<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents</td>
<td>3</td>
</tr>
<tr>
<td>1.1 Overview</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Installing SimpleITK</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Build Instructions</td>
<td>5</td>
</tr>
<tr>
<td>1.4 User Guide</td>
<td>33</td>
</tr>
<tr>
<td>1.5 Gallery</td>
<td>94</td>
</tr>
<tr>
<td>1.6 Frequently Asked Questions</td>
<td>96</td>
</tr>
<tr>
<td>1.7 API Reference</td>
<td>102</td>
</tr>
</tbody>
</table>

2 Indices and tables 103
SimpleITK is an image analysis toolkit with a large number of components supporting general filtering operations, image segmentation and registration. It provides a simplified interface to ITK in a variety of languages.
1.1 Overview

SimpleITK is an image analysis toolkit with a large number of components supporting general filtering operations, image segmentation and registration. It is built on top of the Insight Segmentation and Registration Toolkit ITK with the intent of providing a simplified interface to ITK. SimpleITK itself is written in C++ but is available for a large number of programming languages. Currently these include:

- Python
- R
- Java
- C#
- Lua
- TCL
- Ruby

Wrapping of the C++ code is accomplished through SWIG, in principle, any language wrapped by SWIG should be applicable to SimpleITK.

Unlike ITK's support of n-dimensional spatio-temporal images, SimpleITK supports 2D, 3D and optionally 3D+time. The dimensionality refers to spatio-temporal dimensions, the voxels can be n-dimensional vectors.

SimpleITK is licensed under the Apache License in the same way as ITK.

1.2 Installing SimpleITK

Typically, you don’t need to build SimpleITK to use it. You can simply download the binaries and get started right away!

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

1.2.1 Python

Virtual environments are highly recommended. They allow you to elegantly deal with package compatibility issues.
Standard Python installation

From the shell, execute

```
pip install simpleitk
```

You can always manually download the wheels for your operating system and Python version directly from sourceforge. You might need to update your pip using `pip install -U pip`.

The pip package manager should automatically find the correct package for MS Windows and GNU Linux if your version is supported. For Apple OS X you need to manually specify:

```
pip install https://sourceforge.net/projects/simpleitk/files/SimpleITK/0.10.0/Python/SimpleITK-0.10.0-cp27-cp27m-macosx_10_6_intel.whl
```

Conda-based distributions (Anaconda, Miniconda)

From the shell/command prompt, execute

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

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

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

1.2.2 C Sharp (C#)

Binaries for select C# platform can be found on SimpleITK’s SourceForge page. Installing the library should only involve importing the unzipped files into you C# environment.

The files have the following naming convention:

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

Following files are currently available:

```
SimpleITK-0.10.0-CSharp-win32-x86.zip
SimpleITK-0.10.0-CSharp-win64-x64.zip
```

A guide describing how to set up a C# Visual Studio project with SimpleITK can be found [here](#). For platforms other than Windows, you might have to build manually as described in Building SimpleITK.

1.2.3 Java

Binaries for select Java platforms can be found on SimpleITK’s SourceForge page.

Following files are currently available:

```
SimpleITK-0.10.0-Java-win64.zip
SimpleITK-0.10.0-Java-win32.zip
```

Detailed instructions are available at [Visual guide to SimpleITK with Java](#). For platforms other than Windows, you might have to build manually as described in Building SimpleITK.
1.2.4 Nightly binaries

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

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

1.3 Build Instructions

1.3.1 Building SimpleITK

In many cases, you do not need to build SimpleITK because of the pre-built binaries available (see Installing SimpleITK). However there are several reasons you might prefer to build SimpleITK from source:

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

Prerequisites

To build SimpleITK you need:

• A recent version of CMake >= 3.3 with SSL support for https.
• A supported compiler.
• To use the latest developmental version, source code can be downloaded with git >= 1.65
  – Git is required if building SimpleITK using “SuperBuild” (see below) to automatically download the matching version of ITK and SWIG
  – Windows users may prefer msysGit
It is recommended to have numpy installed when testing Python bindings.

Recipes/ Formulas/ Short Cuts

For some environments we have short cuts, scripts, for automated building of SimpleITK.

Mac

On the Mac, with the Homebrew package manager, a SimpleITK formula is available: https://github.com/Homebrew/homebrew-science/blob/master/simpleitk.rb for multiple language wrappings.

```bash
# available install options can be listed using
brew options homebrew/science/simpleitk
# for example, you can install with java and lua support as follows
brew install homebrew/science/simpleitk --with-java --with-lua
```

Anaconda Python

For the Anaconda Python distribution: The recipe for the SimpleITK build is in the official conda-recipe repository.

```bash
# get the recipe
git clone https://github.com/conda/conda-recipes.git
cd conda-recipes/python
# build with your default Python (likely 2.7)
conda build simpleitk
# or build with 3.x
CONDA_PY=34 conda build simpleitk
```

R

For the R language you can use the devtools installer (currently only for Linux/OS X): This script will download, build and install SimpleITK into your R environment. Please modify the number of processors to use based on your system (in our example below we set it to six)

```r
devtools::install_github("SimpleITK/SimpleITKRInstaller", args=c('--configure-vars="MAKEJ=6"'))
```

Lua

For the Lua language with the Luarocks module deployment system, a SimpleITK rockspec is available on github.

```bash
# get the rock spec
git clone https://github.com/SimpleITK/SimpleITKLuaRock.git
cd SimpleITKLuaRock
# build and install the SimpleITK Lua module
luarocks install simpleitk-1.0-0.rockspec
```
Building

Obtaining Source Code

There are two options to obtain the SimpleITK source code:

1. Download a released version from the SimpleITK SourceForge page
2. Download the latest development version using git: `git clone http://itk.org/SimpleITK.git`

Building using SuperBuild

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

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

The SuperBuild will automatically download and build the matching version of ITK and SWIG needed to compile SimpleITK. Additionally, it will set recommended compilation flags to minimize the size of the library and enable support for large libraries. This is the recommended way to build SimpleITK and is easiest.

**Note:** 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 subdirectory.

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/or correct the advanced cmake variables to the language specific executable, libraries and include directories. For example if you have multiple Python installations ensure that all related Python variable refer to the same versions.

Then use your make utility or your cmake chosen build utility to build SimpleITK. SimpleITK takes a while to build.

Building manually

This is **not** the recommended way of building SimpleITK, but it can be useful if you want to use a system version of ITK and/or SWIG, or if you do not want to (or can not) use git (due to firewall, etc).

1. Setup the prerequisites as described above (i.e. CMake and supported compiler)
2. **Install the matching version of SWIG >= 3.0.5** e.g. Windows users may install swigwin-3.0.5
3. Download the SimpleITK source code from the SourceForge page
4. Download the matching version of ITK e.g. SimpleITK 0.10.0 uses ITK 4.10.0 (tag v4.10.0)
5. **Configure ITK using CMake** `BUILD_EXAMPLES=OFF, BUILD_TESTING=OFF, BUILD_SHARED_LIBS=OFF, ITK_USE_REVIEW=ON`
6. **Build ITK** Be sure to note the build settings e.g. Release x64
7. **Configure SimpleITK using CMake** Set ITK_DIR to the location of the ITK build location from the previous steps
8. **Build SimpleITK**  Be sure to configure the build settings exactly the same as ITK e.g. Release x64 and CXX_FLAGS

**Testing**

After compilation the prudent thing to do is to test SimpleITK to ensure your build is stable and suitable for use and installation. 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
```

If all tests fail, verify that you have the testing data in your source tree (the reason for the “–recursive” flag in the git command) AND that you have added the correct path to your * _LIBRARY_PATH.

**Python installation**

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

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

Alternatively, a Python virtual environment can be created and the distribution installed there. If you build the “dist” target a Python Wheel file (.whl) will be created in the “Wrapping/Python/dist” directory.

**R installation**

To install a built R package:

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

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

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

1. **Create a local directory where you will install your R packages**  mkdir my_R_libs
2. **Add an environment variable to your .bashrc**  export R_LIBS="/path_to/my_R_libs"
3. source your .bashrc and check the R library path, in an R shell >.libPaths()
4. **install**

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

**Fiji (Fiji is Just ImageJ)**

SimpleITK has a built in function, `itk::simple::Show()`, which can be used for viewing images in an interactive session. Currently, this function by default `Show` invokes Fiji then ImageJ to display images. ImageJ was chosen because it can handle all the image types that SimpleITK supports, even 3D vector images with n components.

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

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

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

**iPython and Jupyter**

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

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

1.3.2 **Visual guide to building on Linux**

This guide gives detailed instructions for building SimpleITK on Linux. It is written for beginners getting started with SimpleITK. There are examples of how to develop and execute simple programs in C Sharp and Lua.

---

**On this page**

- Why Linux?
- Get Linux
- Install build tools
- Get SimpleITK source code
- Build SimpleITK
- Use SimpleITK
  - A simple C# program
  - A simple lua program

**Why Linux?**

- Linux is freely available
• It has all the required tools
• Did I mention it’s free?

Get Linux

The first step is to install a Linux distribution. Some popular ones are:

• Ubuntu
• Linux Mint
• Debian

And here is a comparison of those distributions.

There are many online tutorials explaining how to install your chosen Linux distribution:

• Ubuntu
  – Ubuntu Installation guide
  – Learn How to Install Ubuntu Linux in 5 Minutes (youtube)
  – The Ubuntu Installation Guide (with pictures)

• Mint
  – Linux Mint User Guide (English)
  – Install Linux Mint on your Windows computer (howtogeek)
  – How to Install Linux Mint (youtube)

• Debian
  – Debian 7 Installation (youtube)
  – Install Debian

If you are a Windows user, you may consider running Linux on a virtual machine. Some popular virtual machine environments are:

• VirtualBox
• VMWare

If you are a Mac OS X user, you can also run Linux in a virtual machine. Two virtual machine environments for OS X are:

• Parallels
• VMWare Fusion

Again, there are heaps of tutorials:

• Installing Ubuntu inside Windows using VirtualBox
• How to Install Ubuntu on VirtualBox
• Install Mint 16 on VirtualBox
• How to Install Linux Mint in Virtualbox (youtube)

This guide uses Debian 7, but the steps are very similar for other Linux distributions.
Install build tools

The next step is to install the required build tools.

Open a terminal window (Application Menu > Terminal Emulator) and run the following command:

```
sudo apt-get install cmake cmake-curses-gui gcc g++ git
```

Confirm that you want to install the packages (press “y”), then wait for the installation to complete.

Alternatively, you could manually select each software package from the Synaptic Package Manager (Application Menu > Settings > Synaptic Package Manager).

By default building SimpleITK produces the SimpleITK C++ libraries and the SimpleITK Lua interpreter. It also supports bindings for other languages. To build this support, additional packages need to be installed. The following table shows the supported language bindings and the corresponding command to install the additional packages required for each language.

<table>
<thead>
<tr>
<th>Programming Language</th>
<th>Command to install the build tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>C#</td>
<td>sudo apt-get install monodevelop</td>
</tr>
<tr>
<td>Java</td>
<td>sudo apt-get install eclipse</td>
</tr>
<tr>
<td>R</td>
<td>sudo apt-get install r-base r-base-dev</td>
</tr>
<tr>
<td>Ruby</td>
<td>sudo apt-get install ruby</td>
</tr>
<tr>
<td>Python</td>
<td>sudo apt-get install python python-dev</td>
</tr>
<tr>
<td>Tcl</td>
<td>sudo apt-get install tcl tcl-dev tk tk-dev</td>
</tr>
<tr>
<td>All languages</td>
<td>sudo apt-get install monodevelop eclipse r-base r-base-dev ruby python python-dev tcl tcl-dev tk tk-dev</td>
</tr>
</tbody>
</table>

1.3. Build Instructions
Fig. 1.2: In the terminal, use `apt-get` to install the build tools.
Fig. 1.3: Build tools could also be installed using the software manager
Get SimpleITK source code

The next step is to get the SimpleITK source code using git. Decide where you want to put the source code. I’m putting mine in my home directory:

```bash
cd ~
```

Now download the SimpleITK source code, by entering the following command in the Terminal:

```bash
git clone --recursive http://itk.org/SimpleITK.git
```

![Fig. 1.4: Get the SimpleITK source code using git](image)

Now change to the SimpleITK directory:

```bash
cd SimpleITK
```

Build SimpleITK

The next step is to start building.

The recommended way to build is via the so-called “super build”. The build directory should not be inside the source tree. I put the build directory in the same directory as the source tree.

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

The SuperBuild generates make files which takes care of downloading and building ITK, SWIG, and Lua, as well as SimpleITK.
To start the (long) build process, type:

```
make
```

On my test system, a 4 core virtual machine with 16 GB of RAM, the build took just over an hour.

After the build is finished, you need to add SimpleITK to your `LD_LIBRARY_PATH`:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/SimpleITK-build/lib
```

You can now (optionally) check whether the build was successful:

```
cd ~/SimpleITK-build
cmake
```

All (or at least most) of the tests should pass.

## Use SimpleITK

SimpleITK is available to a variety of languages. In this section we give simple example programs demonstrating the SimpleITK API in C Sharp and Lua.

### A simple C# program

This sub-section will describe how to create a simple C# application using SimpleITK from MonoDevelop. To start launch the C# development environment, MonoDevelop (Application Menu > Development > MonoDevelop).

![Fig. 1.5: Open MonoDevelop](image-url)
Then create our new Solution (Select File > New > Solution).

Select a C# console project (C# > Console Project). Enter a suitable name e.g. “sitk” and uncheck “Create directory for directory”. Select “Forward” and then “OK”.

![Create Console Project](image)

In the Solution explorer, right-click “Selection” and select “Edit References...”.

Select the “.Net Assembly” tab, navigate to “~/SimpleITK-build/SimpleITK-build/Wrapping/CSharpBinaries”, select “SimpleITKCSharpManaged.dll”, click “Add”, and then close the window by selecting “OK”. This will copy “SimpleITKCSharpManaged.dll” to your build directory e.g. “bin/Debug” or “bin/Release”.

You must also manually copy “libSimpleITKCSharpNative.so” to your build directories:

```
mkdir ~/sitk/bin/Debug
mkdir ~/sitk/bin/Release
cp ~/SimpleITK-build/SimpleITK-build/Wrapping/CSharpBinaries/libSimpleITKCSharpNative.so ~/sitk/bin/Debug
cp ~/SimpleITK-build/SimpleITK-build/Wrapping/CSharpBinaries/libSimpleITKCSharpNative.so ~/sitk/bin/Release
```

SimpleITK has now been added as a project reference.

The following short program creates an image of a Gaussian blob, generates a derivative image from the Gaussian, scales and windows the derivative’s intensities, converts the result to 8-bit unsigned ints, and writes out a PNG file:

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

namespace itk.simple
{
    class MainClass
    {
```
1.3. Build Instructions
Fig. 1.8: Add SimpleITKSharpManaged.dll
```csharp
public static void Main (string[] args)
{
    var size = new VectorUInt32 (new uint[] { 128, 128 });
    var sigma = new VectorDouble (new Double[] { 32.0, 32.0 });
    var center = new VectorDouble (new Double[] { 64.0, 64.0 });

    var gauss = sitk.GaussianSource (PixelIDValueEnum.sitkFloat32, size, sigma, center);
    var deriv = sitk.Cast (128.0 + 24.0 * sitk.Derivative (gauss), PixelIDValueEnum.sitkUInt8);

    sitk.WriteImage (deriv, "gauss-deriv-test.png");
}
```

Fig. 1.9: A simple C# program

Note that in the example, the derivative image’s intensities are scaled mathematically to illustrate SimpleITK’s over-loading of the mathematically operators. The image intensities could also be scaled using SimpleITK’s RescaleIntensity function.

To build the project press “F8” or select Build > Build All from the menu.

To debug the project, add a breakpoint at a desired location and press “F5”.

The Gaussian Derivative image below shows the results of the C# example program.
A simple lua program

Lua is a fast, portable, lightweight scripting language that is included with the SimpleITK source code. Because the entire source code for Lua is less than 600kb, it takes very little space relative to large projects such as SimpleITK. That makes Lua very popular as an embedded scripting language.

In this SimpleITK/Lua example we show how to use a text editor to produce a SimpleITK example in Lua and execute the program.

By default, Debian with the Xfce user interface, comes with Mousepad (Application Menu > Accessories > Mousepad), a simple text editor. Other possible editors include gedit with Gnome or kedit with KDE.

The following is a simple Lua example similar to the C# example in the previous section. This program creates an
image of a Gaussian blob, computes a derivative image of the Gaussian, rescales the floating point image to 0-255, casts it to a unsigned char image, and writes the result to a PNG file.

```lua
local sitk = {}
sitk = SimpleITK

size = sitk.VectorUInt32();
size:push_back(128);
size:push_back(128);

sigma = sitk.VectorDouble();
sigma:push_back(32.0);
sigma:push_back(32.0);

center = sitk.VectorDouble();
center:push_back(64.0);
center:push_back(64.0);

gauss = sitk.GaussianSource (sitk.sitkFloat32, size, sigma, center);
deriv = sitk.Derivative(gauss);
result = sitk.RescaleIntensity(deriv, 0, 255.0)
result = sitk.Cast(result, sitk.sitkUInt8)
sitk.WriteImage(result, "sitk-lua-test.png");
```

Fig. 1.12: Simple Lua Program

The script is slightly different than the C# example in that the RescaleIntensity filter is used. In C# mathematical operators are overloaded for SimpleITK images. This is not the case for Lua, so mathematical operations on SimpleITK images are a bit more complicated. Therefore I chose to use a built in filter.

To try out the program, copy the code and paste it into Mousepad. Then Save it as “DerivativeExample.lua” and enter the following command in a Terminal window.

```bash
~/SimpleITK-build/SimpleITK-build/bin/SimpleITKLua DerivativeExample.lua
```

The Lua Derivative image below shows the output of the our SimpleITK Lua example. The result is similar to, although not the same as the C Sharp produced image. They are different because the image intensities are not scaled in the same manner.

### 1.3.3 Visual guide to SimpleITK with 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.
Eclipse Setup

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

The first step in developing for SimpleITK’s Java bindings is to create a new project in Eclipse. Simply 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.

GaussianExample

Here is our first class Code/GaussianExample.java:

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

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

public class GaussianExample {

    /**
     * @param args
     */
    public static void main(String[] args) {
```
if (args.length < 2) {
    System.err.println("Usage: 'Gaussian <input> <output>'");
    System.exit(1);
}
System.out.println("Starting to blur " + args[0]);
// Grab a file
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 19-21. These errors occurs because we have not told Eclipse where to find SimpleITK’s jar file.

Adding SimpleITK to the Build Path

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

![Fig. 1.14: Change Build Path](image)
Configure the Build Path for our Eclipse project

In the Properties dialog, click on Add External JARs... and 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.

**Source attachment**

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.

**Javadoc location**

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.

**Native library location**

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.

**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 the example.

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`.

### 1.3.4 Visual guide to SimpleITK with CSharp

In this guide we will show how to setup a C# project in Microsoft Visual Studio 2012 which uses the available built binaries for SimpleITK. The same steps and options are needed for the other versions of Visual Studio.

**On this page**

- Download
- Adding C# SimpleITK to a Project
- Selecting Architecture
- Adding Managed Library
- Adding Native Library
- Building an Example

**Download**

Binary downloads are readily available for C# for Microsoft Visual Studio. They are available on SourceForge.

**Download the correct binary for you architecture you are going to target.** C# for SimpleITK has two components: Native and Managed. The native code contains the SimpleITK C++ library and is compiled for the particular architecture. There is the “win32” for the Intel x86 32-bit architecture, and the “win64” for the Intel x64 architecture. The correct architecture needs to be chosen.

**Unzip the downloaded zip file into your “Documents” folder.** Inside you will find two “dll” files: “SimpleITKCSharpManaged.dll” and “SimpleITKCSharpNative.dll”, as well as some documentation files.
Fig. 1.16: Download an unzip the CSharp distribution.
Adding C# SimpleITK to a Project

We will start off with a new C# console solution. This is created by selecting “File->New->Project”, then selecting under Templates “Visual C#” and then choosing the “Console Application”.

![Create C# Console Application](image.png)

**Fig. 1.17: Create a C# Console Application.**

Selecting 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 Manager...”.

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”.
Fig. 1.18: The default configuration and platform in the toolbar.
Fig. 1.19: The Configuration Manager.
Adding 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 “SimpleITKSharpManaged.dll” from the binary download, then click OK to add.

![Adding managed library as reference.](image)

Adding 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 “SimpleITKSharpNative.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 “SimpleITKSharpNative.dll”, and select “Properties”.
Then for “Build Action”, choose “Content”, and “Copy to OutputDirectory” choose “Copy always”.

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 from the file menu “BUILD->Build Solution” can be selected.

If all the steps were followed correctly you should now have an executable which can be run from the command line or from within Visual Studio with the appropriate arguments provided.
1.3. Build Instructions

Fig. 1.21: Adding Native Library

Fig. 1.22: Configuring properties of native library.
Fig. 1.23: A successful build of the example.
1.4 User Guide

Here is a guide on SimpleITK.

1.4.1 Basics of Image

Image Basics

Image class is basic container for image data in SimpleITK. It can hold 2 or 3 dimensional images and pixel can be either be a scalar or a vector. A RGB image for example, is a 2 dimensional image with 3 component vector pixel.

Contents

- Construction
- Pixel Types
- Images and Physical Space
  - Transform voxels to physical space
- Accessing Pixels
  - Arrays/Tensors
  - Slicing
- Image Operations

Construction

There are a variety of ways to create an image. All images’ initial value is well defined as zero

\[
\text{Image(width, height, pixelID)} \\
\text{Image(width, height, depth, pixelID)} \\
\text{Image(sizeVector, pixelID)} \\
\text{Image(sizeVector, pixelID, numberOfComponents)}
\]

For example, in python you can create a RGB image of size 128x64 as

```python
import SimpleITK as sitk
image_RGB = sitk.Image([128, 64], sitk.sitkVectorUInt8, 3)
```

Pixel Types

The pixel type is represented as an enumerated type `PixelIDValueEnum`. The following is a table of the enumerated list.
<table>
<thead>
<tr>
<th>EnumValue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sitkUInt8</td>
<td>Unsigned 8 bit integer</td>
</tr>
<tr>
<td>sitkInt8</td>
<td>Signed 8 bit integer</td>
</tr>
<tr>
<td>sitkUInt16</td>
<td>Unsigned 16 bit integer</td>
</tr>
<tr>
<td>sitkInt16</td>
<td>Signed 16 bit integer</td>
</tr>
<tr>
<td>sitkUInt32</td>
<td>Unsigned 32 bit integer</td>
</tr>
<tr>
<td>sitkInt32</td>
<td>Signed 32 bit integer</td>
</tr>
<tr>
<td>sitkUInt64</td>
<td>Unsigned 64 bit integer</td>
</tr>
<tr>
<td>sitkInt64</td>
<td>Signed 64 bit integer</td>
</tr>
<tr>
<td>sitkFloat32</td>
<td>32 bit float</td>
</tr>
<tr>
<td>sitkFloat64</td>
<td>64 bit float</td>
</tr>
<tr>
<td>sitkComplexFloat32</td>
<td>Complex number of 32 bit float</td>
</tr>
<tr>
<td>sitkComplexFloat64</td>
<td>Complex number of 64 bit float</td>
</tr>
<tr>
<td>sitkVectorUInt8</td>
<td>Multi-component of unsigned 8 bit integer</td>
</tr>
<tr>
<td>sitkVectorInt8</td>
<td>Multi-component of signed 8 bit integer</td>
</tr>
<tr>
<td>sitkVectorUInt16</td>
<td>Multi-component of unsigned 16 bit integer</td>
</tr>
<tr>
<td>sitkVectorInt16</td>
<td>Multi-component of signed 16 bit integer</td>
</tr>
<tr>
<td>sitkVectorUInt32</td>
<td>Multi-component of unsigned 32 bit integer</td>
</tr>
<tr>
<td>sitkVectorInt32</td>
<td>Multi-component of signed 32 bit integer</td>
</tr>
<tr>
<td>sitkVectorUInt64</td>
<td>Multi-component of unsigned 64 bit integer</td>
</tr>
<tr>
<td>sitkVectorInt64</td>
<td>Multi-component of signed 64 bit integer</td>
</tr>
<tr>
<td>sitkVectorFloat32</td>
<td>Multi-component of 32 bit float</td>
</tr>
<tr>
<td>sitkVectorFloat64</td>
<td>Multi-component of 64 bit float</td>
</tr>
<tr>
<td>sitkLabelUInt8</td>
<td>RLE label of unsigned 8 bit integers</td>
</tr>
<tr>
<td>sitkLabelUInt16</td>
<td>RLE label of unsigned 16 bit integers</td>
</tr>
<tr>
<td>sitkLabelUInt32</td>
<td>RLE label of unsigned 32 bit integers</td>
</tr>
<tr>
<td>sitkLabelUInt64</td>
<td>RLE label of unsigned 64 bit integers</td>
</tr>
</tbody>
</table>

There is also `sitkUnknown`, which is used for undefined or erroneous pixel ID's. It has a value of -1. The 64-bit integer types are not available on all distributions. When not available the value is `sitkUnknown`.

You can get the type of pixel using `Image::GetPixelID()` and its string representation using `Image::GetPixelIDTypeAsString()`.

### Images and Physical Space

The unique feature of SimpleITK (derived from ITK) as a toolkit for image manipulation and analysis is that it views images as physical objects occupying a bounded region in physical space. In addition images can have different spacing between pixels along each axis, and the axes are not necessarily orthogonal. The following figure illustrates these concepts.
Each `Image` has following properties:

**Pixel type**  Type of pixel/voxel. Refer to table above. In case of a vector image, number of components per pixel can be greater than 1. This is fixed on creation.

Can get pixel type through `Image::GetPixelID()`. Number of components per pixel can be found by `Image::GetNumberOfComponentsPerPixel()`.

**Size**  Number of pixels/voxels in each dimension. This quantity implicitly defines the image dimension. This is also fixed on creation.

Size of the image can be found by `Image::GetSize()` and dimension by `Image::GetDimension()`.

**Origin**  Coordinates of the pixel/voxel with index (0,0,0) in physical units (i.e. mm). Default is zero i.e. origin of physical space.

`Image::GetOrigin()` and `Image::SetOrigin()` can be used to get and set origin respectively.

**Spacing**  Distance between adjacent pixels/voxels in each dimension given in physical units. Default is one i.e. (1 mm, 1 mm, 1 mm).

`Image::GetSpacing()` and `Image::SetSpacing()` can be used to get and set spacing respectively.

**Direction Matrix**  Mapping/rotation between direction of the pixel/voxel axes and physical directions. Default is identity matrix. The matrix is passed as a 1D array in row-major form.

`Image::GetDirection()` and `Image::SetDirection()` can be used to get and set direction matrix respectively.

---

**Note:**  All the transformations like rotation or affine transform are done on the underlying physical space. You can think of image of a view of this physical space.

Let’s illustrate this in python:

```python
from downloaddata import fetch_data as fdata
image = sitk.ReadImage(
    fdata("nac-hncma-atlas2013-Slicer4Version/Data/A1_grayT1.nrrd"))
```
import SimpleITK as sitk

t = sitk.ReadImage("nac-hncma-atlas2013-Slicer4Version/Data/A1_grayT1.nrrd")

print("Pixel Type {}".format(t.GetPixelID()))
print("Size {}".format(t.GetSize()))
print("Origin {}".format(t.GetOrigin()))
print("Spacing {}".format(t.GetSpacing()))
print("Direction {}".format(t.GetDirection()))

Out:
Fetching nac-hncma-atlas2013-Slicer4Version/Data/A1_grayT1.nrrd
Pixel Type 8
Size (288, 320, 208)
Origin (-77.625, -107.625, 119.625)
Spacing (0.75, 0.75, 0.75)
Direction (0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, -1.0, 0.0)

Transform voxels to physical space

Following equation can be used to convert voxel coordinates/indices to physical coordinates:

\[ x = D.S.v + o \]

where \( x \) is coordinate of the voxel in physical space, \( v \) is voxel index, \( o \) is origin, \( D \) is direction matrix and \( S \) is \text{diag} (spacing).

These functions can be directly used to transform between voxel and physical space:

- \text{Image::TransformContinuousIndexToPhysicalPoint()} 
- \text{Image::TransformIndexToPhysicalPoint()} 
- \text{Image::TransformPhysicalPointToContinuousIndex()} 
- \text{Image::TransformPhysicalPointToIndex()}

Let's try these out in python

pt = (4.0, 2.0, 0.0)
idx = (30, 87, 45)

print("Point {} is voxel {}.".format(pt, t.TransformPhysicalPointToIndex(pt)))
print("Voxel {} is point {}.".format(idx, t.TransformIndexToPhysicalPoint(idx)))

Out:
Point (4.0, 2.0, 0.0) is voxel (146, 160, 109).
Voxel (30, 87, 45) is point (-43.875, -85.125, 54.375).

Accessing Pixels

You can get the pixel values using one of \text{Image::GetPixelAsInt8()}, \text{Image::GetPixelAsUInt32()}, \text{Image::GetPixelAsFloat()} \text{Image::GetPixelAsDouble()} etc.

Similarly, you can set the pixel values using \text{Image::SetPixelAsInt8()}, \text{Image::SetPixelAsUInt32()}, \text{Image::SetPixelAsFloat()} \text{Image::SetPixelAsDouble()} etc.

In dynamic type languages like python and lua, \text{GetPixel} and \text{SetPixel} are available. In python, you can also use pythonic indexing to get and set pixel values.
For example

```python
x, y, z = 10, 15, 20
# These two mean the same
print("Get pixels: {:.3f}, {:.3f}".format(image.GetPixel(x, y, z),
                   image[x, y, z]))

# These two mean the same
image.SetPixel(x, y, z, 1.2)
image[x, y, z] = 1.2
```

Out:

Get pixels: 8.638, 8.638

**Arrays/Tensors**  If you have `numpy` library installed in python, you can convert images to arrays and vice versa using `GetArrayFromImage()` and `GetImageFromArray()`. Similarly, if you have `torch` installed, you can use `GetTensorFromImage()` and `GetImageFromTensor()`. Numpy and torch are numerical computational libraries for python and lua respectively.

**Note:** While converting from tensor/array to Image, remember to set the image’s origin, spacing, and possibly direction cosine matrix. The default values may not match the physical dimensions of your image.

**Note:** Image access is in x,y,z order (image.GetPixel(x,y,z) or image[x,y,z]) with zero based indexing. Note that this is different from numpy or torch indexing which uses z, y, x order.

In numpy for example:

```python
sitkimg = sitk.Image(10, 20, 30, sitk.sitkFloat32)
sitkimg[1, 2, 3] = 1.5
npimg = sitk.GetArrayFromImage(sitkimg)
print("Size from SimpleITK: {}, numpy: {}".format(
       sitkimg.GetSize(), npimg.shape))
print("npimg[1, 2, 3] = {}, npimg[3, 2, 1] = {}".format(
       npimg[1, 2, 3], npimg[3, 2, 1]))
```

Out:

Size from SimpleITK: (10, 20, 30), numpy: (30, 20, 10)
npimg[1, 2, 3] = 0.0, npimg[3, 2, 1] = 1.5

In torch, indexing starts with 1:

```python
sitk = require 'SimpleITK'
sitkimg = sitk.Image(10, 20, 30, sitk.sitkFloat32)
sitkimg:SetPixel({1, 2, 3}, 1.5)
thing = sitk.GetTensorFromImage(sitkimg)
sitksize = sitkimg:GetSize()
thesize = thing:size()
print(sitksize[0], sitksize[1], sitksize[2]) -- prints 10 20 30
print(thesize[1], thesize[2], thesize[3]) -- prints 30 20 10
print(thing[{2, 3, 4}], thing[{3, 2, 1}],
     thing[{4, 3, 2}]) -- prints 0 0 1.5
```

1.4. User Guide
Slicing  Slice() can be used to slice the image and a dimension can be collapsed with Extract(). In python, you can use pythonic slicing without having to use these:

```python
import matplotlib.pyplot as plt

logo = sitk.ReadImage(fdata('SimpleITK.jpg'))
# Brute force subsampling
logo_subsampled = logo[::2, ::2]
# Get the sub-image containing the word Simple
simple = logo[0:115, :]
# Get the sub-image containing the word Simple and flip it
simple_flipped = logo[115:0:-1, :]

# display results
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(nrows=2, ncols=2,
                                           sharex=True,
                                           sharey=True)
ax1.imshow(sitk.GetArrayFromImage(logo))
ax1.axis('off')
ax1.set_title("logo")
ax2.imshow(sitk.GetArrayFromImage(logo_subsampled))
ax2.axis('off')
ax2.set_title("logo_subsampled")
ax3.imshow(sitk.GetArrayFromImage(simple))
ax3.axis('off')
ax3.set_title("simple")
ax4.imshow(sitk.GetArrayFromImage(simple_flipped))
ax4.axis('off')
ax4.set_title("simple_flipped")
fig.tight_layout()
plt.show()
```
Image Operations

SimpleITK supports basic arithmetic operations between images, **taking into account their physical space**

```python
img1 = sitk.Image(24, 24, sitk.sitkUInt8)
img2 = sitk.Image(img1.GetSize(), sitk.sitkUInt8)
img1[0, 0] = 10
img2[0, 0] = 30
img3 = img1 + img2
img4 = img1 + 72
print("Test add: {} {}", format(img3[0, 0], img4[0, 0]))
img2.SetOrigin([3, 5])
# Following raises error as the images are not in the # same physical space
try:
    img5 = img1 + img2
except Exception as e:
    print(e)
```

Out:
Exception thrown in SimpleITK Add: /tmp/SimpleITK-build/ITK-prefix/include/ITK-4.10/itkImageToImageFilter.hxx:250:
  itk::ERROR: AddImageFilter(0x49bd350): Inputs do not occupy the same physical space!
  InputImage Origin: [0.0000000e+00, 0.0000000e+00], InputImage_1 Origin: [3.0000000e+00, 5.0000000e+00]
  Tolerance: 1.0000000e-06

Following are some of the pixel-wise operations that can be used with image, image pairs or image, scalar pairs:

- Addition +
- Subtraction -
- Multiplication *
- Division /
- Modulo %
- Power **

Lot more operations like sine, cosine, exponentation etc. are also available.

**Total running time of the script:** (0 minutes 11.451 seconds)

Download Python source code: plot_image.py
Download Jupyter notebook: plot_image.ipynb

Generated by Sphinx-Gallery

**myshow.py**

Functions created in guides for visualization in other guides. If you want to run the code from other guides, please download this file (by clicking Download Python source code at the bottom of the page) and add it to your python path.

```python
import matplotlib.pyplot as plt
import SimpleITK as sitk

def myshow(img, title=None, margin=0.05, dpi=80):
nda = sitk.GetArrayFromImage(img)
spacing = img.GetSpacing()

if nda.ndim == 3:
    # fastest dim, either component or x
    c = nda.shape[-1]

    # the the number of components is 3 or 4 consider it an RGB image
    if c not in (3, 4):
        nda = nda[nda.shape[0] // 2, :, :]

elif nda.ndim == 4:
    c = nda.shape[-1]

    if c not in (3, 4):
        raise RuntimeError("Unable to show 3D-vector Image")

    # take a z-slice
    nda = nda[nda.shape[0] // 2, :, :]
```

```
```python
xsize = nda.shape[1]
ysize = nda.shape[0]

# Make a figure big enough to accommodate an axis of xpixels by ypixels
# as well as the ticklabels, etc...
figsize = (1 + margin) * xsize / dpi, (1 + margin) * ysize / dpi

plt.figure(figsize=figsize, dpi=dpi, tight_layout=True)
ax = plt.gca()

extent = (0, xsize * spacing[0], ysize * spacing[1], 0)
t = ax.imshow(nda, extent=extent, interpolation=None)

if nda.ndim == 2:
    t.set_cmap("gray")
if (title):
    plt.title(title)
plt.show()


def myshow3d(img, xslices=[], yslices=[], zslices=[], title=None, margin=0.05,
              dpi=80):
    img_xslices = [img[s, :, :] for s in xslices]
    img_yslices = [img[:, s, :] for s in yslices]
    img_zslices = [img[:, :, s] for s in zslices]

    maxlen = max(len(img_xslices), len(img_yslices), len(img_zslices))

    img_null = sitk.Image([0, 0], img.GetPixelID(),
                          img.GetNumberOfComponentsPerPixel())

    img_slices = []
    d = 0

    if len(img_xslices):
        img_slices += img_xslices + [img_null] * (maxlen - len(img_xslices))
        d += 1

    if len(img_yslices):
        img_slices += img_yslices + [img_null] * (maxlen - len(img_yslices))
        d += 1

    if len(img_zslices):
        img_slices += img_zslices + [img_null] * (maxlen - len(img_zslices))
        d += 1

    if maxlen != 0:
        if img.GetNumberOfComponentsPerPixel() == 1:
            img = sitk.Tile(img_slices, [maxlen, d])

        else:
            img_comps = []
            for i in range(0, img.GetNumberOfComponentsPerPixel()):
                img_slices_c = {sitk.VectorIndexSelectionCast(s, i)
```
for s in img_slices:
    img_comps.append(sitk.Tile(img_slices_c, [maxlen, d]))
img = sitk.Compose(img_comps)

myshow(img, title, margin, dpi)

Total running time of the script: (0 minutes 0.000 seconds)
Download Python source code: myshow.py
Download Jupyter notebook: myshow.ipynb
Generated by Sphinx-Gallery

downloaddata.py

Download this file and Data folder and add them to your python path to run other examples.

Since we do not want to store large binary data files in our Git repository, we fetch_data_all from a network resource.

The data we download is described in a json file. The file format is a dictionary of dictionaries. The top level key is the file name. The returned dictionary contains an md5 checksum and possibly a url and boolean flag indicating the file is part of an archive. The md5 checksum is mandatory. When the optional url is given, we attempt to download from that url, otherwise we attempt to download from the list of MIDAS servers returned by the get_midas_servers() function. Files that are contained in archives are identified by the archive flag.

Example json file contents:

```json
{
    "SimpleITK.jpg": { "md5sum": "2685660c4f50c5929516127ae9e5b1a" },
    "POPI/meta/00.mhd": { "md5sum": "3bfc3c92e18a8e6e8494482c44654f3d" },
    "CIRS057A_MR_CT_DICOM/readme.txt": {
        "md5sum": "d92e97e6fe6520cb5b1a50b96eb9eb96", "archive": "true"
    }
}
```

Notes: 1. The file we download can be inside an archive. In this case, the md5 checksum is that of the archive.

2. For the md5 verification to work we need to store archives on MIDAS and cannot use its on-the-fly archive download mechanism (this mechanism allows users to download “directories/communities” as a single zip archive). The issue is that every time the archive is created its md5 changes. It is likely MIDAS is also encoding the archive’s modification/creation time as part of the md5.

Another issue is that when downloading from this type of url (e.g. http://midas3.kitware.com/midas/download/folder/11610/ipythonNotebookData.zip) the returned data does not have a “Content-Length” field in the header. The current implementation will throw an exception.

```python
import hashlib
import sys
import os
import json
import errno
import warnings

# http://stackoverflow.com/questions/2028517/python-urlllib2-progress-hook

def url_download_report(bytes_so_far, url_download_size, total_size):
    percent = float(bytes_so_far) / total_size
```
percent = round(percent * 100, 2)
if bytes_so_far > url_download_size:
    # Note that the carriage return is at the beginning of the
    # string and not the end. This accommodates usage in
    # IPython usage notebooks. Otherwise the string is not
    # displayed in the output.
    sys.stdout.write('Downloaded %d of %d bytes (%0.2f%%)
' % (bytes_so_far, total_size, percent))
    sys.stdout.flush()
if bytes_so_far >= total_size:
    sys.stdout.write('Downloaded %d of %d bytes (%0.2f%%)
' % (bytes_so_far, total_size, percent))
    sys.stdout.flush()

def url_download_read(url, outputfile, url_download_size=8192 * 2, report_hook=None):
    # Use the urllib2 to download the data. The Requests package, highly
    # recommended for this task, doesn't support the file scheme so we opted
    # for urllib2 which does.
    try:
        # Python 3
        from urllib.request import urlopen, URLError, HTTPError
    except ImportError:
        from urllib2 import urlopen, URLError, HTTPError
    from xml.dom import minidom

    # Open the url
    try:
        url_response = urlopen(url)
    except HTTPError as e:
        return "HTTP Error: {0} {1}\n".format(e.code, url)
    except URLError as e:
        return "URL Error: {0} {1}\n".format(e.reason, url)
    # MIDAS is a service and therefore will not generate the expected URLError
    # when given a nonexistent url. It does return an error message in xml.
    # When the response is xml then we have an error, we read the whole message
    # and return the 'msg' attribute associated with the 'err' tag.
    # The URLError above is not superfluous as it will occur when the url
    # refers to a non existent file ('file://nonexistent_file_name') or url
    # which is not a service ('http://nonexistent_address').
    try:
        # Python 3
        content_type = url_response.info().get("Content-Type")
    except AttributeError:
        content_type = url_response.info().getheader("Content-Type")
    # MIDAS error message in json format
    if content_type == "text/html; charset=UTF-8":
        doc = json.loads(url_response.read().decode("utf-8"))
        if doc["stat"]=='fail':
            return doc["message"] + url
    # MIDAS error message in xml format
    if content_type == "text/xml":
        doc = minidom.parseString(url_response.read())
        if doc.getElementsByTagName("err")[0].getAttribute("msg") + ':' + url
        # We download all content types - the assumption is that the md5sum ensures
        # that what we received is the expected data.
try:
    # Python 3
    content_length = url_response.info().get("Content-Length")
except AttributeError:
    content_length = url_response.info().getheader("Content-Length")
total_size = content_length.strip()
total_size = int(total_size)
bytes_so_far = 0
with open(outputfile, "wb") as local_file:
    while 1:
        try:
            url_download = url_response.read(url_download_size)
            bytes_so_far += len(url_download)
            if not url_download:
                break
            local_file.write(url_download)
        except HTTPError as e:
            return "HTTP Error: {0} {1}\n\n".format(e.code, url)
        except URLError as e:
            return "URL Error: {0} {1}\n\n".format(e.reason, url)
        if report_hook:
            report_hook(bytes_so_far, url_download_size, total_size)
return "Downloaded Successfully"


def mkdir_p(path):
    try:
        os.makedirs(path)
    except OSError as exc:
        # Python >2.5
        if exc.errno == errno.EEXIST and os.path.isdir(path):
            pass
        else:
            raise


def deprecated(func):
    """This is a decorator which can be used to mark functions
    as deprecated. It will result in a warning being emmitted
    when the function is used.""
    def new_func(*args, **kwargs):
        warnings.simplefilter('always', DeprecationWarning) # turn off filter
        warnings.warn("Call to deprecated function {0} \n\n".format(func.__name__), category=DeprecationWarning)
        warnings.simplefilter('default', DeprecationWarning) # reset filter
        return func(*args, **kwargs)
    new_func.__name__ = func.__name__
    new_func.__doc__ = func.__doc__
    new_func.__dict__.update(func.__dict__)
    return new_func

def get_midas_servers():
    import os
    midas_servers = list()
    if 'ExternalData_OBJECTSTORES' in os.environ.keys():
local_object_stores = os.environ['ExternalData_OBJECT_STORES']
for local_object_store in local_object_stores.split(';;'):
    midas_servers.append("file://(0)/MD5/%(hash)".format(local_object_store))
midas_servers.extend([
    # Data published by MIDAS
    # Data published by developers using git-gerrit-push.
    "http://www.itk.org/files/ExternalData/%(algo)/%(hash)",
    # Mirror supported by the Slicer community.
    # Insight journal data server
])
return midas_servers

def output_hash_is_valid(known_md5sum, output_file):
    md5 = hashlib.md5()
    if not os.path.exists(output_file):
        return False
    with open(output_file, 'rb') as fp:
        for url_download in iter(lambda: fp.read(128 * md5.block_size), b''):
            md5.update(url_download)
        retrieved_md5sum = md5.hexdigest()
    return retrieved_md5sum == known_md5sum

def fetch_data_one(onefilename, output_directory, manifest_file, verify=True, force=False):
    import tarfile, zipfile

    with open(manifest_file, 'r') as fp:
        manifest = json.load(fp)

    assert onefilename in manifest, "ERROR: {0} does not exist in {1}".format(onefilename, manifest_file)

    sys.stdout.write("Fetching {0}\n".format(onefilename))
    output_file = os.path.realpath(os.path.join(output_directory, onefilename))
    data_dictionary = manifest[onefilename]
    md5sum = data_dictionary['md5sum']
    all_urls = []
    if "url" in data_dictionary:
        all_urls.append(data_dictionary["url"])
    else:
        for url_base in get_midas_servers():
            all_urls.append(url_base.replace("%(hash)", md5sum).replace("%(algo)", "md5"))

    new_download = False

    for url in all_urls:
        # Only download if force is true or the file does not exist.
        if force or not os.path.exists(output_file):
            mkdir_p(os.path.dirname(output_file))
            url_download_read(url, output_file, report_hook=url_download_report)
        url_download_read(url, output_file, report_hook=None)

        # Check if a file was downloaded and has the correct hash
        if output_hash_is_valid(md5sum, output_file):
            new_download = True
            # Stop looking once found

    return new_download
elif os.path.exists(output_file):
    error_msg = "File " + output_file
    error_msg += " has incorrect hash value, " + md5sum + " was expected."
    raise Exception(error_msg)

# Did not find the file anywhere.
if not os.path.exists(output_file):
    error_msg = "File " + os.path.basename(output_file) + " could not be found in any of the following locations:
    error_msg += ", ".join(all_urls)
    raise Exception(error_msg)

if not new_download and verify:
    # If the file was part of an archive then we don't verify it. These
    # files are only verified on download
    if (not "archive" in data_dictionary) and (not output_hash_is_valid(md5sum, output_file)):
        # Attempt to download if md5sum is incorrect.
        fetch_data_one(onefilename, output_directory, manifest_file, verify, force=True)

    # If the file is in an archive, unpack it.
    if tarfile.is_tarfile(output_file) or zipfile.is_zipfile(output_file):
        tmp_output_file = output_file + ".tmp"
        os.rename(output_file, tmp_output_file)
        if tarfile.is_tarfile(tmp_output_file):
            archive = tarfile.open(tmp_output_file)
        elif zipfile.is_zipfile(tmp_output_file):
            archive = zipfile.ZipFile(tmp_output_file, 'r')
        archive.extractall(os.path.dirname(tmp_output_file))
        archive.close()
        os.remove(tmp_output_file)

        return output_file

@deprecated
def fetch_midas_data_one(onefilename, output_directory, manifest_file, verify=True, force=False):
    return fetch_data_one(onefilename, output_directory, manifest_file, verify, force)

def fetch_data_all(output_directory, manifest_file, verify=True):
    with open(manifest_file, 'r') as fp:
        manifest = json.load(fp)
    for filename in manifest:
        fetch_data_one(filename, output_directory, manifest_file, verify, force=False)

@deprecated
def fetch_midas_data_all(output_directory, manifest_file, verify=True):
    return fetch_data_all(output_directory, manifest_file, verify)

def fetch_data(cache_file_name, verify=False, cache_directory_name="Data"):
    ""
    fetch_data is a simplified interface that requires
    relative pathing with a manifest.json file located in the
    same cache_directory_name name.
By default the cache_directory_name is "Data" relative to the current python script. An absolute path can also be given.

```python
if not os.path.isabs(cache_directory_name):
    cache_root_directory_name = os.path.dirname(__file__)
    cache_directory_name = os.path.join(cache_root_directory_name, cache_directory_name)
    cache_manifest_file = os.path.join(cache_directory_name, 'manifest.json')
assert os.path.exists(cache_manifest_file), "ERROR, {0} does not exist".format(cache_manifest_file)
return fetch_data_one(cache_file_name, cache_directory_name, cache_manifest_file, verify=verify)
```

@deprecated
def fetch_midas_data(cache_file_name, verify=False, cache_directory_name="Data"):
    return fetch_data(cache_file_name, verify, cache_directory_name)

if __name__ == '__main__':

    if len(sys.argv) < 3:
        print('Usage: ' + sys.argv[0] + ' output_directory manifest.json')
        sys.exit(1)
    output_directory = sys.argv[1]
    if not os.path.exists(output_directory):
        os.makedirs(output_directory)
    manifest = sys.argv[2]
    fetch_data_all(output_directory, manifest)

Total running time of the script: 0.000 seconds

Download Python source code: downloaddata.py
Download Jupyter notebook: downloaddata.ipynb
Generated by Sphinx-Gallery

1.4.2 Segmentation

Image segmentation filters process an image to partition it into (hopefully) meaningful regions. The output is commonly an image of integers where each integer can represent an object. The value 0 is commonly used for the background, and 1 (sometimes 255) for a foreground object.

Thresholding

Thresholding is the most basic form of segmentation. It simply labels the pixels of an image based on the intensity range without respect to geometry or connectivity.
Load Image

```python
img_T1 = sitk.ReadImage(
    fdata("nac-hncma-atlas2013-Slicer4Version/Data/A1_grayT1.nrrd"))

# To visualize the labels image in RGB needs a image with 0-255 range
img_T1_255 = sitk.Cast(sitk.RescaleIntensity(img_T1), sitk.sitkUInt8)

size = img_T1.GetSize()
myshow3d(img_T1_255, zslices=range(50, size[2] - 50, 20))
```

Out:

```
Fetching nac-hncma-atlas2013-Slicer4Version/Data/A1_grayT1.nrrd
```

Threshold

Let’s pick a threshold 200 for thresholding.

```python
seg = img_T1 > 200
myshow(sitk.LabelOverlay(img_T1_255, seg), "Basic Thresholding")
```
You can also use a upper and lower threshold.

```python
seg = sitk.BinaryThreshold(img_T1,
    lowerThreshold=100, upperThreshold=400,
    insideValue=1, outsideValue=0)
myshow(sitk.LabelOverlay(img_T1_255, seg), "Binary Thresholding")
```
Otsu Thresholding

ITK has a number of histogram based automatic thresholding filters including Huang, MaximumEntropy, Triangle, and the popular Otsu’s method. These methods create a histogram then use a heuristic to determine a threshold value.

Otsu Thresholding for example, assumes that the image contains two classes of pixels following bi-modal histogram and calculates the optimum threshold separating these two classes.

```python
otsu_filter = sitk.OtsuThresholdImageFilter()
otsu_filter.SetInsideValue(0)
otsu_filter.SetOutsideValue(1)
seg = otsu_filter.Execute(img_T1)
myshow(sitk.LabelOverlay(img_T1_255, seg), "Otsu Thresholding")

print("Computed Threshold: {}".format(otsu_filter.GetThreshold()))
```
Region Growing Segmentation

Thresholding is the most basic form of segmentation. The first step of improvement upon the naive thresholding is a class of algorithms called region growing.

The common theme in this class of algorithms is that a voxel’s neighbor is considered to be in the same class if its intensities are similar to the current voxel. The definition of similar is what varies. Initial set of voxel are called seed points. These initial seed points are usually manually selected.

We illustrate the use of three variants of this family of algorithms:

- `ConnectedThreshold`
- `ConfidenceConnected`
- `VectorConfidenceConnected`

We will illustrate the usage of these three filters using a cranial MRI scan (T1 and T2) and attempt to segment one of the ventricles.
```
import SimpleITK as sitk
from myshow import myshow, myshow3d
from downloaddata import fetch_data as fdata

Load Images

img_T1 = sitk.ReadImage(
    fdata("nac-hncma-atlas2013-Slicer4Version/Data/A1_grayT1.nrrd"))

# To visualize the labels image in RGB needs a image with 0-255 range
img_T1_255 = sitk.Cast(sitk.RescaleIntensity(img_T1), sitk.sitkUInt8)

size = img_T1.GetSize()
myshow3d(img_T1_255, zslices=range(50, size[2] - 50, 20), title='T1')
```

Out:

```
Fetching nac-hncma-atlas2013-Slicer4Version/Data/A1_grayT1.nrrd
```

Seed selection

Earlier we used 3D Slicer to determine that index: (132,142,96) was a good seed for the left lateral ventricle.

```
seed = (132, 142, 96)

seg = sitk.Image(img_T1.GetSize(), sitk.sitkUInt8)
seg.CopyInformation(img_T1)
seg[seed] = 1

seg = sitk.BinaryDilate(seg, 3)
myshow3d(sitk.LabelOverlay(img_T1_255, seg),
    xslices=range(132, 133), yslices=range(142, 143),
    zslices=range(96, 97), title="Initial Seed")
```
Region Growing

Let’s start with `ConnectedThreshold`.

`ConnectedThreshold` Here, voxel’s neighbor is considered to be in the same class if the neighboring voxel’s intensity is within explicitly specified thresholds.

We start by using explicitly specified thresholds, you should modify these (lower/upper) to see the effects on the resulting segmentation.

```python
seg_con = sitk.ConnectedThreshold(img_T1, seedList=[seed],
                                  lower=100, upper=190)

myshow3d(sitk.LabelOverlay(img_T1_255, seg_con),
         xslices=range(132, 133), yslices=range(142, 143),
         zslices=range(96, 97), title="Connected Threshold")
```
Connected Threshold
**ConfidenceConnected** Unlike in ConnectedThreshold, you need not select the bounds in ConfidenceConnected filter. Bounds are implicitly specified as $\mu \pm c\sigma$, where $\mu$ is the mean intensity of the seed points, $\sigma$ their standard deviation and $c$ a user specified constant.

This algorithm has some flexibility which you should familiarize yourself with:

- The `multiplier` parameter is the constant $c$ from the formula above.
- You can specify a region around each seed point `initialNeighborhoodRadius` from which the statistics are estimated, see what happens when you set it to zero.
- The `numberOfIterations` allows you to rerun the algorithm. In the first run the bounds are defined by the seed voxels you specified, in the following iterations $\mu$ and $\sigma$ are estimated from the segmented points and the region growing is updated accordingly.

```python
seg_conf = sitk.ConfidenceConnected(img_T1, seedList=[seed],
                                    numberOfIterations=1,
                                    multiplier=2.5,
                                    initialNeighborhoodRadius=1,
                                    replaceValue=1)

myshow3d(sitk.LabelOverlay(img_T1_255, seg_conf),
          xslices=range(132, 133), yslices=range(142, 143),
          zslices=range(96, 97), title="ConfidenceConnected")
```
**VectorConfidenceConnected**  A generalization of the previous approach to vector valued images, for instance multi-spectral images or multi-parametric MRI. Here, the bounds for neighboring voxel’s intensity vector is implicitly specified bounds using the Mahalanobis distance $\sqrt{(x - \mu)^T \Sigma^{-1} (x - \mu)} < c$, where $\mu$ is the mean of the vectors at the seed points, $\Sigma$ is the covariance matrix and $c$ is a user specified constant.

Let’s load a T2 image from the same person and combine it with the T1 image to create a vector image and apply the algorithm on it.

```python
img_T2 = sitk.ReadImage(  
    fdata("nac-hncma-atlas2013-Slicer4Version/Data/A1_grayT2.nrrd"))
img_T2_255 = sitk.Cast(sitk.RescaleIntensity(img_T2), sitk.sitkUInt8)

img_multi = sitk.Compose(img_T1, img_T2)
seg_vec = sitk.VectorConfidenceConnected(img_multi, seedList=[seed],
    numberOfIterations=1,
    multiplier=4,
    initialNeighborhoodRadius=1)

myshow3d(sitk.LabelOverlay(img_T2_255, seg_vec),
    xslices=range(132, 133), yslices=range(142, 143),
    zslices=range(96, 97), title="VectorConfidenceConnected")
```
Clean up, clean up

Use of low level segmentation algorithms such as region growing is often followed by a clean up step. In this step we fill holes and remove small connected components. Both of these operations are achieved by using binary morphological operations, opening (`BinaryMorphologicalOpening`) to remove small connected components and closing (`BinaryMorphologicalClosing`) to fill holes.

SimpleITK supports several shapes for the structuring elements (kernels) including:

- `sitkAnnulus`
- `sitkBall`
- `sitkBox`
- `sitkCross`

The size of the kernel can be specified as a scalar (same for all dimensions) or as a vector of values, size per dimension.

The following code illustrates the results of such a clean up, using closing to remove holes in the original segmentation.

```python
vectorRadius = (1, 1, 1)
kernal = sitk.sitkBall
seg_clean = sitk.BinaryMorphologicalClosing(seg_vec, vectorRadius, kernel)

myshow3d(sitk.LabelOverlay(img_T1_255, seg_clean),
        xslices=range(132, 133), yslices=range(142, 143),
        zslices=range(96, 97), title="Cleaned up segmentation")
```
1.4.3 Transforms

Image Interpolation

We demonstrate the different interpolators available in SimpleITK available for image resampling. Their effect is demonstrated on the Marschner-Lobb image.

```python
import SimpleITK as sitk
import math
from myshow import myshow

Marschner-Lobb image

Add formula for it here.

def marschner_lobb(size=40, alpha=0.25, f_M=6.0):
    img = sitk.PhysicalPointSource(sitk.sitkVectorFloat32, [size] * 3, [-1] * 3, [2.0 / size] * 3)
    imgx = sitk.VectorIndexSelectionCast(img, 0)
    imgy = sitk.VectorIndexSelectionCast(img, 1)
    imgz = sitk.VectorIndexSelectionCast(img, 2)
    r = sitk.Sqrt(imgx**2 + imgy**2)
    pr = sitk.Cos((2.0 * math.pi * f_M) * sitk.Cos((math.pi / 2.0) * r))
    return (1.0 - sitk.Sin((math.pi / 2.0) * imgz) + alpha * (1.0 + pr)) / (2.0 * (1.0 + alpha))

myshow(marschner_lobb(200), title='Marschner-Lobb Image')
```
Interpolations

Let's use `Expand` to rescale image by a factor of 5 in all dimensions.

Compare these interpolations with the image above.

```python
ml = marschner_lobb(40)
ml = ml[:, :, ml.GetSize()[-1] // 2]

myshow(sitk.Expand(ml, [5] * 3, sitk.sitkNearestNeighbor),
       title="nearest neighbor")
myshow(sitk.Expand(ml, [5] * 3, sitk.sitkLinear),
       title="linear")
myshow(sitk.Expand(ml, [5] * 3, sitk.sitkBSpline),
       title="b-spline")
myshow(sitk.Expand(ml, [5] * 3, sitk.sitkGaussian),
       title="Gaussian")
myshow(sitk.Expand(ml, [5] * 3, sitk.sitkHammingWindowedSinc),
       title="Hamming windowed sinc")
myshow(sitk.Expand(ml, [5] * 3, sitk.sitkBlackmanWindowedSinc),
       title="Blackman windowed sinc")
myshow(sitk.Expand(ml, [5] * 3, sitk.sitkCosineWindowedSinc),
       title="Cosine windowed sinc")
myshow(sitk.Expand(ml, [5] * 3, sitk.sitkWelchWindowedSinc),
       title="Welch windowed sinc")
myshow(sitk.Expand(ml, [5] * 3, sitk.sitkLanczosWindowedSinc),
       title="Lanczos windowed sinc")
```
Total running time of the script: ( 0 minutes 1.715 seconds)
Download Python source code: plot_interpolation.py
Download Jupyter notebook: plot_interpolation.ipynb
Generated by Sphinx-Gallery
More details about Transforms

This guide introduces the transformation types supported by SimpleITK and illustrates how to “promote” transformations from a lower to higher parameter space (e.g. 3D translation to 3D rigid).

On this Page

- Points in SimpleITK
  - Utility functions
    - Print Point
    - Generate Random Cloud of points
    - Compute Registration Error
    - Check difference between two transformations
- Global Transformations
  - TranslationTransform
  - Euler2DTransform
  - VersorTransform
  - Translation to Rigid [3D]

SimpleITK Transformation Types

<table>
<thead>
<tr>
<th>Transformation Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Translation-Transform</td>
<td>2D or 3D, translation</td>
</tr>
<tr>
<td>VersorTransform</td>
<td>3D, rotation represented by a versor</td>
</tr>
<tr>
<td>Versor-Rigid3DTransform</td>
<td>3D, rigid transformation with rotation represented by a versor</td>
</tr>
<tr>
<td>Euler2DTransform</td>
<td>2D, rigid transformation with rotation represented by a Euler angle</td>
</tr>
<tr>
<td>Euler3DTransform</td>
<td>3D, rigid transformation with rotation represented by Euler angles</td>
</tr>
<tr>
<td>Similarity2DTransform</td>
<td>2D, composition of isotropic scaling and rigid transformation with rotation represented by a Euler angle</td>
</tr>
<tr>
<td>Similarity3DTransform</td>
<td>3D, composition of isotropic scaling and rigid transformation with rotation represented by a versor</td>
</tr>
<tr>
<td>ScaleTransform</td>
<td>2D or 3D, anisotropic scaling</td>
</tr>
<tr>
<td>ScaleVersor3DTransform</td>
<td>3D, rigid transformation and anisotropic scale is added to the rotation matrix part (not composed as one would expect)</td>
</tr>
<tr>
<td>ScaleSkewVersor3DTransform</td>
<td>3D, rigid transformation with anisotropic scale and skew matrices added to the rotation matrix part (not composed as one would expect)</td>
</tr>
<tr>
<td>AffineTransform</td>
<td>2D or 3D, affine transformation.</td>
</tr>
<tr>
<td>BSplineTransform</td>
<td>2D or 3D, deformable transformation represented by a sparse regular grid of control points.</td>
</tr>
<tr>
<td>Displacement-FieldTransform</td>
<td>2D or 3D, deformable transformation represented as a dense regular grid of vectors.</td>
</tr>
<tr>
<td>Transform</td>
<td>A generic transformation. Can represent any of the SimpleITK transformations, and a composite transformation (stack of transformations concatenated via composition, last added, first applied).</td>
</tr>
</tbody>
</table>

```python
import SimpleITK as sitk
import numpy as np
import matplotlib.pyplot as plt
```
Points in SimpleITK

Utility functions  Let’s write some functions that deal with point data in a uniform manner.

**Print Point**  Format a point for printing, based on specified precision with trailing zeros.

```python
def point2str(point, precision=1):
    format_str = '.{0}f'.format(precision)
    return ' '.join(format(c, format_str) for c in point)
```

**Generate Random Cloud of points**  Generate random (uniform withing bounds) nD point cloud. Dimension is based on the number of pairs in the bounds input.

```python
def uniform_random_points(bounds, num_points):
    internal_bounds = [sorted(b) for b in bounds]
    # Generate rows for each of the coordinates according to
    # the given bounds, stack into an array, and split into
    # a list of points.
    mat = np.vstack([np.random.uniform(b[0], b[1], num_points)
                     for b in internal_bounds])
    return list(mat[:len(bounds)].T)
```

**Compute Registration Error**  Distances between points transformed by the given transformation and their location in another coordinate system. When the points are only used to evaluate registration accuracy (not used in the registration) this is the target registration error (TRE).

```python
def target_registration_errors(tx, point_list, reference_point_list):
    return [np.linalg.norm(np.array(tx.TransformPoint(p)) - np.array(p_ref))
            for p, p_ref in zip(point_list, reference_point_list)]
```

**Check difference between two transformations**  Check whether two transformations are “equivalent” in an arbitrary spatial region either 3D or 2D, \([x=(-10,10), y=(-100,100), z=(-1000,1000)]\). This is just a sanity check, as we are just looking at the effect of the transformations on a random set of points in the region.

```python
def print_transformation_differences(tx1, tx2):
    if tx1.GetDimension() == 2 and tx2.GetDimension() == 2:
        bounds = [(-10, 10), (-100, 100)]
    elif tx1.GetDimension() == 3 and tx2.GetDimension() == 3:
        bounds = [(-10, 10), (-100, 100), (-1000, 1000)]
    else:
        raise ValueError('Transformation dimensions mismatch, '
                         'or unsupported transformation dimensionality')

    num_points = 10
    point_list = uniform_random_points(bounds, num_points)
    tx1_point_list = [tx1.TransformPoint(p) for p in point_list]
    differences = target_registration_errors(tx2, point_list, tx1_point_list)

    print('{} - {} :	minDifference: {:.2f} maxDifference: {:.2f}'.format(
        tx1.GetName(), tx2.GetName(), min(differences), max(differences)))
```

In SimpleITK points can be represented by any vector-like data type. In Python these include Tuple, Numpy array, and List. In general Python will treat these data types differently, as illustrated by the print function below.

SimpleITK points represented by vector-like data structures.
point_tuple = (9.0, 10.531, 11.8341)
point_np_array = np.array([9.0, 10.531, 11.8341])
point_list = [9.0, 10.531, 11.8341]

print('tuple : {}'.format(point_tuple))
print('numpy array : {}'.format(point_np_array))
print('list: {}'.format(point_list))

# Uniform printing with specified precision.
precision = 2
print(point2str(point_tuple, precision))
print(point2str(point_np_array, precision))
print(point2str(point_list, precision))

Out:

tuple : (9.0, 10.531, 11.8341)
numpy array : [  9.  10.531  11.8341]
list: [9.0, 10.531, 11.8341]
9.00 10.53 11.83
9.00 10.53 11.83
9.00 10.53 11.83

**Global Transformations**

All global transformations except translation are of the form:

\[ T(x) = A(x - c) + t + c \]

In ITK speak (when printing your transformation):

- Matrix: the matrix \( A \)
- Center: the point \( c \)
- Translation: the vector \( t \)
- Offset: \( t + cA c \)

**TranslationTransform**  A 3D translation. Note that you need to specify the dimensionality, as the sitk TranslationTransform represents both 2D and 3D translations.

dimension = 3
offset = (1, 2, 3)  # offset can be any vector-like data
translation = sitk.TranslationTransform(dimension, offset)
print(translation)

Out:

```
itk::simple::Transform
TranslationTransform (0x3ece350)
  RTTI typeinfo: itk::TranslationTransform<double, 3u>
  Reference Count: 1
  Modified Time: 11580
  Debug: Off
  Object Name:
  Observers:
    none
  Offset: [1, 2, 3]
```
Transform a point and use the inverse transformation to get the original back.

```python
point = [10, 11, 12]
transformed_point = translation.TransformPoint(point)
translation_inverse = translation.GetInverse()
print('original point: {}
''transformed point: {}
''back to original: {}').format(
    point2str(point),
    point2str(transformed_point),
    point2str(translation_inverse.TransformPoint(transformed_point))))
```

Out:
```
original point: 10.0 11.0 12.0
transformed point: 11.0 13.0 15.0
back to original: 10.0 11.0 12.0
```

**Euler2DTransform**  This transform applies a rigid transformation is 2D space. The transform is specified as a rotation around arbitrary center and is followed by a translation.

```python
rotation2D = sitk.Euler2DTransform()
rotation2D.SetTranslation((7.2, 8.4))
rotation2D.SetAngle(np.pi / 2)
point = [10, 11]
print('original point: {}
''transformed point: {}').format(
    point2str(point),
    point2str(rotation2D.TransformPoint(point))))
```

Out:
```
original point: 10.0 11.0
transformed point: -3.8 18.4
```

Change the center of rotation so that it coincides with the point we want to transform, why is this a unique configuration?

```python
rotation2D.SetCenter(point)
print('original point: {}
''transformed point: {}').format(
    point2str(point),
    point2str(rotation2D.TransformPoint(point))))
```

Out:
```
original point: 10.0 11.0
transformed point: 17.2 19.4
```

**VersorTransform**  Rotation only, parametrized by Versor (vector part of unit quaternion), quaternion defined by rotation of theta around axis n:

\[ q = [n \sin(\theta/2), \cos(\theta/2)] \]

180 degree rotation around z axis

68 Chapter 1. Contents
# Using a versor
rotation1 = sitk.VersorTransform([0, 0, 1, 0])

# Using axis-angle:
rotation2 = sitk.VersorTransform((0, 0, 1), np.pi)

# Using a matrix:
rotation3 = sitk.VersorTransform()
rotation3.SetMatrix([-1, 0, 0, 0, -1, 0, 0, 0, 1])

point = (10, 100, 1000)
p1 = rotation1.TransformPoint(point)
p2 = rotation2.TransformPoint(point)
p3 = rotation3.TransformPoint(point)

print(('Points after transformation:
    p1 = {}
    p2 = {}
    p3 = {}
').format(p1, p2, p3))

Out:

Points after transformation:
p1 = (-10.0, -100.0, 1000.0)
p2 = (-10.000000000000012, -100.0, 1000.0)
p3 = (-10.0, -100.0, 1000.0)

We applied the “same” transformation to the same point, so why are the results slightly different for the second initialization method?

This is where theory meets practice. Using the axis-angle initialization method involves trigonometric functions which on a fixed precision machine lead to these slight differences. In many cases this is not an issue, but it is something to remember. From here on we will sweep it under the rug (printing with a more reasonable precision).

Translation to Rigid [3D]

Todo

Fill all the notebook. It’s too boring and requires too much work.

Total running time of the script: ( 0 minutes 0.008 seconds)
Download Python source code: plot_moretransforms.py
Download Jupyter notebook: plot_moretransforms.ipynb
Generated by Sphinx-Gallery

Transforms and Resampling

This tutorial explains how to apply transforms to images.

SimpleITK provides you with variety of transforms like Translation, Affine (which includes rotations, shear, scaling)

It is important to keep in mind that these transforms are applied on the points in the physical space, not the image coordinates itself.

A few conventions:
- Points are represented by vector-like data types: Tuple, Numpy array, List.
SimpleITK Documentation, Release 0.11

- Matrices are represented by vector-like data types in row major order.

On this Page

- Creating and Manipulating Transforms
  - Identity Transform
  - Translation Transform
  - Affine Transform
- Applying Transforms to Images
  - Translation
  - Scaling
  - Rotation
  - Shearing
- Composite Transform

# License: CC-BY
# sphinx_gallery_thumbnail_number = 7

```python
import SimpleITK as sitk
import numpy as np
from myshow import myshow
```

Creating and Manipulating Transforms

A number of different spatial transforms are available in SimpleITK.

**Identity Transform**  The simplest is the Identity Transform. This transform simply returns input points unaltered.

```python
dimension = 2

print('Identity Transform')
identity = sitk.Transform(dimension, sitk.sitkIdentity)
print('Dimension: {}'.format(identity.GetDimension()))

# Points are always defined in physical space
point = (1.0, 1.0)

def transform_point(transform, point):
    transformed_point = transform.TransformPoint(point)
    print('Point {} is transformed to {}'.format(point, transformed_point))

transform_point(identity, point)
```

Out:

*Identity Transform*
Dimension: 2
Point (1.0, 1.0) is transformed to (1.0, 1.0)

Transform are defined by two sets of parameters, the `Parameters` and `FixedParameters`. `FixedParameters` are not changed during the optimization process when performing registration.
**Translation Transform**  This transform simply translates input points by a offset. For the TranslationTransform, the Parameters are the values of this translation Offset.

```python
print('Translation Transform')
translation = sitk.TranslationTransform(dimension)

print('Parameters: {}'.format(translation.GetParameters()))
print('Offset: {}'.format(translation.GetOffset()))
print('FixedParameters: {}'.format(translation.GetFixedParameters()))
transform_point(translation, point)

print('')
translation.SetParameters((3.1, 4.4))
print('Parameters: {}'.format(translation.GetParameters()))
transform_point(translation, point)
```

Out:

```
*Translation Transform*
Parameters: (0.0, 0.0)
Offset: (0.0, 0.0)
FixedParameters: ()
Point (1.0, 1.0) is transformed to (1.0, 1.0)

Parameters: (3.1, 4.4)
Point (1.0, 1.0) is transformed to (4.1, 5.4)
```

**Affine Transform**  The affine transform is capable of representing translations, rotations, shearing, and scaling. Affine transformation can be described with the following equation:

\[ x' = A(x - c) + t \]

where \( x \) is the input coordinate vector and \( x' \) is the output coordinate vector, \( A \) is the affine matrix, \( t \) is the translation and \( c \) is the centre of the affine transform. By default, \( A = I \) and \( C = 0 \)

```python
print('Affine Transform')
affine = sitk.AffineTransform(dimension)

print('Parameters: {}'.format(affine.GetParameters()))
print('FixedParameters: {}'.format(affine.GetFixedParameters()))
transform_point(affine, point)

print('')
affine.SetTranslation((3.1, 4.4))
print('Parameters: {}'.format(affine.GetParameters()))
transform_point(affine, point)
```

Out:

```
*Affine Transform*
Parameters: (1.0, 0.0, 0.0, 1.0, 0.0, 0.0)
FixedParameters: (0.0, 0.0)
Point (1.0, 1.0) is transformed to (1.0, 1.0)

Parameters: (1.0, 0.0, 0.0, 1.0, 3.1, 4.4)
Point (1.0, 1.0) is transformed to (4.1, 5.4)
```

A number of other transforms exist to represent non-affine deformations, well-behaved rotation in 3D, etc. See the Next guide for more information.
Applying Transforms to Images

Let’s create a grid image to illustrate our transforms.

```python
grid = sitk.GridSource(outputPixelType=sitk.sitkUInt16,
                         size=(250, 250),
                         sigma=(0.5, 0.5),
                         gridSpacing=(5.0, 5.0),
                         gridOffset=(0.0, 0.0),
                         spacing=(0.2, 0.2))
myshow(grid, 'Grid Input')
```

To apply the transform, a resampling operation is required.

**Note:** Resample applies transform to physical space, not voxel coordinates directly. Once physical space is transformed, you will need to specify how you view this space by setting output origin, spacing and direction. Alternatively, you can specify a reference image so that output origin, spacing and direction are set to that of the reference image.

In the following `resample` function, output image Origin, Spacing, Size, Direction are taken from the reference.

```python
def resample(image, transform):
    reference_image = image
    interpolator = sitk.sitkCosineWindowedSinc
    default_value = 100.0
    return sitk.Resample(image, reference_image, transform, interpolator, default_value)
```

Let’s apply translation to image

```python
translation = sitk.TranslationTransform(2)
translation.SetOffset((3.1, 4.6))
transform_point(translation, point)
resampled = resample(grid, translation)
myshow(resampled, 'Resampled Translation')
```
Out:

Point (1.0, 1.0) is transformed to (4.1, 5.6)

What happened? The translation is positive in both directions. Why does the output image move down and to the left?

**Note:** It is important to keep in mind that a transform in a resampling operation defines the transform from the output space to the input space.

```python
inv_translation = translation.GetInverse()
transform_point(inv_translation, point)
resampled = resample(grid, inv_translation)
myshow(resampled, 'Inverse Resampled')
```
Out:

Point (1.0, 1.0) is transformed to (-2.1, -3.5999999999999996)

An affine (line preserving) transformation, can perform translation, scaling, rotation and shearing:

**Translation**

```python
x_translation, y_translation = (3.1, 4.6)
affine = sitk.AffineTransform(2)
affine.SetTranslation((x_translation, y_translation))
resampled = resample(grid, affine)
myshow(resampled, 'Translated')
```

**Scaling**

```python
x_scale, y_scale = 3.0, 0.7
affine = sitk.AffineTransform(2)
affine.Scale((x_scale, y_scale))
resampled = resample(grid, affine)
myshow(resampled, 'Scaled')
```
Rotation  We can either use `AffineTransform::Rotate` or directly set rotation matrix. Let's take the first route.

```python
degrees = 20
affine = sitk.AffineTransform(2)
radians = np.pi * degrees / 180.
affine.Rotate(axis1=0, axis2=1, angle=radians)
resampled = resample(grid, affine)
myshow(resampled, 'Rotated')
```

Shearing  This time, let's directly set the matrix.
```python
x_shear, y_shear = 0.3, 0.1

matrix = np.eye(2)
matrix[0, 1] = -x_shear
matrix[1, 0] = -y_shear
print(matrix)

affine.SetMatrix(matrix.ravel())
resampled = resample(grid, affine)
myshow(resampled, 'Sheared')
```

Out:
```
[[ 1. -0.3]
 [-0.1  1.]]
```

**Composite Transform**

It is possible to compose multiple transform together into a single transform object. With a composite transform, multiple resampling operations are prevented, so interpolation errors are not accumulated. For example, an affine transformation that consists of a translation and rotation:

```python
translate = (8.0, 16.0)
rotate = 20.0

affine = sitk.AffineTransform(2)
affine.SetTranslation(translate)
affine.Rotate(axis1=0, axis2=1, angle=np.pi / 180 * rotate)

resampled = resample(grid, affine)
myshow(resampled, 'Single Transform')
```
This can also be represented with two Transform objects applied in sequence with a Composite Transform:

```python
translation = sitk.TranslationTransform(2)
translation.SetOffset(translate)

affine = sitk.AffineTransform(2)
affine.Rotate(axis1=0, axis2=1, angle=np.pi / 180 * rotate)

composite = sitk.Transform(2, sitk.sitkComposite)
composite.AddTransform(translation)
composite.AddTransform(affine)

resampled = resample(grid, composite)
myshow(resampled, 'Composite of Two Transforms')
```
Beware, transforms are non-commutative – order matters!

```python
composite = sitk.Transform(2, sitk.sitkComposite)
composite.AddTransform(affine)
composite.AddTransform(translation)
resampled = resample(grid, composite)
myshow(resampled, 'Composite in reverse')
```

**Total running time of the script:** (0 minutes 1.636 seconds)

Download Python source code: plot_transforms.py
Download Jupyter notebook: plot_transforms.ipynb
Generated by Sphinx-Gallery

### 1.4.4 Visualizing Images

**Visualizing 2D images**

In this example, we will explore using matplotlib to display images in our notebooks, and work towards developing a reusable function to display 2D,3D, color for SimpleITK images.

```python
import matplotlib.pyplot as plt
import SimpleITK as sitk
from downloaddata import fetch_data as fdata
```

Fetch and read image

```python
img1 = sitk.ReadImage(fdata("cthead1.png"))
img2 = sitk.ReadImage(fdata("VM1111Shrink-RGB.png"))
```

Out:

Fetching cthead1.png
Fetching VM1111Shrink-RGB.png
SimpleITK has a built-in `Show` method which saves the image to disk and launches a user configurable program (defaults to ImageJ), to display the image.

```python
sitk.Show(img1, title="cthead1")
sitk.Show(img2, title="Visible Human Head")
```

**Plotting with matplotlib**

You can also use matplotlib to show images.

```python
nda = sitk.GetArrayFromImage(img1)
plt.imshow(nda)
nda = sitk.GetArrayFromImage(img2)
plt.imshow(nda)
```
Let's write a function which directly takes Image object and shows it.

```python
def myshow(img):
    nda = sitk.GetArrayFromImage(img)
    plt.imshow(nda)
    plt.show()

myshow(sitk.Expand(img2, [10] * 5))
```
This image does not appear bigger.

There are numerous improvements that we can make:

- support 3d images
- include a title
- use physical pixel size for axis labels
- show the image as gray values

```python
def myshow(img, title=None, margin=0.05, dpi=80):
    nda = sitk.GetArrayFromImage(img)
    spacing = img.GetSpacing()

    if nda.ndim == 3:
        # fastest dim, either component or x
        c = nda.shape[-1]

        # the the number of components is 3 or 4 consider it an RGB image
        if c not in (3, 4):
            nda = nda[nda.shape[0] // 2, :, :]

    elif nda.ndim == 4:
        c = nda.shape[-1]

        if c not in (3, 4):
```

1.4. User Guide
raise RuntimeError("Unable to show 3D-vector Image")

# take a z-slice
nda = nda[nda.shape[0] // 2, :, :, :]

xsize = nda.shape[1]
ysize = nda.shape[0]

# Make a figure big enough to accommodate an axis of xpixels by ypixels
# as well as the ticklabels, etc...
figsize = (1 + margin) * xsize / dpi, (1 + margin) * ysize / dpi

plt.figure(figsize=figsize, dpi=dpi, tight_layout=True)
ax = plt.gca()

extent = (0, xsize * spacing[0], ysize * spacing[1], 0)

if nda.ndim == 2:
    t.set_cmap("gray")

if (title):
    plt.title(title)

plt.show()

myshow(sitk.Expand(img2, [2, 2]), title="Big Visible Human Head")
The `myshow` function is really useful. We will build up on it for 3d images (`myshow3d`) in the next guide. They have been copied into a “myshow.py” file so that they can be imported into other guides.

**Total running time of the script:** (0 minutes 1.667 seconds)
Visualizing Segmentations

In previous guides, we’ve seen how to visualize 2d and 3d images. We’ve written functions `myshow` and `myshow3d` which we will be using in this guide.

We will also look at the subtleties of working with image filters that require the input images’ to be overlapping.

```
import matplotlib.pyplot as plt
import SimpleITK as sitk
from downloaddata import fetch_data as fdata
from myshow import myshow, myshow3d
```

LabelOverlay In 2D

We start by loading a segmented image. As the segmentation is just an image with integral data, we can display the labels as we would any other image.

```
img1 = sitk.ReadImage(fdata("cthead1.png"))
img1_seg = sitk.ReadImage(fdata("2th_cthead1.png"))

myshow(img1, title="cthead1")
myshow(img1_seg, title="Label Image as Grayscale")
```

Out:

Fetching cthead1.png
Fetching 2th_cthead1.png

We can also map the scalar label image to a color image as shown below.
Most filters which take multiple images as arguments require that the images occupy the same physical space. That is the pixel you are operating must refer to the same location. Luckily for us our image and labels do occupy the same physical space, allowing us to overlay the segmentation onto the original image.

We can also overlay the labels as contours.
LabelOverlay In 3D

The Surgical Planning Laboratory at Brigham and Women’s Hospital has a wonderful Multi-modality MRI-based Atlas of the Brain that we can use.

```python
img_T1 = sitk.ReadImage(
    fdata("nac-hncma-atlas2013-Slicer4Version/Data/A1_grayT1.nrrd"))
img_T2 = sitk.ReadImage(
    fdata("nac-hncma-atlas2013-Slicer4Version/Data/A1_grayT2.nrrd"))
img_labels = sitk.ReadImage(
    fdata("nac-hncma-atlas2013-Slicer4Version/Data/hncma-atlas.nrrd"))

myshow(img_T1, title='T1')
myshow(img_T2, title='T2')
myshow(sitk.LabelToRGB(img_labels), title='lables')
```
Physical Space Issues

Why doesn’t this work? The images do not overlap in physical space.

All the functions in SimpleITK work on underlying physical space. Therefore, mismatch in physical space like image origin etc. will raise errors.

```
try:
    size = img_T1.GetSize()
    myshow3d(sitk.LabelOverlay(img_T1, img_labels),
             yslices=range(50, size[1] - 50, 20),
             zslices=range(50, size[2] - 50, 20), dpi=30)
except Exception as e:
    print(e)
```

Out:

```
Fetching nac-hncma-atlas2013-Slicer4Version/Data/A1_grayT1.nrrd
Fetching nac-hncma-atlas2013-Slicer4Version/Data/A1_grayT2.nrrd
Fetching nac-hncma-atlas2013-Slicer4Version/Data/hncma-atlas.nrrd

```

Two ways to solve our problem:

1. resample the labels onto the image grid
2. resample the image onto the label grid.

The difference between the two from a computation standpoint depends on the grid sizes and on the interpolator used to estimate values at non-grid locations.
Note interpolating a label image with an interpolator that can generate non-label values is problematic as you may end up with an image that has more classes/labels than your original. This is why we only use the nearest neighbor interpolator when working with label images.

**Option 1**: Resample the label image using the identity transformation

```
resampled_img_labels = sitk.Resample(img_labels, img_T1, sitk.Transform(),
        sitk.sitkNearestNeighbor, 0.0,
        img_labels.GetPixelID())
```

Overlay onto the T1 image, requires us to rescale the intensity of the T1 image to $[0,255]$ and cast it so that it can be combined with the color overlay (we use an alpha blending of 0.5).

```
rescaled_T1 = sitk.Cast(sitk.RescaleIntensity(img_T1), sitk.sitkUInt8)
myshow3d(sitk.LabelOverlay(rescaled_T1, resampled_img_labels, 0.5),
        ysllices=range(50, size[1] - 50, 20),
        zsllices=range(50, size[2] - 50, 20),
        dpi=100)
```

**Option 2**: Resample the T1 image using the identity transformation.

```
resampled_T1 = sitk.Resample(img_T1, img_labels, sitk.Transform(),
        sitk.sitkLinear, 0.0, img_T1.GetPixelID())
```

As above, we need to use rescale the intensity of resampled T1

```
rescaled_T1 = sitk.Cast(sitk.RescaleIntensity(resampled_T1), sitk.sitkUInt8)
myshow3d(sitk.LabelOverlay(rescaled_T1, img_labels, 0.5),
        ysllices=range(50, size[1] - 50, 20),
        zsllices=range(50, size[2] - 50, 20),
        dpi=100)
```

Why are the two displays above different? (hint: in the calls to the “myshow3d” function the indexes of the y and z slices are the same).

**Total running time of the script**: (0 minutes 7.928 seconds)

Download Python source code: plot_visseg.py

Download Jupyter notebook: plot_visseg.ipynb

Generated by Sphinx-Gallery
Visualizing 3D images

In previous guide, we’ve seen how to visualize 2d images. We’ve written a function `myshow` which we will be using in this guide.

Now let’s move on to visualizing real MRI images. The Surgical Planning Laboratory at Brigham and Women’s Hospital has a wonderful Multi-modality MRI-based Atlas of the Brain that we can use.

Please note, what is done here is for convenience and is not the common way images are displayed for radiological work.

Get Images

```python
import matplotlib.pyplot as plt
import SimpleITK as sitk
from downloaddata import fetch_data as fdata
from myshow import myshow

img_T1 = sitk.ReadImage(fdata("nac-hncma-atlas2013-Slicer4Version/Data/A1_grayT1.nrrd"))
img_T2 = sitk.ReadImage(fdata("nac-hncma-atlas2013-Slicer4Version/Data/A1_grayT2.nrrd"))

myshow(img_T1, title='T1')
myshow(img_T2, title='T2')
myshow(sitk.LabelToRGB(img_labels), title='lables')
```
Visualize another axis.

```python
size = img_T1.GetSize()
myshow(img_T1[::, size[1] // 2, :])
```

Let's visualize all three orthogonal views. You can use `sitk.Tile` for tiling images.

```python
slices = [img_T1[size[0] // 2, :, :], img_T1[::, size[1] // 2, :],
          img_T1[::, ::, size[2] // 2]]
myshow(sitk.Tile(slices, [3, 1]), dpi=20)
```
Visualize 5 slices in one axis.

```python
nslices = 5
slices = [img_T1[:, :, s] for s in range(0, size[2], size[0] // (nslices + 1))]
myshow(sitk.Tile(slices, [1, 0]), dpi=100)
```
Let’s create a version of the show methods which allows the selection of slices to be displayed.

```python
def myshow3d(img, xslices=[], yslices=[], zslices=[], title=None, margin=0.05, dpi=80):
    img_xslices = [img[s, :, :] for s in xslices]
    img_yslices = [img[:, s, :] for s in yslices]
    img_zslices = [img[:, :, s] for s in zslices]

    maxlen = max(len(img_xslices), len(img_yslices), len(img_zslices))

    img_null = sitk.Image([0, 0], img.GetPixelID(),
                          img.GetNumberOfComponentsPerPixel())

    img_slices = []
    d = 0
    if len(img_xslices):
        img_slices += img_xslices + [img_null] * (maxlen - len(img_xslices))
        d += 1
    if len(img_yslices):
        img_slices += img_yslices + [img_null] * (maxlen - len(img_yslices))
        d += 1
    if len(img_zslices):
        img_slices += img_zslices + [img_null] * (maxlen - len(img_zslices))
        d += 1

    if maxlen != 0:
        if img.GetNumberOfComponentsPerPixel() == 1:
            img = sitk.Tile(img_slices, [maxlen, d])
            # TO DO check in code to get Tile Filter working with vector images
        else:
            img_comps = []
            for i in range(0, img.GetNumberOfComponentsPerPixel()):
                img_slices_c = [sitk.VectorIndexSelectionCast(s, i)
                                for s in img_slices]
                img_comps.append(sitk.Tile(img_slices_c, [maxlen, d]))
            img = sitk.Compose(img_comps)

    myshow(img, title, margin, dpi)

myshow3d(img_T1, yslices=range(50, size[1] - 50, 20),
          zslices=range(50, size[2] - 50, 20), dpi=100)
```

```python
myshow3d(img_T2, yslices=range(50, size[1] - 50, 30),
          zslices=range(50, size[2] - 50, 20), dpi=100)
```
1.5 Gallery

General-purpose and introductory examples for SimpleITK.

User Guide introduces conventions and basic image manipulations.

1.5.1 Edges and Lines

Canny edge detector

The Canny filter is a multi-stage edge detector. It uses a filter based on the derivative of a Gaussian in order to compute the intensity of the gradients. The Gaussian reduces the effect of noise present in the image. Then, potential edges are thinned down to 1-pixel curves by removing non-maximum pixels of the gradient magnitude. Finally, edge pixels are kept or removed using hysteresis thresholding on the gradient magnitude.

The Canny has three adjustable parameters: the width of the Gaussian (the noisier the image, the greater the width), and the low and high threshold for the hysteresis thresholding.
Input image

Canny filter, $\sigma = 1$
import matplotlib.pyplot as plt
import SimpleITK as sitk

image = sitk.ReadImage('../example_images/lena_bw.png')
image = sitk.Cast(image, sitk.sitkFloat64)

# Compute the Canny filter for two values of sigma
edges1 = sitk.CannyEdgeDetection(image, lowerThreshold=5, upperThreshold=10,
                                  variance=[1, 1])
edges2 = sitk.CannyEdgeDetection(image, lowerThreshold=5, upperThreshold=10,
                                  variance=[3, 3])

# Convert to numpy array for display
image = sitk.GetArrayFromImage(image)
edges1 = sitk.GetArrayFromImage(edges1)
edges2 = sitk.GetArrayFromImage(edges2)

# display results
fig, (ax1, ax2, ax3) = plt.subplots(nrows=3, ncols=1, figsize=(5, 15),
                                   sharex=True, sharey=True)

ax1.imshow(image, cmap=plt.cm.gray)
ax1.axis('off')
ax1.set_title('Input image', fontsize=20)

ax2.imshow(edges1, cmap=plt.cm.gray)
ax2.axis('off')
ax2.set_title('Canny filter, $\sigma=1$', fontsize=20)

ax3.imshow(edges2, cmap=plt.cm.gray)
ax3.axis('off')
ax3.set_title('Canny filter, $\sigma=3$', fontsize=20)

fig.tight_layout()
plt.show()
1.6.1 Installation

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

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

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

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

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

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

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

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

### 1.6.2 How to Use

#### What filters are currently available in SimpleITK?

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

#### 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)

#### Can I use another image file viewer beside ImageJ?

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

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

```
SITK_SHOW_EXTENSION=".png"
export SITK_SHOW_EXTENSION
```
Use of an extension unsupported by ITK results in an error message. For the supported image formats, here is the ITK Image IO Filters.

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

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

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

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

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

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

**Mac OS X**

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

**Linux**

```bash
export SITK_SHOW_COMMAND="Slicer"
```

**Windows**

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

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

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

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

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

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

---

1.6. Frequently Asked Questions 99
The SITK_SHOW_COMMAND tells SimpleITK.Show() to launch Java with ij.jar and then execute the open macro with an image file. The SITK_SHOW_COLOR_COMMAND does these same things and then executes the ImageJ “Make Composite” command to treat a multichannel image as a composite color image.

### 1.6.3 Wrapping

Python

Tcl

Java

C#

R

### 1.6.4 Compilation

**Is my compiler supported?**

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

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

**Committed to Support**

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

**Noted Problems**

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

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

```
typedef std::tr1::function< MemberFunctionResultType ( ) > FunctionObjectType;
```

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

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

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

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

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

<table>
<thead>
<tr>
<th>Clang 5.0 compatibility</th>
<th>-stdlib=libc++</th>
<th>-stdlib=libstdc++</th>
</tr>
</thead>
<tbody>
<tr>
<td>(c++03)</td>
<td>FAIL</td>
<td>OK</td>
</tr>
<tr>
<td>-std=c++11</td>
<td>OK (&gt;=0.8)</td>
<td>OK</td>
</tr>
</tbody>
</table>

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

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

**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. To install the command line developer tools enter the following: ```xcode-select --install```

To reset the default command line tools path: ```xcode-select --reset```

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

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

Do I need to download an optional package for C99?

SimpleITK will proved a “stdint.h” header if missing on the system.
How do I build with Visual Studio 2008?

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

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

What Configurations on Windows are Supported For Building?

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

<table>
<thead>
<tr>
<th>Legend</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nightly</td>
<td>This combination of options is nightly tested, and known to work.</td>
</tr>
<tr>
<td></td>
<td>This combinations has been manually tested, and is expected to work.</td>
</tr>
<tr>
<td></td>
<td>It is not known if this combinations of options will work.</td>
</tr>
<tr>
<td></td>
<td>This combination likely has problems, and is not recommended.</td>
</tr>
<tr>
<td>FAIL</td>
<td>These options are known not to work.</td>
</tr>
</tbody>
</table>

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

Why are all of the configurations not supported on Windows?

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

LINK : fatal error LNK1102: out of memory
LINK : fatal error LNK1248: image size (80000010) exceeds maximum allowable size (80000000)

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

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

Where is the Test Data?

The way testing data is obtained changed with SimpleITK 0.7. If you download the source tar-ball is should be included. If you have obtained the source code from the git repository, it should be downloaded as part of the build process.

1.7 API Reference
Indices and tables

- genindex
- modindex
- search