SimpleITK Documentation

Release 2.0rc2

Insight Software Consortium

Mar 14, 2024

TABLE OF CONTENTS

1 About 3

2 Getting Started 9

3 Fundamental Concepts 13

4 Registration Overview 19

5 Common Conventions 25

6 Reading and Writing for Images and Transforms 27

7 SimpleITK Filters 35

8 Building SimpleITK 45

9 Setting Up Eclipse and Visual Studio 57

10 Tutorials and Courses 65

11 Frequently Asked Questions 67

12 Migration Guide 2.0 75

13 Developer 81

14 Examples 89

15 Relevant Resources 271

16 How to Cite 273

SimpleITK is a simplified, open source, interface to the Insight 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

ONE

ABOUT

SimpleITK is a simplified programming interface to the algorithms and data structures of the Insight Toolkit (ITK). It supports bindings for multiple programming languages including C++, Python, R, Java, C#, Lua, Ruby and TCL. These bindings enable scientists to develop image analysis workflows in the programming language they are most familiar with. The toolkit supports more than 15 different image file formats, provides over 280 image analysis filters, and implements a unified interface to the ITK intensity-based registration framework.

1.1 History

SimpleITK was created as part of a concerted effort to simplify the use of the Insight Toolkit, making it more accessible to a wider audience.

The initial funding for the toolkit was provided by the United States National Library of Medicine (NLM) under the American Recovery and Reinvestment Act (ARRA), with the initial version of the toolkit developed as a collaboration between The Mayo Clinic, Kitware Inc, The University of Iowa and NLM's intramural research program. The first major release of the toolkit was announced in April-May 2017.

Between 2013 and 2019, development was supported by the NLM intramural research program with collaborators at The University of Iowa and Monash University.

Since 2019, SimpleITK development is supported by the Office of Cyber Infrastructure and Computational Biology at the National Institute of Allergy and Infectious Diseases.

The toolkit release history is available on GitHub.

1.2 License

```
                                 Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

   TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

   1. Definitions.

      "License" shall mean the terms and conditions for use, reproduction,
      and distribution as defined by Sections 1 through 9 of this document.

      "Licensor" shall mean the copyright owner or entity authorized by
```

(continues on next page)

(continued from previous page)

```
      the copyright owner that is granting the License.

      "Legal Entity" shall mean the union of the acting entity and all
      other entities that control, are controlled by, or are under common
      control with that entity. For the purposes of this definition,
      "control" means (i) the power, direct or indirect, to cause the
      direction or management of such entity, whether by contract or
      otherwise, or (ii) ownership of fifty percent (50%) or more of the
      outstanding shares, or (iii) beneficial ownership of such entity.

      "You" (or "Your") shall mean an individual or Legal Entity
      exercising permissions granted by this License.

      "Source" form shall mean the preferred form for making modifications,
      including but not limited to software source code, documentation
      source, and configuration files.

      "Object" form shall mean any form resulting from mechanical
      transformation or translation of a Source form, including but
      not limited to compiled object code, generated documentation,
      and conversions to other media types.

      "Work" shall mean the work of authorship, whether in Source or
      Object form, made available under the License, as indicated by a
      copyright notice that is included in or attached to the work
      (an example is provided in the Appendix below).

      "Derivative Works" shall mean any work, whether in Source or Object
      form, that is based on (or derived from) the Work and for which the
      editorial revisions, annotations, elaborations, or other modifications
      represent, as a whole, an original work of authorship. For the purposes
      of this License, Derivative Works shall not include works that remain
      separable from, or merely link (or bind by name) to the interfaces of,
      the Work and Derivative Works thereof.

      "Contribution" shall mean any work of authorship, including
      the original version of the Work and any modifications or additions
      to that Work or Derivative Works thereof, that is intentionally
      submitted to Licensor for inclusion in the Work by the copyright owner
      or by an individual or Legal Entity authorized to submit on behalf of
      the copyright owner. For the purposes of this definition, "submitted"
      means any form of electronic, verbal, or written communication sent
      to the Licensor or its representatives, including but not limited to
      communication on electronic mailing lists, source code control systems,
      and issue tracking systems that are managed by, or on behalf of, the
      Licensor for the purpose of discussing and improving the Work, but
      excluding communication that is conspicuously marked or otherwise
      designated in writing by the copyright owner as "Not a Contribution."

      "Contributor" shall mean Licensor and any individual or Legal Entity
      on behalf of whom a Contribution has been received by Licensor and
      subsequently incorporated within the Work.
```

(continues on next page)

(continued from previous page)

```
2. Grant of Copyright License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   copyright license to reproduce, prepare Derivative Works of,
   publicly display, publicly perform, sublicense, and distribute the
   Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   (except as stated in this section) patent license to make, have made,
   use, offer to sell, sell, import, and otherwise transfer the Work,
   where such license applies only to those patent claims licensable
   by such Contributor that are necessarily infringed by their
   Contribution(s) alone or by combination of their Contribution(s)
   with the Work to which such Contribution(s) was submitted. If You
   institute patent litigation against any entity (including a
   cross-claim or counterclaim in a lawsuit) alleging that the Work
   or a Contribution incorporated within the Work constitutes direct
   or contributory patent infringement, then any patent licenses
   granted to You under this License for that Work shall terminate
   as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:

   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
       attribution notices from the Source form of the Work,
       excluding those notices that do not pertain to any part of
       the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its
       distribution, then any Derivative Works that You distribute must
       include a readable copy of the attribution notices contained
       within such NOTICE file, excluding those notices that do not
       pertain to any part of the Derivative Works, in at least one
       of the following places: within a NOTICE text file distributed
       as part of the Derivative Works; within the Source form or
       documentation, if provided along with the Derivative Works; or,
       within a display generated by the Derivative Works, if and
       wherever such third-party notices normally appear. The contents
       of the NOTICE file are for informational purposes only and
```

(continues on next page)

(continued from previous page)

```
        do not modify the License. You may add Your own attribution
        notices within Derivative Works that You distribute, alongside
        or as an addendum to the NOTICE text from the Work, provided
        that such additional attribution notices cannot be construed
        as modifying the License.

    You may add Your own copyright statement to Your modifications and
    may provide additional or different license terms and conditions
    for use, reproduction, or distribution of Your modifications, or
    for any such Derivative Works as a whole, provided Your use,
    reproduction, and distribution of the Work otherwise complies with
    the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
    any Contribution intentionally submitted for inclusion in the Work
    by You to the Licensor shall be under the terms and conditions of
    this License, without any additional terms or conditions.
    Notwithstanding the above, nothing herein shall supersede or modify
    the terms of any separate license agreement you may have executed
    with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
    names, trademarks, service marks, or product names of the Licensor,
    except as required for reasonable and customary use in describing the
    origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
    agreed to in writing, Licensor provides the Work (and each
    Contributor provides its Contributions) on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
    implied, including, without limitation, any warranties or conditions
    of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
    PARTICULAR PURPOSE. You are solely responsible for determining the
    appropriateness of using or redistributing the Work and assume any
    risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
    whether in tort (including negligence), contract, or otherwise,
    unless required by applicable law (such as deliberate and grossly
    negligent acts) or agreed to in writing, shall any Contributor be
    liable to You for damages, including any direct, indirect, special,
    incidental, or consequential damages of any character arising as a
    result of this License or out of the use or inability to use the
    Work (including but not limited to damages for loss of goodwill,
    work stoppage, computer failure or malfunction, or any and all
    other commercial damages or losses), even if such Contributor
    has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
    the Work or Derivative Works thereof, You may choose to offer,
    and charge a fee for, acceptance of support, warranty, indemnity,
    or other liability obligations and/or rights consistent with this
```

(continues on next page)

(continued from previous page)

```
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

   To apply the Apache License to your work, attach the following
   boilerplate notice, with the fields enclosed by brackets "[]"
   replaced with your own identifying information. (Don't include
   the brackets!)  The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
   same "printed page" as the copyright notice for easier
   identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

CHAPTER

TWO

GETTING STARTED

Welcome to SimpleITK!

SimpleITK provides a simplified interface to ITK in a variety of languages. A user can either download pre-built binaries, if they are available for the desired platform and language, or SimpleITK can be *built from the source code*.

In the following sections we describe how to install SimpleITK and auxiliary software tools.

After installing SimpleITK, we recommend going over the *latest tutorial* and skimming the Doxygen pages for more information.

2.1 Downloading the binaries

One of the great advantages of SimpleITK is that (typically) the user does not have to build it — he or she can simply download the binaries and get started right away!

Currently, **Python** binaries are available on Microsoft Windows, GNU Linux and Mac OS X. **C# and Java** binaries are available for Windows. **R** binaries are not available but a convenient devtools based installer for Linux and Mac is.

2.1.1 Python binary files

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

Wheels for Generic Python Distribution

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

```
pip install SimpleITK
```

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

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

Conda-based distributions (Anaconda, Miniconda)

From the command line prompt, execute:

```
conda install -c conda-forge simpleitk
```

This will install the latest version of SimpleITK that is compatible with the package versions already installed in your environment. To install the latest version of SimpleITK and update all version dependencies, execute:

```
conda install --update-deps -c conda-forge simpleitk
```

Finally, to install a specific version, for example 2.2.1, execute:

```
conda install -c conda-forge simpleitk=2.2.1
```

2.1.2 C# binary files

Binaries for select C# platforms can be found on SimpleITK's GitHub releases under the appropriate version. Installing the library should only involve importing the unzipped files into the C# environment. The files have the following naming convention:

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

e.g.

`SimpleITK-1.2.4-CSharp-win64-x64.zip`

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

2.1.3 Java binary files

Binaries for select Java platforms can be found on SimpleITK's GitHub releases page under the appropriate version. Installation instructions are available at *setup SimpleITK with Java*.

2.1.4 R binary files

R binaries are currently only available for Linux and Mac.

SimpleITK/R is installed using a devtools based installer (GitHub Repository). Strictly speaking, this is not a binary distribution, but it is a convenient way of automatically compiling and installing on a system.

This installation requires R devtools, CMake, git, and a compiler in the path.

```
devtools::install_github("SimpleITK/SimpleITKRInstaller")
```

2.1.5 Latest binaries

If the latest SimpleITK bug fixes and features are required then pre-release binaries are available. The pre-release binaries are built from the actively developed master branch. The SimpleITK packages are available on Github releases under the "latest" tag. The packages include Python, CSharp and Java. To upgrade to the latest pre-release Python binary package run:

```
pip install --upgrade --pre SimpleITK --find-links https://github.com/SimpleITK/
↪SimpleITK/releases/tag/latest
```

2.2 Recommended Software

2.2.1 Fiji (Fiji is Just ImageJ)

SimpleITK has a built in function, itk::simple::Show(), which can be used for viewing images in an interactive session. By default this Show function searches for an installed Fiji to display images. If Fiji is not found, then it searches for ImageJ. Fiji/ImageJ were chosen because they can handle all the image types that SimpleITK supports, including 3D vector images with n components per pixel.

The Show function searches for Fiji/ImageJ in a number of standard locations. The exact locations depend on the system type (Linux, Mac or Windows). Enabling the **"debugOn**" option to **"Show**", allows the user to see where SimpleITK is searching for Fiji (or other viewing application) and the actual command used to launch the viewing application.

Show is a functional interface to the ImageViewer class. Other viewing applications can be configured using an ImageViewer object, see the following section of the *Frequently Asked Questions*:

> *Can I use another image file viewer beside Fiji?*.

2.2.2 IPython and Jupyter

For python users, IPython with Jupyter is a terrific environment to perform interactive computing for image processing. The addition of the numpy and scipy packages creates a powerful interactive development environment.

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

CHAPTER

THREE

FUNDAMENTAL CONCEPTS

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

3.1 Images

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

SimpleITK images are multi-dimensional (the default configuration includes images from 2D upto 5D) and can be a scalar, labelmap (scalar with run length encoding), complex value or have an arbitrary number of scalar channels (also known as a vector image). The region in physical space which an image occupies is defined by the image's:

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

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

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

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

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

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

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

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

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

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

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

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

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

1. How do you access the pixel values in an image?
   a. In dynamically typed languages such as Python and R use a type agnostic function:

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

   b. In statically typed languages such as C# and C++ use type specific functions:

   ```
   image.GetPixelAsUInt8( {0, 0} )
   ```

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

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

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

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

3.1.1 Channels

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

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

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

3.1.2 Additional Resources

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

3.2 Transforms

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

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

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

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

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

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

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

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

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

```
composite_transform = CompositeTransform([T0, T1])
composite_transform.AddTransform(T2)
```

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

3.2.1 Additional Resources

1. The API for the SimpleITK transformation classes is available in Doxygen format:
   - 2D or 3D translation.
   - VersorTransform.
   - Euler2DTransform and Euler3DTransform.
   - Similarity2DTransform and Similarity3DTransform.
   - 2D or 3D ScaleTransform.
   - ScaleVersor3DTransform.
   - ScaleSkewVersor3DTransform.
   - ComposeScaleSkewVersor3DTransform.
   - 2D or 3D AffineTransform.
   - 2D or 3D BSplineTransform.
   - 2D or 3D DisplacementFieldTransform.
   - CompositeTransform.
   - Transform.
2. To really understand the structure of SimpleITK transforms and how to work with them, we recommend some hands-on interaction using the SimpleITK Jupyter notebooks (Python and R only).

3.3 Resampling

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

Generally speaking, resampling in SimpleITK involves four components:

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

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

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

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

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

3.3.1 Common Errors

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

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

3.3.2 Additional Resources

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

CHAPTER

FOUR

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.

4.1 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.

4.2 Initialization and Center of Rotation

The task of registration is formulated using non-linear optimization which requires an initial estimate. The two most common initialization approaches are (1) Use the identity transform (a.k.a. forgot to initialize). (2) Align the physical centers of the two images (see CenteredTransformInitializerFilter). If after initialization there is no overlap between the images, registration will fail. The closer the initialization transformation is to the actual transformation, the higher the probability of convergence to the correct solution.

If your registration involves the use of a global domain transform (*described here*), you should also set an appropriate center of rotation. In many cases you want the center of rotation to be the physical center of the fixed image (the CenteredTransformCenteredTransformInitializerFilter ensures this). This is of significant importance for registration convergence due to the non-linear nature of rotation. When the center of rotation is far from our physical region of interest (ROI), a small rotational angle results in a large displacement. Think of moving the pivot/fulcrum point of a lever. For the same rotation angle, the farther you are from the fulcrum the larger the displacement. For numerical stability we do not want our computations to be sensitive to very small variations in the rotation angle, thus the ideal center of rotation is the point which minimizes the distance to the farthest point in our ROI:

$$p_{center} = \arg\min_{p_{rotation}} dist(p_{rotation}, \{p_{roi}\})$$

Without additional knowledge we can only assume that the ROI is the whole fixed image. If your ROI is only in a sub region of the image, a more appropriate point would be the center of the oriented bounding box of that ROI.

4.3 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.

4.3.1 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.
- ComposeScaleSkewVersor3DTransform.
- 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.

4.3.2 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.

4.3.3 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).

4.3.4 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.

4.3.5 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_opt - 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:

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

Multi Resolution Framework

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

Sampling

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

The registration framework supports three sampling strategies:

1. NONE - use all voxels, sampled points are the voxel centers.
2. REGULAR - sample every n-th voxel while traversing the image in scan-line order, then within each voxel randomly perturb from center.
3. RANDOM - sample image voxels with replacement using a uniform distribution, then within each voxel randomly perturb from center.

When using the REGULAR or RANDOM sampling strategies, running the same registration code multiple times will yield different results. To remove the randomness, set the pseudo-random number generator's seed in the SetMetricSamplingPercentage method to a constant. The default seed value is wall clock time.

Note that using the RANDOM sampling strategy with a 100% sampling rate is not equivalent to using the sampling strategy of NONE. Given an image with N voxels, the former randomly selects N voxels with repetition and perturbs the points within each voxel, the latter uses the centers of all N voxels. Thus, for repeated random sampling with 100% rate, different samples are produced and likely none of them is of the centers of all N voxels.

Combining a mask with sampling is done using a rejection approach. First a sample is generated and then it is accepted or rejected if it is inside or outside the mask. This may cause problems when the mask region occupies a very small region in the original image. Because the sampling only discards data,the sample rate may be reduced from the requested one. For some similarity metrics (e.g. mutual information) this can result in an insufficient number of samples for metric value computation, leading to registration failure. Other metrics are more robust to small sample sizes (e.g. mean squares), but they all suffer from it. In such cases it is better to use a cropped version of the image for registration, possibly the mask's bounding box, instead of the original image with a mask.

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.

Reproducibility

Generally speaking, repeated registrations of the same datasets will yield slightly different results. This is associated with the registration framework's similarity metric computation implementation which utilizes both randomization and multi-threading.

The primary source of variability is the use of randomization. To eliminate randomization variability, set the seed parameter for the SetMetricSamplingPercentage to a fixed value (e.g. 42).

The secondary, and much more minor, source of variability has to do with the multithreading implementation, different order of operations in each registration run. To eliminate multithreading variability, set the number of threads to one via the SetGlobalDefaultNumberOfThreads method. From a practical standpoint, in the tradeoff between full reproducibility and computational efficiency with minor variability in results, computational efficiency is most often more important.

CHAPTER

FIVE

COMMON CONVENTIONS

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

5.1 Naming

The object oriented interface of process objects are named based on the specific operation they perform on their input image(s) with a suffix of **ImageFilter**, the ITK convention. For example, AbsImageFilter, BinaryMinMaxCurvatureFlowImageFilter etc. For almost all process objects there is also a procedural interface. This interface provides one or more functions which allow setting parameter values and invoking the operation in one call. This results in concise code and is useful if the filter is only applied once. These functions use the same naming as the object oriented interface only without the suffix. That is, the corresponding procedural methods to the object oriented interface examples above are Abs and BinaryMinMaxCurvatureFlow.

Note that the complete/detailed documentation appears in the doxygen of the object oriented interface.

5.2 Dimensional Vectors

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

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

5.3 Image Access

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

5.4 Mask Image Types and Default Values

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

These defaults are for filter which create masks such as thresholding, and certain other segmentation filters. Additionally, these are the defaults for the binary morphology filters, so that they can easily be applied after segmentation. This choice makes many mask manipulations easier. For example, "masking" an image (scalar or vector) is simply a matter of multiplying by a mask an image. Additionally, the set of $\{0, 1\}$ pixels is closed under the logical Boolean operators.

5.5 Order of Procedural Parameters

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

5.6 Matrices as Parameters

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

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

5.7 Image Regions as Index and Size

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

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

5.8 Images As Parameters

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

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

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

CHAPTER

SIX

READING AND WRITING FOR IMAGES AND TRANSFORMS

6.1 Images

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

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

A read and write example using SimpleITK's ImageFileReader and ImageFileWriter classes:

C#

C++

Java

Lua

Python

R

Ruby

Tcl

```
ImageFileReader reader = new ImageFileReader();
reader.SetImageIO("PNGImageIO");
reader.SetFileName(inputImageFileName);
Image image = reader.Execute();

ImageFileWriter writer = new ImageFileWriter();
writer.SetFileName(outputImageFileName);
writer.Execute(image);
```

```
itk::simple::ImageFileReader reader;
reader.SetImageIO("PNGImageIO");
reader.SetFileName(inputImageFileName);

itk::simple::Image image;
image = reader.Execute();

itk::simple::ImageFileWriter writer;
writer.SetFileName(outputImageFileName);
```

```
ImageFileReader reader = new ImageFileReader();
reader.setImageIO("PNGImageIO");
reader.setFileName(inputImageFileName);
Image image = reader.execute();

ImageFileWriter writer = new ImageFileWriter();
writer.setFileName(outputImageFileName);
writer.execute(image);
```

```
require "SimpleITK"

reader = SimpleITK.ImageFileReader()
reader:SetImageIO("PNGImageIO")
reader:SetFileName(inputImageFileName)
image = reader:Execute();

writer = SimpleITK.ImageFileWriter()
writer:SetFileName(outputImageFileName)
writer:Execute(image);
```

```
import SimpleITK as sitk

reader = sitk.ImageFileReader()
```

(continues on next page)

(continued from previous page)

```
  reader.SetImageIO("PNGImageIO")
  reader.SetFileName(inputImageFileName)
  image = reader.Execute()

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

```
library(SimpleITK)

reader <- ImageFileReader()
reader$SetImageIO("PNGImageIO")
reader$SetFileName(inputImageFileName)
image <- reader$Execute()

writer <- ImageFileWriter()
writer$SetFileName(outputImageFileName)
writer$Execute(image)
```

```
require 'simpleitk'

reader = Simpleitk::ImageFileReader.new
reader.set_image_io("PNGImageIO")
reader.set_file_name(inputImageFileName)
image = reader.execute

writer = Simpleitk::ImageFileWriter.new
writer.set_file_name outputImageFileName
writer.execute image
```

```
ImageFileReader reader
reader SetImageIO "PNGImageIO"
reader SetFileName $inputImageFileName
set image [ reader Execute ]

ImageFileWriter writer
writer SetFileName $outputImageFileName
writer Execute $image

reader -delete
writer -delete
$image -delete
```

The above example specifies using the PNGImageIO to read the file. If that line is omitted, SimpleITK would determine which IO to use automatically, based on the file name's suffix and/or the file's header.

A more compact example using SimpleITK's procedural interface:

C#

C++

Java

Lua

Python

R

Ruby

Tcl

```
  Image image = sitk.ReadImage(inputImageFileName, PixelIDValueEnum.sitkUnknown,
↪"PNGImageIO");
  sitk.WriteImage(image, outputImageFileName);
```

```
  itk::simple::Image image;
  image = itk::simple::ReadImage(inputImageFileName, itk::simple::sitkUnknown,
↪"PNGImageIO");
```

```
  Image image = SimpleITK.readImage(inputImageFileName, PixelIDValueEnum.sitkUnknown,
↪"PNGImageIO");
  SimpleITK.writeImage(image, outputImageFileName);
```

```
  require "SimpleITK"

  image = SimpleITK.ReadImage(inputImageFileName, SimpleITK.sitkUnknown, "PNGImageIO")
  SimpleITK.WriteImage(image, outputImageFileName)
```

```

    import SimpleITK as sitk

    image = sitk.ReadImage(inputImageFileName, imageIO="PNGImageIO")
```

```
  library(SimpleITK)

  image <- ReadImage(inputImageFileName)
  WriteImage(image, outputImageFileName)
```

```
  require 'simpleitk'

  image = Simpleitk::read_image(inputImageFileName, Simpleitk::SitkUnknown, "PNGImageIO")
  Simpleitk::write_image(image, outputImageFileName)
```

```
  global sitkUnknown
  set image [ ReadImage $inputImageFileName $sitkUnknown "PNGImageIO" ]
  WriteImage $image $outputImageFileName

  $image -delete
```

Similarly, if the imageIO parameter is omitted, SimpleITK will determine which IO to use automatically.

Finally, one can also specify the desired pixel type for the returned image, regardless of the underlying pixel type. This is done via the `SetOutputPixelType` of the `ImageFileReader` or setting the `ReadImage`'s `outputPixelType` parameter.

When reading a scalar image and specifying a different scalar type from the underlying one, the returned image's pixel type will be a simple casting of the actual type. This casting can result in loss of data when performing a narrowing

conversion. For example, if the underlying pixel type is `sitk.sitkFloat32` and the specified pixel type is `sitk.sitkUInt8`. Explicitly specifying a pixel type is commonly done in conjunction with the SimpleITK registration framework. This framework only accepts images with float pixel types. Therefore, it is common to read the images and specify their pixel type as `sitk.sitkFloat32` or `sitk.sitkFloat64`.

When reading a vector image and specifying the pixel type as a scalar, the returned pixel values are computed based on the number of channels in the original image. This conversion makes several assumptions, three channels mean an RGB image, four channels an RGBA image. These assumptions are useful, but may not be what you need if the original image was not in these color-spaces (e.g. HSV).

For those looking for further details, the ITK class which deals with these conversions is ConvertPixelBuffer.

6.2 Transformations

In SimpleITK, transformation files can be written in several different formats. Just as there are numerous IOs for images, there are several for transforms, including TxtTransformIO, MINCTransformIO, HDF5TransformIO, and MatlabTransformIO (although this list can be extended as well). These support a variety of file formats, including .txt, .tfm, .xfm, .hdf and .mat.

Because of the size of displacement fields, writing them may require more careful attention. To save a displacement field we recommend using one of the binary transformation file formats (e.g. .hdf, .mat). Saving it in a text based format results in significantly larger files and longer IO runtimes. Another option is to save the displacement field found in a DisplacementFieldTransform object as an image (.nrrd, .nhdr, .mha, .mhd, .nii, .nii.gz).

Here is a simple example of creating a transformation, writing it to a file, reading it back, and then comparing the results.

C#

C++

Java

Lua

Python

R

Ruby

Tcl

```
Euler2DTransform basic_transform = new Euler2DTransform();
VectorDouble trans = new VectorDouble( new double[] {2.0, 3.0} );
basic_transform.SetTranslation(trans);

sitk.WriteTransform(basic_transform, "euler2D.tfm");

Transform read_result = sitk.ReadTransform("euler2D.tfm");

Debug.Assert( basic_transform.GetName() != read_result.GetName() );
```

```
itk::simple::Euler2DTransform basic_transform;
basic_transform.SetTranslation(std::vector<double> {2.0, 3.0});

itk::simple::WriteTransform(basic_transform, "euler2D.tfm");
```

(continues on next page)

(continued from previous page)

```
itk::simple::Transform read_result = itk::simple::ReadTransform("euler2D.tfm");
```

```
Euler2DTransform basic_transform = new Euler2DTransform();
double[] t = {2.0, 3.0};
VectorDouble trans = new VectorDouble(t);
basic_transform.setTranslation(trans);

SimpleITK.writeTransform(basic_transform, "euler2D.tfm");
Transform read_result = SimpleITK.readTransform("euler2D.tfm");

assert basic_transform.getClass() != read_result.getClass();
```

```
require "SimpleITK"

basic_transform = SimpleITK.Euler2DTransform()
trans = SimpleITK.VectorDouble()
trans:push_back(2.0)
trans:push_back(3.0)
basic_transform:SetTranslation(trans)

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

assert(read_result:GetName() ~= basic_transform:GetName(), "type error")
```

```
  import SimpleITK as sitk

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

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

```
library(SimpleITK)

basic_transform <- Euler2DTransform()
basic_transform$SetTranslation(c(2,3))

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

stopifnot(read_result$GetName() != basic_transform$GetName())
```

```
require 'simpleitk'

basic_transform = Simpleitk::Euler2DTransform.new
trans = Simpleitk::VectorDouble.new
trans << 2.0
```

(continues on next page)

(continued from previous page)

```
trans << 3.0
basic_transform.set_translation(trans)

Simpleitk::write_transform(basic_transform, 'euler2D.tfm')
read_result = Simpleitk::read_transform('euler2D.tfm')

raise "This shouldn't happen." unless basic_transform.class != read_result.class
```

```
Euler2DTransform basic_transform
set trans { 2.0 3.0 }
basic_transform SetTranslation $trans

WriteTransform basic_transform "euler2D.tfm"
set read_result [ ReadTransform "euler2D.tfm" ]

if { [basic_transform GetName] == [$read_result GetName] } {
  error "This should not happen" 1
}

basic_transform -delete
$read_result -delete
```

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

CHAPTER

SEVEN

SIMPLEITK FILTERS

Filter Name	Brief Description
AbsImageFilter	Computes the absolute value of each pixel.
AbsoluteValueDifferenceImage-Filter	Implements pixel-wise the computation of absolute value difference.
AcosImageFilter	Computes the inverse cosine of each pixel.
AdaptiveHistogramEqualization-ImageFilter	Power Law Adaptive Histogram Equalization.
AddImageFilter	Pixel-wise addition of two images.
AdditiveGaussianNoiseImage-Filter	Alter an image with additive Gaussian white noise.
AggregateLabelMapFilter	Collapses all labels into the first label.
AndImageFilter	Implements the AND bitwise operator pixel-wise between two images.
AntiAliasBinaryImageFilter	A method for estimation of a surface from a binary volume.
ApproximateSignedDis-tanceMapImageFilter	Create a map of the approximate signed distance from the boundaries of a binary image.
AsinImageFilter	Computes the sine of each pixel.
Atan2ImageFilter	Computes two argument inverse tangent.
AtanImageFilter	Computes the one-argument inverse tangent of each pixel.
BSplineDecompositionImage-Filter	Calculates the B-Spline coefficients of an image. Spline order may be from 0 to 5.
BilateralImageFilter	Blurs an image while preserving edges.
BinShrinkImageFilter	Reduce the size of an image by an integer factor in each dimension while performing averaging of an input neighborhood.
BinaryClosingByReconstruction-ImageFilter	binary closing by reconstruction of an image.
BinaryContourImageFilter	Labels the pixels on the border of the objects in a binary image.
BinaryDilateImageFilter	Fast binary dilation of a single intensity value in the image.
BinaryErodeImageFilter	Fast binary erosion of a single intensity value in the image.
BinaryFillholeImageFilter	Remove holes not connected to the boundary of the image.
BinaryGrindPeakImageFilter	Remove the objects not connected to the boundary of the image.
BinaryImageToLabelMapFilter	Label the connected components in a binary image and produce a collection of label objects.
BinaryMagnitudeImageFilter	Computes the square root of the sum of squares of corresponding input pix-els.
BinaryMedianImageFilter	Applies a version of the median filter optimized for binary images.
BinaryMinMaxCurvatureFlowIm-ageFilter	Denoise a binary image using min/max curvature flow.

continues on next page

Table 1 – continued from previous page

Filter Name	Brief Description
BinaryMorphologicalClosingImageFilter	binary morphological closing of an image.
BinaryMorphologicalOpeningImageFilter	binary morphological opening of an image.
BinaryNotImageFilter	Implements the BinaryNot logical operator pixel-wise between two images.
BinaryOpeningByReconstructionImageFilter	binary morphological closing of an image.
BinaryProjectionImageFilter	Binary projection.
BinaryPruningImageFilter	This filter removes "spurs" of less than a certain length in the input image.
BinaryReconstructionByDilationImageFilter	binary reconstruction by dilation of an image
BinaryReconstructionByErosionImageFilter	binary reconstruction by erosion of an image
BinaryThinningImageFilter	This filter computes one-pixel-wide edges of the input image.
BinaryThresholdImageFilter	Binarize an input image by thresholding.
BinaryThresholdProjectionImageFilter	BinaryThreshold projection.
BinomialBlurImageFilter	Performs a separable blur on each dimension of an image.
BitwiseNotImageFilter	Implements pixel-wise generic operation on one image.
BlackTopHatImageFilter	Black top hat extracts local minima that are smaller than the structuring element.
BoundedReciprocalImageFilter	Computes 1/(1+x) for each pixel in the image.
BoxMeanImageFilter	Implements a fast rectangular mean filter using the accumulator approach.
BoxSigmaImageFilter	Implements a fast rectangular sigma filter using the accumulator approach.
CannyEdgeDetectionImageFilter	This filter is an implementation of a Canny edge detector for scalar-valued images.
CannySegmentationLevelSetImageFilter	Segments structures in images based on image features derived from pseudo-canny-edges.
ChangeLabelImageFilter	Change Sets of Labels.
ChangeLabelLabelMapFilter	Replace the label Ids of selected LabelObjects with new label Ids.
CheckerBoardImageFilter	Combines two images in a checkerboard pattern.
ClampImageFilter	Casts input pixels to output pixel type and clamps the output pixel values to a specified range.
ClosingByReconstructionImageFilter	Closing by reconstruction of an image.
CoherenceEnhancingDiffusionImageFilter	Coherence enhancing diffusion and edge enhancing diffusion.
CollidingFrontsImageFilter	Selects a region of space where two independent fronts run towards each other.
ComplexToImaginaryImageFilter	Computes pixel-wise the imaginary part of a complex image.
ComplexToModulusImageFilter	Computes pixel-wise the Modulus of a complex image.
ComplexToPhaseImageFilter	Computes pixel-wise the modulus of a complex image.
ComplexToRealImageFilter	Computes pixel-wise the real(x) part of a complex image.
ComposeImageFilter	ComposeImageFilter combine several scalar images into a multi-component image.
ConfidenceConnectedImageFilter	Segment pixels with similar statistics using connectivity.
ConnectedComponentImageFilter	Label the objects in a binary image.
ConnectedThresholdImageFilter	Label pixels that are connected to a seed and lie within a range of values.

continues on next page

Table 1 – continued from previous page

Filter Name	Brief Description
ConstantPadImageFilter	Increase the image size by padding with a constant value.
ConvolutionImageFilter	Convolve a given image with an arbitrary image kernel.
CosImageFilter	Computes the cosine of each pixel.
CropImageFilter	Decrease the image size by cropping the image by an itk::Size at both the upper and lower bounds of the largest possible region.
CurvatureAnisotropicDiffusionImageFilter	
CurvatureFlowImageFilter	Denoise an image using curvature driven flow.
CyclicShiftImageFilter	Perform a cyclic spatial shift of image intensities on the image grid.
DICOMOrientImageFilter	Permute axes and flip images as needed to obtain an approximation to the desired orientation.
DanielssonDistanceMapImageFilter	This filter computes the distance map of the input image as an approximation with pixel accuracy to the Euclidean distance.
DemonsRegistrationFilter	Deformably register two images using the demons algorithm.
DerivativeImageFilter	Computes the directional derivative of an image. The directional derivative at each pixel location is computed by convolution with a derivative operator of user-specified order.
DiffeomorphicDemonsRegistrationFilter	Deformably register two images using a diffeomorphic demons algorithm.
DilateObjectMorphologyImageFilter	dilation of an object in an image
DiscreteGaussianDerivativeImageFilter	Calculates image derivatives using discrete derivative gaussian kernels. This filter calculates Gaussian derivative by separable convolution of an image and a discrete Gaussian derivative operator (kernel).
DiscreteGaussianImageFilter	Blurs an image by separable convolution with discrete gaussian kernels. This filter performs Gaussian blurring by separable convolution of an image and a discrete Gaussian operator (kernel).
DisplacementFieldJacobianDeterminantFilter	Computes a scalar image from a vector image (e.g., deformation field) input, where each output scalar at each pixel is the Jacobian determinant of the vector field at that location. This calculation is correct in the case where the vector image is a "displacement" from the current location. The computation for the jacobian determinant is: det[dT/dx] = det[I + du/dx].
DivideFloorImageFilter	Implements pixel-wise generic operation of two images, or of an image and a constant.
DivideImageFilter	Pixel-wise division of two images.
DivideRealImageFilter	Implements pixel-wise generic operation of two images, or of an image and a constant.
DoubleThresholdImageFilter	Binarize an input image using double thresholding.
EdgePotentialImageFilter	Computes the edge potential of an image from the image gradient.
EqualImageFilter	Implements pixel-wise generic operation of two images, or of an image and a constant.
ErodeObjectMorphologyImageFilter	Erosion of an object in an image.
ExpImageFilter	Computes the exponential function of each pixel.
ExpNegativeImageFilter	Computes the function exp(-K.x) for each input pixel.
ExpandImageFilter	Expand the size of an image by an integer factor in each dimension.
ExtractImageFilter	Decrease the image size by cropping the image to the selected region bounds.
FFTConvolutionImageFilter	Convolve a given image with an arbitrary image kernel using multiplication in the Fourier domain.

continues on next page

Table 1 – continued from previous page

Filter Name	Brief Description
FFTNormalizedCorrelationImageFilter	Calculate normalized cross correlation using FFTs.
FFTPadImageFilter	Pad an image to make it suitable for an FFT transformation.
FFTShiftImageFilter	Shift the zero-frequency components of a Fourier transform to the center of the image.
FastApproximateRankImageFilter	A separable rank filter.
FastMarchingBaseImageFilter	Apply the Fast Marching method to solve an Eikonal equation on an image.
FastMarchingImageFilter	Solve an Eikonal equation using Fast Marching.
FastMarchingUpwindGradientImageFilter	Generates the upwind gradient field of fast marching arrival times.
FastSymmetricForcesDemonsRegistrationFilter	Deformably register two images using a symmetric forces demons algorithm.
FlipImageFilter	Flips an image across user specified axes.
ForwardFFTImageFilter	Base class for forward Fast Fourier Transform .
GaborImageSource	Generate an n-dimensional image of a Gabor filter.
GaussianImageSource	Generate an n-dimensional image of a Gaussian.
GeodesicActiveContourLevelSetImageFilter	Segments structures in images based on a user supplied edge potential map.
GradientAnisotropicDiffusionImageFilter	
GradientImageFilter	Computes the gradient of an image using directional derivatives.
GradientMagnitudeImageFilter	Computes the gradient magnitude of an image region at each pixel.
GradientMagnitudeRecursiveGaussianImageFilter	Computes the Magnitude of the Gradient of an image by convolution with the first derivative of a Gaussian.
GradientRecursiveGaussianImageFilter	Computes the gradient of an image by convolution with the first derivative of a Gaussian.
GrayscaleConnectedClosingImageFilter	Enhance pixels associated with a dark object (identified by a seed pixel) where the dark object is surrounded by a brighter object.
GrayscaleConnectedOpeningImageFilter	Enhance pixels associated with a bright object (identified by a seed pixel) where the bright object is surrounded by a darker object.
GrayscaleDilateImageFilter	Grayscale dilation of an image.
GrayscaleErodeImageFilter	Grayscale erosion of an image.
GrayscaleFillholeImageFilter	Remove local minima not connected to the boundary of the image.
GrayscaleGeodesicDilateImageFilter	geodesic gray scale dilation of an image
GrayscaleGeodesicErodeImageFilter	geodesic gray scale erosion of an image
GrayscaleGrindPeakImageFilter	Remove local maxima not connected to the boundary of the image.
GrayscaleMorphologicalClosingImageFilter	gray scale dilation of an image
GrayscaleMorphologicalOpeningImageFilter	gray scale dilation of an image
GreaterEqualImageFilter	Implements pixel-wise generic operation of two images, or of an image and a constant.
GreaterImageFilter	Implements pixel-wise generic operation of two images, or of an image and a constant.
GridImageSource	Generate an n-dimensional image of a grid.
HConcaveImageFilter	Identify local minima whose depth below the baseline is greater than h.
HConvexImageFilter	Identify local maxima whose height above the baseline is greater than h.

continues on next page

Table 1 – continued from previous page

Filter Name	Brief Description
HMaximaImageFilter	Suppress local maxima whose height above the baseline is less than h.
HMinimaImageFilter	Suppress local minima whose depth below the baseline is less than h.
HalfHermitianToRealInverseFFTImageFilter	Base class for specialized complex-to-real inverse Fast Fourier Transform .
HausdorffDistanceImageFilter	Computes the Hausdorff distance between the set of non-zero pixels of two images.
HistogramMatchingImageFilter	Normalize the grayscale values for a source image by matching the shape of the source image histogram to a reference histogram.
HuangThresholdImageFilter	Threshold an image using the Huang Threshold.
IntensityWindowingImageFilter	Applies a linear transformation to the intensity levels of the input Image that are inside a user-defined interval. Values below this interval are mapped to a constant. Values over the interval are mapped to another constant.
IntermodesThresholdImageFilter	Threshold an image using the Intermodes Threshold.
InverseDeconvolutionImageFilter	The direct linear inverse deconvolution filter.
InverseDisplacementFieldImageFilter	Computes the inverse of a displacement field.
InverseFFTImageFilter	Base class for inverse Fast Fourier Transform .
InvertDisplacementFieldImageFilter	Iteratively estimate the inverse field of a displacement field.
InvertIntensityImageFilter	Invert the intensity of an image.
IsoContourDistanceImageFilter	Compute an approximate distance from an interpolated isocontour to the close grid points.
IsoDataThresholdImageFilter	Threshold an image using the IsoData Threshold.
IsolatedConnectedImageFilter	Label pixels that are connected to one set of seeds but not another.
IsolatedWatershedImageFilter	Isolate watershed basins using two seeds.
IterativeInverseDisplacementFieldImageFilter	Computes the inverse of a displacement field.
JoinSeriesImageFilter	Join N-D images into an (N+1)-D image.
KittlerIllingworthThresholdImageFilter	Threshold an image using the KittlerIllingworth Threshold.
LabelContourImageFilter	Labels the pixels on the border of the objects in a labeled image.
LabelImageToLabelMapFilter	convert a labeled image to a label collection image
LabelIntensityStatisticsImageFilter	a convenient class to convert a label image to a label map and valuate the statistics attributes at once
LabelMapContourOverlayImageFilter	Apply a colormap to the contours (outlines) of each object in a label map and superimpose it on top of the feature image.
LabelMapMaskImageFilter	Mask and image with a LabelMap .
LabelMapOverlayImageFilter	Apply a colormap to a label map and superimpose it on an image.
LabelMapToBinaryImageFilter	Convert a LabelMap to a binary image.
LabelMapToLabelImageFilter	Converts a LabelMap to a labeled image.
LabelMapToRGBImageFilter	Convert a LabelMap to a colored image.
LabelOverlapMeasuresImageFilter	Computes overlap measures between the set same set of labels of pixels of two images. Background is assumed to be 0.
LabelOverlayImageFilter	Apply a colormap to a label image and put it on top of the input image.
LabelSetDilateImageFilter	Class for binary morphological erosion of label images.
LabelSetErodeImageFilter	Class for binary morphological erosion of label images.
LabelShapeStatisticsImageFilter	Converts a label image to a label map and valuates the shape attributes.

continues on next page

Table 1 – continued from previous page

Filter Name	Brief Description
LabelStatisticsImageFilter	Given an intensity image and a label map, compute min, max, variance and mean of the pixels associated with each label or segment.
LabelToRGBImageFilter	Apply a colormap to a label image.
LabelUniqueLabelMapFilter	Make sure that the objects are not overlapping.
LabelVotingImageFilter	This filter performs pixelwise voting among an arbitrary number of input images, where each of them represents a segmentation of the same scene (i.e., image).
LandweberDeconvolutionImageFilter	Deconvolve an image using the Landweber deconvolution algorithm.
LaplacianImageFilter	
LaplacianRecursiveGaussianImageFilter	Computes the Laplacian of Gaussian (LoG) of an image.
LaplacianSegmentationLevelSetImageFilter	Segments structures in images based on a second derivative image features.
LaplacianSharpeningImageFilter	This filter sharpens an image using a Laplacian. LaplacianSharpening highlights regions of rapid intensity change and therefore highlights or enhances the edges. The result is an image that appears more in focus.
LessEqualImageFilter	Implements pixel-wise generic operation of two images, or of an image and a constant.
LessImageFilter	Implements pixel-wise generic operation of two images, or of an image and a constant.
LevelSetMotionRegistrationFilter	Deformably register two images using level set motion.
LiThresholdImageFilter	Threshold an image using the Li Threshold.
Log10ImageFilter	Computes the log10 of each pixel.
LogImageFilter	Computes the log() of each pixel.
MagnitudeAndPhaseToComplexImageFilter	Implements pixel-wise conversion of magnitude and phase data into complex voxels.
MaskImageFilter	Mask an image with a mask.
MaskNegatedImageFilter	Mask an image with the negation (or logical compliment) of a mask.
MaskedFFTNormalizedCorrelationImageFilter	Calculate masked normalized cross correlation using FFTs.
MaximumEntropyThresholdImageFilter	Threshold an image using the MaximumEntropy Threshold.
MaximumImageFilter	Implements a pixel-wise operator Max(a,b) between two images.
MaximumProjectionImageFilter	Maximum projection.
MeanImageFilter	Applies an averaging filter to an image.
MeanProjectionImageFilter	Mean projection.
MedianImageFilter	Applies a median filter to an image.
MedianProjectionImageFilter	Median projection.
MergeLabelMapFilter	Merges several Label Maps.
MinMaxCurvatureFlowImageFilter	Denoise an image using min/max curvature flow.
MinimumImageFilter	Implements a pixel-wise operator Min(a,b) between two images.
MinimumMaximumImageFilter	Computes the minimum and the maximum intensity values of an image.
MinimumProjectionImageFilter	Minimum projection.
MirrorPadImageFilter	Increase the image size by padding with replicants of the input image value.
ModulusImageFilter	Computes the modulus (x % dividend) pixel-wise.
MomentsThresholdImageFilter	Threshold an image using the Moments Threshold.
MorphologicalGradientImageFilter	gray scale dilation of an image

continues on next page

Table 1 – continued from previous page

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

continues on next page

Table 1 – continued from previous page

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

continues on next page

Table 1 – continued from previous page

Filter Name	Brief Description
SqrtImageFilter	Computes the square root of each pixel.
SquareImageFilter	Computes the square of the intensity values pixel-wise.
SquaredDifferenceImageFilter	Implements pixel-wise the computation of squared difference.
StandardDeviationProjectionImageFilter	Mean projection.
StatisticsImageFilter	Compute min, max, variance and mean of an Image .
StochasticFractalDimensionImageFilter	This filter computes the stochastic fractal dimension of the input image.
SubtractImageFilter	Pixel-wise subtraction of two images.
SumProjectionImageFilter	Sum projection.
SymmetricForcesDemonsRegistrationFilter	Deformably register two images using the demons algorithm.
TanImageFilter	Computes the tangent of each input pixel.
TernaryAddImageFilter	Pixel-wise addition of three images.
TernaryMagnitudeImageFilter	Compute the pixel-wise magnitude of three images.
TernaryMagnitudeSquaredImageFilter	Compute the pixel-wise squared magnitude of three images.
ThresholdImageFilter	Set image values to a user-specified value if they are below, above, or between simple threshold values.
ThresholdMaximumConnectedComponentsImageFilter	Finds the threshold value of an image based on maximizing the number of objects in the image that are larger than a given minimal size.
ThresholdSegmentationLevelSetImageFilter	Segments structures in images based on intensity values.
TikhonovDeconvolutionImageFilter	An inverse deconvolution filter regularized in the Tikhonov sense.
TileImageFilter	Tile multiple input images into a single output image.
TobogganImageFilter	toboggan image segmentation The Toboggan segmentation takes a gradient magnitude image as input and produces an (over-)segmentation of the image based on connecting each pixel to a local minimum of gradient. It is roughly equivalent to a watershed segmentation of the lowest level.
TransformToDisplacementFieldFilter	Generate a displacement field from a coordinate transform.
TriangleThresholdImageFilter	Threshold an image using the Triangle Threshold.
UnaryMinusImageFilter	Implements pixel-wise generic operation on one image.
UnsharpMaskImageFilter	Edge enhancement filter.
ValuedRegionalMaximaImageFilter	Transforms the image so that any pixel that is not a regional maxima is set to the minimum value for the pixel type. Pixels that are regional maxima retain their value.
ValuedRegionalMinimaImageFilter	Transforms the image so that any pixel that is not a regional minima is set to the maximum value for the pixel type. Pixels that are regional minima retain their value.
VectorConfidenceConnectedImageFilter	Segment pixels with similar statistics using connectivity.
VectorConnectedComponentImageFilter	A connected components filter that labels the objects in a vector image. Two vectors are pointing similar directions if one minus their dot product is less than a threshold. Vectors that are 180 degrees out of phase are similar. Assumes that vectors are normalized.
VectorIndexSelectionCastImageFilter	Extracts the selected index of the vector that is the input pixel type.

continues on next page

Table 1 – continued from previous page

Filter Name	Brief Description
VectorMagnitudeImageFilter	Take an image of vectors as input and produce an image with the magnitude of those vectors.
VotingBinaryHoleFillingImage-Filter	Fills in holes and cavities by applying a voting operation on each pixel.
VotingBinaryImageFilter	Applies a voting operation in a neighborhood of each pixel.
VotingBinaryIterativeHoleFill-ingImageFilter	Fills in holes and cavities by iteratively applying a voting operation.
WarpImageFilter	Warps an image using an input displacement field.
WhiteTopHatImageFilter	White top hat extracts local maxima that are larger than the structuring element.
WienerDeconvolutionImage-Filter	The Wiener deconvolution image filter is designed to restore an image convolved with a blurring kernel while keeping noise enhancement to a minimum.
WrapPadImageFilter	Increase the image size by padding with replicants of the input image value.
XorImageFilter	Computes the XOR bitwise operator pixel-wise between two images.
YenThresholdImageFilter	Threshold an image using the Yen Threshold.
ZeroCrossingBasedEdgeDetec-tionImageFilter	This filter implements a zero-crossing based edge detector.
ZeroCrossingImageFilter	This filter finds the closest pixel to the zero-crossings (sign changes) in a signed itk::Image .
ZeroFluxNeumannPadImage-Filter	Increase the image size by padding according to the zero-flux Neumann boundary condition.

CHAPTER

EIGHT

BUILDING SIMPLEITK

Pre-built binaries (see *Downloading the binaries*) are available in many cases so users do not need to build SimpleITK. However, there are several reasons a user might prefer to **build SimpleITK from source code**:

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

8.1 Prerequisites

To build SimpleITK you need:

- A recent version of CMake >= 3.16.3 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* to automatically download the matching version of ITK, SWIG, etc…
  - Windows users may prefer msysGit
- The Python interface requires Python version 3.5 or greater.
- It is recommended to have numpy installed when testing Python bindings.
- The R interface requires R version 3.3 or greater.

8.2 System Requirements

Building SimpleITK requires a significant amount of compute power, and we recommend a reasonably powerful, modern computer. We suggest at least 4 gigabytes (GB) of RAM plus an additional 2 GB per thread when building with multiple threads. We also suggest a system with at least 4 cores for multithreaded compilation.

Building a SuperBuild of SimpleITK's C++ core including all the testing and example code requires approximately 10 GB of disk space. Enabling the wrapping of Python or R can add 2 GB per language, and enabling the wrapping of all the languages SimpleITK supports, can bring the entire build folder to 16 GB.

8.3 Recipes / Formulas / Short cuts

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

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

- For **Python**: The scikit-build based distutils based setup.py frontend can be used to build, install, and package SimpleITK for Python.
- For the **Anaconda Python** distribution: The recipe and instructions for the SimpleITK build are in this GitHub repository.
- For the **R language** (version 3.3 or greater): A devtools installer and instructions are available from this GitHub repository.
- For the **Lua language** with the Luarocks module deployment system, a SimpleITK rockspec is available at the Luarocks repository or from this GitHub repository.

8.4 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://github.com/SimpleITK/SimpleITK.git
```

8.5 Building using SuperBuild

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

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

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

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

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

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

For **building SimpleITK on Windows** using the CMake GUI see:

8.5.1 Building SimpleITK on Windows

Path Length Issues on Windows

The Windows API, by default, has a Maximum Path Length Limitation of 260 characters. This limitation can cause problems when building SimpleITK using the Superbuild process.

For this reason we recommend placing SimpleITK's source and build directories at the top of the file system. For example, place the source code in `C:SimpleITK` and set the build directory to be `C:SimpleITK-build`.

Trying to build SimpleITK with too long a path length will result in an error like so:

```
CMake Error at CMakeLists.txt:109 (message):
  -- Configuring incomplete, errors occurred!
  See also "C:/Users/dchen/VeryVeryLongPathName/SimpleITK-build/ITK-build/CMakeFiles/
↪CMakeOutput.log".
    ITK source code directory path length is too long (55 > 50).Please move the
    ITK source code directory to a directory with a shorter path.


C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\MSBuild\Microsoft\VC\v160\
↪Microsoft.CppCommon.targets(238,5): error MSB8066: C
ustom build for 'C:\Users\dchen\VeryVeryLongPathName\SimpleITK-build\CMakeFiles\
↪49f9d16effa529c310c9be3182f62407\ITK-mkdir.rule;C:\Users\dch
en\VeryVeryLongPathName\SimpleITK-build\CMakeFiles\49f9d16effa529c310c9be3182f62407\ITK-
↪download.rule;C:\Users\dchen\VeryVeryLongPathName\Si
mpleITK-build\CMakeFiles\49f9d16effa529c310c9be3182f62407\ITK-update.rule;C:\Users\
↪dchen\VeryVeryLongPathName\SimpleITK-build\CMakeFiles\49f
9d16effa529c310c9be3182f62407\ITK-patch.rule;C:\Users\dchen\VeryVeryLongPathName\
↪SimpleITK-build\CMakeFiles\49f9d16effa529c310c9be3182f62407
\ITK-configure.rule;C:\Users\dchen\VeryVeryLongPathName\SimpleITK-build\CMakeFiles\
↪49f9d16effa529c310c9be3182f62407\ITK-build.rule;C:\Users\
dchen\VeryVeryLongPathName\SimpleITK-build\CMakeFiles\49f9d16effa529c310c9be3182f62407\
↪ITK-install.rule;C:\Users\dchen\VeryVeryLongPathName\
SimpleITK-build\CMakeFiles\0cd4fd1b047c29b91c2ff7b21fd713a6\ITK-complete.rule;C:\Users\
↪dchen\VeryVeryLongPathName\SimpleITK-build\CMakeFiles
\ff9535a930f370e2434114c6e547ebba\ITK.rule' exited with code 1. [C:\Users\dchen\
↪VeryVeryLongPathName\SimpleITK-build\ITK.vcxproj]
```

Building SimpleITK with long path names

If SimpleITK must be built with long path names, two things need to be done:

1. Enable Long Paths in Windows 10, Version 1607, and Later
2. Set the ITK_SKIP_PATH_LENGTH_CHECKS cmake variable to ON in the External_ITK.cmake

Enabling Long Paths in Windows

As described in Microsoft's documentation, Long path names are enabled in Windows by editing the registry with the following PowerShell command:

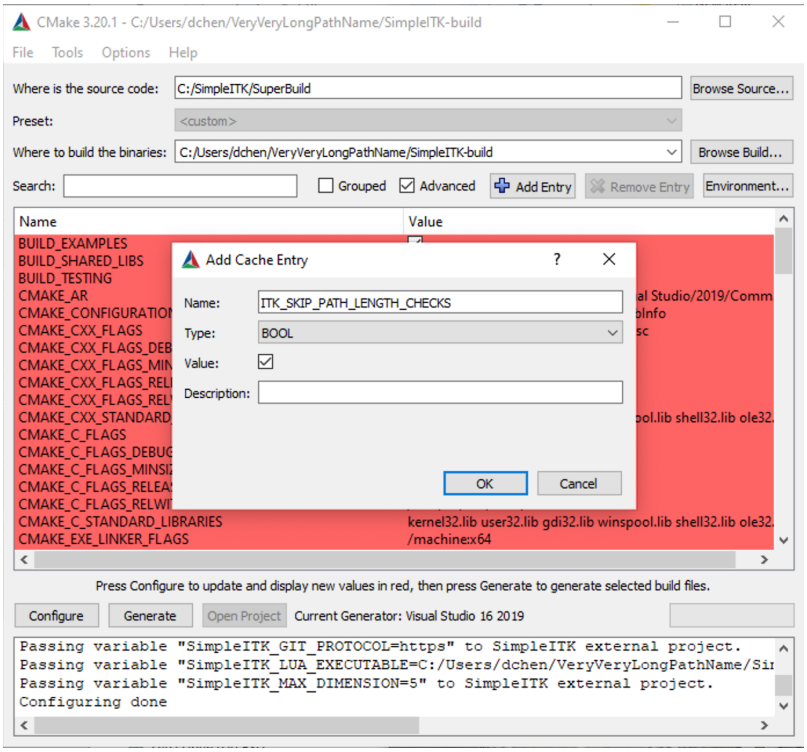
```
New-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Control\FileSystem" `-Name
↪"LongPathsEnabled" -Value 1 -PropertyType DWORD -Force
```

The LongPathsEnabled property can also be edited using the Registry Editor app.

Setting ITK_SKIP_PATH_LENGTH_CHECKS

Next ITK needs to be configured so that it does **NOT** check path lengths in the build process. To disable the path length check, we simply need to add the entry ITK_SKIP_PATH_LENGTH_CHECKS to the CMake configuration, and set its value to ON. We can do this in the CMake GUI when we configure our *SimpleITK SuperBuild*.

After the initial configuration of the build, click on the "Add Entry" button, enter ITK_SKIP_PATH_LENGTH_CHECKS for the name, and click the Value checkbox to turn set the entry to ON.

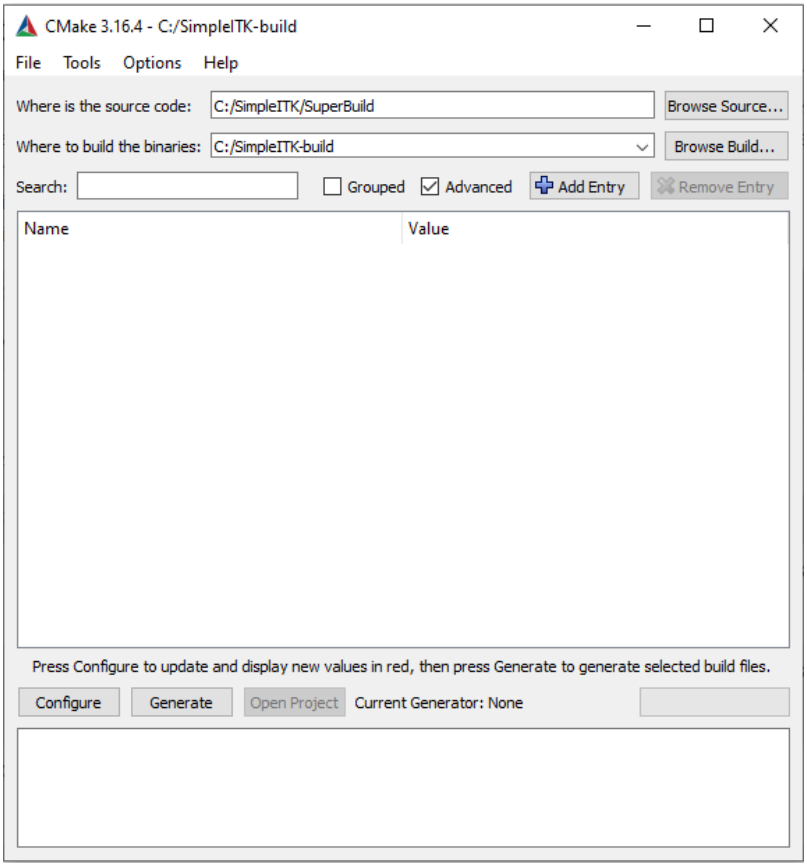


Now we can build ITK within the SimpleITK SuperBuild with long path names.

This page describes how to build a SimpleITK **SuperBuild** on Windows. To build SimpleITK we use CMake to generate a Visual Studio project, and Visual Studio to compile the C/C++ code. To obtain the SimpleITK source code see *Source code*.

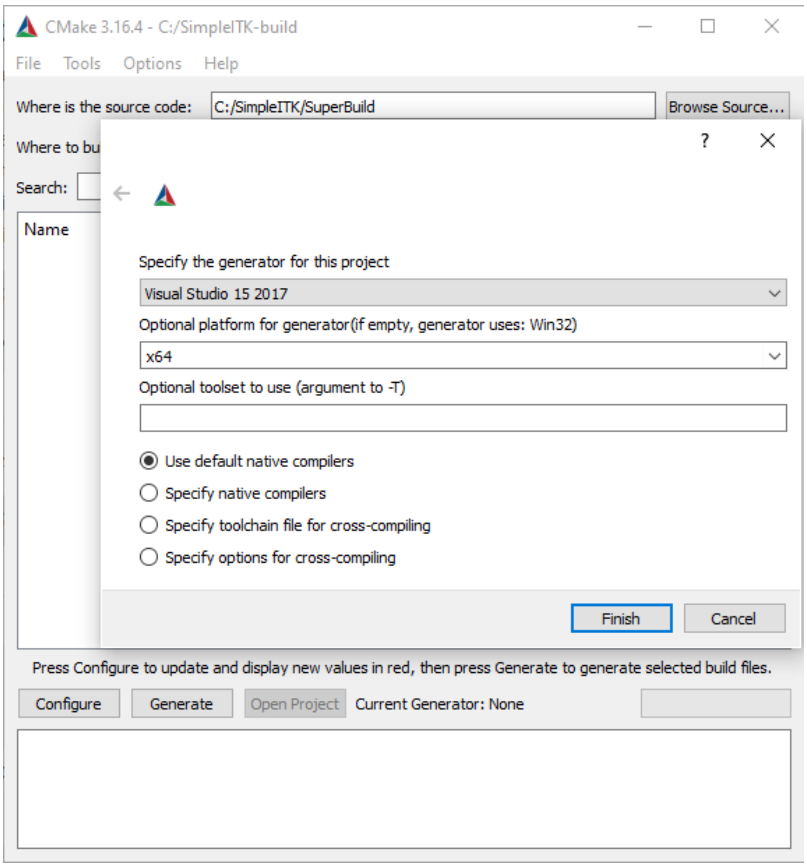
CMake GUI on Windows

The Windows version of CMake includes a graphical-user-interface (GUI) version of the program, in addition to the standard command-line CMake. Here is what the CMake GUI looks like:



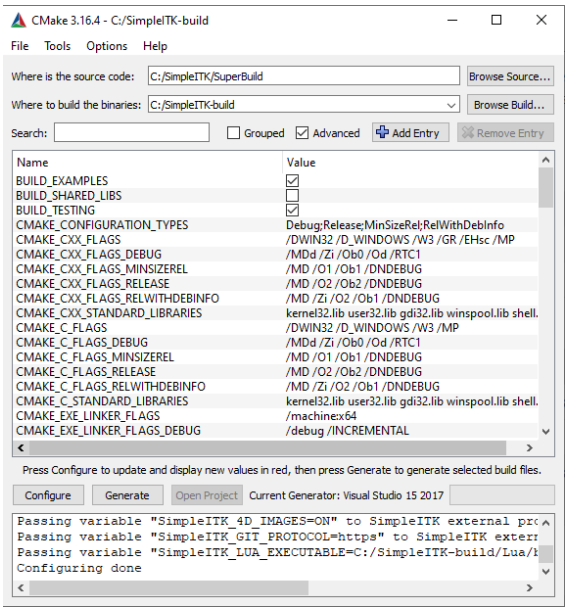
In the above example, the source directory is `C:/SimpleITK/SuperBuild`, and the build directory is `C:/SimpleITK-build`. It is best to place the source and build directories at the root of a volume to minimize path name lengths. Windows has a default max path length of 260 characters, and ITK has long paths which can cause build issues.

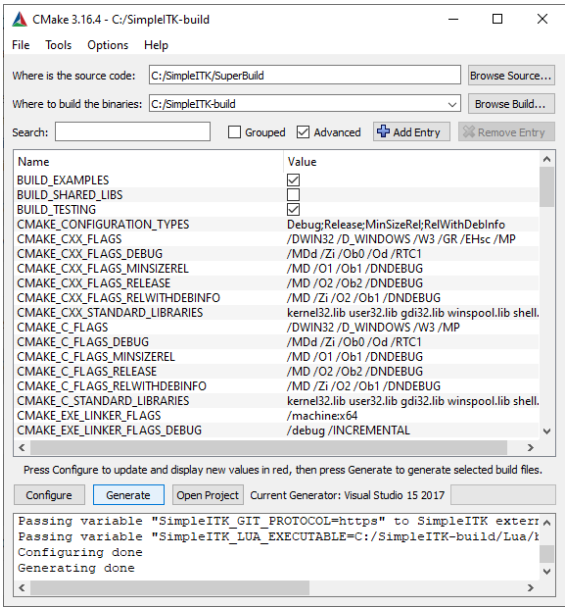
See Windows Path Length for more information.



After the user specifies the **source** and **build** directories, CMake will ask to select which compiler generator to use. The options are determined by what versions of Visual Studio are installed on the system. The user must specify **x64** for the platform version, since the default option (x86) is not supported by SimpleITK.

Configuring and Generating the Build





The next steps are the configuration and generation processes. By clicking the **Configure** button, cmake reads the `CMakeLists.txt` file and sets up all the variables in the CMake cache (`CMakeCache.txt`). Then when the **Generate** button is clicked, CMake creates the Visual Studio build system.

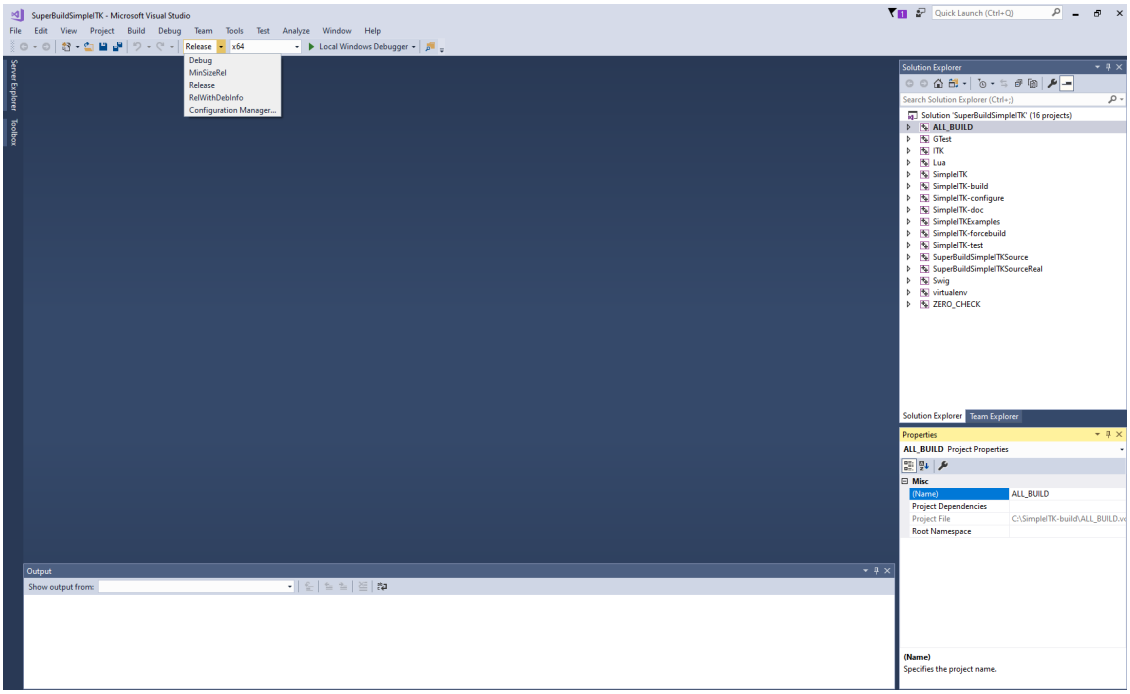
The left image of the above figure is the result of the configure process, and on the right is the result of the generation process.

Note for the **CMAKE_CXX_FLAGS** and **CMAKE_C_FLAGS** variables, we have appended the **'/MP'** flag, enabling multiple processor when building with Visual Studio. This flag is *not* enabled by default.

Building SimpleITK in Visual Studio

After CMake has completed the generation process, the user can actually build SimpleITK with Visual Studio. The user should select a build type in Visual Studio. We recommend choosing **Release** or **MinSizeRel** (release with minimum size), since the performance of SimpleITK with the Debug option is several orders of magnitude slower. Also the user must select the **x64** build platform, since the default platform (x86) is not supported by SimpleITK.

Many man hours and cpu cycles have been wasted building SimpleITK without setting the proper platform.



To begin the build process in Visual Studio, select **Build ALL_BUILD** in the **Build** menu.

Once the build has completed with no failures, the build can be verified with the following commands entered in a PowerShell:

```
cd C:\SimpleITK-build\SimpleITK-build
ctest -R Release
```

Running ctest executes all the built in test code for SimpleITK, more than 500 tests for the core C++ code, and approximately another 500 tests per wrapped language (e.g. Python, CSharp, R, Lua, etc.).

These commands assume that the SimpleITK build directory is `C:\SimpleITK-build` and that the build is a Release build.

8.6 Building Manually

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

8.6.1 Additional Prerequisites

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

1. Setup the prerequisites as described above (i.e. CMake and supported compiler)
2. Insight Toolkit (ITK) the version specified in the External_ITK.cmake file is the version of ITK used for the binary release. This can be seen as the minimum version of ITK to be used with SimpleITK, as future ITK versions are generally backwards compatible.
3. Lua 5.3.5
4. SWIG >= 4.0
5. GTest or Google Test >= 1.10 is needed if testing is enabled.

8.6.2 Configuration and Building

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

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

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

After proper configuration, SimpleITK can be built:

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

8.6.3 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.

8.6.4 Testing

After compilation the prudent thing to do is to test SimpleITK to ensure that the build is stable and suitable for installation and use. To enable testing the cmake *BUILD_TESTING* variable must be enabled during the build process. Then 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
```

8.6.5 Installation from Build Tree

Python Installation

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

```
python -m pip install SimpleITK-build/Wrapping/Python
```

Preferably, a Python virtual environment is created and the distribution installed there.

A Python Wheel file (.whl) is built by the "dist" target and produced in the "Wrapping/Python/dist" directory. If you use 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 installs the R package "SimpleITK" in /usr/local as root or your local R installation directory.

If you work 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 to install your R packages

```
mkdir my_R_libs
```

2. In your home directory create a file named .Renviron with the following line

```
R_LIBS_USER="/path_to/my_R_libs"
```

3. Check that the R library paths include this directory, in an R shell or from RStudio:

```
.libPaths()
```

4. install

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

8.7 Building SimpleITK Sphinx based documentation

SimpleITK uses Sphinx to create the documentation hosted on read-the-docs. To build this documentation we set up a Python environment with the appropriate packages.

8.7.1 Set up Sphinx Python virtual environment

The following are steps for setting up Sphinx:

1. Create a Python virtual environment ("sitkenv" is the name for the new virtual environment and its directory in this example).

```
python -m venv sitkenv
```

2. Activate the virtual environment

```
$ source sitkenv/bin/activate
(sitkenv) $
```

3. Install Sphinx and packages using the requirements.txt (found in SimpleITK/docs directory).

```
(sitkenv) $ pip install -r SimpleITK/docs/requirements.txt
```

8.7.2 Build documentation

Once the Sphinx environment is configured, build the SimpleITK documentation. From the SimpleITK/docs directory (on OSX/Linux):

```
(sitkenv) $ make html
```

Results are in the SimpleITK/docs/build/html directory.

CHAPTER

NINE

SETTING UP ECLIPSE AND VISUAL STUDIO

9.1 Java

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

9.1.1 Build/Install SimpleITK

For Windows, you have two options:

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

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

9.1.2 Set Up Eclipse

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

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

9.1.3 GaussianExample

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

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

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

public class GaussianExample {

        /**
         * @param args
```

(continues on next page)

(continued from previous page)

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

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

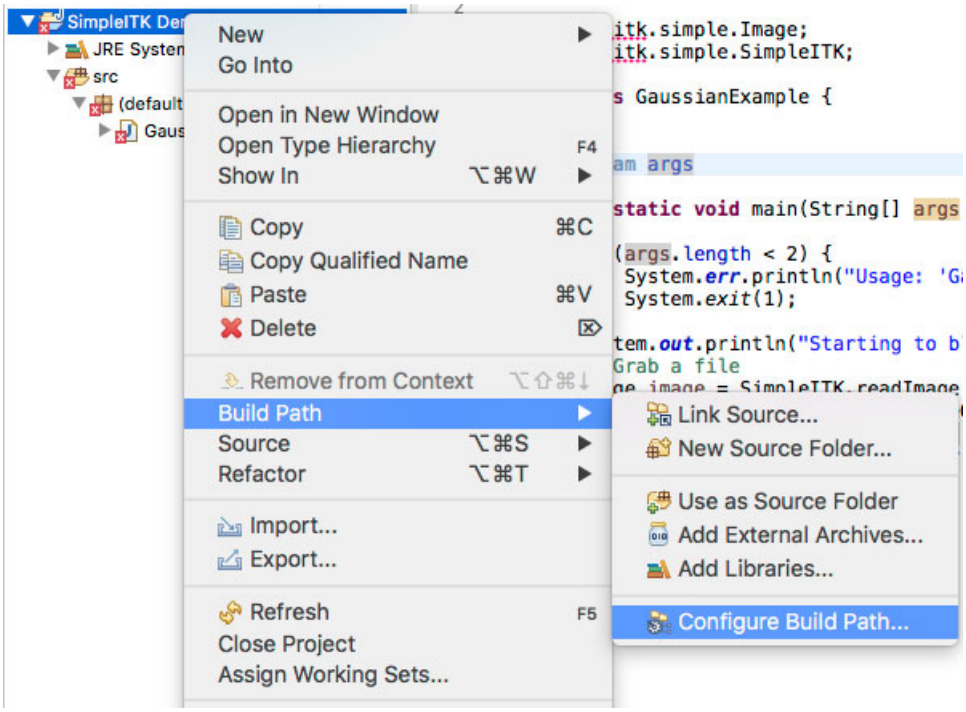
        }

}
```

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

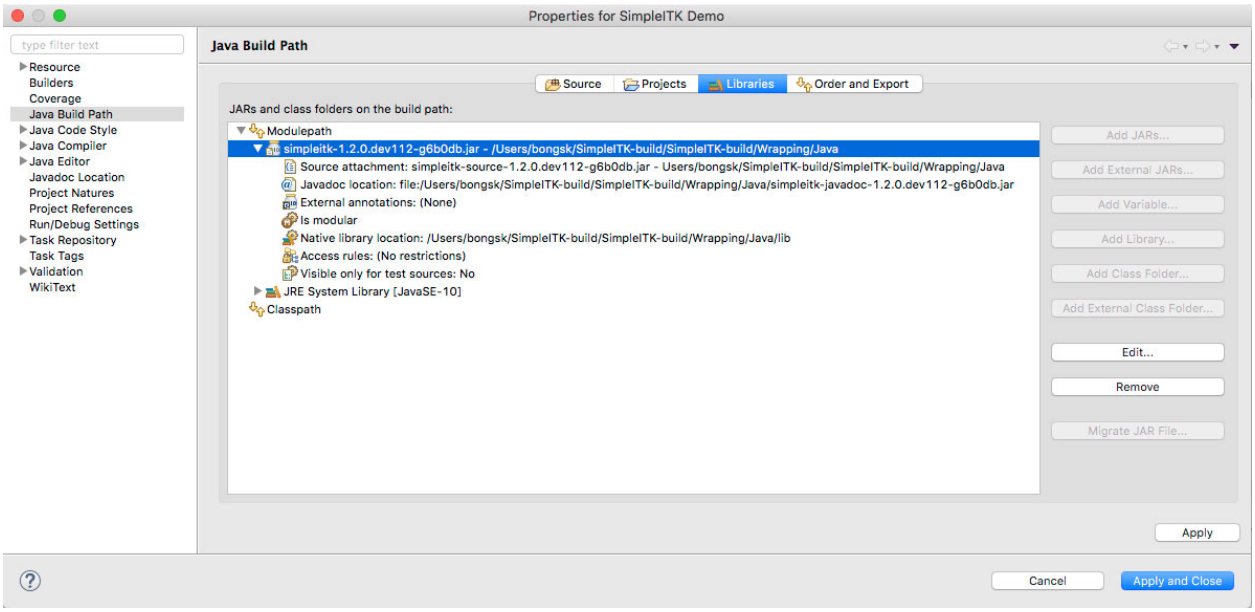
9.1.4 Add SimpleITK to the Build Path

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



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

The three options of interest are: *Source attachment*, *Javadoc location* and *Native library location*. The *Source attachment* specifies where the source code for the SimpleITK jar file resides. In our case, it is distributed as *simpleitk-*



source.x.x.x.jar where *x.x.x* is the version number of SimpleITK. The source attachment is useful for debugging the SimpleITK library, if necessary, because it allows the debugger to step through classes provided in the SimpleITK jar file. This setting is optional.

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

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

```
Exception in thread "main" java.lang.UnsatisfiedLinkError: no SimpleITKJava in java.
↪library.path
                at java.lang.ClassLoader.loadLibrary(ClassLoader.java:1758)
                at java.lang.Runtime.loadLibrary0(Runtime.java:823)
                at java.lang.System.loadLibrary(System.java:1045)
                at org.itk.simple.SimpleITKJNI.<clinit>(SimpleITKJNI.java:62)
                at org.itk.simple.SimpleITK.readImage(SimpleITK.java:33)
                at org.itk.simple.example.GaussianExample.main(GaussianExample.java:19)
```

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

9.1.5 SimpleITK Java Conventions

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

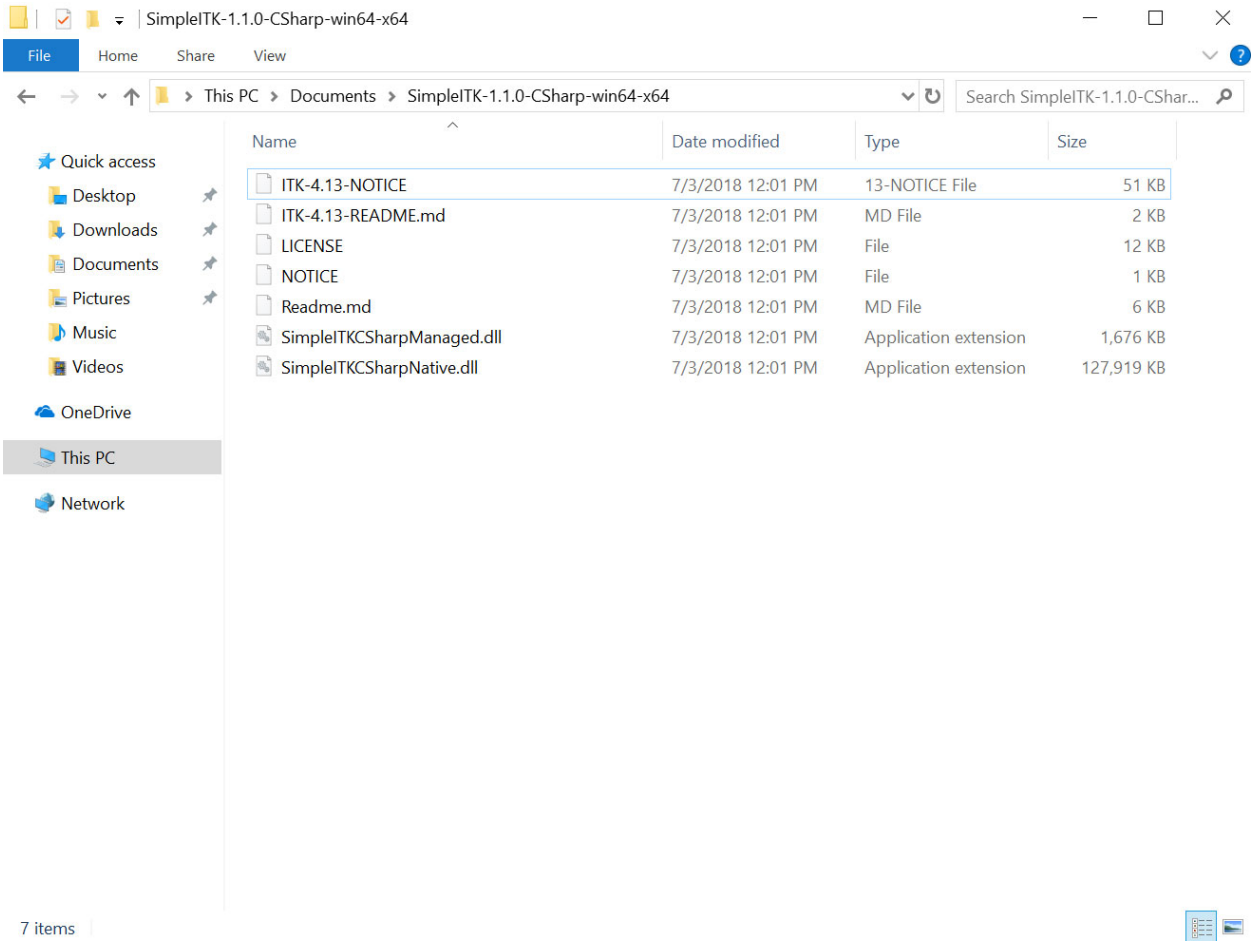
9.2 C#

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

9.2.1 Building SimpleITK

For Windows, you have two options:

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

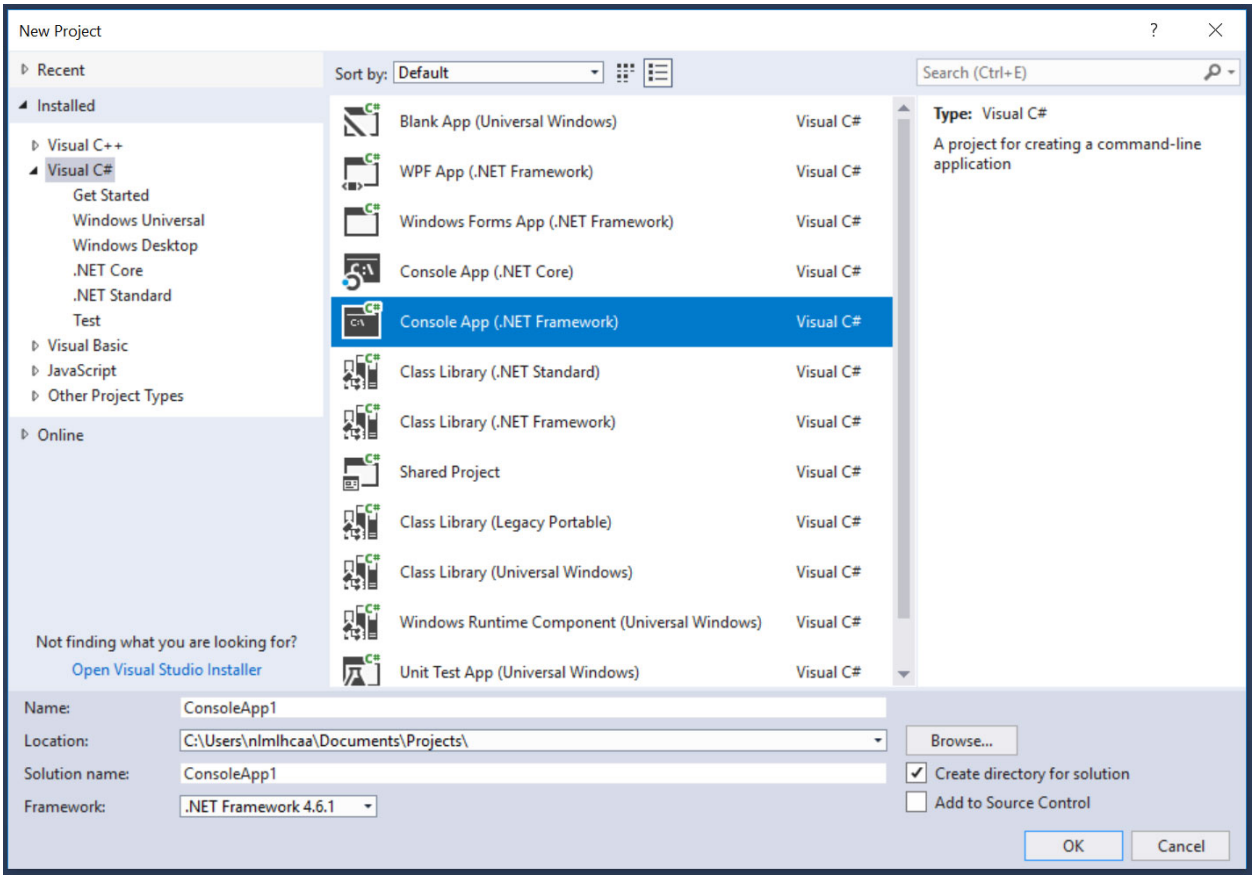


For any other OS, you must build the binaries yourself.

9.2.2 Set Up Visual Studio

Download Visual Studio if you don't have it.

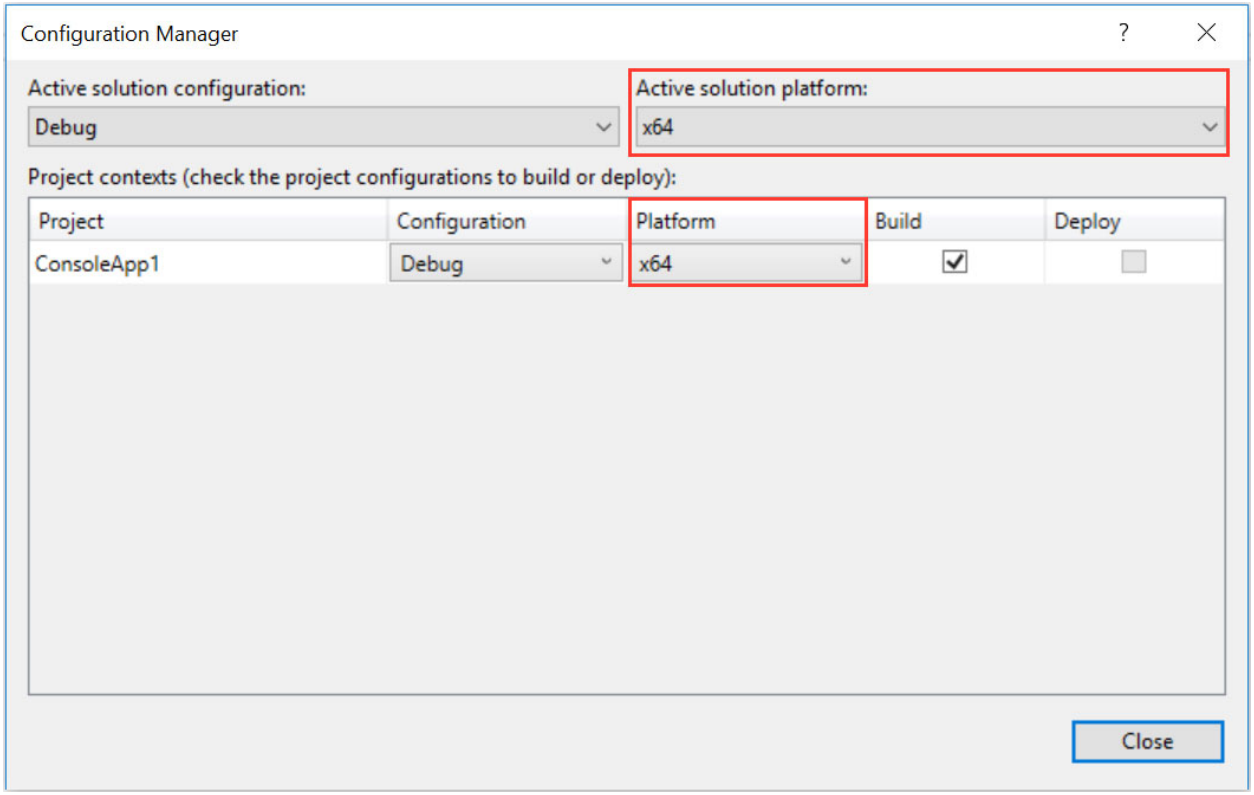
Then, create a new project in Visual Studio. Choose *File –> New –> Project...*, then select "Visual C#" and "Console App".



9.2.3 Select Architecture

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

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



9.2.4 Add Managed Library

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

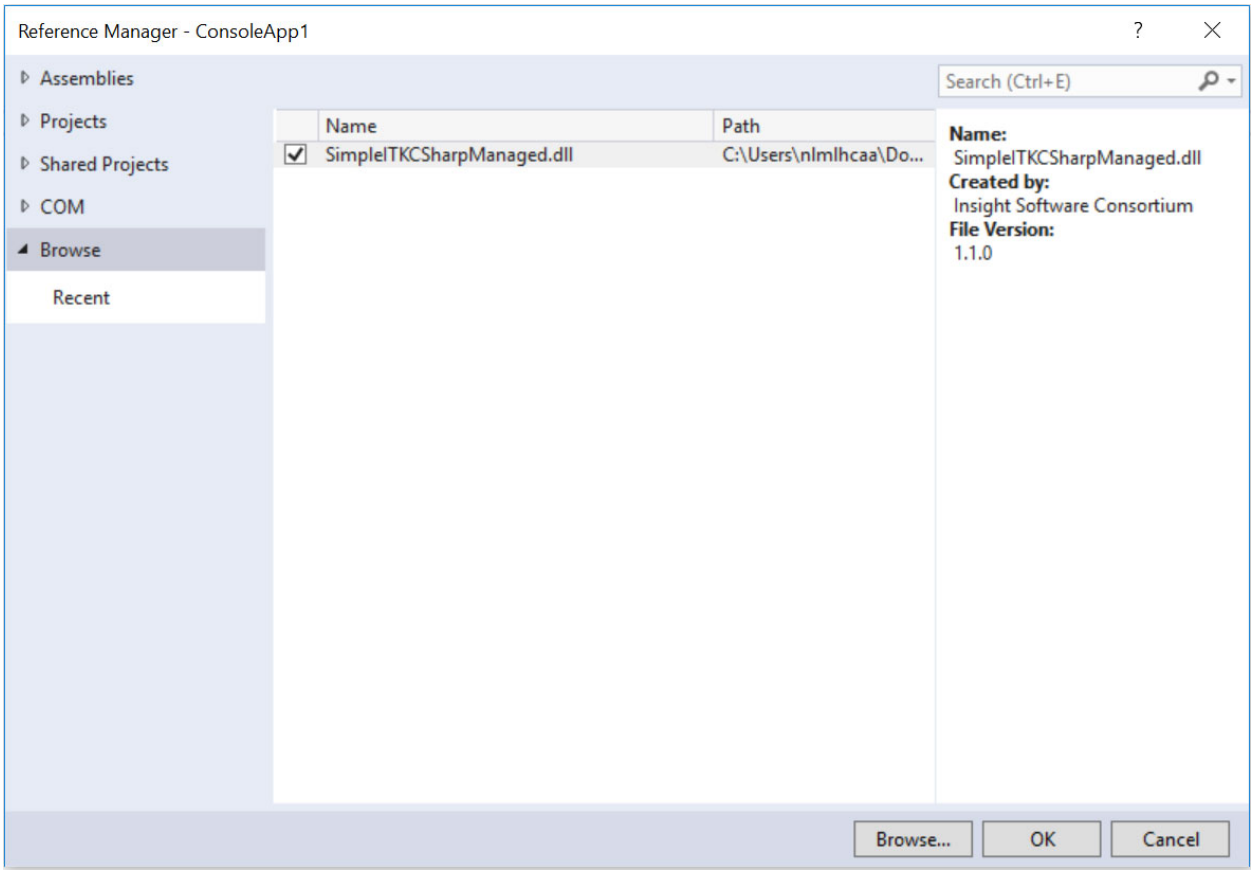
9.2.5 Add Native Library

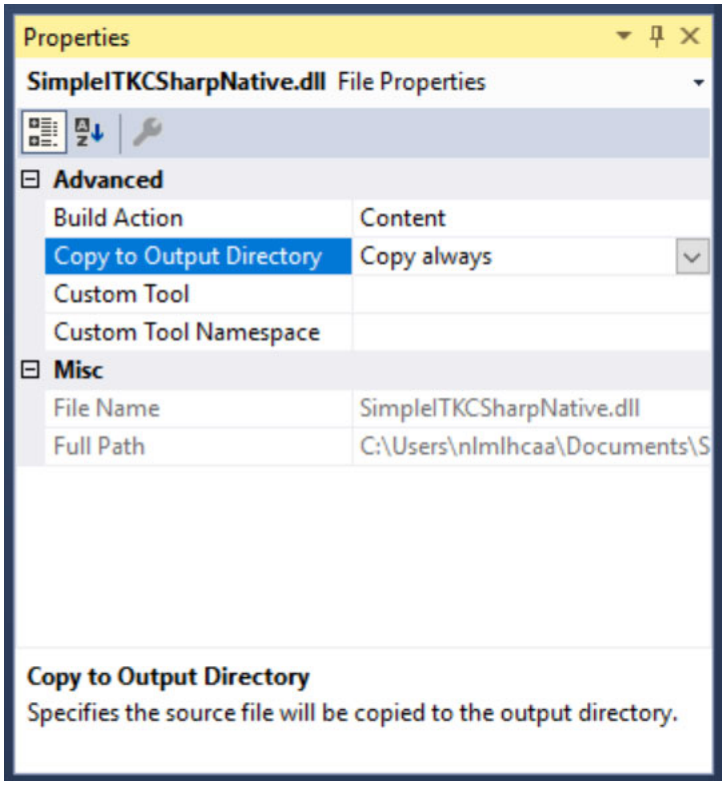
- From the menu bar select *PROJECT->Add Existing Item...* Select "Executable Files" for the extension type. Then navigate the file system to the unzipped "SimpleITKCSharpNative.dll" file from the binary download. **IMPORTANT** in the "Add" button's pull down menu select "Add As Link".
- In the Solution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

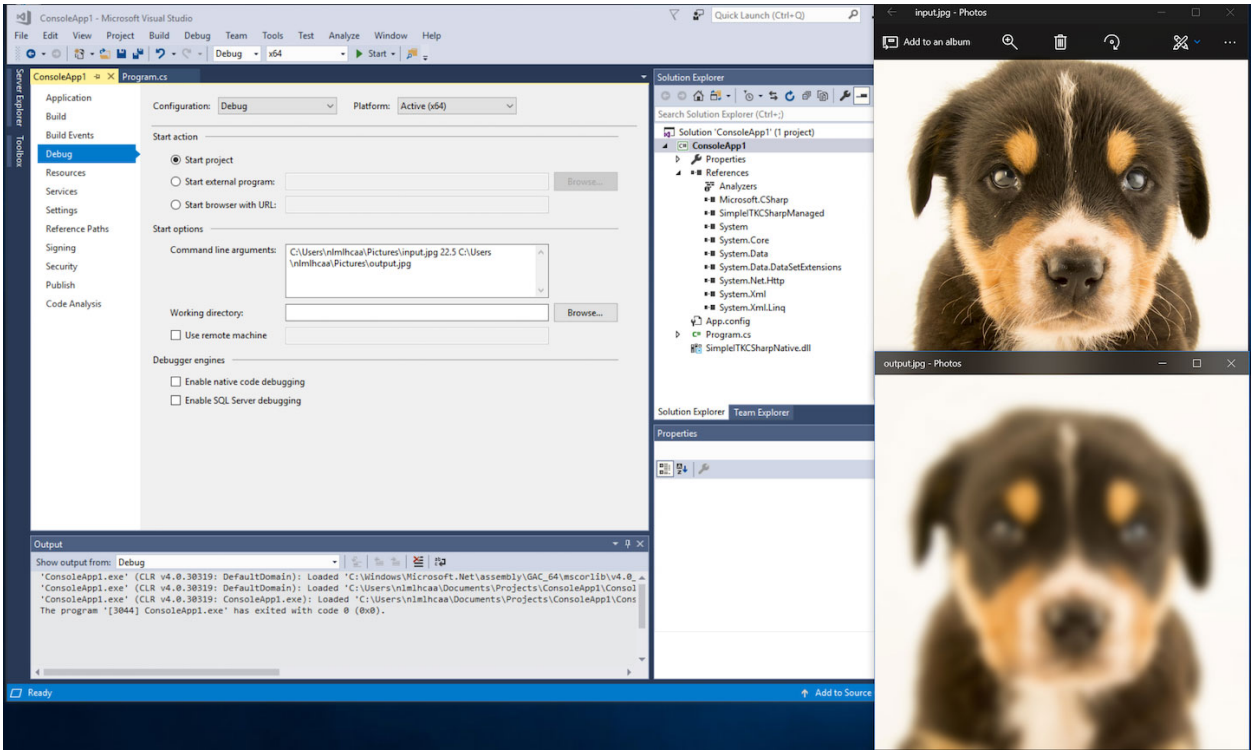
9.2.6 Building an Example

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

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


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


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


ConsoleApp1 - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug x64 Start
ConsoleApp1 Program.cs
Application
Build
Build Events
Debug
Resources
Services
Settings
Reference Paths
Signing
Security
Publish
Code Analysis
Configuration: Debug
Platform: Active (x64)
Start action
Start project
Start external program:
Start browser with URL:
Start options
Command line arguments: C:\Users\nlmlhcaa\Pictures\input.jpg 22.5 C:\Users\nlmlhcaa\Pictures\output.jpg
Working directory:
Use remote machine
Debugger engines
Enable native code debugging
Enable SQL Server debugging
Output
Show output from: Debug
'ConsoleApp1.exe' (CLR v4.0.30319: DefaultDomain): Loaded 'C:\Windows\Microsoft.Net\assembly\GAC_64\mscorlib\v4.0_
'ConsoleApp1.exe' (CLR v4.0.30319: DefaultDomain): Loaded 'C:\Users\nlmlhcaa\Documents\Projects\ConsoleApp1\Consol
'ConsoleApp1.exe' (CLR v4.0.30319: ConsoleApp1.exe): Loaded 'C:\Users\nlmlhcaa\Documents\Projects\ConsoleApp1\Cons
The program '[3044] ConsoleApp1.exe' has exited with code 0 (0x0).
Ready
Add to Source
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'ConsoleApp1' (1 project)
ConsoleApp1
Properties
References
Analyzers
Microsoft.CSharp
SimpleITKCSharpManaged
System
System.Core
System.Data
System.Data.DataSetExtensions
System.Net.Http
System.Xml
System.Xml.Linq
App.config
Program.cs
SimpleITKCSharpNative.dll
Solution Explorer Team Explorer
Properties
input.jpg - Photos
Add to an album
output.jpg - Photos

CHAPTER

TEN

TUTORIALS AND COURSES

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

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

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









10.1 Virtual

SimpleITK: A Simplified Path to Insight [git repository].

10.2 Upcoming (in person)

10.3 Past (in person)

Canceled by ISBI organizers due to COVID-19

- International Symposium on Biomedical Imaging (ISBI)2020 Tutorial, Iowa City IA, USA: SimpleITK: image analysis for all levels of programming expertise [git repository].

- IEEE Engineering in Medicine and Biology Conference (EMBC) 2019, Berlin Germany: SimpleITK: A Tool for Biomedical Image Processing, from Cells to Anatomical Structures [git repository].
- SPIE Medical Imaging 2019 Course, San Diego CA, USA: SimpleITK Jupyter Notebooks: Biomedical Image Analysis in Python [git repository].
- IEEE International Symposium on Biomedical Imaging (ISBI)2018 Tutorial, Washington DC, USA: Biomedical Image Analysis in Python and R using SimpleITK Jupyter Notebooks [git repository].
- SPIE Medical Imaging 2018 Course, Houston TX, USA: SimpleITK Jupyter Notebooks: Biomedical Image Analysis in Python [git repository].
- The International Symposium on Biomedical Imaging (ISBI) 2016, Prague, Czech republic: SimpleITK: An Interactive, Python-Based Introduction to SimpleITK with the Insight Segmentation and Registration Toolkit (ITK).
- SPIE Medical Imaging 2016, San Diego, USA: ITK in Biomedical Research and Commercial Applications [git repository , additional presentations].
- Medical Image Computing and Computer Assisted Intervention (MICCAI) 2015, Munich, Germany: a Python based tutorial on the use of the ITKv4 registration framework via SimpleITK [git repository].
- ImageJ User & Developer Conference 2015, Madison, WI, USA: an introductory tutorial in Python [git repository].
- Medical Image Computing and Computer Assisted Intervention (MICCAI) 2011, Toronto, Canada: a general tutorial on SimpleITK [pdf of presentation, git repository].

CHAPTER

ELEVEN

FREQUENTLY ASKED QUESTIONS

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

11.1 General

11.1.1 Does SimpleITK offer all the functionality as the C++ ITK?

SimpleITK supports most ITK image filters (*see list*) and the IO and registration frameworks. It exposes fewer settings than ITK, hence the Simple in the name. The main ITK elements omitted from SimpleITK are the pipeline architecture, spatial objects framework, point sets, and the mesh framework.

11.1.2 What is the toolkit's Python version support policy?

For the Python programming language, SimpleITK stops providing binary packages when the Python version has reached end-of-life status. Extensions may be made such as when the EOL Python version is included as part of a long-term support operating system version. If you need a newer SimpleITK binary than the one available for your Python version you can *build it on your own*.

11.2 Installation

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

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

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

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

11.2.2 Why am I getting "ModuleNotFoundError: No module named '_SimpleITK'" when importing SimpleITK on Windows?

Traceback (most recent call last):

File "C:Python37libsite-packagesSimpleITKSimpleITK.py", line 14, in swig_import_helper
return importlib.import_module(mname) ... File "<frozen importlib._bootstrap>", line 219, in _call_with_frames_removed ImportError: DLL load failed: The specified module could not be found.

The above error occurs on Microsoft windows when system libraries SimpleITK depends on are not available on the system. This issue can be resolved by installing the appropriate Redistributable for Visual Studio package.

11.2.3 Why am I getting "DllNotFoundException: Unable to load DLL 'SimpleITKCSharpNative'" when running a CSharp application on Windows?

This error message can be the result of an incomplete installation of SimpleITK in the CSharp application. Below is the complete error message that can result:

```
System.TypeInitializationException: 'The type initializer for 'itk.simple.
↪SimpleITKPINVOKE' threw an exception.'

TypeInitializationException: The type initializer for 'SWIGExceptionHelper' threw an␣
↪exception.

DllNotFoundException: Unable to load DLL 'SimpleITKCSharpNative': The specified module␣
↪could not be found. (Exception from HRESULT: 0x8007007E)
```

The application is missing the SimpleITK dynamically loaded library (DLL). To fix this problem, follow the instructions in the Add Native Library section of the SimpleITK Visual Studio/C# build instructions.

11.3 How to Use

11.3.1 What filters are currently available in SimpleITK?

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

11.3.2 What image file formats can SimpleITK read?

See *here*.

11.3.3 How do I read a RAW image into SimpleITK?

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

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

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

An example implementing this approach is available here *here*.

11.3.4 Why does my image appear to be empty / all black / blank when visualized?

There are two possible reasons for this:

1. The image is indeed empty. This is rarely the case, and indicates that there is something incorrect with the code, will require debugging.
2. The image contains very low values (1, 2, 3...). This is very common with segmentation and *binary mask* images. What you are experiencing is an issue with data visualization, not with the data itself. The code below illustrates the difference, and shows how to quickly visualize such images using the existing SimpleITK functionality.

```
import SimpleITK as sitk

# Create segmentation image
segmentation_image = sitk.Image([128,128], sitk.sitkUInt8)
segmentation_image[40:50,20:120] = 1
segmentation_image[50:60,20:120] = 2
segmentation_image[60:70,20:120] = 3

# When visualized with Fiji, image looks all black. To see the data
# change the display settings Image->Adjust->Brightness/Contrast.
sitk.Show(segmentation_image)

# For quick visualization, convert to float and add
# 255 so that the data is immediately visible in Fiji.
sitk.Show(sitk.Cast(segmentation_image,sitk.sitkFloat32) + 255)
```

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

The SimpleITK Show function expects the Fiji or ImageJ application to be installed in specific locations. The recommended installation locations are:

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

To see the locations where the function is searching set Show's debugOn flag.

In Python:

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

In R:

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

Show is a functional interface to the ImageViewer class. Other viewing applications can be configured using an ImageViewer object, as described in the next section.

11.3.6 Can I use another image file viewer beside Fiji?

The ImageViewer class allows a user to configure what application SimpleITK uses to display images. An ImageViewer object displays an image via the Execute method.

The default display application for all image types is Fiji. To override Fiji with some other application, use the `ImageViewer::SetCommand` method. For example in Python on Linux systems, using ImageMagick's display program would look like this:

```
import SimpleITK as sitk

viewer = sitk.ImageViewer()
viewer.SetFileExtension('.png')
viewer.SetCommand('/usr/bin/display')
```

By default when `ImageViewer::Execute` is called, it writes out a temporary image in Nifti format then launches Fiji. If the viewing application has been changed to one that does not support Nifti, the file format of the temporary file can be overridden using the `ImageViewer::SetFileExtension` method. In the above example, we use PNG, a format ImageMagick does support, unlike Nifti.

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

More details into ImageViewer configuration can be found in the ImageViewer class documentation.

11.3.7 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. An ImageViewer object can be configured to use Slicer instead of SimpleITK's default viewer, Fiji. The following are examples of how to configure an ImageViewer object in Python for Mac OS X, Linux and Windows to use Slicer.

Mac OS X

```
import SimpleITK as sitk

viewer = sitk.ImageViewer()
viewer.SetCommand('/Applications/Slicer.app/Contents/MacOS/Slicer')
```

Linux

```
import SimpleITK as sitk

viewer = sitk.ImageViewer()
viewer.SetCommand('Slicer')
```

Windows

```
import SimpleITK as sitk

viewer = sitk.ImageViewer()
viewer.SetCommand( 'c:\Program Files\Slicer 4.10.2\Slicer' )
```

The call to SetCommand should be modified to point to wherever the Slicer executable is installed.

11.4 Wrapping

11.4.1 Python

Why should I use a virtual environment?

Before installing SimpleITK we highly recommend creating a virtual environment into which the package can be installed. Note that different Python versions and distributions have different programs for creating and managing virtual environments.

The use of a virtual environment allows a user to elegantly deal with package compatibility 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 venv 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, we recommend compiling SimpleITK explicitly against this Python distribution to ensure compatibility.

11.4.2 Tcl

11.4.3 Java

11.4.4 C#

11.4.5 R

11.5 Compilation

11.5.1 Is my compiler supported?

SimpleITK uses advanced C++ meta-programming to instantiate ITK's Images and Filters. SimpleITK is developed to require the C++11 standard.

In practice the list of compilers actively supported are those that are used for continuous testing and integration. These can be seen on the SimpleITK dashboard. We welcome user contributions to the nightly dashboard to expand the list of these compilers and contributions to fix additional compilation problems.

Noted Problems

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

11.5.2 Are 32-bits architectures supported?

While 32-bit binaries are no longer pre-compiled, the intel 32-architecture are still tested to help ensure robustness of the toolkit. Contributions and bug reports to support additional architectures are welcomed.

11.5.3 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
```

11.5.4 What Configurations on Windows are Supported For Building?

We recommend using at least Microsoft Visual Studio 15 (2017) with MSVC v140 toolset.

11.5.5 Path Length Issues on Windows

The location of the build and source directories on Windows can cause the build to fail. By default, Windows does not allow path lengths longer than 260 characters.

See Windows Path Length for more information.

11.5.6 Where is the Test Data?

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

If the source code was obtained from the git repository, the test data should be downloaded as part of the build process via the CMake ExternalData module.

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

11.5.7 Why is CMake unable to download ExternalData?

When compiling SimpleITK an error like the following may occur:

```
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.org/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. To re-build CMake with OpenSSL support, simply reconfigure CMake with the "CMAKE_USE_OPENSSL" option enabled.

CHAPTER

TWELVE

MIGRATION GUIDE 2.0

With the release of SimpleITK 2.0 there are changes to the interface and to the behaviour of some methods and classes. The intent of this document is to provide guidance on how to modify code so that it is compatible with SimpleITK 2.0.

This document will continue to evolve after the SimpleITK 2.0 release, consider reading the latest version. Contributions to update this document that share problems and solutions when updating to SimpleITK 2.0 are welcomed.

12.1 Interface Compatibility

These changes list API changes that cause compilation or runtime errors.

12.1.1 Composite Transforms

The *Transform::AddTransform* method was removed.

Composite transformations were previously represented using the generic Transform class. This class served both as a base class for all transformations and as a composite transformation. This dual purpose has been separated. Composite transformations are now represented by a dedicated class CompositeTransform and the associated functionality was removed from the *Transform* class.

1.x	2.x

`tx = sitk.Transform()`
`tx1 = sitk.TranslationTransform(3,(1,7,`
`↪76))`
`tx2 = sitk.Euler3DTransform()`
`tx.AddTransform(tx1)`
`tx.AddTransform(tx2)`
	`tx1 = sitk.TranslationTransform(3,(1,7,`
	`↪76))`
	`tx2 = sitk.Euler3DTransform()`
	`tx = sitk.CompositeTransform([tx1, tx2])`

This change does effect compatibility when reading or writing *CompositeTransforms* between versions.

12.1.2 Filter's Execute Method

The Filters' *Execute* methods with filter parameters as arguments was removed. A filter object's parameters are only set via the SetXYZ methods. For example the STAPLEImageFilter class:

1.x	2.x

`staple_filter = sitk.STAPLEImageFilter()`
 	
`staple_image = staple_filter.`
`↪Execute(image1,`
`↪image2, image3,`
`↪confidenceWeight,`
`↪foregroundValue,`
`↪maximumIterations)`
`staple_specificity = staple_filter.`
`↪GetSpecificity()`
	`staple_filter = sitk.STAPLEImageFilter()`
	`staple_filter.`
	`↪ConfidenceWeight(confidenceWeight)`
	`staple_filter.`
	`↪SetForegroundValue(foregroundValue)`
	`staple_filter.`
	`↪SetMaximumIterations(maximumIterations)`
	`staple_image = staple_filter.`
	`↪Execute([image1, image2, image3])`
	
	`staple_specificity = staple_filter.`
	`↪GetSpecificity()`

Alternatively, the procedural method could be used if no measurements or side effects on the object are needed.

12.1.3 Morphology Radius

The procedural interfaces for morphology filters only accept vector values for the radius parameter. Previously the procedures accepted a scalar for the radius parameter. For example, a structuring element of radius 5 used on a 3D image is specified as [5, 5, 5]:

1.x	2.x
`seg = sitk.BinaryDilate(seg, 5)`	`seg = sitk.BinaryDilate(seg, [5]*seg.` `↪GetDimension())`

The following is a list of procedures affected by this change:

- *BinaryClosingByReconstruction*
- *BinaryDilate*
- *BinaryErode*
- *BinaryMorphologicalClosing*
- *BinaryMorphologicalOpening*
- *BinaryOpeningByReconstruction*
- *BlackTopHat*
- *ClosingByReconstruction*
- *DilateObjectMorphology*

- *ErodeObjectMorphology*
- *GrayscaleDilate*
- *GrayscaleErode*
- *GrayscaleMorphologicalClosing*
- *GrayscaleMorphologicalOpening*
- *MorphologicalGradient*
- *OpeningByReconstruction*
- *WhiteTopHat*

12.1.4 FastMarchingImageFilter

The *FastMarchingImageFilter::AddTrialPoints* and *FastMarchingImageFilter::ClearTrialPoints* methods were removed. The *SetTrialPoints* method can be used instead to set the whole list of trial points.

12.2 Behavior Changes

12.2.1 ITKv5

SimpleITK 2.0 uses ITK version 5.0. This major version change to ITK includes numerous changes to the usage and behavior of ITK. The ITK v5 Migration Guide contains valuable information documenting changes.

12.2.2 Show Environment Variables

The environment variables related to the *Show* method are only checked once. The *Show* method is a wrapper for the ImageViewer class, changing the behavior from SimpleITK 1.0. The SITK_SHOW_EXTENSION and SITK_SHOW_COMMAND environment variables if defined initialize the corresponding global defaults only the first time accessed. The environment variables SITK_COLOR_COMMAND and SITK_SHOW_3D_COMMAND were removed as the ImageViewer class allows for direct configuration of the command.

12.2.3 Images

Images support sub-image left assignment of constants and matching image regions.

```
img[0:10,5:15] = 128
img[5:10,10:15] = img2[0:5,5:10]
```

12.2.4 DICOM MOCHOROME1

Changes to DICOM reading, due to SimpleITK's move from ITK 4.x to 5.x. Photometric interpretation, tag 0028|0004, is now taken into account when reading the image.

1. MOCHOROME1: previously intensity values loaded as is, now they are inverted. Code that previously inverted the intensities explicitly needs to be removed.
2. Color images (e.g. YBR_FULL_422): Previously channels loaded as is, now they are converted to RGB. If previous code used the photometric interpretation tag value to perform color space conversions outside of SimpleITK it will need to be modified. For example, previously the first channel for an image with photometric interpretation of YBR_FULL was luminance and could be used as the grayscale version of the image. Now all three channels are used to compute the image luminance:

1.x	2.x

`image = sitk.ReadImage(file_name)`
`grayscale_image = sitk.`
`↪VectorIndexSelectionCast(image,0)`
	`image = sitk.ReadImage(file_name)`
	`grayscale_image = 0.299*sitk.`
	`↪VectorIndexSelectionCast(image,0,sitk.`
	`↪sitkFloat32) +`
	`0.587*sitk.`
	`↪VectorIndexSelectionCast(image,1,sitk.`
	`↪sitkFloat32) +`
	`0.114*sitk.`
	`↪VectorIndexSelectionCast(image,2,sitk.`
	`↪sitkFloat32)`
	`grayscale_image = sitk.Cast(sitk.`
	`↪RescaleIntensity(grayscale_image))`

12.2.5 Python Installation

The Python installation procedure to use *setup.py* changed. To install see the updated Python Installation documentation.

12.2.6 Multi-threading

The upgrade to ITKv5 contains a major refactoring of ITK's threading model. Details on the behavior of the new default PoolMultiThreader can be found in the ITK v5 Multi-Threading Section.

The SimpleITK ProcessObject class contains the methods to configure the ITK Threader used and the number of threads and number of work units used for an algorithm.

12.2.7 Registration Smoothing

Results from the registration framework may change due to the the Gaussian smoothing algorithm changing from the *DiscreteGaussianImageFilter* to the *SmoothingRecursiveGaussianImageFilter*.

CHAPTER

THIRTEEN

DEVELOPER

13.1 Image Filter Templates ReadMe

This document outlines the system used to generate SimpleITK files using json data files in conjunction with the template .h and .cxx files.

13.2 Overview

In order to create a uniform body of image filters, we use an automated code generation system to wrap existing ITK filters into SimpleITK filters. The development process consists of writing a JSON data file which specifies the information needed to fill in the fields of the template file. During compilation, all of the .json files are parsed using a lua template engine which fills in the appropriate template file from the provided fields. This ReadMe provides a detailed description of the fields that can be used in the json data files and what effect each field has on the resulting image filter. When writing your own json file, it should be saved in: `SimpleITK_SOURCE_DIR/Code/$FILTER_DIR/json`

Here `$FILTER_DIR` is the directory in which the code for the generated filter should live. For simple filters this is the "BasicFilters" directory.

13.3 JSON File Structure

JSON (JavaScript Object Notation) is a simple text formatting system used to specify data values. A full description of the specification can be found at http://www.json.org/. For the purposes of SimpleITK, a json file consists of a single data object (denoted by {}) which contains several string : value pairs (here value may be a list denoted by []) and sub-objects. Here is an example from LaplacianRecursiveGaussian:

```
{
  "name" : "LaplacianRecursiveGaussianImageFilter",
  "template_code_filename" : "ImageFilter",
  "template_test_filename" : "ImageFilter",
  "doc" : "",
  "number_of_inputs" : 1,
  "pixel_types" : "BasicPixelIDTypeList",
  "output_pixel_type" : "float",
  "members" : [
    {
      "name" : "Sigma",
      "type" : "double",
      "default" : "1.0",
```

(continues on next page)

(continued from previous page)

```
      "doc" : "",
      "briefdescriptionSet" : "",
      "detaileddescriptionSet" : "Set Sigma value. Sigma is measured in the units of␣
↪image spacing.",
      "briefdescriptionGet" : "",
      "detaileddescriptionGet" : "Set Sigma value. Sigma is measured in the units of␣
↪image spacing."
    },
    {
      "name" : "NormalizeAcrossScale",
      "type" : "bool",
      "default" : "false",
      "doc" : "",
      "briefdescriptionSet" : "",
      "detaileddescriptionSet" : "Define which normalization factor will be used for the␣
↪Gaussian \\see RecursiveGaussianImageFilter::SetNormalizeAcrossScale",
      "briefdescriptionGet" : "",
      "detaileddescriptionGet" : "Define which normalization factor will be used for the␣
↪Gaussian \\see RecursiveGaussianImageFilter::SetNormalizeAcrossScale"
    }
  ],
  "tests" : [
    {
      "tag" : "default",
      "description" : "Simply run with default settings",
      "settings" : [],
      "tolerance" : 0.0001,
      "inputs" : [
        "Input/RA-Float.nrrd"
      ]
    }
  ],
  "briefdescription" : "Computes the Laplacian of Gaussian (LoG) of an image.",
  "detaileddescription" : "Computes the Laplacian of Gaussian (LoG) of an image by␣
↪convolution with the second derivative of a Gaussian. This filter is implemented using␣
↪the recursive gaussian filters.",
  "itk_module" : "ITKImageFeature",
  "itk_group" : "ImageFeature"
}
```

In this example, we see that all pairs must use a string as the key, but may have any valid type as the value (string - “Abs”, number - 1, list - [], object - {}).

13.3.1 Fields For All Templates

This section describes the minimal set of fields required to wrap an ITK filter into SimpleITK.

13.3.2 Top Level Fields

- **name**: (*string*) This must be the name of the ITK filter.
- **template_code_filename**: (*string*) Which of the template files should be used to generate the filter. Choices are: "ImageFilter", "RegionGrowingImageFilter", "StructuringElementImageFilter", "LevelSetImageFilter", "MathematicalMorphologyImageFilter"
- **template_test_filename**: (*string*) Which of the template files should be used to generate the test for this filter. Choices are: "ImageFilter", "RegionGrowingImageFilter"
- **number_of_inputs**: (*integer*) How many input images are used for this filter. Currently, the template system only supports filters with 1 or 2 input images
- **briefdescription**: (*string*) Brief description of the class (Doxygen format). This string is automatically generated by the Utilities/BuildDocs and Utilities/GenerateDocumentation.groovy scripts. If no corresponding ITK class exists, this field will not be modified. This field is maintained with the JSONDocUpdate.sh script.
- **detaileddescription**: (*string*) More detailed description of the class (Doxygen format). This string is automatically generated by the Utilities/BuildDocs and Utilities/GenerateDocumentation.groovy scripts. If no corresponding ITK class exists, this field will not be modified. This field is maintained with the JSONDocUpdate.sh script.
- **pixel_types**: (*string*) Which of the pixel type lists should be supported for this filter. The type lists are defined in sitkPixelIDTypeLists.h
- **members**: (*list*) This list specifies all member variables that should be accessible to the user with a Get/Set mechanism and should be part of the Execute signature for the filter. Each entry in the list must be an object
- **tests**: (*list*) This list specifies all automatic tests that should be generated for the filter. Each entry in the list must be an object. If no tests are specified, a test will be generated that fails with the message that a test must be written for the filter
- **itk_module**: (*string*) A string naming the ITK module the filter originates from. If the ITK installation used for building SimpleITK does not have this named module then the filter will be omitted in SimpleITK. This filed is maintained JSONUpdateITKModules.py script.
- **itk_group**: (*string*) A string naming the ITK group the ITK filter originates.
- [OPTIONAL] **include_files**: (*list*) This list of strings specifies additional header files to include in the cxx file for this filter.
- [OPTIONAL] **custom_set_input**: (*string*) Code which is used to set input or multiple inputs to the filter. This overrides the standard setting of the inputs.
- [OPTIONAL] **output_pixel_type**: (*string*) String representing the specific pixel type to use for the output image of the filter.
- [OPTIONAL] **filter_type**: (*string*) Explicitly sets the FilterType used. This method overrides additional_template_types. Useful typedefs are InputImageType, and OutputImageType.
- [OPTIONAL] **public_declarations**: (*string*) Any c++ code written in this field will be placed into the public section of the filter's header file
- [OPTIONAL] **additional_template_types**: (*list*) The standard ITK image filter template signature is:

```
<InputImageType, (if 2 inputs)InputImageType, OutputImageType)>
```

Some filters require additional template types between the list of input image types and the output image type such as the input type of a constant. This list allows these additional parameters to be added so that the template signature looks like:

```
<InputImageType, (if 2 inputs)InputImageType, (ADDITIONAL TYPES), OutputImageType)>
```

Each type in the list is specified by an object with a only a **type** field which is a string specifying the type to be used.

- [OPTIONAL] **custom_methods**: (*list*) This is a list of objects specifying custom methods that should be added to the filter. The fields required for a custom method are:
  - **doc**: (*string*) Documentation for this custom method
  - **name**: (*string*) The name of the method
  - **return_type:** (*string*) the return type of the method
  - [OPTIONAL] **parameters**: (*list*) A list of parameters for the method. Each method is specified by an object with the following fields:
    - **type**: (*string*) The parameter's type
    - **var_name**: (*string*) The name of the variable to be used in the **body field for the custom method
  - **body**: (*string*) This string is the body of the method and will be placed directly into the header file. The parameter names specified with **var_name** in each of the parameter objects can be used in this body

A sample custom method might look like:

```
{ "name" : "Foo",
  "doc" : "This method checks if a string is equal to Foo"
  "return_type" : "bool",
  "parameters" : [
    { "type" : "std::string",
      "var_name" : "inString"
    }
  ],
  "body" : "if (!strcmp(inString.c_str(), "Foo")) { return true; } else { return false; }
↪"
}
```

This will translate to a custom method that looks like:

```
/**
 * This method checks if a string is equal to Foo
 */
bool Foo( std::string inString )
  {
  if (!strcmp(inString.c_str(), "Foo")) { return true; } else { return false; }
  }
```

13.3.3 MemberFields Member Fields

A basic member object looks like:

```
{
  "name" : "Sigma",
  "type" : "double",
  "default" : 1.0,
  "briefdescriptionSet" : "",
  "detaileddescriptionSet" : "Define which normalization factor will be used for the␣
↪Gaussian\\see RecursiveGaussianImageFilter::SetNormalizeAcrossScale\n\n",
  "briefdescriptionGet" : "",
  "detaileddescriptionGet" : ""
 }
```

- **name**: (*string*) The name of the member variable. A variable named "Var" will be accessible with the methods "GetVar" and "SetVar"
- **type**: (*string*) The c++ type for this member. If using a type contained in a certain namespace (itk::simple for example), this must be included in the type description. If the member variable is of type bool, the additional ${name}On, and ${name}Off member functions are added.
- **default**: (*string* or *number*) The default value for this member. This can be a number for numerical types or a string specifying the constructor for non-basic types (std::vector< unsigned int >() for example)
- **briefdescriptionSet/briefdescriptionGet**: (*string*) Brief Doxygen documentation for this individual parameter. The two forms (Set/Get) are used for the SetMember and GetMember functions. This field is automatically generated from ITK's Doxygen for members that mirror ITK's members. If the member if a custom, this field will not be modified.
- **detaileddescriptionSet/detaileddescriptionGet**: (*string*) Detailed Doxygen documentation for this individual parameter. The two forms (Set/Get) are used for the SetMember and GetMember functions. This field is automatically generated from ITK's Doxygen for members that mirror ITK's members. If the member if a custom, this field will not be modified.
- [OPTIONAL] **no_set_method**: (0 or 1) If set to 1, the filter will not have a Set method for this member
- [OPTIONAL] **no_get_method**: (0 or 1) If set to 1, the filter will not have a Get method for this member
- [OPTIONAL] **no_print**: (0 or 1) If set to 1, this member will not be printed in the ToString method
- [OPTIONAL] **dim_vec**: (0 or 1) If set to 1, this indicates that the member is a std::vector with size equal to the dimension of the input image. As such, it will be printed out correctly and cast to its corresponding ITK type correctly
- [OPTIONAL] **set_as_scalar**: (0 or 1) If parameter is a dim_vec, then this method adds an additional set member method to set the variable as a scalar so that all components are the same
- [OPTIONAL] **enum**: `[ "enum1", "enum2", ... ]` Creates a member type of the class with the array of strings with a type defined by the name of this parameter. This field makes the "type" parameter optional. It is recommended to add a test to ensure the SimpleITK values correspond to the ITK values.
- [OPTIONAL] **custom_itk_cast**: (*string*) Some non-basic types will require a custom cast before the corresponding ITK filter's member can be set. This field allows the developer to specify exactly how the ITK filter sets the member's value from the SimpleITK member's value for example:

```
"custom_itk_cast" :
  "filter->SetObjectValue(static_cast<typename FilterType::PixelType>(this->
↪GetObjectValue()) );"
```

13.3.4 TestFields Test Fields

A basic test object looks like:

```
{
  "tag" : "float",
  "description" : "Dilate a float image",
  "inputA" : "Input/RA-Slice-Float.nrrd",
  "tolerance" : 0.01,
  "settings" : []
}
```

There are three ways of checking the resulting image: against a baseline image, a sha1 hash, or an md5 hash. At least one of these options must be used for each test.

- **tag**: (*string*) The tag to identify this specific test for the filter
- **description**: (*string*) Documentation to describe this specific test
- **inputA**: (*string*) The path from the SOURCE_DIR/Testing/Data directory for the (first) input image
- **inputA_cast**: (*string*) an `sitkPixelIDEnum`, after inputA is read the CastImageFilter is run, to covert the image type.
- [OPTIONAL] **inputB**: (*string*) If **number_of_inputs** is 2, this specifies the path from SOURCE_DIR/Testing/Data to the second input image
- [OPTIONAL] **tolerance**: (*float*) An error tolerance fo be used when comparing the resulting image to the baseline. This is used to specify that the result should be checked against a baseline image
- [OPTIONAL] **md5hash**: (*string*) An md5 hash value to compare the resulting image against.
- [OPTIONAL] **sha1hash**: (*string*) A sha1 hash value to compare the resulting image against.
- [OPTIONAL] **settings**: (*list*) A list of any specific parameters values for this test. Each entry is an object of the form:

```
{ "parameter" : "Sigma",
  "value" : 1.0
}
```

Settings Options

- **parameter**: (*string*) The name of the parameter. This must match the name used when creating the filter
- **value**: (*string* or *number*) The value to assign to the parameter. This can be a number for numerical types or a string representing the constructor for more complex types
- [OPTIONAL] **python_value**: (*string*) For types that require a different syntax for setting the value in python, this will override the string in **value**
- [OPTIONAL] **lua_value**: (*string*) For types that require a different syntax for setting the value in lua, this will override the string in **value**
- [OPTIONAL] **ruby_value**: (*string*) For types that require a different syntax for setting the value in ruby, this will override the string in **value**
- [OPTIONAL] **java_value**: (*string*) For types that require a different syntax for setting the value in java, this will override the string in **value**

- [OPTIONAL] **tcl_value**: (*string*) For types that require a different syntax for setting the value in tcl, this will override the string in **value**
- [OPTIONAL] **R_value**: (*string*) For types that require a different syntax for setting the value in R, this will override the string in **value**
- [OPTIONAL] **dim_vec** (0 or 1): Use this flag if this parameter is a std::vector. If set to 1, the **type** option must also be set
- [OPTIONAL] **no_get_method**: (0 or 1). Set to 1 when the filter does not have a get method, so that this method will not be tested.
- [OPTIONAL] **type**: (*string*): What type the vector should be. Due to constraints with java, this should only be a primitive type. For unsigned types, the java test will convert to the signed equivalent. An example for a `dim_vec` parameter is:

```
{ "parameter" : "Size",
  "dim_vec" : 1,
  "type" : "unsigned int"
  "value" : "{1,2}",
  "java_value" : "{1,2}",
  "tcl_value" : "[list 1 2]",
  "python_value" : "[1,2]",
  "lua_value" : "{1,2}",
  "ruby_value" : "[1,2]"
}
```

13.3.5 Structure Directory Structure

The code generation system is designed to be agnostic of what subdirectory is being parsed. An example of this is BasicFilters. Here will just refer to `$FILTER_DIR`

There are four important subdirectories in `$SimpleITK_SOURCE_DIR/Code/$FILTER_DIR`:

- include: Manually written header files
- src: Manually written c++ files
- json: The input json files used to generate new filters at build time
- templates: The code templates that can be used to generate filters for this directory

When a filter is generated, it produces a .h and a .cxx file that are placed in `$SimpleITK_BUILD_DIR/Code/$FILTER_DIR/include` and `$SimpleITK_BUILD_DIR/Code/$FILTER_DIR/src` respectively.

It is also worth noting that the template files in the templates directory are comprised of a combination of c++ code, lua template expansion directives, and `$(include xxx)` statements. These include statements each pull in a section of template code from a file in $SimpleITK_SOURCE_DIR/ExpandTemplateGenerator/Components. These component files represent pieces of templates that are shared by multiple unique filter template types.

13.3.6 Templates Specialized Templates

The standard template is ImageFilter. This section describes the differences for each of the other template types.

- **Region Growing Image Filters**: The region growing filters support an additional set of methods for manipulating the seed list. Each region growing filter has the SetSeed, AddSeed, and ClearSeeds methods. See sitkConnectedThresholdImageFilter.h for details.
- **Kernel Image Filters**: The kernel image filters use a KernelType and have an external enum which allows the user to select which kernel to use. See sitkErodeObjectMorphologyImageFilter.h for details.
- **Dual Image Filter**: The dual image filter template's usage should be avoided because it instantiates the combination of two pixel type lists.

CHAPTER

FOURTEEN

EXAMPLES

14.1 Hello World

14.1.1 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 the image with Fiji.

14.1.2 Sample Output

Running the python code produces this image:



Fig. 1: **Input Image**

14.1.3 Code

C#

C++

Java

Lua

Python

R

Ruby

Tcl

```
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 );

        // 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 );

        // Display the results
```

(continues on next page)

(continued from previous page)

```
        if (Environment.GetEnvironmentVariable("SITK_NOSHOW") == null)
          SimpleITK.Show( image, "Hello World: CSharp", true );

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

  }
}
```

```
// 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[] ) {

  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 = { 80.0, 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 = { 30.0, 20.0 };
  std::vector<double> mouthCenter = { 64.0, 76.0 };
  sitk::Image mouth = 255 - sitk::BinaryThreshold(
```

(continues on next page)

(continued from previous page)

```
                              sitk::GaussianSource(pixelType, imageSize, mouthRadii,␣
↪mouthCenter),
                              200, 255, 255 );

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

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

  // Display the results
  if (getenv("SITK_NOSHOW") == NULL)
    sitk::Show( image, "Hello World: C++", true );

}
```

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

class HelloWorld {

  public static void main(String argv[]) {

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

    // Create a face image
    double[] fsize = {64., 64.};
    VectorDouble faceSize = new VectorDouble( fsize );
    double[] fcenter = {64., 64.};
    VectorDouble faceCenter = new VectorDouble( fcenter );
    Image face = SimpleITK.gaussianSource( pixelType, imageSize, faceSize, faceCenter );

    // Create eye images
    double[] esize = {5., 5.};
    VectorDouble eyeSize = new VectorDouble( esize );
    double[] e1center = {48., 48.};
    VectorDouble eye1Center = new VectorDouble( e1center );
    double[] e2center = {80., 48.};
    VectorDouble eye2Center = new VectorDouble( e2center );
    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 );
```

(continues on next page)

(continued from previous page)

```
    face = SimpleITK.binaryThreshold( face, 200., 255., (short) 255 );

    // Create the mouth
    double[] mradius = {30., 20.};
    VectorDouble mouthRadius = new VectorDouble( mradius );
    double[] mcenter = {64., 76.};
    VectorDouble mouthCenter = new VectorDouble( mcenter );
    Image mouth = SimpleITK.gaussianSource( pixelType, imageSize, mouthRadius,␣
↪mouthCenter );
    mouth = SimpleITK.binaryThreshold( mouth, 200, 255, (short) 255 );
    mouth = SimpleITK.subtract( 255, mouth );

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

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

    // Display the results
    if(System.getenv("SITK_NOSHOW") == null) {
      SimpleITK.show( image, "Hello World: Java", true );
    }
  }

}
```

```
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
```

(continues on next page)

(continued from previous page)

```
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 )

-- 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
if os.getenv("SITK_NOSHOW") == nil then
    sitk.Show( image, "Hello World: Lua", true )
end
```

```
import os
import sys
import SimpleITK as sitk


def main(args):
    # Create an image
```

(continues on next page)

(continued from previous page)

```
    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]
    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
    return {"output_image": image}
```

```
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
```

(continues on next page)

(continued from previous page)

```
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
if(Sys.getenv("SITK_NOSHOW") == "") {
    Show(image, "Hello World: R", debugOn=TRUE)
}
```

```
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
```

(continues on next page)

(continued from previous page)

```
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 )


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

if (ENV["SITK_NOSHOW"] == "")
    Simpleitk.show( image, "Hello World: Ruby", true )
end
```

```
# 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 ]
```

(continues on next page)

(continued from previous page)

```
# 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
if { ![info exists ::env(SITK_NOSHOW)] } {
    Show $img "Hello World: TCL" 1
}
```

14.2 A Simple C++/CMake Example

Building a C++ SimpleITK application is supported by using CMake to configure the build process. CMake can generate project buildsystems for Unix Makefiles, Ninja, Microsoft Visual Studio, or Mac OS Xcode.

Here is a basic **CMakeLists.txt** configuration file for building a SimpleITK C++ example program.

```
cmake_minimum_required(VERSION 3.16.3)

project(sitk_example)

find_package(SimpleITK)
add_executable ( sitk_example sitk_example.cxx )
target_link_libraries ( sitk_example ${SimpleITK_LIBRARIES} )
```

Here is our SimpleITK C++ example file **sitk_example.cxx**.

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

(continues on next page)

(continued from previous page)

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

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

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

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

    // Add a Gaussian blob to the image
    std::vector<double> blobSize ( 2, 64.0 );
    std::vector<double> blobCenter ( 2, 64.0 );
    image = image + sitk::GaussianSource( pixelType, imageSize, blobSize, blobCenter );

    // Write the image as a PNG file
    sitk::WriteImage(image, "blob.png" );

}
```

Invoking **cmake** as below in the same directory as these files will create the project buildsystem:

```
cmake .
```

14.2.1 Required CMake Variables

Running CMake will produce a **CMakeCache.txt** file that is the CMake configuration. Within that text file there are two variables needed to build a SimpleITK program, **SimpleITK_DIR** and **ITK_DIR**. SimpleITK_DIR needs to be set to the directory that contains the file `SimpleITKConfig.cmake`, and ITK_DIR needs to be the directory containing the file `ITKConfig.cmake`.

CMake attempts to find these files automatically, but it is unlikely to find them unless SimpleITK and ITK are installed into the system.

If SimpleITK has been built using the SuperBuild process, as described in *Building SimpleITK*, then these required directories can be found in the SuperBuild location. The SimpleITK_DIR variable should refer to the directory SimpleITK-build within the SuperBuild location, and the ITK_DIR variable should refer to the directory ITK-build in the SuperBuild location.

Here are what the settings in a CMakeCache.txt file should look like:

```
//The directory containing a CMake configuration file for ITK.
ITK_DIR:PATH=path_to_simpleitk_build_dir/ITK-build

//The directory containing a CMake configuration file for SimpleITK.
SimpleITK_DIR:PATH=path_to_simpleitk_build_dir/SimpleITK-build
```

14.2.2 Compiling using CMake

A SimpleITK program can also be compiled by running CMake. Here is the command to perform the build in the current directory:

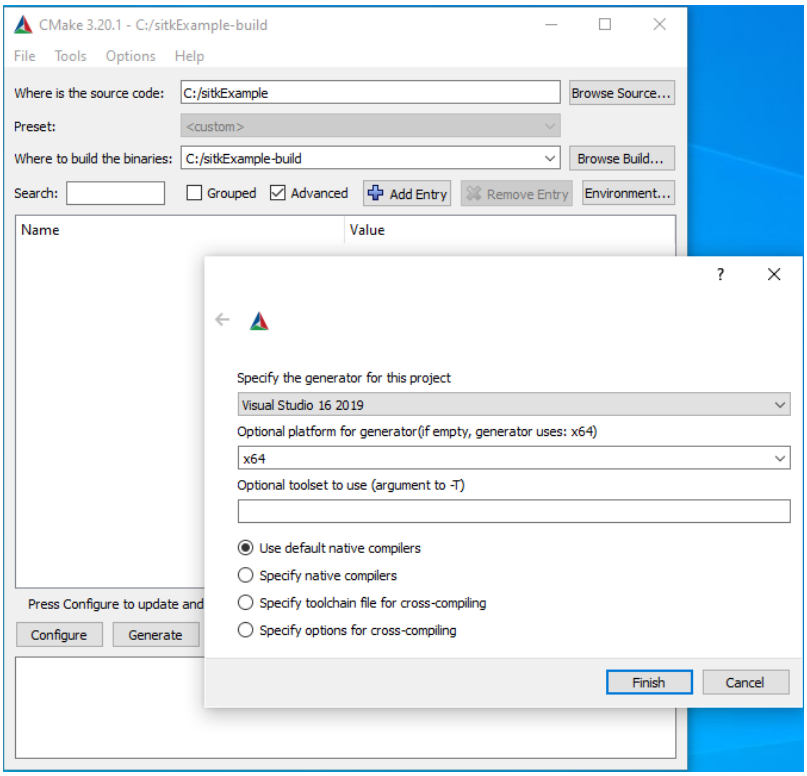
```
cmake --build . --config Release
```

The 'config' parameter could also be set to Debug, RelWithDebInfo or MinSizeRel. On Unix-type systems Release is the default. On Windows the config must be specified explicitly. We recommend Release or MinSizeRel, since SimpleITK compiled in Debug mode is several orders of magnitude slower.

Of course traditional methods can be used to build a SimpleITK program (e.g. make on Unix, or Visual Studio on Windows), but it is often convenient to be able to use the same build command on all system types.

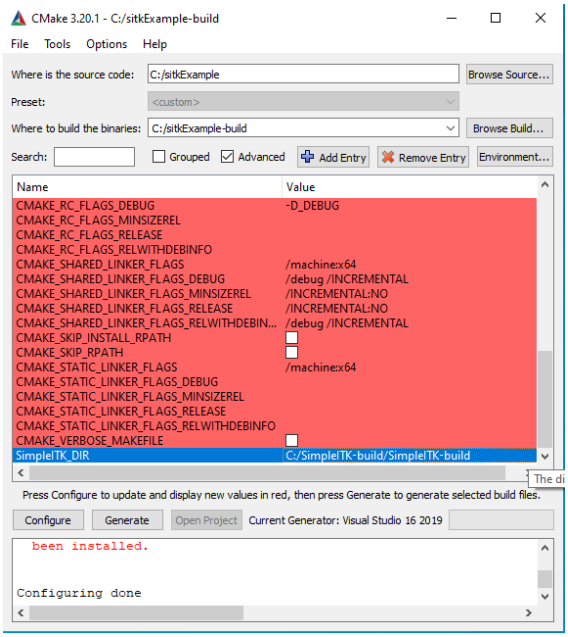
14.2.3 Using CMake-gui on Windows

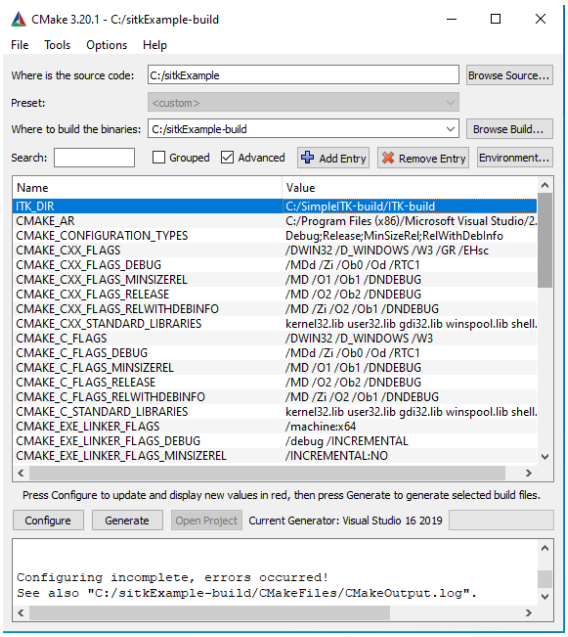
In addition to the standard command-line CMake, a graphical-user-interface (GUI) version of CMake is also available. Here is what the CMake GUI looks like building our same example:



We have placed the files sitk_example.cxx and CMakeLists.txt in the folder `C:sitkExample`, and in CMake we have set the **source** directory to point there. Also, in CMake, we have set the **build** directory to point to `C:sitkExample-Build`.

After the user specifies the source and build directories, CMake will ask to select which compiler generator to use. The options are determined by what versions of Visual Studio are installed on the system, in this example Visual Studio 16 2019. The user must specify **x64** for the platform generator, as the default option (x86) is not supported by SimpleITK.

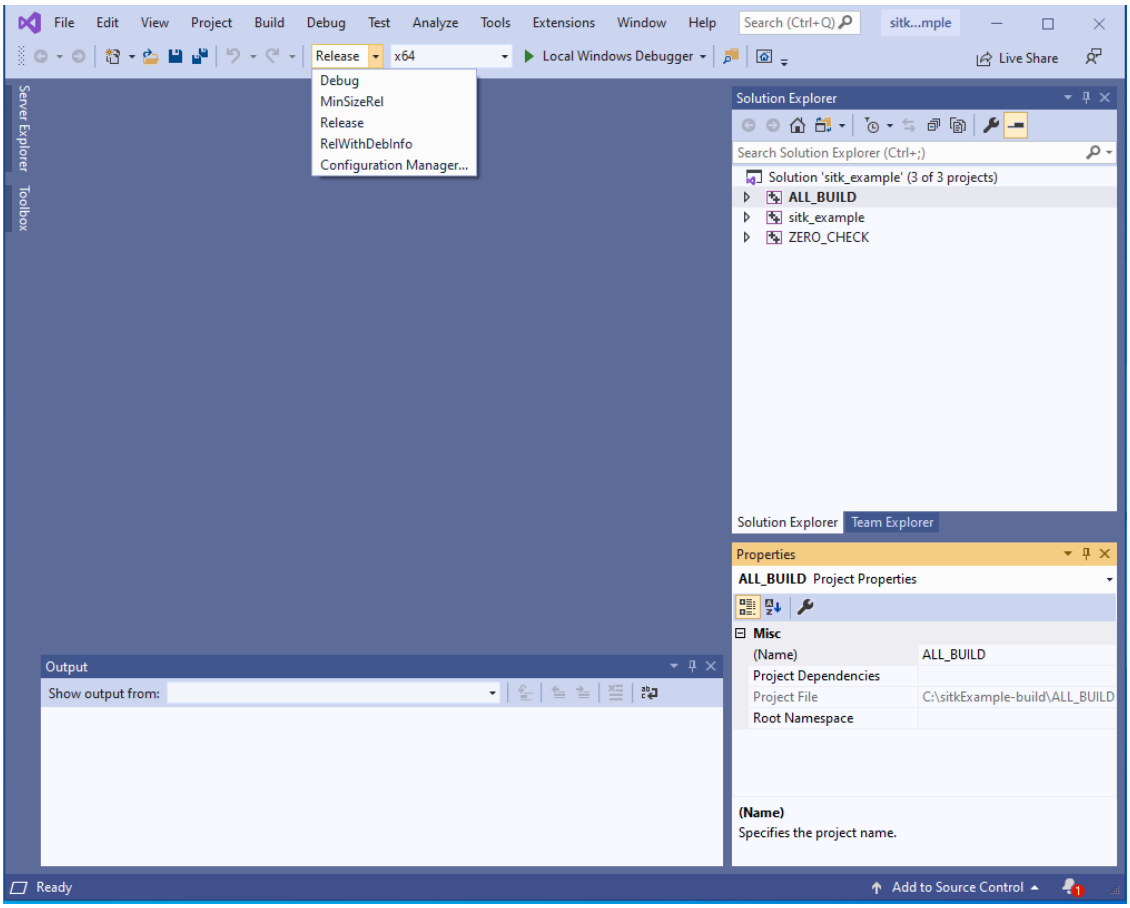




After clicking **Configure**, CMake will attempt to find the SimpleITK library. That can be set in the `SimpleITK_DIR` variable, as seen in the above left image. CMake is looking for the directory containing the `SimpleITKConfig.cmake` file, in our example `C:SimpleITK-buildSimpleITK-build`.

Clicking Configure again, CMake will then try to find the ITK library for the `ITK_DIR` variable (see above right). CMake is looking for the directory containing `ITKConfig.cmake`. For a SimpleITK SuperBuild, as we have done, `ITK_DIR` will be the `ITK-build` directory within the SuperBuild, in our case `C:SimpleITK-buildITK-build`.

Once the `SimpleITK_DIR` and `ITK_DIR` variables have been set, we can click **Generate** to create the Visual Studio project file, `ALL_BUILD.vcxproj`.



After opening `ALL_BUILD.vcxproj` in Visual Studio, the user should select a build type in Visual Studio. We recommend choosing **Release** for performance reasons. Then the user should select the **x64** build platform, not the unsupported default, x86. Many man-hours and cpu cycles have been wasted building SimpleITK without setting the proper platform.

14.3 C++ InPlace Filter Execution

14.3.1 Overview

With the C++ interface of SimpleITK, some filters and image operators can run "inplace" to reduce memory usage and improve performance. When a filter is run in-place the memory allocated to the image buffer of the input image is reused for the output. Since the output buffer is not separately allocated, the peak memory usage can be reduced by 50%. The performance improvement of running an algorithm in-place comes from not only removing the cost of memory allocating but also reduced memory accessed during algorithm execution. Speed up of 1.5x are common while speedup of 10x have been observed.

The SimpleITK filters automatically run in-place with rvalue references. When a temporary result is passed as an input argument to a filter (procedure or a filter's *Execute* method), the temporary's buffer will be re-used for the output. Move semantics and rvalue references are an advanced C++ topic which require additional studying to fully understand the details. Procedures and *Execute* methods with rvalue references (*&&* before argument name) as the first argument support the in-place execution. The following is an example of two procedural methods which support execution in-place.

```
Image UnaryMinus(Image &&image1);
Image SmoothingRecursiveGaussian(Image &&image1, std::vector<double> sigma, bool␣
↪normalizeAcrossScale);
```

Additionally, most C++ overloaded Image numeric operators such as +, -, *, %, etc. support in-place execution.

The SimpleITK Image class supports move semantics with the constructor and assignment operator. Since the Image class also uses a lazy copy-on-write implement for the constructors and assignment there is not a significant direct performance benefit using these operations. However, the move semantics can be useful to reduce the number of Image objects referencing the same image buffer to prevent unnecessary copies. Lastly, if a rvalue Image object is passed to a filter and that Image object's buffer is not unique, then the image buffer will be copied before the filter is executed in-place. Generally, copying the image then executing the filter in-place is faster than executing the not in-place implementation.

The following example demonstrates implicit and explicit use of C++ rvalues to execute filter in-place for improve performance.

14.3.2 Code

C++

```
#include "SimpleITK.h"
#include "sitkImageOperators.h"

#include <cstdlib>
#include <iostream>

namespace sitk = itk::simple;


// Rescale the intensity range for image to -1 to 1.
sitk::Image MinusOneToOne(const sitk::Image &image)
{
  sitk::MinimumMaximumImageFilter mm;
  mm.Execute(image);

  const double shift = 0.5 * (mm.GetMinimum() + mm.GetMaximum());
  const double scale = (mm.GetMaximum() - mm.GetMinimum())*2.0;

  // The CastImageFilter does not operate in-place, so the image argument to
  // this function is declared as a constant reference.
  //
  // The Image operators use in-place execution with the rvalue reference
  // returned from the Cast procedure.
  return  (sitk::Cast(image, sitk::sitkFloat32) - shift) / scale;
}


// Compute the difference between two Gaussian convoluted images.
sitk::Image DifferenceOfGaussian(sitk::Image image, double sigma1, double sigma2)
{
  sitk::SmoothingRecursiveGaussianImageFilter g1;
  g1.SetSigma(sigma1);
```

(continues on next page)

(continued from previous page)

```
  sitk::SmoothingRecursiveGaussianImageFilter g2;
  g2.SetSigma(sigma2);

  // This temporary is create to ensure the method is executed before image is
  // moved and made invalid.
  sitk::Image &&temp = g1.Execute(image);

  // Both Subtract and g2 are executed in-place.
  return sitk::Subtract(temp, g2.Execute(std::move(image)));
}

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

  //
  // Check command line parameters
  //
  if( argc < 5 )
  {
    std::cerr << "Missing Parameters " << std::endl;
    std::cerr << "Usage: " << argv[0];
    std::cerr << " inputImage outputImage lowerSigma upperSigma" << std::endl;
    return 1;
  }


  const double sigma1 = atof(argv[3]);
  const double sigma2 = atof(argv[4]);

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


  // The below explicitly declares the image variable as a rvalue temporary.
  // Having a rvalue typed variable enables explicit declaration of temporary
  // temporary images used between filters.Alternatively this could be written
  // in one line at the sacrifice of readability and explicitness:
  // image = DifferenceOfGaussian(MinusOneToOne(reader.Execute()), sigma1, sigma2);
  //

  sitk::Image &&image = reader.Execute();
  image = MinusOneToOne(image);
  image = DifferenceOfGaussian(image, sigma1, sigma2);


  //
  // Set up writer
  //
```

(continues on next page)

(continued from previous page)

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

  return 0;
}
```

14.4 CSharp Integration

14.4.1 Overview

This example illustrates how to transfer image information from and to a C# buffer.

14.4.2 Code

C#

```
// Get and Set Buffer of SimpleITK Image

using System;
using System.Runtime.InteropServices;

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

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

            if (args.Length < 1)
            {
                Console.WriteLine("Usage: inputImage outputImage");
                return;
            }

            string inputFilename = args[0];
            string outputFilename = args[1];

            // Read input image
            SitkImage input = SimpleITK.ReadImage(inputFilename);

            // Cast so 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++)
```

(continues on next page)

(continued from previous page)

```
            {
                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
            //     }
            //   }
            // }

            // (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);

            // Set buffer of new SimpleITK Image from managed array.
            // bufferAsArray could also have come from a bmp,png,etc...

            uint width = input.GetWidth();
            uint height = input.GetHeight();

            SitkImage outImage = new SitkImage(width, height, PixelId.sitkFloat32);
            IntPtr outImageBuffer = outImage.GetBufferAsFloat();
```

(continues on next page)

(continued from previous page)

```
            Marshal.Copy(bufferAsArray, 0, outImageBuffer, (int)(size[0] * size[1]));

            //
            // Write out the resulting file
            //
            outImage = SimpleITK.RescaleIntensity(outImage, 0, 255);
            outImage = SimpleITK.Cast(outImage, PixelId.sitkUInt8);

            SimpleITK.WriteImage(outImage, outputFilename);


        }
    }
}
```

14.5 DemonsRegistration1

14.5.1 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*.

14.5.2 Code

C++

Python

R

```
// 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::DemonsRegistrationFilter &m)
    : m_Filter(m)
    {}

  void Execute( ) override
    {
      // 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;
```

(continues on next page)

(continued from previous page)

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

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

  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;
}
```

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

import SimpleITK as sitk
import sys
import os


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


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

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

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

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

# The basic Demons Registration Filter
# Note there is a whole family of Demons Registration algorithms included in
# SimpleITK
demons = sitk.DemonsRegistrationFilter()
```

(continues on next page)

(continued from previous page)

```
demons.SetNumberOfIterations(50)
# Standard deviation for Gaussian smoothing of displacement field
demons.SetStandardDeviations(1.0)

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

displacementField = demons.Execute(fixed, moving)

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

outTx = sitk.DisplacementFieldTransform(displacementField)

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

if "SITK_NOSHOW" not 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 // 2.0 + simg2 // 2.0)
    sitk.Show(cimg, "DeformableRegistration1 Composition")
```

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

library(SimpleITK)

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

args <- commandArgs( TRUE )

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

(continues on next page)

(continued from previous page)

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

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


matcher <- HistogramMatchingImageFilter()
matcher$SetNumberOfHistogramLevels(1024)
matcher$SetNumberOfMatchPoints(7)
matcher$ThresholdAtMeanIntensityOn()
moving <- matcher$Execute(moving, fixed)

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

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

displacementField <- demons$Execute( fixed, moving )

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

outTx <- DisplacementFieldTransform( displacementField )

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

14.6 DemonsRegistration2

14.6.1 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*.

14.6.2 Code

C++

Python

R

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

  void Execute( ) override
    {
      // 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 )
```

(continues on next page)

(continued from previous page)

```
    {
    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 );
  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 );
    }
```

(continues on next page)

(continued from previous page)

```
  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;
}
```

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

import SimpleITK as sitk
import sys
import os


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


if len(sys.argv) < 4:
    print(
        "Usage:",
        sys.argv[0],
        "<fixedImageFilter> <movingImageFile>",
        "[initialTransformFile] <outputTransformFile>",
    )
    sys.exit(1)

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

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

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

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

(continues on next page)

(continued from previous page)

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

outTx = sitk.DisplacementFieldTransform(displacementField)

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

if "SITK_NOSHOW" not 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 // 2.0 + simg2 // 2.0)
    sitk.Show(cimg, "DeformableRegistration1 Composition")
```

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

library(SimpleITK)

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

(continues on next page)

(continued from previous page)

```
args <- commandArgs( TRUE )

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

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

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

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

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

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


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

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

    displacementField <- toDisplacementFilter$Execute(initialTransform)

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


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

outTx <- DisplacementFieldTransform(displacementField)
```

(continues on next page)

(continued from previous page)

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

14.7 Read Image Meta-Data Dictionary and Print

14.7.1 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 ImageFileReader'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*.

14.7.2 Sample Output

Running the above Python script with the image "docs/images/Image0075.dcm" produces the following output:

```
(0008|0000) = = "406"
(0008|0005) = = "ISO_IR 100"
(0008|0008) = = "ORIGINAL\PRIMARY\OTHER"
(0008|0016) = = "1.2.840.10008.5.1.4.1.1.4"
(0008|0018) = = "1.2.840.113619.2.133.1762890640.1886.1055165016.74"
(0008|0020) = = "20030625"
(0008|0021) = = "20030625"
(0008|0022) = = "20030625"
(0008|0023) = = "20030625"
(0008|0030) = = "152734"
(0008|0031) = = "153128"
(0008|0032) = = "153128"
(0008|0033) = = "153128"
(0008|0050) = = ""
(0008|0060) = = "MR"
(0008|0070) = = "GE MEDICAL SYSTEMS"
(0008|0080) = = "crd "
(0008|0090) = = ""
(0008|1010) = = "sig3ow0 "
(0008|1030) = = "Tech_1"
(0008|103e) = = "head scan "
(0008|1070) = = "RPM "
(0008|1090) = = "GENESIS_SIGNA "
(0009|0000) = = "184"
```

(continues on next page)

(continued from previous page)

```
(0009|0010) = = "GEMS_IDEN_01"
(0009|1001) = = "GE_GENESIS_FF "
(0009|1002) = = "sig3"
(0009|1004) = = "SIGNA "
(0009|1027) = = "1056555088"
(0009|1030) = = "18387rdmr3"
(0009|1031) = = "999 "
(0009|10e3) = = "1.2.840.113619.1.1.4.1762890640"
(0009|10e6) = = "09"
(0009|10e7) = = "3238884814"
(0009|10e9) = = "1056555088"
(0010|0000) = = "82"
(0010|0010) = = "Wes Turner"
(0010|0020) = = "1111"
(0010|0030) = = ""
(0010|0040) = = "O "
(0010|1010) = = "000Y"
(0010|1030) = = "68.039"
(0010|21b0) = = ""
(0011|0000) = = "30"
(0011|0010) = = "GEMS_PATI_01"
(0011|1010) = = "0"
(0018|0000) = = "428"
(0018|0020) = = "GR"
(0018|0021) = = "SS\SP "
(0018|0022) = = "FC\VB_GEMS\VASCTOF_GEMS\PFF "
(0018|0023) = = "3D"
(0018|0025) = = "N "
(0018|0050) = = " 1.6"
(0018|0080) = = "23"
(0018|0081) = = "10"
(0018|0083) = = " 1"
(0018|0084) = = " 6.38883e+08"
(0018|0085) = = "H1"
(0018|0086) = = " 1"
(0018|0087) = = "15000 "
(0018|0088) = = "1.6 "
(0018|0091) = = "0 "
(0018|0093) = = " 100"
(0018|0094) = = "75"
(0018|0095) = = " 97.6562"
(0018|1000) = = "0000018387rdmr3 "
(0018|1020) = = "09"
(0018|1050) = = " 1.14585"
(0018|1088) = = "0 "
(0018|1090) = = "0 "
(0018|1094) = = "0 "
(0018|1100) = = " 220"
(0018|1250) = = "HEAD"
(0018|1251) = = "HEAD"
(0018|1310) = = "0\256\192\0"
(0018|1312) = = "ROW "
```

(continues on next page)

(continued from previous page)

```
(0018|1314) = = "25"
(0018|1315) = = "N "
(0018|1316) = = "0.004169"
(0018|5100) = = "HFS "
(0019|0000) = = "1004"
(0019|0010) = = "GEMS_ACQU_01"
(0019|100f) = = "418.4 "
(0019|1011) = = "0"
(0019|1012) = = "10"
(0019|1017) = = "16"
(0019|1018) = = "A "
(0019|1019) = = "75.073532 "
(0019|101a) = = "P "
(0019|101b) = = "-93.324310"
(0019|101e) = = "165 "
(0019|105a) = = "4.14274e+08"
(0019|107d) = = "0 "
(0019|107e) = = "1"
(0019|107f) = = "0 "
(0019|1081) = = "0"
(0019|1084) = = "0.218518"
(0019|1085) = = "1"
(0019|1087) = = "0 "
(0019|1088) = = "0"
(0019|108a) = = "12"
(0019|108b) = = "14"
(0019|108d) = = "0 "
(0019|108f) = = "0"
(0019|1090) = = "0"
(0019|1091) = = "0 "
(0019|1092) = = "0"
(0019|1093) = = "638883410 "
(0019|1094) = = "97"
(0019|1095) = = "12"
(0019|1096) = = "14"
(0019|1097) = = "0"
(0019|1098) = = "2"
(0019|109b) = = "1"
(0019|109c) = = "3dtof "
(0019|109d) = = "20021226173544"
(0019|109e) = = "3dtof "
(0019|109f) = = "1"
(0019|10a0) = = "0"
(0019|10a1) = = "0"
(0019|10a2) = = "5196"
(0019|10a3) = = "0"
(0019|10a4) = = "0"
(0019|10a7) = = "0 "
(0019|10a8) = = "0 "
(0019|10a9) = = "0 "
(0019|10aa) = = "0 "
(0019|10ab) = = "0 "
```

(continues on next page)

(continued from previous page)

```
(0019|10ac) = = "0 "
(0019|10ad) = = "0 "
(0019|10ae) = = "0 "
(0019|10af) = = "0 "
(0019|10b0) = = "0 "
(0019|10b1) = = "0 "
(0019|10b2) = = "0 "
(0019|10b3) = = "0 "
(0019|10b4) = = "0 "
(0019|10b5) = = "0 "
(0019|10b6) = = "0 "
(0019|10b7) = = "0 "
(0019|10b8) = = "0 "
(0019|10b9) = = "0 "
(0019|10ba) = = "0 "
(0019|10bb) = = "0 "
(0019|10bc) = = "0 "
(0019|10bd) = = "0 "
(0019|10be) = = "0 "
(0019|10c0) = = "0"
(0019|10c1) = = "0"
(0019|10c2) = = "9990"
(0019|10c3) = = "9990"
(0019|10c4) = = "9990"
(0019|10c5) = = "9990"
(0019|10c6) = = "9990"
(0019|10c7) = = "9990"
(0019|10c8) = = "0"
(0019|10c9) = = "0"
(0019|10ca) = = "0"
(0019|10cb) = = "0"
(0019|10cc) = = "0"
(0019|10cd) = = "0"
(0019|10ce) = = "2"
(0019|10cf) = = "0"
(0019|10d2) = = "0"
(0019|10d3) = = ""
(0019|10d5) = = "1"
(0019|10d7) = = "0"
(0019|10d8) = = "0"
(0019|10d9) = = "0 "
(0019|10df) = = "0 "
(0019|10e0) = = "0 "
(0019|10e2) = = "0 "
(0019|10f2) = = "0"
(0019|10f9) = = "97"
(0020|0000) = = "340"
(0020|000d) = = "1.2.840.113619.2.133.1762890640.1886.1055165015.961"
(0020|000e) = = "1.2.840.113619.2.133.1762890640.1886.1055165015.999"
(0020|0010) = = "361 "
(0020|0011) = = " 3"
(0020|0012) = = " 0"
```

(continues on next page)

(continued from previous page)

```
(0020|0013) = = "75"
(0020|0032) = = "-112\ -21.688\ 126.894"
(0020|0037) = = "  1\ 0\ 0\ 0\0.466651\ -0.884442"
(0020|0052) = = "1.2.840.113619.2.133.1762890640.1910.1055164903.673"
(0020|0060) = = ""
(0020|0110) = = " 0"
(0020|1040) = = ""
(0020|1041) = = "-29.6444568634"
(0021|0000) = = "328"
(0021|0010) = = "GEMS_RELA_01"
(0021|1003) = = "0"
(0021|1005) = = "09"
(0021|1007) = = "2539681714"
(0021|1018) = = "09"
(0021|1019) = = "3223022508"
(0021|1035) = = "2"
(0021|1036) = = "9"
(0021|1037) = = "16"
(0021|104f) = = "120"
(0021|1050) = = "0"
(0021|1051) = = "0.000000"
(0021|1052) = = "0.000000"
(0021|1053) = = "0.000000"
(0021|1056) = = "1"
(0021|1057) = = "124"
(0021|1058) = = "0"
(0021|1059) = = "0"
(0021|105a) = = "0"
(0021|105b) = = "180 "
(0021|105c) = = "45"
(0021|105d) = = "50000 "
(0021|105e) = = "0 "
(0021|105f) = = "0 "
(0021|1081) = = "0 "
(0021|1082) = = "0 "
(0021|1083) = = "0 "
(0021|1084) = = "0 "
(0023|0000) = = "58"
(0023|0010) = = "GEMS_STDY_01"
(0023|1070) = = "0"
(0023|1074) = = "0"
(0023|107d) = = "0"
(0025|0000) = = "128"
(0025|0010) = = "GEMS_SERS_01"
(0025|1006) = = "10"
(0025|1007) = = "120"
(0025|1010) = = "0"
(0025|1011) = = "1"
(0025|1014) = = "0"
(0025|1017) = = "0"
(0025|1018) = = "0"
(0025|1019) = = "120"
```

(continues on next page)

(continued from previous page)

```
(0025|101a) = = "sig3ow0 "
(0027|0000) = = "306"
(0027|0010) = = "GEMS_IMAG_01"
(0027|1006) = = "0"
(0027|1010) = = "0"
(0027|1030) = = ""
(0027|1031) = = "9"
(0027|1032) = = "10"
(0027|1033) = = "1032"
(0027|1035) = = "16"
(0027|1036) = = "24"
(0027|1040) = = "P "
(0027|1041) = = "-29.6445"
(0027|1042) = = "2"
(0027|1043) = = "-29.6445"
(0027|1044) = = "29.6041"
(0027|1045) = = "-0"
(0027|1046) = = "0.884443"
(0027|1047) = = "-0.466649"
(0027|1048) = = "-108"
(0027|1049) = = "21.688"
(0027|104a) = = "126.894"
(0027|104b) = = "-108"
(0027|104c) = = "-80.9769"
(0027|104d) = = "-67.6862"
(0027|1060) = = "256"
(0027|1061) = = "192"
(0027|1062) = = "1"
(0028|0000) = = "148"
(0028|0002) = = "1"
(0028|0004) = = "MONOCHROME2 "
(0028|0010) = = "256"
(0028|0011) = = "256"
(0028|0030) = = "  0.85939\0.859375"
(0028|0100) = = "16"
(0028|0101) = = "16"
(0028|0102) = = "15"
(0028|0103) = = "1"
(0028|0120) = = "0"
(0028|1050) = = "96"
(0028|1051) = = " 192"
(0029|0000) = = "102"
(0029|0010) = = "GEMS_IMPS_01"
(0029|1015) = = "0"
(0029|1016) = = "0"
(0029|1017) = = "0"
(0029|1018) = = "0"
(0029|1026) = = "2"
(0029|1034) = = "16384"
(0029|1035) = = "0"
(0043|0000) = = "5162"
(0043|0010) = = "GEMS_PARM_01"
```

(continues on next page)

(continued from previous page)

```
(0043|1001) = = "1"
(0043|1002) = = "0"
(0043|1003) = = "0"
(0043|1004) = = "0"
(0043|1006) = = "0"
(0043|1007) = = "0"
(0043|1008) = = "0"
(0043|1009) = = "0"
(0043|100a) = = "1"
(0043|100b) = = "0 "
(0043|100c) = = "66"
(0043|100d) = = "0 "
(0043|100e) = = "0 "
(0043|100f) = = "0.0874073 "
(0043|1010) = = "0"
(0043|101c) = = "0"
(0043|101d) = = "0"
(0043|1028) = = "TVIgUmVjb24gPDA+AC8zNjEvMy8xAAAAAAAAACkAAED/////
↪As64qALOuNgB1WTkAKp5KALNAAAAkmMIAdewqAAAAAAAAAAAAAAAAACTWfg="
(0043|1029) = = "AAAAAUImFTMAJgAAAMAAAAAAA/
↪5B+AGWAbQB+wIEAikCEgIxAkwCLQIqAiUCFwH8AeABzAG9AZoBlwGpAYYBSgFQAUgBHQEqAQoBCgDnAOoA/
↪gDcANIAuwC/
↪AKwAoACtAKQAkQCCAHEAbwBpAGQAVwBgAGEAVgBRAEcATgA7ADYAUwBDAD4AMAA0ADQAMQA5ADQAMAAqACkAMQAwACIAHwAlABYANw
↪ADIARAA7AD0AOwA2AEcANQBHADMAQwAzADUANABEAEYAOgA/
↪AD8ATABDAEYARABDADwAUgBBAEsAQwBHAEYAWgBNAEgAMwBDADEAXABCAFIAVQA/
↪AEcARgBdAFEARgBZAFYASgBTAF4AVwBVAGUAawBWAFoATwBNAFgAVABXAFEAUgBYAFQAZQBtAFgAagBUAFoAaQBUAF8AWgBhAFcATw
↪AGAAYABlAG0AcABZAGcAYgBwAEkAaABhAGoAYQBzAGYAcgBwAFoAWwBuAGcAVQBnAG0AeABvAGwAYwBTAG0AaABmAGUAXwBhAF0AWG
↪"
(0043|102a) = =
↪"VUEAAAAAACwAAgAAAAAALAAAAUAAAQAAAAABbAAACAAAAwAAAAAJbAAAAAgAAAAAAAADAAAAA4AAABhJhSWUgAAAAwAAAAOAAAAY
↪3FUOAAAAAA+jmXwAAAACAAAAAAAAAAAAAAAA
```

(continued from previous page)

```
(0043|1038) = = "0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"
(0043|1039) = = "  0\ 0\ 0\ 7"
(0043|1060) = = "  0\ 0\ 0\10\ 0\ 0\ 0\ 0"
(0043|1061) = = "1.2.840.113619.2.133.1762890640.1886.1055165015.961"
(0043|1062) = = "361 "
(0043|106f) = = " 0\ 0\ 0"
(7fe0|0000) = = "131084"
Image Size: (256, 256, 1)
Image PixelType: 16-bit signed integer
```

14.7.3 Code

Python

R

```python
import SimpleITK as sitk
import sys
def main(args):
    if len(args) < 2:
        print("Usage: DicomImagePrintTags <input_file>")
        sys.exit(1)

    reader = sitk.ImageFileReader()

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

    reader.ReadImageInformation()

    for k in reader.GetMetaDataKeys():
        v = reader.GetMetaData(k)
        print(f'({k}) = = "{v}"')

    print(f"Image Size: {reader.GetSize()}")
    print(f"Image PixelType: {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)
}
```

(continues on next page)

(continued from previous page)

```
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')
```

14.8 Dicom Series Reader

14.8.1 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*.

14.8.2 Code

C#

C++

Java

Lua

Python

R

```
using System;
using System.Diagnostics;
using itk.simple;
using sitk = itk.simple.SimpleITK;


namespace itk.simple.examples {
```

(continues on next page)

(continued from previous page)

```
    class DicomSeriesReader {

static void Main(string[] args) {
  try {
    if (args.Length < 2) {
      Console.WriteLine( "Usage: DicomSeriesReader <input_directory> <output_file>" );
      return;
    }

    Console.WriteLine( "Reading Dicom directory: " + args[0] );
    ImageSeriesReader reader = new ImageSeriesReader();

    VectorString dicom_names = ImageSeriesReader.GetGDCMSeriesFileNames( args[0] );
    reader.SetFileNames( dicom_names );

    Image image = reader.Execute();

    VectorUInt32 size = image.GetSize();
    Console.WriteLine( "Image size: " + size[0] + " " + size[1] + " " + size[2] );

    Console.WriteLine( "Writing image: " + args[1] );
    ImageFileWriter writer = new ImageFileWriter();
    writer.SetFileName( args[1] );
    writer.Execute( image );

    if (Environment.GetEnvironmentVariable("SITK_NOSHOW") == null)
      SimpleITK.Show( image, "Dicom Series" );


  } catch (Exception ex) {
    Console.WriteLine( "Usage: DicomSeriesReader <input_directory> <output_file>" );
    Console.WriteLine( ex );
  }
}

    }
}
```

```
// 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";
```

(continues on next page)

(continued from previous page)

```
    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] );

  if (getenv("SITK_NOSHOW") == NULL)
    sitk::Show( image, "Hello World: C++", true );

  return 0;
}
```

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

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

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

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

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

    Image image = imageSeriesReader.execute();

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

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

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

(continues on next page)

(continued from previous page)

```
    if (System.getenv("SITK_NOSHOW") == null) {
      SimpleITK.show( image, "Hello World: Java", true );
    }

  }
}
```

```
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] )

if os.getenv("SITK_NOSHOW") == nil then
    sitk.Show( image, "Hello World: Lua", true )
end
```

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


import SimpleITK as sitk
import sys
import os

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

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

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

image = reader.Execute()
```

(continues on next page)

(continued from previous page)

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

```
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]])

if(Sys.getenv("SITK_NOSHOW") == "") {
    Show(image, "Hello World: R", debugOn=TRUE)
}
```

14.9 Dicom Series Read Modify Write

14.9.1 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 dictionary 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 characteristics (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.

Always represent DICOM tags using lower case hexadecimals. DICOM tags represent hexadecimal numbers, so 0020|000D and 0020|000d are equivalent. The ITK/SimpleITK dictionary is string based, so these are two different keys, case sensitive. When read from a DICOM file the hexadecimal string representations are in lower case. To ensure consistency, always use lower case for the tags.

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

14.9.2 Code

Python

R

```
import SimpleITK as sitk

import sys
import time
import 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(
```

(continues on next page)

(continued from previous page)

```
        '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 operation and requires
#            knowledge of the DICOM standard. This example only modifies some.
#            For a more complete list of tags that need to be modified see:
#                http://gdcm.sourceforge.net/wiki/index.php/Writing_DICOM

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

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

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

(continues on next page)

(continued from previous page)

```
# 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.
# NOTE: Always represent DICOM tags using lower case hexadecimals.
#       DICOM tags represent hexadecimal numbers, so 0020|000D and 0020|000d
#       are equivalent. The ITK/SimpleITK dictionary is string based, so these
#       are two different keys, case sensitive. When read from a DICOM file the
#       hexadecimal string representations are in lower case. To ensure consistency,
#       always use lower case for the tags.
# 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],
                    direction[1],
                    direction[4],
                    direction[7],
                ),  # Image Orientation (Patient)
            )
        ),
    ),
    (
        "0008|103e",
        (
            series_reader.GetMetaData(0, "0008|103e")
            if series_reader.HasMetaDataKey(0, "0008|103e")
            else "" + " Processed-SimpleITK"
        ),
    ),  # Series Description is an optional tag, so may not exist
]
```

(continues on next page)

(continued from previous page)

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

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

```
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.
```

(continues on next page)

(continued from previous page)

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

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

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

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

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

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

# Copy some of the tags and add the relevant tags indicating the change.
# When attempting to copy we are not ensured that the tags exist, in which case
# we get a NULL as an entry in the list returned by lapply, these are removed
# by the Filter.
# For the series instance UID (0020|000e), each of the components is a number, cannot␣
↪start
# with zero, and separated by a '.' We create a unique series ID using the date and time.
# NOTE: Always represent DICOM tags using lower case hexadecimals.
#       DICOM tags represent hexadecimal numbers, so 0020|000D and 0020|000d
#       are equivalent. The ITK/SimpleITK dictionary is string based, so these
#       are two different keys, case sensitive. When read from a DICOM file the
#       hexadecimal string representations are in lower case. To ensure consistency,
#       always use lower case for the tags.
# Tags of interest:
direction <- filtered_image$GetDirection()
```

(continues on next page)

(continued from previous page)

```
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(if(series_reader$HasMetaDataKey(0,
↪"0008|103e")) series_reader$GetMetaData(0,"0008|103e") else "", " Processed-SimpleITK
↪")))) # Series Description is an optional tag, so may not exist

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

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

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

14.10 Dicom Series From Array

14.10.1 Overview

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

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

Writing the 3D image as a DICOM series is done by configuring the meta-data dictionary for each of the slices and then writing it in DICOM format. In our case we generate all of the meta-data to indicate that this series is derived. If the new image has float values we need to encode this via the rescale slope (0028|1053), rescale intercept (0028|1052), and several additional meta-data dictionary values specifying how the values are stored.

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

14.10.2 Code

Python

R

```
import SimpleITK as sitk

import sys
import time
import os
import numpy as np

pixel_dtypes = {"int16": np.int16, "float64": np.float64}


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

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

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

    # Setting the type to CT so that the slice location is preserved and
    # the thickness is carried over.
    image_slice.SetMetaData("0008|0060", "CT")

    # (0020, 0032) image position patient determines the 3D spacing between
```

(continues on next page)

(continued from previous page)

```
    # slices.
    #   Image Position (Patient)
    image_slice.SetMetaData(
        "0020|0032",
        "\\".join(map(str, new_img.TransformIndexToPhysicalPoint((0, 0, i)))),
    )
    #   Instance Number
    image_slice.SetMetaData("0020|0013", str(i))

    # Write to the output directory and add the extension dcm, to force
    # writing in DICOM format.
    writer.SetFileName(os.path.join(out_dir, str(i) + ".dcm"))
    writer.Execute(image_slice)


if len(sys.argv) < 3:
    print(
        "Usage: python "
        + __file__
        + " <output_directory> ["
        + ", ".join(pixel_dtypes)
        + "]"
    )
    sys.exit(1)

# Create a new series from a numpy array
try:
    pixel_dtype = pixel_dtypes[sys.argv[2]]
except KeyError:
    pixel_dtype = pixel_dtypes["int16"]

new_arr = np.random.uniform(-10, 10, size=(3, 4, 5)).astype(pixel_dtype)
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 operation and requires
#            knowledge of the DICOM standard. This example only modifies some.
#            For a more complete list of tags that need to be modified see:
#                  http://gdcm.sourceforge.net/wiki/index.php/Writing_DICOM
#            If it is critical for your work to generate valid DICOM files,
#            It is recommended to use David Clunie's Dicom3tools to validate
#            the files:
#                  http://www.dclunie.com/dicom3tools.html

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

modification_time = time.strftime("%H%M%S")
```

(continues on next page)

(continued from previous page)

```
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],
                    direction[1],
                    direction[4],
                    direction[7],
                ),
            )
        ),
    ),  # Image Orientation
    # (Patient)
    ("0008|103e", "Created-SimpleITK"),  # Series Description
]

if pixel_dtype == np.float64:
    # If we want to write floating point values, we need to use the rescale
    # slope, "0028|1053", to select the number of digits we want to keep. We
    # also need to specify additional pixel storage and representation
    # information.
    rescale_slope = 0.001  # keep three digits after the decimal point
    series_tag_values = series_tag_values + [
        ("0028|1053", str(rescale_slope)),  # rescale slope
        ("0028|1052", "0"),  # rescale intercept
        ("0028|0100", "16"),  # bits allocated
        ("0028|0101", "16"),  # bits stored
        ("0028|0102", "15"),  # high bit
        ("0028|0103", "1"),
    ]  # pixel representation

# Write slices to output directory
list(
```

(continues on next page)

(continued from previous page)

```
    map(
        lambda i: writeSlices(series_tag_values, new_img, sys.argv[1], i),
        range(new_img.GetDepth()),
    )
)

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

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

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

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

library(SimpleITK)

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

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

    # Slice specific tags.
    image_slice$SetMetaData("0008|0012", format(Sys.time(), "%Y%m%d")) # Instance␣
```

(continues on next page)

(continued from previous page)

```
↪Creation Date
    image_slice$SetMetaData("0008|0013", format(Sys.time(), "%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$SetMetaData("0020|0032", paste(new_img$TransformIndexToPhysicalPoint(c(0,
↪0,i)), collapse='\\')) # Image Position (Patient)
    image_slice$SetMetaData("0020|0013", i-1) # Instance Number

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


args <- commandArgs( TRUE )
if (length(args) < 2) {
   stop("Two arguments expected - output_directory pixel_type [int, float]")
}

# Create a new series from an array
image_dim = c(5,4,3)

if( args[[2]] == "int" ) {
  new_arr = array(as.integer(runif(60, min=-10, max=10)), dim=image_dim)
} else if( args[[2]] == "float" ) {
    new_arr = array(runif(60, min=-10, max=10), dim=image_dim)
} else {
  stop("Unexpected pixel type, valid values are [int, float].")
}

new_img <- as.image(new_arr)
new_img$SetSpacing(c(2.5,3.5,4.5))

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

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

(continues on next page)

(continued from previous page)

```
writer$KeepOriginalImageUIDOn()

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

# Copy some of the tags and add the relevant tags indicating the change.
# 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 <- c("0008|0031",modification_time, # Series Time
                       "0008|0021",modification_date, # Series Date
                       "0008|0008","DERIVED\\SECONDARY", # Image Type
                       "0020|000e", paste("1.2.826.0.1.3680043.2.1125.",modification_
↪date,".1",modification_time, sep=''), # Series Instance UID
                       "0020|0037", paste(direction[[1]], direction[[4]], direction[[7]],
↪# Image Orientation (Patient)
                                          direction[[2]],direction[[5]],direction[[8]],␣
↪sep='\\'),
                       "0008|103e", "Created-SimpleITK")

if(args[[2]] == "float") {
    # If we want to write floating point values, we need to use the rescale slope,
↪"0028|1053", to select the
    # number of digits we want to keep. We also need to specify additional pixel storage␣
↪and representation
    # information.
    rescale_slope <- 0.001 #keep three digits after the decimal point
    series_tag_values <- c(series_tag_values,
                           c("0028|1053", paste(rescale_slope), #rescale slope
                             "0028|1052","0",   #rescale intercept
                             "0028|0100", "16", #bits allocated
                             "0028|0101", "16", #bits stored
                             "0028|0102", "15", #high bit
                             "0028|0103", "1")) #pixel representation
}

series_tag_values <- matrix(series_tag_values, nrow=length(series_tag_values)/2, ncol=2,␣
↪byrow=TRUE) # Series Description

# Write slices to output directory
invisible(lapply(1:(new_img$GetDepth()), function(i){writeSlices(series_tag_values, new_
↪img, args[[1]], i)}))

# Re-read the series
# Read the original series. First obtain the series file names using the
# image series reader.
data_directory <- args[[1]]
series_IDs <- ImageSeriesReader_GetGDCMSeriesIDs(data_directory)
if (length(series_IDs)==0) {
```

(continues on next page)

(continued from previous page)

```
    stop("ERROR: given directory \"", data_directory, "\" does not contain a DICOM␣
↪series.")
}
series_file_names <- ImageSeriesReader_GetGDCMSeriesFileNames(data_directory, series_
↪IDs[[1]])

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

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

14.11 Resample and Convert DICOM to Common Image Formats

14.11.1 Overview

This example illustrates the use of SimpleITK for converting a set of DICOM images to other file formats (tif, jpg, png,...). The output file format is specified by the user, and the output image width can also be specified by the user (height is determined from the width as resulting pixel sizes are required to be isotropic). Grayscale images with high dynamic range are rescaled to [0,255] before conversion to the new format. Output is written in the same location as the input image, or to a user specified output directory. An additional csv file mapping between original and converted file names is also written, either to the specified output directory or to the current working directory.

14.11.2 Code

Python

R

```
import argparse
import csv
import functools
import itertools
import multiprocessing
import os
import sys

import SimpleITK as sitk


def convert_image(input_file_name, output_file_name, new_width=None):
    try:
        image_file_reader = sitk.ImageFileReader()
```

(continues on next page)

(continued from previous page)

```
        # only read DICOM images
        image_file_reader.SetImageIO("GDCMImageIO")
        image_file_reader.SetFileName(input_file_name)
        image_file_reader.ReadImageInformation()
        image_size = list(image_file_reader.GetSize())
        if len(image_size) == 3 and image_size[2] == 1:
            image_size[2] = 0
        image_file_reader.SetExtractSize(image_size)
        image = image_file_reader.Execute()
        if new_width:
            original_size = image.GetSize()
            original_spacing = image.GetSpacing()
            new_spacing = [
                (original_size[0] - 1) * original_spacing[0] / (new_width - 1)
            ] * 2
            new_size = [
                new_width,
                int((original_size[1] - 1) * original_spacing[1] / new_spacing[1]),
            ]
            image = sitk.Resample(
                image1=image,
                size=new_size,
                transform=sitk.Transform(),
                interpolator=sitk.sitkLinear,
                outputOrigin=image.GetOrigin(),
                outputSpacing=new_spacing,
                outputDirection=image.GetDirection(),
                defaultPixelValue=0,
                outputPixelType=image.GetPixelID(),
            )
        # If a single channel image, rescale to [0,255]. Also modify the
        # intensity values based on the photometric interpretation. If
        # MONOCHROME2 (minimum should be displayed as black) we don't need to
        # do anything, if image has MONOCRHOME1 (minimum should be displayed as
        # white) we flip # the intensities. This is a constraint imposed by ITK
        # which always assumes MONOCHROME2.
        if image.GetNumberOfComponentsPerPixel() == 1:
            image = sitk.RescaleIntensity(image, 0, 255)
            if image_file_reader.GetMetaData("0028|0004").strip() == "MONOCHROME1":
                image = sitk.InvertIntensity(image, maximum=255)
            image = sitk.Cast(image, sitk.sitkUInt8)
        sitk.WriteImage(image, output_file_name)
        return True
    except BaseException:
        return False


def convert_images(input_file_names, output_file_names, new_width):
    MAX_PROCESSES = 15
    with multiprocessing.Pool(processes=MAX_PROCESSES) as pool:
        return pool.starmap(
            functools.partial(convert_image, new_width=new_width),
```

(continues on next page)

(continued from previous page)

```
            zip(input_file_names, output_file_names),
        )


def positive_int(int_str):
    value = int(int_str)
    if value <= 0:
        raise argparse.ArgumentTypeError(int_str + " is not a positive integer value")
    return value


def directory(dir_name):
    if not os.path.isdir(dir_name):
        raise argparse.ArgumentTypeError(dir_name + " is not a valid directory name")
    return dir_name


def main(argv=None):
    parser = argparse.ArgumentParser(
        description="Convert and resize DICOM files to common image types."
    )
    parser.add_argument(
        "root_of_data_directory",
        type=directory,
        help="Path to the topmost directory containing data.",
    )
    parser.add_argument(
        "output_file_extension",
        help="Image file extension, this determines output file type " "(e.g. png) .",
    )
    parser.add_argument("--w", type=positive_int, help="Width of converted images.")
    parser.add_argument("--od", type=directory, help="Output directory.")
    args = parser.parse_args(argv)

    input_file_names = []
    for dir_name, subdir_names, file_names in os.walk(args.root_of_data_directory):
        input_file_names += [
            os.path.join(os.path.abspath(dir_name), fname) for fname in file_names
        ]
    if args.od:
        # if all output files are written to the same directory we need them
        # to have a unique name, so use an index.
        file_names = [
            os.path.join(os.path.abspath(args.od), str(i))
            for i in range(len(input_file_names))
        ]
    else:
        file_names = input_file_names
    output_file_names = [
        file_name + "." + args.output_file_extension for file_name in file_names
    ]
```

(continues on next page)

(continued from previous page)

```
    res = convert_images(input_file_names, output_file_names, args.w)
    input_file_names = list(itertools.compress(input_file_names, res))
    output_file_names = list(itertools.compress(output_file_names, res))

    # save csv file mapping input and output file names.
    # using csv module and not pandas so as not to create more dependencies
    # for the examples. pandas based code is more elegant/shorter.
    dir_name = args.od if args.od else os.getcwd()
    with open(os.path.join(dir_name, "file_names.csv"), mode="w") as fp:
        fp_writer = csv.writer(
            fp, delimiter=",", quotechar='"', quoting=csv.QUOTE_MINIMAL
        )
        fp_writer.writerow(["input file name", "output file name"])
        for data in zip(input_file_names, output_file_names):
            fp_writer.writerow(data)


if __name__ == "__main__":
    sys.exit(main())
```

```
library(SimpleITK)
library(argparser)

convert_image <- function(input_file_name, output_file_name, new_width = NA)
{
  image_file_reader = ImageFileReader()
  # only read DICOM images
  image_file_reader$SetImageIO('GDCMImageIO')
  image_file_reader$SetFileName(input_file_name)
  try(image_file_reader$ReadImageInformation(), silent = TRUE)
  image_size <- image_file_reader$GetSize()
  #if this isn't a DICOM image return FALSE
  if(length(image_size) == 0) {
    return(FALSE)
  }
  if(length(image_size) == 3 & image_size[3] == 1) {
    image_size[3] <- 0
    image_file_reader$SetExtractSize(image_size)
  }
  image <- NULL
  try(image <- image_file_reader$Execute(), silent = TRUE)
  if(is.null(image))
  {
    return(FALSE)
  }
  if(!is.na(new_width))
  {
    original_size <- image$GetSize()
    original_spacing <- image$GetSpacing()
    new_spacing <- c((original_size[1]-1)*original_spacing[1]/(new_width-1),
                     (original_size[1]-1)*original_spacing[1]/(new_width-1))
    new_size <- c(new_width, as.integer((original_size[2]-1)*original_spacing[2]/new_
```

(continues on next page)

(continued from previous page)

```
↪spacing[2]))
    image <- Resample(image, new_size, Transform(), 'sitkLinear',
                      image$GetOrigin(), new_spacing, image$GetDirection(),0,
                      image$GetPixelID())
  }
  # If a single channel image, rescale to [0,255]. Also modify the intensity values based
  # on the photomertic interpretation. If MONOCHROME2 (minimum should be displayed as␣
↪black) we
  # don't need to do anything, if image has MONOCRHOME1 (minimum should be displayed as␣
↪white) we flip
  # the intensities. This is a constraint imposed by ITK which always assumes␣
↪MONOCHROME2.
  if(image$GetNumberOfComponentsPerPixel() == 1)
  {
    image <- RescaleIntensity(image, 0, 255)
    if(trimws(image_file_reader$GetMetaData('0028|0004')) == 'MONOCHROME1')
    {
      image <-  InvertIntensity(image, maximum=255)
    }
    image <- Cast(image, 'sitkUInt8')
  }
  WriteImage(image, output_file_name)
  return(TRUE)
 }

parser<-arg_parser('Convert and resize DICOM files to common image types.')
parser<-add_argument(parser, 'root_of_data_directory', help='Path to the topmost␣
↪directory containing data.')
parser<-add_argument(parser, 'output_file_extension', help='Image file extension, this␣
↪determines output file type (e.g. png) .')
parser<-add_argument(parser, '--w', help='Width of converted images.')
parser<-add_argument(parser, '--od', help='Output directory.')
argv<-parse_args(parser)

input_file_names <- list.files(argv$root_of_data_directory, recursive=TRUE, full.
↪names=TRUE)
if(!is.na(argv$od)) {
  # if all output files are written to the same directory we need them to have a unique␣
↪name, so use an index.
  file_names <- lapply(seq_len(length(input_file_names)), function(new_file_name)␣
↪return(file.path(argv$od, toString(new_file_name))))
} else {
  file_names <- input_file_names
}
#append the file extension to the output file names
output_file_names <- lapply(file_names, function(file_name) return(paste0(file_name,'.',
↪argv$output_file_extension)))
res <- mapply(convert_image, input_file_names, output_file_names, new_width=as.
↪integer(argv$w), SIMPLIFY=FALSE)
df <- data.frame(input_file_name = unlist(input_file_names[unlist(res)]),
                 output_file_name = unlist(output_file_names[unlist(res)]))
dir_name <- if(!is.na(argv$od)) argv$od else getwd()
```

(continues on next page)

(continued from previous page)

```
write.csv(df,file.path(dir_name, 'file_names.csv'), row.names=FALSE)
```

14.12 Fast Marching Segmentation

14.12.1 Overview

The fast marching method is a simple form of level-set evolution where only a positive speed term is used to govern the differential equation. The resulting level-set contour only grows over time. Practically, this algorithm can be used as an advanced region growing segmentation which is controlled by a speed image.

A good propagation speed image for segmentation is close to zero near object boundaries and relatively high in between. In this example, an input feature image is smoothed with an anisotropic diffusion method, then the gradient magnitude is used to produce an edge image. A Gaussian with a parameterized *sigma* is used during the gradient computation to enable the level-set to slow down near edges. The Sigmoid filter performs a linear transform on the gradient magnitude so that boundaries are near zero and homogeneous regions are close to one. The values for *alpha* and *beta* are provided in the testing code. The heuristics used to estimate good values are dependent on the minimum value along a boundary and the mean value of the gradient in the object's region.

Lastly the fast marching filter is configured with an initial trial point and starting value. Each trial point consists of a tuple for an image index including an optional unsigned starting seed value at the trial point. The trial points are the starting location of the level-set. The output of the fast marching filter is a *time-crossing map* that indicate the time of arrival of the propagated level-set front. We threshold the result to the region the level-set front propagated through to form the segmented object. A graphical interface can be constructed to show the contour propagation over time, enabling a user to select a the desired segmentation.

14.12.2 Example Run

Running the Python code with the following inputs:

```
main('BrainProtonDensitySlice.png', 'fastMarchingOutput.mha', 81 114 1.0 -0.5 3.0 100
110)
```

produces the input & output images below.

Input/Output Images

14.12.3 Code

C#

C++

Python

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

//INPUTS:  {BrainProtonDensitySlice.png}
//    OUTPUTS: {FastMarchingImageFilterOutput5.png}
//    ARGUMENTS:    81 114 1.0  -0.5  3.0   100 100
```

(continues on next page)

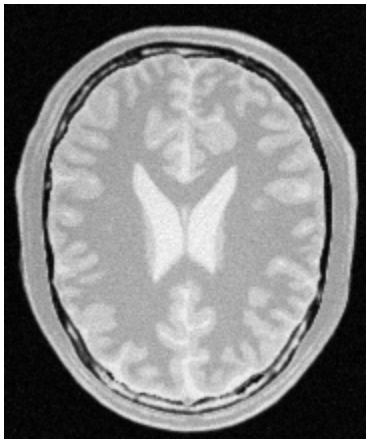


Fig. 2: **Input Image**



Fig. 3: **Speed Image**



Fig. 4: **Time Crossing Map**

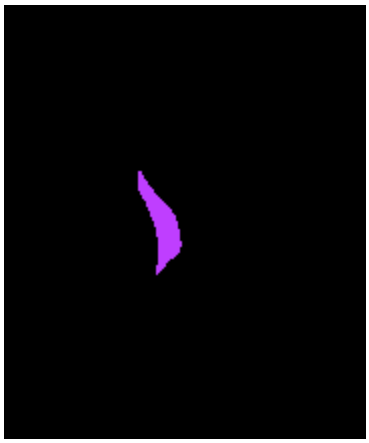


Fig. 5: **Final Segmentation**

(continued from previous page)

```
using System;

using SitkImage = itk.simple.Image;
using System.Globalization;

namespace itk.simple.examples
{
    public class FastMarchingSegmentation
    {
        static void Main(string[] args)
        {
            if (args.Length < 9)
            {
                Console.WriteLine("Missing Parameters ");
                Console.WriteLine("Usage: " + System.AppDomain.CurrentDomain.
↪FriendlyName +
                                  " inputImage outputImage seedX seedY " +
                                  " Sigma SigmoidAlpha SigmoidBeta TimeThreshold");
                return;
            }

            string inputFilename = args[0];
            string outputFilename = args[1];

            uint[] seedPosition = { Convert.ToUInt32(args[2]), Convert.ToUInt32(args[3]),
↪0 };

            double sigma = double.Parse(args[4], CultureInfo.InvariantCulture);
            double alpha = double.Parse(args[5], CultureInfo.InvariantCulture); ;
            double beta = double.Parse(args[6], CultureInfo.InvariantCulture);
            double timeThreshold = double.Parse(args[7], CultureInfo.InvariantCulture);
            double stoppingTime = double.Parse(args[8], CultureInfo.InvariantCulture);

            // Read input image
```

(continues on next page)

(continued from previous page)

```
            SitkImage inputImage = SimpleITK.ReadImage(inputFilename, PixelIDValueEnum.
↪sitkFloat32);

            //  The input image will be processed with a few iterations of
            //  feature-preserving diffusion.  We create a filter and set the
            //  appropriate parameters.

            CurvatureAnisotropicDiffusionImageFilter smoothing = new␣
↪CurvatureAnisotropicDiffusionImageFilter();
            smoothing.SetTimeStep(0.125);
            smoothing.SetNumberOfIterations(5);
            smoothing.SetConductanceParameter(9.0);
            SitkImage smoothingOutput = smoothing.Execute(inputImage);

            SitkImage gradientMagnitudeOutput = SimpleITK.
↪GradientMagnitudeRecursiveGaussian(smoothingOutput, sigma);

            SitkImage sigmoidOutput = SimpleITK.Sigmoid(gradientMagnitudeOutput, alpha,␣
↪beta, 1.0, 0.0);


            FastMarchingImageFilter fastMarching = new FastMarchingImageFilter();

            //VectorUIntList trialPoints; Add trialPoints into list if using multiple␣
↪seeds. Here we only use one seedpoint

            VectorUInt32 trialPoint = new VectorUInt32(3);
            trialPoint.Add(seedPosition[0]);
            trialPoint.Add(seedPosition[1]);
            trialPoint.Add(seedPosition[2]);

            fastMarching.AddTrialPoint(trialPoint);

            //  Since the front representing the contour will propagate continuously
            //  over time, it is desirable to stop the process once a certain time has
            //  been reached. This allows us to save computation time under the
            //  assumption that the region of interest has already been computed. The
            //  value for stopping the process is defined with the method
            //  SetStoppingValue(). In principle, the stopping value should be a
            //  little bit higher than the threshold value.

            fastMarching.SetStoppingValue(stoppingTime);

            SitkImage fastmarchingOutput = fastMarching.Execute(sigmoidOutput);

            BinaryThresholdImageFilter thresholder = new BinaryThresholdImageFilter();
            thresholder.SetLowerThreshold(0.0);
            thresholder.SetUpperThreshold(timeThreshold);
            thresholder.SetOutsideValue(0);
            thresholder.SetInsideValue(255);
            SitkImage result = thresholder.Execute(fastmarchingOutput);
```

(continues on next page)

(continued from previous page)

```
            SimpleITK.WriteImage(result, outputFilename);

        }
    }
}
```

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


#include <SimpleITK.h>
#include <iostream>
#include <string>
#include <cstdlib>

namespace sitk = itk::simple;


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

  if ( argc < 10 )
    {
    std::cerr << "Missing Parameters " << std::endl;
    std::cerr << "Usage: " << argv[0];
    std::cerr << " inputImage  outputImage seedX seedY";
    std::cerr << " Sigma SigmoidAlpha SigmoidBeta TimeThreshold StoppingTime" <<␣
↪std::endl;

    return EXIT_FAILURE;
    }

  const std::string inputFilename(argv[1]);
  const std::string outputFilename(argv[2]);

  unsigned int seedPosition[2];
  seedPosition[0] = atoi( argv[3] );
  seedPosition[1] = atoi( argv[4] );

  const double sigma = atof( argv[5] );
  const double alpha =  atof( argv[6] );
  const double beta  =  atof( argv[7] );
  const double timeThreshold = atof( argv[8] );
  const double stoppingTime = atof( argv[9] );

  sitk::Image inputImage = sitk::ReadImage( inputFilename, sitk::sitkFloat32 );


  sitk::CurvatureAnisotropicDiffusionImageFilter smoothing;
  smoothing.SetTimeStep( 0.125 );
  smoothing.SetNumberOfIterations(  5 );
  smoothing.SetConductanceParameter( 9.0 );
  sitk::Image smoothingOutput = smoothing.Execute( inputImage );
```

(continues on next page)

(continued from previous page)

```
  sitk::GradientMagnitudeRecursiveGaussianImageFilter gradientMagnitude;
  gradientMagnitude.SetSigma(  sigma  );
  sitk::Image gradientMagnitudeOutput = gradientMagnitude.Execute( smoothingOutput );

  sitk::SigmoidImageFilter sigmoid;
  sigmoid.SetOutputMinimum(  0.0  );
  sigmoid.SetOutputMaximum(  1.0  );
  sigmoid.SetAlpha( alpha );
  sigmoid.SetBeta(  beta  );
  sitk::Image sigmoidOutput = sigmoid.Execute( gradientMagnitudeOutput );


  sitk::FastMarchingImageFilter fastMarching;

  std::vector< unsigned int > trialPoint(3);


  trialPoint[0] = seedPosition[0];
  trialPoint[1] = seedPosition[1];

  trialPoint[2] = 0u; // Seed Value

  fastMarching.AddTrialPoint( trialPoint );

  fastMarching.SetStoppingValue(stoppingTime);

  sitk::Image fastmarchingOutput = fastMarching.Execute( sigmoidOutput );


  sitk::BinaryThresholdImageFilter thresholder;
  thresholder.SetLowerThreshold(           0.0  );
  thresholder.SetUpperThreshold( timeThreshold  );
  thresholder.SetOutsideValue(  0  );
  thresholder.SetInsideValue(  255 );

  sitk::Image result = thresholder.Execute(fastmarchingOutput);

  sitk::WriteImage(result, outputFilename);

  return 0;
}
```

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


import SimpleITK as sitk
import sys


def main(args):
    if len(args) < 10:
```

(continues on next page)

(continued from previous page)

```
        print(
            "Usage:",
            "FastMarchingSegmentation",
            "<inputImage> <outputImage> <seedX> <seedY> <Sigma>",
            "<SigmoidAlpha> <SigmoidBeta> <TimeThreshold>",
            "<StoppingTime>",
        )
        sys.exit(1)

    inputFilename = args[1]
    outputFilename = args[2]

    seedPosition = (int(args[3]), int(args[4]))

    sigma = float(args[5])
    alpha = float(args[6])
    beta = float(args[7])
    timeThreshold = float(args[8])
    stoppingTime = float(args[9])

    inputImage = sitk.ReadImage(inputFilename, sitk.sitkFloat32)

    smoothing = sitk.CurvatureAnisotropicDiffusionImageFilter()
    smoothing.SetTimeStep(0.125)
    smoothing.SetNumberOfIterations(5)
    smoothing.SetConductanceParameter(9.0)
    smoothingOutput = smoothing.Execute(inputImage)

    gradientMagnitude = sitk.GradientMagnitudeRecursiveGaussianImageFilter()
    gradientMagnitude.SetSigma(sigma)
    gradientMagnitudeOutput = gradientMagnitude.Execute(smoothingOutput)

    sigmoid = sitk.SigmoidImageFilter()
    sigmoid.SetOutputMinimum(0.0)
    sigmoid.SetOutputMaximum(1.0)
    sigmoid.SetAlpha(alpha)
    sigmoid.SetBeta(beta)
    sigmoidOutput = sigmoid.Execute(gradientMagnitudeOutput)

    fastMarching = sitk.FastMarchingImageFilter()

    seedValue = 0
    trialPoint = (seedPosition[0], seedPosition[1], seedValue)

    fastMarching.AddTrialPoint(trialPoint)

    fastMarching.SetStoppingValue(stoppingTime)

    fastMarchingOutput = fastMarching.Execute(sigmoidOutput)

    thresholder = sitk.BinaryThresholdImageFilter()
    thresholder.SetLowerThreshold(0.0)
```

(continues on next page)

(continued from previous page)

```
    thresholder.SetUpperThreshold(timeThreshold)
    thresholder.SetOutsideValue(0)
    thresholder.SetInsideValue(255)

    result = thresholder.Execute(fastMarchingOutput)

    sitk.WriteImage(result, outputFilename)

    image_dict = {
        "InputImage": inputImage,
        "SpeedImage": sigmoidOutput,
        "TimeCrossingMap": fastMarchingOutput,
        "Segmentation": result,
    }
    return image_dict


if __name__ == "__main__":
    return_dict = main(sys.argv)
```

14.13 Filter Progress Reporting

14.13.1 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.

14.13.2 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.

14.13.3 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.

14.13.4 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#

Java

Python

Ruby

```
  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() );
    }
  }
```

```
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);
  }
}
```

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

    def Execute(self):
        print(
            f"{self.processObject.GetName()}"
            + f" Progress: {self.processObject.GetProgress():1.2f}"
        )
```

(continues on next page)

(continued from previous page)

```
```

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

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

14.13.5 Command Functions and Lambdas for Wrapped Languages

Not all 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:

C++

Python

R

```
  writer.AddCommand(sitk::sitkStartEvent, [] {std::cout << "Writting..." << std::flush;}
↪);
  writer.AddCommand(sitk::sitkEndEvent, [] {std::cout << "done" << std::endl;});
```

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

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

14.13.6 Code

CSharp

C++

Java

Python

R

Ruby

```
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) {
      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);
      }
    }
  }
```

(continues on next page)

(continued from previous page)

```
}
```

```
// 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)
    {}

  void Execute( ) override
    {
      // 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;
  }
```

(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 blurring 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] ) );

  //! [cpp lambda command]
  writer.AddCommand(sitk::sitkStartEvent, [] {std::cout << "Writting..." << std::flush;}
↪);
  writer.AddCommand(sitk::sitkEndEvent, [] {std::cout << "done" << std::endl;});
  //! [cpp lambda command]

  writer.Execute ( outputImage );

  return 0;
}
```

```
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() {
```

(continues on next page)

(continued from previous page)

```
    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);

    }

}
```

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

import os
import sys

import SimpleITK as sitk

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

(continues on next page)

(continued from previous page)

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

    def Execute(self):
        print(
            f"{self.processObject.GetName()}"
            + f" Progress: {self.processObject.GetProgress():1.2f}"
        )


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

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

(continues on next page)

(continued from previous page)

```
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 )
```

```
require 'simpleitk'

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


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

(continues on next page)

(continued from previous page)

```
  # 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
```

14.14 Image Grid Manipulation

14.14.1 Overview

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

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

- Image Slicing Operator - Uses slicing (`img[i:j, k:l]`) to extract a subregion of an image

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

14.14.2 Comparisons

Composition Filters

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

Extraction Filters

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

14.14.3 Code

Python

R

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


import SimpleITK as sitk
import sys

if len(sys.argv) < 3:
    print("Usage: " + sys.argv[0] + " <input-1> <input-2>")
    sys.exit(1)

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

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

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

# Recompose image (should be same as joined_image)
compose = sitk.ComposeImageFilter()
```

(continues on next page)

(continued from previous page)

```
composed_image = compose.Execute(channel1_image, channel2_image, channel3_image)

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

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

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

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

library(SimpleITK)

args <- commandArgs( TRUE )

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

# Two vector images of same pixel type and dimension expected
reader <- ImageFileReader()
reader$SetFileName(args[[1]])
image_1 <- reader$Execute()
reader$SetFileName(args[[2]])
image_2 <- reader$Execute()

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

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

(continues on next page)

(continued from previous page)

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

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

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

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

14.15 Image Registration Method 1

14.15.1 Overview

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

14.15.2 Example Run

Running the Python code with the following inputs:

```
main('BrainProtonDensitySliceBorder20.png', 'BrainProtonDensitySliceShifted13x17y.png',
'displaceMeth1.hdf5')
```

produces the text and images below.

Output Text

```
 0 = 4499.45303 : (2.928695951245586, 2.7244705953923805)
 1 = 3860.83626 : (6.135143776902402, 5.115849348610004)
 2 = 3508.02147 : (8.822660051952475, 8.078492808653918)
 3 = 3117.30771 : (10.968558473732326, 11.454158663474674)
 4 = 2125.42619 : (13.105290365964755, 14.835634202454191)
 5 =  911.30826 : (12.75173580401588, 18.819978461140323)
 6 =  741.41676 : (13.139053510563274, 16.857840597942413)
 7 =   16.89176 : (12.356787624301035, 17.480785285045815)
```

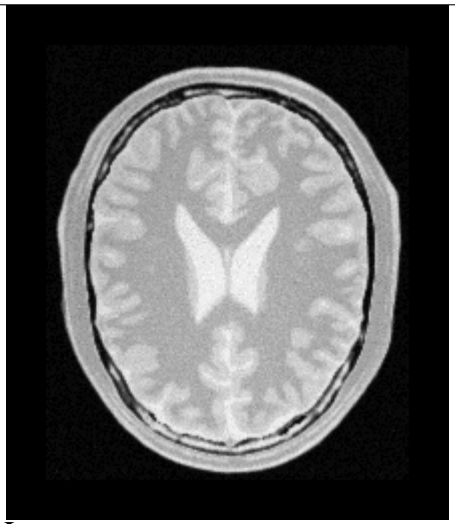
(continues on next page)

(continued from previous page)

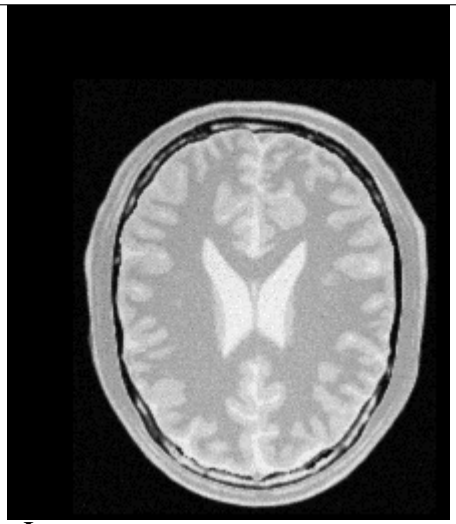
```
  8 =  233.71446 : (12.79212443526829, 17.234854683011704)
  9 =   39.80270 : (13.167510875734614, 16.904574468172815)
 10 =   16.57314 : (12.938831371165355, 17.005597654570586)
 11 =    1.68763 : (13.063495692092735, 16.996443033457986)
 12 =    1.79437 : (13.001061362657559, 16.999307384689935)
 13 =    0.00076 : (12.945418587211314, 17.0277701944711)
 14 =    1.74802 : (12.974454390534774, 17.01621663980765)
 15 =    0.43025 : (13.002439510423766, 17.002309966416835)
 16 =    0.00532 : (12.989877586882951, 16.99301810428082)
-------
itk::simple::TranslationTransform
 TranslationTransform (0x556710e9c780)
   RTTI typeinfo:   itk::TranslationTransform<double, 2u>
   Reference Count: 2
   Modified Time: 2526
   Debug: Off
   Object Name:
   Observers:
     none
   Offset: [12.9899, 16.993]

Optimizer stop condition: RegularStepGradientDescentOptimizerv4: Step too small after 17␣
↪iterations. Current step (0.0078125) is less than minimum step (0.01).
 Iteration: 18
 Metric value: 0.07213459794461137
```

Input Images



Fixed Image



Moving Image

Output Image

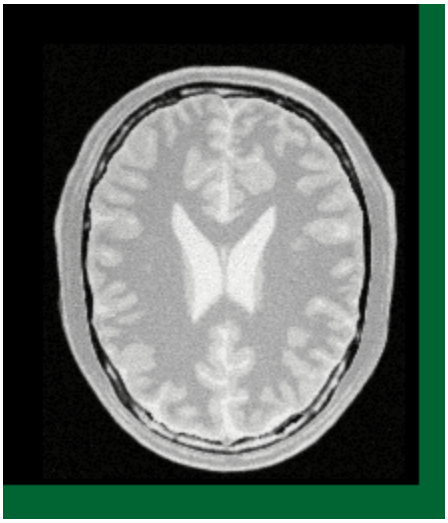


Fig. 6: **Composition Image**

14.15.3 Code

C#

C++

Java

Lua

Python

R

```
using System;
using itk.simple;


namespace itk.simple.examples {


  class IterationUpdate : Command {

    private ImageRegistrationMethod m_Method;

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

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

(continues on next page)

(continued from previous page)

```
                        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 );

      // 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]);
```

(continues on next page)

(continued from previous page)

```
    }

  }

}
```

```
// 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)
    {}

  void Execute( ) override
    {
      // 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;

};


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

  if ( argc < 4 )
    {
```

(continues on next page)

(continued from previous page)

```
    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;
}
```

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


class IterationUpdate  extends Command {

  private ImageRegistrationMethod m_Method;

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

(continues on next page)

(continued from previous page)

```
  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.
```

(continues on next page)

(continued from previous page)

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

  outTx.writeTransform(argv[2]);

  }

}
```

```
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

import SimpleITK as sitk
import sys
import os


def command_iteration(method):
    print(
        f"{method.GetOptimizerIteration():3} "
        + f"= {method.GetMetricValue():10.5f} "
        + f": {method.GetOptimizerPosition()}"
    )


def main(args):
    if len(args) < 3:
        print(
            "Usage:",
            "ImageRegistrationMethod1",
            "<fixedImageFilter> <movingImageFile>",
            "<outputTransformFile>",
        )
        sys.exit(1)

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

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

    R = sitk.ImageRegistrationMethod()
    R.SetMetricAsMeanSquares()
    R.SetOptimizerAsRegularStepGradientDescent(4.0, 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(f"Optimizer stop condition: {R.GetOptimizerStopConditionDescription()}")
    print(f" Iteration: {R.GetOptimizerIteration()}")
    print(f" Metric value: {R.GetMetricValue()}")

    sitk.WriteTransform(outTx, args[3])

    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 // 2.0 + simg2 // 2.0)

    return_images = {"fixed": fixed, "moving": moving, "composition": cimg}
    return return_images


if __name__ == "__main__":
```

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


library(SimpleITK)


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

args <- commandArgs( TRUE )

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

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

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


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

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

outTx = Reg$Execute(fixed, moving)

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

14.16 Image Registration Method 2

14.16.1 Overview

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

14.16.2 Example Run

Running the Python code with the following inputs:

```
main('BrainProtonDensitySliceBorder20.png', 'BrainProtonDensitySliceShifted13x17y.png',
'displaceMeth2.hdf5')
```

produces the text and images below.

Output Text

```
  0  = -0.49019  : (0.0028307206468947536, 0.0023491904331766162)
  1  = -0.49030  : (0.015968780135700314, 0.01326142597839447)
  2  = -0.49068  : (0.08103357252207889, 0.06751733183069583)
  3  = -0.49262  : (0.4031918173866258, 0.3356312129926553)
  4  = -0.50145  : (1.9891643379821866, 1.6551894820472324)
  5  = -0.53055  : (10.002028651827025, 8.700643712668963)
  6  = -0.71953  : (14.301161173235505, 16.02941144181758)
  7  = -1.14334  : (12.934080126605934, 16.847083218629955)
  8  = -1.28582  : (13.021228858202821, 16.980930809778222)
  9  = -1.29232  : (13.010721376329036, 17.001152119792206)
 10  = -1.29329  : (13.011578940166988, 17.000992823315222)
 11  = -1.29327  : (13.01231557664238, 17.000844562218692)
 12  = -1.29324  : (13.012426459622057, 17.000820248130232)
 13  = -1.29323  : (13.012465052549832, 17.000811591622085)
-------
itk::simple::TranslationTransform
 TranslationTransform (0x5633675e2c40)
   RTTI typeinfo:   itk::TranslationTransform<double, 2u>
   Reference Count: 2
   Modified Time: 25514
   Debug: Off
   Object Name:
   Observers:
     none
   Offset: [13.0116, 17.001]

Optimizer stop condition: GradientDescentLineSearchOptimizerv4Template: Convergence␣
↪checker passed at iteration 14.
 Iteration: 14
 Metric value: -1.2932878917340445
```

Input Images

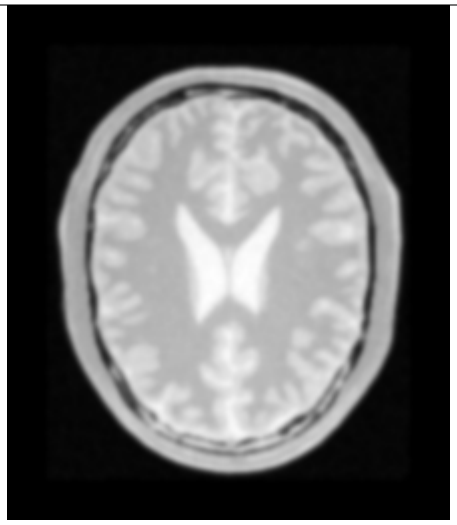


Fig. 7: **Fixed Image**

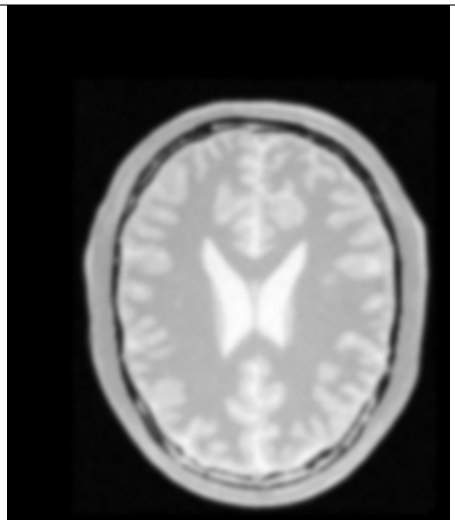


Fig. 8: **Moving Image**

Output Image

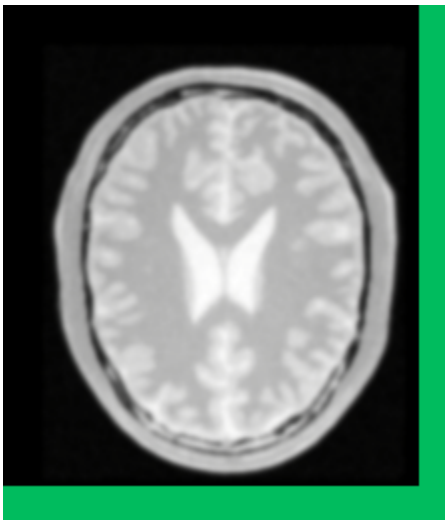


Fig. 9: **Composition Image**

14.16.3 Code

C#

C++

Python

R

```
using System;
using itk.simple;

namespace itk.simple.examples
{


    class IterationUpdate : Command
    {

        private ImageRegistrationMethod m_Method;

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

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

    class ImageRegistrationMethod2
    {

        static void Main(string[] args)
        {

            if (args.Length < 3)
            {
                Console.WriteLine("Usage: %s <fixedImageFilter> <movingImageFile>
↪<outputTransformFile>\n", "ImageRegistrationMethod2");
                return;
            }

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

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

(continues on next page)

(continued from previous page)

```
            fixedImage = SimpleITK.Normalize(fixedImage);
            SimpleITK.DiscreteGaussian(fixedImage, 2.0);

            reader.SetFileName(args[1]);
            Image movingImage = reader.Execute();
            movingImage=SimpleITK.Normalize(movingImage);
            movingImage = SimpleITK.DiscreteGaussian(movingImage, 2.0);

            ImageRegistrationMethod R = new ImageRegistrationMethod();
            R.SetMetricAsJointHistogramMutualInformation();

            double learningRate = 1;
            uint  numberOfIterations = 200;
            double convergenceMinimumValue = 1e-4;
            uint convergenceWindowSize = 5;

            R.SetOptimizerAsGradientDescentLineSearch(learningRate,
                                                      numberOfIterations,
                                                      convergenceMinimumValue,
                                                      convergenceWindowSize);

            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);

            outTx.WriteTransform(args[2]);

        }

    }

}
```

```
// 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
```

(continues on next page)

(continued from previous page)

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

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

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

      std::cout.copyfmt(state);
    }

private:
  const sitk::ImageRegistrationMethod &m_Method;

};


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

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


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

  sitk::Image moving = sitk::ReadImage( argv[2], sitk::sitkFloat32 );
  moving = sitk::Normalize( moving );
  moving = sitk::DiscreteGaussian( moving, 2.0);


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

  const double learningRate = 1;
```

(continues on next page)

(continued from previous page)

```
  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;
}
```

```
#!/usr/bin/env python
import SimpleITK as sitk
import sys
import os


def command_iteration(method):
    print(
        f"{method.GetOptimizerIteration():3} "
        + f" = {method.GetMetricValue():7.5f} "
        + f" : {method.GetOptimizerPosition()}"
    )


def main(args):
    if len(args) < 3:
        print(
            "Usage:",
            "ImageRegistrationMethod2",
            "<fixedImageFilter> <movingImageFile>  <outputTransformFile>",
        )
        sys.exit(1)
```

(continues on next page)

(continued from previous page)

```
    fixed = sitk.ReadImage(args[1], sitk.sitkFloat32)
    fixed = sitk.Normalize(fixed)
    fixed = sitk.DiscreteGaussian(fixed, 2.0)

    moving = sitk.ReadImage(args[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(f"Optimizer stop condition: {R.GetOptimizerStopConditionDescription()}")
    print(f" Iteration: {R.GetOptimizerIteration()}")
    print(f" Metric value: {R.GetMetricValue()}")

    sitk.WriteTransform(outTx, args[3])

    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 // 2.0 + simg2 // 2.0)

    return {"fixed": fixed, "moving": moving, "composition": cimg}


if __name__ == "__main__":
```

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

library(SimpleITK)

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

args <- commandArgs( TRUE )

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

pixelType <- 'sitkFloat32'

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


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

R <- ImageRegistrationMethod()

R$SetMetricAsJointHistogramMutualInformation()

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

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

R$SetInterpolator('sitkLinear')

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

outTx <- R$Execute(fixed, moving)

cat("-------\n")
outTx
cat("Optimizer stop condition:", R$GetOptimizerStopConditionDescription(), '\n')
```

(continues on next page)

(continued from previous page)

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

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

14.17 Image Registration Method 3

14.17.1 Overview

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

14.17.2 Example Run

Running the Python code with the following inputs:

```
main('BrainProtonDensitySliceBorder20.png', 'BrainProtonDensitySliceShifted13x17y.png',
'displaceMeth3.hdf5')
```

produces the text and images below.

Output Text

```
Estimated Scales:  (29682.41582933708, 29682.168476696, 1.0000000000010232, 0.
↪999999999998181)
  0 = -0.99671 : (1.0001225188582936, -4.8155319621057874e-05, 14.24580144271548, 18.
↪231972935092553)
  1 = -0.90442 : (1.0001395698120492, -5.723494632868289e-05, 13.436891660253242, 17.
↪644040146872914)
  2 = -0.97294 : (1.0000524825397714, -3.538901200507155e-05, 12.784457666827517, 16.
↪886194597858392)
  3 = -0.99664 : (1.000078552246351, -3.493898453141627e-05, 13.248020397809663, 17.
↪07357013992443)
  4 = -0.99617 : (1.0000080585357958, -3.574957852612509e-05, 13.003964467172034, 17.
↪01937883295938)
  5 = -0.99998 : (0.9999690242080485, -8.938378761798914e-06, 12.941455002408405, 16.
↪77731982244243)
  6 = -0.99743 : (0.9999729946034386, -1.4195394490341945e-05, 12.980498953463153, 16.
↪89606564275891)
  7 = -0.99946 : (0.9999822072360259, -1.9442192886012526e-05, 13.009300985166485, 17.
↪01770216218198)
  8 = -0.99998 : (1.0000081565952237, -1.536015920575109e-05, 12.97497046153363, 16.
↪965475115002747)
  9 = -0.99990 : (1.000006961859454, -1.5839846033647187e-05, 12.996178272885958, 16.
↪98842704816722)
 10 = -0.99999 : (0.999997104263667, -1.5781032972153657e-05, 13.008157869541614, 17.
↪01728968260679)
 11 = -0.99998 : (0.9999981089260577, -1.480066274538104e-05, 13.000149494110696, 17.
↪003873030789478)
```

(continues on next page)

(continued from previous page)

```
 12 = -1.00000 : (1.0000035245702283, -1.1964202117479094e-05, 12.999557108219037, 16.
↪988259265497895)
 13 = -0.99999 : (1.0000015232779387, -1.2092316764425097e-05, 12.999892054939323, 16.
↪996064581822232)
 14 = -1.00000 : (0.9999991283248564, -1.1669994922994523e-05, 13.00013101903772, 17.
↪003873425942555)
 15 = -1.00000 : (0.9999999572679382, -1.1067338273415268e-05, 12.99999566483145, 16.
↪99996952183884)
 16 = -1.00000 : (1.0000015262190347, 2.7428357741776532e-05, 13.001481019896447, 17.
↪00123717053477)
 17 = -1.00000 : (1.0000010761172178, 2.70338949452921e-05, 13.00064749562188, 17.
↪000728337318556)
 18 = -1.00000 : (1.0000004942557568, 2.6224543927636213e-05, 12.999879933384591, 17.
↪000124581130446)
 19 = -1.00000 : (0.9999997860745542, 2.334309634129523e-05, 13.000293057931712, 16.
↪99986431610512)
 20 = -1.00000 : (0.9999997971406921, 2.288036936123565e-05, 13.000070149678, 16.
↪99996389693141)
 21 = -1.00000 : (0.9999999481117745, 2.1399925381340785e-05, 12.999868757877147, 17.
↪000101896108283)
 22 = -1.00000 : (0.999999997953508, 2.0861222466294866e-05, 12.999980029544846, 17.
↪000051701641695)
 23 = -1.00000 : (1.0000000486509963, 1.633859430051924e-05, 13.000031070402398, 16.
↪999940906626833)
-------
itk::simple::Similarity2DTransform
 Similarity2DTransform (0x560e721f5580)
   RTTI typeinfo:   itk::Similarity2DTransform<double>
   Reference Count: 3
   Modified Time: 2649
   Debug: Off
   Object Name:
   Observers:
     none
   Matrix:
     1 -1.63386e-05
     1.63386e-05 1
   Offset: [13.0022, 16.9981]
   Center: [111.204, 131.591]
   Translation: [13, 16.9999]
   Inverse:
     1 1.63386e-05
     -1.63386e-05 1
   Singular: 0
   Angle       = 1.63386e-05
   Scale =1

Optimizer stop condition: RegularStepGradientDescentOptimizerv4: Step too small after 24␣
↪iterations. Current step (6.10352e-05) is less than minimum step (0.0001).
 Iteration: 25
 Metric value: -0.9999999568784146
```

Input Images

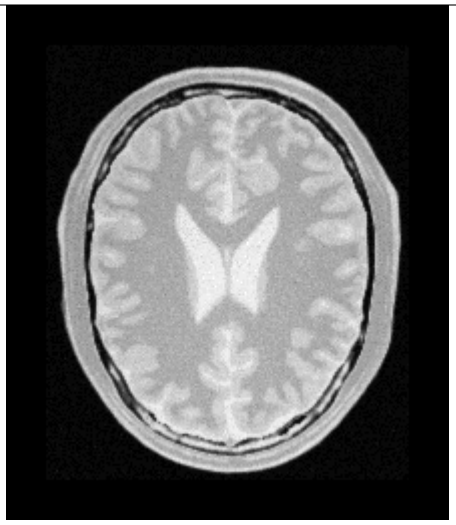


Fig. 10: **Fixed Image**

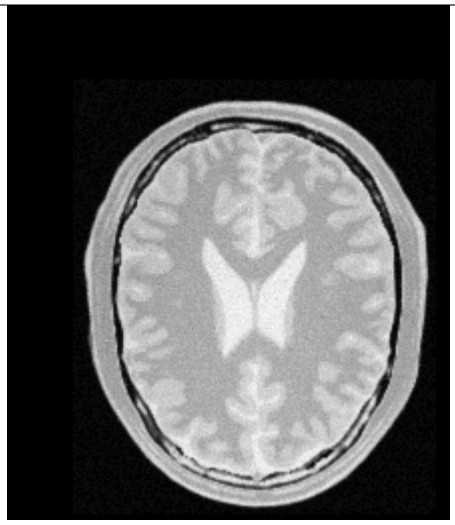


Fig. 11: **Moving Image**

Output Image

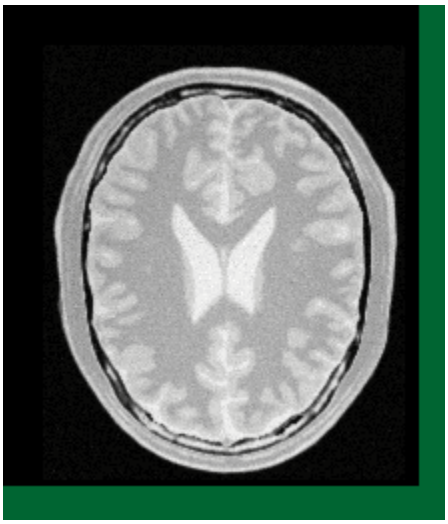


Fig. 12: **Composition Image**

14.17.3 Code

Python

R

```python
#!/usr/bin/env python
import SimpleITK as sitk
import sys
import os


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


def main(args):
    if len(args) < 3:
        print(
            "Usage:",
            "ImageRegistrationMethod3"
            "<fixedImageFilter> <movingImageFile>  <outputTransformFile>",
        )
        sys.exit(1)

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

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

    R = sitk.ImageRegistrationMethod()

    R.SetMetricAsCorrelation()

    R.SetOptimizerAsRegularStepGradientDescent(
        learningRate=2.0,
        minStep=1e-4,
        numberOfIterations=500,
        gradientMagnitudeTolerance=1e-8,
    )
    R.SetOptimizerScalesFromIndexShift()

    tx = sitk.CenteredTransformInitializer(fixed, moving, sitk.Similarity2DTransform())
    R.SetInitialTransform(tx)

    R.SetInterpolator(sitk.sitkLinear)

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

(continues on next page)

(continued from previous page)

```
    outTx = R.Execute(fixed, moving)

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

    sitk.WriteTransform(outTx, args[3])

    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 // 2.0 + simg2 // 2.0)

    return {"fixed": fixed, "moving": moving, "composition": cimg}


if __name__ == "__main__":
    return_dict = main(sys.argv)
    if "SITK_NOSHOW" not in os.environ:
        sitk.Show(return_dict["composition"], "ImageRegistration3 Composition")
```

```
# Run with:
#
# Rscript --vanilla ImageRegistrationMethod3.R fixedImageFilter movingImageFile␣
↪outputTransformFile
#

library(SimpleITK)

commandIteration <- function(method)
{
    if (method$GetOptimizerIteration()==0) {
        cat("Estimated Scales:", method$GetOptimizerScales())
    }
    msg <- paste(method$GetOptimizerIteration(), "=",
                 method$GetMetricValue(), ":",
                 method$GetOptimizerPosition(), "\n" )
    cat(msg)
}

args <- commandArgs( TRUE )

if (length(args) != 3) {
```

(continues on next page)

(continued from previous page)

```
    stop("3 arguments expected - fixedImageFilter, movingImageFile, outputTransformFile")
}

pixelType <- 'sitkFloat32'

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

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

R <- ImageRegistrationMethod()

R$SetMetricAsCorrelation()

R$SetOptimizerAsRegularStepGradientDescent(learningRate=2.0,
                                           minStep=1e-4,
                                           numberOfIterations=500,
                                           relaxationFactor=0.5,
                                           gradientMagnitudeTolerance=1e-8 )
R$SetOptimizerScalesFromIndexShift()

tx <- CenteredTransformInitializer(fixed, moving, Similarity2DTransform())
R$SetInitialTransform(tx)

R$SetInterpolator('sitkLinear')

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

outTx <- R$Execute(fixed, moving)

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


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

14.18 Image Registration Method 4

14.18.1 Overview

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

14.18.2 Example Run

Running the Python code with the following inputs:

```
main('BrainProtonDensitySliceBorder20.png', 'BrainProtonDensitySliceShifted13x17y.png',
'displaceMeth4.hdf5')
```

produces the text and images below.

Output Text

```
  0 =   -0.36408 : (0.6480599915239152, 0.7615892904880052)
  1 =   -0.37188 : (1.2696822410664477, 1.5449064541411972)
  2 =   -0.38623 : (1.728451100909272, 2.4334620985656277)
  3 =   -0.39306 : (2.2894677218546713, 3.2612666352909385)
  4 =   -0.40611 : (3.210519625827277, 3.6507064768434407)
  5 =   -0.41745 : (3.8851656450643826, 4.388847891971685)
  6 =   -0.42624 : (4.422217251010506, 5.232397281486425)
  7 =   -0.43564 : (5.111666807560848, 5.956730981498054)
  8 =   -0.44299 : (5.988104593535712, 6.4382460978915885)
  9 =   -0.46063 : (6.766019050408342, 7.066616289568884)
 10 =   -0.47604 : (7.3493222851367825, 7.878870767152934)
 11 =   -0.49378 : (8.227184517818442, 8.357784016532372)
 12 =   -0.51152 : (8.69714163208972, 9.2404732657848)
 13 =   -0.53136 : (9.22318788779119, 10.090929223999254)
 14 =   -0.54806 : (9.785328926046697, 10.917970609365803)
 15 =   -0.58083 : (10.278699181858894, 11.787790010551937)
 16 =   -0.61501 : (10.839990336791079, 12.61540842467459)
 17 =   -0.66102 : (11.364090804240067, 13.467064869504165)
 18 =   -0.72635 : (11.682530308102086, 14.415008054711016)
 19 =   -0.79455 : (12.308925390390241, 15.194513796117207)
 20 =   -0.88661 : (12.691552740702406, 16.118416556581256)
 21 =   -1.05536 : (13.043952889227963, 17.054265975728995)
 22 =   -1.38873 : (12.72406148875578, 16.6699880920251)
 23 =   -1.22627 : (12.88766255574607, 16.859024307813606)
 24 =   -1.34539 : (13.053290860531256, 17.046286863211062)
 25 =   -1.38869 : (12.957760802894011, 16.96567043754095)
 26 =   -1.39322 : (13.009970870875847, 17.000026787773944)
 27 =   -1.39977 : (12.979573420547538, 16.992777131676032)
 28 =   -1.39859 : (12.994952034923315, 16.995540977706685)
 29 =   -1.39980 : (13.010022578944076, 16.999666426520854)
 30 =   -1.39978 : (13.002363800393388, 16.998124263312402)
 31 =   -1.39994 : (12.995096966756398, 16.995255764551324)
 32 =   -1.39981 : (12.99881054940698, 16.996467413411267)
 33 =   -1.39991 : (13.002419786329824, 16.997961467592773)
 34 =   -1.39994 : (13.000541240645328, 16.997426899190334)
-------
itk::simple::TranslationTransform
 TranslationTransform (0x55f4cdf50310)
   RTTI typeinfo:   itk::TranslationTransform<double, 2u>
   Reference Count: 2
   Modified Time: 13969
   Debug: Off
```

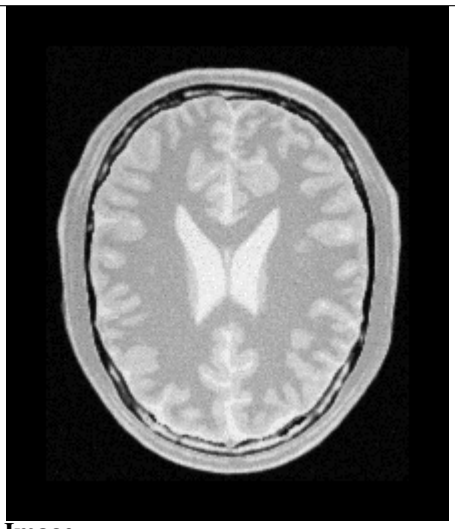
(continues on next page)

(continued from previous page)

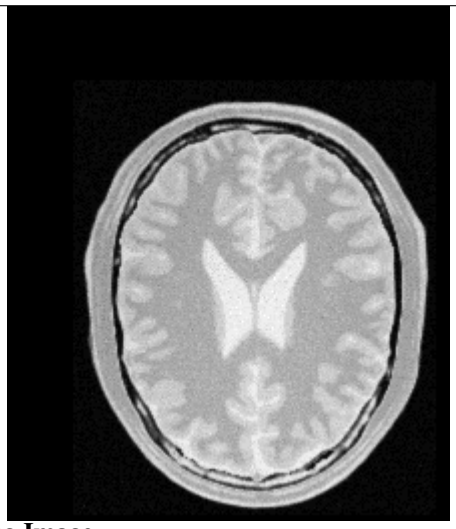
```
   Object Name:
   Observers:
     none
   Offset: [13.0005, 16.9974]

Optimizer stop condition: RegularStepGradientDescentOptimizerv4: Step too small after 35␣
↪iterations. Current step (0.000976562) is less than minimum step (0.001).
 Iteration: 36
 Metric value: -1.3999347066058423
```

Input Images



Fixed Image



Moving Image

Output Image

14.18.3 Code

Python

R

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

import SimpleITK as sitk
import sys
import os


def command_iteration(method):
    print(
        f"{method.GetOptimizerIteration():3} "
        + f"= {method.GetMetricValue():10.5f} "
```

(continues on next page)

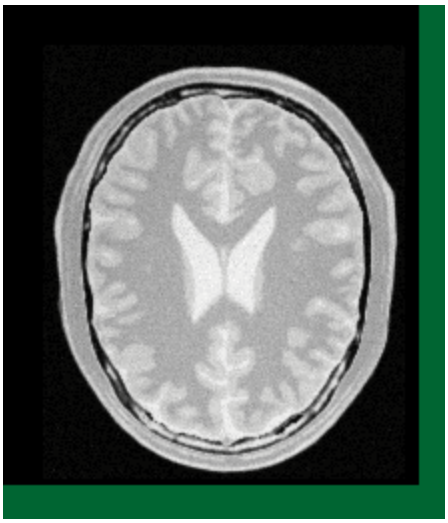


Fig. 13: **Composition Image**

(continued from previous page)

```
        + f": {method.GetOptimizerPosition()}"
    )


def main(args):
    if len(args) < 3:
        print(
            "Usage:",
            "ImageRegistrationMethod4",
            "<fixedImageFilter> <movingImageFile>",
            "<outputTransformFile> <numberOfBins> <samplingPercentage>",
        )
        sys.exit(1)

    fixed = sitk.ReadImage(args[1], sitk.sitkFloat32)
    moving = sitk.ReadImage(args[2], sitk.sitkFloat32)

    numberOfBins = 24
    samplingPercentage = 0.10

    if len(args) > 4:
        numberOfBins = int(args[4])
    if len(args) > 5:
        samplingPercentage = float(args[5])

    R = sitk.ImageRegistrationMethod()
    R.SetMetricAsMattesMutualInformation(numberOfBins)
    R.SetMetricSamplingPercentage(samplingPercentage, sitk.sitkWallClock)
    R.SetMetricSamplingStrategy(R.RANDOM)
    R.SetOptimizerAsRegularStepGradientDescent(1.0, 0.001, 200)
    R.SetInitialTransform(sitk.TranslationTransform(fixed.GetDimension()))
    R.SetInterpolator(sitk.sitkLinear)
```

(continues on next page)

(continued from previous page)

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

    outTx = R.Execute(fixed, moving)

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

    sitk.WriteTransform(outTx, args[3])

    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 // 2.0 + simg2 // 2.0)

    return {"fixed": fixed, "moving": moving, "composition": cimg}


if __name__ == "__main__":
```

```
# Run with:
#
# Rscript --vanilla ImageRegistrationMethod4.R fixedImageFilter movingImageFile␣
↪outputTransformFile numberOfBins samplingPercentage
#

library(SimpleITK)

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

args <- commandArgs( TRUE )

if (length(args) < 3) {
    stop("3, 4, or 5 arguments expected - fixedImageFilter, movingImageFile,␣
↪outputTransformFile [numberOfBins] [samplingPercentage]")
}
```

(continues on next page)

(continued from previous page)

```
fixed <- ReadImage(args[[1]], 'sitkFloat32')
moving <- ReadImage(args[[2]], 'sitkFloat32')

numberOfBins <- 24
samplingPercentage <- 0.10

if (length(args) > 4) {
    numberOfBins <- strtoi(args[[4]])
}
if (length(args) > 5) {
    samplingPercentage <- as.numeric(args[[5]])
}

R <- ImageRegistrationMethod()
R$SetMetricAsMattesMutualInformation(numberOfBins)
R$SetMetricSamplingPercentage(samplingPercentage)
R$SetMetricSamplingStrategy('RANDOM')
R$SetOptimizerAsRegularStepGradientDescent(1.0,.001,200)
R$SetInitialTransform(TranslationTransform(fixed$GetDimension()))
R$SetInterpolator('sitkLinear')

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

outTx <- R$Execute(fixed, moving)

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

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

14.19 Image Registration Optimizer Weights

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

14.19.1 Overview

The image registration framework allows us to disable subsets of the transformation parameters so that we can use a lower dimensional transformation. In some cases, if we have some better prior knowledge we can weigh the parameters accordingly (this is much harder to utilize to get a desired effect than disabling parameters).

In this example, we work with a 3D rigid transformation using the Euler parameterization, but we only allow rotation around the z axis. As there is no transformation that represents this subspace of rigid transformations, we use the *SetOptimizerWeights* method to disable rotations around the x and y axes. The order of the weights in the weights parameter matches the order of parameters returned from the transformation *GetParameters* method. For the Euler3DTransform the order is *[angleX, angleY, angleZ, tx, ty, tz]*, so our weights vector is *[0,0,1,1,1,1]*.

14.19.2 Code

Python

R

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

import SimpleITK as sitk
import sys

if len(sys.argv) < 4:
    print(
        "Usage:",
        sys.argv[0],
        "<fixedImageFile> <movingImageFile>",
        "<outputTransformFile>",
    )
    sys.exit(1)

fixed_image = sitk.ReadImage(sys.argv[1], sitk.sitkFloat32)
moving_image = sitk.ReadImage(sys.argv[2], sitk.sitkFloat32)

# initialization
transform = sitk.CenteredTransformInitializer(
    fixed_image,
    moving_image,
    sitk.Euler3DTransform(),
    sitk.CenteredTransformInitializerFilter.GEOMETRY,
)
# registration
registration_method = sitk.ImageRegistrationMethod()
registration_method.SetMetricAsCorrelation()
registration_method.SetMetricSamplingStrategy(registration_method.NONE)
registration_method.SetInterpolator(sitk.sitkLinear)
registration_method.SetOptimizerAsGradientDescent(
    learningRate=1.0,
    numberOfIterations=300,
    convergenceMinimumValue=1e-6,
    convergenceWindowSize=10,
)
registration_method.SetOptimizerScalesFromPhysicalShift()
registration_method.SetInitialTransform(transform, inPlace=True)
registration_method.SetOptimizerWeights([0, 0, 1, 1, 1, 1])
registration_method.Execute(fixed_image, moving_image)

print("-------")
print(f"Final transform parameters: {transform.GetParameters()}")
print(
    "Optimizer stop condition: "
    + f"{registration_method.GetOptimizerStopConditionDescription()}"
)
print(f"Iteration: {registration_method.GetOptimizerIteration()}")
print(f"Metric value: {registration_method.GetMetricValue()}")
```

(continues on next page)

(continued from previous page)

```
sitk.WriteTransform(transform, sys.argv[3])
```

```
# Run with:
#
# Rscript --vanilla ImageRegistrationOptimizerWeights.R <fixedImageFile>
↪<movingImageFile> <outputTransformFile>
#


library(SimpleITK)

args <- commandArgs( TRUE )

if (length(args) != 3) {
    stop("Expected arguments: <fixedImageFile> <movingImageFile> <outputTransformFile>")
}

fixed_image = ReadImage(args[[1]], "sitkFloat32")
moving_image = ReadImage(args[[2]], "sitkFloat32")

# initialization
transform <- CenteredTransformInitializer(fixed_image,
                                          moving_image,
                                          Euler3DTransform(),
                                          "GEOMETRY")
# registration
registration_method <- ImageRegistrationMethod()
registration_method$SetMetricAsCorrelation()
registration_method$SetMetricSamplingStrategy("NONE")
registration_method$SetInterpolator("sitkLinear")
registration_method$SetOptimizerAsGradientDescent(learningRate=1.0,
                                                  numberOfIterations=300,
                                                  convergenceMinimumValue=1e-6,
                                                  convergenceWindowSize=10)
registration_method$SetOptimizerScalesFromPhysicalShift()
registration_method$SetInitialTransform(transform, inPlace=TRUE)
registration_method$SetOptimizerWeights(c(0,0,1,1,1,1))
output_sink_tx<-registration_method$Execute(fixed_image, moving_image)

cat("-------\n")
cat("Final transform parameters: ", transform$GetParameters(), "\n")
cat("Optimizer stop condition: ", registration_method
↪$GetOptimizerStopConditionDescription(), "\n")
cat("Iteration: ", registration_method$GetOptimizerIteration(), "\n")
cat("Metric value:", registration_method$GetMetricValue(), "\n")

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

14.20 Image Registration Method BSpline 1

14.20.1 Overview

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

14.20.2 Example Run

Running the Python code with the following inputs:

```
main('BrainProtonDensitySliceBorder20.png', 'BrainProtonDensitySliceBSplined10.png',
'ImageRegistrationMethodBSpline1.hdf5')
```

produces the text and images below.

Output Text

```
Initial Parameters:
(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
↪0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,␣
↪0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
↪0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,␣
↪0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
↪0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,␣
↪0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
↪0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,␣
↪0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
↪0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
  0 =   -0.83872
  0 =   -0.85295
  0 =   -0.89317
  0 =   -0.91369
  0 =   -0.91369
  1 =   -0.95662
  1 =   -0.95662
  2 =   -0.96795
  2 =   -0.96795
  3 =   -0.97045
  3 =   -0.97045
  4 =   -0.97769
  4 =   -0.97769
  5 =   -0.98248
  5 =   -0.98248
  6 =   -0.98273
  6 =   -0.98273
  7 =   -0.98739
```

(continues on next page)

(continued from previous page)

```
  7 =   -0.98739
  8 =   -0.98956
  8 =   -0.98956
  9 =   -0.99150
  9 =   -0.99150
 10 =   -0.99169
 10 =   -0.99169
 11 =   -0.99289
 11 =   -0.99289
 12 =   -0.99322
 12 =   -0.99322
 13 =   -0.99362
 13 =   -0.99362
 14 =   -0.99409
 14 =   -0.99409
 15 =   -0.99400
 15 =   -0.99420
 15 =   -0.99420
 16 =   -0.99434
 16 =   -0.99434
 17 =   -0.99444
 17 =   -0.99444
 18 =   -0.99456
 18 =   -0.99456
 19 =   -0.99462
 19 =   -0.99462
 20 =   -0.99467
 20 =   -0.99467
 21 =   -0.99469
 21 =   -0.99469
 22 =   -0.99471
 22 =   -0.99471
 23 =   -0.99473
 23 =   -0.99473
 24 =   -0.99476
 24 =   -0.99476
 25 =   -0.99478
 25 =   -0.99478
 26 =   -0.99481
 26 =   -0.99481
 27 =   -0.99484
 27 =   -0.99484
 28 =   -0.99486
 28 =   -0.99486
 29 =   -0.99487
 29 =   -0.99487
 30 =   -0.99488
 30 =   -0.99488
-------
itk::simple::BSplineTransform
 BSplineTransform (0x5588fddf32a0)
   RTTI typeinfo:   itk::BSplineTransform<double, 2u, 3u>
```

(continues on next page)

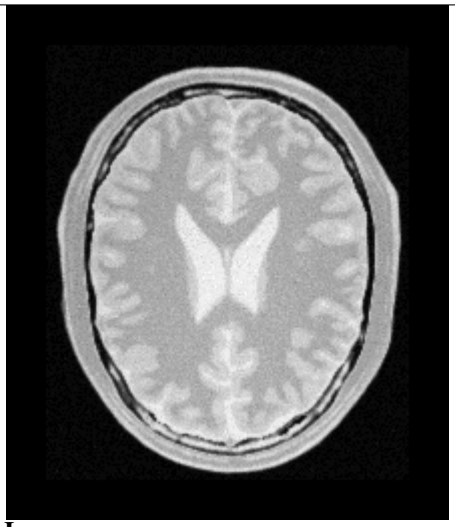
(continued from previous page)

```
   Reference Count: 3
   Modified Time: 2650
   Debug: Off
   Object Name:
   Observers:
     none
   CoefficientImage: [ 0x5588fdd61440, 0x5588fdd54130 ]
   TransformDomainOrigin: [-0.625, -0.625]
   TransformDomainPhysicalDimensions: [221.25, 257.25]
   TransformDomainDirection: 1 0
0 1

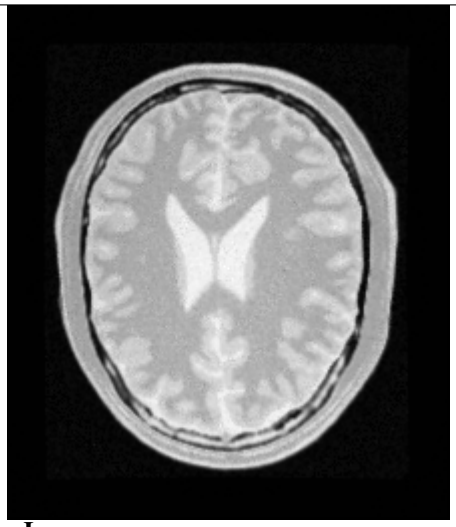
   TransformDomainMeshSize: [8, 8]
   GridSize: [11, 11]
   GridOrigin: [-28.2812, -32.7812]
   GridSpacing: [27.6562, 32.1562]
   GridDirection: 1 0
0 1


Optimizer stop condition: LBFGSBOptimizerv4: Gradient tolerance reached. Gradient␣
↪tolerance is 1e-05
 Iteration: 31
 Metric value: -0.9948792040893044
```

Input Images



Fixed Image



Moving Image

Output Image

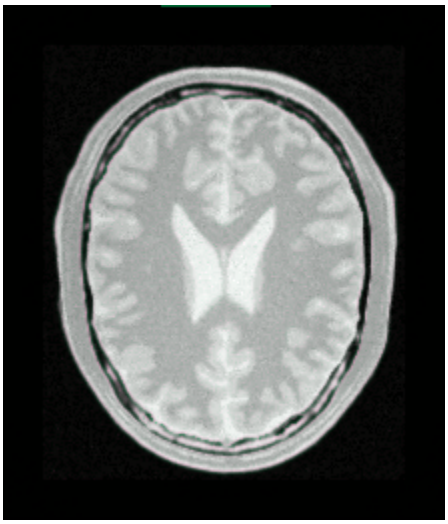


Fig. 14: **Composition Image**

14.20.3 Code

C++

Python

R

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

namespace sitk = itk::simple;

// use sitk's output operator for std::vector etc..
using sitk::operator<<;




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

(continues on next page)

(continued from previous page)

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

      if (m_Method.GetOptimizerIteration() == 0)
        {
        std::cout << m_Method.ToString() << std::endl;
        }

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

private:
  const sitk::ImageRegistrationMethod &m_Method;

};




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

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

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

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

  std::vector<unsigned int> transformDomainMeshSize(fixed.GetDimension(),8);

  sitk::BSplineTransform tx = sitk::BSplineTransformInitializer(fixed,␣
↪transformDomainMeshSize);

  std::cout << "Initial Parameters:" << tx.GetParameters() << std::endl;


  sitk::ImageRegistrationMethod R;
  R.SetMetricAsCorrelation();
```

(continues on next page)

(continued from previous page)

```
  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 );

  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;
}
```

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

import SimpleITK as sitk
import sys
import os


def command_iteration(method):
    print(f"{method.GetOptimizerIteration():3} " + f"= {method.GetMetricValue():10.5f}")


def main(args):
    if len(args) < 4:
        print(
            "Usage:",
            sys.args[0],
            "<fixedImageFilter> <movingImageFile>",
            "<outputTransformFile>",
        )
        sys.exit(1)

    fixed = sitk.ReadImage(args[1], sitk.sitkFloat32)
```

(continues on next page)

(continued from previous page)

```
    moving = sitk.ReadImage(args[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=1e7,
    )
    R.SetInitialTransform(tx, True)
    R.SetInterpolator(sitk.sitkLinear)

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

    outTx = R.Execute(fixed, moving)

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

    sitk.WriteTransform(outTx, args[3])

    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 // 2.0 + simg2 // 2.0)

    return_images = {"fixed": fixed, "moving": moving, "composition": cimg}
    return return_images


if __name__ == "__main__":
    return_dict = main(sys.argv)
    if "SITK_NOSHOW" not in os.environ:
```

(continues on next page)

(continued from previous page)

```
        sitk.Show(
            return_dict["composition"], "ImageRegistrationMethodBSpline1 Composition"
        )
```

```
# Run with:
#
# Rscript --vanilla ImageRegistrationMethodBSpline1.R fixedImageFilter movingImageFile␣
↪outputTransformFile
#

library(SimpleITK)

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

args <- commandArgs( TRUE )

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

fixed <- ReadImage(args[[1]], 'sitkFloat32')
moving <- ReadImage(args[[2]], 'sitkFloat32')

transformDomainMeshSize <- rep(8, moving$GetDimension())
tx <- BSplineTransformInitializer(fixed, transformDomainMeshSize)

cat("Initial Parameters:\n", tx$GetParameters())

R <- ImageRegistrationMethod()
R$SetMetricAsCorrelation()

R$SetOptimizerAsLBFGSB(gradientConvergenceTolerance=1e-5,
                       numberOfIterations=100,
                       maximumNumberOfCorrections=5,
                       maximumNumberOfFunctionEvaluations=1000,
                       costFunctionConvergenceFactor=1e+7)
R$SetInitialTransform(tx, TRUE)
R$SetInterpolator('sitkLinear')

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

outTx <- R$Execute(fixed, moving)

cat("-------\n")
outTx
cat("Optimizer stop condition:", R$GetOptimizerStopConditionDescription(), '\n')
cat(" Iteration:", R$GetOptimizerIteration(), '\n')
```

(continues on next page)

(continued from previous page)

```
cat(" Metric value:", R$GetMetricValue(), '\n')

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

14.21 Image Registration Method BSpline 2

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

14.21.1 Overview

14.21.2 Code

Python

R

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

import SimpleITK as sitk
import sys
import os


def command_iteration(method):
    print(f"{method.GetOptimizerIteration():3} " + f"= {method.GetMetricValue():10.5f}")
    print("\t#: ", len(method.GetOptimizerPosition()))


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


if len(sys.argv) < 4:
    print(
        "Usage:",
        sys.argv[0],
        "<fixedImageFilter> <movingImageFile>",
        "<outputTransformFile>",
    )
    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)
```

(continues on next page)

(continued from previous page)

```
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(f"Optimizer stop condition: {R.GetOptimizerStopConditionDescription()}")
print(f" Iteration: {R.GetOptimizerIteration()}")
print(f" Metric value: {R.GetMetricValue()}")

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

if "SITK_NOSHOW" not 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 // 2.0 + simg2 // 2.0)
    sitk.Show(cimg, "ImageRegistration1 Composition")
```

```
# Run with:
#
# Rscript --vanilla ImageRegistrationMethodBSpline2.R fixedImageFilter movingImageFile␣
↪outputTransformFile
#

library(SimpleITK)

commandIteration <- function(method)
{
    msg <- paste(method$GetOptimizerIteration(), "=",
```

(continues on next page)

(continued from previous page)

```
                  method$GetMetricValue(), "\n\t#:",
                  method$GetOptimizerPosition(), '\n')
    cat(msg)
}

commandMultiIteration <- function(method)
{
    msg <- paste("--------- Resolution Changing ---------\n")
    cat(msg)
}

args <- commandArgs( TRUE )

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

fixed <- ReadImage(args[[1]], 'sitkFloat32')
moving <- ReadImage(args[[2]], 'sitkFloat32')

transformDomainMeshSize <- rep(10, moving$GetDimension())
tx <- BSplineTransformInitializer(fixed, transformDomainMeshSize)

cat("Initial Parameters:\n", tx$GetParameters())

R <- ImageRegistrationMethod()
R$SetMetricAsMattesMutualInformation(50)
R$SetOptimizerAsGradientDescentLineSearch(5.0, 100,
                                          convergenceMinimumValue=1e-4,
                                          convergenceWindowSize=5)
R$SetOptimizerScalesFromPhysicalShift()
R$SetInitialTransform(tx)
R$SetInterpolator('sitkLinear')

R$SetShrinkFactorsPerLevel(c(6,2,1))
R$SetSmoothingSigmasPerLevel(c(6,2,1))

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

outTx <- R$Execute(fixed, moving)

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

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

14.22 Image Registration Method BSpline 3

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

14.22.1 Overview

This example performs registration of multi-modality images with a multi-resolution BSpline approach.

A BSplineTransform usually has a large number of parameters which increases the complexity and duration of optimizing the deformation. The multi-resolution BSpline approach initially performs the registration at a lower resolution with fewer parameters at the first level and then adapts or resamples the BSpline control points to a higher resolution at subsequent levels. This transformation adaption is done concurrently with the ImageRegistrationMethod's multi-level feature. This enables the setting of the shrink factor, smoothing sigmas, sampling percentage and BSpline resolution to vary per level to efficiently solve a diverse set of registration problems.

The initial transform is the low resolution BSpline. The scaling factor of each BSpline used per level is specified with the `ImageRegistration::SetInitialBSpline` method's third argument as an integer array. The first value is usually 1 and it is reasonable to double the resolution at each registration level. For this example the last resolution is 5, so that the result is comparable to the transformation from the *previous* example.

It can be important to monitor and observe the transform at each level or iteration. When the "inplace" option is enabled, the transform passed as the initial transform will be up to date during the registration process which enables it to be used in *event commands*.

See also: *Image Registration Method BSpline 1*, *Image Registration Method BSpline 2*.

14.22.2 Code

C++

Python

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

namespace sitk = itk::simple;

// use sitk's output operator for std::vector etc..
using sitk::operator<<;


class IterationUpdate
  : public sitk::Command
{
public:
  IterationUpdate( const sitk::ImageRegistrationMethod &m, const sitk::BSplineTransform &
↪tx)
    : m_Method(m),
```

(continues on next page)

(continued from previous page)

```
      m_BSplineTransform(tx)
    {}

  // Override method from parent which is called at for the requested event
  void Execute( ) override
    {
      if (m_Method.GetOptimizerIteration() == 0)
        {
        // The BSpline is resized before the first optimizer
        // iteration is completed per level. Print the transform object
        // to show the adapted BSpline transform.
        std::cout << m_BSplineTransform.ToString() << std::endl;
        }

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

private:
  const sitk::ImageRegistrationMethod &m_Method;
  const sitk::BSplineTransform &m_BSplineTransform;

};


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

  // Override method from parent which is called at for the requested event
  void Execute( ) override
    {
      // The sitkMultiResolutionIterationEvent occurs before the
      // resolution of the transform. This event is used here to print
      // the status of the optimizer from the previous registration level.
      if (m_Method.GetCurrentLevel() > 0)
        {
        std::cout << "Optimizer stop condition: " << m_Method.
↪GetOptimizerStopConditionDescription() << std::endl;
        std::cout << " Iteration: " << m_Method.GetOptimizerIteration() << std::endl;
        std::cout << " Metric value: " << m_Method.GetMetricValue() << std::endl;
        }
```

(continues on next page)

(continued from previous page)

```
      std::cout << "--------- Resolution Changing ---------" << std::endl;
    }

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




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

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

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

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

  std::vector<unsigned int> transformDomainMeshSize(fixed.GetDimension(), 2);

  sitk::BSplineTransform tx = sitk::BSplineTransformInitializer(fixed,␣
↪transformDomainMeshSize);

  std::cout << "Initial Number of Parameters:" << tx.GetNumberOfParameters() <<␣
↪std::endl;


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


  const double learningRate = 5.0;
  const unsigned int numberOfIterations = 100u;
  const double convergenceMinimumValue = 1e-4;
  const unsigned int convergenceWindowSize = 5;
  R.SetOptimizerAsGradientDescentLineSearch( learningRate,
                                             numberOfIterations,
                                             convergenceMinimumValue,
                                             convergenceWindowSize);

  R.SetInterpolator(sitk::sitkLinear);

  std::vector<unsigned int> scaleFactors = { 1, 2, 5 };
  const bool inPlace = true;
  R.SetInitialTransformAsBSpline(tx,
                                 inPlace,
```

(continues on next page)

(continued from previous page)

```
                                scaleFactors);

  std::vector<unsigned int> shrinkFactors = { 4, 2, 1 };
  R.SetShrinkFactorsPerLevel( shrinkFactors );

  std::vector<double> smoothingSigmas = { 4.0, 2.0, 1.0 };
  R.SetSmoothingSigmasPerLevel( smoothingSigmas );

  IterationUpdate cmd1(R, tx);
  R.AddCommand( sitk::sitkIterationEvent, cmd1);

  MultiResolutionIterationUpdate cmd2(R);
  R.AddCommand( sitk::sitkMultiResolutionIterationEvent, cmd2);

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

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

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

  return 0;
}
```

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

import SimpleITK as sitk
import sys
import os


def command_iteration(method, bspline_transform):
    if method.GetOptimizerIteration() == 0:
        # The BSpline is resized before the first optimizer
        # iteration is completed per level. Print the transform object
        # to show the adapted BSpline transform.
        print(bspline_transform)

    print(f"{method.GetOptimizerIteration():3} " + f"= {method.GetMetricValue():10.5f}")


def command_multi_iteration(method):
    # The sitkMultiResolutionIterationEvent occurs before the
    # resolution of the transform. This event is used here to print
    # the status of the optimizer from the previous registration level.
    if R.GetCurrentLevel() > 0:
        print(
            "Optimizer stop condition: " + f"{R.GetOptimizerStopConditionDescription()}"
```

(continues on next page)

(continued from previous page)

```
        )
        print(f" Iteration: {R.GetOptimizerIteration()}")
        print(f" Metric value: {R.GetMetricValue()}")

    print("--------- Resolution Changing ---------")


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

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

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

transformDomainMeshSize = [2] * fixed.GetDimension()
tx = sitk.BSplineTransformInitializer(fixed, transformDomainMeshSize)

print(f"Initial Number of Parameters: {tx.GetNumberOfParameters()}")

R = sitk.ImageRegistrationMethod()
R.SetMetricAsJointHistogramMutualInformation()

R.SetOptimizerAsGradientDescentLineSearch(
    5.0, 100, convergenceMinimumValue=1e-4, convergenceWindowSize=5
)

R.SetInterpolator(sitk.sitkLinear)

R.SetInitialTransformAsBSpline(tx, inPlace=True, scaleFactors=[1, 2, 5])
R.SetShrinkFactorsPerLevel([4, 2, 1])
R.SetSmoothingSigmasPerLevel([4, 2, 1])

R.AddCommand(sitk.sitkIterationEvent, lambda: command_iteration(R, tx))
R.AddCommand(sitk.sitkMultiResolutionIterationEvent, lambda: command_multi_iteration(R))

outTx = R.Execute(fixed, moving)

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

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

(continues on next page)

(continued from previous page)

```
if "SITK_NOSHOW" not 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 // 2.0 + simg2 // 2.0)
    sitk.Show(cimg, "Image Registration Composition")
```

14.23 Image Registration Method Displacement 1

14.23.1 Overview

14.23.2 Code

C++

Python

R

```
// 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)
    {}

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

      // stash the stream state
```

(continues on next page)

(continued from previous page)

```
      std::ios  state(NULL);
      state.copyfmt(std::cout);
      std::cout << std::fixed << std::setfill(' ') << std::setprecision( 5 );
      if ( m_Method.GetOptimizerIteration() == 0 )
        {
        std::cout << "\tLevel: " << std::setw(3) << m_Method.GetCurrentLevel() <<␣
↪std::endl;
        std::cout << "\tScales: " << m_Method.GetOptimizerScales() << std::endl;
        }
      std::cout << '#' << m_Method.GetOptimizerIteration() << std::endl;
      std::cout << "\tMetric Value: " <<  m_Method.GetMetricValue() << std::endl;
      std::cout << "\tLearning Rate: " << m_Method.GetOptimizerLearningRate() <<␣
↪std::endl;
      if (m_Method.GetOptimizerConvergenceValue() != std::numeric_limits<double>::max())
        {
        std::cout << "\tConvergence Value: " << std::scientific << m_Method.
↪GetOptimizerConvergenceValue() << std::endl;
        }
      std::cout.copyfmt(state);
    }

private:
  const sitk::ImageRegistrationMethod &m_Method;

};

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

  void Execute( ) override
    {
      // 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;
```

(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::Transform initialTx = sitk::CenteredTransformInitializer(fixed, moving,␣
↪sitk::AffineTransform(fixed.GetDimension()));

  sitk::ImageRegistrationMethod R;

  {
  std::vector<unsigned int> shrinkFactors = { 3, 2, 1 };

  std::vector<double> smoothingSigmas = { 2.0, 1.0, 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);
```

(continues on next page)

(continued from previous page)

```
  R.SetInterpolator(sitk::sitkLinear);

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

  MultiResolutionIterationUpdate cmd2(R);
  R.AddCommand( sitk::sitkMultiResolutionIterationEvent, cmd2);

  sitk::Transform outTx1 = R.Execute( fixed, moving );

  std::cout << "-------" << std::endl;
  std::cout << outTx1.ToString() << std::endl;
  std::cout << "Optimizer stop condition: " << R.GetOptimizerStopConditionDescription() <
↪< std::endl;
  std::cout << " Iteration: " << R.GetOptimizerIteration() << std::endl;
  std::cout << " Metric value: " << R.GetMetricValue() << std::endl;


  sitk::Image displacementField = sitk::Image(fixed.GetSize(), sitk::sitkVectorFloat64);
  displacementField.CopyInformation(fixed);
  sitk::DisplacementFieldTransform displacementTx(displacementField);
  const double varianceForUpdateField = 0.0;
  const double varianceForTotalField = 1.5;
  displacementTx.SetSmoothingGaussianOnUpdate(varianceForUpdateField,
                                              varianceForTotalField);



  R.SetMovingInitialTransform(outTx1);
  R.SetInitialTransform(displacementTx, true);

  R.SetMetricAsANTSNeighborhoodCorrelation(4);
  R.MetricUseFixedImageGradientFilterOff();
  R.MetricUseFixedImageGradientFilterOff();

  {
  std::vector<unsigned int> shrinkFactors = { 3, 2, 1 };

  std::vector<double> smoothingSigmas = { 2.0, 1.0, 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;
```

(continues on next page)

(continued from previous page)

```
  sitk::ImageRegistrationMethod::EstimateLearningRateType estimateLearningRate = R.
↪EachIteration;
  R.SetOptimizerAsGradientDescent( learningRate,
                                   numberOfIterations,
                                   convergenceMinimumValue,
                                   convergenceWindowSize,
                                   estimateLearningRate
    );
  }
  R.Execute(fixed, moving);


  std::cout << "-------" << std::endl;
  std::cout << displacementTx.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::CompositeTransform outTx( {outTx1, displacementTx} );
  sitk::WriteTransform(outTx, argv[3]);

  return 0;
}
```

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

import SimpleITK as sitk
import sys
import os


def command_iteration(method):
    if method.GetOptimizerIteration() == 0:
        print(f"\tLevel: {method.GetCurrentLevel()}")
        print(f"\tScales: {method.GetOptimizerScales()}")
    print(f"#{method.GetOptimizerIteration()}")
    print(f"\tMetric Value: {method.GetMetricValue():10.5f}")
    print(f"\tLearningRate: {method.GetOptimizerLearningRate():10.5f}")
    if method.GetOptimizerConvergenceValue() != sys.float_info.max:
        print("\tConvergence Value: " + f"{method.GetOptimizerConvergenceValue():.5e}")


def command_multiresolution_iteration(method):
    print(f"\tStop Condition: {method.GetOptimizerStopConditionDescription()}")
    print("============= Resolution Change =============")


if len(sys.argv) < 4:
    print(
        "Usage:",
```

(continues on next page)

(continued from previous page)

```
        sys.argv[0],
        "<fixedImageFilter> <movingImageFile>",
        "<outputTransformFile>",
    )
    sys.exit(1)

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

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

initialTx = sitk.CenteredTransformInitializer(
    fixed, moving, sitk.AffineTransform(fixed.GetDimension())
)

R = sitk.ImageRegistrationMethod()

R.SetShrinkFactorsPerLevel([3, 2, 1])
R.SetSmoothingSigmasPerLevel([2, 1, 1])

R.SetMetricAsJointHistogramMutualInformation(20)
R.MetricUseFixedImageGradientFilterOff()

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

R.SetInitialTransform(initialTx)

R.SetInterpolator(sitk.sitkLinear)

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

outTx1 = R.Execute(fixed, moving)

print("-------")
print(outTx1)
print(f"Optimizer stop condition: {R.GetOptimizerStopConditionDescription()}")
print(f" Iteration: {R.GetOptimizerIteration()}")
print(f" Metric value: {R.GetMetricValue()}")

displacementField = sitk.Image(fixed.GetSize(), sitk.sitkVectorFloat64)
displacementField.CopyInformation(fixed)
displacementTx = sitk.DisplacementFieldTransform(displacementField)
del displacementField
displacementTx.SetSmoothingGaussianOnUpdate(
```

(continues on next page)

(continued from previous page)

```
    varianceForUpdateField=0.0, varianceForTotalField=1.5
)

R.SetMovingInitialTransform(outTx1)
R.SetInitialTransform(displacementTx, inPlace=True)

R.SetMetricAsANTSNeighborhoodCorrelation(4)
R.MetricUseFixedImageGradientFilterOff()

R.SetShrinkFactorsPerLevel([3, 2, 1])
R.SetSmoothingSigmasPerLevel([2, 1, 1])

R.SetOptimizerScalesFromPhysicalShift()
R.SetOptimizerAsGradientDescent(
    learningRate=1,
    numberOfIterations=300,
    estimateLearningRate=R.EachIteration,
)

R.Execute(fixed, moving)

print("-------")
print(displacementTx)
print(f"Optimizer stop condition: {R.GetOptimizerStopConditionDescription()}")
print(f" Iteration: {R.GetOptimizerIteration()}")
print(f" Metric value: {R.GetMetricValue()}")

compositeTx = sitk.CompositeTransform([outTx1, displacementTx])
sitk.WriteTransform(compositeTx, sys.argv[3])

if "SITK_NOSHOW" not in os.environ:
    sitk.Show(displacementTx.GetDisplacementField(), "Displacement Field")

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

    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 // 2.0 + simg2 // 2.0)
    sitk.Show(cimg, "ImageRegistration1 Composition")
```

```
# Run with:
#
# Rscript --vanilla ImageRegistrationMethodDisplacement1.R fixedImageFilter␣
↪movingImageFile outputTransformFile
#

library(SimpleITK)
```

(continues on next page)

(continued from previous page)

```
commandIteration <- function(method)
{
    if (method$GetOptimizerIteration()==0) {
        cat("\tLevel:", method$GetCurrentLevel(),
            "\n\tScales:", method$GetOptimizerScales(), "\n")
    }
    msg <- paste("#", method$GetOptimizerIteration(),
                 "\n\tMetric Value: ", method$GetMetricValue(),
                 "\n\tLearning Rate: ", method$GetOptimizerLearningRate(),
                 '\n', sep="")
    cat(msg)
}

commandMultiIteration <- function(method)
{
    msg <- paste("\tStop Condition: ", method$GetOptimizerStopConditionDescription(),
                 "\n============== Resolution Change ==============\n", sep="")
    cat(msg)
}

args <- commandArgs( TRUE )

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

fixed <- ReadImage(args[[1]], 'sitkFloat32')
moving <- ReadImage(args[[2]], 'sitkFloat32')

initialTx <- CenteredTransformInitializer(fixed, moving, AffineTransform(fixed
↪$GetDimension()))

R <- ImageRegistrationMethod()

R$SetShrinkFactorsPerLevel(c(3,2,1))
R$SetSmoothingSigmasPerLevel(c(2,1,1))

R$SetMetricAsJointHistogramMutualInformation(20)
R$MetricUseFixedImageGradientFilterOff()

R$SetOptimizerAsGradientDescent(learningRate=1.0,
                                numberOfIterations=100,
                                convergenceMinimumValue=1e-6,
                                convergenceWindowSize=10,
                                estimateLearningRate = 'EachIteration')
R$SetOptimizerScalesFromPhysicalShift()

R$SetInitialTransform(initialTx)

R$SetInterpolator('sitkLinear')
```

(continues on next page)

(continued from previous page)

```
R$AddCommand( 'sitkIterationEvent', function() commandIteration(R) )
R$AddCommand( 'sitkMultiResolutionIterationEvent', function() commandMultiIteration(R) )

outTx1 <- R$Execute(fixed, moving)

cat("-------\n")
outTx1
cat("Optimizer stop condition:", R$GetOptimizerStopConditionDescription(), '\n')
cat("Iteration:", R$GetOptimizerIteration(), '\n')
cat("Metric value:", R$GetMetricValue(), '\n')


displacementField <- Image(fixed$GetSize(), 'sitkVectorFloat64')
displacementField$CopyInformation(fixed)
displacementTx <- DisplacementFieldTransform(displacementField)
rm(displacementField)
displacementTx$SetSmoothingGaussianOnUpdate(varianceForUpdateField=0.0,
                                            varianceForTotalField=1.5)

R$SetMovingInitialTransform(outTx1)
R$SetInitialTransform(displacementTx, inPlace=TRUE)

R$SetMetricAsANTSNeighborhoodCorrelation(4)
R$MetricUseFixedImageGradientFilterOff()

R$SetShrinkFactorsPerLevel(c(3,2,1))
R$SetSmoothingSigmasPerLevel(c(2,1,1))

R$SetOptimizerScalesFromPhysicalShift()
R$SetOptimizerAsGradientDescent(learningRate=1,
                                numberOfIterations=300,
                                convergenceMinimumValue=1e-6,
                                convergenceWindowSize=10,
                                estimateLearningRate = 'EachIteration')

R$Execute(fixed, moving)



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

outTx <- CompositeTransform(outTx1$GetDimension())
outTx$AddTransform(outTx1)
outTx$AddTransform(displacementTx)

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

14.24 Image Registration Method Exhaustive

14.24.1 Overview

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.

14.24.2 Code

Python

R

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

import SimpleITK as sitk
import sys
import os
from math import pi


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


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

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

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

R = sitk.ImageRegistrationMethod()
```

(continues on next page)

(continued from previous page)

```
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(f"Optimizer stop condition: {R.GetOptimizerStopConditionDescription()}")
print(f" Iteration: {R.GetOptimizerIteration()}")
print(f" Metric value: {R.GetMetricValue()}")
```

(continues on next page)

(continued from previous page)

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

if "SITK_NOSHOW" not 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 // 2.0 + simg2 // 2.0)
    sitk.Show(cimg, "ImageRegistrationExhaustive Composition")
```

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

library(SimpleITK)

commandIteration <- function(method)
{
    if (method$GetOptimizerIteration()==0) {
        cat("Scales:", method$GetOptimizerScales(), "\n")
    }
    msg <- paste(method$GetOptimizerIteration(), "=",
                 method$GetMetricValue(), ":",
                 method$GetOptimizerPosition(), "\n" )
    cat(msg)
}

args <- commandArgs( TRUE )

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

fixed <- ReadImage(args[[1]], 'sitkFloat32')
moving <- ReadImage(args[[2]], 'sitkFloat32')

R <- ImageRegistrationMethod()
R$SetMetricAsMattesMutualInformation(numberOfHistogramBins = 50)

sample_per_axis <- 12
if (fixed$GetDimension() == 2) {
    tx <- Euler2DTransform()
    # Set the number of samples (radius) in each dimension, with a
    # default step size of 1.0
```

(continues on next page)

(continued from previous page)

```
    R$SetOptimizerAsExhaustive(c(sample_per_axis%/%2,0,0))
    # Utilize the scale to set the step size for each dimension
    R$SetOptimizerScales(c(2.0*pi/sample_per_axis,1.0,1.0))
} else if (fixed.GetDimension() == 3) {
    tx <- Euler3DTransform()
    R$SetOptimizerAsExhaustive(c(sample_per_axis%/%2,sample_per_axis%/%2,sample_per_axis
↪%/%4,0,0,0))
    R$SetOptimizerScales(c(2.0*pi/sample_per_axis,2.0*pi/sample_per_axis,2.0*pi/sample_
↪per_axis,1.0,1.0,1.0))
}

# Initialize the transform with a translation and the center of
# rotation from the moments of intensity.
tx <- CenteredTransformInitializer(fixed, moving, tx)
R$SetInitialTransform(tx)
R$SetInterpolator('sitkLinear')
R$AddCommand( 'sitkIterationEvent', function() commandIteration(R) )

outTx <- R$Execute(fixed, moving)

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

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

14.25 Landmark Registration

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

14.25.1 Overview

14.25.2 Code

Python

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

import sys
import SimpleITK as sitk

# Black image with a small white square in it.
fixed_image = sitk.Image(100, 100, sitk.sitkUInt8)
fixed_image[11:20, 11:20] = 200

# Black image with a small grey square at a different location.
```

(continues on next page)

(continued from previous page)

```
moving_image = sitk.Image(100, 100, sitk.sitkUInt8)
moving_image[51:60, 51:60] = 69


# Landmarks are 3 corners of the squares.
# 3 (X, Y) pairs are flattened into 1-d lists.
fixed_landmarks = [10, 10, 20, 10, 20, 20]
moving_landmarks = [50, 50, 60, 50, 60, 60]

# Set up the LandmarkBasedTransformInitializerFilter.
landmark_initializer = sitk.LandmarkBasedTransformInitializerFilter()

landmark_initializer.SetFixedLandmarks(fixed_landmarks)
landmark_initializer.SetMovingLandmarks(moving_landmarks)

transform = sitk.Euler2DTransform()

# Compute the transform.
output_transform = landmark_initializer.Execute(transform)

print(output_transform)


# Resample the transformed moving image onto the fixed image.
output_image = sitk.Resample(
    moving_image, fixed_image, transform=output_transform, defaultPixelValue=150
)

# Write the resampled image.
sitk.WriteImage(output_image, "landmark_example.png")


# Write the transform.
if len(sys.argv) > 1:
    out_name = sys.argv[1]
else:
    out_name = "landmark_transform.tfm"

sitk.WriteTransform(output_transform, out_name)
```

14.26 Integration with ITK

14.26.1 Overview

14.26.2 Code

C++

```
#if defined(_MSC_VER)
#pragma warning ( disable : 4786 )
#endif

// SimpleITK includes
#include "SimpleITK.h"

// ITK includes
#include "itkImage.h"
#include "itkCurvatureFlowImageFilter.h"

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

/**
 * This example shows how ITK and SimpleITK can be used together to work
 * on the same data. We use the same example application as the one presented
 * in the Segmentation/ConnectedThresholdImageFilter.cxx example, but we
 * replace the SimpleITK version of CurvatureFlowImageFilter with the
 * corresponding ITK version. While not terribly useful in this situation since
 * CurvatureFlowImageFilter is already available in SimpleITK this demonstrates
 * how ITK filters that have not been converted for SimpleITK can still be used
 * in a SimpleITK context
 */
int main( int argc, char *argv[])
{

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

(continues on next page)

(continued from previous page)

```
  //////
  // Blur using CurvatureFlowImageFilter
  //
  // Here we demonstrate the use of the ITK version of CurvatureFlowImageFilter
  // instead of the SimpleITK version.
  //////

  //
  // First, define the type alias that correspond to the types of the input
  // image. This requires foreknowledge of the data type of the input image.
  //
  const unsigned int                                Dimension = 2;
  using InternalPixelType = float;
  using InternalImageType = itk::Image< InternalPixelType, Dimension >;

  //
  // We must check the image dimension and the pixel type of the
  // SimpleITK image match the ITK image we will cast to.s
  //
  if ( image.GetDimension() != Dimension )
    {
    std::cerr << "Input image is not a " << Dimension << " dimensional image as expected!
↪" << std::endl;
    return 1;
    }

  //
  // The read sitk::Image could be any pixel type. Cast the image, to
  // float so we know what type we have.
  //
  sitk::CastImageFilter caster;
  caster.SetOutputPixelType( sitk::sitkFloat32 );
  image = caster.Execute( image );

  //
  // Extract the itk image from the SimpleITK image
  //
  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;
    }

  //
```

(continues on next page)

(continued from previous page)

```
  // Set up the blur filter and attach it to the pipeline.
  //
  using BlurFilterType = itk::CurvatureFlowImageFilter< InternalImageType,␣
↪InternalImageType >;
  BlurFilterType::Pointer blurFilter = BlurFilterType::New();
  blurFilter->SetInput( itkImage );
  blurFilter->SetNumberOfIterations( 5 );
  blurFilter->SetTimeStep( 0.125 );



  //
  // Execute the  Blur pipeline by calling Update() on the blur filter.
  //
  blurFilter->Update();


  //
  // Return to the simpleITK setting by making a SimpleITK image using the
  // output of the blur filter.
  //
  sitk::Image blurredImage = sitk::Image( blurFilter->GetOutput() );


  //////
  // Now that we have finished the ITK section, we return to the SimpleITK API
  //////


  //
  // Set up ConnectedThresholdImageFilter for segmentation
  //
  sitk::ConnectedThresholdImageFilter segmentationFilter;
  segmentationFilter.SetLower( atof( argv[3] ) );
  segmentationFilter.SetUpper( atof( argv[4] ) );
  segmentationFilter.SetReplaceValue( 255 );

  for (int i = 5; i+1 < argc; i+=2)
    {
    std::vector<unsigned int> seed = { (unsigned int) atoi(argv[i]), (unsigned int)␣
↪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);
```

(continues on next page)

(continued from previous page)

```
  //
  // Write out the resulting file
  //
  writer.Execute(outImage);

  return 0;
}
```

14.27 N4 Bias Field Correction

14.27.1 Overview

The N4 bias field correction algorithm is a popular method for correcting low frequency intensity non-uniformity present in MRI image data known as a bias or gain field. The method has also been successfully applied as flat-field correction in microscopy data. This method assumes a simple parametric model and does not require tissue classification.

This example demonstrates how to use the SimpleITK N4BiasFieldCorrectionImageFilter class. This filter has one required input image which is affected by a bias field we wish to correct. The primary input is required to have a "real" pixel type of either *sitkFloat32* or *sitkFloat64*. Additionally, there is an optional "MaskImage" input which specifies which pixels are used to estimate the bias-field and suppress pixels close to zero. It is recommended that the mask image follows the common conventions for *masked images*, which is being of pixel type *sitkUint8* and having values of 0 and 1 representing the mask. Additionally, the mask image and the main input image must occupy the same physical space to ensure pixel to pixel correspondence.

Some basic parameters for using this algorithm are parsed from the command line in this example. The shrink factor is used to reduce the size and complexity of the image. The N4 algorithm uses a multi-scale optimization approach to compute the bias field. The **SetMaximumNumberOfIterations** method takes an array used to set the limit of iterations per resolution level, thereby setting both the iterations and the number of scales (from the length of the array). The output of the filter is the bias corrected input image.

14.27.2 Example Run

Running the Python code with the following inputs:

```
main("BrainProtonDensitySlice.png", "N4_corrected.nrrd", 1)
```

produces the images below.

Input Images



Fig. 15: **Original Input**

Output Images

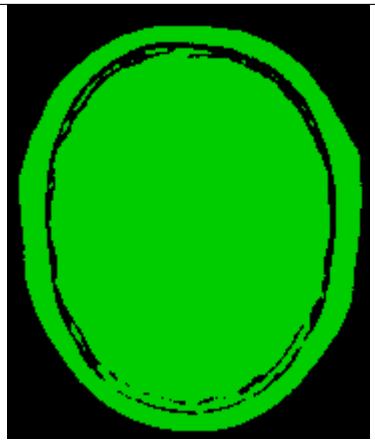


Fig. 16: **Default Mask Image**

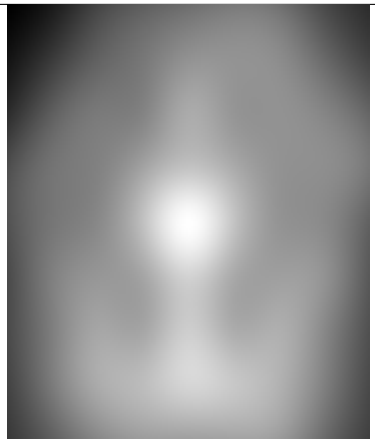


Fig. 17: **Log Bias Image**

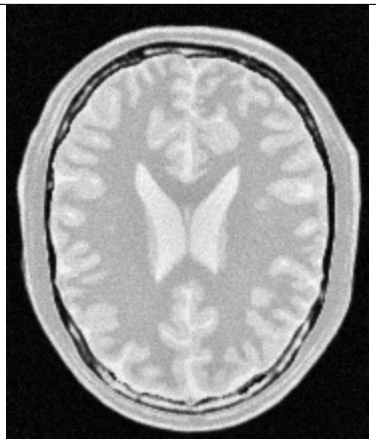


Fig. 18: **N4 Bias Corrected Image**

14.27.3 Code

C++

C#

Python

R

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


namespace sitk = itk::simple;

```

(continues on next page)

(continued from previous page)

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

  if ( argc < 2 ) {
    std::cerr << "Usage: N4BiasFieldCorrection inputImage outputImage";
    std::cerr << " [shrinkFactor] [maskImage] [numberOfIterations]";
    std::cerr << " [numberOfFittingLevels]\n";
    return 1;
  }

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

  sitk::Image maskImage;
  if ( argc > 4 ) {
    maskImage = sitk::ReadImage( argv[4], sitk::sitkUInt8 );
  } else {
    maskImage = sitk::OtsuThreshold( image, 0, 1, 200 );
  }

  unsigned int shrinkFactor = 1;
  if ( argc > 3 ) {
    shrinkFactor = atoi( argv[3] );
    std::vector<unsigned int> shrink( inputImage.GetDimension(), shrinkFactor );
    image = sitk::Shrink( inputImage, shrink );
    maskImage = sitk::Shrink( maskImage, shrink );
  }

  sitk::N4BiasFieldCorrectionImageFilter corrector;

  unsigned int numFittingLevels = 4;

  if ( argc > 6) {
    numFittingLevels = atoi(argv[6]);
  }

  if ( argc > 5 ) {
    unsigned int it = atoi( argv[5] );
    std::vector<unsigned int> iterations( numFittingLevels, it );
    corrector.SetMaximumNumberOfIterations( iterations );
  }

  sitk::Image corrected_image = corrector.Execute( image, maskImage );

  sitk::Image log_bias_field = corrector.GetLogBiasFieldAsImage( inputImage );

  sitk::Image corrected_image_full_resolution = sitk::Divide( inputImage, sitk::Exp( log_
↪bias_field ) );

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

  if (shrinkFactor > 1) {
    sitk::WriteImage( corrected_image, "CXX-Example-N4BiasFieldCorrection-shrunk.nrrd" );
```

(continues on next page)

(continued from previous page)

```
  }

  if (getenv("SITK_NOSHOW") == NULL)
    sitk::Show(corrected_image, "N4 Corrected");

  return 0;
}
```

```
using System;
using itk.simple;
using sitk = itk.simple.SimpleITK;

namespace itk.simple.examples {
  class N4BiasFieldCorrection {
    static void Main(string[] args) {
      try {
        if (args.Length < 2) {
          Console.WriteLine("Usage: N4BiasFieldCorrection inputImage outputImage"
            + " [shrinkFactor] [maskImage] [numberOfIterations]"
            + " [numberOfFittingLevels]\n");

          return;
        }
        // Read input image
        Image inputImage = sitk.ReadImage(args[0], PixelIDValueEnum.sitkFloat32);
        Image image = inputImage;


        Image maskImage;
        if (args.Length>3) {
          maskImage = sitk.ReadImage(args[3], PixelIDValueEnum.sitkUInt8);
        } else {
          maskImage = sitk.OtsuThreshold(image, 0, 1, 200);
        }

        uint shrinkFactor = 1;
        if (args.Length>2) {
          shrinkFactor = UInt32.Parse(args[2]);
          uint[] s_array = new uint[image.GetDimension()];
          for (uint i=0; i<image.GetDimension(); i++) {
            s_array[i] = shrinkFactor;
          }
          VectorUInt32 shrink = new VectorUInt32(s_array);
          image = sitk.Shrink(inputImage, shrink);
          maskImage = sitk.Shrink(maskImage, shrink);
        }

        N4BiasFieldCorrectionImageFilter corrector
          = new N4BiasFieldCorrectionImageFilter();

        uint numFittingLevels = 4;
```

(continues on next page)

(continued from previous page)

```
        if (args.Length>5) {
          numFittingLevels = UInt32.Parse(args[5]);
        }

        if (args.Length>4) {
          uint it = UInt32.Parse(args[4]);
          uint[] it_array = new uint[numFittingLevels];
          for (uint i=0; i<numFittingLevels; i++) {
            it_array[i] = it;
          }
          VectorUInt32 iterations = new VectorUInt32(it_array);
          corrector.SetMaximumNumberOfIterations(iterations);
        }

        Image corrected_image = corrector.Execute(image, maskImage);

        Image log_bias_field = corrector.GetLogBiasFieldAsImage(inputImage);

        Image corrected_image_full_resolution = sitk.Divide(inputImage, sitk.Exp(log_
↪bias_field));

        sitk.WriteImage(corrected_image_full_resolution, args[1]);

        if (shrinkFactor>1) {
          sitk.WriteImage(corrected_image, "CSharp-Example-N4BiasFieldCorrection-shrunk.
↪nrrd");
        }

        if (Environment.GetEnvironmentVariable("SITK_NOSHOW") == null)
          SimpleITK.Show(corrected_image, "N4 Corrected");

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

```
#!/usr/bin/env python
import SimpleITK as sitk
import sys
import os


def main(args):
    if len(args) < 2:
        print(
            "Usage: N4BiasFieldCorrection inputImage "
            + "outputImage [shrinkFactor] [maskImage] [numberOfIterations] "
            + "[numberOfFittingLevels]"
        )
        sys.exit(1)
```

(continues on next page)

(continued from previous page)

```
    inputImage = sitk.ReadImage(args[1], sitk.sitkFloat32)
    image = inputImage

    if len(args) > 4:
        maskImage = sitk.ReadImage(args[4], sitk.sitkUInt8)
    else:
        maskImage = sitk.OtsuThreshold(inputImage, 0, 1, 200)

    shrinkFactor = 1
    if len(args) > 3:
        shrinkFactor = int(args[3])
        if shrinkFactor > 1:
            image = sitk.Shrink(inputImage, [shrinkFactor] * inputImage.GetDimension())
            maskImage = sitk.Shrink(
                maskImage, [shrinkFactor] * inputImage.GetDimension()
            )

    corrector = sitk.N4BiasFieldCorrectionImageFilter()

    numberFittingLevels = 4

    if len(args) > 6:
        numberFittingLevels = int(args[6])

    if len(args) > 5:
        corrector.SetMaximumNumberOfIterations([int(args[5])] * numberFittingLevels)

    corrected_image = corrector.Execute(image, maskImage)

    log_bias_field = corrector.GetLogBiasFieldAsImage(inputImage)

    corrected_image_full_resolution = inputImage / sitk.Exp(log_bias_field)

    sitk.WriteImage(corrected_image_full_resolution, args[2])

    if shrinkFactor > 1:
        sitk.WriteImage(
            corrected_image, "Python-Example-N4BiasFieldCorrection-shrunk.nrrd"
        )

    return_images = {
        "input_image": inputImage,
        "mask_image": maskImage,
        "log_bias_field": log_bias_field,
        "corrected_image": corrected_image,
    }
    return return_images


if __name__ == "__main__":
```

```
# Run with:
#
# Rscript --vanilla N4BiasFieldCorrection.R inputImage, outputImage, shrinkFactor,␣
↪maskImage, numberOfIterations, numberOfFittingLevels
#

library(SimpleITK)

args <- commandArgs( TRUE )

if (length(args) < 2) {
    stop("At least 2 arguments expected - inputImage, outputImage, [shrinkFactor], ",
         "[maskImage], [numberOfIterations], [numberOfFittingLevels]")
}

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

if (length( args ) > 4) {
    maskImage <- ReadImage( args[[4]], 'sitkUint8' )
} else {
    maskImage <- OtsuThreshold( inputImage, 0, 1, 200 )
}

shrinkFactor = 1
if (length( args ) > 3) {
    shrinkFactor = strtoi(args[3])
    image <- Shrink( inputImage, rep(shrinkFactor, inputImage$GetDimension()) )
    maskImage <- Shrink( maskImage, rep(shrinkFactor, inputImage$GetDimension()) )
}

inputImage <- Cast( inputImage, 'sitkFloat32' )

corrector <- N4BiasFieldCorrectionImageFilter()

numberFittingLevels <- 4

if (length ( args ) > 6) {
    numberFittingLevels <- strtoi( args[[6]] )
}

if (length ( args ) > 5) {
    corrector$SetMaximumNumberOfIterations( rep(strtoi( args[[5]], numberFittingLevels))␣
↪)
}

correctedImage <- corrector$Execute( image, maskImage )

logBiasField <- corrector$GetLogBiasFieldAsImage(inputImage)

correctedImageFullResolution <- inputImage / Exp( logBiasField )

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

(continues on next page)

(continued from previous page)

```
if (shrinkFactor > 1) {
    WriteImage(correctedImage, "R-Example-N4BiasFieldCorrection-shrunk.nrrd")
}

if(Sys.getenv("SITK_NOSHOW") == "") {
    Show(correctedImage, "N4 Corrected")
}
```

14.28 Reading-Gaussian Blurring-Writing

14.28.1 Overview

Introductory example which demonstrates 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.

14.28.2 Example Run

Running the Python code as follows:

```
main('docs/images/BrainProtonDensitySlice.png', 2.0, 'output.png')
```

modifies the input image to the output image below.

Input/Output Images

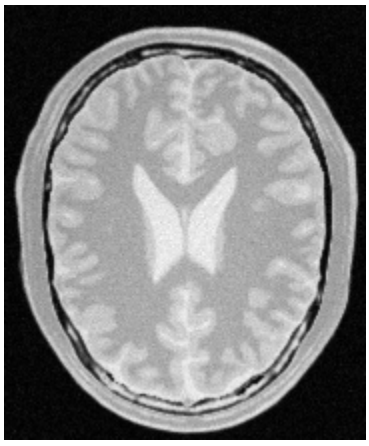


Fig. 19: **Input Image**

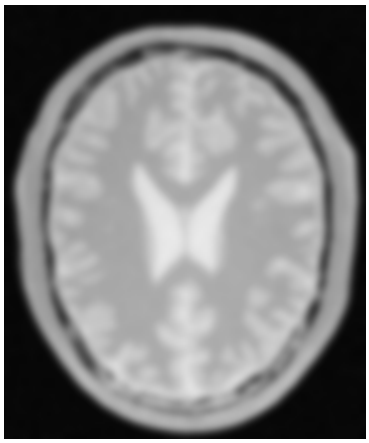


Fig. 20: **Output Image**

14.28.3 Code

C#

C++

Java

Lua

Python

R

Ruby

Tcl

```
using System;
using itk.simple;

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

                // Execute Gaussian smoothing filter
                SmoothingRecursiveGaussianImageFilter gaussian = new␣
↪SmoothingRecursiveGaussianImageFilter();
                gaussian.SetSigma(Double.Parse(args[1]));
                Image blurredImage = gaussian.Execute(image);
```

(continues on next page)

(continued from previous page)

```
                // 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);
            }
        }
    }
}
```

```
// 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;
  }

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

  // This filters perform a gaussian blurring 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;
```

(continues on next page)

(continued from previous page)

```
  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;
}
```

```
/**
 * 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 );

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

  }

}
```

```
require "SimpleITK"

if #arg < 3 then
```

(continues on next page)

(continued from previous page)

```
  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 );
```

```
#!/usr/bin/env python
import sys
import SimpleITK as sitk


def main(args):
    if len(args) < 4:
        print("Usage: SimpleGaussian <input> <sigma> <output>")
        sys.exit(1)

    reader = sitk.ImageFileReader()
    reader.SetFileName(args[1])
    input_image = reader.Execute()

    pixelID = input_image.GetPixelID()

    gaussian = sitk.SmoothingRecursiveGaussianImageFilter()
    gaussian.SetSigma(float(args[2]))
    blur_image = gaussian.Execute(input_image)

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

    writer = sitk.ImageFileWriter()
    writer.SetFileName(args[3])
    writer.Execute(blur_image)
```

(continues on next page)

(continued from previous page)

```
    return_dict = {"input_image": input_image, "blur_image": blur_image}
    return return_dict


# Display the results
```

```
# 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 )
```

```
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] )
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
```

(continues on next page)

(continued from previous page)

```
caster.set_output_pixel_type inputPixelType
image = caster.execute image

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

```
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
```

14.29 Image Viewing

14.29.1 Overview

This example illustrates the basic image viewing capabilities provided by SimpleITK.

The focus of SimpleITK is on image analysis and not display. We therefor use an ad-hoc image viewing approach, relying on an external program. By default this is the Fiji/ImageJ program.

When using this functionality, SimpleITK writes the image to disc in a temporary location and then launches the viewer. Potential issues with this approach:

1. The external viewer program is not found. Either it is not installed, or not installed in the expected location, for details see *Why isn't Fiji or ImageJ found by the Show function (RuntimeError: Exception thrown...)?*)

2. Writing the image to disc fails when there isn't enough disc space.
3. The external program cannot read the image file because it was saved in a format not supported by the program.

To control viewing you have two options:

1. Procedural approach. Use the Show function, with control primarily done via environment variable settings.
2. Object oriented approach. Use the ImageViewer class, with control via the object's interface.

One can either set the viewing application, if you just want to change the specific viewer, or set the viewing command. The latter allows you to specify viewer specific arguments such as -z for ITK-SNAP, see below, to open in a zoomed view.

14.29.2 Code

Python

R

```python
import sys
import SimpleITK as sitk

grid_image = sitk.GridSource(
    outputPixelType=sitk.sitkUInt16,
    size=(512, 512),
    sigma=(0.1, 0.1),
    gridSpacing=(20.0, 20.0),
)

# Procedural interface, using the default image viewer (Fiji/ImageJ) or
# any viewer specified by the SITK_SHOW_COMMAND environment variable.
sitk.Show(grid_image, title="grid using Show function", debugOn=True)

# Object oriented interface:
image_viewer = sitk.ImageViewer()
image_viewer.SetTitle("grid using ImageViewer class")

# Use the default image viewer.
image_viewer.Execute(grid_image)

# Change viewer, and display again.
image_viewer.SetApplication("/Applications/ITK-SNAP.app/Contents/MacOS/ITK-SNAP")
image_viewer.Execute(grid_image)

# Change the viewer command, (use ITK-SNAP's -z option to open the image in
# zoomed mode)
image_viewer.SetCommand("/Applications/ITK-SNAP.app/Contents/MacOS/ITK-SNAP -z 2")
image_viewer.Execute(grid_image)

sys.exit(0)
```

```r
# Run with:
#
# Rscript --vanilla ImageViewing.R
```

(continues on next page)

(continued from previous page)

```
#

library(SimpleITK)

grid_image <- GridSource(outputPixelType="sitkUInt16", size=c(512,512),
                         sigma=c(0.1,0.1), gridSpacing=c(20.0,20.0))

# Procedural interface, using the default image viewer (Fiji/ImageJ) or
# any viewer specified by the SITK_SHOW_COMMAND environment variable.
Show(grid_image, title = "grid using Show function", debugOn = TRUE)

# Object oriented interface:
image_viewer <- ImageViewer()
image_viewer$SetTitle('grid using ImageViewer class')

# Use the default image viewer.
image_viewer$Execute(grid_image)

# Change viewer, and display again.
image_viewer$SetApplication('/Applications/ITK-SNAP.app/Contents/MacOS/ITK-SNAP')
image_viewer$Execute(grid_image)

# Change the viewer command, (use ITK-SNAP's -z option to open the image in zoomed mode)
image_viewer$SetCommand('/Applications/ITK-SNAP.app/Contents/MacOS/ITK-SNAP -z 2')
image_viewer$Execute(grid_image)
```

14.30 Slice by Slice Adaptive Histogram Equalization

14.30.1 Overview

Most SimpleITK filters can only operate on 2 or 3 dimensional images, with the exception of filters such as *ExtractImageFilter*, *PasteImageFilter*, *SliceImageFilter* and *JoinSeriesImageFilter*. However, SimpleITK (by default) supports upto 5 dimensional images. A high dimensional image can be processed by extracting 2 or 3 dimensional images, then either using the *JoinSeriesImageFilter* to join the sub-volumes together or the *PasteImageFilter* to copy the results back to the original image. Possible reasons include when the z direction spacing is too great, or for computation or memory efficient reasons. Additionally, it may be desired to process a volume (3d or higher) as a sequence of 2 dimensional images.

In this example, the *AdaptiveHistogramEqualizationImageFilter* is used to processes a higher dimensional image as a sequence of two dimensional images.. When the filter is run only a single X, Y cross-section of the volume.

Both the Python and the C++ examples demonstrate a reusable "decorator" to wrap the SimpleITK *AdaptiveHistogramEqualization* procedure to process the input image by 2d slices. A function decorator is a function which takes a function as an argument and returns a modified or wrapped function. The decorators are written generically, so they can work with other SimpleITK procedures or a custom procedure. The decorators wrap the procedure to accept a 2, 3, 4 or 5 dimensional image as input. The pasting approach is used to provide a memory efficient approach.

The process of extracting a slice, processing, and then pasting back to the original image is straight forward. However, to create a reusable decorator requires advanced language specific features. Additionally, to efficiently do pasting *in place execution* is done in C++, and sliced indexed assignment is used in Python.

14.30.2 Code

Python

C++

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


import SimpleITK as sitk
import sys

import itertools
from functools import wraps


def slice_by_slice_decorator(func):
    """
    A function decorator which executes func on each 3D sub-volume and
    *in-place* pastes the results into the input image. The input image type
    and the output image type are required to be the same type.

    :param func: A function which take a SimpleITK Image as it's first
    argument and returns an Image as results.

    :return: A decorated function.
    """

    iter_dim = 2

    @wraps(func)
    def slice_by_slice(image, *args, **kwargs):

        dim = image.GetDimension()

        if dim <= iter_dim:
            #
            image = func(image, *args, **kwargs)
            return image

        extract_size = list(image.GetSize())
        extract_size[iter_dim:] = itertools.repeat(0, dim - iter_dim)

        extract_index = [0] * dim
        paste_idx = [slice(None, None)] * dim

        extractor = sitk.ExtractImageFilter()
        extractor.SetSize(extract_size)

        for high_idx in itertools.product(
            *[range(s) for s in image.GetSize()[iter_dim:]]
        ):

            # The lower 2 elements of extract_index are always 0.
```

(continues on next page)

(continued from previous page)

```
            # The remaining indices are iterated through all indexes.
            extract_index[iter_dim:] = high_idx
            extractor.SetIndex(extract_index)

            # Sliced based indexing for setting image values internally uses
            # the PasteImageFilter executed "in place".  The lower 2 elements
            # are equivalent to ":". For a less general case the assignment
            # could be written as image[:,:,z] = ...
            paste_idx[iter_dim:] = high_idx
            image[paste_idx] = func(extractor.Execute(image), *args, **kwargs)

        return image

    return slice_by_slice


if len(sys.argv) < 3:
    print("Usage: SubDimensionProcess inputImage outputImage", file=sys.stderr)
    sys.exit(1)

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

# Decorate the function
adaptive_histogram_equalization_2d = slice_by_slice_decorator(
    sitk.AdaptiveHistogramEqualization
)

adaptive_histogram_equalization_2d(inputImage, radius=[20] * 2, alpha=0.3, beta=0.3)

sitk.WriteImage(inputImage, sys.argv[2])
```

```
#include "SimpleITK.h"
#include "sitkExtractImageFilter.h"

#include <cstdlib>
#include <iostream>

namespace sitk = itk::simple;

using itk::simple::operator<<;

// Forward declaration to specialize the implementation with the function's
// argument and returns types.
template <class T> struct SliceBySliceDecorator;

/* \brief A function decorator to adapt an function to process an image as a
 * sequence of 2D (slices or) images.
 *
 * For ease of use the makeSliceBySlice procedure should be used to construct
 * the SliceBySliceDecorator.
 *
 * The overloaded function call operator(), enable objects to be used as
```

(continues on next page)

(continued from previous page)

```
 * functions.
 *
 * The return image is the first argument modified with the slice by slice
 * results of the f function.
 */
template <class R, class ImageArg, class... Args>
struct SliceBySliceDecorator<R(ImageArg, Args...)>
{
  using FunctionType = std::function<R(ImageArg, Args...)>;

  explicit SliceBySliceDecorator(FunctionType f) : f_(std::move(f)) {}

  R operator()(sitk::Image &image, Args... args)
  {
    const auto image_size = image.GetSize();

    unsigned int dim = image.GetDimension();

    if (dim <= iter_dim)
    {
      // If no sub-dimension extraction is required then directly run the
      // function on the input and replace it.
      image = f_(image, args...);
      return image;
    }

    std::vector<unsigned int> extract_size = image.GetSize();
    std::fill(extract_size.begin() + iter_dim, extract_size.end(), 0);

    std::vector<int> extract_index(dim, 0);

    // The extract filter is used to extract a 2D image from the higher
    // dimensional input image.
    sitk::ExtractImageFilter extractor;
    extractor.SetSize(extract_size);

    // The paste filter places the processed slice back into the original image
    // at the same location where the extraction occurred. The default
    // value of DestinationSkipAxes is [ false, false, true, ... ], which is
    // correct for the situation of preserving the first dimensions and
    // collapsing the remainder.
    sitk::PasteImageFilter paster;
    paster.SetSourceSize(std::vector<unsigned int>(
        extract_size.begin(), extract_size.begin() + iter_dim));


    while (static_cast<unsigned int>(extract_index.back()) < image.GetSize().back())
    {
      extractor.SetIndex(extract_index);

      // Store the results of the function as a r-value, so that the
      // paste filter will run "in place" and reuse the buffer for output.
```

(continues on next page)

(continued from previous page)

```
      sitk::Image &&temp_image = f_(extractor.Execute(image), args...);

      paster.SetDestinationIndex(extract_index);

      image = paster.Execute(image, temp_image);

      // increment extraction index
      ++extract_index[iter_dim];
      for (unsigned int e = iter_dim; e + 1 < dim; ++e)
      {
        // if the index element is beyond the size, propagate to next element.
        if (static_cast<unsigned int>(extract_index[e]) > image.GetSize()[e])
        {
          extract_index[e] = 0;
          ++extract_index[e + 1];
        }
      }
    }
    return image;
  }
  FunctionType f_;
  constexpr static unsigned int iter_dim = 2;
};

/** Construct a function object to wrap a function f to process a volume slice by
 * slice with the f function.
 *
 * The f function must take an sitk::Image object as the first parameter and
 * return an sitk::Image.
 */
template <class R, class... Args>
SliceBySliceDecorator<R(Args...)> makeSliceBySlice(R (*f)( Args...))
{
  using DecoratorType = SliceBySliceDecorator<R(Args...)>;
  return DecoratorType(typename DecoratorType::FunctionType(f));
}

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

  if ( argc < 3 )
  {
    std::cerr << "Usage: " << argv[0] << " <inputImage> <outputImage>" << std::endl;
    return 1;
  }

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

  // The parameters to the filter are hard coded to simplify the example.
  float alpha = 0.3f;
  float beta = 0.3f;
  std::vector<unsigned int> radius(2, 20);
```

(continues on next page)

(continued from previous page)

```
  // The return type of the function decorator is complex, so the auto type is
  // used for brevity.
  auto AdaptiveHistogramEqualization2D =␣
↪makeSliceBySlice(sitk::AdaptiveHistogramEqualization);

  // Execute the decorated function.
  tempImage = AdaptiveHistogramEqualization2D(tempImage, radius, alpha, beta);

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

  return 0;
}
```

14.31 Logging

14.31.1 Overview

SimpleITK creates debug, warning and error messages intended for display or logging. The messaging and logging are handled by a customizable class in ITK, the *OutputWindow*. The default implementation prints these messages to the standard error. A derived class could print the messages to standard out or adapt the output to fit within another logging framework. Errors in SimpleITK usually cause exceptions so any error message is unlikely. Warnings are more likely and are enabled globally by default, but can be disabled with the ProcessObject::SetGlobalWarningDisplay method. Debug messages are set on a per object basis with the *SetDebug* method. The default value for new objects is set by the ProcessObject::SetGlobalDefaultDebug method. Disabled messages are not sent to the logging handler for efficiency.

The following example illustrates how to create a custom logging handler in SimpleITK. The example derives a class from the LoggerBase class to customize the logging behavior. The custom logger is then set as the ITK global logging handler, and then the original logger is restored.

14.31.2 Code

Python

```
import SimpleITK as sitk
import logging
import os


class SimpleITKLogger(sitk.LoggerBase):
    """
    Adapts SimpleITK messages to be handled by a Python Logger object.

    Allows using the logging module to control the handling of messages coming
    from ITK and SimpleTK. Messages such as debug and warnings are handled by
    objects derived from sitk.LoggerBase.

    The LoggerBase.SetAsGlobalITKLogger method must be called to enable
    SimpleITK messages to use the logger.
```

(continues on next page)

(continued from previous page)

```
    The Python logger module adds a second layer of control for the logging
    level in addition to the controls already in SimpleITK.

    The "Debug" property of a SimpleITK object must be enabled (if
    available) and the support from the Python "logging flow" hierarchy
    to handle debug messages from a SimpleITK object.

    Warning messages from SimpleITK are globally disabled with
    ProcessObject:GlobalWarningDisplayOff.
    """

    def __init__(self, logger: logging.Logger = logging.getLogger("SimpleITK")):
        """
        Initializes with a Logger object to handle the messages emitted from
        SimpleITK/ITK.
        """
        super(SimpleITKLogger, self).__init__()
        self._logger = logger

    @property
    def logger(self):
        return self._logger

    @logger.setter
    def logger(self, logger):
        self._logger = logger

    def __enter__(self):
        self._old_logger = self.SetAsGlobalITKLogger()
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        self._old_logger.SetAsGlobalITKLogger()
        del self._old_logger

    def DisplayText(self, s):
        # Remove newline endings from SimpleITK/ITK messages since the Python
        # logger adds during output.
        self._logger.info(s.rstrip())

    def DisplayErrorText(self, s):
        self._logger.error(s.rstrip())

    def DisplayWarningText(self, s):
        self._logger.warning(s.rstrip())

    def DisplayGenericOutputText(self, s):
        self._logger.info(s.rstrip())

    def DisplayDebugText(self, s):
        self._logger.debug(s.rstrip())
```

(continues on next page)

(continued from previous page)

```
# Enable all debug messages for all ProcessObjects, and procedures
sitk.ProcessObject.GlobalDefaultDebugOn()

# Construct a SimpleITK logger to Python Logger adaptor
sitkLogger = SimpleITKLogger()

# Configure ITK to use the logger adaptor
sitkLogger.SetAsGlobalITKLogger()

# Configure the Python root logger, enabling debug and info level messages.
logging.basicConfig(format="%(name)s (%(levelname)s): %(message)s", level=logging.DEBUG)


img = sitk.GaborSource(size=[64, 64, 64], frequency=4.0 / 64)

with SimpleITKLogger(logging.getLogger("Show")) as showLogger:
    print(f"Logger name: {showLogger.GetName()}")

    if "SITK_NOSHOW" not in os.environ:
        sitk.Show(img, debugOn=True)
```

14.32 Advanced Image Reading

14.32.1 Overview

This example illustrates advanced image reading options:

1. Querying an image for its meta-data without reading the bulk intensity data.
2. Reading a sub-region of an image.

Querying an image for its meta-data

In some cases you may only want to read an image's meta-data without reading the bulk intensity data into memory. For instance, if you want to read subregions of an image and not the whole image or if you want to display image information (e.g. patient name) to a user in a GUI which then allows them to select a specific image.

Reading an image sub-region

In some cases you are only interested in reading a sub-region of an image. This may be due to memory constraints or if it is known that the relevant content is always in a sub region of the image. If the specific image IO used for this operation supports streaming then this will indeed only read the sub-region, otherwise the whole image is read into memory and the sub-region is returned. The IOs that support streaming are: 1. TIFFImageIO (supports a subset of encodings) 2. MetaImageIO 3. NrrdImageIO 4. HDF5ImageIO (supports a subset of encodings) 5. MRCImageIO 6. VTKImageIO (supports a subset of encodings)

14.32.2 Code

Python

R

```python

import sys

import SimpleITK as sitk
import numpy as np

if len(sys.argv) < 2:
    print("Wrong number of arguments.", file=sys.stderr)
    print("Usage: " + __file__ + " image_file_name", file=sys.stderr)
    sys.exit(1)

# Read image information without reading the bulk data.
file_reader = sitk.ImageFileReader()
file_reader.SetFileName(sys.argv[1])
file_reader.ReadImageInformation()
print(f"image size: {file_reader.GetSize()}")
print(f"image spacing: {file_reader.GetSpacing()}")

# Some files have a rich meta-data dictionary (e.g. DICOM)
for key in file_reader.GetMetaDataKeys():
    print(key + ": " + file_reader.GetMetaData(key))
print("-" * 20)

# When low on memory, we can incrementally work on sub-images. The following
# subtracts two images (ok, the same image) by reading them as multiple'
# sub-images.

image1_file_name = sys.argv[1]
image2_file_name = sys.argv[1]

parts = 5  # Number of sub-regions we use

file_reader = sitk.ImageFileReader()
file_reader.SetFileName(image1_file_name)
file_reader.ReadImageInformation()
image_size = file_reader.GetSize()

result_img = sitk.Image(
    file_reader.GetSize(),
    file_reader.GetPixelID(),
    file_reader.GetNumberOfComponents(),
)
result_img.SetSpacing(file_reader.GetSpacing())
result_img.SetOrigin(file_reader.GetOrigin())
result_img.SetDirection(file_reader.GetDirection())

extract_size = list(file_reader.GetSize())
extract_size[-1] = extract_size[-1] // parts
```

(continues on next page)

(continued from previous page)

```
current_index = [0] * file_reader.GetDimension()
for i in range(parts):
    if i == (parts - 1):
        extract_size[-1] = image_size[-1] - current_index[-1]
    file_reader.SetFileName(image1_file_name)
    file_reader.SetExtractIndex(current_index)
    file_reader.SetExtractSize(extract_size)
    sub_image1 = file_reader.Execute()

    file_reader.SetFileName(image2_file_name)
    file_reader.SetExtractIndex(current_index)
    file_reader.SetExtractSize(extract_size)
    sub_image2 = file_reader.Execute()
    idx = [slice(None, None)] * file_reader.GetDimension()
    idx[-1] = current_index[-1]
    result_img[idx] = sub_image1 - sub_image2
    current_index[-1] += extract_size[-1]
del sub_image1
del sub_image2

# Check that our iterative approach is equivalent to reading the whole images.
if np.any(
    sitk.GetArrayViewFromImage(
        result_img - sitk.ReadImage(image1_file_name) + sitk.ReadImage(image2_file_name)
    )
):
    print("Subtraction error.")
    sys.exit(1)
sys.exit(0)
```

```
library(SimpleITK)

args <- commandArgs( TRUE )

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

# Read image information without reading the bulk data.
file_reader <- ImageFileReader()
file_reader$SetFileName(args[1])
file_reader$ReadImageInformation()
cat('image size:', file_reader$GetSize(), '\n')
cat('image spacing:', file_reader$GetSpacing(), '\n')
# Some files have a rich meta-data dictionary (e.g. DICOM)
for(key in file_reader$GetMetaDataKeys())
{
  cat(paste0(key, ': ', file_reader$GetMetaData(key), '\n'))
}
cat(rep('-',20),'\n', sep='')
```

(continues on next page)

(continued from previous page)

```
# When low on memory, we can incrementally work on sub-images. The following
# subtracts two images (ok, the same image) by reading them as multiple sub-images.

image1_file_name <- args[1]
image2_file_name <- args[1]

parts <- 5 # Number of sub-regions we use

file_reader <- ImageFileReader()
file_reader$SetFileName(image1_file_name)
file_reader$ReadImageInformation()
image_size <- file_reader$GetSize()

result_img <- Image(file_reader$GetSize(), file_reader$GetPixelID(), file_reader
↪$GetNumberOfComponents())
result_img$SetSpacing(file_reader$GetSpacing())
result_img$SetOrigin(file_reader$GetOrigin())
result_img$SetDirection(file_reader$GetDirection())

extract_size <- file_reader$GetSize()
extract_size[-1] <- extract_size[-1]%/%parts
current_index <- rep(0,file_reader$GetDimension())
for(i in 1:parts)
{
  if(i == parts)
  {
    extract_size[-1] <- image_size[-1] - current_index[-1]
  }
  file_reader$SetFileName(image1_file_name)
  file_reader$SetExtractIndex(current_index)
  file_reader$SetExtractSize(extract_size)
  sub_image1 <- file_reader$Execute()

  file_reader$SetFileName(image2_file_name)
  file_reader$SetExtractIndex(current_index)
  file_reader$SetExtractSize(extract_size)
  sub_image2 <- file_reader$Execute()
  result_img <- Paste(result_img, sub_image1 - sub_image2, extract_size, rep(0,file_
↪reader$GetDimension()), current_index)
  current_index[-1] <- current_index[-1] + extract_size[-1]
}

rm(sub_image1, sub_image2)

# Check that our iterative approach is equivalent to reading the whole images.
if(any(as.array(result_img - ReadImage(image1_file_name) + ReadImage(image2_file_name))!
↪=0))
{
  cat('Subtraction error.\n')
  quit(save="no", status=1)
}
quit(save="no", status=0)
```

14.33 IO Selection for Image Reading

14.33.1 Overview

This example illustrates how to explicitly select a specific IO for image reading.

When using the default settings for the ImageFileReader class or the ReadImage function you have minimal control over the reading. That is, the specific IO mechanism is determined automatically based on the file type and the file is read into memory.

In some cases there are multiple IO classes that support reading the same image format and you do not know which one was used. For example the 'SCIFIOImageIO' and 'JPEGImageIO' both support reading JPEG images. As these are different implementations they may vary in performance and in the support of the features associated with the specific format. As a consequence, you may want or need to select one implementation over the other. Explicitly selecting the IO allows you to do so.

14.33.2 Code

C#

C++

Java

Lua

Python

R

Ruby

Tcl

```
using System;
using itk.simple;

namespace itk.simple.examples {
    class ImageIOSelection {
        static void Main(string[] args) {
            try {
                if (args.Length < 1) {
                    Console.WriteLine("Usage: ImageIOSelection image_input_file");
                    return;
                }

                // Find out which image IOs are supported
                ImageFileReader reader = new ImageFileReader();
                itk.simple.VectorString image_ios = reader.GetRegisteredImageIOs();
                Console.Write("The supported image IOs are: ");
                for (int i=0; i<image_ios.Count; i++) {
                    Console.Write(image_ios[i] + " ");
                }
                Console.WriteLine("\n--------------------");

                // Another option is to just print the reader and see which
```

(continues on next page)

(continued from previous page)

```
                // IOs are supported
                Console.WriteLine(reader.ToString());
                Console.WriteLine("--------------------");

                // Force the use of a specific IO. If the IO doesn't support
                // reading the image type it will throw an exception.
                reader.SetImageIO("PNGImageIO");
                reader.SetFileName(args[0]);
                Image image = reader.Execute();
                Console.WriteLine("Read image: " + args[0]);

                VectorUInt32 size = image.GetSize();
                Console.WriteLine( "Image size: " + size[0] + " " + size[1] );

            } catch (Exception ex) {
                Console.WriteLine("Read failed: " + ex);
            }
        }
    }
}
```

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


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

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

  if ( argc < 2 )
    {
    std::cerr << "Usage: " << argv[0] << " image_file_name\n";
    return 1;
    }

  // Find out which image IOs are supported
  sitk::ImageFileReader reader;
  std::vector<std::string> image_ios = reader.GetRegisteredImageIOs();
  std::cout << "The supported image IOs are: ";
  std::vector<std::string>::iterator it;
  for (it=image_ios.begin(); it!=image_ios.end(); it++)
    {
    std::cout << (*it) << " ";
    }
  std::cout << std::endl;
  std::cout << "--------------------" << std::endl;
```

(continues on next page)

(continued from previous page)

```
  // Another option is to just print the reader and see which
  // IOs are supported
  std::cout << reader.ToString();
  std::cout << "--------------------" << std::endl;


  // Read the image file
  reader.SetImageIO ( "PNGImageIO" );
  reader.SetFileName ( std::string ( argv[1] ) );

  try
    {
    sitk::Image image = reader.Execute();
    std::cout << "Read image: " << argv[1] << std::endl;

    std::vector<unsigned int> size = image.GetSize();
    std::cout << "Image size: " << size[0] << " " << size[1] << std::endl;
    }
  catch (std::exception& e)
    {
    std::cout << "Read failed: " << e.what() << std::endl;
    }

  return 0;
  }
```

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

class ImageIOSelection {

  public static void main(String argv[]) {

    if ( argv.length < 1 ) {
      System.out.println("Usage: java ImageIOSelection image_file_name");
      return;
    }

    org.itk.simple.ImageFileReader reader = new org.itk.simple.ImageFileReader();

    // Find out which image IOs are supported
    VectorString image_ios;
    image_ios = reader.getRegisteredImageIOs();
    System.out.print("The supported image IOs are: ");
    System.out.println(image_ios);
    System.out.println("--------------------");

    // Another option is to just print the reader and see which
    // IOs are supported
    System.out.println(reader.toString());
    System.out.println("--------------------");

    reader.setImageIO("PNGImageIO");
```

(continues on next page)

(continued from previous page)

```
    reader.setFileName(argv[0]);

    try {
      Image image = reader.execute();
      System.out.println("Read image: " + argv[0]);

      VectorUInt32 size = image.getSize();
      System.out.println("Image size: " + size.get(0) + " " + size.get(1));
    } catch(Exception e) {
      System.out.println("Read failed: " + e);
    }

  }

}
```

```
local write = io.write

require "SimpleITK"

if #arg < 1 then
  print ( "Usage: ImageIOSelection image_file_name" )
  os.exit ( 1 )
end

-- Find out which image IOs are supported
reader = SimpleITK.ImageFileReader()
image_ios = reader:GetRegisteredImageIOs()
write("The supported image IOs are: ")
for i=0,image_ios:size()-1 do
  write(image_ios[i], " ")
end
write("\n")
print("--------------------")

-- Another option is to just print the reader and see which
-- IOs are supported
print(reader:ToString())
print("--------------------")

-- Force the use of a specific IO.
reader:SetImageIO("PNGImageIO")

-- Remember that Lua arrays are 1-based, and that arg does not contain the application␣
↪name!
reader:SetFileName ( arg[1] )

-- We need a function to pass to pcall, the Lua error handler.
-- So we embed the reader's Execute method in this function.
function read_execute( reader )
  return reader:Execute()
end
```

(continues on next page)

(continued from previous page)

```
-- Call the reader's execute method through Lua's pcall error handler
--
local status, rv = pcall(read_execute, reader)
if status then
  -- The reader succeeded
  write("Read image: ")
  write(arg[1])
  write("\n")

  image = rv
  size = image:GetSize();
  print("Image size:", size[0], size[1]);
else
  -- The reader failed
  write("Read failed: ")
  write(err)
  write("\n")
end
```

```
import sys
import SimpleITK as sitk

if len(sys.argv) < 2:
    print("Wrong number of arguments.", file=sys.stderr)
    print("Usage: " + __file__ + " image_file_name", file=sys.stderr)
    sys.exit(1)

# Find out which image IOs are supported
file_reader = sitk.ImageFileReader()
image_ios_tuple = file_reader.GetRegisteredImageIOs()
print("The supported image IOs are: " + str(image_ios_tuple))
print("-" * 20)

# Another option is to just print the reader and see which
# IOs are supported
print(file_reader)
print("-" * 20)

# Force the use of a specific IO. If the IO doesn't support
# reading the image type it will throw an exception.
file_reader.SetImageIO("PNGImageIO")
file_reader.SetFileName(sys.argv[1])
try:
    image = file_reader.Execute()
    print("Read image: " + sys.argv[1])

    size = image.GetSize()
    print("Image size:", size[0], size[1])
except Exception as err:
    print("Reading failed: ", err)
```

(continues on next page)

(continued from previous page)

```
    sys.exit(1)

sys.exit(0)
```

```
library(SimpleITK)

args <- commandArgs( TRUE )

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

# Find out which image IOs are supported
file_reader <- ImageFileReader()
image_ios <- file_reader$GetRegisteredImageIOs()
cat('The supported image IOs are: ', image_ios, '\n')
cat(rep('-',20),'\n', sep='')

# Another option is to just print the reader and see which
# IOs are supported
print(file_reader)
cat(rep('-',20),'\n', sep='')

# Force the use of a specific IO.
file_reader$SetImageIO('PNGImageIO')
file_reader$SetFileName(args[1])

# If the IO doesn't support reading the image type it
# will throw an exception.
image <- tryCatch(file_reader$Execute(),
                  warning = function(err) {
                      message(err)
                      quit(save="no", status=1)
                  }
                  )
cat('Read image:',args[1],'\n')

size <- image$GetSize()
cat ("Image size:", size[1], size[2])

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

```
require 'simpleitk'

if ARGV.length != 1 then
  puts "Usage: SimpleGaussian image_input_file";
  exit( 1 )
end

# Find out which image IOs are supported
```

(continues on next page)

(continued from previous page)

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

print "The supported image IOs are: "
for i in image_ios do
  print i, " "
end
puts "\n--------------------"

# Another option is to just print the reader and see which
# IOs are supported
puts reader.to_string()
puts "\n--------------------"

# Force the use of a specific IO. If the IO doesn't support
# reading the image type it will throw an exception.
reader.set_image_io("PNGImageIO")

begin
  image = reader.execute
  puts "Read image: " + ARGV[0]

  size = image.get_size
  puts "Image size: " + size[0] +  " " + size[1]
rescue StandardError => e
  puts "Read failed: " + e.message
end
```

```
if { $argc < 1 } {
    puts "Usage: ImageIOSelection image_input_file"
    exit 1
}

# Find out which image IOs are supported
ImageFileReader reader
set image_ios [ reader GetRegisteredImageIOs]
puts "The supported image IOs are: $image_ios"
puts "--------------------"

# Another option is to just print the reader and see which
# IOs are supported
puts [reader ToString]
puts "--------------------"

reader SetFileName [ lindex $argv 0 ]
reader SetImageIO "PNGImageIO"
if {[catch {set image [ reader Execute ]} errmsg]} {
  puts "Read failed: $errmsg"
}
set size [ $image GetSize ]
puts "Image size: $size"
```

(continues on next page)

(continued from previous page)

```
# Tcl requires explicit cleanup Cleanup
reader -delete
if {[info exists image]} {
  $image -delete
}
```

14.34 Raw Image Reading

14.34.1 Overview

In some, hopefully rare, cases you may only have the image information in raw pixel format, a binary blob file. For example the chest x-ray dataset from the Japanese Society of Radiological Technology (JSRT).

To read such an image you are required to know three things:

1. Image size (number of pixels per dimension).
2. Pixel type (e.g. sitkUInt16).
3. The byte order (or endianness) where little endian is most common.

If the byte order in unknown since it only has two options and we can try both. The one resulting in a visually correct image is the correct byte order.

As a SimpleITK image is a spatial object, we may also need to provide its origin, spacing, and direction cosine matrix. Without explicit information we can make reasonable assumptions that often work, as shown in the code below.

14.34.2 Code

Python

R

```
import argparse
import os
import tempfile

import SimpleITK as sitk


def read_raw(
    binary_file_name,
    image_size,
    sitk_pixel_type,
    image_spacing=None,
    image_origin=None,
    big_endian=False,
):
    """
    Read a raw binary scalar image.
```

(continues on next page)

(continued from previous page)

```
    Parameters
    ----------
    binary_file_name (str): Raw, binary image file content.
    image_size (tuple like): Size of image (e.g. [2048,2048])
    sitk_pixel_type (SimpleITK pixel type: Pixel type of data (e.g.
        sitk.sitkUInt16).
    image_spacing (tuple like): Optional image spacing, if none given assumed
        to be [1]*dim.
    image_origin (tuple like): Optional image origin, if none given assumed to
        be [0]*dim.
    big_endian (bool): Optional byte order indicator, if True big endian, else
        little endian.

    Returns
    -------
    SimpleITK image or None if fails.
    """

    pixel_dict = {
        sitk.sitkUInt8: "MET_UCHAR",
        sitk.sitkInt8: "MET_CHAR",
        sitk.sitkUInt16: "MET_USHORT",
        sitk.sitkInt16: "MET_SHORT",
        sitk.sitkUInt32: "MET_UINT",
        sitk.sitkInt32: "MET_INT",
        sitk.sitkUInt64: "MET_ULONG_LONG",
        sitk.sitkInt64: "MET_LONG_LONG",
        sitk.sitkFloat32: "MET_FLOAT",
        sitk.sitkFloat64: "MET_DOUBLE",
    }
    direction_cosine = [
        "1 0 0 1",
        "1 0 0 0 1 0 0 0 1",
        "1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1",
    ]
    dim = len(image_size)
    header = [
        "ObjectType = Image\n".encode(),
        (f"NDims = {dim}\n").encode(),
        ("DimSize = " + " ".join([str(v) for v in image_size]) + "\n").encode(),
        (
            "ElementSpacing = "
            + (
                " ".join([str(v) for v in image_spacing])
                if image_spacing
                else " ".join(["1"] * dim)
            )
            + "\n"
        ).encode(),
        (
            "Offset = "
```

(continues on next page)

(continued from previous page)

```
            + (
                " ".join([str(v) for v in image_origin])
                if image_origin
                else " ".join(["0"] * dim) + "\n"
            )
        ).encode(),
        ("TransformMatrix = " + direction_cosine[dim - 2] + "\n").encode(),
        ("ElementType = " + pixel_dict[sitk_pixel_type] + "\n").encode(),
        "BinaryData = True\n".encode(),
        ("BinaryDataByteOrderMSB = " + str(big_endian) + "\n").encode(),
        # ElementDataFile must be the last entry in the header
        ("ElementDataFile = " + os.path.abspath(binary_file_name) + "\n").encode(),
    ]
    fp = tempfile.NamedTemporaryFile(suffix=".mhd", delete=False)

    print(header)

    # Not using the tempfile with a context manager and auto-delete
    # because on windows we can't open the file a second time for ReadImage.
    fp.writelines(header)
    fp.close()
    img = sitk.ReadImage(fp.name)
    os.remove(fp.name)
    return img


parser = argparse.ArgumentParser()
parser.add_argument("raw_file_name", help="path to raw binary image file")
parser.add_argument(
    "out_file_name", help="output file name when image read as little endian"
)
parser.add_argument(
    "big_endian",
    type=lambda v: v.lower() in {"1", "true"},
    help="'false' for little ending or 'true'for big " "endian",
)
parser.add_argument(
    "sitk_pixel_type", help="SimpleITK pixel type (e.g. sitk.sitkUInt16)"
)
parser.add_argument("sz", nargs="+", help="image size, x,y,...", type=int)
args = parser.parse_args()

string_to_pixelType = {
    "sitkUInt8": sitk.sitkUInt8,
    "sitkInt8": sitk.sitkInt8,
    "sitkUInt16": sitk.sitkUInt16,
    "sitkInt16": sitk.sitkInt16,
    "sitkUInt32": sitk.sitkUInt32,
    "sitkInt32": sitk.sitkInt32,
    "sitkUInt64": sitk.sitkUInt64,
    "sitkInt64": sitk.sitkInt64,
    "sitkFloat32": sitk.sitkFloat32,
```

(continues on next page)

(continued from previous page)

```
    "sitkFloat64": sitk.sitkFloat64,
}

# Read the image using both big and little endian
image = read_raw(
    binary_file_name=args.raw_file_name,
    image_size=args.sz,
    sitk_pixel_type=string_to_pixelType[args.sitk_pixel_type],
    big_endian=args.big_endian,
)

sitk.WriteImage(image, args.out_file_name)

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

```
library(SimpleITK)

#' Read a raw binary scalar image.
#'
#' @param binary_file_name (str): Raw, binary image file content.
#' @param image_size (tuple like): Size of image (e.g. [2048,2048])
#' @param sitk_pixel_type (SimpleITK pixel type: Pixel type of data (e.g. sitk.
↪sitkUInt16).
#' @param image_spacing (tuple like): Optional image spacing, if none given assumed to be␣
↪[1]*dim.
#' @param image_origin (tuple like): Optional image origin, if none given assumed to be␣
↪[0]*dim.
#' @param big_endian (bool): Optional byte order indicator, if True big endian, else␣
↪little endian.
#'
#' @return SimpleITK image.
read_raw<-function(binary_file_name, image_size, sitk_pixel_type, image_spacing=NULL,
                   image_origin=NULL, big_endian=FALSE)
{
    PIXEL_TYPE <- list("sitkUInt8" = "MET_UCHAR",
                       "sitkInt8" = "MET_CHAR",
                       "sitkUInt16" = "MET_USHORT",
                       "sitkInt16" = "MET_SHORT",
                       "sitkUInt32" = "MET_UINT",
                       "sitkInt32" = "MET_INT",
                       "sitkUInt64" = "MET_ULONG_LONG",
                       "sitkInt64" = "MET_LONG_LONG",
                       "sitkFloat32" = "MET_FLOAT",
                       "sitkFloat64" = "MET_DOUBLE")
    direction_cosine <- list("1 0 0 1", "1 0 0 0 1 0 0 0 1", "1 0 0 0 0 1 0 0 0 0 1 0 0␣
↪0 0 1")
    dim <- length(image_size)
    header <- c("ObjectType = Image\n",
                paste0("NDims = ",toString(dim),"\n"),
                paste0("DimSize = ", paste(image_size, collapse = " "), "\n"),
                paste0("ElementSpacing = ", paste((if (!is.null(image_spacing)) image_
```

(continues on next page)

(continued from previous page)

```
↪spacing else rep(1,dim)), collapse = " "), "\n"),
                paste0("Offset = ", paste(if (!is.null(image_origin)) image_origin else␣
↪rep(0,dim), collapse = " "), "\n"),
                paste0("TransformMatrix = ", direction_cosine[dim-1], "\n"),
                paste0("ElementType = ", PIXEL_TYPE[sitk_pixel_type], "\n"),
                "BinaryData = True\n",
                paste0("BinaryDataByteOrderMSB = ", if (big_endian==FALSE) "False" else
↪"True", "\n"),
                paste0("ElementDataFile = ", normalizePath(binary_file_name), "\n")) #␣
↪ElementDataFile must be the last entry in the header

    fname <- tempfile(fileext = ".mhd") #generate temp file name
    writeLines(header, con=fname) #create and write to file
    img <- ReadImage(fname)
    file.remove(fname) #immediatly remove tempfile, otherwise it is automatically␣
↪removed at the end of the R session
    return(img)
}

args <- commandArgs( TRUE )

if (length(args) < 6) {
   write("Usage arguments: <raw_file_name> <out_file_name> <big_endian false|true> <sitk_
↪pixel_type> <image_size>", stderr())
   quit(save="no", status=1)
}

# Read the image using both big and little endian
image <- tryCatch(read_raw(binary_file_name=args[[1]], image_size=tail(args,-4), sitk_
↪pixel_type=args[[4]], big_endian=as.logical(args[3])),
                                warning = function(err) {
                                    message(err)
                                    quit(save="no", status=1)
                                }
                               )

WriteImage(image, args[[2]])

if (exists("SITK_NOSHOW", mode="environment")) {
   Show(image, "little endian")
}

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

14.35 Java Image Buffer

This example demonstrate efficient access to the Image's buffer in SimpleITK with Java. The SimpleITK Image class has the *java.nio.Buffer Image::getBufferAsBuffer(void) method. The `java.ni.Buffer* class has derived types for Java's signed integers, float, and double types. These derived buffer type are returned, when the SimpleITK Image's pixel type as scalar, vector, or complex corresponds. Additionally, the *UInt16* types are returned as a *CharBuffer*, other wise a *ByteBuffer* is returned. The returned buffer is the Image's buffer and no copy has occurred. However, SimpleITK's copy on write policies are followed, and *Image::MakeUnique* is called.

14.35.1 Overview

14.35.2 Code

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


class ImageGetSetBuffer {

   public static void main(String args[]) {

     if (args.length < 1) {
       System.out.format("Usage: ImageSetGetBuffer <input>");
       System.exit(-1);
     }

     Image image = SimpleITK.readImage(args[0]);

     // The loaded image may be of any type and number of components, so convert to␣
↪float,
     // so there is a known type.
     if (image.getNumberOfComponentsPerPixel() != 1) {
       image = SimpleITK.cast(image, PixelIDValueEnum.sitkVectorFloat32);
     } else {
       image = SimpleITK.cast(image, PixelIDValueEnum.sitkFloat32);
     }

     java.nio.FloatBuffer buffer = (java.nio.FloatBuffer)image.getBufferAsBuffer();

     System.out.println(buffer.toString());

     for (int i = 0; i < 10 && buffer.hasRemaining(); ++i) {
       System.out.println(buffer.get());
     }

     buffer.rewind();
     while (buffer.hasRemaining()) {
      buffer.put(0);
     }
```

(continues on next page)

(continued from previous page)

```
    }
  }
```

CHAPTER

FIFTEEN

RELEVANT RESOURCES

- Doxygen API Documentation
- Jupyter Notebooks in Python and R
- ITK Forum for Discussions and Questions

CHAPTER

SIXTEEN

HOW TO CITE

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

R. Beare, B. C. Lowekamp, Z. Yaniv, "Image Segmentation, Registration and Characterization in R with SimpleITK", *J Stat Software*, 86(8), https://doi.org/10.18637/jss.v086.i08, 2018.

Z. Yaniv, B. C. Lowekamp, H. J. Johnson, R. Beare, "SimpleITK Image-Analysis Notebooks: a Collaborative Environment for Education and Reproducible Research", *J Digit Imaging.*, https://doi.org/10.1007/s10278-017-0037-8, 31(3): 290-303, 2018.

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.