SimpleITK Documentation

Release 1.0rc1

Insight Software Consortium

May 01, 2020

Contents

This is a prototype of consolidated SimpleITK documentation in Sphinx and is under development.

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.

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.

On this page

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 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 pip using `pip install -U pip`. It also requires that your Python environment is compatible with one of the pre-compiled binary wheels.

Alternatively, the wheels can be manually downloaded from sourceforge or PyPI, 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# platform can be found on SimpleITK's SourceForge page. 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-1.0.0-CSharp-win32-x86.zip
SimpleITK-1.0.0-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. 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. Currently, this function by default Show invokes Fiji then ImageJ to display images. ImageJ was chosen because it can handle all the image types that SimpleITK supports, even 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.

ImageJ

If ImageJ is used then we recommend downloading a recent version of ImageJ from the official home page. Recent versions come with support for the Nifti (*.nii) file format, which SimpleITK uses to export to ImageJ.

Note: Linux installation requires an additional step. The **"Show**" function searches for an executable named ImageJ or imagej, however the default tarball does not come with this file. Instead it comes with a file names script. This file contains the installation instructions. In short the file should be renamed to "imagej" and the site specific variables for the installation location, and java must be set. Also consider the "newwindow" variable... Do you really want a new instance of ImageJ launched each time you use Show? Lastly, as the installation instructions indicate, the imagej wrapper should be in your path.

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'll have a powerful interactive environment.

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

CHAPTER 2

Building SimpleITK

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

- 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

On this page

2.1 Prerequisites

To build SimpleITK you need:

- A recent version of CMake >= 3.3 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

2.2 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**: 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 form this GitHub repository.

2.3 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
```

2.3.1 Building using SuperBuild

After downloading SimpleITK's source code we **STRONGLY** recommend to run 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.

2.3.2 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 project 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 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:

```
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*, or as arguments on the command line by using the following format: *-D<var>=<value>*.

After proper configuration, SimpleITK can be built:

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

2.3.3 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.

2.3.4 Testing

After compilation the prudent thing to due is to test SimpleITK to ensure 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
```

2.3.5 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 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(inverse(T_f)(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

Frequently Asked Questions

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

On this page

4.1 Installation

4.1.1 Why do I get an error about a missing Dynamic Library when running SimpleITK with Python on windows?

This error has been resolved with SimpleITK version 0.5.1 and should no longer occur. Upgrading to the latest SimpleITK is encouraged.

This error occurs after you have downloaded the Windows SimpleITK binaries when you are running python and try to import SimpleITK. There is an error about a missing DLL on Windows when you don't have Visual Studio 10 and no other application has installed certain libraries before. You will need to download the Visual Studio 10 redistribution libraries. The libraries are available for download here.

4.1.2 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 can be resolved by installing the version of libpng that SimpleITK 0.9 was built against:

```
conda create -n sitkpy anaconda libpng=1.5
source activate sitkpy #unix/mac
# for win: activate sitkpy
conda install -c https://conda.binstar.org/simpleitk/channel/main SimpleITK
conda install libpng=1.5
```

This set of commands:

- creates the virtual environment with our choice of libpng version, all other anaconda packages will be compatible with this version.
- activate the virtual environment.
- installs SimpleITK into the virtual environment (unfortunately this will automatically upgrade you to libpng 1.6).
- downgrades to libpng 1.5 so that library versions are compatible.

We are currently investigating why the anaconda build system is not expressing version dependency for shared libraries. We how this will not be an issue with the next binary package.

4.2 How to Use

4.2.1 What filters are currently available in SimpleITK?

As of March 2014 we have approximately **260 ITK image filters** wrapped for SimpleITK. The **filter coverage table** shows the current set of ITK filters in SimpleITK. Additionally the Doxygen can be looked at to determine if a filter is available.

4.2.2 What image file formats can SimpleITK read?

4.2.3 How do I read a RAW image into SimpleITK?

In general raw image files are missing information. They do not contain the nessesary 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)
```

4.2.4 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.

4.2.5 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.

4.2.6 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.

4.3 Wrapping

4.3.1 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.

4.3.2 Tcl

4.3.3 Java

4.3.4 C#

4.3.5 R

4.4 Compilation

4.4.1 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.

Committed to Support

- GCC 4.2-4.7
- Visual Studio 2008 with Service Pack 1 (VS9)
- Visual Studio 2012 (VS10) (including Express)
- Visual Studio 2012 (VS11)

Noted Problems

- Compiling on a MS Windows 32-bit OS with static libraries is not supported due to lack of memory.
- With SimpleITK release 0.4.0, Visual Studio 2008 was not compiling. This problem has since been remedied in the development branch on April 18th, 2012.
- With SimpleITK release 0.7.0, Visual Studio 2008 is not able to compile all wrapped languages at the same time, it's recommenced to choose one at a time.

4.4.2 Why am I getting a compilation error on OSX Mavericks?

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.

4.4.3 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'.
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

4.4.4 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.

4.4.5 Do I need to download an optional package for C99?

SimpleITK will proved a “stdint.h” header if missing on the system.

4.4.6 How do I build with Visual Studio 2008?

Visual Studio 2008 is the oldest supported Microsoft development environment that SimpleITK supports. To build SimpleITK, certain features of C++TR1 are required. These features are best provided by the “Microsoft Visual Studio 2008 Service Pack 1” (or try this link 1). Alternatively just the Visual C++ 2008 Feature Pack Release can be installed. Please note that all our dashboard machines now use SP1.

Older versions of SimpleITK (<0.7.0) requires a also required a separately downloaded stdint.h for this compiler. This is not automatically provided if needed. If it's still needed the file can be downloaded here. For 64-bit Microsoft Windows it should be dragged with the GUI into the appropriate include path for the architecture.

4.4.7 What Configurations on Windows are Supported For Building?

There are quite a large number of configuration options available for the Windows platform. The following table is a guide line of what is regularly tested and confirmed to work or fail.

	Architecture	Library Type	Visual Studio 2008 SP1 (VS9)	Visual Studio 2010 (VS10)	Visual Studio 2012 (VS11)	Visual Studio 2013 (VS13)				
			Release	Debug	Release	Debug	Release	Debug	Release	Debug
32-bit Window OS	Only Intel 32-bit	Static	FAIL	FAIL	FAIL	FAIL				
Shared	FAIL	FAIL	Nightly	Nightly						
64-bit Window OS	Intel 32-bit	Static	Nightly		Nightly		Nightly		Nightly	
Shared										
Intel 64-bit	Static	Nightly		Nightly	Nightly					
Shared					Nightly		Nightly			

	Legend
Nightly	This combination of options is nightly tested, and known to work.
	This combinations has been manually tested, and is expected to work.
	It is not known if this combinations of options will work.
	This combination likely has problems, and is not recommended.
FAIL	These options are known not to work.

This table has been updated for the release branch, master, as of February 15th 2013.

4.4.8 Why are all of the configurations not supported on Windows?

One of the following errors frequently occur when the set of configuration options fail:

```
LINK : fatal error LNK1102: out of memory
```

```
LINK : fatal error LNK1248: image size (80000010) exceeds maximum allowable size (80000000)
```

These errors occur because of limitations in the compiler's linker or the operating system. For 64-bit architectures the linker is still only 32-bits on some Visual Studios. In certain configurations the linker can run out of memory. Also the Windows operating systems have a hard limit of 2GB for the size of libraries. For Debug mode configurations this limit can be encounted.

In general building in Debug mode should not be necessary, unless you are trying to debug SimpleITK or ITK. This configuration produces libraries that are very large because the compiler must maintain symbols for all instantiated ITK classes and member functions for each template parameters that a class is instantiating.

4.4.9 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.

If you have downloaded a tar-ball of the source code there should be an accompanying "SimpleITKData" tar-ball available, which contains the external data. It should populate the .ExternalData subdirectory of the SimpleITK source code directory when extracted.

4.4.10 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 5

Examples

5.1 Hello World

5.1.1 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.

5.1.2 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
```

5.2 CSharp Integration

5.2.1 Overview

5.2.2 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);

    }
  }
}
```

5.3 DemonsRegistration1

5.3.1 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*.

5.3.2 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)

demons = sitk.DemonsRegistrationFilter()
demons.SetNumberOfIterations( 50 )
demons.SetStandardDeviations( 1.0 )

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

displacementField = demons.Execute( fixed, moving )


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

(continues on next page)

(continued from previous page)

```
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" )
```

5.4 DemonsRegistration2

5.4.1 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*.

5.4.2 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)

demons = sitk.FastSymmetricForcesDemonsRegistrationFilter()
demons.SetNumberOfIterations(200)
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)
    resampler.SetDefaultPixelValue(100)
    resampler.SetTransform(outTx)
```

(continues on next page)

(continued from previous page)

```
    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" )
```

5.5 Read Image Meta-Data Dictionary and Print

5.5.1 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*.

5.5.2 Code

Python

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

from __future__ import print_function

import SimpleITK as sitk
import sys, os

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))
```

(continues on next page)

(continued from previous page)

```
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')
```

5.6 Dicom Series Reader

5.6.1 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*.

5.6.2 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]);
```

(continues on next page)

(continued from previous page)

```
    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

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)
```

(continues on next page)

(continued from previous page)

```
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]])
```

5.7 Dicom Series Read Modify Write

5.7.1 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*.

5.7.2 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.
# 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
```

(continues on next page)

(continued from previous page)

```
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

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.
```

(continues on next page)

(continued from previous page)

```
    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()

# 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).
```

(continues on next page)

(continued from previous page)

```
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$GetMetaData(0,k))
                                 }
                              })),
                       list(list("0008|0031",modification_time), # Series Time
                              list("0008|0021",modification_date), # Series Date
                              list("0008|0008","DERIVED\\SECONDARY"), # Image Type
                              list("0020|000e", paste0("1.2.826.0.1.3680043.2.1125.",
↪modification_date,".1",modification_time)), # Series Instance UID
                              list("0020|0037", paste(c(direction[1], direction[4],␣
↪direction[7],
                                                        direction[2],direction[5],
↪direction[8]), collapse="\\")), # Image Orientation (Patient)
                              list("0008|103e", paste0(series_reader$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)
```

(continues on next page)

(continued from previous page)

```
  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)
```

5.8 Dicom Series From Array

5.8.1 Overview

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

5.8.2 Code

Python

```
from __future__ import print_function

import SimpleITK as sitk

import sys, time, os
import numpy as np

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


# 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:
```

(continues on next page)

(continued from previous page)

```
#                           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.
# 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

for i in range(new_img.GetDepth()):
    image_slice = new_img[:,:,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
    # 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)
```

(continues on next page)

(continued from previous page)

```
# 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 )
```

5.9 Filter Progress Reporting

5.9.1 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.

5.9.2 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.

5.9.3 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.

5.9.4 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()
```

(continues on next page)

(continued from previous page)

```
    @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
```

5.9.5 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")} )
```

5.9.6 Code

CSharp

```
using System;
using itk.simple;

namespace itk.simple.examples {

//! [csharp director command]
  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() );
    }
  }
//! [csharp director command]

  class FilterProgressReporting {

    static void Main(string[] args) {
```

(continues on next page)

(continued from previous page)

```
      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
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)
    {}
```

(continues on next page)

(continued from previous page)

```
  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 );

  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);

      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
```

(continues on next page)

(continued from previous page)

```
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)
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

```
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()
```

(continues on next page)

(continued from previous page)

```
    @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
```

5.10 Image Registration Method1

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

5.10.1 Overview

5.10.2 Code

C#

```
using System;
using itk.simple;


namespace itk.simple.examples {
```

(continues on next page)

(continued from previous page)

```
  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);

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

(continues on next page)

(continued from previous page)

```
      // 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;

  public IterationUpdate(ImageRegistrationMethod m) {
    super();
```

(continues on next page)

(continued from previous page)

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

(continues on next page)

(continued from previous page)

```
  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
```

(continues on next page)

(continued from previous page)

```
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:
#
```

(continues on next page)

(continued from previous page)

```
# Rscript --vanilla ImageRegistrationMethod1.R fixedImage movingImage outputTransform
#


library(SimpleITK)


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

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', commandIteration(Reg))

outTx = Reg$Execute(fixed, moving)

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

5.11 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.

5.11.1 Overview

5.11.2 Code

C++

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

(continues on next page)

(continued from previous page)

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

(continues on next page)

(continued from previous page)

```
    simg2 = sitk.Cast(sitk.RescaleIntensity(out), sitk.sitkUInt8)
    cimg = sitk.Compose(simg1, simg2, simg1//2.+simg2//2.)
    sitk.Show( cimg, "ImageRegistration2 Composition" )
```

5.12 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.

5.12.1 Overview

5.12.2 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()

R.SetOptimizerAsRegularStepGradientDescent(learningRate=2.0,
```

(continues on next page)

(continued from previous page)

```
                                          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" )
```

5.13 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.

5.13.1 Overview

5.13.2 Code

Python

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

(continues on next page)

(continued from previous page)

```
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()))

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, "ImageRegistration4 Composition" )
```

5.14 Image Registration Method BSpline 1

5.14.1 Overview

5.14.2 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)
    {}

  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);
```

(continues on next page)

(continued from previous page)

```
      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;
  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 );
```

(continues on next page)

(continued from previous page)

```
  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())

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) )
```

(continues on next page)

(continued from previous page)

```
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" )
```

5.15 Image Registration Method BSpline 2

5.15.1 Overview

5.15.2 Code

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()))
    print("\t#: ", len(method.GetOptimizerPosition()))


def command_multi_iteration(method) :
    print("--------- Resolution Changing ---------")


if len ( sys.argv ) < 4:
```

(continues on next page)

(continued from previous page)

```
    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])

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" )
```

5.16 Image Registration Method Displacement 1

5.16.1 Overview

5.16.2 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
  : 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:
```

(continues on next page)

(continued from previous page)

```
  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 );
      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);
```

(continues on next page)

(continued from previous page)

```
  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;
  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,
```

(continues on next page)

(continued from previous page)

```
                                                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);

  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()))


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.MetricUseFixedImageGradientFilterOff()


R.SetOptimizerAsGradientDescent(learningRate=1.0,
                                numberOfIterations=100,
                                estimateLearningRate = R.EachIteration)
R.SetOptimizerScalesFromPhysicalShift()

R.SetInitialTransform(initialTx,inPlace=True)
```

(continues on next page)

(continued from previous page)

```
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
displacementTx.SetSmoothingGaussianOnUpdate(varianceForUpdateField=0.0,
                                            varianceForTotalField=1.5)



R.SetMovingInitialTransform(outTx)
R.SetInitialTransform(displacementTx, inPlace=True)

R.SetMetricAsANTSNeighborhoodCorrelation(4)
R.MetricUseFixedImageGradientFilterOff()
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 ):
```

(continues on next page)

(continued from previous page)

```
    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" )
```

5.17 Image Registration Method Exhaustive

5.17.1 Overview

5.17.2 Code

Python

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

"""
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) :
```

(continues on next page)

(continued from previous page)

```
    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:
    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])
```

(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(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" )
```

5.18 Integration with ITK

5.18.1 Overview

5.18.2 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
```

(continues on next page)

(continued from previous page)

```
  //
  if( argc < 7 )
    {
    std::cerr << "Missing Parameters " << std::endl;
    std::cerr << "Usage: " << argv[0];
    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 );
```

(continues on next page)

(continued from previous page)

```
  image = caster.Execute( image );

  //
  // Extract the itk image from the SimpleITK image
  //
  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)
```

(continues on next page)

(continued from previous page)

```
    {
    std::vector<unsigned int> seed;
    seed.push_back(atoi(argv[i]));
    seed.push_back(atoi(argv[i+1]));
    segmentationFilter.AddSeed(seed);
    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;
}
```

5.19 N4 Bias Field Correction

5.19.1 Overview

5.19.2 Code

Python

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

from __future__ import print_function

import SimpleITK as sitk
import sys
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:
```

(continues on next page)

(continued from previous page)

```
    maskImage = sitk.ReadImage( sys.argv[4] )
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();

numberFilltingLevels = 4

if len ( sys.argv ) > 6:
    numberFilltingLevels = int( sys.argv[6] )

if len ( sys.argv ) > 5:
    corrector.SetMaximumNumberOfIterations( [ int( sys.argv[5] ) ]␣
↪*numberFilltingLevels  )



output = corrector.Execute( inputImage, maskImage )

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


if ( not "SITK_NOSHOW" in os.environ ):
    sitk.Show( output, "N4 Corrected" )
```

5.20 Simple Gaussian

5.20.1 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.

5.20.2 Code

C#

```
using System;
using itk.simple;

namespace itk.simple.examples {
```

(continues on next page)

(continued from previous page)

```
    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 ) {
    std::cerr << "Usage: " << argv[0] << " <input> <sigma> <output>\n";
    return 1;
  }
```

(continues on next page)

(continued from previous page)

```
  // 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);

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

(continues on next page)

(continued from previous page)

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

pixelID = image.GetPixelID()
```

(continues on next page)

(continued from previous page)

```
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 )
end

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

(continues on next page)

(continued from previous page)

```
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
```

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

6.1 How to Cite

If you find SimpleITK useful in your research, support our efforts by citing the relevant publication(s):

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, 2017.

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.