimensions of the image, elements beyond the image's dimension will be truncated.

The SimpleITK image class can contain 2 or 3 dimensional images. In ITK proper, certain types such as indexes, points, and sizes are templeted over the image dimensions. In SimpleITK we utilize the variable length `std::vector` for these types, so that non-templeted calls can wrap ITK's templeted image class. These types are called Dimensional Vectors. If a dimensional vector's length is less that the dimension of the image, an exception will be generated when converted to the ITK type. If there are extra elements in the dimensional vector these values will be ignored.

4.2 Image Access

Image access is in x,y,z order, `GetPixel(x,y,z)` or `image[x,y,z]`, with zero based indexing.

4.3 Mask Image Types and Default Values

The default mask image type is a scalar image of `sitkUInt8` or `uint8_t` pixels. The default values are 0 and 1, with 1s representing the mask.

These defaults are for filter which create masks such as thresholding, and certain other segmentation filters. Additionally, these are the defaults for the binary morphology filters, so that they can easily be applied after segmentation. This

choice makes many mask manipulations easier. For example, "masking" an image (scalar or vector) is simply a matter of multiplying by a mask an image. Additionally, the set of $\{0, 1\}$ pixels is closed under the logical Boolean operators.

4.4 Order of Procedural Parameters

The order of the procedural parameters should be with the most frequently modified parameters at the beginning of the parameter list. While the last parameter, may be ones that the user does not need to set or are infrequently used.

4.5 Matrices as Parameters

Matrices are represented as a single dimensional vector with the entries in row major order. The vector $[1, 2, 3, 4, 5, 6]$ represents the matrix

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

4.6 Image Regions as Index and Size

The itk::ImageRegion is a frequently used class in ITK to define a sub-image area. It is defined by itk::Index and itk::Size of signed and unsigned integer types respecfully. In SimpleITK, the index and size elements are usually separated into two arguments with separate Set and Get methods.

When specified as a single argument value, it is a 1 dimensional array with the index values followed by the size values i.e. $[idx_x, idx_y, idx_z, size_x, size_y, size_z]$.

4.7 Images As Parameters

The dimensionality (2D or 3D) and pixel type (`sitkUInt8`, `sitkFloat64`...) of images is required to be the same for most methods that receive multiple images as input.

The ImageRegistrationMethod only supports images with `sitkFloat32` and `sitkFloat64` pixel types.

Casting an image's pixel type into another is done with the SimpleITK Cast() function.

CHAPTER 5

Reading and Writing for Images and Transforms

5.1 Images

There are numerous file formats support by SimpleITK's image readers and writers. Support for a particular format is handled by a specific ITK ImageIO class. By default, the ImageIO is automatically determined for a particular file. Advanced SimpleITK installations can configure or extend which file formats are supported by SimpleITK. A list of registered ImageIO's can be found using the `GetRegisteredImageIOs()` method, but is posted here:

- BMPImageIO (*.bmp, *.BMP)
- BioRadImageIO (*.PIC, *.pic)
- Bruker2dseqImageIO
- GDCMImageIO
- GE4ImageIO
- GE5ImageIO
- GiplImageIO (*.gipl *.gipl.gz)
- HDF5ImageIO
- JPEGImageIO (*.jpg, *.JPG, *.jpeg, *.JPEG)
- LSMImageIO (*.tif, *.TIF, *.tiff, *.TIFF, *.lsm, *.LSM)
- MINCImageIO (*.mnc, *.MNC)
- MRCImageIO (*.mrc, *.rec)
- MetaImageIO (*.mha, *.mhd)
- NiftiImageIO (*.nia, *.nii, *.nii.gz, *.hdr, *.img, *.img.gz)
- NrrdImageIO (*.nrrd, *.nhdr)
- PNGImageIO (*.png, *.PNG)
- StimulateImageIO

- TIFFImageIO (*.tif, *.TIF, *.tiff, *.TIFF)
- VTKImageIO (*.vtk)

Example read and write:

```
import SimpleITK as sitk

reader = sitk.ImageFileReader()
reader.SetImageIO("BMPImageIO")
reader.SetFileName(inputImageFileName)
image = reader.Execute();

writer = sitk.ImageFileWriter()
writer.SetFileName(outputImageFileName)
writer.Execute(image)
```

5.2 Transformations

In SimpleITK, transformation files can be written in several different formats. Just like there are numerous IOs for images, there are several for transforms, including TxtTransformIO, MINCTransformIO, HDF5TransformIO, and MatlabTransformIO (although this list can be extended as well). These support a variety of file formats, including .txt, .tfm, .mat, and .xfm. A displacement field, such as one stored in a DisplacementFieldTransform object, can also be saved as an image (.nrrd, .nhdr, .mha, .mhd, .nii, .nii.gz).

Take an example of a transformation written to and read from a file in Python:

```
basic_transform = sitk.Euler2DTransform()
basic_transform.SetTranslation((2,3))

sitk.WriteTransform(basic_transform, 'euler2D.tfm')
read_result = sitk.ReadTransform('euler2D.tfm')

assert(str(type(read_result) != type(basic_transform)))
```

`read_result` will be an object of the generic `sitk.Transform()` class and `basic_transform` will be of `sitk.Euler2DTransform()`, but both represent the same transformation. Although this example only uses a single SimpleITK transformation, a .tfm file can hold a composite (set of transformations).

CHAPTER 6

SimpleITK Filters

Filter Name	Brief Description
AbsImageFilter	Computes the absolute value of each pixel.
AbsoluteValueDifferenceImageFilter	Implements pixel-wise the computation of absolute value difference.
AcosImageFilter	Computes the inverse cosine of each pixel.
AdaptiveHistogramEqualizationImageFilter	Power Law Adaptive Histogram Equalization.
AddImageFilter	Pixel-wise addition of two images.
AdditiveGaussianNoiseImageFilter	Alter an image with additive Gaussian white noise.
AggregateLabelMapFilter	Collapses all labels into the first label.
AndImageFilter	Implements the AND bitwise operator pixel-wise between two images.
AntiAliasBinaryImageFilter	A method for estimation of a surface from a binary volume.
ApproximateSignedDistanceMapImageFilter	Create a map of the approximate signed distance from the boundaries of a binary image.
AsinImageFilter	Computes the sine of each pixel.
Atan2ImageFilter	Computes two argument inverse tangent.
AtanImageFilter	Computes the one-argument inverse tangent of each pixel.
BSplineDecompositionImageFilter	Calculates the B-Spline coefficients of an image. Spline order may be from 0 to 5.
BilateralImageFilter	Blurs an image while preserving edges.
BinShrinkImageFilter	Reduce the size of an image by an integer factor in each dimension while performing averaging of an input neighborhood.
BinaryClosingByReconstructionImageFilter	Binary closing by reconstruction of an image.
BinaryContourImageFilter	Labels the pixels on the border of the objects in a binary image.
BinaryDilateImageFilter	Fast binary dilation.
BinaryErodeImageFilter	Fast binary erosion.
BinaryFillholeImageFilter	Remove holes not connected to the boundary of the image.
BinaryGrindPeakImageFilter	Remove the objects not connected to the boundary of the image.
BinaryImageToLabelMapFilter	Label the connected components in a binary image and produce a collection of label objects.
BinaryMagnitudeImageFilter	Computes the square root of the sum of squares of corresponding input pixels.

Continued on next page

Table 1 – continued from previous page

Filter Name	Brief Description
BinaryMedianImageFilter	Applies a version of the median filter optimized for binary images.
BinaryMinMaxCurvatureFlowImageFilter	Denoise a binary image using min/max curvature flow.
BinaryMorphologicalClosingImageFilter	Binary morphological closing of an image.
BinaryMorphologicalOpeningImageFilter	Binary morphological opening of an image.
BinaryNotImageFilter	Implements the BinaryNot logical operator pixel-wise between two images.
BinaryOpeningByReconstructionImageFilter	Binary morphological closing of an image.
BinaryProjectionImageFilter	Binary projection.
BinaryReconstructionByDilationImageFilter	Binary reconstruction by dilation of an image
BinaryReconstructionByErosionImageFilter	Binary reconstruction by erosion of an image
BinaryThinningImageFilter	This filter computes one-pixel-wide edges of the input image.
BinaryThresholdImageFilter	Binarize an input image by thresholding.
BinaryThresholdProjectionImageFilter	BinaryThreshold projection.
BinomialBlurImageFilter	Performs a separable blur on each dimension of an image.
BitwiseNotImageFilter	Implements pixel-wise generic operation on one image.
BlackTopHatImageFilter	Black top hat extracts local minima that are smaller than the structuring element.
BoundedReciprocalImageFilter	Computes 1/(1+x) for each pixel in the image.
BoxMeanImageFilter	Implements a fast rectangular mean filter using the accumulator approach.
BoxSigmaImageFilter	Implements a fast rectangular sigma filter using the accumulator approach.
CannyEdgeDetectionImageFilter	This filter is an implementation of a Canny edge detector for scalar-valued images.
ChangeLabelImageFilter	Change Sets of Labels.
ChangeLabelLabelMapFilter	Replace the label Ids of selected LabelObjects with new label Ids.
CheckerBoardImageFilter	Combines two images in a checkerboard pattern.
ClampImageFilter	Casts input pixels to output pixel type and clamps the output pixel values to a specified range.
ClosingByReconstructionImageFilter	Closing by reconstruction of an image.
CoherenceEnhancingDiffusionImageFilter	Coherence enhanging diffusion and edge enhancing diffusion.
CollidingFrontsImageFilter	Selects a region of space where two independent fronts run towards each other.
ComplexToImaginaryImageFilter	Computes pixel-wise the imaginary part of a complex image.
ComplexToModulusImageFilter	Computes pixel-wise the Modulus of a complex image.
ComplexToPhaseImageFilter	Computes pixel-wise the modulus of a complex image.
ComplexToRealImageFilter	Computes pixel-wise the real(x) part of a complex image.
ComposeImageFilter	ComposeImageFilter combine several scalar images into a multicomponent image.
ConfidenceConnectedImageFilter	Segment pixels with similar statistics using connectivity.
ConnectedComponentImageFilter	Label the objects in a binary image.
ConnectedThresholdImageFilter	Label pixels that are connected to a seed and lie within a range of values.
ConstantPadImageFilter	Increase the image size by padding with a constant value.
ConvolutionImageFilter	Convolve a given image with an arbitrary image kernel.
CosImageFilter	Computes the cosine of each pixel.
CropImageFilter	Decrease the image size by cropping the image by an itk::Size at both the upper and lower bounds of the largest possible region.
CurvatureAnisotropicDiffusionImageFilter	
CurvatureFlowImageFilter	Denoise an image using curvature driven flow.
CyclicShiftImageFilter	Perform a cyclic spatial shift of image intensities on the image grid.
DanielssonDistanceMapImageFilter	This filter computes the distance map of the input image as an approximation with pixel accuracy to the Euclidean distance.

Continued on next page

Table 1 – continued from previous page

Filter Name	Brief Description
DemonsRegistrationFilter	Deformably register two images using the demons algorithm.
DerivativeImageFilter	Computes the directional derivative of an image. The directional derivative at each pixel location is computed by convolution with a derivative operator of user-specified order.
DiffeomorphicDemonsRegistrationFilter	Deformably register two images using a diffeomorphic demons algorithm.
DilateObjectMorphologyImageFilter	dilation of an object in an image
DiscreteGaussianDerivativeImageFilter	Calculates image derivatives using discrete derivative gaussian kernels. This filter calculates Gaussian derivative by separable convolution of an image and a discrete Gaussian derivative operator (kernel).
DiscreteGaussianImageFilter	Blurs an image by separable convolution with discrete gaussian kernels. This filter performs Gaussian blurring by separable convolution of an image and a discrete Gaussian operator (kernel).
DisplacementFieldJacobianDeterminantFilter	Computes a scalar image from a vector image (e.g., deformation field) input, where each output scalar at each pixel is the Jacobian determinant of the vector field at that location. This calculation is correct in the case where the vector image is a "displacement" from the current location. The computation for the jacobian determinant is: det[dT/dx] = det[I + du/dx].
DivideFloorImageFilter	Implements pixel-wise generic operation of two images, or of an image and a constant.
DivideImageFilter	Pixel-wise division of two images.
DivideRealImageFilter	Implements pixel-wise generic operation of two images, or of an image and a constant.
DoubleThresholdImageFilter	Binarize an input image using double thresholding.
EdgePotentialImageFilter	Computes the edge potential of an image from the image gradient.
EqualImageFilter	Implements pixel-wise generic operation of two images, or of an image and a constant.
ErodeObjectMorphologyImageFilter	Erosion of an object in an image.
ExpImageFilter	Computes the exponential function of each pixel.
ExpNegativeImageFilter	Computes the function exp(-K.x) for each input pixel.
ExpandImageFilter	Expand the size of an image by an integer factor in each dimension.
ExtractImageFilter	Decrease the image size by cropping the image to the selected region bounds.
FFTConvolutionImageFilter	Convolve a given image with an arbitrary image kernel using multiplication in the Fourier domain.
FFTNormalizedCorrelationImageFilter	Calculate normalized cross correlation using FFTs.
FFTPadImageFilter	Pad an image to make it suitable for an FFT transformation.
FFTShiftImageFilter	Shift the zero-frequency components of a Fourier transform to the center of the image.
FastApproximateRankImageFilter	A separable rank filter.
FastMarchingBaseImageFilter	Apply the Fast Marching method to solve an Eikonal equation on an image.
FastMarchingImageFilter	Solve an Eikonal equation using Fast Marching.
FastMarchingUpwindGradientImageFilter	Generates the upwind gradient field of fast marching arrival times.
FastSymmetricForcesDemonsRegistrationFilter	Deformably register two images using a symmetric forces demons algorithm.
FlipImageFilter	Flips an image across user specified axes.
ForwardFFTImageFilter	Base class for forward Fast Fourier Transform .
GaborImageSource	Generate an n-dimensional image of a Gabor filter.
GaussianImageSource	Generate an n-dimensional image of a Gaussian.
GeodesicActiveContourLevelSetImageFilter	Segments structures in images based on a user supplied edge potential map.
GradientAnisotropicDiffusionImageFilter	

Continued on next page

Table 1 – continued from previous page

Filter Name	Brief Description
GradientImageFilter	Computes the gradient of an image using directional derivatives.
GradientMagnitudeImageFilter	Computes the gradient magnitude of an image region at each pixel.
GradientMagnitudeRecursiveGaussianImageFilter	Computes the Magnitude of the Gradient of an image by convolution with the first derivative of a Gaussian.
GradientRecursiveGaussianImageFilter	Computes the gradient of an image by convolution with the first derivative of a Gaussian.
GrayscaleConnectedClosingImageFilter	Enhance pixels associated with a dark object (identified by a seed pixel) where the dark object is surrounded by a brigher object.
GrayscaleConnectedOpeningImageFilter	Enhance pixels associated with a bright object (identified by a seed pixel) where the bright object is surrounded by a darker object.
GrayscaleDilateImageFilter	Grayscale dilation of an image.
GrayscaleErodeImageFilter	Grayscale erosion of an image.
GrayscaleFillholeImageFilter	Remove local minima not connected to the boundary of the image.
GrayscaleGeodesicDilateImageFilter	geodesic gray scale dilation of an image
GrayscaleGeodesicErodeImageFilter	geodesic gray scale erosion of an image
GrayscaleGrindPeakImageFilter	Remove local maxima not connected to the boundary of the image.
GrayscaleMorphologicalClosingImageFilter	gray scale dilation of an image
GrayscaleMorphologicalOpeningImageFilter	gray scale dilation of an image
GreaterEqualImageFilter	Implements pixel-wise generic operation of two images, or of an image and a constant.
GreaterImageFilter	Implements pixel-wise generic operation of two images, or of an image and a constant.
GridImageSource	Generate an n-dimensional image of a grid.
HConcaveImageFilter	Identify local minima whose depth below the baseline is greater than h.
HConvexImageFilter	Identify local maxima whose height above the baseline is greater than h.
HMaximaImageFilter	Suppress local maxima whose height above the baseline is less than h.
HMinimaImageFilter	Suppress local minima whose depth below the baseline is less than h.
HalfHermitianToRealInverseFFTImageFilter	Base class for specialized complex-to-real inverse Fast Fourier Transform .
HausdorffDistanceImageFilter	Computes the Hausdorff distance between the set of non-zero pixels of two images.
HistogramMatchingImageFilter	Normalize the grayscale values between two images by histogram matching.
HuangThresholdImageFilter	Threshold an image using the Huang Threshold.
IntensityWindowingImageFilter	Applies a linear transformation to the intensity levels of the input Image that are inside a user-defined interval. Values below this interval are mapped to a constant. Values over the interval are mapped to another constant.
IntermodesThresholdImageFilter	Threshold an image using the Intermodes Threshold.
InverseDeconvolutionImageFilter	The direct linear inverse deconvolution filter.
InverseDisplacementFieldImageFilter	Computes the inverse of a displacement field.
InverseFFTImageFilter	Base class for inverse Fast Fourier Transform .
InvertDisplacementFieldImageFilter	Iteratively estimate the inverse field of a displacement field.
InvertIntensityImageFilter	Invert the intensity of an image.
IsoContourDistanceImageFilter	Compute an approximate distance from an interpolated isocontour to the close grid points.
IsoDataThresholdImageFilter	Threshold an image using the IsoData Threshold.
IsolatedConnectedImageFilter	Label pixels that are connected to one set of seeds but not another.
IsolatedWatershedImageFilter	Isolate watershed basins using two seeds.
IterativeInverseDisplacementFieldImageFilter	Computes the inverse of a displacement field.
JoinSeriesImageFilter	Join N-D images into an (N+1)-D image.
KittlerIllingworthThresholdImageFilter	Threshold an image using the KittlerIllingworth Threshold.

Continued on next page

Table 1 – continued from previous page

Filter Name	Brief Description
LabelContourImageFilter	Labels the pixels on the border of the objects in a labeled image.
LabelImageToLabelMapFilter	convert a labeled image to a label collection image
LabelIntensityStatisticsImageFilter	a convenient class to convert a label image to a label map and valuate the statistics attributes at once
LabelMapContourOverlayImageFilter	Apply a colormap to the contours (outlines) of each object in a label map and superimpose it on top of the feature image.
LabelMapMaskImageFilter	Mask and image with a LabelMap .
LabelMapOverlayImageFilter	Apply a colormap to a label map and superimpose it on an image.
LabelMapToBinaryImageFilter	Convert a LabelMap to a binary image.
LabelMapToLabelImageFilter	Converts a LabelMap to a labeled image.
LabelMapToRGBImageFilter	Convert a LabelMap to a colored image.
LabelOverlapMeasuresImageFilter	Computes overlap measures between the set same set of labels of pixels of two images. Background is assumed to be 0.
LabelOverlayImageFilter	Apply a colormap to a label image and put it on top of the input image.
LabelSetDilateImageFilter	Class for binary morphological erosion of label images.
LabelSetErodeImageFilter	Class for binary morphological erosion of label images.
LabelShapeStatisticsImageFilter	Converts a label image to a label map and valuates the shape attributes.
LabelStatisticsImageFilter	Given an intensity image and a label map, compute min, max, variance and mean of the pixels associated with each label or segment.
LabelToRGBImageFilter	Apply a colormap to a label image.
LabelUniqueLabelMapFilter	Make sure that the objects are not overlapping.
LabelVotingImageFilter	This filter performs pixelwise voting among an arbitrary number of input images, where each of them represents a segmentation of the same scene (i.e., image).
LandweberDeconvolutionImageFilter	Deconvolve an image using the Landweber deconvolution algorithm.
LaplacianImageFilter	
LaplacianRecursiveGaussianImageFilter	Computes the Laplacian of Gaussian (LoG) of an image.
LaplacianSegmentationLevelSetImageFilter	Segments structures in images based on a second derivative image features.
LaplacianSharpeningImageFilter	This filter sharpens an image using a Laplacian. LaplacianSharpening highlights regions of rapid intensity change and therefore highlights or enhances the edges. The result is an image that appears more in focus.
LessEqualImageFilter	Implements pixel-wise generic operation of two images, or of an image and a constant.
LessImageFilter	Implements pixel-wise generic operation of two images, or of an image and a constant.
LevelSetMotionRegistrationFilter	Deformably register two images using level set motion.
LiThresholdImageFilter	Threshold an image using the Li Threshold.
Log10ImageFilter	Computes the log10 of each pixel.
LogImageFilter	Computes the log() of each pixel.
MagnitudeAndPhaseToComplexImageFilter	Implements pixel-wise conversion of magnitude and phase data into complex voxels.
MaskImageFilter	Mask an image with a mask.
MaskNegatedImageFilter	Mask an image with the negation (or logical compliment) of a mask.
MaskedFFTNormalizedCorrelationImageFilter	Calculate masked normalized cross correlation using FFTs.
MaximumEntropyThresholdImageFilter	Threshold an image using the MaximumEntropy Threshold.
MaximumImageFilter	Implements a pixel-wise operator Max(a,b) between two images.
MaximumProjectionImageFilter	Maximum projection.
MeanImageFilter	Applies an averaging filter to an image.
MeanProjectionImageFilter	Mean projection.
MedianImageFilter	Applies a median filter to an image.

Continued on next page

Table 1 – continued from previous page

Filter Name	Brief Description
MedianProjectionImageFilter	Median projection.
MergeLabelMapFilter	Merges several Label Maps.
MinMaxCurvatureFlowImageFilter	Denoise an image using min/max curvature flow.
MinimumImageFilter	Implements a pixel-wise operator Min(a,b) between two images.
MinimumMaximumImageFilter	Computes the minimum and the maximum intensity values of an image.
MinimumProjectionImageFilter	Minimum projection.
MirrorPadImageFilter	Increase the image size by padding with replicants of the input image value.
ModulusImageFilter	Computes the modulus (x % dividend) pixel-wise.
MomentsThresholdImageFilter	Threshold an image using the Moments Threshold.
MorphologicalGradientImageFilter	gray scale dilation of an image
MorphologicalWatershedFromMarkersImageFilter	Morphological watershed transform from markers.
MorphologicalWatershedImageFilter	Watershed segmentation implementation with morphogical operators.
MultiLabelSTAPLEImageFilter	This filter performs a pixelwise combination of an arbitrary number of input images, where each of them represents a segmentation of the same scene (i.e., image).
MultiplyImageFilter	Pixel-wise multiplication of two images.
N4BiasFieldCorrectionImageFilter	Implementation of the N4 bias field correction algorithm.
NaryAddImageFilter	Pixel-wise addition of N images.
NaryMaximumImageFilter	Computes the pixel-wise maximum of several images.
NeighborhoodConnectedImageFilter	Label pixels that are connected to a seed and lie within a neighborhood.
NoiseImageFilter	Calculate the local noise in an image.
NormalizeImageFilter	Normalize an image by setting its mean to zero and variance to one.
NormalizeToConstantImageFilter	Scales image pixel intensities to make the sum of all pixels equal a user-defined constant.
NormalizedCorrelationImageFilter	Computes the normalized correlation of an image and a template.
NotEqualImageFilter	Implements pixel-wise generic operation of two images, or of an image and a constant.
NotImageFilter	Implements the NOT logical operator pixel-wise on an image.
ObjectnessMeasureImageFilter	Enhance M-dimensional objects in N-dimensional images.
OpeningByReconstructionImageFilter	Opening by reconstruction of an image.
OrImageFilter	Implements the OR bitwise operator pixel-wise between two images.
OtsuMultipleThresholdsImageFilter	Threshold an image using multiple Otsu Thresholds.
OtsuThresholdImageFilter	Threshold an image using the Otsu Threshold.
PasteImageFilter	Paste an image into another image.
PatchBasedDenoisingImageFilter	Derived class implementing a specific patch-based denoising algorithm, as detailed below.
PermuteAxesImageFilter	Permutes the image axes according to a user specified order.
PhysicalPointImageSource	Generate an image of the physical locations of each pixel.
PowImageFilter	Computes the powers of 2 images.
ProjectedLandweberDeconvolutionImageFilter	Deconvolve an image using the projected Landweber deconvolution algorithm.
RankImageFilter	Rank filter of a greyscale image.
RealAndImaginaryToComplexImageFilter	ComposeImageFilter combine several scalar images into a multicomponent image.
RealToHalfHermitianForwardFFTImageFilter	Base class for specialized real-to-complex forward Fast Fourier Transform .
ReconstructionByDilationImageFilter	grayscale reconstruction by dilation of an image
ReconstructionByErosionImageFilter	grayscale reconstruction by erosion of an image
RecursiveGaussianImageFilter	Base class for computing IIR convolution with an approximation of a Gaussian kernel.

Continued on next page

Table 1 – continued from previous page

Filter Name	Brief Description
RegionOfInterestImageFilter	Extract a region of interest from the input image.
RegionalMaximaImageFilter	Produce a binary image where foreground is the regional maxima of the input image.
RegionalMinimaImageFilter	Produce a binary image where foreground is the regional minima of the input image.
RelabelComponentImageFilter	Relabel the components in an image such that consecutive labels are used.
RelabelLabelMapFilter	This filter relabels the LabelObjects; the new labels are arranged consecutively with consideration for the background value.
RenyiEntropyThresholdImageFilter	Threshold an image using the RenyiEntropy Threshold.
ResampleImageFilter	Resample an image via a coordinate transform.
RescaleIntensityImageFilter	Applies a linear transformation to the intensity levels of the input Image .
RichardsonLucyDeconvolutionImageFilter	Deconvolve an image using the Richardson-Lucy deconvolution algorithm.
RoundImageFilter	Rounds the value of each pixel.
SLICImageFilter	Simple Linear Iterative Clustering (SLIC) super-pixel segmentation.
STAPLEImageFilter	The STAPLE filter implements the Simultaneous Truth and Performance Level Estimation algorithm for generating ground truth volumes from a set of binary expert segmentations.
SaltAndPepperNoiseImageFilter	Alter an image with fixed value impulse noise, often called salt and pepper noise.
ScalarChanAndVeseDenseLevelSetImageFilter	Dense implementation of the Chan and Vese multiphase level set image filter.
ScalarConnectedComponentImageFilter	A connected components filter that labels the objects in an arbitrary image. Two pixels are similar if they are within threshold of each other. Uses ConnectedComponentFunctorImageFilter .
ScalarImageKmeansImageFilter	Classifies the intensity values of a scalar image using the K-Means algorithm.
ScalarToRGBColormapImageFilter	Implements pixel-wise intensity->rgb mapping operation on one image.
ShanbhagThresholdImageFilter	Threshold an image using the Shanbhag Threshold.
ShapeDetectionLevelSetImageFilter	Segments structures in images based on a user supplied edge potential map.
ShiftScaleImageFilter	Shift and scale the pixels in an image.
ShotNoiseImageFilter	Alter an image with shot noise.
ShrinkImageFilter	Reduce the size of an image by an integer factor in each dimension.
SigmoidImageFilter	Computes the sigmoid function pixel-wise.
SignedDanielssonDistanceMapImageFilter	
SignedMaurerDistanceMapImageFilter	This filter calculates the Euclidean distance transform of a binary image in linear time for arbitrary dimensions.
SimilarityIndexImageFilter	Measures the similarity between the set of non-zero pixels of two images.
SimpleContourExtractorImageFilter	Computes an image of contours which will be the contour of the first image.
SinImageFilter	Computes the sine of each pixel.
SliceImageFilter	Slices an image based on a starting index and a stopping index, and a step size.
SmoothingRecursiveGaussianImageFilter	Computes the smoothing of an image by convolution with the Gaussian kernels implemented as IIR filters.
SobelEdgeDetectionImageFilter	A 2D or 3D edge detection using the Sobel operator.
SpeckleNoiseImageFilter	Alter an image with speckle (multiplicative) noise.
SqrtImageFilter	Computes the square root of each pixel.
SquareImageFilter	Computes the square of the intensity values pixel-wise.
SquaredDifferenceImageFilter	Implements pixel-wise the computation of squared difference.
StandardDeviationProjectionImageFilter	Mean projection.
StatisticsImageFilter	Compute min. max, variance and mean of an Image .

Continued on next page

Table 1 – continued from previous page

Filter Name	Brief Description
StochasticFractalDimensionImageFilter	This filter computes the stochastic fractal dimension of the input image.
SubtractImageFilter	Pixel-wise subtraction of two images.
SumProjectionImageFilter	Sum projection.
SymmetricForcesDemonsRegistrationFilter	Deformably register two images using the demons algorithm.
TanImageFilter	Computes the tangent of each input pixel.
TernaryAddImageFilter	Pixel-wise addition of three images.
TernaryMagnitudeImageFilter	Compute the pixel-wise magnitude of three images.
TernaryMagnitudeSquaredImageFilter	Compute the pixel-wise squared magnitude of three images.
ThresholdImageFilter	Set image values to a user-specified value if they are below, above, or between simple threshold values.
ThresholdMaximumConnectedComponentsImageFilter	Finds the threshold value of an image based on maximizing the number of objects in the image that are larger than a given minimal size.
ThresholdSegmentationLevelSetImageFilter	Segments structures in images based on intensity values.
TikhonovDeconvolutionImageFilter	An inverse deconvolution filter regularized in the Tikhonov sense.
TileImageFilter	Tile multiple input images into a single output image.
TobogganImageFilter	toboggan image segmentation The Toboggan segmentation takes a gradient magnitude image as input and produces an (over-)segmentation of the image based on connecting each pixel to a local minimum of gradient. It is roughly equivalent to a watershed segmentation of the lowest level.
TransformToDisplacementFieldFilter	Generate a displacement field from a coordinate transform.
TriangleThresholdImageFilter	Threshold an image using the Triangle Threshold.
UnaryMinusImageFilter	Implements pixel-wise generic operation on one image.
UnsharpMaskImageFilter	Edge enhancement filter.
ValuedRegionalMaximaImageFilter	Transforms the image so that any pixel that is not a regional maxima is set to the minimum value for the pixel type. Pixels that are regional maxima retain their value.
ValuedRegionalMinimaImageFilter	Transforms the image so that any pixel that is not a regional minima is set to the maximum value for the pixel type. Pixels that are regional minima retain their value.
VectorConfidenceConnectedImageFilter	Segment pixels with similar statistics using connectivity.
VectorConnectedComponentImageFilter	A connected components filter that labels the objects in a vector image. Two vectors are pointing similar directions if one minus their dot product is less than a threshold. Vectors that are 180 degrees out of phase are similar. Assumes that vectors are normalized.
VectorIndexSelectionCastImageFilter	Extracts the selected index of the vector that is the input pixel type.
VectorMagnitudeImageFilter	Take an image of vectors as input and produce an image with the magnitude of those vectors.
VotingBinaryHoleFillingImageFilter	Fills in holes and cavities by applying a voting operation on each pixel.
VotingBinaryImageFilter	Applies a voting operation in a neighborhood of each pixel.
VotingBinaryIterativeHoleFillingImageFilter	Fills in holes and cavities by iteratively applying a voting operation.
WarpImageFilter	Warps an image using an input displacement field.
WhiteTopHatImageFilter	White top hat extracts local maxima that are larger than the structuring element.
WienerDeconvolutionImageFilter	The Wiener deconvolution image filter is designed to restore an image convolved with a blurring kernel while keeping noise enhancement to a minimum.
WrapPadImageFilter	Increase the image size by padding with replicants of the input image value.
XorImageFilter	Computes the XOR bitwise operator pixel-wise between two images.
YenThresholdImageFilter	Threshold an image using the Yen Threshold.
ZeroCrossingBasedEdgeDetectionImageFilter	This filter implements a zero-crossing based edge detecor.

Continued on next page

Table 1 – continued from previous page

Filter Name	Brief Description
ZeroCrossingImageFilter	This filter finds the closest pixel to the zero-crossings (sign changes) in a signed itk::Image .
ZeroFluxNeumannPadImageFilter	Increase the image size by padding according to the zero-flux Neumann boundary condition.

CHAPTER 7

Building SimpleITK

In many cases a user does not need to build SimpleITK because of the available pre-built binaries (see *Downloading the binaries*). However there are several reasons a user might prefer to **build SimpleITK from source code**:

- The binary files for your programming language of choice are not (yet) distributed
- You want the live on the bleeding edge by using the latest-and-greatest version of SimpleITK
- You want to wrap your own filters using the SimpleITK infrastructure
- You want to contribute to the development of SimpleITK
- To use the SimpleITK's C++ interface and/or use ITK directly

7.1 Prerequisites

To build SimpleITK you need:

- A recent version of CMake >= 3.10 with SSL support for https.
- A supported *compiler*.
- To use the latest developmental version, source code can be downloaded with git >= 1.65
  - Git is required if building SimpleITK using "SuperBuild" (see below) to automatically download the matching version of ITK, SWIG, etc. . .
  - Windows users may prefer msysGit
- It is recommended to have numpy installed when testing Python bindings
- The R package requires R version 3.3 or greater.
- The Lua package requires Lua version 5.1 or greater.

7.2 Recipes / Formulas / Short cuts

Before you start please make sure you have the required *Prerequisites* installed.

For some environments we have short cuts, scripts, for automated building of SimpleITK (see their repository for more details):

- For **Python**: The scikit-build based distutils based setup.py frontend can be used to build, install, and package SimpleITK for Python.
- For the **Anaconda Python** distribution: The recipe and instructions for the SimpleITK build are in this GitHub repository.
- For the **R language** (version 3.3 or greater): A devtools installer and instructions are available from this GitHub repository.
- **On the Mac**, with the Homebrew package manager, a SimpleITK formula is available: https://github.com/Homebrew/homebrew-science/blob/master/simpleitk.rb for multiple language wrappings.
- For the **Lua language** with the Luarocks module deployment system, a SimpleITK rockspec is available at the Luarocks repository or from this GitHub repository.

7.3 Source code

If one of the above language specific front-ends are not used then SimpleITK must be built directly.

Before you start please make sure you have the required *Prerequisites* installed.

All of the instructions assume you are working on the command line.

Words of caution for building on the Windows operating system:

- Windows has issues with long directory paths. We recommend cloning the source code near the root (e.g. C:\src).
- To avoid potential issues do not clone the source into a path which has spaces in the directory name (e.g. C:\Users\SimpleITK Source).

First obtain the SimpleITK source code:

1. Download the latest development version using git

```
git clone  https://itk.org/SimpleITK.git
```

7.3.1 Building using SuperBuild

After downloading SimpleITK's source code we **STRONGLY** recommend running cmake on the SuperBuild subdirectory of SimpleITK. Execute the following commands in the parent of the SimpleITK source directory to configure the SuperBuild:

```
mkdir SimpleITK-build
cd SimpleITK-build
cmake ../SimpleITK/SuperBuild
```

The SuperBuild will automatically download and build the matching versions of ITK, SWIG, Lua, and GTest (if testing is enabled) needed to compile SimpleITK.

If you get an error message saying that ITK_DIR is not set then, you did not correctly point cmake to the SuperBuild sub-directory. Please erase your binary directory, and point cmake to the SimpleITK/SuperBuild sub-directory.

The CMake configuration process should automatically find supported languages and enable SimpleITK wrapping for them. To manually enable a language toggle the appropriate WRAP_LANGUAGE cmake variable to ON. Verify and correct the advanced cmake variables for the language specific executable, libraries and include directories. For example if you have multiple Python installations ensure that all related Python variables refer to the same versions.

Then use your make utility or your cmake chosen build utility to build SimpleITK. As the SimpleITK build process may take a while, it is important to use the appropriate flags to enable multi-process compilation i.e. "-j" for make, "/MP" for Visual Studio, or use the CMake Ninja generator.

7.3.2 Building Manually

By not using the superbuild, you must manually specify all dependencies used during the building of SimpleITK instead of using the known working versions provided by the superbuild as external projects. This may be useful if you are providing a system package of SimpleITK or tightly integrating it into another build system. The versions of external projects used and tested by SimpleITK can be found by examining the External CMake files in the Superbuild sub-directory.

Additional Prerequisites

The following are dependencies you will need when not using the SuperBuild:

1. Setup the prerequisites as described above (i.e. CMake and supported compiler)
2. Insight Toolkit (ITK) the version specified in the External_ITK.cmake file is the version of ITK used for the binary release. This can be seen as the minimum version of ITK to be used with SimpleITK, as future ITK versions are generally backwards compatible.
3. Lua 5.1
4. SWIG >= 3.0.11
5. GTest or Google >= 1.0.8 is needed if testing is enabled.

Configuration and Building

After the source code is obtained, SimpleITK can be configured on Unix-like systems like so:

```
mkdir SimpleITK-build
cd SimpleITK-build
cmake ../SimpleITK
```

If all the dependencies are installed in standard places, then the CMake configuration should detect them properly. Otherwise, if there are configuration errors, the proper CMake variable should be set. CMake variables can be either set with a CMake interactive GUI such as *ccmake* or *cmake-qt*, as arguments on the command line by using the following format: *-D<var>=<value>*, or by editing the CMakeCache.txt file.

After proper configuration, SimpleITK can be built:

```
make -j$(nproc)
```

7.3.3 Advanced Build Options

SimpleITK is aware of the modularity of ITK and automatically enables and disables filters based on which modules are available from the ITK build which SimpleITK is compiled against. This makes it possible to customize SimpleITK to be a small library or to wrap additional ITK remote modules simply by configuring ITK with the desired modules enabled.

For example, the CoherenceEnhancingDiffusionImageFilter is an optional filter in SimpleITK as it's part of the ITK remote module AnisotropicDiffusionLBR. This remote module is not enabled by default when building ITK and SimpleITK. To enable it when using SimpleITK's Superbuild add *-DModule_AnisotropicDiffusionLBR:BOOL=ON* to the command line or in the CMake GUI press the "Add Entry" button to define the variable as above.

SimpleITK has a very flexible and robust build system utilizing CMake. It enables packagers to build SimpleITK in a variety of ways to suit their requirements and minimize recompilation of SimpleITK so that it can be wrapped for many different languages. Each of the language wrapping sub-directories e.g. "Wrapping/Python" can be configured and built as an independent project which is dependent on SimpleITK as an installed package of its libraries and header filers.

7.3.4 Testing

After compilation the prudent thing to do is to test SimpleITK to ensure that your build is stable and suitable for installation and use. The following commands execute the SimpleITK tests.

```
cd SimpleITK-build/SimpleITK-build
ctest .
```

On Windows you will need to specify configuration. Typically that would be the Release configuration, as such:

```
cd SimpleITK-build/SimpleITK-build
ctest -C Release
```

7.3.5 Installation from Build Tree

Python Installation

To install a built python package into the system Python, as root run:

```
cd SimpleITK-build/Wrapping/Python
python Packaging/setup.py install
```

Alternatively, a Python virtual environment can be created and the distribution installed there.

A Python Wheel file (.whl) can be created in the "Wrapping/Python/dist" directory, by building the "dist" target. If you have used the Superbuild with the "make" generator then issue the following command:

```
make -C SimpleITK-build dist
```

R Installation

To install a built R package:

```
cd SimpleITK-build/Wrapping/R/Packaging
R CMD INSTALL SimpleITK
```

This will install the R package "SimpleITK" in /usr/local as root or your local R installation directory.

If you are working in a multi-user environment, and are considerate of your fellow users you can install the package in a local directory:

1. Create a local directory where you will install your R packages

```
mkdir my_R_libs
```

2. Add an environment variable to your .bashrc

```
export R_LIBS="/path_to/my_R_libs"
```

3. source your .bashrc and check the R library path, in an R shell

```
.libPaths()
```

4. install

```
cd SimpleITK-build/Wrapping/R/Packaging
R CMD INSTALL -l /path_to/my_R_libs SimpleITK
```

CHAPTER 8

Setting Up Eclipse and Visual Studio

8.1 Java

Java and SimpleITK are a natural fit. Like the bindings of other languages wrapped by SimpleITK, SimpleITK's Java bindings have a language-specific component (traditional Jar file), and a native component (native shared library). This combination requires a little more setup, but is largely transparent to the developer.

8.1.1 Build/Install SimpleITK

For Windows, you have two options:

- *Download* the binary for SimpleITK
- *Build* the binary

For any other OS, you must build the binaries yourself.

8.1.2 Set Up Eclipse

Download Eclipse on your platform of choice. Eclipse is a commonly used integrated development environemnt (IDE) used for Java because it makes development, debugging and deployment streamlined.

Then, create a new project in Eclipse. Choose *File –> New –> Project...*, choosing Java Project in the Eclipse project wizard dialog, and name the project as you like. In this example, our project is called SimpleITK Demo. Create a new class by choosing *File –> New –> Class*, or simply copy the code below and paste into the project item in the Package Explorer view and Eclipse will automatically create the class and hierarchy for you.

8.1.3 GaussianExample

Here is our first class *Code/GaussianExample.java*:

```
package org.itk.simple.example;

import org.itk.simple.Image;
import org.itk.simple.SimpleITK;

public class GaussianExample {

        /**
         * @param args
         */
        public static void main(String[] args) {

                if (args.length < 2) {
                        System.err.println("Usage: 'Gaussian <input> <output>'");
                        System.exit(1);
                }
                System.out.println("Starting to blur " + args[0]);
                // Grab a file (there will be an error if the image is not scalar)
                Image image = SimpleITK.readImage(args[0]);
                Image output = SimpleITK.discreteGaussian(image);
                SimpleITK.writeImage(output, args[1]);
                System.out.println("Finished blurring, writing to " + args[1]);

        }

}
```

If Eclipse is working as expected, you should see errors on lines 16-20. These errors occurs because we have not told Eclipse where to find SimpleITK's jar file.

8.1.4 Add SimpleITK to the Build Path

Right click on the project in the *Package Explorer* view and choose *Build Path –> Configure Build Path...*

In the *Properties* dialog, click on *Java Build Path* and *Add External JARs...* then navigate to the SimpleITK jar file. When selected, click the down arrow to expose the options for the jar.

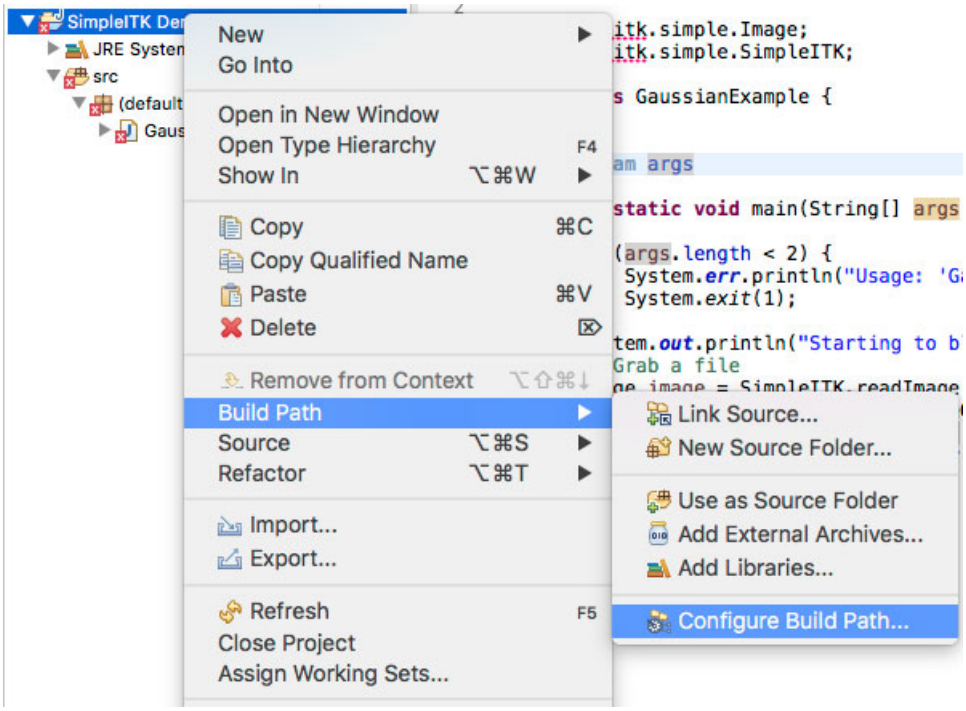
The three options of interest are: *Source attachment*, *Javdoc location* and *Native library location*. The *Source attachment* specifies where the source code for the SimpleITK jar file resides. In our case, it is distributed as *simpleitk-source.x.x.x.jar* where *x.x.x* is the version number of SimpleITK. The source attachment is useful for debugging the SimpleITK library, if necessary, because it allows the debugger to step through classes provided in the SimpleITK jar file. This setting is optional.

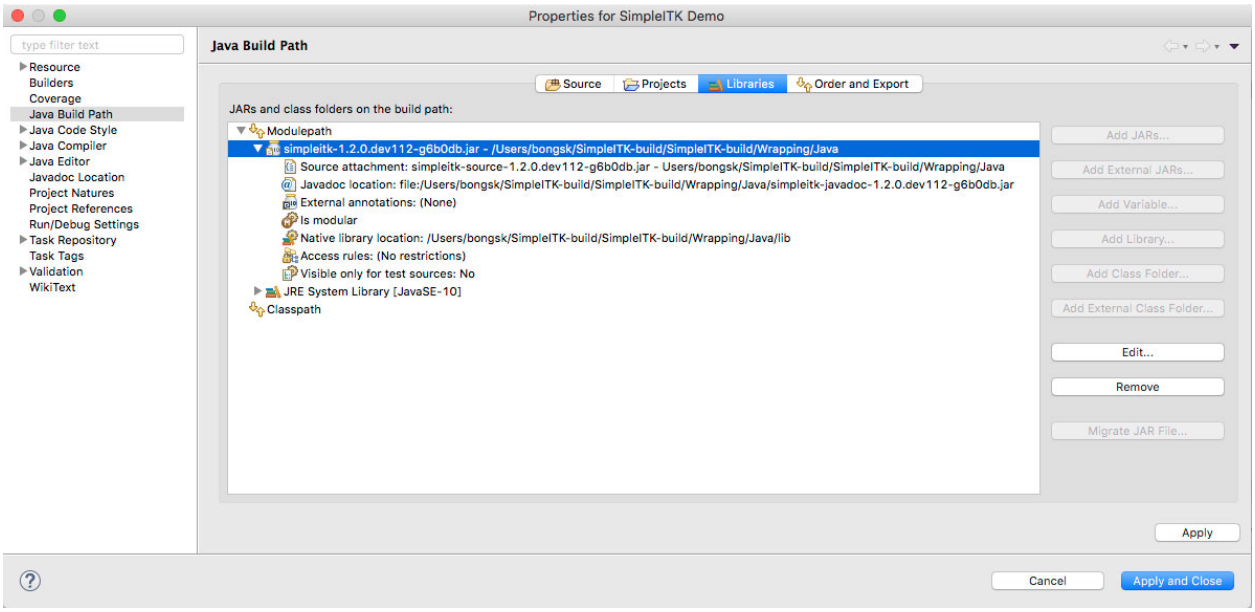
The *Javadoc location* is also optional, but extremely helpful in developing with Java. Having Javadoc available provides Eclipse with in-line documentation for each function, if provided. We highly recommend supplying the Javadoc location to Eclipse.

The last option, *Native library location* is required. Because SimpleITK is a C++ library, all functionality is provided through the *JNI (Java Native Interface)* specification. When the SimpleITK classes are loaded, a static block loads the native library to provide all the functionality to Java. This option tells Eclipse where to search for the library; without it a *UnsatisfiedLinkError* is thrown:

```
Exception in thread "main" java.lang.UnsatisfiedLinkError: no SimpleITKJava in java.
↪library.path
                at java.lang.ClassLoader.loadLibrary(ClassLoader.java:1758)
                at java.lang.Runtime.loadLibrary0(Runtime.java:823)
```

(continues on next page)


SimpleITK De
JRE Systen
src
(default
Gaus
New
Go Into
Open in New Window
Open Type Hierarchy F4
Show In ⌥⌘W
Copy ⌘C
Copy Qualified Name
Paste ⌘V
Delete
Remove from Context ⌥⇧⌘↓
Build Path
Source ⌥⌘S
Refactor ⌥⌘T
Import...
Export...
Refresh F5
Close Project
Assign Working Sets...
Link Source...
New Source Folder...
Use as Source Folder
Add External Archives...
Add Libraries...
Configure Build Path...
itk.simple.Image;
itk.simple.SimpleITK;
s GaussianExample {
am args
static void main(String[] args
(args.length < 2) {
System.err.println("Usage: 'G
System.exit(1);
tem.out.println("Starting to b
Grab a file


Properties for SimpleITK Demo
type filter text
Resource
Builders
Coverage
Java Build Path
Java Code Style
Java Compiler
Java Editor
Javadoc Location
Project Natures
Project References
Run/Debug Settings
Task Repository
Task Tags
Validation
WikiText
Java Build Path
Source
Projects
Libraries
Order and Export
JARs and class folders on the build path:
Modulepath
simpleitk-1.2.0.dev112-g6b0db.jar - /Users/bongsk/SimpleITK-build/SimpleITK-build/Wrapping/Java
Source attachment: simpleitk-source-1.2.0.dev112-g6b0db.jar - Users/bongsk/SimpleITK-build/SimpleITK-build/Wrapping/Java
Javadoc location: file:/Users/bongsk/SimpleITK-build/SimpleITK-build/Wrapping/Java/simpleitk-javadoc-1.2.0.dev112-g6b0db.jar
External annotations: (None)
Is modular
Native library location: /Users/bongsk/SimpleITK-build/SimpleITK-build/Wrapping/Java/lib
Access rules: (No restrictions)
Visible only for test sources: No
JRE System Library [JavaSE-10]
Classpath
Add JARs...
Add External JARs...
Add Variable...
Add Library...
Add Class Folder...
Add External Class Folder...
Edit...
Remove
Migrate JAR File...
Apply
Cancel
Apply and Close

(continued from previous page)

```
                at java.lang.System.loadLibrary(System.java:1045)
                at org.itk.simple.SimpleITKJNI.<clinit>(SimpleITKJNI.java:62)
                at org.itk.simple.SimpleITK.readImage(SimpleITK.java:33)
                at org.itk.simple.example.GaussianExample.main(GaussianExample.
↪java:19)
```

Set the *Native library location* to the directory containing the platform specific JNI library, i.e. *libSimpleITKJava.jnilib* on Mac OSX, *libSimpleITKJava.so* on Linux and *SimpleITKJava.dll* on Windows. After providing the library location, our example code runs correctly. When running this example from the command line, the native library location needs to be specified to the JVM, e.g. `-Djava.library.path=/path/to/SimpleITKRuntime`.

8.1.5 SimpleITK Java Conventions

The SimpleITK Java bindings closely follow the C++ conventions, i.e. each class contains the public member functions. However, the functional interface is handled differently in Java. In particular, every static Java function must belong to a class, unlike C++. In SimpleITK, the functional interface is contained in a class called *org.itk.simple.SimpleITK*. This class contains the functional interfaces as static member functions, i.e. *org.itk.simple.SimpleITK.readImage* as shown in *GaussianExample*. The naming conventions for all SimpleITK classes follows the C++ conventions, but member functions and the function interface follow the Java conventions of using *CamelCase* with the first letter lowercase. In Java, the C++ function *itk::simple::ReadImage* becomes *org.itk.simple.SimpleITK.readImage*.

8.2 C#

This guide will show how to set up a C# project in Microsoft Visual Studio 2017 using SimpleITK binaries. The same steps should apply for all other versions of Visual Studio.

8.2.1 Building SimpleITK

For Windows, you have two options:

- *Download* the binary for SimpleITK in your Documents
- *Build* the binary

For any other OS, you must build the binaries yourself.

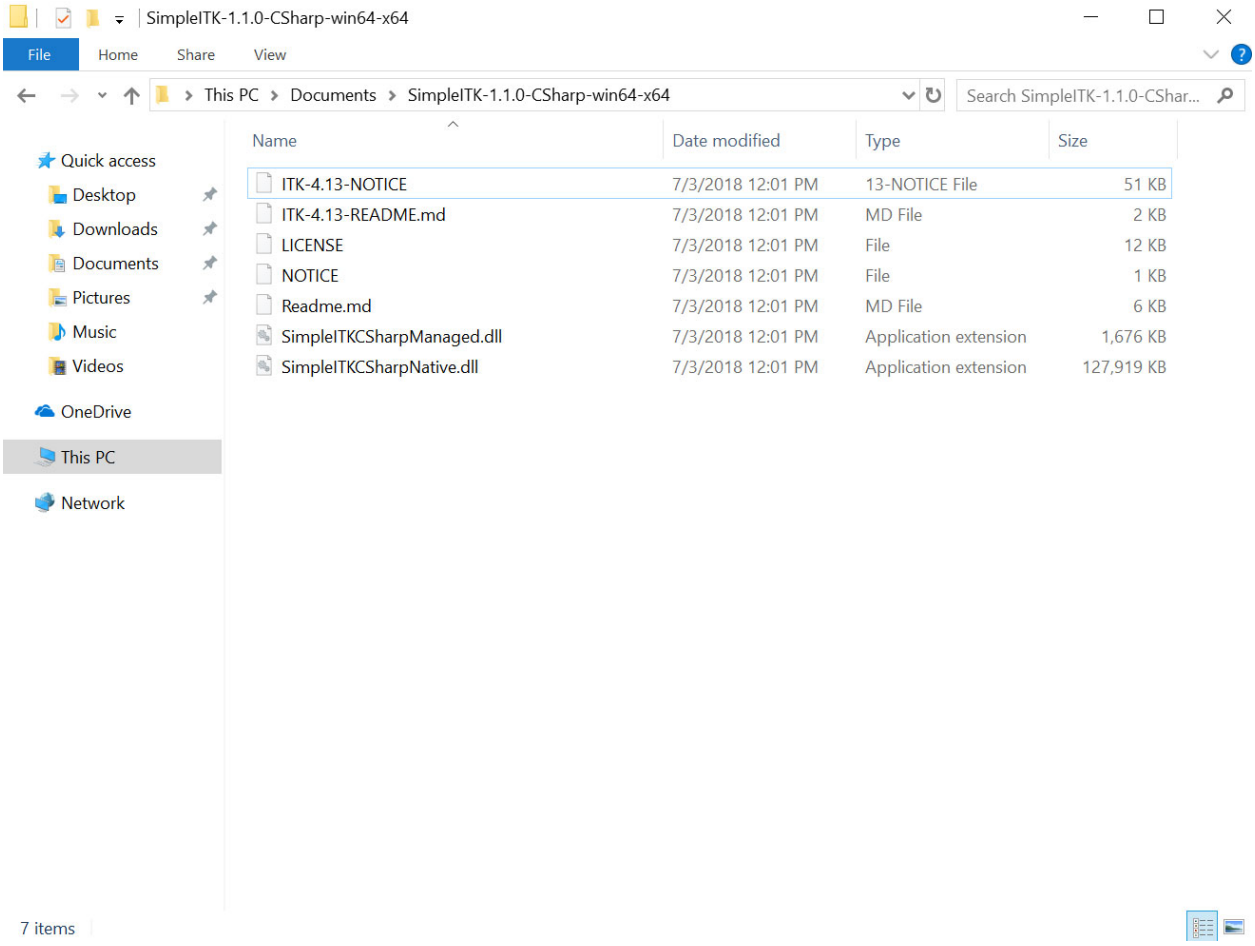
8.2.2 Set Up Visual Studio

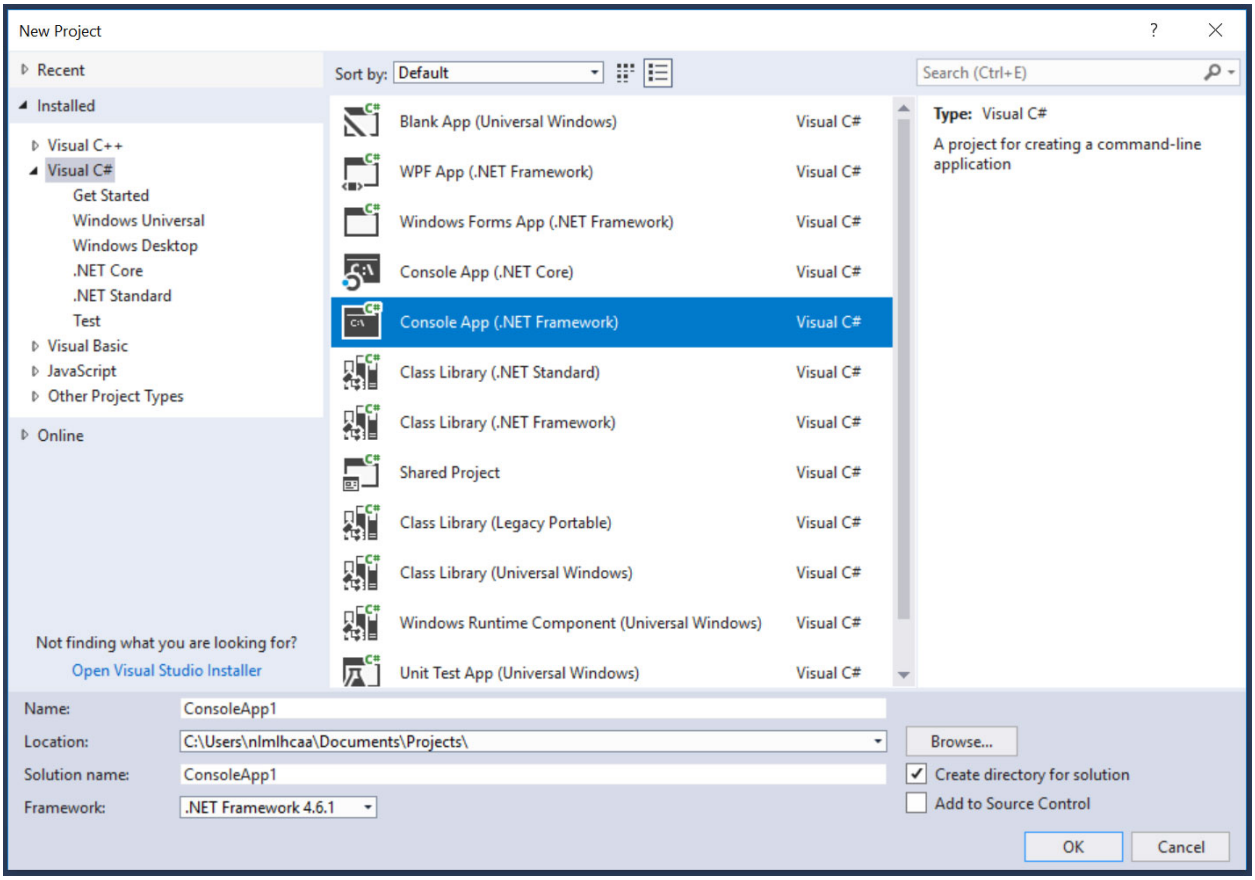
Download Visual Studio if you don't have it.

Then, create a new project in Visual Studio. Choose *File –> New –> Project...*, then select "Visual C#" and "Console App".

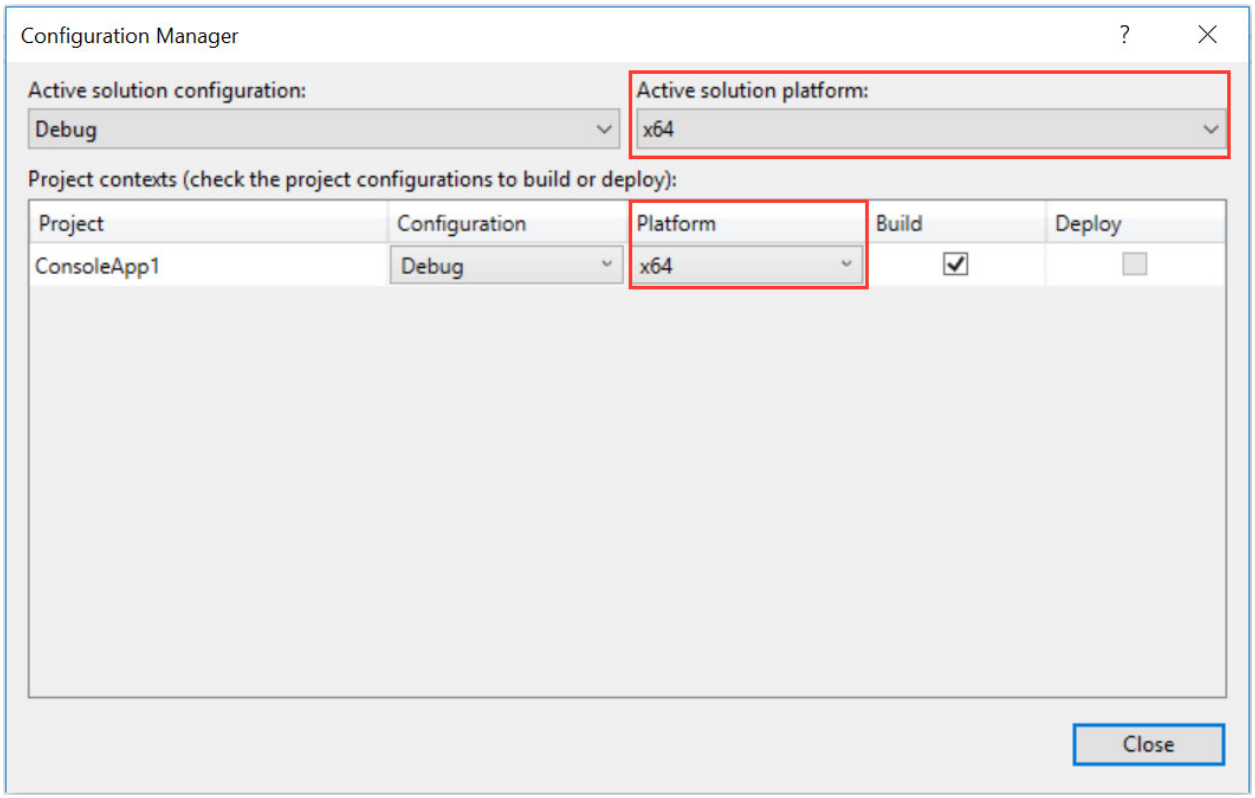
8.2.3 Select Architecture

The SimpleITK binary only supports a single architecture platform. Your project should be configured to match that same platform. By default, in the Toolbar "Debug" is selected for the Solution Configuration and "Any CPU" is selected for the Solution Platform, this needs to be changed.


SimpleITK-1.1.0-CSharp-win64-x64
File
Home
Share
View
This PC
Documents
SimpleITK-1.1.0-CSharp-win64-x64
Search SimpleITK-1.1.0-CSharp...
Quick access
Desktop
Downloads
Documents
Pictures
Music
Videos
OneDrive
This PC
Network
Name
Date modified
Type
Size
ITK-4.13-NOTICE 7/3/2018 12:01 PM 13-NOTICE File 51 KB
ITK-4.13-README.md 7/3/2018 12:01 PM MD File 2 KB
LICENSE 7/3/2018 12:01 PM File 12 KB
NOTICE 7/3/2018 12:01 PM File 1 KB
Readme.md 7/3/2018 12:01 PM MD File 6 KB
SimpleITKCSharpManaged.dll 7/3/2018 12:01 PM Application extension 1,676 KB
SimpleITKCSharpNative.dll 7/3/2018 12:01 PM Application extension 127,919 KB
7 items


New Project
Recent
Installed
Visual C++
Visual C#
Get Started
Windows Universal
Windows Desktop
.NET Core
.NET Standard
Test
Visual Basic
JavaScript
Other Project Types
Online
Not finding what you are looking for?
Open Visual Studio Installer
Sort by: Default
Search (Ctrl+E)
Blank App (Universal Windows) Visual C#
WPF App (.NET Framework) Visual C#
Windows Forms App (.NET Framework) Visual C#
Console App (.NET Core) Visual C#
Console App (.NET Framework) Visual C#
Class Library (.NET Standard) Visual C#
Class Library (.NET Framework) Visual C#
Shared Project Visual C#
Class Library (Legacy Portable) Visual C#
Class Library (Universal Windows) Visual C#
Windows Runtime Component (Universal Windows) Visual C#
Unit Test App (Universal Windows) Visual C#
Type: Visual C#
A project for creating a command-line application
Name: ConsoleApp1
Location: C:\Users\nlmlhcaa\Documents\Projects\
Solution name: ConsoleApp1
Framework: .NET Framework 4.6.1
Browse...
Create directory for solution
Add to Source Control
OK
Cancel

- Bring up the "Configuration Manager" dialog from the menu *BUILD->Configuration Manger...*
- The architecture of the SimpleITK binary needs to be added, and the "Any CPU" architecture needs to be removed. This needs to be done for both the "Active solution platforms" and the "Platform".



8.2.4 Add Managed Library

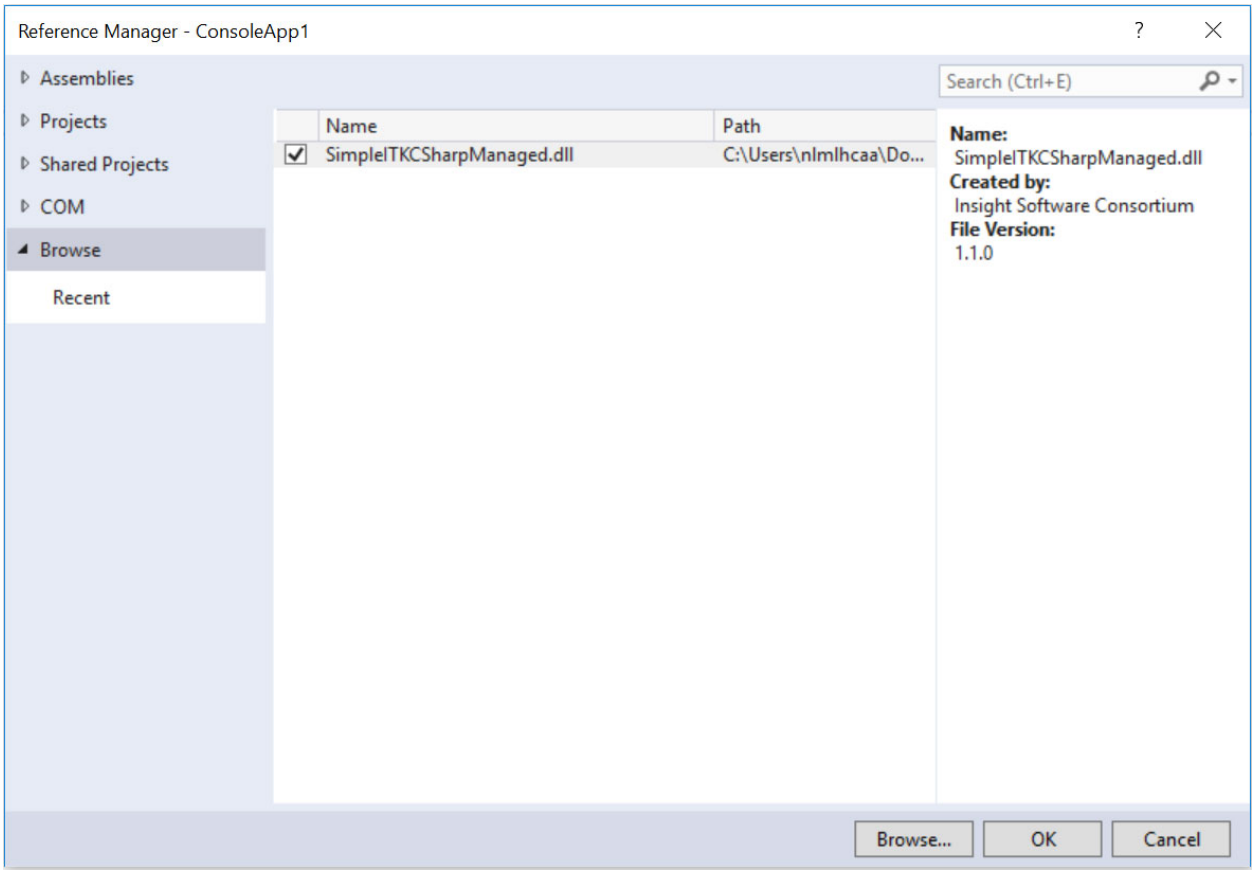
- From the menu bar select *PROJECT->Add Reference...* to bring up the Reference Manager. Click *Browse...* and navigate the file system to unzip "SimpleITKCSharpManaged.dll" from the binary download, then click OK to add.

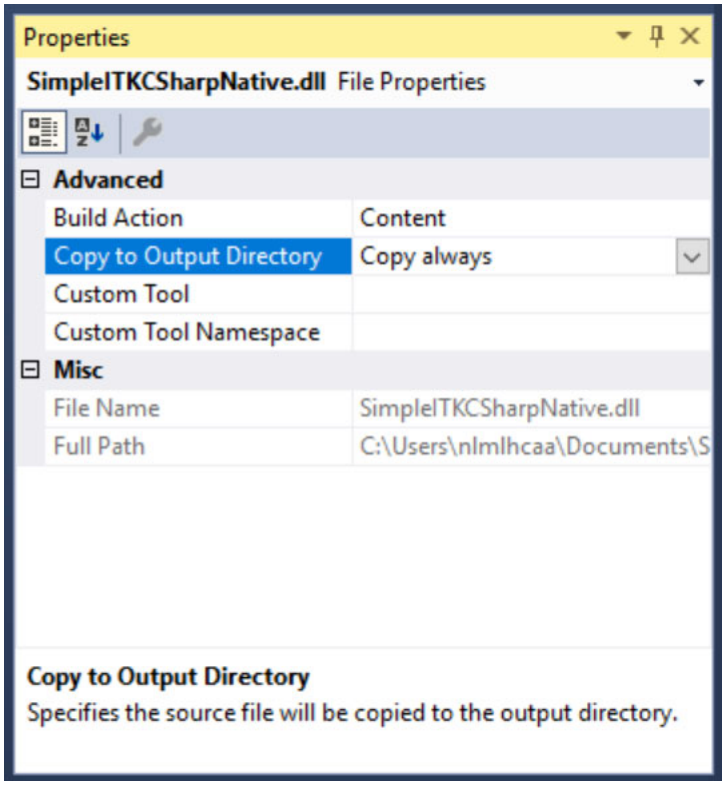
8.2.5 Add Native Library

- From the menu bar select *PROJECT->Add Existing Item...* Select "Executable Files" for the extension type. Then navigate the file system to the unzipped "SimpleITKCSharpNative.dll" file from the binary download. **IMPORTANT** in the "Add" button's pull down menu select "Add As Link".
- In the Solution Explorer right click on the "SimpleITKCSharpNative.dll", and select "Properties".
- Then for "Build Action", choose "Content", and "Copy to OutputDirectory" choose "Copy always".

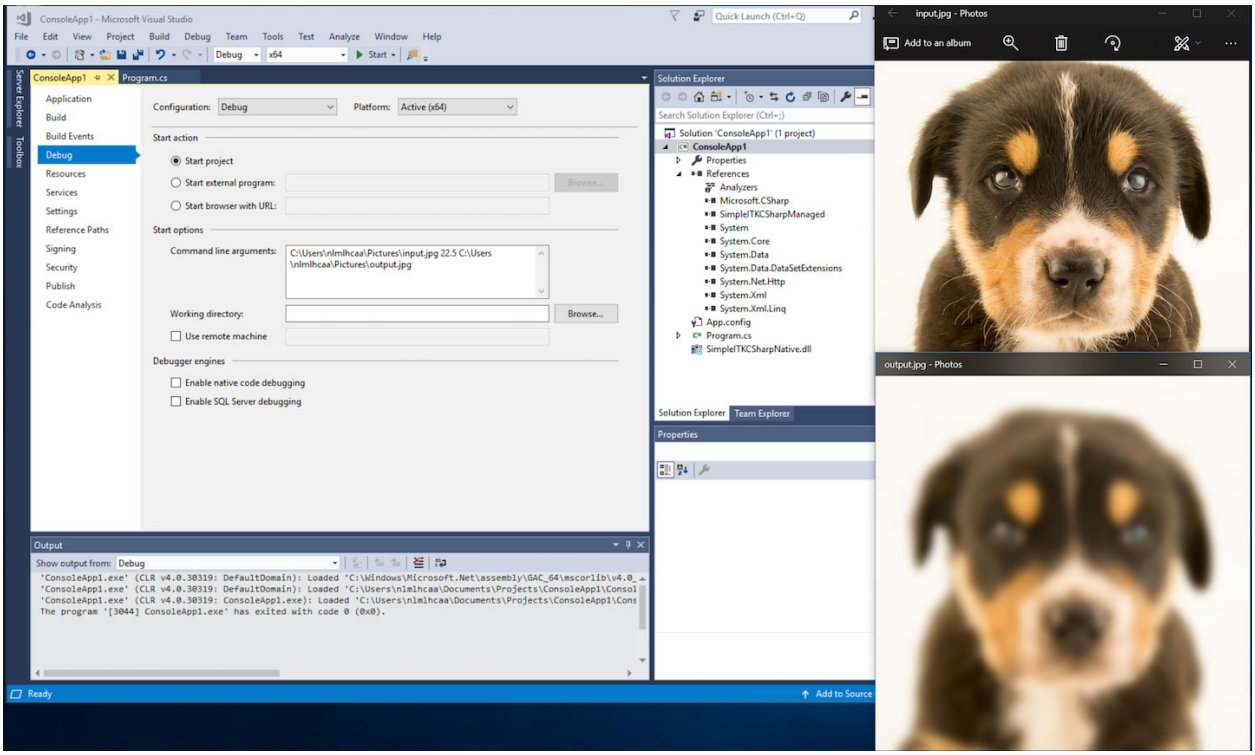
8.2.6 Building an Example

Now that we have configured the project, let up copy a basic SimpleITK example to compile and run. The *Simple-Gaussian in C#* is a simple one to test our configuration. This can just be copied and pasted into the code editor.


Reference Manager - ConsoleApp1
Assemblies
Projects
Shared Projects
COM
Browse
Recent
Search (Ctrl+E)
Name
Path
SimpleITKCSharpManaged.dll
C:\Users\nlmlhcaa\Do...
Name:
SimpleITKCSharpManaged.dll
Created by:
Insight Software Consortium
File Version:
1.1.0
Browse...
OK
Cancel


Properties
SimpleITKCSharpNative.dll File Properties
Advanced
Build Action
Content
Copy to Output Directory
Copy always
Custom Tool
Custom Tool Namespace
Misc
File Name
SimpleITKCSharpNative.dll
Full Path
C:\Users\nlmlhcaa\Documents\S
Copy to Output Directory
Specifies the source file will be copied to the output directory.

Then click *Build -> Build Solution*. Assuming you have followed all of the steps thus far, you should have an executable you can run from the command line or from Visual Studio by adding command line arguments in *Project -> ConsoleApp1 Properties... -> Debug*.



CHAPTER 9

Tutorials and Courses

Whether you are a student at the beginging of a research career in biomedical image analysis or a seasoned veteran, you will likely find attending a SimpleITK tutorial beneficial. These tutorials will shorten the time it takes you to master SimpleITK, with all of the material (code, data, presentations) provided using permisive licenses. After the tutorial, you can readily use the code in your research or go on to use it in your own teaching activities. Additionaly, attending a tutorial allows you to directly interact with the SimpleITK team. This is your oppurtunity to request specific features, and get advice on how to best utilize SimpleITK for your research needs.

While SimpleITK supports eight programming languages, the lingua franca of the biomedical data science community is either Python or R, hence most tutorials utilize Jupyter notebooks in one or both of these languagues.

SimpleITK tutorials and courses have been given in the following venues:

9.1 Upcoming

- SPIE Medical Imaging 2019 Course, San Diego CA, USA: SimpleITK Jupyter Notebooks: Biomedical Image Analysis in Python [git repository].

9.2 Past

- IEEE International Symposium on Biomedical Imaging (ISBI)2018 Tutorial, Washington DC, USA: Biomedical Image Analysis in Python and R using SimpleITK Jupyter Notebooks [git repository].
- SPIE Medical Imaging 2018 Course, Houston TX, USA: SimpleITK Jupyter Notebooks: Biomedical Image Analysis in Python [git repository].

- The International Symposium on Biomedical Imaging (ISBI) 2016, Prague, Czech republic: SimpleITK: An Interactive, Python-Based Introduction to SimpleITK with the Insight Segmentation and Registration Toolkit (ITK).
- SPIE Medical Imaging 2016, San Diego, USA: ITK in Biomedical Research and Commercial Applications [git repository , additional presentations].
- Medical Image Computing and Computer Assisted Intervention (MICCAI) 2015, Munich, Germany: a Python based tutorial on the use of the ITKv4 registration framework via SimpleITK [git repository].
- ImageJ User & Developer Conference 2015, Madison, WI, USA: an introductory tutorial in Python [git repository].
- Medical Image Computing and Computer Assisted Intervention (MICCAI) 2011, Toronto, Canada: a general tutorial on SimpleITK [pdf of presentation, git repository].

CHAPTER 10

Frequently Asked Questions

This page hosts frequently asked questions about SimpleITK, and their answers.

10.1 Installation

10.1.1 I am using the binary distribution of SimpleITK for Anaconda, why do I get an error about libpng?

> ImportError: dlopen(./_SimpleITK.so, 2): Library not loaded: @rpath/libpng15.15.dylib Referenced from: . . . /lib/python2.7/site-packages/SimpleITK/_SimpleITK.so Reason: image not found

This type of error can occour if a library SimpleITK depends on can not be found. It may be that the version of the dependent library has changed in the Python environment and is no longer compatible. One solution is to create a *environment.yml* file with all the packages required for your project, then create a new environment:

```
conda env create -f environment.yml
```

10.2 How to Use

10.2.1 What filters are currently available in SimpleITK?

There are nearly **300 ITK image filters** wrapped in SimpleITK. We have a *list of filters* accompanied by a brief description. Additionally the Doxygen can be examined to determine the availability of a filter.

10.2.2 What image file formats can SimpleITK read?

See *here*.

10.2.3 How do I read a RAW image into SimpleITK?

In general raw image files are missing information. They do not contain the necessary header information to describe the basic size and type for the data, so this format is intrinsically deficient. The RawImageIO class is not available in SimpleITK so there is no direct way to programmatically hard code this header information. The suggested way is to create a Meta image header file (*.mhd) which references the raw data file and describes the size and type of the data. The documentation on how to write a Meta image header can be found here.

The following is a sample Meta image header file, perhaps of name sample.mhd:

```
ObjectType = Image
NDims = 3
DimSize = 256 256 64
ElementType = MET_USHORT
ElementDataFile = image.raw    (this tag must be last in a MetaImageHeader)
```

10.2.4 Why isn't ImageJ found by the Show function (RuntimeError: Exception thrown...)?

The SimpleITK `Show` function expects the ImageJ program to be installed in specific locations. The recommended installation locations are:

- On Windows: in your user directory (e.g. C:\Users\your_user_name\Fiji.app).
- On Linux: in ~/bin .
- On Mac: in /Applications .

To see the locations where the function is searching set its debugOn flag.

In Python:

```
sitk.Show(image, debugOn=True)
```

In R:

```
Show(image, "file_name", TRUE)
```

You can also indicate where a viewer (not necessarily ImageJ) is found by setting the path to the viewer in an environment variable SITK_SHOW_COMMAND.

10.2.5 Can I use another image file viewer beside ImageJ?

By default when the Show function is called, SimpleITK writes out a temporary image in Nifti format then launches ImageJ. The user can override the file format of the temporary file and/or the application used to handle that file.

The temporary file format can be specified via the **SITK_SHOW_EXTENSION** environment variable. For example, if the user wanted to export a PNG file, on Linux it might look like this:

```
SITK_SHOW_EXTENSION=".png"
export SITK_SHOW_EXTENSION
```

Use of an extension unsupported by ITK results in an error message. For the supported image formats, here is the ITK Image IO Filters.

The default display application for all image types is ImageJ. To override ImageJ with some other application, use the **SITK_SHOW_COMMAND** environment variable. For instance, on Unix systems, using GNOME's image viewer eog would be:

```
SITK_SHOW_EXTENSION=".png"
export SITK_SHOW_EXTENSION
SITK_SHOW_COMMAND="eog"
export SITK_SHOW_COMMAND
```

To override the default display applications for only color or 3d images, there are the **SITK_SHOW_COLOR_COMMAND** and **SITK_SHOW_3D_COMMAND** environment variables.

More details on the Show function, including use of the "%a" and "%f" tokens, is at the Show function Doxygen page.

10.2.6 How can I use 3D Slicer to view my images?

3D Slicer is a very powerful and popular application for visualization and medical image computing. The SITK_SHOW_COMMAND environment variable may be used to display images in Slicer instead of SimpleITK's default viewer, ImageJ. The following are examples of what settings for SITK_SHOW_COMMAND might look like for Mac OS X, Linux and Windows to use Slicer.

Mac OS X

```
export SITK_SHOW_COMMAND=/Applications/Slicer.app/Contents/MacOS/Slicer
```

Linux

```
export SITK_SHOW_COMMAND=Slicer
```

Windows

```
set SITK_SHOW_COMMAND=:"c:\Program Files\Slicer 4.2.2-1\Slicer"
```

The value of SITK_SHOW_COMMAND should be modified to point to wherever Slicer is installed. If you only want to use Slicer for volumetric 3D images, use the SITK_SHOW_3D_COMMAND environment variable instead of SITK_SHOW_COMMAND.

10.2.7 How can I use a newer Java with ImageJ on Mac OS X?

By default on Mac OS X, the ImageJ application expects Java 6, which is old and unsupported. The latest supported version of Java (currently version 8u25) can be downloaded from Oracle's Java Development kit page. The following bash commands will set up the SITK_SHOW_COMMAND and SITK_SHOW_COLOR_COMMAND to invoke ImageJ's jar file using the Java compiler.

```
ij="/Applications/ImageJ/"
ijcmd="java -Dplugins.dir=$ij/plugins -jar $ij/ImageJ.app/Contents/Resources/Java/ij.
↪jar"
export SITK_SHOW_COMMAND="$ijcmd -eval 'open( \"%f\" );'"
export SITK_SHOW_COLOR_COMMAND="$ijcmd -eval 'open( \"%f\" ); run(\"Make Composite\",
↪ \"display=Composite\");'"
```

The first lines set a variable pointing to the standard location for the ImageJ directory. If ImageJ is installed somewhere else, the line should be modified. The second line provides the command to launch ImageJ using the Java compiler. It includes flags that point to ImageJ's plugiin directory and ImageJ's ij.jar file.

The SITK_SHOW_COMMAND tells SimpleITK.Show() to launch Java with ij.jar and then execute the open macro with an image file. The SITK_SHOW_COLOR_COMMAND does these same things and then executes the ImageJ "Make Composite" command to treat a multichannel image as a composite color image.

10.3 Wrapping

10.3.1 Python

Why should I use a virtual environment?

Before you install SimpleITK we highly recommend that you create a virtual environment into which you install the package. Note that different Python versions and distributions have different programs for creating and managing virtual environments.

The use of a virtual environment allows you to elegantly deal with package compatability issues, to quote The Hitchhiker's Guide to Python!:

> A Virtual Environment is a tool to keep the dependencies required by different projects in separate places, by creating virtual Python environments for them. It solves the "Project X depends on version 1.x but, Project Y needs 4.x" dilemma, and keeps your global site-packages directory clean and manageable.

Programs for creating virtual environments include virtualenv and pyvenv for generic Python distributions, conda for the anaconda distribution, and canopy_cli for the canopy distribution.

Are the Python Wheels compatible with Enthought Canopy Distribution?

The *Generic Python Wheels* frequently seem to work with the Enthought Canopy Python distribution. However, it is recommended that you compile SimpleITK explicitly against this Python distribution to ensure compatibility.

10.3.2 Tcl

10.3.3 Java

10.3.4 C#

10.3.5 R

10.4 Compilation

10.4.1 Is my compiler supported?

SimpleITK uses advanced C++ meta-programming to instantiate ITK's Images and Filters. Addtionally, we use some headers which are included in the C99 and C++ TR1 extension. Therefore SimpleITK places additional requirements on the compiler beyond what is required for ITK. In principle we require C++x03 with C99's "stdint.h" and TR1's "functional". If your compiler has those features it is likely able to be supported.

The additional requirement for a supported compiler is that it is on the nightly dashboard. With this regard, the list of supported compilers is on the SimpleITK SimpleITK dashboard. We welcome user contributions to the nightly dashboard to expand the list of supported compilers.

Noted Problems

- Microsoft compilers before Visual Studio 14 (2015) have had memory limitation issues.

10.4.2 Why am I getting a compilation error on OSX?

With SimpleITK <=0.7 the following error occurred during compilation on Apple OSX 10.9 Mavericks with **clang 5.0**:

```
SimpleITK/Code/Common/include/sitkMemberFunctionFactoryBase.h:106:16:  error: no␣
↪member named 'tr1' in namespace 'std'
typedef std::tr1::function< MemberFunctionResultType ( ) > FunctionObjectType;
~~~~~^
```

With Xcode 5.0, Apple's distributed version of clang (5.0) changed which implementation of the C++ Standard Library it uses by default. Previous versions of clang (4.2 and earlier) used GNU's libstdc++ , while clang 5.0 now uses LLVM's libc++. SimpleITK 0.7 and earlier require certain features from C++ tr1 which are not implemented in LLVM's libc++ but are available in GNU's libstdc++.

To build SimpleITK <=0.7 with clang 5.0, you can configure the compiler to use GNU's stdlibc++. This change must be done at the initial configuration:

```
cmake "-DCMAKE_CXX_FLAGS:STRING=-stdlib=libstdc++" ../SimpleITK/SuperBuild
```

NOTE: If you already have a build directory which has been partially configured the contents must be deleted. The above line needs to be done for an initial configuration in an empty build directory. NOTE: This work around does not work when with the CMake "Xcode" generator. It is recommended to just use the default "Unix Makefiles" generator, to build SimpleITK, and get using SimpleITK, not building it.

The following is a **compatibility table for clang 5.0**. It shows that the default of libc++ does not work with SimpleITK, while the other options do. The choice of which standard library to use and which C++ language standard to use are independent.

Clang 5.0 compatibility	-stdlib=libc++	-stdlib=libstdc++
(c++03)	FAIL	OK
-std=c++11	OK (>=0.8)	OK

For SimpleITK >=0.8, support for the tr1 features migrated to C++11 has been improved with better feature detection, and the necessary flags are now automatically added. LLVM's libc++ will now work if compiling with the C++11 standard by adding the flag "-std=c++11" in the initial configuration.

To further complicate dependencies and interactions, some downloadable languages such as Java, or R, may be compiled against GNU's libstdc++. This may cause a conflict in the types used in the interface resulting in compilation errors while wrapping the language.

10.4.3 Why does the Superbuild fail compiling PCRE on Mac OS X?

If the Xcode command line tools are not properly set up on OS X, PCRE could fail to build in the Superbuild process with messages such as:

```
checking whether we are cross compiling... configure: error: in `/your/build/path/
↪SimpleITK/PCRE-prefix/src/PCRE-build':
configure: error: cannot run C compiled programs.
If you meant to cross compile, use `--host'.
```

(continues on next page)

(continued from previous page)

```
See `config.log' for more details
[10/13] Performing build step for 'PCRE'
```

To install the command line developer tools enter the following:

```
xcode-select --install
```

To reset the default command line tools path:

```
xcode-select --reset
```

10.4.4 Do I need to download an option package for TR1 support?

Visual Studio 2008 requires an additional download for TR1 support. This support is best provided with the Service Pack 1. There is a separate TR1 feature pack which can be downloaded, but it is no longer recommended since Service Pack 1 includes TR1 and numerous bug and performance improvements.

10.4.5 What Configurations on Windows are Supported For Building?

We recommend to use Microsoft Visual Studio 14 (2015) or newer to compile SimpleITK.

10.4.6 Where is the Test Data?

The testing data is not stored in the SimpleITK repository or as part of the source code. It is mirrored on several data repositories on the web.

If you have obtained the source code from the git repository, it should be downloaded as part of the build process via the CMake ExternalData module.

You can download a tar-ball of the "SimpleITKData" for a release from the GitHub Assets, which contains the external data. It should populate the .ExternalData subdirectory of the SimpleITK source code directory when extracted.

10.4.7 Why is CMake unable to download ExternalData?

When compiling SimpleITK you may get and error like the following:

```
Object MD5=2e115fe26e435e33b0d5c022e4490567 not found at:
 https://placid.nlm.nih.gov/api/rest?method=midas.bitstream.download&
↪checksum=2e115fe26e435e33b0d5c022e4490567&algorithm=MD5 ("Unsupported protocol")
 https://simpleitk.github.io/SimpleITKExternalData/MD5/
↪2e115fe26e435e33b0d5c022e4490567 ("Unsupported protocol")
 https://midas3.kitware.com/midas/api/rest?method=midas.bitstream.download&
↪checksum=2e115fe26e435e33b0d5c022e4490567&algorithm=MD5 ("Unsupported protocol")
 https://insightsoftwareconsortium.github.io/ITKTestingData/MD5/
↪2e115fe26e435e33b0d5c022e4490567 ("Unsupported protocol")
 https://itk.org/files/ExternalData/MD5/2e115fe26e435e33b0d5c022e4490567 (
↪"Unsupported protocol")
```

This indicates that CMake was not compiles with SSL support. The "Unsupported protocol" message indicate that CMake can not communicate via "https".

The solution is to use a compiled version of CMake which supports SSL. If you compile CMake yourself, simply reconfigure CMake with the "CMAKE_USE_OPENSSL" option enabled.

CHAPTER 11

Examples

11.1 Hello World

11.1.1 Overview

A "Hello World" example for SimpleITK. The example constructs a 128x128 greyscale image, draws a smiley face made of Gaussian blobs, and calls the Show function to display with image with ImageJ.

11.1.2 Code

C#

```
using System;
using itk.simple;

namespace itk.simple.examples {
  class HelloWorld {

    static void Main(string[] args) {

      try {

        // Create an image
        PixelIDValueEnum pixelType = PixelIDValueEnum.sitkUInt8;
        VectorUInt32 imageSize = new VectorUInt32( new uint[] { 128, 128 } );
        Image image = new Image( imageSize, pixelType );

        // Create a face image
        VectorDouble faceSize = new VectorDouble( new double[] { 64, 64 } );
        VectorDouble faceCenter = new VectorDouble( new double[] { 64, 64 } );
        Image face = SimpleITK.GaussianSource( pixelType, imageSize, faceSize,␣
↪faceCenter );
```

(continues on next page)

(continued from previous page)

```
        // Create eye images
        VectorDouble eyeSize = new VectorDouble( new double[] { 5, 5 } );
        VectorDouble eye1Center = new VectorDouble( new double[] { 48, 48 } );
        VectorDouble eye2Center = new VectorDouble( new double[] { 80, 48 } );
        Image eye1 = SimpleITK.GaussianSource( pixelType, imageSize, eyeSize,␣
↪eye1Center, 150 );
        Image eye2 = SimpleITK.GaussianSource( pixelType, imageSize, eyeSize,␣
↪eye2Center, 150 );

        // Apply the eyes to the face
        face = SimpleITK.Subtract( face, eye1 );
        face = SimpleITK.Subtract( face, eye2 );
        face = SimpleITK.BinaryThreshold( face, 200, 255, 255 );


        // Create the mouth
        VectorDouble mouthRadii = new VectorDouble( new double[] { 30, 20 } );
        VectorDouble mouthCenter = new VectorDouble( new double[] { 64, 76 } );
        Image mouth = SimpleITK.GaussianSource( pixelType, imageSize, mouthRadii,␣
↪mouthCenter );
        mouth = SimpleITK.BinaryThreshold( mouth, 200, 255, 255 );
        mouth = SimpleITK.Subtract( 255, mouth );

        // Paste the mouth onto the face
        VectorUInt32 mouthSize = new VectorUInt32( new uint[] { 64, 18 } );
        VectorInt32 mouthLoc = new VectorInt32(  new int[] { 32, 76 } );
        face = SimpleITK.Paste( face, mouth, mouthSize, mouthLoc, mouthLoc );

        // Apply the face to the original image
        image = SimpleITK.Add( image, face );

        SimpleITK.Show( image, "Hello World: CSharp", true );

      } catch (Exception ex) {
        Console.WriteLine(ex);
      }
    }

  }
}
```

C++

```
// This one header will include all SimpleITK filters and external
// objects.
#include <SimpleITK.h>
#include <sitkImageOperators.h>
#include <iostream>
#include <stdlib.h>


// create convenient namespace alias
namespace sitk = itk::simple;

int main ( int argc, char* argv[] ) {
```

(continues on next page)

(continued from previous page)

```
  sitk::PixelIDValueEnum pixelType = sitk::sitkUInt8;
  std::vector<unsigned int> imageSize ( 2, 128 );

  // Create an image
  sitk::Image image( imageSize, pixelType );

  // Create a face image
  std::vector<double> faceSize ( 2, 64.0 );
  std::vector<double> faceCenter ( 2, 64.0 );;
  sitk::Image face = sitk::GaussianSource( pixelType, imageSize, faceSize, faceCenter␣
↪);

  // Create eye images
  std::vector<double> eyeSize ( 2, 5.0 );
  std::vector<double> eye1Center ( 2, 48.0 );
  std::vector<double> eye2Center;
  eye2Center.push_back( 80.0 );
  eye2Center.push_back( 48.0 );
  sitk::Image eye1 = sitk::GaussianSource( pixelType, imageSize, eyeSize, eye1Center,␣
↪150 );
  sitk::Image eye2 = sitk::GaussianSource( pixelType, imageSize, eyeSize, eye2Center,␣
↪150 );

  // Apply the eyes to the face
  face = face - eye1 - eye2;
  face = sitk::BinaryThreshold( face, 200, 255, 255 );

  // Create the mouth
  std::vector<double> mouthRadii;
  mouthRadii.push_back( 30.0 );
  mouthRadii.push_back( 20.0 );
  std::vector<double> mouthCenter;
  mouthCenter.push_back( 64.0 );
  mouthCenter.push_back( 76.0 );
  sitk::Image mouth = 255 - sitk::BinaryThreshold(
                              sitk::GaussianSource(pixelType, imageSize, mouthRadii,␣
↪mouthCenter),
                              200, 255, 255 );

  // Paste the mouth onto the face
  std::vector<unsigned int> mouthSize;
  mouthSize.push_back( 64 );
  mouthSize.push_back( 18 );
  std::vector<int> mouthLoc;
  mouthLoc.push_back( 32 );
  mouthLoc.push_back( 76 );
  face = sitk::Paste( face, mouth, mouthSize, mouthLoc, mouthLoc );

  // Apply the face to the original image
  image = image + face;

  // Display the results
  sitk::Show( image, "Hello World: C++", true );

}
```

Java

```
import org.itk.simple.*;

class HelloWorld {

  public static void main(String argv[]) {

    // Create an image
    PixelIDValueEnum pixelType = PixelIDValueEnum.sitkUInt8;
    VectorUInt32 imageSize = new VectorUInt32( 2 );
    imageSize.set( 0, 128 );
    imageSize.set( 1, 128 );
    Image image = new Image( imageSize, pixelType );

    // Create a face image
    VectorDouble faceSize = new VectorDouble( 2 );
    faceSize.set( 0, 64 );
    faceSize.set( 1, 64 );
    VectorDouble faceCenter = new VectorDouble( 2 );
    faceCenter.set( 0, 64 );
    faceCenter.set( 1, 64 );
    Image face = SimpleITK.gaussianSource( pixelType, imageSize, faceSize, faceCenter␣
↪);

    // Create eye images
    VectorDouble eyeSize = new VectorDouble( 2 );
    eyeSize.set( 0, 5 );
    eyeSize.set( 1, 5 );
    VectorDouble eye1Center = new VectorDouble( 2 );
    eye1Center.set( 0, 48 );
    eye1Center.set( 1, 48 );
    VectorDouble eye2Center = new VectorDouble( 2 );
    eye2Center.set( 0, 80 );
    eye2Center.set( 1, 48 );
    Image eye1 = SimpleITK.gaussianSource( pixelType, imageSize, eyeSize, eye1Center,␣
↪150 );
    Image eye2 = SimpleITK.gaussianSource( pixelType, imageSize, eyeSize, eye2Center,␣
↪150 );

    // Apply the eyes to the face
    face = SimpleITK.subtract( face, eye1 );
    face = SimpleITK.subtract( face, eye2 );
    face = SimpleITK.binaryThreshold( face, 200., 255., (short) 255 );

    // Create the mouth
    VectorDouble mouthRadii = new VectorDouble( 2 );
    mouthRadii.set( 0, 30 );
    mouthRadii.set( 1, 20 );
    VectorDouble mouthCenter = new VectorDouble( 2 );
    mouthCenter.set( 0, 64 );
    mouthCenter.set( 1, 76 );
    Image mouth = SimpleITK.gaussianSource( pixelType, imageSize, mouthRadii,␣
↪mouthCenter );
    mouth = SimpleITK.binaryThreshold( mouth, 200, 255, (short) 255 );
    mouth = SimpleITK.subtract( 255, mouth );

    // Paste the mouth onto the face
    VectorUInt32 mouthSize = new VectorUInt32( 2 );
```

(continues on next page)

(continued from previous page)

```
    mouthSize.set( 0, 64 );
    mouthSize.set( 1, 18 );
    VectorInt32 mouthLoc = new VectorInt32( 2 );
    mouthLoc.set( 0, 32 );
    mouthLoc.set( 1, 76 );
    face = SimpleITK.paste( face, mouth, mouthSize, mouthLoc, mouthLoc );

    // Apply the face to the original image
    image = SimpleITK.add( image, face );

    // Display the results
    SimpleITK.show( image, "Hello World: Java", true );
  }

}
```

Lua

```
require "SimpleITK"

local sitk = SimpleITK

-- Create an image
pixelType = sitk.sitkUInt8
imageSize = sitk.VectorUInt32()
imageSize:push_back( 128 )
imageSize:push_back( 128 )
image = sitk.Image( imageSize, sitk.sitkUInt8 )


-- Create a face image
faceSize = sitk.VectorDouble()
faceSize:push_back( 64 )
faceSize:push_back( 64 )
faceCenter = sitk.VectorDouble()
faceCenter:push_back( 64 )
faceCenter:push_back( 64 )
face = sitk.GaussianSource( pixelType, imageSize, faceSize, faceCenter )

-- Create eye images
eyeSize = sitk.VectorDouble()
eyeSize:push_back( 5 )
eyeSize:push_back( 5 )
eye1Center = sitk.VectorDouble()
eye1Center:push_back( 48 )
eye1Center:push_back( 48 )
eye2Center = sitk.VectorDouble()
eye2Center:push_back( 80 )
eye2Center:push_back( 48 )
eye1 = sitk.GaussianSource( pixelType, imageSize, eyeSize, eye1Center, 150 )
eye2 = sitk.GaussianSource( pixelType, imageSize, eyeSize, eye2Center, 150 )

-- Apply the eyes to the face
face = sitk.Subtract( face, eye1 )
face = sitk.Subtract( face, eye2 )
face = sitk.BinaryThreshold( face, 200, 255, 255 )
```

(continues on next page)

(continued from previous page)

```
-- Create the mouth
mouthRadii = sitk.VectorDouble()
mouthRadii:push_back( 30.0 )
mouthRadii:push_back( 20.0 )
mouthCenter = sitk.VectorDouble()
mouthCenter:push_back( 64.0 )
mouthCenter:push_back( 76.0 )
mouth = sitk.GaussianSource( pixelType, imageSize, mouthRadii, mouthCenter )
mouth = sitk.BinaryThreshold( mouth, 200, 255, 255 )
mouth = sitk.Subtract( 255, mouth )

-- Paste the mouth onto the face
mouthSize = sitk.VectorUInt32()
mouthSize:push_back( 64 )
mouthSize:push_back( 18 )
mouthLoc = sitk.VectorInt32()
mouthLoc:push_back( 32 )
mouthLoc:push_back( 76 )
face = sitk.Paste( face, mouth, mouthSize, mouthLoc, mouthLoc );

-- Apply the face to the original image
image = sitk.Add( image, face )

-- Display the results
sitk.Show( image, "Hello World: Lua", true )
```

Python

```
#!/usr/bin/env python

import SimpleITK as sitk

# Create an image
pixelType = sitk.sitkUInt8
imageSize = [128, 128]
image     = sitk.Image( imageSize, pixelType )


# Create a face image
faceSize   = [64, 64]
faceCenter = [64, 64]
face       = sitk.GaussianSource(pixelType, imageSize, faceSize, faceCenter)

# Create eye images
eyeSize    = [5, 5]
eye1Center = [48, 48]
eye2Center = [80, 48]
eye1       = sitk.GaussianSource(pixelType, imageSize, eyeSize, eye1Center, 150)
eye2       = sitk.GaussianSource(pixelType, imageSize, eyeSize, eye2Center, 150)

# Apply the eyes to the face
face = face - eye1 - eye2
face = sitk.BinaryThreshold(face, 200, 255, 255)

# Create the mouth
mouthRadii  = [30, 20]
mouthCenter = [64, 76]
```

(continues on next page)

(continued from previous page)

```
mouth       = 255 - sitk.BinaryThreshold( sitk.GaussianSource(pixelType, imageSize,␣
↪mouthRadii, mouthCenter),
                                          200, 255, 255 )
# Paste the mouth into the face
mouthSize = [64, 18]
mouthLoc  = [32, 76]
face      = sitk.Paste(face, mouth, mouthSize, mouthLoc, mouthLoc)

# Apply the face to the original image
image = image+face

# Display the results
sitk.Show( image, title="Hello World: Python", debugOn=True )
```

R

```
library(SimpleITK)

# Create an image
pixelType <- 'sitkUInt8'
imageSize <- c( 128, 128 )
image     <- Image( imageSize, pixelType )

# Create a face image
faceSize   <- c( 64, 64)
faceCenter <- c( 64, 64)
face       <- GaussianSource( pixelType, imageSize, faceSize, faceCenter )

# Create eye images
eyeSize    <- c( 5, 5 )
eye2Center <- c( 48, 48 )
eye1Center <- c( 80, 48 )
eye1       <- GaussianSource( pixelType, imageSize, eyeSize, eye1Center, 150 )
eye2       <- GaussianSource( pixelType, imageSize, eyeSize, eye2Center, 150 )

# Apply the eyes to the face
face <- face - eye1 - eye2
face <- BinaryThreshold( face, 200, 255, 255 )

# Create the mouth
mouthRadii  <- c( 30, 20 )
mouthCenter <- c( 64, 76 )
mouth       <- 255 - BinaryThreshold( GaussianSource( pixelType, imageSize,␣
↪mouthRadii, mouthCenter ),
                                      200, 255, 255 )
# Paste mouth onto the face
mouthSize <- c( 64, 18 )
mouthLoc  <- c( 32, 76 )
face = Paste( face, mouth, mouthSize, mouthLoc, mouthLoc )

# Apply the face to the original image
image <- image + face

# Display the results
Show(image, "Hello World: R", debugOn=TRUE)
```

Ruby

```
require 'simpleitk'


# Create an image
pixelType = Simpleitk::SitkUInt8
imageSize = Simpleitk::VectorUInt32.new
imageSize << 128
imageSize << 128
image = Simpleitk::Image.new( imageSize, pixelType )

# Create a face image
faceSize = Simpleitk::VectorDouble.new
faceSize << 64
faceSize << 64
faceCenter = Simpleitk::VectorDouble.new
faceCenter << 64
faceCenter << 64
face = Simpleitk::gaussian_source( pixelType, imageSize, faceSize, faceCenter )

# Create eye images
eyeSize = Simpleitk::VectorDouble.new
eyeSize << 5
eyeSize << 5
eye1Center = Simpleitk::VectorDouble.new
eye1Center << 48
eye1Center << 48
eye2Center = Simpleitk::VectorDouble.new
eye2Center << 80
eye2Center << 48
eye1 = Simpleitk::gaussian_source( pixelType, imageSize, eyeSize, eye1Center, 150 )
eye2 = Simpleitk::gaussian_source( pixelType, imageSize, eyeSize, eye2Center, 150 )

# Apply the eyes to the face
face = Simpleitk.subtract( face, eye1 )
face = Simpleitk.subtract( face, eye2 )
face = Simpleitk.binary_threshold( face, 200, 255, 255 );

# Create the mouth
mouthRadii = Simpleitk::VectorDouble.new
mouthRadii << 30
mouthRadii << 20
mouthCenter = Simpleitk::VectorDouble.new
mouthCenter << 64
mouthCenter << 76
mouth = Simpleitk::gaussian_source( pixelType, imageSize, mouthRadii, mouthCenter )
mouth = Simpleitk::binary_threshold( mouth, 200, 255, 255 )
mouth = Simpleitk::subtract( 255, mouth )

# Paste the mouth onto the face
mouthSize = Simpleitk::VectorUInt32.new
mouthSize << 64
mouthSize << 18
mouthLoc = Simpleitk::VectorInt32.new
mouthLoc << 32
mouthLoc << 76
face = Simpleitk::paste( face, mouth, mouthSize, mouthLoc, mouthLoc )
```

(continues on next page)

(continued from previous page)

```
# Apply the face to the original image
image = Simpleitk.add( image, face )

Simpleitk.show( image, "Hello World: Ruby", true )
```

Tcl

```
# Create an image
set pixelType $sitkUInt16
set imageSize { 128 128 }
Image img   $imageSize $pixelType

# Create a face
set faceSize   { 64 64 }
set faceCenter { 64 64 }
set face       [ GaussianSource $pixelType $imageSize $faceSize $faceCenter ]

# Create eye images
set eyeSize    { 5 5 }
set eye1Center { 48 48 }
set eye2Center { 80 48 }
set eye1       [ GaussianSource $pixelType $imageSize $eyeSize $eye1Center 150 ]
set eye2       [ GaussianSource $pixelType $imageSize $eyeSize $eye2Center 150 ]

# Apply the eyes to the face
set face [ Subtract $face $eye1 ]
set face [ Subtract $face $eye2 ]
set face [ BinaryThreshold $face 200 255 255 ]

# Create the mouth
set mouthRadii  { 30 20 }
set mouthCenter { 64 76 }
set mouth [ GaussianSource $pixelType $imageSize $mouthRadii $mouthCenter ]
set mouth [ Subtract 255 [ BinaryThreshold $mouth 200 255 255 ] ]

# Paste the mouth onto the face
set mouthSize { 64 18 }
set mouthLoc  { 32 76 }
set face [ Paste $face $mouth $mouthSize $mouthLoc $mouthLoc ]

# Apply the face to the original image
set img $face

# Display the results
Show $img "Hello World: TCL" 1
```

11.2 CSharp Integration

11.2.1 Overview

11.2.2 Code

```
using System;
using System.Runtime.InteropServices;

using itk.simple;
using PixelId = itk.simple.PixelIDValueEnum;

namespace itk.simple.examples {
  public class Program {
    static void Main(string[] args) {

      if (args.Length < 1) {
        Console.WriteLine("Usage: SimpleGaussian <input>");
        return;
      }

      // Read input image
      itk.simple.Image input = SimpleITK.ReadImage(args[0]);

      // Cast to we know the the pixel type
      input = SimpleITK.Cast(input, PixelId.sitkFloat32);

      // calculate the number of pixels
      VectorUInt32 size = input.GetSize();
      int len = 1;
      for (int dim = 0; dim < input.GetDimension(); dim++) {
        len *= (int)size[dim];
      }
      IntPtr buffer = input.GetBufferAsFloat();

      // There are two ways to access the buffer:

      // (1) Access the underlying buffer as a pointer in an "unsafe" block
      // (note that in C# "unsafe" simply means that the compiler can not
      // perform full type checking), and requires the -unsafe compiler flag
      // unsafe {
      //   float* bufferPtr = (float*)buffer.ToPointer();

      //   // Now the byte pointer can be accessed as per Brad's email
      //   // (of course this example is only a 2d single channel image):
      //   // This is a 1-D array but can be access as a 3-D. Given an
      //   // image of size [xS,yS,zS], you can access the image at
      //   // index [x,y,z] as you wish by image[x+y*xS+z*xS*yS],
      //   // so x is the fastest axis and z is the slowest.
      //   for (int j = 0; j < size[1]; j++) {
      //     for (int i = 0; i < size[0]; i++) {
      //       float pixel = bufferPtr[i + j*size[1]];
      //       // Do something with pixel here
      //     }
      //   }
      // }
```

(continues on next page)

(continued from previous page)

```
      // (2) Copy the buffer to a "safe" array (i.e. a fully typed array)
      // (note that this means memory is duplicated)
      float[] bufferAsArray = new float[len]; // Allocates new memory the size of␣
↪input
      Marshal.Copy(buffer, bufferAsArray, 0, len);
      double total = 0.0;
      for (int j = 0; j < size[1]; j++) {
        for (int i = 0; i < size[0]; i++) {
          float pixel = bufferAsArray[i + j*size[1]];
          total += pixel;
        }
      }
      Console.WriteLine("Pixel value total: {0}", total);

    }
  }
}
```

11.3 DemonsRegistration1

11.3.1 Overview

This example illustrates how to use the classic Demons registration algorithm. The user supplied parameters for the algorithm are the number of iterations and the standard deviations for the Gaussian smoothing of the total displacement field. Additional methods which control regularization, total field smoothing for elastic model or update field smoothing for viscous model are available.

The underlying assumption of the demons framework is that the intensities of homologous points are equal. The example uses histogram matching to make the two images similar prior to registration. This is relevant for registration of MR images where the assumption is not valid. For other imaging modalities where the assumption is valid, such as CT, this step is not necessary. Additionally, the command design pattern is used to monitor registration progress. The resulting deformation field is written to file.

See also: *DemonsRegistration2*.

11.3.2 Code

C++

```
// This example is based on ITK's DeformableRegistration2.cxx example


#include <SimpleITK.h>
#include <iostream>
#include <stdlib.h>
#include <iomanip>

namespace sitk = itk::simple;


class IterationUpdate
```

(continues on next page)

(continued from previous page)

```
  : public sitk::Command
{
public:
  IterationUpdate( const sitk::DemonsRegistrationFilter &m)
    : m_Filter(m)
    {}

  virtual void Execute( )
    {
      // use sitk's output operator for std::vector etc..
      using sitk::operator<<;

      // stash the stream state
      std::ios  state(NULL);
      state.copyfmt(std::cout);
      std::cout << std::fixed << std::setfill(' ') << std::setprecision( 5 );
      std::cout << std::setw(3) << m_Filter.GetElapsedIterations();
      std::cout << " = " << std::setw(10) << m_Filter.GetMetric();
      std::cout << std::endl;

      std::cout.copyfmt(state);
    }

private:
  const sitk::DemonsRegistrationFilter &m_Filter;

};



int main(int argc, char *argv[])
{

  if ( argc < 4 )
    {
    std::cerr << "Usage: " << argv[0] << " <fixedImageFilter> <movingImageFile>
↪<outputTransformFile>" << std::endl;
    return 1;
    }

  sitk::Image fixed = sitk::ReadImage( argv[1], sitk::sitkFloat32 );

  sitk::Image moving = sitk::ReadImage( argv[2], sitk::sitkFloat32 );

  sitk::HistogramMatchingImageFilter matcher;
  matcher.SetNumberOfHistogramLevels( 1024 );
  matcher.SetNumberOfMatchPoints( 7 );
  matcher.ThresholdAtMeanIntensityOn();
  moving = matcher.Execute(moving, fixed);

  sitk::DemonsRegistrationFilter filter;

  IterationUpdate cmd(filter);
  filter.AddCommand( sitk::sitkIterationEvent, cmd );

  filter.SetNumberOfIterations( 50 );
  filter.SetStandardDeviations( 1.0 );
```

(continues on next page)

(continued from previous page)

```
  sitk::Image displacementField = filter.Execute( fixed, moving );

  std::cout << "-------" << std::endl;
  std::cout << "Number Of Iterations: " << filter.GetElapsedIterations() << std::endl;
  std::cout << " RMS: " << filter.GetRMSChange() << std::endl;

  sitk::DisplacementFieldTransform outTx( displacementField );

  sitk::WriteTransform(outTx, argv[3]);

  return 0;
}
```

Python

```
#!/usr/bin/env python

from __future__ import print_function

import SimpleITK as sitk
import sys
import os


def command_iteration(filter) :
    print("{0:3} = {1:10.5f}".format(filter.GetElapsedIterations(),
                                    filter.GetMetric()))

if len ( sys.argv ) < 4:
    print( "Usage: {0} <fixedImageFilter> <movingImageFile> <outputTransformFile>".
↪format(sys.argv[0]))
    sys.exit ( 1 )


fixed = sitk.ReadImage(sys.argv[1], sitk.sitkFloat32)

moving = sitk.ReadImage(sys.argv[2], sitk.sitkFloat32)


matcher = sitk.HistogramMatchingImageFilter()
matcher.SetNumberOfHistogramLevels(1024)
matcher.SetNumberOfMatchPoints(7)
matcher.ThresholdAtMeanIntensityOn()
moving = matcher.Execute(moving,fixed)

# The basic Demons Registration Filter
# Note there is a whole family of Demons Registration algorithms included in SimpleITK
demons = sitk.DemonsRegistrationFilter()
demons.SetNumberOfIterations( 50 )
# Standard deviation for Gaussian smoothing of displacement field
demons.SetStandardDeviations( 1.0 )

demons.AddCommand( sitk.sitkIterationEvent, lambda: command_iteration(demons) )

displacementField = demons.Execute( fixed, moving )
```

(continues on next page)

(continued from previous page)

```
print("-------")
print("Number Of Iterations: {0}".format(demons.GetElapsedIterations()))
print(" RMS: {0}".format(demons.GetRMSChange()))

outTx = sitk.DisplacementFieldTransform( displacementField )

sitk.WriteTransform(outTx, sys.argv[3])

if ( not "SITK_NOSHOW" in os.environ ):

    resampler = sitk.ResampleImageFilter()
    resampler.SetReferenceImage(fixed);
    resampler.SetInterpolator(sitk.sitkLinear)
    resampler.SetDefaultPixelValue(100)
    resampler.SetTransform(outTx)

    out = resampler.Execute(moving)
    simg1 = sitk.Cast(sitk.RescaleIntensity(fixed), sitk.sitkUInt8)
    simg2 = sitk.Cast(sitk.RescaleIntensity(out), sitk.sitkUInt8)
    # Use the // floor division operator so that the pixel type is
    # the same for all three images which is the expectation for
    # the compose filter.
    cimg = sitk.Compose(simg1, simg2, simg1//2.+simg2//2.)
    sitk.Show( cimg, "DeformableRegistration1 Composition" )
```

R

```
# Run with:
#
# Rscript --vanilla DemonsRegistration1.R fixedImageFilter movingImageFile␣
↪outputTransformFile
#

library(SimpleITK)

commandIteration <- function(filter)
{
    msg <- paste("Iteration number ", filter$GetElapsedIterations(),
                 " = ", filter$GetMetric(), "\n" )
    cat(msg)
}

args <- commandArgs( TRUE )

if (length(args) != 3) {
    stop("3 arguments expected - fixedImageFilter, movingImageFile,␣
↪outputTransformFile")
}

fixed <- ReadImage(args[[1]], 'sitkFloat32')

moving <- ReadImage(args[[2]], 'sitkFloat32')


matcher <- HistogramMatchingImageFilter()
matcher$SetNumberOfHistogramLevels(1024)
```

(continues on next page)

(continued from previous page)

```
matcher$SetNumberOfMatchPoints(7)
matcher$ThresholdAtMeanIntensityOn()
moving <- matcher$Execute(moving, fixed)

# The basic Demons Registration Filter
# Note there is a whole family of Demons Registration algorithms included in SimpleITK
demons <- DemonsRegistrationFilter()
demons$SetNumberOfIterations( 50 )
# Standard deviation for Gaussian smoothing of displacement field
demons$SetStandardDeviations( 1.0 )

demons$AddCommand( 'sitkIterationEvent', function() commandIteration(demons) )

displacementField <- demons$Execute( fixed, moving )

cat("-------\n")
cat("Number Of Iterations: ", demons$GetElapsedIterations(), "\n")
cat("RMS: ", demons$GetRMSChange(), "\n")

outTx <- DisplacementFieldTransform( displacementField )

WriteTransform(outTx, args[[3]])
```

11.4 DemonsRegistration2

11.4.1 Overview

This example illustrates how to use the fast symmetric forces Demons algorithm. As the name implies, unlike the classical algorithm, the forces are symmetric.

The underlying assumption of the demons framework is that the intensities of homologous points are equal. The example uses histogram matching to make the two images similar prior to registration. This is relevant for registration of MR images where the assumption is not valid. For other imaging modalities where the assumption is valid, such as CT, this step is not necessary. Additionally, the command design pattern is used to monitor registration progress. The resulting deformation field is written to file.

See also: *DemonsRegistration1*.

11.4.2 Code

C++

```
// This example is based on ITK's DeformableRegistration2.cxx example


#include <SimpleITK.h>
#include <iostream>
#include <stdlib.h>
#include <iomanip>

namespace sitk = itk::simple;
```

(continues on next page)

(continued from previous page)

```
class IterationUpdate
  : public sitk::Command
{
public:
  IterationUpdate( const sitk::FastSymmetricForcesDemonsRegistrationFilter &m)
    : m_Filter(m)
    {}

  virtual void Execute( )
    {
      // use sitk's output operator for std::vector etc..
      using sitk::operator<<;

      // stash the stream state
      std::ios  state(NULL);
      state.copyfmt(std::cout);
      std::cout << std::fixed << std::setfill(' ') << std::setprecision( 5 );
      std::cout << std::setw(3) << m_Filter.GetElapsedIterations();
      std::cout << " = " << std::setw(10) << m_Filter.GetMetric();
      std::cout << std::endl;

      std::cout.copyfmt(state);
    }

private:
  const sitk::FastSymmetricForcesDemonsRegistrationFilter &m_Filter;

};



int main(int argc, char *argv[])
{

  if ( argc < 4 )
    {
    std::cerr << "Usage: " << argv[0] << " <fixedImageFilter> <movingImageFile>␣
↪[initialTransformFile] <outputTransformFile>" << std::endl;
    return 1;
    }

  sitk::Image fixed = sitk::ReadImage( argv[1] );

  sitk::Image moving = sitk::ReadImage( argv[2] );

  sitk::HistogramMatchingImageFilter matcher;
  if ( fixed.GetPixelID() == sitk::sitkUInt8
       || fixed.GetPixelID() == sitk::sitkInt8 )
    {
    matcher.SetNumberOfHistogramLevels( 128 );
    }
  else
    {
    matcher.SetNumberOfHistogramLevels( 1024 );
    }
  matcher.SetNumberOfMatchPoints( 7 );
```

(continues on next page)

(continued from previous page)

```
  matcher.ThresholdAtMeanIntensityOn();

  moving = matcher.Execute(moving, fixed);

  sitk::FastSymmetricForcesDemonsRegistrationFilter filter;

  IterationUpdate cmd(filter);
  filter.AddCommand( sitk::sitkIterationEvent, cmd );

  filter.SetNumberOfIterations( 200 );
  filter.SetStandardDeviations( 1.0 );

  sitk::Image displacementField;

  if ( argc > 4 )
    {

    sitk::Transform initialTransform = sitk::ReadTransform( argv[3] );
    argv[3] = argv[4];
    --argc;

    sitk::TransformToDisplacementFieldFilter toDisplacementFilter;
    toDisplacementFilter.SetReferenceImage(fixed);
    displacementField = toDisplacementFilter.Execute(initialTransform);
    displacementField = filter.Execute( fixed, moving, displacementField );
    }
  else
    {
    displacementField = filter.Execute( fixed, moving );
    }

  std::cout << "-------" << std::endl;
  std::cout << "Number Of Iterations: " << filter.GetElapsedIterations() << std::endl;
  std::cout << " RMS: " << filter.GetRMSChange() << std::endl;

  sitk::DisplacementFieldTransform outTx( displacementField );

  sitk::WriteTransform(outTx, argv[3]);

  return 0;
}
```

Python

```
#!/usr/bin/env python

from __future__ import print_function

import SimpleITK as sitk
import sys
import os


def command_iteration(filter) :
    print("{0:3} = {1:10.5f}".format(filter.GetElapsedIterations(),
                                     filter.GetMetric()))

```

(continues on next page)

(continued from previous page)

```
if len ( sys.argv ) < 4:
    print( "Usage: {0} <fixedImageFilter> <movingImageFile> [initialTransformFile]
↪<outputTransformFile>".format(sys.argv[0]))
    sys.exit ( 1 )


fixed = sitk.ReadImage(sys.argv[1])

moving = sitk.ReadImage(sys.argv[2])

matcher = sitk.HistogramMatchingImageFilter()
if ( fixed.GetPixelID() in ( sitk.sitkUInt8, sitk.sitkInt8 ) ):
    matcher.SetNumberOfHistogramLevels(128)
else:
    matcher.SetNumberOfHistogramLevels(1024)
matcher.SetNumberOfMatchPoints(7)
matcher.ThresholdAtMeanIntensityOn()
moving = matcher.Execute(moving,fixed)

# The fast symmetric forces Demons Registration Filter
# Note there is a whole family of Demons Registration algorithms included in SimpleITK
demons = sitk.FastSymmetricForcesDemonsRegistrationFilter()
demons.SetNumberOfIterations(200)
# Standard deviation for Gaussian smoothing of displacement field
demons.SetStandardDeviations(1.0)

demons.AddCommand( sitk.sitkIterationEvent, lambda: command_iteration(demons) )

if len( sys.argv ) > 4:
    initialTransform = sitk.ReadTransform(sys.argv[3])
    sys.argv[-1] = sys.argv.pop()

    toDisplacementFilter = sitk.TransformToDisplacementFieldFilter()
    toDisplacementFilter.SetReferenceImage(fixed)

    displacementField = toDisplacementFilter.Execute(initialTransform)

    displacementField = demons.Execute(fixed, moving, displacementField)

else:

    displacementField = demons.Execute(fixed, moving)


print("-------")
print("Number Of Iterations: {0}".format(demons.GetElapsedIterations()))
print(" RMS: {0}".format(demons.GetRMSChange()))

outTx = sitk.DisplacementFieldTransform(displacementField)

sitk.WriteTransform(outTx, sys.argv[3])

if (not "SITK_NOSHOW" in os.environ):

    resampler = sitk.ResampleImageFilter()
    resampler.SetReferenceImage(fixed);
    resampler.SetInterpolator(sitk.sitkLinear)
```

(continues on next page)

(continued from previous page)

```
    resampler.SetDefaultPixelValue(100)
    resampler.SetTransform(outTx)

    out = resampler.Execute(moving)
    simg1 = sitk.Cast(sitk.RescaleIntensity(fixed), sitk.sitkUInt8)
    simg2 = sitk.Cast(sitk.RescaleIntensity(out), sitk.sitkUInt8)
    cimg = sitk.Compose(simg1, simg2, simg1//2.+simg2//2.)
    sitk.Show( cimg, "DeformableRegistration1 Composition" )
```

R

```
# Run with:
#
# Rscript --vanilla DemonsRegistration2.R fixedImageFilter movingImageFile␣
↪[initialTransformFile] outputTransformFile
#

library(SimpleITK)

commandIteration <- function(filter)
{
    msg <- paste("Iteration number ", filter$GetElapsedIterations(),
                 " = ", filter$GetMetric(), "\n" )
    cat(msg)
}

args <- commandArgs( TRUE )

if (length(args) < 3) {
    stop("3 (or 4) arguments expected - fixedImageFilter, movingImageFile,␣
↪[initialTransformFile], outputTransformFile")
}

fixed <- ReadImage(args[[1]])

moving <- ReadImage(args[[2]])

matcher <- HistogramMatchingImageFilter()
if ( fixed$GetPixelID() %in% c( 'sitkUInt8', 'sitkInt8' ) ) {
    matcher$SetNumberOfHistogramLevels(128)
} else {
    matcher$SetNumberOfHistogramLevels(1024)
}
matcher$SetNumberOfMatchPoints(7)
matcher$ThresholdAtMeanIntensityOn()
moving <- matcher$Execute(moving, fixed)

# The fast symmetric forces Demons Registration Filter
# Note there is a whole family of Demons Registration algorithms included in SimpleITK
demons <- FastSymmetricForcesDemonsRegistrationFilter()
demons$SetNumberOfIterations(200)
# Standard deviation for Gaussian smoothing of displacement field
demons$SetStandardDeviations(1.0)

demons$AddCommand( 'sitkIterationEvent', function() commandIteration(demons) )
```

(continues on next page)

(continued from previous page)

```
if (length(args) > 3) {
    initialTransform <- ReadTransform(args[[3]])

    toDisplacementFilter <- TransformToDisplacementFieldFilter()
    toDisplacementFilter$SetReferenceImage(fixed)

    displacementField <- toDisplacementFilter$Execute(initialTransform)

    displacementField <- demons$Execute(fixed, moving, displacementField)
} else {
    displacementField <- demons$Execute(fixed, moving)
}


cat("-------\n")
cat("Number Of Iterations: ",demons$GetElapsedIterations(), "\n")
cat(" RMS: ", demons$GetRMSChange(), "\n")

outTx <- DisplacementFieldTransform(displacementField)

WriteTransform(outTx, tail(args, n=1))
```

11.5 Read Image Meta-Data Dictionary and Print

11.5.1 Overview

This example illustrates how to read only an image's information and meta-data dictionary without loading the pixel content via the ImageFileReader.

Reading an entire image potentially is memory and time intensive operation when the image is large or many files must be read. The image information and meta-data dictionary can be read without the bulk data by using the ImageFilerReader's object oriented interface, with use of the ImageFileReader::ReadImageInformation method.

While all file formats support loading image information such as size, pixel type, origin, and spacing many image types do not have a meta-data dictionary. The most common case for images with a dictionary is DICOM, but also the fields from TIFF, NIFTI, MetaIO and other file formats maybe loaded into the meta-data dictionary.

For efficiency, the default DICOM reader settings will only load public tags (even group numbers). In the example we explicitly set the reader to also load private tags (odd group numbers). For further information on DICOM data elements see the standard part 5, Data Structures and Encoding.

See also *Dicom Series Read Modify Write*, *Dicom Series Reader*.

11.5.2 Code

Python

```
#!/usr/bin/env python

from __future__ import print_function

import SimpleITK as sitk
import sys, os
```

(continues on next page)

(continued from previous page)

```
if len ( sys.argv ) < 2:
    print( "Usage: DicomImagePrintTags <input_file>" )
    sys.exit ( 1 )

reader = sitk.ImageFileReader()

reader.SetFileName( sys.argv[1] )
reader.LoadPrivateTagsOn();

reader.ReadImageInformation();

for k in reader.GetMetaDataKeys():
    v = reader.GetMetaData(k)
    print("({0}) = = \"{1}\"".format(k,v))

print("Image Size: {0}".format(reader.GetSize()));
print("Image PixelType: {0}".format(sitk.GetPixelIDValueAsString(reader.
↪GetPixelID())));
```

R

```
# Run with:
#
# Rscript --vanilla DicomImagePrintTags.R input_file
#

library(SimpleITK)

args <- commandArgs( TRUE )

if (length(args) <  1) {
   write("Usage arguments: <input_file>", stderr())
   quit(1)
}

reader <- ImageFileReader()

reader$SetFileName(args[[1]])
reader$LoadPrivateTagsOn()

reader$ReadImageInformation()

keys <- reader$GetMetaDataKeys()

for ( k in keys)
{
 print(sprintf("(%s) = \"%s\"", k, reader$GetMetaData(k)))
}

cat("Image Size: ", reader$GetSize(), '\n')

pixelType = GetPixelIDValueAsString(reader$GetPixelID())

cat("Image PixelType: ", pixelType, '\n')
```

11.6 Dicom Series Reader

11.6.1 Overview

This example illustrates how to read a DICOM series into a 3D volume. Additional actions include printing some information, writing the image and possibly displaying it using the default display program via the SimpleITK *Show* function. The program makes several assumptions: the given directory contains at least one DICOM series, if there is more than one series the first series is read, and the default SimpleITK external viewer is installed.

See also *Dicom Series Read Modify Write*, *Read Image Meta-Data Dictionary and Print*.

11.6.2 Code

C++

```
// This one header will include all SimpleITK filters and external
// objects.
#include <SimpleITK.h>
#include <iostream>
#include <stdlib.h>


// create convenient namespace alias
namespace sitk = itk::simple;

int main ( int argc, char* argv[] ) {

  if ( argc < 3 ) {
    std::cerr << "Usage: " << argv[0] << " <input_directory> <output_file>\n";
    return 1;
  }

  std::cout << "Reading Dicom directory: " << argv[1] << std::endl;

  // Read the Dicom image series
  sitk::ImageSeriesReader reader;
  const std::vector<std::string> dicom_names =␣
↪sitk::ImageSeriesReader::GetGDCMSeriesFileNames( argv[1] );
  reader.SetFileNames( dicom_names );

  sitk::Image image = reader.Execute();

  std::vector<unsigned int> size = image.GetSize();
  std::cout << "Image size: " << size[0] << " " << size[1] << " " << size[2] <<␣
↪std::endl;

  std::cout << "Writing " << argv[2] << std::endl;

  sitk::WriteImage( image, argv[2] );

  return 0;
}
```

Java

```
import org.itk.simple.*;

public class DicomSeriesReader {
  public static void main(String[] args) {

    if (args.length < 2) {
      System.out.println("Usage: DicomSeriesReader <input_directory> <output_file>");
      System.exit(1);
    }

    System.out.println("Reading Dicom directory: " + args[0]);

    ImageSeriesReader imageSeriesReader = new ImageSeriesReader();
    final VectorString dicomNames = ImageSeriesReader.getGDCMSeriesFileNames(args[0]);
    imageSeriesReader.setFileNames(dicomNames);

    Image image = imageSeriesReader.execute();

    VectorUInt32 size = image.getSize();
    System.out.println("Image size: " + size.get(0) + " " + size.get(1) + " " + size.
↪get(2));

    System.out.println("Writing " + args[1]);

    SimpleITK.writeImage(image, args[1]);
  }
}
```

Lua

```
require "SimpleITK"

if #arg < 2 then
  print ( "Usage: DicomSeriesReader <input_directory> <output_file>" )
  os.exit ( 1 )
end

print( "Reading Dicom directory:", arg[1] )
reader = SimpleITK.ImageSeriesReader()

dicom_names = SimpleITK.ImageSeriesReader.GetGDCMSeriesFileNames( arg[1] )
reader:SetFileNames ( dicom_names )

image = reader:Execute();

size = image:GetSize();
print("Image size:", size[0], size[1], size[2]);

print( "Writing image:", arg[2] )
SimpleITK.WriteImage( image, arg[2] )
```

Python

```
#!/usr/bin/env python

from __future__ import print_function
```

(continues on next page)

(continued from previous page)

```
import SimpleITK as sitk
import sys, os

if len ( sys.argv ) < 3:
    print( "Usage: DicomSeriesReader <input_directory> <output_file>" )
    sys.exit ( 1 )


print( "Reading Dicom directory:", sys.argv[1] )
reader = sitk.ImageSeriesReader()

dicom_names = reader.GetGDCMSeriesFileNames( sys.argv[1] )
reader.SetFileNames(dicom_names)

image = reader.Execute()

size = image.GetSize()
print( "Image size:", size[0], size[1], size[2] )

print( "Writing image:", sys.argv[2] )

sitk.WriteImage( image, sys.argv[2] )

if ( not "SITK_NOSHOW" in os.environ ):
    sitk.Show( image, "Dicom Series" )
```

R

```
library(SimpleITK)

args <- commandArgs( TRUE )

if (length(args) <  2) {
   write("Usage arguments: <input_directory> <output_file>", stderr())
   quit(1)
}


cat( "Reading Dicom directory:", args[[1]] )

reader <- ImageSeriesReader()

dicomNames <- ImageSeriesReader_GetGDCMSeriesFileNames(args[[1]])
reader$SetFileNames(dicomNames)

image <- reader$Execute()

size <- image$GetSize()

cat ("Image size:", size[1], size[2], size[3])

cat("Writing image:", args[[2]])

WriteImage(image, args[[2]])
```

11.7 Dicom Series Read Modify Write

11.7.1 Overview

This example illustrates how to read a DICOM series, modify the 3D image, and then write the result as a DICOM series.

Reading the DICOM series is a three step process: first obtain the series ID, then obtain the file names associated with the series ID, and finally use the series reader to read the images. By default the DICOM meta-data dicitonary for each of the slices is not read. In this example we configure the series reader to load the meta-data dictionary including all of the private tags.

Modifying the 3D image can involve changes to its physical charecteristics (spacing, direction cosines) and its intensities. In our case we only modify the intensities by blurring the image.

Writing the 3D image as a DICOM series is done by configuring the meta-data dictionary for each of the slices and then writing it in DICOM format. In our case we copy some of the meta-data from the original dictionaries which are available from the series reader. We then set some additional meta-data values to indicate that this series is derived from the original acquired data. Note that we write the intensity values as is and thus do not set the rescale slope (0028|1053), rescale intercept (0028|1052) meta-data dictionary values.

See also *Read Image Meta-Data Dictionary and Print*, *Dicom Series Reader*.

11.7.2 Code

Python

```
from __future__ import print_function

import SimpleITK as sitk

import sys, time, os

if len( sys.argv ) < 3:
    print( "Usage: python " + __file__ + " <input_directory_with_DICOM_series>
↪<output_directory>" )
    sys.exit ( 1 )


# Read the original series. First obtain the series file names using the
# image series reader.
data_directory = sys.argv[1]
series_IDs = sitk.ImageSeriesReader.GetGDCMSeriesIDs(data_directory)
if not series_IDs:
    print("ERROR: given directory \""+data_directory+"\" does not contain a DICOM␣
↪series.")
    sys.exit(1)
series_file_names = sitk.ImageSeriesReader.GetGDCMSeriesFileNames(data_directory,␣
↪series_IDs[0])

series_reader = sitk.ImageSeriesReader()
series_reader.SetFileNames(series_file_names)

# Configure the reader to load all of the DICOM tags (public+private):
# By default tags are not loaded (saves time).
# By default if tags are loaded, the private tags are not loaded.
```

(continues on next page)

(continued from previous page)

```
# We explicitly configure the reader to load tags, including the
# private ones.
series_reader.MetaDataDictionaryArrayUpdateOn()
series_reader.LoadPrivateTagsOn()
image3D = series_reader.Execute()

# Modify the image (blurring)
filtered_image = sitk.DiscreteGaussian(image3D)

# Write the 3D image as a series
# IMPORTANT: There are many DICOM tags that need to be updated when you modify an
#            original image. This is a delicate opration and requires knowlege of
#            the DICOM standard. This example only modifies some. For a more complete
#            list of tags that need to be modified see:
#                          http://gdcm.sourceforge.net/wiki/index.php/Writing_DICOM

writer = sitk.ImageFileWriter()
# Use the study/series/frame of reference information given in the meta-data
# dictionary and not the automatically generated information from the file IO
writer.KeepOriginalImageUIDOn()

# Copy relevant tags from the original meta-data dictionary (private tags are also
# accessible).
tags_to_copy = ["0010|0010", # Patient Name
                "0010|0020", # Patient ID
                "0010|0030", # Patient Birth Date
                "0020|000D", # Study Instance UID, for machine consumption
                "0020|0010", # Study ID, for human consumption
                "0008|0020", # Study Date
                "0008|0030", # Study Time
                "0008|0050", # Accession Number
                "0008|0060"  # Modality
]

modification_time = time.strftime("%H%M%S")
modification_date = time.strftime("%Y%m%d")

# Copy some of the tags and add the relevant tags indicating the change.
# For the series instance UID (0020|000e), each of the components is a number, cannot␣
↪start
# with zero, and separated by a '.' We create a unique series ID using the date and␣
↪time.
# tags of interest:
direction = filtered_image.GetDirection()
series_tag_values = [(k, series_reader.GetMetaData(0,k)) for k in tags_to_copy if␣
↪series_reader.HasMetaDataKey(0,k)] + \
                [("0008|0031",modification_time), # Series Time
                 ("0008|0021",modification_date), # Series Date
                 ("0008|0008","DERIVED\\SECONDARY"), # Image Type
                 ("0020|000e", "1.2.826.0.1.3680043.2.1125."+modification_date+".1
↪"+modification_time), # Series Instance UID
                 ("0020|0037", '\\'.join(map(str, (direction[0], direction[3],␣
↪direction[6],# Image Orientation (Patient)
                                                   direction[1],direction[4],
↪direction[7])))),
                 ("0008|103e", series_reader.GetMetaData(0,"0008|103e") + "␣
↪Processed-SimpleITK")] # Series Description
```

(continues on next page)

(continued from previous page)

```
for i in range(filtered_image.GetDepth()):
    image_slice = filtered_image[:,:,i]
    # Tags shared by the series.
    for tag, value in series_tag_values:
        image_slice.SetMetaData(tag, value)
    # Slice specific tags.
    image_slice.SetMetaData("0008|0012", time.strftime("%Y%m%d")) # Instance Creation␣
↪Date
    image_slice.SetMetaData("0008|0013", time.strftime("%H%M%S")) # Instance Creation␣
↪Time
    image_slice.SetMetaData("0020|0032", '\\'.join(map(str,filtered_image.
↪TransformIndexToPhysicalPoint((0,0,i))))) # Image Position (Patient)
    image_slice.SetMetaData("0020|0013", str(i)) # Instance Number

    # Write to the output directory and add the extension dcm, to force writing in␣
↪DICOM format.
    writer.SetFileName(os.path.join(sys.argv[2],str(i)+'.dcm'))
    writer.Execute(image_slice)
sys.exit( 0 )
```

R

```
library(SimpleITK)

args <- commandArgs( TRUE )

if (length(args) < 2) {
   write("Usage arguments: <input_directory_with_DICOM_series> <output_directory>",␣
↪stderr())
   quit(save="no", status=1)
}

# Read the original series. First obtain the series file names using the
# image series reader.
data_directory <- args[1]
series_IDs <- ImageSeriesReader_GetGDCMSeriesIDs(data_directory)
if(length(series_IDs)==0) {
    write(paste0("ERROR: given directory \"",data_directory,"\" does not contain a␣
↪DICOM series."))
    quit(save="no", status=1)
}
series_file_names <- ImageSeriesReader_GetGDCMSeriesFileNames(data_directory, series_
↪IDs[1])

series_reader <- ImageSeriesReader()
series_reader$SetFileNames(series_file_names)

# Configure the reader to load all of the DICOM tags (public+private):
# By default tags are not loaded (saves time).
# By default if tags are loaded, the private tags are not loaded.
# We explicitly configure the reader to load tags, including the
# private ones.
series_reader$MetaDataDictionaryArrayUpdateOn()
series_reader$LoadPrivateTagsOn()
image3D <- series_reader$Execute()
```

(continues on next page)

(continued from previous page)

```
# Modify the image (blurring)
filtered_image <- DiscreteGaussian(image3D)

# Write the 3D image as a series
# IMPORTANT: There are many DICOM tags that need to be updated when you modify an
#            original image. This is a delicate opration and requires knowlege of
#            the DICOM standard. This example only modifies some. For a more complete
#            list of tags that need to be modified see:
#                           http://gdcm.sourceforge.net/wiki/index.php/Writing_DICOM

writer <- ImageFileWriter()
# Use the study/series/frame of reference information given in the meta-data
# dictionary and not the automatically generated information from the file IO
writer$KeepOriginalImageUIDOn()

# Copy relevant tags from the original meta-data dictionary (private tags are also
# accessible).
tags_to_copy <- c("0010|0010", # Patient Name
                  "0010|0020", # Patient ID
                  "0010|0030", # Patient Birth Date
                  "0020|000D", # Study Instance UID, for machine consumption
                  "0020|0010", # Study ID, for human consumption
                  "0008|0020", # Study Date
                  "0008|0030", # Study Time
                  "0008|0050", # Accession Number
                  "0008|0060"  # Modality
)

modification_time <- format(Sys.time(), "%H%M%S")
modification_date <- format(Sys.time(), "%Y%m%d")

# Copy some of the tags and add the relevant tags indicating the change.
# When attempting to copy we are not ensured that the tags exist, in which case
# we get a NULL as an entry in the list returned by lapply, these are removed
# by the Filter.
# For the series instance UID (0020|000e), each of the components is a number, cannot␣
↪start
# with zero, and separated by a '.' We create a unique series ID using the date and␣
↪time.
# tags of interest:
direction <- filtered_image$GetDirection()
series_tag_values <- c(Filter(Negate(is.null),
                              lapply(tags_to_copy,
                              function(k) {
                                if(series_reader$HasMetaDataKey(0,k)) {
                                  list(k, series_reader$GetMetaData(0,k))
                                }
                              })),
                       list(list("0008|0031",modification_time), # Series Time
                              list("0008|0021",modification_date), # Series Date
                              list("0008|0008","DERIVED\\SECONDARY"), # Image Type
                              list("0020|000e", paste0("1.2.826.0.1.3680043.2.1125.",
↪modification_date,".1",modification_time)), # Series Instance UID
                              list("0020|0037", paste(c(direction[1], direction[4],␣
↪direction[7],
                                                        direction[2],direction[5],
↪direction[8]), collapse="\\")), # Image Orientation (Patient)
```

(continues on next page)

(continued from previous page)

```
                              list("0008|103e", paste0(series_reader$GetMetaData(0,
↪"0008|103e"), " Processed-SimpleITK")))) # Series Description

for(i in 1:filtered_image$GetDepth()) {
  image_slice <- filtered_image[,,i]
  # Tags shared by the series.
  invisible(lapply(series_tag_values,
                   function(tag_value) {image_slice$SetMetaData(tag_value[1], tag_
↪value[2])}))
  # Slice specific tags.
  image_slice$SetMetaData("0008|0012", format(Sys.time(), "%Y%m%d")) # Instance␣
↪Creation Date
  image_slice$SetMetaData("0008|0013", format(Sys.time(), "%H%M%S")) # Instance␣
↪Creation Time
  image_slice$SetMetaData("0020|0032", paste(filtered_image
↪$TransformIndexToPhysicalPoint(c(0,0,i-1)), collapse="\\")) # Image Position␣
↪(Patient)
  image_slice$SetMetaData("0020|0013", as.character(i)) # Instance Number

  # Write to the output directory and add the extension dcm, to force writing in␣
↪DICOM format.
  writer$SetFileName(file.path(args[[2]], sprintf("%d.dcm",i)))
  writer$Execute(image_slice)
}

quit(save="no", status=0)
```

11.8 Dicom Series From Array

11.8.1 Overview

This example illustrates how to write a DICOM series from a numeric array and create appropriate meta-data so it can be read by DICOM viewers.

Generating an array is done using a simple random number generator for this case but can come from other sources.

Writing the 3D image as a DICOM series is done by configuring the meta-data dictionary for each of the slices and then writing it in DICOM format. In our case we generate all of the meta-data to indicate that this series is derived. Note that we write the intensity values as is and thus do not set the rescale slope (0028|1053), rescale intercept (0028|1052) meta-data dictionary values.

See also *Read Image Meta-Data Dictionary and Print*, *Dicom Series Reader*.

11.8.2 Code

Python

```
from __future__ import print_function

import SimpleITK as sitk

import sys, time, os
import numpy as np
```

(continues on next page)

(continued from previous page)

```
if len( sys.argv ) < 2:
    print( "Usage: python " + __file__ + "<output_directory>" )
    sys.exit ( 1 )

def writeSlices(series_tag_values, new_img, i):
    image_slice = new_img[:,:,i]

    # Tags shared by the series.
    list(map(lambda tag_value: image_slice.SetMetaData(tag_value[0], tag_value[1]),␣
↪series_tag_values))

    # Slice specific tags.
    image_slice.SetMetaData("0008|0012", time.strftime("%Y%m%d")) # Instance Creation␣
↪Date
    image_slice.SetMetaData("0008|0013", time.strftime("%H%M%S")) # Instance Creation␣
↪Time

    # Setting the type to CT preserves the slice location.
    image_slice.SetMetaData("0008|0060", "CT")  # set the type to CT so the thickness␣
↪is carried over

    # (0020, 0032) image position patient determines the 3D spacing between slices.
    image_slice.SetMetaData("0020|0032", '\\'.join(map(str,new_img.
↪TransformIndexToPhysicalPoint((0,0,i))))) # Image Position (Patient)
    image_slice.SetMetaData("0020,0013", str(i)) # Instance Number

    # Write to the output directory and add the extension dcm, to force writing in␣
↪DICOM format.
    writer.SetFileName(os.path.join(sys.argv[1],str(i)+'.dcm'))
    writer.Execute(image_slice)

# Create a new series from a numpy array
new_arr = np.random.uniform(-10, 10, size = (3,4,5)).astype(np.int16)
new_img = sitk.GetImageFromArray(new_arr)
new_img.SetSpacing([2.5,3.5,4.5])

# Write the 3D image as a series
# IMPORTANT: There are many DICOM tags that need to be updated when you modify an
#            original image. This is a delicate opration and requires knowlege of
#            the DICOM standard. This example only modifies some. For a more complete
#            list of tags that need to be modified see:
#                           http://gdcm.sourceforge.net/wiki/index.php/Writing_DICOM
#            If it is critical for your work to generate valid DICOM files,
#            It is recommended to use David Clunie's Dicom3tools to validate the␣
↪files
#                           (http://www.dclunie.com/dicom3tools.html).

writer = sitk.ImageFileWriter()
# Use the study/series/frame of reference information given in the meta-data
# dictionary and not the automatically generated information from the file IO
writer.KeepOriginalImageUIDOn()

modification_time = time.strftime("%H%M%S")
modification_date = time.strftime("%Y%m%d")

# Copy some of the tags and add the relevant tags indicating the change.
```

(continues on next page)

(continued from previous page)

```
# For the series instance UID (0020|000e), each of the components is a number, cannot␣
↪start
# with zero, and separated by a '.' We create a unique series ID using the date and␣
↪time.
# tags of interest:
direction = new_img.GetDirection()
series_tag_values = [("0008|0031",modification_time), # Series Time
                  ("0008|0021",modification_date), # Series Date
                  ("0008|0008","DERIVED\\SECONDARY"), # Image Type
                  ("0020|000e", "1.2.826.0.1.3680043.2.1125."+modification_date+".1
↪"+modification_time), # Series Instance UID
                  ("0020|0037", '\\'.join(map(str, (direction[0], direction[3],␣
↪direction[6],# Image Orientation (Patient)
                                                    direction[1],direction[4],
↪direction[7])))),
                  ("0008|103e", "Created-SimpleITK")] # Series Description

# Write slices to output directory
list(map(lambda i: writeSlices(series_tag_values, new_img, i), range(new_img.
↪GetDepth())))

# Re-read the series
# Read the original series. First obtain the series file names using the
# image series reader.
data_directory = sys.argv[1]
series_IDs = sitk.ImageSeriesReader.GetGDCMSeriesIDs(data_directory)
if not series_IDs:
    print("ERROR: given directory \""+data_directory+"\" does not contain a DICOM␣
↪series.")
    sys.exit(1)
series_file_names = sitk.ImageSeriesReader.GetGDCMSeriesFileNames(data_directory,␣
↪series_IDs[0])

series_reader = sitk.ImageSeriesReader()
series_reader.SetFileNames(series_file_names)

# Configure the reader to load all of the DICOM tags (public+private):
# By default tags are not loaded (saves time).
# By default if tags are loaded, the private tags are not loaded.
# We explicitly configure the reader to load tags, including the
# private ones.
series_reader.LoadPrivateTagsOn()
image3D = series_reader.Execute()
print(image3D.GetSpacing(),'vs',new_img.GetSpacing())
sys.exit( 0 )
```

R

```
# Run with:
#
# Rscript --vanilla DicomSeriesFromArray.R output_directory
#

library(SimpleITK)

args <- commandArgs( TRUE )
```

(continues on next page)

(continued from previous page)

```
if (length(args) < 1) {
   stop("1 argument expected - output_directory")
}

writeSlices <- function(series_tag_values, new_img, i) {
    image_slice <- new_img[1:new_img$GetWidth(), 1:new_img$GetHeight(), i]

    # Tags shared by the series.
    lapply(1:nrow(series_tag_values),
      function(tag_index){image_slice$SetMetaData(series_tag_values[tag_index, 1],␣
↪series_tag_values[tag_index, 2])})

    # Slice specific tags.
    image_slice$SetMetaData("0008|0012", format(Sys.time(), "%H%M%S")) # Instance␣
↪Creation Date
    image_slice$SetMetaData("0008|0013", format(Sys.time(), "%Y%m%d")) # Instance␣
↪Creation Time

    # Setting the type to CT preserves the slice location.
    image_slice$SetMetaData("0008|0060", "CT")  # set the type to CT so the thickness␣
↪is carried over

    # (0020, 0032) image position patient determines the 3D spacing between slices.
    image_slice$SetMetaData("0020|0032", paste(new_img
↪$TransformIndexToPhysicalPoint(c(0,0,i)), collapse='\\')) # Image Position (Patient)
    image_slice$SetMetaData("0020,0013", i-1) # Instance Number

    # Write to the output directory and add the extension dcm, to force writing in␣
↪DICOM format.
    writer$SetFileName(file.path(args[[1]], paste(i-1, '.dcm', sep="")))
    writer$Execute(image_slice)
}

# Create a new series from an array
new_arr <- array(sample(-10:10, 60, replace=T), dim=c(5, 4, 3))
new_img <- as.image(new_arr)
new_img$SetSpacing(c(2.5,3.5,4.5))

# Write the 3D image as a series
# IMPORTANT: There are many DICOM tags that need to be updated when you modify an
#            original image. This is a delicate opration and requires knowlege of
#            the DICOM standard. This example only modifies some. For a more complete
#            list of tags that need to be modified see:
#                          http://gdcm.sourceforge.net/wiki/index.php/Writing_DICOM
#            If it is critical for your work to generate valid DICOM files,
#            It is recommended to use David Clunie's Dicom3tools to validate the files
#                          (http://www.dclunie.com/dicom3tools.html).

writer <- ImageFileWriter()
# Use the study/series/frame of reference information given in the meta-data
# dictionary and not the automatically generated information from the file IO
writer$KeepOriginalImageUIDOn()

modification_time <- format(Sys.time(), "%H%M%S")
modification_date <- format(Sys.time(), "%Y%m%d")

# Copy some of the tags and add the relevant tags indicating the change.
```

(continues on next page)

(continued from previous page)

```
# For the series instance UID (0020|000e), each of the components is a number, cannot␣
↪start
# with zero, and separated by a '.' We create a unique series ID using the date and␣
↪time.
# tags of interest:
direction <- new_img$GetDirection()
series_tag_values = matrix(c("0008|0031",modification_time, # Series Time
                  "0008|0021",modification_date, # Series Date
                  "0008|0008","DERIVED\\SECONDARY", # Image Type
                  "0020|000e", paste("1.2.826.0.1.3680043.2.1125.",modification_date,
↪".1",modification_time, sep=''), # Series Instance UID
                  "0020|0037", paste(direction[[1]], direction[[4]], direction[[7]],#␣
↪Image Orientation (Patient)
                                     direction[[2]],direction[[5]],direction[[8]],␣
↪sep='\\'),
                  "0008|103e", "Created-SimpleITK"), nrow=6, ncol=2, byrow=TRUE) #␣
↪Series Description

# Write slices to output directory
invisible(lapply(1:(new_img$GetDepth()), function(i){writeSlices(series_tag_values,␣
↪new_img, i)}))

# Re-read the series
# Read the original series. First obtain the series file names using the
# image series reader.
data_directory <- args[[1]]
series_IDs <- ImageSeriesReader_GetGDCMSeriesIDs(data_directory)
if (length(series_IDs)==0) {
    stop("ERROR: given directory \"", data_directory, "\" does not contain a DICOM␣
↪series.")
}
series_file_names <- ImageSeriesReader_GetGDCMSeriesFileNames(data_directory, series_
↪IDs[[1]])

series_reader <- ImageSeriesReader()
series_reader$SetFileNames(series_file_names)

# Configure the reader to load all of the DICOM tags (public+private):
# By default tags are not loaded (saves time).
# By default if tags are loaded, the private tags are not loaded.
# We explicitly configure the reader to load tags, including the
# private ones.
series_reader$LoadPrivateTagsOn()
image3D <- series_reader$Execute()
cat(image3D$GetSpacing(),'vs',new_img$GetSpacing(),'\n')
```

11.9 Filter Progress Reporting

11.9.1 Overview

SimpleITK has the ability to add commands or callbacks as observers of events that may occur during data processing. This feature can be used to add progress reporting to a console, to monitor the process of optimization, to abort a process, or to improve the integration of SimpleITK into Graphical User Interface event queues.

11.9.2 Events

Events are a simple enumerated type in SimpleITK, represented by the EventEnum type. More information about each event type can be found in the documentation for the enum. All SimpleITK filters, including the reading and writing ones, are derived from the ProcessObject class which has support for events. SimpleITK utilizes the native ITK event system but has simpler events and methods to add an observer or commands. The goal is to provide a simpler interface more suitable for scripting languages.

11.9.3 Commands

The command design pattern is used to allow user code to be executed when an event occurs. It is encapsulated in the Command class. The Command class provides a virtual Execute method to be overridden in derived classes. Additionally, SimpleITK provides internal reference tracking between the ProcessObject and the Command. This reference tracking allows an object to be created on the stack or dynamically allocated, without additional burden.

11.9.4 Command Directors for Wrapped Languages

SimpleITK uses SWIG's director feature to enable wrapped languages to derive classes from the Command class. Thus a user may override the Command class's Execute method for custom call-backs. The following languages support deriving classes from the Command class:

C#

```
  class MyCommand : Command {

    private ProcessObject m_ProcessObject;

    public MyCommand(ProcessObject po){
      m_ProcessObject = po;
    }

    public override void Execute() {
      Console.WriteLine("{0} Progress: {1:0.00}", m_ProcessObject.GetName(), m_
↪ProcessObject.GetProgress() );
    }
  }
```

Java

```
class MyCommand  extends Command {

  private ProcessObject m_ProcessObject;

  public MyCommand(ProcessObject po) {
    super();
    m_ProcessObject=po;
  }

  public void execute() {
    double progress = m_ProcessObject.getProgress();
    System.out.format("%s Progress: %f\n", m_ProcessObject.getName(), progress);
  }
}
```

Python

```
class MyCommand(sitk.Command):
    def __init__(self, po):
        # required
        super(MyCommand,self).__init__()
        self.processObject = po

    def Execute(self):
        print("{0} Progress: {1:1.2f}".format(self.processObject.GetName(),self.
↪processObject.GetProgress()))
```

Ruby

```
class MyCommand < Simpleitk::Command
  def initialize(po)
    # Explicit call to supoer class initlaizer is required to
    # initialize the SWIG director class to enable overloaded methods
    super()
    @po = po
  end

  # The Command method to be executed on the event from the filter.
  def execute
    puts "%s Progress: %0.2f" % [@po.get_name, @po.get_progress]
  end
end
```

11.9.5 Command Functions and Lambdas for Wrapped Languages

Not all scripting languages are naturally object oriented, and it is often easier to simply define a callback inline with a lambda function. The following language supports inline function definitions for functions for the ProcessObject::AddCommand method:

Python

```
gaussian.AddCommand(sitk.sitkStartEvent, lambda: print("StartEvent"))
gaussian.AddCommand(sitk.sitkEndEvent, lambda: print("EndEvent"))
```

R

```
gaussian$AddCommand( 'sitkStartEvent',  function(method) {cat("StartEvent\n")} )
gaussian$AddCommand( 'sitkEndEvent',  function(method) {cat("EndEvent\n")} )
```

11.9.6 Code

CSharp

```
using System;
using itk.simple;

namespace itk.simple.examples {

//! [csharp director command]
  class MyCommand : Command {
```

(continues on next page)

(continued from previous page)

```
    private ProcessObject m_ProcessObject;

    public MyCommand(ProcessObject po){
      m_ProcessObject = po;
    }

    public override void Execute() {
      Console.WriteLine("{0} Progress: {1:0.00}", m_ProcessObject.GetName(), m_
↪ProcessObject.GetProgress() );
    }
  }
//! [csharp director command]

  class FilterProgressReporting {

    static void Main(string[] args) {
      try {
        if (args.Length < 3) {
          Console.WriteLine("Usage: {0} <input> <variance> <output>", args[0]);
          return;
        }
        // Read input image
        ImageFileReader reader = new ImageFileReader();
        reader.SetFileName(args[0]);
        Image image = reader.Execute();

        // Execute Gaussian smoothing filter
        DiscreteGaussianImageFilter filter = new DiscreteGaussianImageFilter();
        filter.SetVariance(Double.Parse(args[1]));

        MyCommand cmd = new MyCommand(filter);
        filter.AddCommand(EventEnum.sitkProgressEvent, cmd);

        image = filter.Execute(image);

        // Write output image
        ImageFileWriter writer = new ImageFileWriter();
        writer.SetFileName(args[2]);
        writer.Execute(image);

      } catch (Exception ex) {
        Console.WriteLine(ex);
      }
    }
  }
}
```

C++

```
// This one header will include all SimpleITK filters and external
// objects.
#include <SimpleITK.h>
#include <iostream>
#include <stdlib.h>
#include <iomanip>

// create convenient namespace alias
```

(continues on next page)

(continued from previous page)

```
namespace sitk = itk::simple;

// Create A Command Callback to be registered with the ProcessObject
// on the ProgressEvent
//
// Internally we maintain a reference to the ProcessObject passed
// during construction. Therefore, it would be an error to execute the
// Execute method after the ProcessObject is delete. But this class
// can be created on the stack, with out issue.
class ProgressUpdate
  : public sitk::Command
{
public:
  ProgressUpdate(const sitk::ProcessObject &po)
    : m_Process(po)
    {}

  virtual void Execute( )
    {
      // stash the stream state
      std::ios  state(NULL);
      state.copyfmt(std::cout);
      std::cout << std::fixed << std::setw( 3 ) << std::setprecision( 2 );

      // Print the Progress "Active Measurement"
      std::cout << m_Process.GetName()<< " Progress: " << m_Process.GetProgress() <<␣
↪std::endl;

      std::cout.copyfmt(state);
    }
private:
  const sitk::ProcessObject &m_Process;
};


int main ( int argc, char* argv[] ) {

  if ( argc < 4 ) {
    std::cerr << "Usage: " << argv[0] << " <input> <variance> <output>\n";
    return 1;
  }

  // Read the image file
  sitk::ImageFileReader reader;
  reader.SetFileName ( std::string ( argv[1] ) );
  sitk::Image image = reader.Execute();

  // This filters perform a gaussian bluring with sigma in physical
  // space. The output image will be of real type.
  sitk::DiscreteGaussianImageFilter gaussian;
  gaussian.SetVariance ( atof ( argv[2] ) );

  // Construct our custom command on the stack
  ProgressUpdate cmd(gaussian);
  // register it with the filter for the ProgressEvent
  gaussian.AddCommand( sitk::sitkProgressEvent, cmd );
```

(continues on next page)

(continued from previous page)

```
  sitk::Image blurredImage = gaussian.Execute ( image );

  // Covert the real output image back to the original pixel type, to
  // make writing easier, as many file formats don't support real
  // pixels.
  sitk::CastImageFilter caster;
  caster.SetOutputPixelType( image.GetPixelID() );
  sitk::Image outputImage = caster.Execute( blurredImage );

  // write the image
  sitk::ImageFileWriter writer;
  writer.SetFileName ( std::string ( argv[3] ) );
  writer.Execute ( outputImage );

  return 0;
}
```

Java

```
import org.itk.simple.*;

//! [java director command]
class MyCommand  extends Command {

  private ProcessObject m_ProcessObject;

  public MyCommand(ProcessObject po) {
    super();
    m_ProcessObject=po;
  }

  public void execute() {
    double progress = m_ProcessObject.getProgress();
    System.out.format("%s Progress: %f\n", m_ProcessObject.getName(), progress);
  }
}
//! [java director command]

class FilterProgressReporting {

  public static void main(String argv[]) {
    if ( argv.length < 3 ) {
      System.out.format("Usage: java %s <input> <variance> <output>",
↪"FilterProgressReporting" );
      System.exit(-1);
    }

      org.itk.simple.ImageFileReader reader = new org.itk.simple.ImageFileReader();
      reader.setFileName(argv[0]);
      Image img = reader.execute();

      DiscreteGaussianImageFilter filter = new DiscreteGaussianImageFilter();
      filter.setVariance(Double.valueOf( argv[1] ).doubleValue());

      MyCommand cmd = new MyCommand(filter);
      filter.addCommand(EventEnum.sitkProgressEvent, cmd);
```

(continues on next page)

(continued from previous page)

```
      Image blurredImg = filter.execute(img);

      CastImageFilter caster = new CastImageFilter();
      caster.setOutputPixelType(img.getPixelID());
      Image castImg = caster.execute(blurredImg);

      ImageFileWriter writer = new ImageFileWriter();
      writer.setFileName(argv[2]);

      writer.execute(castImg);

    }

}
```

Python

```
#!/usr/bin/env python

from __future__ import print_function

import SimpleITK as sitk
import sys
import os

if len ( sys.argv ) < 4:
    print( "Usage: "+sys.argv[0]+ " <input> <variance> <output>" )
    sys.exit ( 1 )

##! [python director command]
class MyCommand(sitk.Command):
    def __init__(self, po):
        # required
        super(MyCommand,self).__init__()
        self.processObject = po

    def Execute(self):
        print("{0} Progress: {1:1.2f}".format(self.processObject.GetName(),self.
↪processObject.GetProgress()))
##! [python director command]

reader = sitk.ImageFileReader()
reader.SetFileName ( sys.argv[1] )
image = reader.Execute()

pixelID = image.GetPixelID()

gaussian = sitk.DiscreteGaussianImageFilter()
gaussian.SetVariance( float ( sys.argv[2] ) )


##! [python lambda command]
gaussian.AddCommand(sitk.sitkStartEvent, lambda: print("StartEvent"))
gaussian.AddCommand(sitk.sitkEndEvent, lambda: print("EndEvent"))
##! [python lambda command]

cmd = MyCommand(gaussian)
```

(continues on next page)

(continued from previous page)

```
gaussian.AddCommand(sitk.sitkProgressEvent, cmd)

image = gaussian.Execute ( image )

caster = sitk.CastImageFilter()
caster.SetOutputPixelType( pixelID )
image = caster.Execute( image )

writer = sitk.ImageFileWriter()
writer.SetFileName ( sys.argv[3] )
writer.Execute ( image );


if ( not "SITK_NOSHOW" in os.environ ):
    sitk.Show( image, "Simple Gaussian" )
```

R

```
# Run with:
#
# Rscript --vanilla FilterProgressReporting.R input variance output
#


library(SimpleITK)

args <- commandArgs( TRUE )

if (length(args) <  3){
   write("Usage arguments: <input> <variance> <output>", stderr())
   quit(save="no", status=1)
}

reader <- ImageFileReader()
reader$SetFileName(args[[1]])
image = reader$Execute()

pixelID <- image$GetPixelID()

gaussian <- DiscreteGaussianImageFilter()
gaussian$SetVariance( as.numeric(args[2]) )

##! [R lambda command]
gaussian$AddCommand( 'sitkStartEvent',  function(method) {cat("StartEvent\n")} )
gaussian$AddCommand( 'sitkEndEvent',  function(method) {cat("EndEvent\n")} )
##! [R lambda command]

image = gaussian$Execute( image )

caster <- CastImageFilter()
caster$SetOutputPixelType( pixelID )
image = caster$Execute( image )

writer <- ImageFileWriter()
writer$SetFileName( args[[3]] )
writer$Execute( image )
```

Ruby

```ruby
require 'simpleitk'

if ARGV.length != 3 then
  puts "Usage: SimpleGaussian <input> <sigma> <output>";
  exit( 1 )
end


# Derive a class from SimpleITK Command class to be used to observe
# events and report progress.
## [ruby director command]
class MyCommand < Simpleitk::Command
  def initialize(po)
    # Explicit call to supoer class initlaizer is required to
    # initialize the SWIG director class to enable overloaded methods
    super()
    @po = po
  end

  # The Command method to be executed on the event from the filter.
  def execute
    puts "%s Progress: %0.2f" % [@po.get_name, @po.get_progress]
  end
end
## [ruby director command]

reader = Simpleitk::ImageFileReader.new
reader.set_file_name( ARGV[0] )
image = reader.execute

inputPixelType = image.get_pixel_idvalue


gaussian = Simpleitk::DiscreteGaussianImageFilter.new
gaussian.set_variance ARGV[1].to_f

# create a new MyCommand class, the references between the objects is
# automatically taked care of. The connection will automatically be
# removed when either object is deleted.
cmd = MyCommand.new gaussian
gaussian.add_command Simpleitk::SitkProgressEvent, cmd

image = gaussian.execute image

caster = Simpleitk::CastImageFilter.new
caster.set_output_pixel_type inputPixelType
image = caster.execute image

writer = Simpleitk::ImageFileWriter.new
writer.set_file_name ARGV[2]
writer.execute image
```

11.10 Image Grid Manipulation

11.10.1 Overview

There are numerous SimpleITK filters that have similar functions, but very important differences. The filters that will be compared are:

- `JoinSeriesImageFilter()` - Joins multiple N-D images into an (N+1)-D image
- `ComposeImageFilter()` - Combines several scalar images into a multicomponent vector image
- `VectorIndexSelectionCastImageFilter()` - Extracts the selected index of the input pixel type vector (the input image pixel type must be a vector and the output a scalar). Additionally, this filter can cast the output pixel type (SetOutputPixelType method).
- `ExtractImageFilter()` - Crops an image to the selected region bounds using vectors; Collapses dimensions unless dimension is two
- `CropImageFilter()` - Similar to `ExtractImageFilter()`, but crops an image by an `itk::Size` at upper and lower bounds
- Image Slicing Operator - Uses slicing (`img[i:j, k:l]`) to extract a subregion of an image

All of these operations will maintain the physical location of the pixels, instead modifying the image's metadata.

11.10.2 Comparisons

Composition Filters

While `JoinSeriesImageFilter()` merges multiple images of the same pixel type in N dimensions into an image in N+1 dimensions, `ComposeImageFilter()` will combine scalar images into a vector image of the same dimension. The former is useful for connecting a series of contiguous images while the latter is more useful for merging multiple channels of the same object into one image (such as RGB).

Extraction Filters

`VectorIndexSelectionCastImageFilter()` will isolate a single channel in a vector image and return a scalar image. On the other hand, `ExtractImageFilter()` and `CropImageFilter()` will extract and return a subregion of an image, using an ExtractionRegion size and index and `itk::Size`'s respectively. However, note that only the `ExtractImageFilter()` collapses dimensions. The image slicing operator can also serve the same purpose.

11.10.3 Code

Python

```
#!/usr/bin/env python

from __future__ import print_function

import SimpleITK as sitk
import sys

if len ( sys.argv ) < 3:
```

(continues on next page)

(continued from previous page)

```
    print( "Usage: " +sys.argv[0]+ " <input-1> <input-2>" )
    sys.exit ( 1 )

# Two vector images of same pixel type and dimension expected
image_1 = sitk.ReadImage(sys.argv[1])
image_2 = sitk.ReadImage(sys.argv[2])

# Join two N-D Vector images to form an (N+1)-D image
join = sitk.JoinSeriesImageFilter()
joined_image = join.Execute(image_1, image_2)

# Extract first three channels of joined image (assuming RGB)
select = sitk.VectorIndexSelectionCastImageFilter()
channel1_image = select.Execute(joined_image, 0, sitk.sitkUInt8)
channel2_image = select.Execute(joined_image, 1, sitk.sitkUInt8)
channel3_image = select.Execute(joined_image, 2, sitk.sitkUInt8)

# Recompose image (should be same as joined_image)
compose = sitk.ComposeImageFilter()
composed_image = compose.Execute(channel1_image, channel2_image, channel3_image)

# Select same subregion using image slicing operator
sliced_image = composed_image[100:400, 100:400, 0]

# Select same subregion using ExtractImageFilter
extract = sitk.ExtractImageFilter()
extract.SetSize([300, 300, 0])
extract.SetIndex([100, 100, 0])
extracted_image = extract.Execute(composed_image)

# Select same subregion using CropImageFilter (NOTE: CropImageFilter cannot reduce␣
↪dimensions
# unlike ExtractImageFilter, so cropped_image is a three dimensional image with depth␣
↪of 1)
crop = sitk.CropImageFilter()
crop.SetLowerBoundaryCropSize([100, 100, 0])
crop.SetUpperBoundaryCropSize([composed_image.GetWidth()-400, composed_image.
↪GetHeight()-400, 1])
cropped_image = crop.Execute(composed_image)
```

R

```
# Run with:
#
# Rscript --vanilla ImageGridManipulation.R input-1 input-2
#

library(SimpleITK)

args <- commandArgs( TRUE )

if (length(args) <  2){
   write("Usage arguments: <input-1> <input-2>", stderr())
   quit(save="no", status=1)
}

# Two vector images of same pixel type and dimension expected
```

(continues on next page)

(continued from previous page)

```
reader <- ImageFileReader()
reader$SetFileName(args[[1]])
image_1 <- reader$Execute()
reader$SetFileName(args[[2]])
image_2 <- reader$Execute()

# Join two N-D Vector images to form an (N+1)-D image
join <- JoinSeriesImageFilter()
joined_image <- join$Execute( image_1, image_2 )

# Extract first three channels of joined image (assuming RGB)
select <- VectorIndexSelectionCastImageFilter()
channel1_image <- select$Execute(joined_image, 0, "sitkUInt8")
channel2_image <- select$Execute(joined_image, 1, "sitkUInt8")
channel3_image <- select$Execute(joined_image, 2, "sitkUInt8")

# Recompose image (should be same as joined_image)
compose <- ComposeImageFilter()
composed_image <- compose$Execute(channel1_image, channel2_image, channel3_image)

# Select same subregion using image slicing operator
sliced_image = composed_image[101:400, 101:400, 1]

# Select same subregion using ExtractImageFilter
extract <- ExtractImageFilter()
extract$SetSize( list(300, 300, 0) )
extract$SetIndex( list(100, 100, 0) )
extracted_image <- extract$Execute(composed_image)

# Select same subregion using CropImageFilter (NOTE: CropImageFilter cannot reduce␣
↪dimensions
# unlike ExtractImageFilter, so cropped_image is a three dimensional image with depth␣
↪of 1)
crop <- CropImageFilter()
crop$SetLowerBoundaryCropSize( list(100, 100, 0) )
crop$SetUpperBoundaryCropSize( list(composed_image$GetWidth()-400, composed_image
↪$GetHeight()-400, 1) )
cropped_image <- crop$Execute(composed_image)
```

11.11 Image Registration Method 1

If you are not familiar with the SimpleITK registration framework we recommend that you read the *registration overview* before continuing with the example.

11.11.1 Overview

11.11.2 Code

C#

```
using System;
using itk.simple;
```

(continues on next page)

(continued from previous page)

```
namespace itk.simple.examples {


  class IterationUpdate : Command {

    private ImageRegistrationMethod m_Method;

    public IterationUpdate(ImageRegistrationMethod m) {
      m_Method=m;
    }

    public override void Execute() {
      VectorDouble pos = m_Method.GetOptimizerPosition();
      Console.WriteLine("{0:3} = {1:10.5} : [{2}, {3}]",
                        m_Method.GetOptimizerIteration(),
                        m_Method.GetMetricValue(),
                        pos[0], pos[1]);
    }
  }

  class ImageRegistrationMethod1 {

    static void Main(string[] args) {

      if ( args.Length < 3 )
        {
          Console.WriteLine("Usage: %s <fixedImageFilter> <movingImageFile>
↪<outputTransformFile>\n", "ImageRegistrationMethod1");
         return;
        }

      ImageFileReader reader = new ImageFileReader();
      reader.SetOutputPixelType( PixelIDValueEnum.sitkFloat32 );

      reader.SetFileName(args[0]);
      Image fixedImage = reader.Execute();

      reader.SetFileName(args[1]);
      Image movingImage = reader.Execute();

      ImageRegistrationMethod R = new  ImageRegistrationMethod();
      R.SetMetricAsMeanSquares();
      double maxStep = 4.0;
      double minStep = 0.01;
      uint numberOfIterations = 200;
      double relaxationFactor = 0.5;
      R.SetOptimizerAsRegularStepGradientDescent( maxStep,
                                                  minStep,
                                                  numberOfIterations,
                                                  relaxationFactor );
      R.SetInitialTransform( new TranslationTransform( fixedImage.GetDimension() ) );
      R.SetInterpolator( InterpolatorEnum.sitkLinear );

      IterationUpdate cmd = new IterationUpdate(R);
      R.AddCommand(EventEnum.sitkIterationEvent, cmd);
```

(continues on next page)

(continued from previous page)

```
      Transform outTx = R.Execute( fixedImage, movingImage );

      // System.out.println("-------");
      // System.out.println(outTx.toString());
      // System.out.format("Optimizer stop condition: %s\n", R.
↪getOptimizerStopConditionDescription());
      // System.out.format(" Iteration: %d\n", R.getOptimizerIteration());
      // System.out.format(" Metric value: %f\n", R.getMetricValue());

      outTx.WriteTransform(args[2]);

    }

  }

}
```

C++

```
// This one header will include all SimpleITK filters and external
// objects.
#include <SimpleITK.h>
#include <iostream>
#include <stdlib.h>
#include <iomanip>

namespace sitk = itk::simple;



class IterationUpdate
  : public sitk::Command
{
public:
  IterationUpdate( const sitk::ImageRegistrationMethod &m)
    : m_Method(m)
    {}

  virtual void Execute( )
    {
      // use sitk's output operator for std::vector etc..
      using sitk::operator<<;

      // stash the stream state
      std::ios  state(NULL);
      state.copyfmt(std::cout);
      std::cout << std::fixed << std::setfill(' ') << std::setprecision( 5 );
      std::cout << std::setw(3) << m_Method.GetOptimizerIteration();
      std::cout << " = " << std::setw(10) << m_Method.GetMetricValue();
      std::cout << " : " << m_Method.GetOptimizerPosition() << std::endl;

      std::cout.copyfmt(state);
    }

private:
  const sitk::ImageRegistrationMethod &m_Method;
```

(continues on next page)

(continued from previous page)

```
};


int main(int argc, char *argv[])
{

  if ( argc < 4 )
    {
    std::cerr << "Usage: " << argv[0] << " <fixedImageFilter> <movingImageFile>
↪<outputTransformFile>" << std::endl;
    return 1;
    }

  sitk::Image fixed = sitk::ReadImage( argv[1], sitk::sitkFloat32 );

  sitk::Image moving = sitk::ReadImage( argv[2], sitk::sitkFloat32 );

  sitk::ImageRegistrationMethod R;
  R.SetMetricAsMeanSquares();
  const double maxStep = 4.0;
  const double minStep = 0.01;
  const unsigned int numberOfIterations = 200;
  const double relaxationFactor = 0.5;
  R.SetOptimizerAsRegularStepGradientDescent( maxStep,
                                              minStep,
                                              numberOfIterations,
                                              relaxationFactor );
  R.SetInitialTransform( sitk::TranslationTransform( fixed.GetDimension() ) );
  R.SetInterpolator( sitk::sitkLinear );

  IterationUpdate cmd(R);
  R.AddCommand( sitk::sitkIterationEvent, cmd);

  sitk::Transform outTx = R.Execute( fixed, moving );

  std::cout << "-------" << std::endl;
  std::cout << outTx.ToString() << std::endl;
  std::cout << "Optimizer stop condition: " << R.
↪GetOptimizerStopConditionDescription() << std::endl;
  std::cout << " Iteration: " << R.GetOptimizerIteration() << std::endl;
  std::cout << " Metric value: " << R.GetMetricValue() << std::endl;

  sitk::WriteTransform(outTx, argv[3]);


  return 0;
}
```

Java

```
import org.itk.simple.*;


class IterationUpdate  extends Command {

  private ImageRegistrationMethod m_Method;
```

(continues on next page)

(continued from previous page)

```
  public IterationUpdate(ImageRegistrationMethod m) {
    super();
    m_Method=m;
  }

  public void execute() {
    org.itk.simple.VectorDouble pos = m_Method.getOptimizerPosition();
    System.out.format("%3d = %10.5f : [%f, %f]\n",
                      m_Method.getOptimizerIteration(),
                      m_Method.getMetricValue(),
                      pos.get(0), pos.get(1));
  }
}

class ImageRegistrationMethod1 {

  public static void main(String argv[]) {

  if ( argv.length < 3 )
    {
      System.out.format( "Usage: %s <fixedImageFilter> <movingImageFile>
↪<outputTransformFile>\n", "ImageRegistrationMethod1");
      System.exit(-1);
    }

  org.itk.simple.ImageFileReader reader = new org.itk.simple.ImageFileReader();
  reader.setOutputPixelType( PixelIDValueEnum.sitkFloat32 );

  reader.setFileName(argv[0]);
  Image fixed = reader.execute();

  reader.setFileName(argv[1]);
  Image moving = reader.execute();

  org.itk.simple.ImageRegistrationMethod R = new  org.itk.simple.
↪ImageRegistrationMethod();
  R.setMetricAsMeanSquares();
  double maxStep = 4.0;
  double minStep = 0.01;
  int numberOfIterations = 200;
  double relaxationFactor = 0.5;
  R.setOptimizerAsRegularStepGradientDescent( maxStep,
                                              minStep,
                                              numberOfIterations,
                                              relaxationFactor );
  R.setInitialTransform( new org.itk.simple.TranslationTransform( fixed.
↪getDimension() ) );
  R.setInterpolator( InterpolatorEnum.sitkLinear );

  IterationUpdate cmd = new IterationUpdate(R);
  R.addCommand( EventEnum.sitkIterationEvent, cmd);

  org.itk.simple.Transform outTx = R.execute( fixed, moving );

  System.out.println("-------");
  System.out.println(outTx.toString());
```

(continues on next page)

(continued from previous page)

```
  System.out.format("Optimizer stop condition: %s\n", R.
↪getOptimizerStopConditionDescription());
  System.out.format(" Iteration: %d\n", R.getOptimizerIteration());
  System.out.format(" Metric value: %f\n", R.getMetricValue());

  outTx.writeTransform(argv[2]);

  }

}
```

Lua

```
require "SimpleITK"

function command_iteration(method)
    print(method)
    pos = method:GetOptimizedPosition()
    print( string.format("%3d = %f : (%f, %f)", method:GetOptimizerIteration(),
        method:GetMetricValue(), pos[1], pos[2] ) )
end

if #arg < 3 then
    print( string.format("Usage: %s <fixedImageFilter> <movingImageFile>
↪<outputTransformFile>", arg[0]) )
    os.exit ( 1 )
end

fixed = SimpleITK.ReadImage( arg[1], SimpleITK.sitkFloat32 )

moving = SimpleITK.ReadImage( arg[2], SimpleITK.sitkFloat32 )


R = SimpleITK.ImageRegistrationMethod()
R:SetMetricAsMeanSquares()
R:SetOptimizerAsRegularStepGradientDescent( 4.0, .01, 200 )
R:SetInitialTransform( SimpleITK.Transform( fixed:GetDimension(), SimpleITK.
↪sitkTranslation ) )
R:SetInterpolator( SimpleITK.sitkLinear )

-- callback for progress reporting doesn't work yet in Lua
-- R:AddCommand( SimpleITK.sitkIterationEvent, command_iteration(R) )

outTx = R:Execute(fixed, moving)

print("-------")
print(outTx)
print(string.format("Optimizer stop condition: %s",␣
↪R:GetOptimizerStopConditionDescription()) )
print(" Iteration: ", R:GetOptimizerIteration() )
print(" Metric value: ", R:GetMetricValue() )

SimpleITK.WriteTransform(outTx,  arg[3])
```

Python

```
#!/usr/bin/env python

from __future__ import print_function

import SimpleITK as sitk
import sys
import os


def command_iteration(method) :
    print("{0:3} = {1:10.5f} : {2}".format(method.GetOptimizerIteration(),
                                   method.GetMetricValue(),
                                   method.GetOptimizerPosition()))

if len ( sys.argv ) < 4:
    print( "Usage: {0} <fixedImageFilter> <movingImageFile> <outputTransformFile>".
↪format(sys.argv[0]))
    sys.exit ( 1 )


fixed = sitk.ReadImage(sys.argv[1], sitk.sitkFloat32)

moving = sitk.ReadImage(sys.argv[2], sitk.sitkFloat32)

R = sitk.ImageRegistrationMethod()
R.SetMetricAsMeanSquares()
R.SetOptimizerAsRegularStepGradientDescent(4.0, .01, 200 )
R.SetInitialTransform(sitk.TranslationTransform(fixed.GetDimension()))
R.SetInterpolator(sitk.sitkLinear)

R.AddCommand( sitk.sitkIterationEvent, lambda: command_iteration(R) )

outTx = R.Execute(fixed, moving)

print("-------")
print(outTx)
print("Optimizer stop condition: {0}".format(R.
↪GetOptimizerStopConditionDescription()))
print(" Iteration: {0}".format(R.GetOptimizerIteration()))
print(" Metric value: {0}".format(R.GetMetricValue()))

sitk.WriteTransform(outTx,  sys.argv[3])

if ( not "SITK_NOSHOW" in os.environ ):

    resampler = sitk.ResampleImageFilter()
    resampler.SetReferenceImage(fixed);
    resampler.SetInterpolator(sitk.sitkLinear)
    resampler.SetDefaultPixelValue(100)
    resampler.SetTransform(outTx)

    out = resampler.Execute(moving)
    simg1 = sitk.Cast(sitk.RescaleIntensity(fixed), sitk.sitkUInt8)
    simg2 = sitk.Cast(sitk.RescaleIntensity(out), sitk.sitkUInt8)
    cimg = sitk.Compose(simg1, simg2, simg1//2.+simg2//2.)
    sitk.Show( cimg, "ImageRegistration1 Composition" )
```

R

```
# Run with:
#
# Rscript --vanilla ImageRegistrationMethod1.R fixedImage movingImage outputTransform
#


library(SimpleITK)


commandIteration <- function(method)
{
    msg <- paste("Optimizer iteration number ", method$GetOptimizerIteration(),
                 " = ", method$GetMetricValue(), " : ", method$GetOptimizerPosition(),
                 "\n" )
    cat(msg)
}

args <- commandArgs( TRUE )

if (length(args) != 3) {
    stop("3 arguments expected - FixedImage, MovingImage, TransformFilename")
}

fixed <- ReadImage(args[[1]], 'sitkFloat32')

moving <- ReadImage(args[[2]], 'sitkFloat32')


Reg = ImageRegistrationMethod()
Reg$SetMetricAsMeanSquares()
Reg$SetOptimizerAsRegularStepGradientDescent(4.0, .01, 200 )
Reg$SetInitialTransform(TranslationTransform(fixed$GetDimension()))
Reg$SetInterpolator('sitkLinear')

Reg$AddCommand('sitkIterationEvent', function() commandIteration(Reg))

outTx = Reg$Execute(fixed, moving)

WriteTransform(outTx, args[[3]])
```

11.12 Image Registration Method 2

If you are not familiar with the SimpleITK registration framework we recommend that you read the *registration overview* before continuing with the example.

11.12.1 Overview

11.12.2 Code

C++

```
// This one header will include all SimpleITK filters and external
// objects.
```

(continues on next page)

(continued from previous page)

```
#include <SimpleITK.h>

#include <iostream>
#include <stdlib.h>
#include <iomanip>
#include <numeric>

namespace sitk = itk::simple;



class IterationUpdate
  : public sitk::Command
{
public:
  IterationUpdate( const sitk::ImageRegistrationMethod &m)
    : m_Method(m)
    {}

  virtual void Execute( )
    {
      // use sitk's output operator for std::vector etc..
      using sitk::operator<<;

      // stash the stream state
      std::ios  state(NULL);
      state.copyfmt(std::cout);
      std::cout << std::fixed << std::setfill(' ') << std::setprecision( 5 );
      std::cout << std::setw(3) << m_Method.GetOptimizerIteration();
      std::cout << " = " << std::setw(7) << m_Method.GetMetricValue();
      std::cout << " : " << m_Method.GetOptimizerPosition() << std::endl;

      std::cout.copyfmt(state);
    }

private:
  const sitk::ImageRegistrationMethod &m_Method;

};


int main(int argc, char *argv[])
{

  if ( argc < 4 )
    {
    std::cerr << "Usage: " << argv[0] << " <fixedImageFilter> <movingImageFile>
↪<outputTransformFile>" << std::endl;
    return 1;
    }


  sitk::Image fixed = sitk::ReadImage( argv[1], sitk::sitkFloat32 );
  fixed = sitk::Normalize( fixed );
  fixed = sitk::DiscreteGaussian( fixed, 2.0 );

  sitk::Image moving = sitk::ReadImage( argv[2], sitk::sitkFloat32 );
```

(continues on next page)

(continued from previous page)

```
  moving = sitk::Normalize( moving );
  moving = sitk::DiscreteGaussian( moving, 2.0);


  sitk::ImageRegistrationMethod R;
  R.SetMetricAsJointHistogramMutualInformation( );

  const double learningRate = 1;
  const unsigned int numberOfIterations = 200;
  const double convergenceMinimumValue = 1e-4;
  const unsigned int convergenceWindowSize=5;
  R.SetOptimizerAsGradientDescentLineSearch ( learningRate,
                                              numberOfIterations,
                                              convergenceMinimumValue,
                                              convergenceWindowSize);

  R.SetInitialTransform( sitk::TranslationTransform( fixed.GetDimension() ) );
  R.SetInterpolator( sitk::sitkLinear );

  IterationUpdate cmd(R);
  R.AddCommand( sitk::sitkIterationEvent, cmd);

  sitk::Transform outTx = R.Execute( fixed, moving );

  std::cout << "-------" << std::endl;
  std::cout << outTx.ToString() << std::endl;
  std::cout << "Optimizer stop condition: " << R.
↪GetOptimizerStopConditionDescription() << std::endl;
  std::cout << " Iteration: " << R.GetOptimizerIteration() << std::endl;
  std::cout << " Metric value: " << R.GetMetricValue() << std::endl;

  sitk::WriteTransform(outTx, argv[3]);


  return 0;
}
```

Python

```
#!/usr/bin/env python

from __future__ import print_function
from functools import reduce


import SimpleITK as sitk
import sys
import os


def command_iteration(method) :
    print("{0:3} = {1:7.5f} : {2}".format(method.GetOptimizerIteration(),
                                          method.GetMetricValue(),
                                          method.GetOptimizerPosition()))
```

(continues on next page)

(continued from previous page)

```
if len ( sys.argv ) < 4:
    print( "Usage: {0} <fixedImageFilter> <movingImageFile>  <outputTransformFile>".
↪format(sys.argv[0]))
    sys.exit ( 1 )

pixelType = sitk.sitkFloat32

fixed = sitk.ReadImage(sys.argv[1], sitk.sitkFloat32)
fixed = sitk.Normalize(fixed)
fixed = sitk.DiscreteGaussian(fixed, 2.0)


moving = sitk.ReadImage(sys.argv[2], sitk.sitkFloat32)
moving = sitk.Normalize(moving)
moving = sitk.DiscreteGaussian(moving, 2.0)


R = sitk.ImageRegistrationMethod()

R.SetMetricAsJointHistogramMutualInformation()

R.SetOptimizerAsGradientDescentLineSearch(learningRate=1.0,
                                          numberOfIterations=200,
                                          convergenceMinimumValue=1e-5,
                                          convergenceWindowSize=5)

R.SetInitialTransform(sitk.TranslationTransform(fixed.GetDimension()))

R.SetInterpolator(sitk.sitkLinear)

R.AddCommand( sitk.sitkIterationEvent, lambda: command_iteration(R) )

outTx = R.Execute(fixed, moving)

print("-------")
print(outTx)
print("Optimizer stop condition: {0}".format(R.
↪GetOptimizerStopConditionDescription()))
print(" Iteration: {0}".format(R.GetOptimizerIteration()))
print(" Metric value: {0}".format(R.GetMetricValue()))


sitk.WriteTransform(outTx,  sys.argv[3])

if ( not "SITK_NOSHOW" in os.environ ):

    resampler = sitk.ResampleImageFilter()
    resampler.SetReferenceImage(fixed);
    resampler.SetInterpolator(sitk.sitkLinear)
    resampler.SetDefaultPixelValue(1)
    resampler.SetTransform(outTx)

    out = resampler.Execute(moving)

    simg1 = sitk.Cast(sitk.RescaleIntensity(fixed), sitk.sitkUInt8)
    simg2 = sitk.Cast(sitk.RescaleIntensity(out), sitk.sitkUInt8)
    cimg = sitk.Compose(simg1, simg2, simg1//2.+simg2//2.)
```

(continues on next page)

(continued from previous page)

```
    sitk.Show( cimg, "ImageRegistration2 Composition" )
```

R

```
# Run with:
#
# Rscript --vanilla ImageRegistrationMethod2.R fixedImageFilter movingImageFile␣
↪outputTransformFile
#

library(SimpleITK)

commandIteration <- function(method)
{
    msg <- paste(method$GetOptimizerIteration(), "=",
                 method$GetMetricValue(), ":",
                 method$GetOptimizerPosition(), "\n" )
    cat(msg)
}

args <- commandArgs( TRUE )

if (length(args) != 3) {
    stop("3 arguments expected - fixedImageFilter, movingImageFile,␣
↪outputTransformFile")
}

pixelType <- 'sitkFloat32'

fixed <- ReadImage(args[[1]], pixelType)
fixed <- Normalize(fixed)
fixed <- DiscreteGaussian(fixed, 2.0)


moving <- ReadImage(args[[2]], pixelType)
moving <- Normalize(moving)
moving <- DiscreteGaussian(moving, 2.0)

R <- ImageRegistrationMethod()

R$SetMetricAsJointHistogramMutualInformation()

R$SetOptimizerAsGradientDescentLineSearch(learningRate=1.0,
                                          numberOfIterations=200,
                                          convergenceMinimumValue=1e-5,
                                          convergenceWindowSize=5)

R$SetInitialTransform(TranslationTransform(fixed$GetDimension()))

R$SetInterpolator('sitkLinear')

R$AddCommand('sitkIterationEvent', function() commandIteration(R))

outTx <- R$Execute(fixed, moving)

cat("-------\n")
outTx
```

(continues on next page)

(continued from previous page)

```
cat("Optimizer stop condition:", R$GetOptimizerStopConditionDescription(), '\n')
cat("Iteration:", R$GetOptimizerIteration(), '\n')
cat("Metric value:", R$GetMetricValue(), '\n')

WriteTransform(outTx, args[[3]])
```

11.13 Image Registration Method 3

If you are not familiar with the SimpleITK registration framework we recommend that you read the *registration overview* before continuing with the example.

11.13.1 Overview

11.13.2 Code

Python

```
#!/usr/bin/env python

from __future__ import print_function
from functools import reduce


import SimpleITK as sitk
import sys
import os


def command_iteration(method) :
    if (method.GetOptimizerIteration()==0):
        print("Estimated Scales: ", method.GetOptimizerScales())
    print("{0:3} = {1:7.5f} : {2}".format(method.GetOptimizerIteration(),
                                          method.GetMetricValue(),
                                          method.GetOptimizerPosition()))



if len ( sys.argv ) < 4:
    print( "Usage: {0} <fixedImageFilter> <movingImageFile>  <outputTransformFile>".
↪format(sys.argv[0]))
    sys.exit ( 1 )

pixelType = sitk.sitkFloat32

fixed = sitk.ReadImage(sys.argv[1], sitk.sitkFloat32)


moving = sitk.ReadImage(sys.argv[2], sitk.sitkFloat32)

R = sitk.ImageRegistrationMethod()

R.SetMetricAsCorrelation()
```

(continues on next page)

(continued from previous page)

```
R.SetOptimizerAsRegularStepGradientDescent(learningRate=2.0,
                                           minStep=1e-4,
                                           numberOfIterations=500,
                                           gradientMagnitudeTolerance=1e-8 )
R.SetOptimizerScalesFromIndexShift()

tx = sitk.CenteredTransformInitializer(fixed, moving, sitk.Similarity2DTransform())
R.SetInitialTransform(tx)

R.SetInterpolator(sitk.sitkLinear)

R.AddCommand( sitk.sitkIterationEvent, lambda: command_iteration(R) )

outTx = R.Execute(fixed, moving)

print("-------")
print(outTx)
print("Optimizer stop condition: {0}".format(R.
↪GetOptimizerStopConditionDescription()))
print(" Iteration: {0}".format(R.GetOptimizerIteration()))
print(" Metric value: {0}".format(R.GetMetricValue()))


sitk.WriteTransform(outTx,  sys.argv[3])

if ( not "SITK_NOSHOW" in os.environ ):

    resampler = sitk.ResampleImageFilter()
    resampler.SetReferenceImage(fixed);
    resampler.SetInterpolator(sitk.sitkLinear)
    resampler.SetDefaultPixelValue(1)
    resampler.SetTransform(outTx)

    out = resampler.Execute(moving)

    simg1 = sitk.Cast(sitk.RescaleIntensity(fixed), sitk.sitkUInt8)
    simg2 = sitk.Cast(sitk.RescaleIntensity(out), sitk.sitkUInt8)
    cimg = sitk.Compose(simg1, simg2, simg1//2.+simg2//2.)
    sitk.Show( cimg, "ImageRegistration2 Composition" )
```

R

```
# Run with:
#
# Rscript --vanilla ImageRegistrationMethod3.R fixedImageFilter movingImageFile␣
↪outputTransformFile
#

library(SimpleITK)

commandIteration <- function(method)
{
    if (method$GetOptimizerIteration()==0) {
        cat("Estimated Scales:", method$GetOptimizerScales())
    }
    msg <- paste(method$GetOptimizerIteration(), "=",
```

(continues on next page)

(continued from previous page)

```
                  method$GetMetricValue(), ":",
                  method$GetOptimizerPosition(), "\n" )
    cat(msg)
}

args <- commandArgs( TRUE )

if (length(args) != 3) {
    stop("3 arguments expected - fixedImageFilter, movingImageFile,␣
↪outputTransformFile")
}

pixelType <- 'sitkFloat32'

fixed <- ReadImage(args[[1]], 'sitkFloat32')

moving <- ReadImage(args[[2]], 'sitkFloat32')

R <- ImageRegistrationMethod()

R$SetMetricAsCorrelation()

R$SetOptimizerAsRegularStepGradientDescent(learningRate=2.0,
                                           minStep=1e-4,
                                           numberOfIterations=500,
                                           relaxationFactor=0.5,
                                           gradientMagnitudeTolerance=1e-8 )
R$SetOptimizerScalesFromIndexShift()

tx <- CenteredTransformInitializer(fixed, moving, Similarity2DTransform())
R$SetInitialTransform(tx)

R$SetInterpolator('sitkLinear')

R$AddCommand( 'sitkIterationEvent', function() commandIteration(R) )

outTx <- R$Execute(fixed, moving)

cat("-------\n")
outTx
cat("Optimizer stop condition:", R$GetOptimizerStopConditionDescription(), '\n')
cat("Iteration:", R$GetOptimizerIteration(), '\n')
cat("Metric value:", R$GetMetricValue(), '\n')


WriteTransform(outTx,  args[[3]])
```

11.14 Image Registration Method 4

If you are not familiar with the SimpleITK registration framework we recommend that you read the *registration overview* before continuing with the example.

11.14.1 Overview

11.14.2 Code

Python

```
#!/usr/bin/env python

from __future__ import print_function

import SimpleITK as sitk
import sys
import os


if len ( sys.argv ) < 4:
    print( "Usage: {0} <fixedImageFilter> <movingImageFile> <outputTransformFile>
↪<numberOfBins> <samplingPercentage>".format(sys.argv[0]))
    sys.exit ( 1 )

def command_iteration(method) :
    print("{0:3} = {1:10.5f} : {2}".format(method.GetOptimizerIteration(),
                                           method.GetMetricValue(),
                                           method.GetOptimizerPosition()))


fixed = sitk.ReadImage(sys.argv[1], sitk.sitkFloat32)
moving = sitk.ReadImage(sys.argv[2], sitk.sitkFloat32)


numberOfBins = 24
samplingPercentage = 0.10

if len ( sys.argv ) > 4:
    numberOfBins = int(sys.argv[4])
if len ( sys.argv ) > 5:
    samplingPercentage = float(sys.argv[5])

R = sitk.ImageRegistrationMethod()
R.SetMetricAsMattesMutualInformation(numberOfBins)
R.SetMetricSamplingPercentage(samplingPercentage,sitk.sitkWallClock)
R.SetMetricSamplingStrategy(R.RANDOM)
R.SetOptimizerAsRegularStepGradientDescent(1.0,.001,200)
R.SetInitialTransform(sitk.TranslationTransform(fixed.GetDimension()))
R.SetInterpolator(sitk.sitkLinear)

R.AddCommand( sitk.sitkIterationEvent, lambda: command_iteration(R) )

outTx = R.Execute(fixed, moving)


print("-------")
print(outTx)
print("Optimizer stop condition: {0}".format(R.
↪GetOptimizerStopConditionDescription()))
print(" Iteration: {0}".format(R.GetOptimizerIteration()))
print(" Metric value: {0}".format(R.GetMetricValue()))
```

(continues on next page)

(continued from previous page)

```
sitk.WriteTransform(outTx,  sys.argv[3])


if ( not "SITK_NOSHOW" in os.environ ):
    resampler = sitk.ResampleImageFilter()
    resampler.SetReferenceImage(fixed);
    resampler.SetInterpolator(sitk.sitkLinear)
    resampler.SetDefaultPixelValue(100)
    resampler.SetTransform(outTx)

    out = resampler.Execute(moving)
    simg1 = sitk.Cast(sitk.RescaleIntensity(fixed), sitk.sitkUInt8)
    simg2 = sitk.Cast(sitk.RescaleIntensity(out), sitk.sitkUInt8)
    cimg = sitk.Compose(simg1, simg2, simg1//2.+simg2//2.)
    sitk.Show( cimg, "ImageRegistration4 Composition" )
```

R

```
# Run with:
#
# Rscript --vanilla ImageRegistrationMethod4.R fixedImageFilter movingImageFile␣
↪outputTransformFile numberOfBins samplingPercentage
#

library(SimpleITK)

commandIteration <- function(method)
{
    msg <- paste(method$GetOptimizerIteration(), "=",
                 method$GetMetricValue(), ":",
                 method$GetOptimizerPosition(), "\n" )
    cat(msg)
}

args <- commandArgs( TRUE )

if (length(args) < 3) {
    stop("3, 4, or 5 arguments expected - fixedImageFilter, movingImageFile,␣
↪outputTransformFile [numberOfBins] [samplingPercentage]")
}

fixed <- ReadImage(args[[1]], 'sitkFloat32')
moving <- ReadImage(args[[2]], 'sitkFloat32')

numberOfBins <- 24
samplingPercentage <- 0.10

if (length(args) > 4) {
    numberOfBins <- strtoi(args[[4]])
}
if (length(args) > 5) {
    samplingPercentage <- as.numeric(args[[5]])
}

R <- ImageRegistrationMethod()
R$SetMetricAsMattesMutualInformation(numberOfBins)
```

(continues on next page)

(continued from previous page)

```
R$SetMetricSamplingPercentage(samplingPercentage)
R$SetMetricSamplingStrategy('RANDOM')
R$SetOptimizerAsRegularStepGradientDescent(1.0,.001,200)
R$SetInitialTransform(TranslationTransform(fixed$GetDimension()))
R$SetInterpolator('sitkLinear')

R$AddCommand( 'sitkIterationEvent', function() commandIteration(R) )

outTx <- R$Execute(fixed, moving)

cat("-------\n")
outTx
cat("Optimizer stop condition:", R$GetOptimizerStopConditionDescription(), '\n')
cat("Iteration:", R$GetOptimizerIteration(), '\n')
cat("Metric value:", R$GetMetricValue(), '\n')

WriteTransform(outTx, args[[3]])
```

11.15 Image Registration Method BSpline 1

If you are not familiar with the SimpleITK registration framework we recommend that you read the *registration overview* before continuing with the example.

11.15.1 Overview

11.15.2 Code

C++

```
// This one header will include all SimpleITK filters and external
// objects.
#include <SimpleITK.h>
#include <iostream>
#include <stdlib.h>
#include <iomanip>

namespace sitk = itk::simple;

// use sitk's output operator for std::vector etc..
using sitk::operator<<;




class IterationUpdate
  : public sitk::Command
{
public:
  IterationUpdate( const sitk::ImageRegistrationMethod &m)
    : m_Method(m)
    {}
```

(continues on next page)

(continued from previous page)

```
  virtual void Execute( )
    {
      // use sitk's output operator for std::vector etc..
      using sitk::operator<<;

      if (m_Method.GetOptimizerIteration() == 0)
        {
        std::cout << m_Method.ToString() << std::endl;
        }

      // stash the stream state
      std::ios  state(NULL);
      state.copyfmt(std::cout);
      std::cout << std::fixed << std::setfill(' ') << std::setprecision( 5 );
      std::cout << std::setw(3) << m_Method.GetOptimizerIteration();
      std::cout << " = " << std::setw(10) << m_Method.GetMetricValue() << std::endl;
      std::cout.copyfmt(state);
    }

private:
  const sitk::ImageRegistrationMethod &m_Method;

};




int main(int argc, char *argv[])
{

  if ( argc < 4 )
    {
    std::cerr << "Usage: " << argv[0] << " <fixedImageFilter> <movingImageFile>
↪<outputTransformFile>" << std::endl;
    return 1;
    }

  sitk::Image fixed = sitk::ReadImage( argv[1], sitk::sitkFloat32 );

  sitk::Image moving = sitk::ReadImage( argv[2], sitk::sitkFloat32 );

  std::vector<unsigned int> transformDomainMeshSize(fixed.GetDimension(),8);

  sitk::BSplineTransform tx = sitk::BSplineTransformInitializer(fixed,␣
↪transformDomainMeshSize);

  std::cout << "Initial Parameters:" << tx.GetParameters() << std::endl;


  sitk::ImageRegistrationMethod R;
  R.SetMetricAsCorrelation();

  const double gradientConvergenceTolerance = 1e-5;
  const unsigned int maximumNumberOfIterations = 100;
  const unsigned int maximumNumberOfCorrections = 5;
  const unsigned int maximumNumberOfFunctionEvaluations = 1000;
  const double costFunctionConvergenceFactor = 1e+7;
```

(continues on next page)

(continued from previous page)

```
  R.SetOptimizerAsLBFGSB(gradientConvergenceTolerance,
                         maximumNumberOfIterations,
                         maximumNumberOfCorrections,
                         maximumNumberOfFunctionEvaluations,
                         costFunctionConvergenceFactor);
  R.SetInitialTransform(tx, true);
  R.SetInterpolator(sitk::sitkLinear);

  IterationUpdate cmd(R);
  R.AddCommand( sitk::sitkIterationEvent, cmd);

  sitk::Transform outTx = R.Execute( fixed, moving );

  std::cout << "-------" << std::endl;
  std::cout << outTx.ToString() << std::endl;
  std::cout << "Optimizer stop condition: " << R.
↪GetOptimizerStopConditionDescription() << std::endl;
  std::cout << " Iteration: " << R.GetOptimizerIteration() << std::endl;
  std::cout << " Metric value: " << R.GetMetricValue() << std::endl;

  sitk::WriteTransform(outTx, argv[3]);

  return 0;
}
```

Python

```
#!/usr/bin/env python

from __future__ import print_function

import SimpleITK as sitk
import sys
import os


def command_iteration(method) :
    print("{0:3} = {1:10.5f}".format(method.GetOptimizerIteration(),
                                     method.GetMetricValue()))

if len ( sys.argv ) < 4:
    print( "Usage: {0} <fixedImageFilter> <movingImageFile> <outputTransformFile>".
↪format(sys.argv[0]))
    sys.exit ( 1 )


fixed = sitk.ReadImage(sys.argv[1], sitk.sitkFloat32)

moving = sitk.ReadImage(sys.argv[2], sitk.sitkFloat32)

transformDomainMeshSize=[8]*moving.GetDimension()
tx = sitk.BSplineTransformInitializer(fixed,
                                      transformDomainMeshSize )

print("Initial Parameters:");
print(tx.GetParameters())
```

(continues on next page)

(continued from previous page)

```
R = sitk.ImageRegistrationMethod()
R.SetMetricAsCorrelation()

R.SetOptimizerAsLBFGSB(gradientConvergenceTolerance=1e-5,
                       numberOfIterations=100,
                       maximumNumberOfCorrections=5,
                       maximumNumberOfFunctionEvaluations=1000,
                       costFunctionConvergenceFactor=1e+7)
R.SetInitialTransform(tx, True)
R.SetInterpolator(sitk.sitkLinear)

R.AddCommand( sitk.sitkIterationEvent, lambda: command_iteration(R) )

outTx = R.Execute(fixed, moving)

print("-------")
print(outTx)
print("Optimizer stop condition: {0}".format(R.
↪GetOptimizerStopConditionDescription()))
print(" Iteration: {0}".format(R.GetOptimizerIteration()))
print(" Metric value: {0}".format(R.GetMetricValue()))

sitk.WriteTransform(outTx,  sys.argv[3])

if ( not "SITK_NOSHOW" in os.environ ):

    resampler = sitk.ResampleImageFilter()
    resampler.SetReferenceImage(fixed);
    resampler.SetInterpolator(sitk.sitkLinear)
    resampler.SetDefaultPixelValue(100)
    resampler.SetTransform(outTx)

    out = resampler.Execute(moving)
    simg1 = sitk.Cast(sitk.RescaleIntensity(fixed), sitk.sitkUInt8)
    simg2 = sitk.Cast(sitk.RescaleIntensity(out), sitk.sitkUInt8)
    cimg = sitk.Compose(simg1, simg2, simg1//2.+simg2//2.)
    sitk.Show( cimg, "ImageRegistration1 Composition" )
```

R

```
# Run with:
#
# Rscript --vanilla ImageRegistrationMethodBSpline1.R fixedImageFilter␣
↪movingImageFile outputTransformFile
#

library(SimpleITK)

commandIteration <- function(method)
{
    msg <- paste(method$GetOptimizerIteration(), "=",
                 method$GetMetricValue(), "\n" )
    cat(msg)
}

args <- commandArgs( TRUE )
```

(continues on next page)

(continued from previous page)

```
if (length(args) != 3) {
    stop("3 arguments expected - fixedImageFilter, movingImageFile,␣
↪outputTransformFile")
}

fixed <- ReadImage(args[[1]], 'sitkFloat32')
moving <- ReadImage(args[[2]], 'sitkFloat32')

transformDomainMeshSize <- rep(8, moving$GetDimension())
tx <- BSplineTransformInitializer(fixed, transformDomainMeshSize)

cat("Initial Parameters:\n", tx$GetParameters())

R <- ImageRegistrationMethod()
R$SetMetricAsCorrelation()

R$SetOptimizerAsLBFGSB(gradientConvergenceTolerance=1e-5,
                       numberOfIterations=100,
                       maximumNumberOfCorrections=5,
                       maximumNumberOfFunctionEvaluations=1000,
                       costFunctionConvergenceFactor=1e+7)
R$SetInitialTransform(tx, TRUE)
R$SetInterpolator('sitkLinear')

R$AddCommand( 'sitkIterationEvent', function() commandIteration(R) )

outTx <- R$Execute(fixed, moving)

cat("-------\n")
outTx
cat("Optimizer stop condition:", R$GetOptimizerStopConditionDescription(), '\n')
cat(" Iteration:", R$GetOptimizerIteration(), '\n')
cat(" Metric value:", R$GetMetricValue(), '\n')

WriteTransform(outTx, args[[3]])
```

11.16 Image Registration Method BSpline 2

If you are not familiar with the SimpleITK registration framework we recommend that you read the *registration overview* before continuing with the example.

11.16.1 Overview

11.16.2 Code

Python

```
#!/usr/bin/env python

from __future__ import print_function

import SimpleITK as sitk
```

(continues on next page)

(continued from previous page)

```
import sys
import os


def command_iteration(method) :
    print("{0:3} = {1:10.5f}".format(method.GetOptimizerIteration(),
                                    method.GetMetricValue()))
    print("\t#: ", len(method.GetOptimizerPosition()))


def command_multi_iteration(method) :
    print("--------- Resolution Changing ---------")


if len ( sys.argv ) < 4:
    print( "Usage: {0} <fixedImageFilter> <movingImageFile> <outputTransformFile>".
↪format(sys.argv[0]))
    sys.exit ( 1 )


fixed = sitk.ReadImage(sys.argv[1], sitk.sitkFloat32)

moving = sitk.ReadImage(sys.argv[2], sitk.sitkFloat32)

transformDomainMeshSize=[10]*moving.GetDimension()
tx = sitk.BSplineTransformInitializer(fixed,
                                      transformDomainMeshSize )

print("Initial Parameters:");
print(tx.GetParameters())

R = sitk.ImageRegistrationMethod()
R.SetMetricAsMattesMutualInformation(50)
R.SetOptimizerAsGradientDescentLineSearch(5.0, 100,
                                          convergenceMinimumValue=1e-4,
                                          convergenceWindowSize=5)
R.SetOptimizerScalesFromPhysicalShift( )
R.SetInitialTransform(tx)
R.SetInterpolator(sitk.sitkLinear)

R.SetShrinkFactorsPerLevel([6,2,1])
R.SetSmoothingSigmasPerLevel([6,2,1])

R.AddCommand( sitk.sitkIterationEvent, lambda: command_iteration(R) )
R.AddCommand( sitk.sitkMultiResolutionIterationEvent, lambda: command_multi_
↪iteration(R) )

outTx = R.Execute(fixed, moving)

print("-------")
print(outTx)
print("Optimizer stop condition: {0}".format(R.
↪GetOptimizerStopConditionDescription()))
print(" Iteration: {0}".format(R.GetOptimizerIteration()))
print(" Metric value: {0}".format(R.GetMetricValue()))

sitk.WriteTransform(outTx,  sys.argv[3])
```

(continues on next page)

(continued from previous page)

```
if ( not "SITK_NOSHOW" in os.environ ):

    resampler = sitk.ResampleImageFilter()
    resampler.SetReferenceImage(fixed);
    resampler.SetInterpolator(sitk.sitkLinear)
    resampler.SetDefaultPixelValue(100)
    resampler.SetTransform(outTx)

    out = resampler.Execute(moving)
    simg1 = sitk.Cast(sitk.RescaleIntensity(fixed), sitk.sitkUInt8)
    simg2 = sitk.Cast(sitk.RescaleIntensity(out), sitk.sitkUInt8)
    cimg = sitk.Compose(simg1, simg2, simg1//2.+simg2//2.)
    sitk.Show( cimg, "ImageRegistration1 Composition" )
```

R

```
# Run with:
#
# Rscript --vanilla ImageRegistrationMethodBSpline2.R fixedImageFilter␣
↪movingImageFile outputTransformFile
#

library(SimpleITK)

commandIteration <- function(method)
{
    msg <- paste(method$GetOptimizerIteration(), "=",
                 method$GetMetricValue(), "\n\t#:",
                 method$GetOptimizerPosition(), '\n')
    cat(msg)
}

commandMultiIteration <- function(method)
{
    msg <- paste("--------- Resolution Changing ---------\n")
    cat(msg)
}

args <- commandArgs( TRUE )

if (length(args) != 3) {
    stop("3 arguments expected - fixedImageFilter, movingImageFile,␣
↪outputTransformFile")
}

fixed <- ReadImage(args[[1]], 'sitkFloat32')
moving <- ReadImage(args[[2]], 'sitkFloat32')

transformDomainMeshSize <- rep(10, moving$GetDimension())
tx <- BSplineTransformInitializer(fixed, transformDomainMeshSize)

cat("Initial Parameters:\n", tx$GetParameters())

R <- ImageRegistrationMethod()
R$SetMetricAsMattesMutualInformation(50)
R$SetOptimizerAsGradientDescentLineSearch(5.0, 100,
```

(continues on next page)

(continued from previous page)

```
                                        convergenceMinimumValue=1e-4,
                                        convergenceWindowSize=5)
R$SetOptimizerScalesFromPhysicalShift()
R$SetInitialTransform(tx)
R$SetInterpolator('sitkLinear')

R$SetShrinkFactorsPerLevel(c(6,2,1))
R$SetSmoothingSigmasPerLevel(c(6,2,1))

R$AddCommand( 'sitkIterationEvent', function() commandIteration(R) )
R$AddCommand( 'sitkMultiResolutionIterationEvent', function()␣
↪commandMultiIteration(R) )

outTx <- R$Execute(fixed, moving)

cat("-------\n")
outTx
cat("Optimizer stop condition:", R$GetOptimizerStopConditionDescription(), '\n')
cat(" Iteration:", R$GetOptimizerIteration(), '\n')
cat(" Metric value:", R$GetMetricValue(), '\n')

WriteTransform(outTx, args[[3]])
```

11.17 Image Registration Method BSpline 3

If you are not familiar with the SimpleITK registration framework we recommend that you read the *registration overview* before continuing with the example.

11.17.1 Overview

This example performs registration of multi-modality images with a multi-resolution BSpline approach.

A BSplineTransform usually has a large number of parameters which increases the complexity and duration of optimizing the deformation. The multi-resolution BSpline approach initially performs the registration at a lower resolution with fewer parameters at the first level and then adapts or resamples the BSpline control points to a higher resolution at subsequent levels. This transformation adaption is done concurrently with the ImageRegistrationMethod's multi-level feature. This enables the setting of the shrink factor, smoothing sigmas, sampling percentage and BSpline resolution to vary per level to efficiently solve a diverse set of registration problems.

The initial transform is the low resolution BSpline. The scaling factor of each BSpline used per level is specified with the `ImageRegistration::SetInitialBSpline` method's third argument as an integer array. The first value is usually 1 and it is reasonable to double the resolution at each registration level. For this example the last resolution is 5, so that the result is comparable to the transformation from the *previous* example.

It can be important to monitor and observe the transform at each level or iteration. When the "inplace" option is enabled, the transform passed as the initial transform will be up to date during the registration process which enables it to be used in *event commands*.

See also: *Image Registration Method BSpline 1*, *Image Registration Method BSpline 2*.

11.17.2 Code

C++

```
// This one header will include all SimpleITK filters and external
// objects.
#include <SimpleITK.h>
#include <iostream>
#include <stdlib.h>
#include <iomanip>

namespace sitk = itk::simple;

// use sitk's output operator for std::vector etc..
using sitk::operator<<;


class IterationUpdate
  : public sitk::Command
{
public:
  IterationUpdate( const sitk::ImageRegistrationMethod &m, const␣
↪sitk::BSplineTransform &tx)
    : m_Method(m),
      m_BSplineTransform(tx)
    {}

  // Override method from parent which is called at for the requested event
  virtual void Execute( )
    {
      if (m_Method.GetOptimizerIteration() == 0)
        {
        // The BSpline is resized before the first optimizer
        // iteration is completed per level. Print the transform object
        // to show the adapted BSpline transform.
        std::cout << m_BSplineTransform.ToString() << std::endl;
        }

      // stash the stream state
      std::ios  state(NULL);
      state.copyfmt(std::cout);
      std::cout << std::fixed << std::setfill(' ') << std::setprecision( 5 );
      std::cout << std::setw(3) << m_Method.GetOptimizerIteration();
      std::cout << " = " << std::setw(10) << m_Method.GetMetricValue() << std::endl;
      std::cout.copyfmt(state);
    }

private:
  const sitk::ImageRegistrationMethod &m_Method;
  const sitk::BSplineTransform &m_BSplineTransform;

};


class MultiResolutionIterationUpdate
  : public sitk::Command
{
public:
  MultiResolutionIterationUpdate( const sitk::ImageRegistrationMethod &m)
    : m_Method(m)
    {}
```

(continues on next page)

(continued from previous page)

```
  // Override method from parent which is called at for the requested event
  virtual void Execute( )
    {
      // The sitkMultiResolutionIterationEvent occurs before the
      // resolution of the transform. This event is used here to print
      // the status of the optimizer from the previous registration level.
      if (m_Method.GetCurrentLevel() > 0)
        {
        std::cout << "Optimizer stop condition: " << m_Method.
↪GetOptimizerStopConditionDescription() << std::endl;
        std::cout << " Iteration: " << m_Method.GetOptimizerIteration() << std::endl;
        std::cout << " Metric value: " << m_Method.GetMetricValue() << std::endl;
        }

      std::cout << "--------- Resolution Changing ---------" << std::endl;
    }

private:
  const sitk::ImageRegistrationMethod &m_Method;
};




int main(int argc, char *argv[])
{

  if ( argc < 4 )
    {
    std::cerr << "Usage: " << argv[0] << " <fixedImageFilter> <movingImageFile>
↪<outputTransformFile>" << std::endl;
    return 1;
    }

  sitk::Image fixed = sitk::ReadImage( argv[1], sitk::sitkFloat32 );

  sitk::Image moving = sitk::ReadImage( argv[2], sitk::sitkFloat32 );

  std::vector<unsigned int> transformDomainMeshSize(fixed.GetDimension(), 2);

  sitk::BSplineTransform tx = sitk::BSplineTransformInitializer(fixed,␣
↪transformDomainMeshSize);

  std::cout << "Initial Number of Parameters:" << tx.GetNumberOfParameters() <<␣
↪std::endl;


  sitk::ImageRegistrationMethod R;
  R.SetMetricAsJointHistogramMutualInformation();


  const double learningRate = 5.0;
  const unsigned int numberOfIterations = 100u;
  const double convergenceMinimumValue = 1e-4;
  const unsigned int convergenceWindowSize = 5;
  R.SetOptimizerAsGradientDescentLineSearch( learningRate,
```

(continues on next page)

(continued from previous page)

```
                                          numberOfIterations,
                                          convergenceMinimumValue,
                                          convergenceWindowSize);

  R.SetInterpolator(sitk::sitkLinear);

  const unsigned int numberOfLevels = 3;
  std::vector<unsigned int> scaleFactors(numberOfLevels);
  scaleFactors[0] = 1;
  scaleFactors[1] = 2;
  scaleFactors[2] = 5;
  const bool inPlace = true;
  R.SetInitialTransformAsBSpline(tx,
                                 inPlace,
                                 scaleFactors);

  std::vector<unsigned int> shrinkFactors( numberOfLevels );
  shrinkFactors[0] = 4;
  shrinkFactors[1] = 2;
  shrinkFactors[2] = 1;
  R.SetShrinkFactorsPerLevel( shrinkFactors );

  std::vector<double> smoothingSigmas( numberOfLevels );
  smoothingSigmas[0] = 4.0;
  smoothingSigmas[1] = 2.0;
  smoothingSigmas[2] = 1.0;
  R.SetSmoothingSigmasPerLevel( smoothingSigmas );

  IterationUpdate cmd1(R, tx);
  R.AddCommand( sitk::sitkIterationEvent, cmd1);

  MultiResolutionIterationUpdate cmd2(R);
  R.AddCommand( sitk::sitkMultiResolutionIterationEvent, cmd2);

  sitk::Transform outTx = R.Execute( fixed, moving );

  std::cout << "-------" << std::endl;
  std::cout << outTx.ToString() << std::endl;
  std::cout << "Optimizer stop condition: " << R.
↪GetOptimizerStopConditionDescription() << std::endl;
  std::cout << " Iteration: " << R.GetOptimizerIteration() << std::endl;
  std::cout << " Metric value: " << R.GetMetricValue() << std::endl;

  sitk::WriteTransform(outTx, argv[3]);

  return 0;
}
```

Python

```
#!/usr/bin/env python

from __future__ import print_function

import SimpleITK as sitk
import sys
import os
```

(continues on next page)

(continued from previous page)

```
def command_iteration(method, bspline_transform) :
    if method.GetOptimizerIteration() == 0:
        # The BSpline is resized before the first optimizer
        # iteration is completed per level. Print the transform object
        # to show the adapted BSpline transform.
        print(bspline_transform)


    print("{0:3} = {1:10.5f}".format(method.GetOptimizerIteration(),
                                     method.GetMetricValue()))



def command_multi_iteration(method) :
    # The sitkMultiResolutionIterationEvent occurs before the
    # resolution of the transform. This event is used here to print
    # the status of the optimizer from the previous registration level.
    if R.GetCurrentLevel() > 0:
        print("Optimizer stop condition: {0}".format(R.
↪GetOptimizerStopConditionDescription()))
        print(" Iteration: {0}".format(R.GetOptimizerIteration()))
        print(" Metric value: {0}".format(R.GetMetricValue()))

    print("--------- Resolution Changing ---------")


if len ( sys.argv ) < 4:
    print( "Usage: {0} <fixedImageFilter> <movingImageFile> <outputTransformFile>".
↪format(sys.argv[0]))
    sys.exit ( 1 )


fixed = sitk.ReadImage(sys.argv[1], sitk.sitkFloat32)

moving = sitk.ReadImage(sys.argv[2], sitk.sitkFloat32)

transformDomainMeshSize=[2]*fixed.GetDimension()
tx = sitk.BSplineTransformInitializer(fixed,
                                      transformDomainMeshSize )

print("Initial Number of Parameters: {0}".format(tx.GetNumberOfParameters()))

R = sitk.ImageRegistrationMethod()
R.SetMetricAsJointHistogramMutualInformation()

R.SetOptimizerAsGradientDescentLineSearch(5.0,
                                          100,
                                          convergenceMinimumValue=1e-4,
                                          convergenceWindowSize=5)

R.SetInterpolator(sitk.sitkLinear)

R.SetInitialTransformAsBSpline(tx,
                               inPlace=True,
                               scaleFactors=[1,2,5])
```

(continues on next page)

(continued from previous page)

```
R.SetShrinkFactorsPerLevel([4,2,1])
R.SetSmoothingSigmasPerLevel([4,2,1])

R.AddCommand( sitk.sitkIterationEvent, lambda: command_iteration(R, tx) )
R.AddCommand( sitk.sitkMultiResolutionIterationEvent, lambda: command_multi_
↪iteration(R) )

outTx = R.Execute(fixed, moving)

print("-------")
print(tx)
print(outTx)
print("Optimizer stop condition: {0}".format(R.
↪GetOptimizerStopConditionDescription()))
print(" Iteration: {0}".format(R.GetOptimizerIteration()))
print(" Metric value: {0}".format(R.GetMetricValue()))

sitk.WriteTransform(outTx,  sys.argv[3])

if ( not "SITK_NOSHOW" in os.environ ):

    resampler = sitk.ResampleImageFilter()
    resampler.SetReferenceImage(fixed);
    resampler.SetInterpolator(sitk.sitkLinear)
    resampler.SetDefaultPixelValue(100)
    resampler.SetTransform(outTx)

    out = resampler.Execute(moving)
    simg1 = sitk.Cast(sitk.RescaleIntensity(fixed), sitk.sitkUInt8)
    simg2 = sitk.Cast(sitk.RescaleIntensity(out), sitk.sitkUInt8)
    cimg = sitk.Compose(simg1, simg2, simg1//2.+simg2//2.)
    sitk.Show( cimg, "Image Registration Composition" )
```

11.18 Image Registration Method Displacement 1

11.18.1 Overview

11.18.2 Code

C++

```
// This one header will include all SimpleITK filters and external
// objects.
#include <SimpleITK.h>
#include <iostream>
#include <stdlib.h>
#include <iomanip>

namespace sitk = itk::simple;



class IterationUpdate
```

(continues on next page)

(continued from previous page)

```
  : public sitk::Command
{
public:
  IterationUpdate( const sitk::ImageRegistrationMethod &m)
    : m_Method(m)
    {}

  virtual void Execute( )
    {
      // use sitk's output operator for std::vector etc..
      using sitk::operator<<;

      // stash the stream state
      std::ios  state(NULL);
      state.copyfmt(std::cout);
      std::cout << std::fixed << std::setfill(' ') << std::setprecision( 5 );
      if ( m_Method.GetOptimizerIteration() == 0 )
        {
        std::cout << "\tLevel: " << std::setw(3) << m_Method.GetCurrentLevel() <<␣
↪std::endl;
        std::cout << "\tScales: " << m_Method.GetOptimizerScales() << std::endl;
        }
      std::cout << '#' << m_Method.GetOptimizerIteration() << std::endl;
      std::cout << "\tMetric Value: " <<  m_Method.GetMetricValue() << std::endl;
      std::cout << "\tLearning Rate: " << m_Method.GetOptimizerLearningRate() <<␣
↪std::endl;
      if (m_Method.GetOptimizerConvergenceValue() != std::numeric_limits<double>
↪::max())
        {
        std::cout << "\tConvergence Value: " << std::scientific << m_Method.
↪GetOptimizerConvergenceValue() << std::endl;
        }
      std::cout.copyfmt(state);
    }

private:
  const sitk::ImageRegistrationMethod &m_Method;

};

class MultiResolutionIterationUpdate
  : public sitk::Command
{
public:
  MultiResolutionIterationUpdate( const sitk::ImageRegistrationMethod &m)
    : m_Method(m)
    {}

  virtual void Execute( )
    {
      // use sitk's output operator for std::vector etc..
      using sitk::operator<<;

      // stash the stream state
      std::ios  state(NULL);
      state.copyfmt(std::cout);
      std::cout << std::fixed << std::setfill(' ') << std::setprecision( 5 );
```

(continues on next page)

(continued from previous page)

```
      std::cout << "\tStop Condition: " << m_Method.
↪GetOptimizerStopConditionDescription() << std::endl;
      std::cout << "============= Resolution Change =============" << std::endl;
      std::cout.copyfmt(state);
    }

private:
  const sitk::ImageRegistrationMethod &m_Method;

};


int main(int argc, char *argv[])
{

  if ( argc < 4 )
    {
    std::cerr << "Usage: " << argv[0] << " <fixedImageFilter> <movingImageFile>
↪<outputTransformFile>" << std::endl;
    return 1;
    }

  sitk::Image fixed = sitk::ReadImage( argv[1], sitk::sitkFloat32 );

  sitk::Image moving = sitk::ReadImage( argv[2], sitk::sitkFloat32 );

  sitk::Transform initialTx = sitk::CenteredTransformInitializer(fixed, moving,␣
↪sitk::AffineTransform(fixed.GetDimension()));

  sitk::ImageRegistrationMethod R;

  {
  std::vector<unsigned int> shrinkFactors;
  shrinkFactors.push_back(3);
  shrinkFactors.push_back(2);
  shrinkFactors.push_back(1);

  std::vector<double> smoothingSigmas;
  smoothingSigmas.push_back(2.0);
  smoothingSigmas.push_back(1.0);
  smoothingSigmas.push_back(1.0);

  R.SetShrinkFactorsPerLevel(shrinkFactors);
  R.SetSmoothingSigmasPerLevel(smoothingSigmas);
  }

  R.SetMetricAsJointHistogramMutualInformation(20);
  R.MetricUseFixedImageGradientFilterOff();
  R.MetricUseFixedImageGradientFilterOff();

  {
  double learningRate=1.0;
  unsigned int numberOfIterations=100;
  double convergenceMinimumValue = 1e-6;
  unsigned int convergenceWindowSize = 10;
  sitk::ImageRegistrationMethod::EstimateLearningRateType estimateLearningRate = R.
↪EachIteration;
```

(continues on next page)

(continued from previous page)

```
  R.SetOptimizerAsGradientDescent( learningRate,
                                   numberOfIterations,
                                   convergenceMinimumValue,
                                   convergenceWindowSize,
                                   estimateLearningRate
    );
  }
  R.SetOptimizerScalesFromPhysicalShift();

  R.SetInitialTransform(initialTx, true);

  R.SetInterpolator(sitk::sitkLinear);

  IterationUpdate cmd(R);
  R.AddCommand( sitk::sitkIterationEvent, cmd);

  MultiResolutionIterationUpdate cmd2(R);
  R.AddCommand( sitk::sitkMultiResolutionIterationEvent, cmd2);

  sitk::Transform outTx = R.Execute( fixed, moving );

  std::cout << "-------" << std::endl;
  std::cout << outTx.ToString() << std::endl;
  std::cout << "Optimizer stop condition: " << R.
↪GetOptimizerStopConditionDescription() << std::endl;
  std::cout << " Iteration: " << R.GetOptimizerIteration() << std::endl;
  std::cout << " Metric value: " << R.GetMetricValue() << std::endl;


  sitk::Image displacementField = sitk::Image(fixed.GetSize(),␣
↪sitk::sitkVectorFloat64);
  displacementField.CopyInformation(fixed);
  sitk::DisplacementFieldTransform displacementTx(displacementField);
  const double varianceForUpdateField=0.0;
  const double varianceForTotalField=1.5;
  displacementTx.SetSmoothingGaussianOnUpdate(varianceForUpdateField,
                                              varianceForTotalField);



  R.SetMovingInitialTransform(outTx);
  R.SetInitialTransform(displacementTx, true);

  R.SetMetricAsANTSNeighborhoodCorrelation(4);
  R.MetricUseFixedImageGradientFilterOff();
  R.MetricUseFixedImageGradientFilterOff();

  {
  std::vector<unsigned int> shrinkFactors;
  shrinkFactors.push_back(3);
  shrinkFactors.push_back(2);
  shrinkFactors.push_back(1);

  std::vector<double> smoothingSigmas;
  smoothingSigmas.push_back(2.0);
  smoothingSigmas.push_back(1.0);
  smoothingSigmas.push_back(1.0);
```

(continues on next page)

(continued from previous page)

```
  R.SetShrinkFactorsPerLevel(shrinkFactors);
  R.SetSmoothingSigmasPerLevel(smoothingSigmas);
  }

  R.SetOptimizerScalesFromPhysicalShift();

  {
  double learningRate=1.0;
  unsigned int numberOfIterations=300;
  double convergenceMinimumValue = 1e-6;
  unsigned int convergenceWindowSize = 10;
  sitk::ImageRegistrationMethod::EstimateLearningRateType estimateLearningRate = R.
↪EachIteration;
  R.SetOptimizerAsGradientDescent( learningRate,
                                   numberOfIterations,
                                   convergenceMinimumValue,
                                   convergenceWindowSize,
                                   estimateLearningRate
    );
  }
  outTx.AddTransform( R.Execute(fixed, moving) );

  std::cout << "-------" << std::endl;
  std::cout << outTx.ToString() << std::endl;
  std::cout << "Optimizer stop condition: " << R.
↪GetOptimizerStopConditionDescription() << std::endl;
  std::cout << " Iteration: " << R.GetOptimizerIteration() << std::endl;
  std::cout << " Metric value: " << R.GetMetricValue() << std::endl;


  sitk::WriteTransform(outTx, argv[3]);

  return 0;
}
```

Python

```
#!/usr/bin/env python

from __future__ import print_function

import SimpleITK as sitk
import sys
import os


def command_iteration(method) :
    if (method.GetOptimizerIteration() == 0):
        print("\tLevel: {0}".format(method.GetCurrentLevel()))
        print("\tScales: {0}".format(method.GetOptimizerScales()))
    print("#{0}".format(method.GetOptimizerIteration()))
    print("\tMetric Value: {0:10.5f}".format( method.GetMetricValue()))
    print("\tLearningRate: {0:10.5f}".format(method.GetOptimizerLearningRate()))
    if (method.GetOptimizerConvergenceValue() != sys.float_info.max):
        print("\tConvergence Value: {0:.5e}".format(method.
↪GetOptimizerConvergenceValue()))
```

(continues on next page)

(continued from previous page)

```
def command_multiresolution_iteration(method):
    print("\tStop Condition: {0}".format(method.
↪GetOptimizerStopConditionDescription()))
    print("============= Resolution Change =============")


if len ( sys.argv ) < 4:
    print( "Usage: {0} <fixedImageFilter> <movingImageFile> <outputTransformFile>".
↪format(sys.argv[0]))
    sys.exit ( 1 )


fixed = sitk.ReadImage(sys.argv[1], sitk.sitkFloat32)

moving = sitk.ReadImage(sys.argv[2], sitk.sitkFloat32)

initialTx = sitk.CenteredTransformInitializer(fixed, moving, sitk.
↪AffineTransform(fixed.GetDimension()))

R = sitk.ImageRegistrationMethod()

R.SetShrinkFactorsPerLevel([3,2,1])
R.SetSmoothingSigmasPerLevel([2,1,1])

R.SetMetricAsJointHistogramMutualInformation(20)
R.MetricUseFixedImageGradientFilterOff()

R.SetOptimizerAsGradientDescent(learningRate=1.0,
                                numberOfIterations=100,
                                estimateLearningRate = R.EachIteration)
R.SetOptimizerScalesFromPhysicalShift()

R.SetInitialTransform(initialTx,inPlace=True)

R.SetInterpolator(sitk.sitkLinear)

R.AddCommand( sitk.sitkIterationEvent, lambda: command_iteration(R) )
R.AddCommand( sitk.sitkMultiResolutionIterationEvent, lambda: command_multiresolution_
↪iteration(R) )

outTx = R.Execute(fixed, moving)


print("-------")
print(outTx)
print("Optimizer stop condition: {0}".format(R.
↪GetOptimizerStopConditionDescription()))
print(" Iteration: {0}".format(R.GetOptimizerIteration()))
print(" Metric value: {0}".format(R.GetMetricValue()))


displacementField = sitk.Image(fixed.GetSize(), sitk.sitkVectorFloat64)
displacementField.CopyInformation(fixed)
displacementTx = sitk.DisplacementFieldTransform(displacementField)
del displacementField
```

(continues on next page)

(continued from previous page)

```
displacementTx.SetSmoothingGaussianOnUpdate(varianceForUpdateField=0.0,
                                            varianceForTotalField=1.5)



R.SetMovingInitialTransform(outTx)
R.SetInitialTransform(displacementTx, inPlace=True)

R.SetMetricAsANTSNeighborhoodCorrelation(4)
R.MetricUseFixedImageGradientFilterOff()

R.SetShrinkFactorsPerLevel([3,2,1])
R.SetSmoothingSigmasPerLevel([2,1,1])

R.SetOptimizerScalesFromPhysicalShift()
R.SetOptimizerAsGradientDescent(learningRate=1,
                                numberOfIterations=300,
                                estimateLearningRate=R.EachIteration)

outTx.AddTransform( R.Execute(fixed, moving) )


print("-------")
print(outTx)
print("Optimizer stop condition: {0}".format(R.
↪GetOptimizerStopConditionDescription()))
print(" Iteration: {0}".format(R.GetOptimizerIteration()))
print(" Metric value: {0}".format(R.GetMetricValue()))


sitk.WriteTransform(outTx,  sys.argv[3])

if ( not "SITK_NOSHOW" in os.environ ):

    sitk.Show(displacementTx.GetDisplacementField(), "Displacement Field")

    resampler = sitk.ResampleImageFilter()
    resampler.SetReferenceImage(fixed);
    resampler.SetInterpolator(sitk.sitkLinear)
    resampler.SetDefaultPixelValue(100)
    resampler.SetTransform(outTx)

    out = resampler.Execute(moving)
    simg1 = sitk.Cast(sitk.RescaleIntensity(fixed), sitk.sitkUInt8)
    simg2 = sitk.Cast(sitk.RescaleIntensity(out), sitk.sitkUInt8)
    cimg = sitk.Compose(simg1, simg2, simg1//2.+simg2//2.)
    sitk.Show( cimg, "ImageRegistration1 Composition" )
```

R

```
# Run with:
#
# Rscript --vanilla ImageRegistrationMethodDisplacement1.R fixedImageFilter␣
↪movingImageFile outputTransformFile
#

library(SimpleITK)
```

(continues on next page)

(continued from previous page)

```
commandIteration <- function(method)
{
    if (method$GetOptimizerIteration()==0) {
        cat("\tLevel:", method$GetCurrentLevel(),
            "\n\tScales:", method$GetOptimizerScales(), "\n")
    }
    msg <- paste("#", method$GetOptimizerIteration(),
                 "\n\tMetric Value: ", method$GetMetricValue(),
                 "\n\tLearning Rate: ", method$GetOptimizerLearningRate(),
                 '\n', sep="")
    cat(msg)
}

commandMultiIteration <- function(method)
{
    msg <- paste("\tStop Condition: ", method$GetOptimizerStopConditionDescription(),
                 "\n============= Resolution Change =============\n", sep="")
    cat(msg)
}

args <- commandArgs( TRUE )

if (length(args) != 3) {
    stop("3 arguments expected - fixedImageFilter, movingImageFile,␣
↪outputTransformFile")
}

fixed <- ReadImage(args[[1]], 'sitkFloat32')
moving <- ReadImage(args[[2]], 'sitkFloat32')

initialTx <- CenteredTransformInitializer(fixed, moving, AffineTransform(fixed
↪$GetDimension()))

R <- ImageRegistrationMethod()

R$SetShrinkFactorsPerLevel(c(3,2,1))
R$SetSmoothingSigmasPerLevel(c(2,1,1))

R$SetMetricAsJointHistogramMutualInformation(20)
R$MetricUseFixedImageGradientFilterOff()

R$SetOptimizerAsGradientDescent(learningRate=1.0,
                                numberOfIterations=100,
                                convergenceMinimumValue=1e-6,
                                convergenceWindowSize=10,
                                estimateLearningRate = 'EachIteration')
R$SetOptimizerScalesFromPhysicalShift()

R$SetInitialTransform(initialTx, inPlace=TRUE)

R$SetInterpolator('sitkLinear')

R$AddCommand( 'sitkIterationEvent', function() commandIteration(R) )
R$AddCommand( 'sitkMultiResolutionIterationEvent', function()␣
↪commandMultiIteration(R) )
```

(continues on next page)

(continued from previous page)

```
outTx <- R$Execute(fixed, moving)

cat("-------\n")
outTx
cat("Optimizer stop condition:", R$GetOptimizerStopConditionDescription(), '\n')
cat("Iteration:", R$GetOptimizerIteration(), '\n')
cat("Metric value:", R$GetMetricValue(), '\n')


displacementField <- Image(fixed$GetSize(), 'sitkVectorFloat64')
displacementField$CopyInformation(fixed)
displacementTx <- DisplacementFieldTransform(displacementField)
rm(displacementField)
displacementTx$SetSmoothingGaussianOnUpdate(varianceForUpdateField=0.0,
                                            varianceForTotalField=1.5)

R$SetMovingInitialTransform(outTx)
R$SetInitialTransform(displacementTx, inPlace=TRUE)

R$SetMetricAsANTSNeighborhoodCorrelation(4)
R$MetricUseFixedImageGradientFilterOff()

R$SetShrinkFactorsPerLevel(c(3,2,1))
R$SetSmoothingSigmasPerLevel(c(2,1,1))

R$SetOptimizerScalesFromPhysicalShift()
R$SetOptimizerAsGradientDescent(learningRate=1,
                                numberOfIterations=300,
                                convergenceMinimumValue=1e-6,
                                convergenceWindowSize=10,
                                estimateLearningRate = 'EachIteration')

outTx$AddTransform( R$Execute(fixed, moving) )

cat("-------\n")
outTx
cat("Optimizer stop condition:", R$GetOptimizerStopConditionDescription(), '\n')
cat("Iteration:", R$GetOptimizerIteration(), '\n')
cat("Metric value:", R$GetMetricValue(), '\n')

WriteTransform(outTx,  args[[3]])
```

11.19 Image Registration Method Exhaustive

11.19.1 Overview

11.19.2 Code

Python

```
#!/usr/bin/env python

"""
```

(continues on next page)

(continued from previous page)

```
This script demonstrates the use of the Exhaustive optimizer in the
ImageRegistrationMethod to estimate a good initial rotation position.

Because gradient descent base optimization can get stuck in local
minima, a good initial transform is critical for reasonable
results. Search a reasonable space on a grid with brute force may be a
reliable way to get a starting location for further optimization.

The initial translation and center of rotation for the transform is
initialized based on the first principle moments of the intensities of
the image. Then in either 2D or 3D a Euler transform is used to
exhaustively search a grid of the rotation space at a certain step
size. The resulting transform is a reasonable guess where to start
further registration.
"""

from __future__ import print_function
from __future__ import division


import SimpleITK as sitk
import sys
import os
from math import pi


def command_iteration(method) :
    if (method.GetOptimizerIteration()==0):
        print("Scales: ", method.GetOptimizerScales())
    print("{0:3} = {1:7.5f} : {2}".format(method.GetOptimizerIteration(),
                                          method.GetMetricValue(),
                                          method.GetOptimizerPosition()))



if len ( sys.argv ) < 4:
    print( "Usage: {0} <fixedImageFilter> <movingImageFile>  <outputTransformFile>".
↪format(sys.argv[0]))
    sys.exit ( 1 )

fixed = sitk.ReadImage(sys.argv[1], sitk.sitkFloat32)

moving = sitk.ReadImage(sys.argv[2], sitk.sitkFloat32)

R = sitk.ImageRegistrationMethod()

R.SetMetricAsMattesMutualInformation(numberOfHistogramBins = 50)

sample_per_axis=12
if fixed.GetDimension() == 2:
    tx = sitk.Euler2DTransform()
    # Set the number of samples (radius) in each dimension, with a
    # default step size of 1.0
    R.SetOptimizerAsExhaustive([sample_per_axis//2,0,0])
    # Utilize the scale to set the step size for each dimension
    R.SetOptimizerScales([2.0*pi/sample_per_axis, 1.0,1.0])
elif fixed.GetDimension() == 3:
```

(continues on next page)

(continued from previous page)

```
    tx = sitk.Euler3DTransform()
    R.SetOptimizerAsExhaustive([sample_per_axis//2,sample_per_axis//2,sample_per_axis/
↪/4,0,0,0])
    R.SetOptimizerScales([2.0*pi/sample_per_axis,2.0*pi/sample_per_axis,2.0*pi/sample_
↪per_axis,1.0,1.0,1.0])

# Initialize the transform with a translation and the center of
# rotation from the moments of intensity.
tx = sitk.CenteredTransformInitializer(fixed, moving, tx)

R.SetInitialTransform(tx)

R.SetInterpolator(sitk.sitkLinear)

R.AddCommand( sitk.sitkIterationEvent, lambda: command_iteration(R) )

outTx = R.Execute(fixed, moving)

print("-------")
print(outTx)
print("Optimizer stop condition: {0}".format(R.
↪GetOptimizerStopConditionDescription()))
print(" Iteration: {0}".format(R.GetOptimizerIteration()))
print(" Metric value: {0}".format(R.GetMetricValue()))


sitk.WriteTransform(outTx,  sys.argv[3])

if ( not "SITK_NOSHOW" in os.environ ):

    resampler = sitk.ResampleImageFilter()
    resampler.SetReferenceImage(fixed);
    resampler.SetInterpolator(sitk.sitkLinear)
    resampler.SetDefaultPixelValue(1)
    resampler.SetTransform(outTx)

    out = resampler.Execute(moving)

    simg1 = sitk.Cast(sitk.RescaleIntensity(fixed), sitk.sitkUInt8)
    simg2 = sitk.Cast(sitk.RescaleIntensity(out), sitk.sitkUInt8)
    cimg = sitk.Compose(simg1, simg2, simg1//2.+simg2//2.)
    sitk.Show( cimg, "ImageRegistrationExhaustive Composition" )
```

R

```
# Run with:
#
# Rscript --vanilla ImageRegistrationMethodExhaustive.R fixedImageFilter␣
↪movingImageFile outputTransformFile
#

# This script demonstrates the use of the Exhaustive optimizer in the
# ImageRegistrationMethod to estimate a good initial rotation position.

# Because gradient descent base optimization can get stuck in local
# minima, a good initial transform is critical for reasonable
# results. Search a reasonable space on a grid with brute force may be a
```

(continues on next page)

(continued from previous page)

```
# reliable way to get a starting location for further optimization.

# The initial translation and center of rotation for the transform is
# initialized based on the first principle moments of the intensities of
# the image. Then in either 2D or 3D a Euler transform is used to
# exhaustively search a grid of the rotation space at a certain step
# size. The resulting transform is a reasonable guess where to start
# further registration.

library(SimpleITK)

commandIteration <- function(method)
{
    if (method$GetOptimizerIteration()==0) {
        cat("Scales:", method$GetOptimizerScales(), "\n")
    }
    msg <- paste(method$GetOptimizerIteration(), "=",
                 method$GetMetricValue(), ":",
                 method$GetOptimizerPosition(), "\n" )
    cat(msg)
}

args <- commandArgs( TRUE )

if (length(args) != 3) {
    stop("3 arguments expected - fixedImageFilter, movingImageFile,␣
↪outputTransformFile")
}

fixed <- ReadImage(args[[1]], 'sitkFloat32')
moving <- ReadImage(args[[2]], 'sitkFloat32')

R <- ImageRegistrationMethod()
R$SetMetricAsMattesMutualInformation(numberOfHistogramBins = 50)

sample_per_axis <- 12
if (fixed$GetDimension() == 2) {
    tx <- Euler2DTransform()
    # Set the number of samples (radius) in each dimension, with a
    # default step size of 1.0
    R$SetOptimizerAsExhaustive(c(sample_per_axis%/%2,0,0))
    # Utilize the scale to set the step size for each dimension
    R$SetOptimizerScales(c(2.0*pi/sample_per_axis,1.0,1.0))
} else if (fixed.GetDimension() == 3) {
    tx <- Euler3DTransform()
    R$SetOptimizerAsExhaustive(c(sample_per_axis%/%2,sample_per_axis%/%2,sample_per_
↪axis%/%4,0,0,0))
    R$SetOptimizerScales(c(2.0*pi/sample_per_axis,2.0*pi/sample_per_axis,2.0*pi/
↪sample_per_axis,1.0,1.0,1.0))
}

# Initialize the transform with a translation and the center of
# rotation from the moments of intensity.
tx <- CenteredTransformInitializer(fixed, moving, tx)
R$SetInitialTransform(tx)
R$SetInterpolator('sitkLinear')
R$AddCommand( 'sitkIterationEvent', function() commandIteration(R) )
```

(continues on next page)

(continued from previous page)

```
outTx <- R$Execute(fixed, moving)

cat("-------\n")
outTx
cat("Optimizer stop condition:", R$GetOptimizerStopConditionDescription(), '\n')
cat("Iteration:", R$GetOptimizerIteration(), '\n')
cat("Metric value:", R$GetMetricValue(), '\n')

WriteTransform(outTx, args[[3]])
```

11.20 Integration with ITK

11.20.1 Overview

11.20.2 Code

C++

```
#if defined(_MSC_VER)
#pragma warning ( disable : 4786 )
#endif

// SimpleITK includes
#include "SimpleITK.h"

// ITK includes
#include "itkImage.h"
#include "itkCurvatureFlowImageFilter.h"

// create convenient namespace alias
namespace sitk = itk::simple;

/**
 * This example shows how ITK and SimpleITK can be used together to work
 * on the same data. We use the same example application as the one presented
 * in the Segmentation/ConnectedThresholdImageFilter.cxx example, but we
 * replace the SimpleITK version of CurvatureFlowImageFilter with the
 * corresponding ITK version. While not terribly useful in this situation since
 * CurvatureFlowImageFilter is already available in SimpleITK this demonstrates
 * how ITK filters that have not been converted for SimpleITK can still be used
 * in a SimpleITK context
 */
int main( int argc, char *argv[])
{

  //
  // Check command line parameters
  //
  if( argc < 7 )
    {
    std::cerr << "Missing Parameters " << std::endl;
    std::cerr << "Usage: " << argv[0];
```

(continues on next page)

(continued from previous page)

```
    std::cerr << " inputImage outputImage lowerThreshold upperThreshold "
      "seedX seedY [seed2X seed2Y ... ]" << std::endl;
    return 1;
    }


  //
  // Read the image
  //
  sitk::ImageFileReader reader;
  reader.SetFileName( std::string( argv[1] ) );
  sitk::Image image = reader.Execute();


  //
  // Set up writer
  //
  sitk::ImageFileWriter writer;
  writer.SetFileName( std::string( argv[2] ) );

  //////
  // Blur using CurvatureFlowImageFilter
  //
  // Here we demonstrate the use of the ITK version of CurvatureFlowImageFilter
  // instead of the SimpleITK version.
  //////

  //
  // First, define the typedefs that correspond to the types of the input
  // image. This requires foreknowlege of the data type of the input image.
  //
  const unsigned int                                  Dimension = 2;
  typedef float                                       InternalPixelType;
  typedef itk::Image< InternalPixelType, Dimension > InternalImageType;

  //
  // We must check the image dimension and the pixel type of the
  // SimpleITK image match the ITK image we will cast to.s
  //
  if ( image.GetDimension() != Dimension )
    {
    std::cerr << "Input image is not a " << Dimension << " dimensional image as␣
↪expected!" << std::endl;
    return 1;
    }

  //
  // The read sitk::Image could be any pixel type. Cast the image, to
  // float so we know what type we have.
  //
  sitk::CastImageFilter caster;
  caster.SetOutputPixelType( sitk::sitkFloat32 );
  image = caster.Execute( image );

  //
  // Extract the itk image from the SimpleITK image
  //
```

(continues on next page)

(continued from previous page)

```
  InternalImageType::Pointer itkImage =
    dynamic_cast <InternalImageType*>( image.GetITKBase() );

  //
  // Always check the results of dynamic_casts
  //
  if ( itkImage.IsNull() )
    {
    std::cerr << "Unexpected error converting SimpleITK image to ITK image!" <<␣
↪std::endl;
    return 1;
    }

  //
  // Set up the blur filter and attach it to the pipeline.
  //
  typedef itk::CurvatureFlowImageFilter< InternalImageType, InternalImageType >
                                                        BlurFilterType;
  BlurFilterType::Pointer blurFilter = BlurFilterType::New();
  blurFilter->SetInput( itkImage );
  blurFilter->SetNumberOfIterations( 5 );
  blurFilter->SetTimeStep( 0.125 );



  //
  // Execute the  Blur pipeline by calling Update() on the blur filter.
  //
  blurFilter->Update();


  //
  // Return to the simpleITK setting by making a SimpleITK image using the
  // output of the blur filter.
  //
  sitk::Image blurredImage = sitk::Image( blurFilter->GetOutput() );


  //////
  // Now that we have finished the ITK section, we return to the SimpleITK API
  //////


  //
  // Set up ConnectedThresholdImageFilter for segmentation
  //
  sitk::ConnectedThresholdImageFilter segmentationFilter;
  segmentationFilter.SetLower( atof( argv[3] ) );
  segmentationFilter.SetUpper( atof( argv[4] ) );
  segmentationFilter.SetReplaceValue( 255 );

  for (int i = 5; i+1 < argc; i+=2)
    {
    std::vector<unsigned int> seed;
    seed.push_back(atoi(argv[i]));
    seed.push_back(atoi(argv[i+1]));
    segmentationFilter.AddSeed(seed);
```

(continues on next page)

(continued from previous page)

```
    std::cout << "Adding a seed at ";
    for( unsigned int j = 0; j < seed.size(); ++i )
      {
      std::cout << seed[j] << " ";
      }
    std::cout << std::endl;
    }

  sitk::Image outImage = segmentationFilter.Execute(blurredImage);


  //
  // Write out the resulting file
  //
  writer.Execute(outImage);

  return 0;
}
```

11.21 N4 Bias Field Correction

11.21.1 Overview

The N4 bias field correction algorithm is a popular method for correcting low frequency intensity non-uniformity present in MRI image data known as a bias or gain field. The method has also been successfully applied as flat-field correction in microscopy data. This method assumes a simple parametric model and does not require tissue classification.

This example demonstrates how to use the SimpleITK N4BiasFieldCorrectionImageFilter class. This filter has one required input image which is affected by a bias field we wish to correct. The primary input is required to have a "real" pixel type of either *sitkFloat32* or *sitkFloat64*. Additionally, there is an optional "MaskImage" input which specifies which pixels are used to estimate the bias-field and suppress pixels close to zero. It is recommended that the mask image follows the common conventions for *masked images*, which is being of pixel type *sitkUint8* and having values of 0 and 1 representing the mask. Additionally, the mask image and the main input image must occupy the same physical space to ensure pixel to pixel correspondence.

Some basic parameters for using this algorithm are parsed from the command line in this example. The shrink factor is used to reduce the size and complexity of the image. The N4 algorithm uses a multi-scale optimization approach to compute the bias field. The **SetMaximumNumberOfIterations** method takes an array used to set the limit of iterations per resolution level, thereby setting both the iterations and the number of scales (from the length of the array). The output of the filter is the bias corrected input image.

11.21.2 Code

Python

```
#!/usr/bin/env python

from __future__ import print_function

import SimpleITK as sitk
import sys
```

(continues on next page)

(continued from previous page)

```
import os

if len ( sys.argv ) < 2:
    print( "Usage: N4BiasFieldCorrection inputImage " + \
        "outputImage [shrinkFactor] [maskImage] [numberOfIterations] " +\
        "[numberOfFittingLevels]" )
    sys.exit ( 1 )


inputImage = sitk.ReadImage( sys.argv[1] )

if len ( sys.argv ) > 4:
    maskImage = sitk.ReadImage( sys.argv[4], sitk.sitkUint8 )
else:
    maskImage = sitk.OtsuThreshold( inputImage, 0, 1, 200 )

if len ( sys.argv ) > 3:
    inputImage = sitk.Shrink( inputImage, [ int(sys.argv[3]) ] * inputImage.
↪GetDimension() )
    maskImage = sitk.Shrink( maskImage, [ int(sys.argv[3]) ] * inputImage.
↪GetDimension() )

inputImage = sitk.Cast( inputImage, sitk.sitkFloat32 )

corrector = sitk.N4BiasFieldCorrectionImageFilter();

numberFittingLevels = 4

if len ( sys.argv ) > 6:
    numberFittingLevels = int( sys.argv[6] )

if len ( sys.argv ) > 5:
    corrector.SetMaximumNumberOfIterations( [ int( sys.argv[5] ) ]␣
↪*numberFittingLevels  )


output = corrector.Execute( inputImage, maskImage )

sitk.WriteImage( output, sys.argv[2] )


if ( not "SITK_NOSHOW" in os.environ ):
    sitk.Show( output, "N4 Corrected" )
```

R

```
# Run with:
#
# Rscript --vanilla N4BiasFieldCorrection.R inputImage, outputImage, shrinkFactor,␣
↪maskImage, numberOfIterations, numberOfFittingLevels
#

library(SimpleITK)

args <- commandArgs( TRUE )

if (length(args) < 2) {
```

(continues on next page)

(continued from previous page)

```
    stop("At least 2 arguments expected - inputImage, outputImage, [shrinkFactor], ",
         "[maskImage], [numberOfIterations], [numberOfFittingLevels]")
}

inputImage <- ReadImage(args[[1]])

if (length( args ) > 4) {
    maskImage <- ReadImage( args[[4]], 'sitkUint8' )
} else {
    maskImage <- OtsuThreshold( inputImage, 0, 1, 200 )
}

if (length( args ) > 3) {
    inputImage <- Shrink( inputImage, rep(strtoi(args[3]), inputImage$GetDimension())␣
↪)
    maskImage <- Shrink( maskImage, rep(strtoi(args[3]), inputImage$GetDimension()) )
}

inputImage <- Cast( inputImage, 'sitkFloat32' )

corrector <- N4BiasFieldCorrectionImageFilter()

numberFittingLevels <- 4

if (length ( args ) > 6) {
    numberFittingLevels <- strtoi( args[[6]] )
}

if (length ( args ) > 5) {
    corrector$SetMaximumNumberOfIterations( rep(strtoi( args[[5]],␣
↪numberFittingLevels)) )
}

output <- corrector$Execute( inputImage, maskImage )

WriteImage(output, args[[2]])
```

11.22 Reading-Gaussian Blurring-Writing

11.22.1 Overview

Introductory example which demotrates the basics of reading, filtering, and writing an image. This examples works for any scalar or vector image type. It processes the image with a Gaussian blurring filter, which produces an image with floating point pixel type, then cast the output back to the input before writing the image to a file.

This example uses the object oriented (OO) interface to SimpleITK classes. The OO style produces more verbose code which clearly labels the parameters set by class member functions.

11.22.2 Code

C#

```
using System;
using itk.simple;

namespace itk.simple.examples {
    class SimpleGaussian {
        static void Main(string[] args) {
            try {
                if (args.Length < 3) {
                    Console.WriteLine("Usage: SimpleGaussian <input> <sigma> <output>
↪");
                    return;
                }
                // Read input image
                ImageFileReader reader = new ImageFileReader();
                reader.SetFileName(args[0]);
                Image image = reader.Execute();

                // Execute Gaussian smoothing filter
                SmoothingRecursiveGaussianImageFilter gaussian = new␣
↪SmoothingRecursiveGaussianImageFilter();
                gaussian.SetSigma(Double.Parse(args[1]));
                Image blurredImage = gaussian.Execute(image);

                // Covert the real output image back to the original pixel type , to
                // make writing easier , as many file formats don 't support real
                // pixels .
                CastImageFilter castFilter = new CastImageFilter();
                castFilter.SetOutputPixelType(image.GetPixelID());
                Image destImage = castFilter.Execute(blurredImage);

                // Write output image
                ImageFileWriter writer = new ImageFileWriter();
                writer.SetFileName(args[2]);
                writer.Execute(destImage);

            } catch (Exception ex) {
                Console.WriteLine(ex);
            }
        }
    }
}
```

C++

```
// This one header will include all SimpleITK filters and external
// objects.
#include <SimpleITK.h>
#include <iostream>
#include <stdlib.h>


// create convenient namespace alias
namespace sitk = itk::simple;

int main ( int argc, char* argv[] ) {

  if ( argc < 4 ) {
```

(continues on next page)

(continued from previous page)

```
    std::cerr << "Usage: " << argv[0] << " <input> <sigma> <output>\n";
    return 1;
  }

  // Read the image file
  sitk::ImageFileReader reader;
  reader.SetFileName ( std::string ( argv[1] ) );
  sitk::Image image = reader.Execute();

  // This filters perform a gaussian bluring with sigma in physical
  // space. The output image will be of real type.
  sitk::SmoothingRecursiveGaussianImageFilter gaussian;
  gaussian.SetSigma ( atof ( argv[2] ) );
  sitk::Image blurredImage = gaussian.Execute ( image );

  // Covert the real output image back to the original pixel type, to
  // make writing easier, as many file formats don't support real
  // pixels.
  sitk::CastImageFilter caster;
  caster.SetOutputPixelType( image.GetPixelID() );
  sitk::Image outputImage = caster.Execute( blurredImage );

  // write the image
  sitk::ImageFileWriter writer;
  writer.SetFileName ( std::string ( argv[3] ) );
  writer.Execute ( outputImage );

  return 0;
}
```

Java

```
/**
 * Simple test of SimpleITK's java wrapping
 */


import org.itk.simple.*;

class SimpleGaussian {

  public static void main(String argv[]) {

    if ( argv.length < 3 ) {
      System.out.println("Usage: java SimpleGaussian <input> <sigma> <output>");
      return;
    }

    org.itk.simple.ImageFileReader reader = new org.itk.simple.ImageFileReader();
    reader.setFileName(argv[0]);
    Image img = reader.execute();

    SmoothingRecursiveGaussianImageFilter filter = new␣
↪SmoothingRecursiveGaussianImageFilter();
    filter.setSigma( Double.valueOf( argv[1] ).doubleValue() );
    Image blurredImg = filter.execute(img);
```

(continues on next page)

(continued from previous page)

```
    CastImageFilter caster = new CastImageFilter();
    caster.setOutputPixelType( img.getPixelID() );
    Image castImg = caster.execute( blurredImg );

    ImageFileWriter writer = new ImageFileWriter();
    writer.setFileName(argv[2]);
    writer.execute( castImg );

  }

}
```

Lua

```
require "SimpleITK"

if #arg < 3 then
  print ( "Usage: SimpleGaussian <input> <sigma> <output>" )
  os.exit ( 1 )
end

reader = SimpleITK.ImageFileReader()
-- Remember that Lua arrays are 1-based, and that arg does not contain the␣
↪application name!
reader:SetFileName ( arg[1] )
image = reader:Execute();

inputPixelType = image:GetPixelID()

gaussian = SimpleITK.SmoothingRecursiveGaussianImageFilter()
gaussian:SetSigma ( arg[2] )
image = gaussian:Execute ( image );

caster = SimpleITK.CastImageFilter();
caster:SetOutputPixelType( inputPixelType );
image = caster:Execute( image )

writer = SimpleITK.ImageFileWriter()
writer:SetFileName ( arg[3] )
writer:Execute ( image );
```

Python

```
#!/usr/bin/env python

from __future__ import print_function

import SimpleITK as sitk
import sys
import os

if len ( sys.argv ) < 4:
    print( "Usage: SimpleGaussian <input> <sigma> <output>" )
    sys.exit ( 1 )


reader = sitk.ImageFileReader()
```

(continues on next page)

(continued from previous page)

```
reader.SetFileName ( sys.argv[1] )
image = reader.Execute()

pixelID = image.GetPixelID()

gaussian = sitk.SmoothingRecursiveGaussianImageFilter()
gaussian.SetSigma ( float ( sys.argv[2] ) )
image = gaussian.Execute ( image )

caster = sitk.CastImageFilter()
caster.SetOutputPixelType( pixelID )
image = caster.Execute( image )

writer = sitk.ImageFileWriter()
writer.SetFileName ( sys.argv[3] )
writer.Execute ( image );


if ( not "SITK_NOSHOW" in os.environ ):
    sitk.Show( image, "Simple Gaussian" )
```

R

```
# Run with:
#
# Rscript --vanilla SimpleGaussian.R  input sigma output
#

library(SimpleITK)

args <- commandArgs( TRUE )

myreader <- ImageFileReader()
myreader$SetFileName(args[[1]] )
myimage <- myreader$Execute()

pixeltype <- myimage$GetPixelID()

myfilter <- SmoothingRecursiveGaussianImageFilter()
myfilter$SetSigma(as.numeric(args[2]) )
smoothedimage <- myfilter$Execute(myimage )

mycaster <- CastImageFilter()
mycaster$SetOutputPixelType(pixeltype )
castedimage <- mycaster$Execute(smoothedimage )

mywriter <- ImageFileWriter()
mywriter$SetFileName(args[[3]] )
mywriter$Execute(castedimage )
```

Ruby

```
require 'simpleitk'

if ARGV.length != 3 then
  puts "Usage: SimpleGaussian <input> <sigma> <output>";
  exit( 1 )
```

(continues on next page)

(continued from previous page)

```
end

reader = Simpleitk::ImageFileReader.new
reader.set_file_name( ARGV[0] )
image = reader.execute

inputPixelType = image.get_pixel_idvalue

gaussian = Simpleitk::SmoothingRecursiveGaussianImageFilter.new
gaussian.set_sigma ARGV[1].to_f
image = gaussian.execute image;

caster = Simpleitk::CastImageFilter.new
caster.set_output_pixel_type inputPixelType
image = caster.execute image

writer = Simpleitk::ImageFileWriter.new
writer.set_file_name ARGV[2]
writer.execute image
```

Tcl

```
if { $argc < 3 } {
    puts "Usage: SimpleGaussian <input> <sigma> <output>"
    exit 1
}

ImageFileReader reader
reader SetFileName [ lindex $argv 0 ]
set image [ reader Execute ]

set pixelID [ $image GetPixelID ]

SmoothingRecursiveGaussianImageFilter gaussian
gaussian SetSigma [ lindex $argv 1 ]
set image [ gaussian Execute $image ]

CastImageFilter caster
caster SetOutputPixelType $pixelID
set image [ caster Execute $image ]

ImageFileWriter writer
writer SetFileName [ lindex $argv 2]
writer Execute $image

# Tcl requires explicit cleanup Cleanup
reader -delete
gaussian -delete
caster -delete
$image -delete
writer -delete
```

11.23 Image Viewing

11.23.1 Overview

This example illustrates the basic image viewing capabilities provided by SimpleITK.

The focus of SimpleITK is on image analysis and not display. We therefor use an ad-hoc image viewing approach, relying on an external program. By default this is the Fiji/ImageJ program.

When using this functionality, SimpleITK writes the image to disc in a temporary location and then launches the viewer. Potential issues with this approach:

1. The external viewer program is not found. Either it is not installed, or not installed in the expected location, for details see *Why isn't ImageJ found by the Show function (RuntimeError: Exception thrown...)?*)
2. Writing the image to disc fails when there isn't enough disc space.
3. The external program cannot read the image file because it was saved in a format not supported by the program.

To control viewing you have two options:

1. Procedural approach. Use the Show function, with control primarily done via environment variable settings.
2. Object oriented approach. Use the ImageViewer class, with control via the object's interface.

One can either set the viewing application, if you just want to change the specific viewer, or set the viewing command. The latter allows you to specify viewer specific arguments such as -z for ITK-SNAP, see below, to open in a zoomed view.

11.23.2 Code

Python

```
import sys
import SimpleITK as sitk


grid_image = sitk.GridSource(outputPixelType=sitk.sitkUInt16, size=(512,512),
                             sigma=(0.1,0.1), gridSpacing=(20.0,20.0))

# Procedural interface, using the default image viewer (Fiji/ImageJ) or
# any viewer specified by the SITK_SHOW_COMMAND environment variable.
sitk.Show(grid_image, title = "grid using Show function", debugOn = True)

# Object oriented interface:
image_viewer = sitk.ImageViewer()
image_viewer.SetTitle('grid using ImageViewer class')

# Use the default image viewer.
image_viewer.Execute(grid_image)

# Change viewer, and display again.
image_viewer.SetApplication('/Applications/ITK-SNAP.app/Contents/MacOS/ITK-SNAP')
image_viewer.Execute(grid_image)

# Change the viewer command, (use ITK-SNAP's -z option to open the image in zoomed␣
↪mode)
image_viewer.SetCommand('/Applications/ITK-SNAP.app/Contents/MacOS/ITK-SNAP -z 2')
```

(continues on next page)

(continued from previous page)

```
image_viewer.Execute(grid_image)

sys.exit( 0 )
```

R

```
# Run with:
#
# Rscript --vanilla ImageViewing.R
#

library(SimpleITK)

grid_image <- GridSource(outputPixelType="sitkUInt16", size=c(512,512),
                         sigma=c(0.1,0.1), gridSpacing=c(20.0,20.0))

# Procedural interface, using the default image viewer (Fiji/ImageJ) or
# any viewer specified by the SITK_SHOW_COMMAND environment variable.
Show(grid_image, title = "grid using Show function", debugOn = TRUE)

# Object oriented interface:
image_viewer <- ImageViewer()
image_viewer$SetTitle('grid using ImageViewer class')

# Use the default image viewer.
image_viewer$Execute(grid_image)

# Change viewer, and display again.
image_viewer$SetApplication('/Applications/ITK-SNAP.app/Contents/MacOS/ITK-SNAP')
image_viewer$Execute(grid_image)

# Change the viewer command, (use ITK-SNAP's -z option to open the image in zoomed␣
↪mode)
image_viewer$SetCommand('/Applications/ITK-SNAP.app/Contents/MacOS/ITK-SNAP -z 2')
image_viewer$Execute(grid_image)
```

11.24 Advanced Image Reading

11.24.1 Overview

This example illustrates advanced image reading options:

1. Querying an image for its meta-data without reading the bulk intensity data.
2. Reading a sub-region of an image.

Querying an image for its meta-data

In some cases you may only want to read an image's meta-data without reading the bulk intensity data into memory. For instance, if you want to read subregions of an image and not the whole image or if you want to display image information (e.g. patient name) to a user in a GUI which then allows them to select a specific image.

Reading an image sub-region

In some cases you are only interested in reading a sub-region of an image. This may be due to memory constraints or if it is known that the relevant content is always in a sub region of the image. If the specific image IO used for this operation supports streaming then this will indeed only read the sub-region, otherwise the whole image is read into memory and the sub-region is returned. The IOs that support streaming are:

1. TIFFImageIO (supports a subset of encodings)
2. MetaImageIO
3. NrrdImageIO
4. HDF5ImageIO (supports a subset of encodings)
5. MRCImageIO
6. VTKImageIO (supports a subset of encodings)

11.24.2 Code

Python

```
from __future__ import print_function

import sys
import numpy as np
import SimpleITK as sitk


if len(sys.argv)<2:
    print('Wrong number of arguments.', file=sys.stderr)
    print('Usage: ' + __file__ + ' image_file_name', file=sys.stderr)
    sys.exit(1)

# Read image information without reading the bulk data.
file_reader = sitk.ImageFileReader()
file_reader.SetFileName(sys.argv[1])
file_reader.ReadImageInformation()
print('image size: {0}\nimage spacing: {1}'.format(file_reader.GetSize(), file_reader.
↪GetSpacing()))
# Some files have a rich meta-data dictionary (e.g. DICOM)
for key in file_reader.GetMetaDataKeys():
    print(key + ': ' + file_reader.GetMetaData(key))
print('-'*20)

# When low on memory, we can incrementally work on sub-images. The following
# subtracts two images (ok, the same image) by reading them as multiple sub-images.

image1_file_name = sys.argv[1]
image2_file_name = sys.argv[1]

parts = 5 # Number of sub-regions we use

file_reader = sitk.ImageFileReader()
file_reader.SetFileName(image1_file_name)
file_reader.ReadImageInformation()
```

(continues on next page)

(continued from previous page)

```
image_size = file_reader.GetSize()

result_img = sitk.Image(file_reader.GetSize(), file_reader.GetPixelID(), file_reader.
↪GetNumberOfComponents())
result_img.SetSpacing(file_reader.GetSpacing())
result_img.SetOrigin(file_reader.GetOrigin())
result_img.SetDirection(file_reader.GetDirection())

extract_size = list(file_reader.GetSize())
extract_size[-1] = extract_size[-1]//parts
current_index = [0]*file_reader.GetDimension()
for i in range(parts):
    if i == (parts-1):
        extract_size[-1] = image_size[-1] - current_index[-1]
    file_reader.SetFileName(image1_file_name)
    file_reader.SetExtractIndex(current_index)
    file_reader.SetExtractSize(extract_size)
    sub_image1 = file_reader.Execute()

    file_reader.SetFileName(image2_file_name)
    file_reader.SetExtractIndex(current_index)
    file_reader.SetExtractSize(extract_size)
    sub_image2 = file_reader.Execute()
    result_img = sitk.Paste(result_img, sub_image1 - sub_image2, extract_size,␣
↪[0]*file_reader.GetDimension(), current_index)
    current_index[-1] += extract_size[-1]
del sub_image1
del sub_image2

# Check that our iterative approach is equivalent to reading the whole images.
if np.any(sitk.GetArrayViewFromImage(result_img - sitk.ReadImage(image1_file_name) +␣
↪sitk.ReadImage(image2_file_name))):
    print('Subtraction error.')
    sys.exit(1)
sys.exit(0)
```

R

```
library(SimpleITK)

args <- commandArgs( TRUE )

if (length(args) < 1) {
   write('Usage arguments: <image_file_name>', stderr())
   quit(save="no", status=1)
}

# Read image information without reading the bulk data.
file_reader <- ImageFileReader()
file_reader$SetFileName(args[1])
file_reader$ReadImageInformation()
cat('image size:', file_reader$GetSize(), '\n')
cat('image spacing:', file_reader$GetSpacing(), '\n')
# Some files have a rich meta-data dictionary (e.g. DICOM)
for(key in file_reader$GetMetaDataKeys())
{
  cat(paste0(key, ': ', file_reader$GetMetaData(key), '\n'))
```

(continues on next page)

(continued from previous page)

```
}
cat(rep('-',20),'\n', sep='')

# When low on memory, we can incrementally work on sub-images. The following
# subtracts two images (ok, the same image) by reading them as multiple sub-images.

image1_file_name <- args[1]
image2_file_name <- args[1]

parts <- 5 # Number of sub-regions we use

file_reader <- ImageFileReader()
file_reader$SetFileName(image1_file_name)
file_reader$ReadImageInformation()
image_size <- file_reader$GetSize()

result_img <- Image(file_reader$GetSize(), file_reader$GetPixelID(), file_reader
↪$GetNumberOfComponents())
result_img$SetSpacing(file_reader$GetSpacing())
result_img$SetOrigin(file_reader$GetOrigin())
result_img$SetDirection(file_reader$GetDirection())

extract_size <- file_reader$GetSize()
extract_size[-1] <- extract_size[-1]%/%parts
current_index <- rep(0,file_reader$GetDimension())
for(i in 1:parts)
{
  if(i == parts)
  {
    extract_size[-1] <- image_size[-1] - current_index[-1]
  }
  file_reader$SetFileName(image1_file_name)
  file_reader$SetExtractIndex(current_index)
  file_reader$SetExtractSize(extract_size)
  sub_image1 <- file_reader$Execute()

  file_reader$SetFileName(image2_file_name)
  file_reader$SetExtractIndex(current_index)
  file_reader$SetExtractSize(extract_size)
  sub_image2 <- file_reader$Execute()
  result_img <- Paste(result_img, sub_image1 - sub_image2, extract_size, rep(0,file_
↪reader$GetDimension()), current_index)
  current_index[-1] <- current_index[-1] + extract_size[-1]
}

rm(sub_image1, sub_image2)

# Check that our iterative approach is equivalent to reading the whole images.
if(any(as.array(result_img - ReadImage(image1_file_name) + ReadImage(image2_file_
↪name))!=0))
{
  cat('Subtraction error.\n')
  quit(save="no", status=1)
}
quit(save="no", status=0)
```

11.25 IO Selection for Image Reading

11.25.1 Overview

This example illustrates how to explicitly select a specific IO for image reading.

When using the default settings for the ImageFileReader class or the ReadImage function you have minimal control over the reading. That is, the specific IO mechanism is determined automatically based on the file type and the file is read into memory.

In some cases there are multiple IO classes that support reading the same image format and you do not know which one was used. For example the 'SCIFIOImageIO' and 'JPEGImageIO' both support reading JPEG images. As these are different implementations they may vary in performance and in the support of the features associated with the specific format. As a consequence, you may want or need to select one implementation over the other. Explicitly selecting the IO allows you to do so.

11.25.2 Code

Python

```
from __future__ import print_function

import sys
import SimpleITK as sitk


if len(sys.argv)<2:
    print('Wrong number of arguments.', file=sys.stderr)
    print('Usage: ' + __file__ + ' image_file_name', file=sys.stderr)
    sys.exit(1)

# Find out which image IOs are supported
file_reader = sitk.ImageFileReader()
image_ios_tuple = file_reader.GetRegisteredImageIOs()
print("The supported image IOs are: " + str(image_ios_tuple))
print('-'*20)

# Another option is to just print the reader and see which
# IOs are supported
print(file_reader)
print('-'*20)

# Force the use of a specific IO. If the IO doesn't support
# reading the image type it will throw an exception.
file_reader.SetImageIO('PNGImageIO')
file_reader.SetFileName(sys.argv[1])
try:
    image = file_reader.Execute()
    print('Read image: ' + sys.argv[1])
except Exception as err:
    print('Reading failed: ', err)
    sys.exit(1)

sys.exit(0)
```

R

```
library(SimpleITK)

args <- commandArgs( TRUE )

if (length(args) < 1) {
   write('Usage arguments: <image_file_name>', stderr())
   quit(save="no", status=1)
}

# Find out which image IOs are supported
file_reader <- ImageFileReader()
image_ios <- file_reader$GetRegisteredImageIOs()
cat('The supported image IOs are: ', image_ios, '\n')
cat(rep('-',20),'\n', sep='')

# Another option is to just print the reader and see which
# IOs are supported
print(file_reader)
cat(rep('-',20),'\n', sep='')

# Force the use of a specific IO.
file_reader$SetImageIO('PNGImageIO')
file_reader$SetFileName(args[1])

# If the IO doesn't support reading the image type it
# will throw an exception.
image <- tryCatch(file_reader$Execute(),
                  warning = function(err) {
                      message(err)
                      quit(save="no", status=1)
                  }
                  )
cat('Read image:',args[1],'\n')
quit(save="no", status=0)
```

CHAPTER 12

Relevant Resources

- Doxygen API Documentation
- Jupyter Notebooks in Python and R
- ITK Forum for Discussions and Questions

CHAPTER 13

How to Cite

If you find SimpleITK useful in your research, support our efforts by citing the relevant publication(s):

R. Beare, B. C. Lowekamp, Z. Yaniv, "Image Segmentation, Registration and Characterization in R with SimpleITK", *J Stat Softw*, 86(8), https://doi.org/10.18637/jss.v086.i08, 2018.

26. Yaniv, B. C. Lowekamp, H. J. Johnson, R. Beare, "SimpleITK Image-Analysis Notebooks: a Collaborative Environment for Education and Reproducible Research", *J Digit Imaging.*, https://doi.org/10.1007/s10278-017-0037-8, 31(3): 290-303, 2018 (freely read here).

B. C. Lowekamp, D. T. Chen, L. Ibáñez, D. Blezek, "The Design of SimpleITK", *Front. Neuroinform.*, 7:45. https://doi.org/10.3389/fninf.2013.00045, 2013.