SimpleITK Tutorial

Image processing for mere mortals

Insight Software Consortium

January 10, 2012

What this Tutorial is about

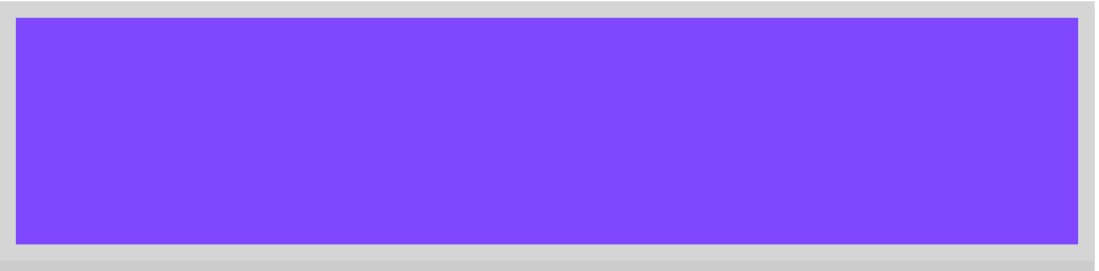
- Provide working knowledge of the SimpleITK platform

Program

- Introduction (15min)
- Basic Tutorials I (45min)
- Basic Tutorials II (45min)
- Intermediate Tutorials (45min)
- Advanced Topics (40min)
- Wrap-up (10min)

Tutorial Goals

- Gentle introduction to ITK
- Introduce SimpleITK



ITK Overview

How many are familiar with ITK?

Ever seen code like this?

```
1  // Setup image types.
2  typedef    float InputPixelType;
3  typedef    float OutputPixelType;
4  typedef itk::Image<InputPixelType, 2> InputImageType;
5  typedef itk::Image<OutputPixelType,2> OutputImageType;
6  // Filter type
7  typedef itk::DiscreteGaussianImageFilter<
8                 InputImageType, OutputImageType >
9          FilterType;
10 // Create a filter
11 FilterType::Pointer filter = FilterType::New();
12 // Create the pipeline
13 filter->SetInput( reader->GetOutput() );
14 filter->SetVariance( 1.0 );
15 filter->SetMaximumKernelWidth( 5 );
16 filter->Update();
17 OutputImageType::Pointer blurred = filter->GetOutput();
```

What if you could write this?

```
import SimpleITK
input = SimpleITK.ReadImage ( filename )
output = SimpleITK.DiscreteGaussianFilter( input, 1.0, 5 )
```

What if you could write this?

```
1 import SimpleITK
2 input = SimpleITK.ReadImage ( filename )
3 output = SimpleITK.DiscreteGaussianFilter( input, 1.0, 5 )
```

We are here to tell you that you can...

Goals of SimpleITK

- Be an "on-ramp" for ITK
- Simplify the use of ITK by
  - Providing a templateless, typeless layer for ITK in C++
  - Providing wrappings in scripting languages
  - Providing access to most ITK algorithms

SimpleITK Architectural Overview

- Conceptually, SimpleITK is an application library built on ITK
- All functionality provided by ITK
- Components:
  - Template expansion system
  - C++ library
  - Small SWIG definition (more details later)
  - "Glue" code for several scripting languages
  - Some language utilities
- Open Source, Apache licensed project (`http://www.opensource.org/licenses/apache2.0.php`)
- Hosted by GitHub (`https://github.com/SimpleITK/SimpleITK`)

Templates in ITK

```
1  typedef unsigned char PixelType;
2  enum {ImageDimension = 2};
3  typedef itk::Image<PixelType,ImageDimension> ImageType;
4  typedef itk::Vector<float,ImageDimension> VectorType;
5  typedef itk::Image<VectorType,ImageDimension> FieldType;
6  typedef itk::Image<VectorType::ValueType,ImageDimension> FloatImageType;
7  typedef ImageType::IndexType  IndexType;
8  typedef ImageType::SizeType   SizeType;
9  typedef ImageType::RegionType RegionType;
10 typedef itk::MultiResolutionPDEDeformableRegistration
11    <ImageType, ImageType, FieldType> RegistrationType;
```

Template Freedom

```
using itk::simple;
// Read the image file
ImageFileReader reader;
reader.SetFileName ( "/my/fancy/file.nrrd" );
Image image = reader.Execute();

// This filters perform a gaussian bluring with sigma in
// physical space. The output image will be of real type.
SmoothingRecursiveGaussianImageFilter gaussian;
gaussian.SetSigma ( 2.0 );
Image blurredImage = gaussian.Execute ( image );
```

Programming Models

- Object oriented
- Function oriented

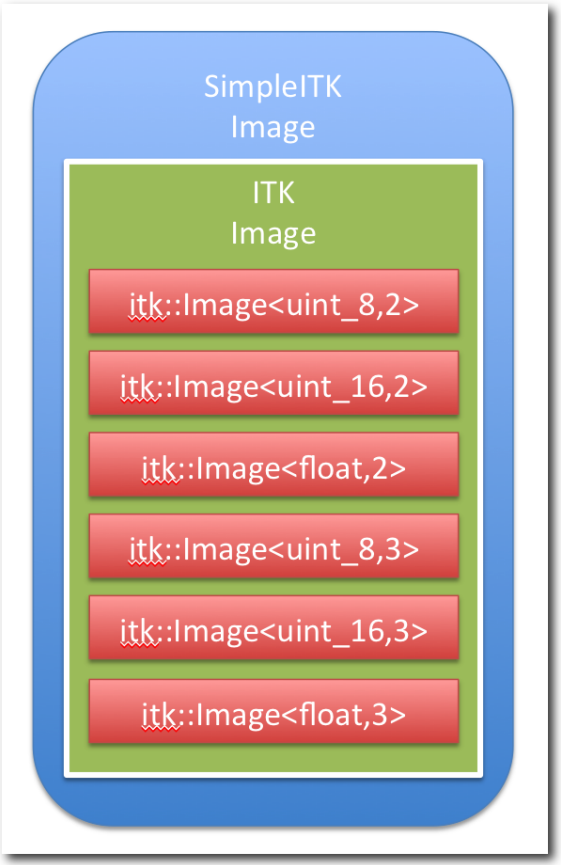
More about this later

Wrapping

Transformed by SWIG

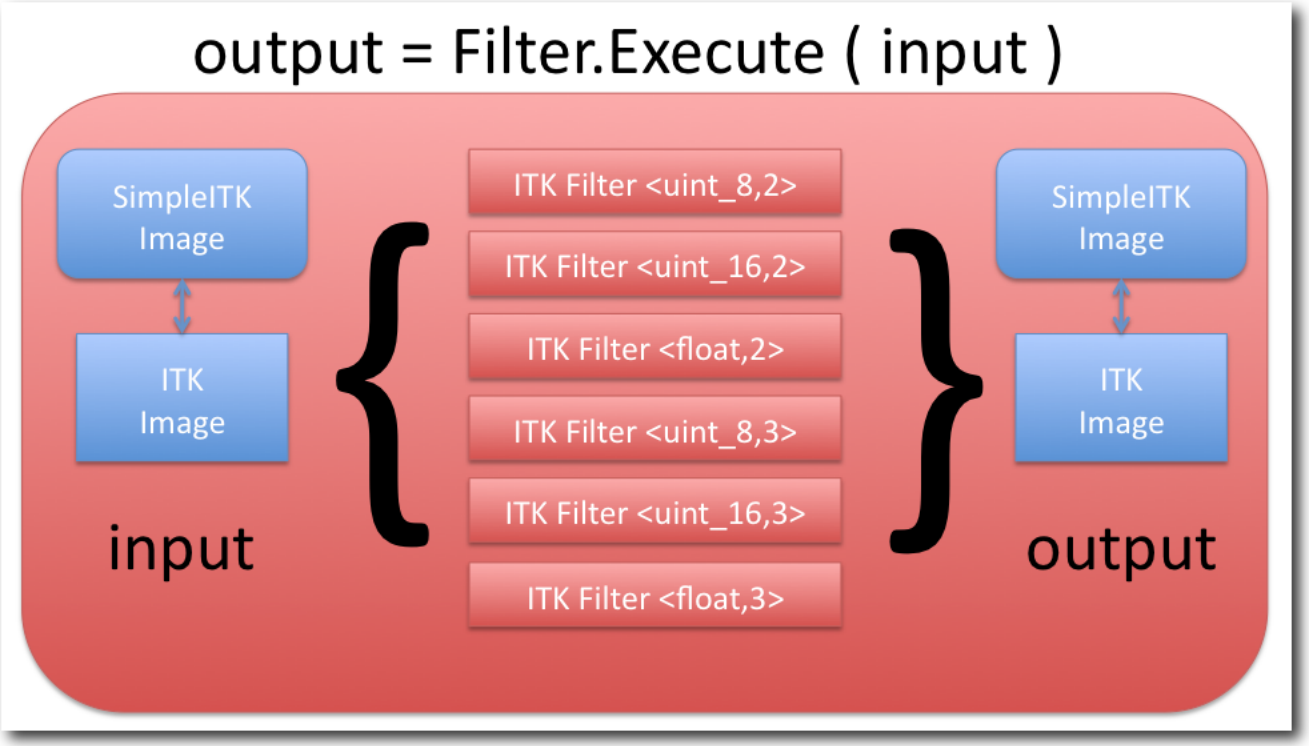
- Parses header and "interface" files
- Automatically creates scripting "glue"
- Wrappings available for:
  - Python
  - Java
  - C#
  - Tcl, Lua, Ruby
  - Others as requested...

Image Anatomy



- SimpleITK images contain `itk::Image`
- Hides the templates
- Adds useful "utility" functions

Filter Anatomy



- SimpleITK filters contain `itk::ImageToImageFilter`
- Hides the templates
- Adds useful "utility" functions
- (Mainly) `output = Filter.Execute ( input )`

Basic Tutorial

Note on the Tutorial

- Most examples will be Python
- Obvious translation to other languages
- C++ usage (generally) obvious

Image Class

- Creation
- Number of dimensions, size, origin, spacing
- Pixel access

Back to iPython (`Examples/BasicTutorial1/Image.py`)

What just happened?

```
1 import SimpleITK as sitk
2 # Create an image
3 image = sitk.Image ( 256, 256, 256, sitk.sitkInt16 );
4 # How about 2d?
5 twoD = sitk.Image ( 64, 64, sitk.sitkFloat32 )
```

- sitk is the module
- Image is the constructor for the Image class
- Height, width, depth (omit depth for 2D images)
- Datatype (more on this later)

Back to iPython (`Examples/BasicTutorial1/Image.py`)

What just happened?

```
# Addressing pixels
image.GetPixel ( 0, 0, 0 )
image.SetPixel ( 0, 0, 0, 1 )
image.GetPixel ( 0, 0, 0 )
```

- Get the voxel value at [0,0,0]?
- Hmm, I don't like it, so set to 1
- What is the value at [0,0,0] now?

Back to iPython (`Examples/BasicTutorial1/Image.py`)

What just happened?

```
1 # Addressing pixels
2 image[0,0,0]
3 image[0,0,0] = 10
4 image[0,0,0]
```

Without warning, we sprinkled syntatic sugar on you!

- image[0,0,0] is shorthand for Image.GetPixel(0,0,0)
- image[0,0,0] = 10 is shorthand for Image.SetPixel(0,0,0,10)

Summary

- Images are created using SimpleITK.Image (w, h, d, Type)
- Images can be 2- or 3-dimensional
- Images can describe themselves
- Images have simple pixel accessors

Questions before we move on?

Memory Management

Images

Images...

- usually allocated on the stack
- are copy-on-write
- use internal smart-pointers

Back to iPython (`Examples/BasicTutorial1/MemoryManagement.py`)

Image Memory Management

```
image = SimpleITK.Image ( 32, 32, 32, SimpleITK.sitkInt16 )
print image
...
Image (0x94f2d98)
  Reference Count: 1
...
# Clone image
b = SimpleITK.Image ( image )
print image
...
Image (0x94f2d98)
  Reference Count: 2
...
print b
...
Image (0x94f2d98)
```

Image Memory Management

```
print b
...
Image (0x94f2d98)
...
b[0,0,0] = 1
print b
...
Image (0x94f4cb0)
  Reference Count: 1
...
print image
...
Image (0x94f2d98)
  Reference Count: 1
...
```

Filters

Filters...

- usually allocated on the stack
- tend to clean up after themselves
- do not hold on to images
- no access to the ITK pipeline

...more on this later...

Memory Management Strategies

C++...

- No need for explicit management
- Let images clean up after themselves
- Let filters clean up after themselves

Wrapped...

- Utilize language-specific memory management
- Automatic in Python, Java, Ruby, C#, Lua
- More manual in Tcl

Basic Tutorial 2

Hands On

`Examples/BasicTutorial2/Filters.py`

What just happened?

```
# Simple smoothing
smooth = sitk.SmoothingRecursiveGaussian ( image, 2.0 )
sitk.Show ( sitk.Subtract ( image, smooth ) )
...
RuntimeError: Exception thrown in SimpleITK Subtract: ...
sitk::ERROR: Both images for SubtractImageFilter don't match type or dimension!
...
```

- The output of SmoothingRecursiveGaussian is of type float
- The input image is signed short
- Most SimpleITK filters with 2 inputs require the same type
- Let's fix the problem

Introducing Cast

```
# Much better
print "Before: ", smooth.GetPixelIDTypeAsString()
smooth = sitk.Cast ( smooth, image.GetPixelIDValue() )
print "After: ", smooth.GetPixelIDTypeAsString()
sitk.Show ( sitk.Subtract ( image, smooth ), "DiffWithGaussian" )
```

Back to iPython (Examples/BasicTutorial2/Filters.py)

Sizes and Indices

```
# Extract
size = [64, 64, 1]
start = [64, 0, 0]
sitk.Show ( sitk.Extract ( image, size, start ), "Extracted" )
```

in C++ / ITK code this would use

```
typedef unsigned char PixelType;
enum {ImageDimension = 2};
typedef itk::Image<PixelType,ImageDimension> ImageType;

typedef ImageType::IndexType  IndexType;
typedef ImageType::SizeType   SizeType;
typedef ImageType::RegionType RegionType;
```

- SimpleITK uses STL vectors
- Wrapping converts to language-specific constructs (tuples/arrays)

Pixel-wise Operators

```
# Use pixel-wise operators
sitk.Show ( 127 * image + 127 * sitk.BinaryErode ( image ), "ThinErosion" )
```

Table: SimpleITK Pixel-wise Operators

Operator	Equivalant	Usage†
$+$	Add, AddConstantTo	$A+B$, $s+A$, $s+B$
$-$	Subtract, SubtractConstantFrom	$A-B$, $s-A$, $s-B$
$*$	Multiply, MultiplyByConstant	$A*B$, $s*A$, $s*B$
$/$	Divide, DivideByConstant	A/B, s/A, B/s
&	And "and"	$A\&B$
|	Or "or"	$A\|B$
$\sim$	Not "not"	A
$**$††	Pow, PowToConstant	$A**B$, $A**s$

† A and B are images (2D or 3D), s is a scalar

†† python only

Operator Example

```
1 sitk.Hash ( image + 2 )
2 sitk.Hash ( sitk.AddConstantTo ( image, 2 ) )
3
4 sitk.Hash ( image * 2 )
5 sitk.Hash ( sitk.MultiplyByConstant ( image, 2 ) )
```

- Hash calculates the MD5 or SHA1 hash of the *image* data
- Used to tell if two images are exactly identical

Back to iPython (`Examples/BasicTutorial2/Morphology.py`)

SimpleITK with Numpy

Numpy is:

- Python's universal numerics foundation
- Provides high performance basic linear algebra

numpy interface functions:

```
1 def GetArrayFromImage(image):
2     """Get a numpy array from a SimpleITK Image."""
3
4 def GetImageFromArray( arr ):
5     """Get a SimpleITK Image from a numpy array."""
```

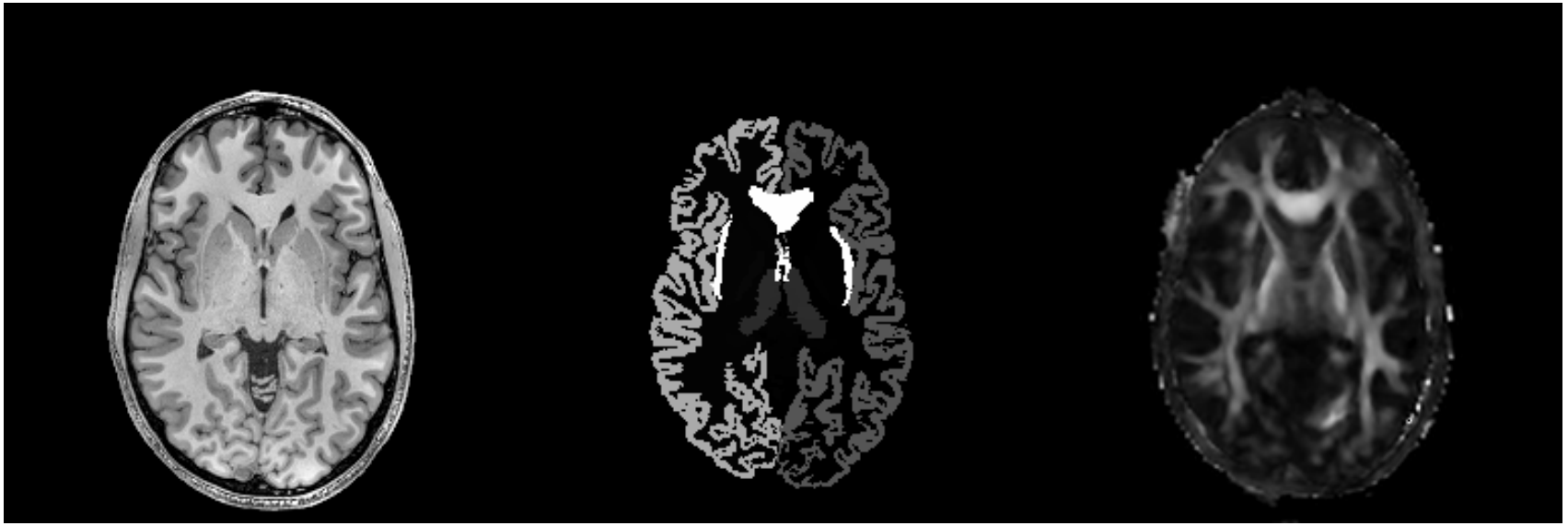
- Both of these methods do a deep copy of the image
- Ensures safety, bypassing error-prone memory issues

Python Example

To iPython (`Examples/InteractiveTutorial/05-02-Numpy.py`)

Image Measurements

The Scenerio



- You collected 1000 Diffusion Weighted Imaging scan sessions
- Now, compute the fractional anisotropy measures for a set of anatomical regions

Python Example

To iPython (`Examples/InteractiveTutorial/MeasureRegions.py`)

Feature Detection

Edge Detection

- **CannyEdgeDetection**

```
1    Image CannyEdgeDetection ( const Image& ,
2      double inLowerThreshold = 0.0,
3      double inUpperThreshold = 0.0,
4      std::vector<double> inVariance = std::vector<double>(3, 0.0),
5      std::vector<double> inMaximumError = std::vector<double>(3, 0.01) );
```

- **SobelEdgeDetection**

```
1    Image SobelEdgeDetection ( const Image&  );
```

- **ZeroCrossingBasedEdge**

```
1    Image ZeroCrossingBasedEdgeDetection ( const Image& ,
2      double inVariance = 1,
3      uint8_t inForegroundValue = 1u,
4      uint8_t inBackgroundValue = 0u,
5      double inMaximumError = 0.1 );
```

- **GradientMagnitudeRecursiveGaussian**

```
1    Image GradientMagnitudeRecursiveGaussian ( const Image& ,
2      double inSigma = 1.0,
3      bool inNormalizeAcrossScale = false );
```

Image Derivatives

- **Derivative**

```
1    Image Derivative ( const Image& ,
2      unsigned int inDirection = 0u,
3      unsigned int inOrder = 1u,
4      bool inUseImageSpacing = true );
```

- **RecursiveGaussian**

```
1    Image RecursiveGaussian ( const Image& ,
2      double inSigma = 1.0,
3      bool inNormalizeAcrossScale = false,
4      itk::simple::RecursiveGaussianImageFilter::OrderEnumType inOrder = itk::simp
5      unsigned int inDirection = 0u );
```

Advanced Topics

Building SimpleITK in 5 Easy Commands

```
git clone --recursive https://github.com/SimpleITK/SimpleITK
mkdir SimpleITK/SimpleITK-build
cd SimpleITK/SimpleITK-build
cmake ../SuperBuild
make -j 5
```

Or as part of Slicer

ITK_VERSION_MAJOR=4
Slicer_USE_SimpleITK=ON

More complete version

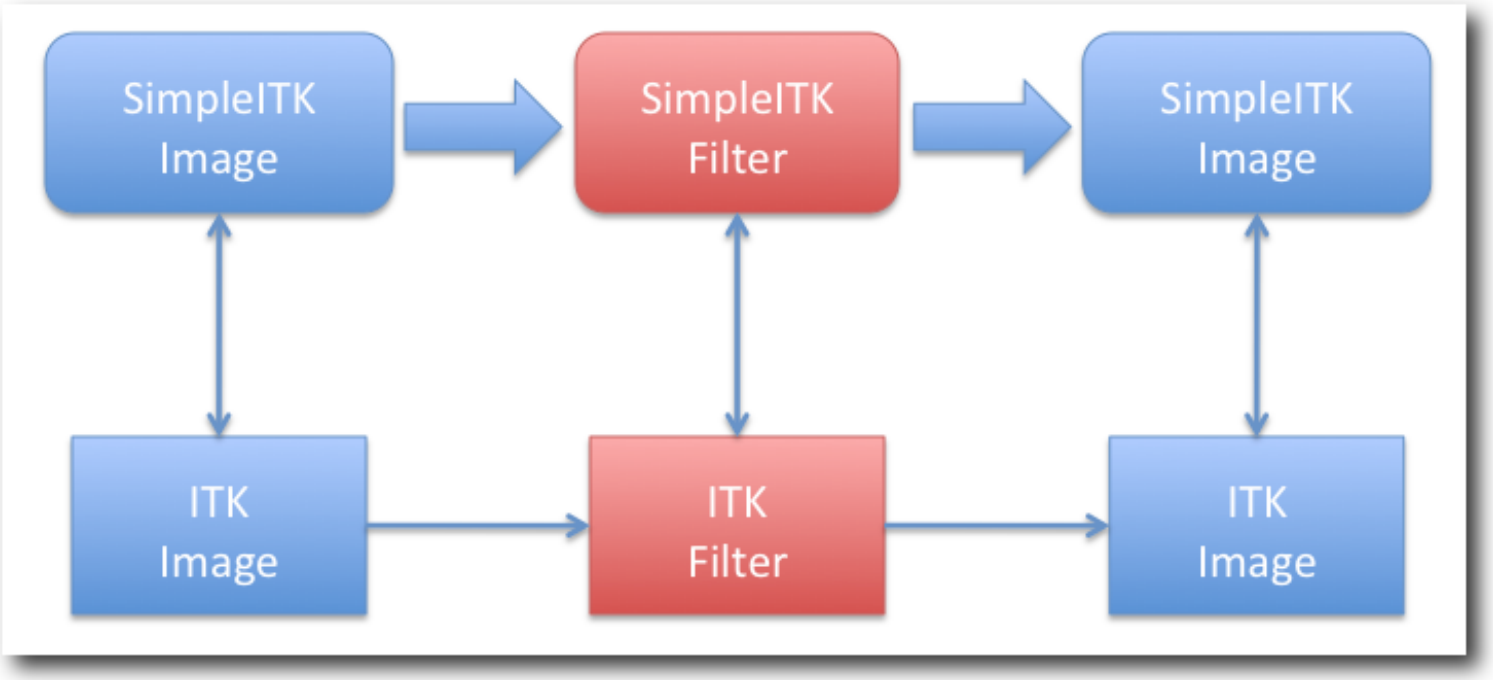
- Check out the code from GitHub (`https://github.com/SimpleITK/SimpleITK`)
- Run CMake (`http://www.cmake.org/`) using SimpleITK/SuperBuild as the source directory
- Build using your favorite compiler

Supported Platforms

- Windows: Visual Studio 10
- Windows: Visual Studio 9 (Requires TR1 service pack)
- Mac OSX: gcc 4.x (Xcode 4.x)
- Linux: gcc 4.x

SimpleITK Architecture

Filter Anatomy

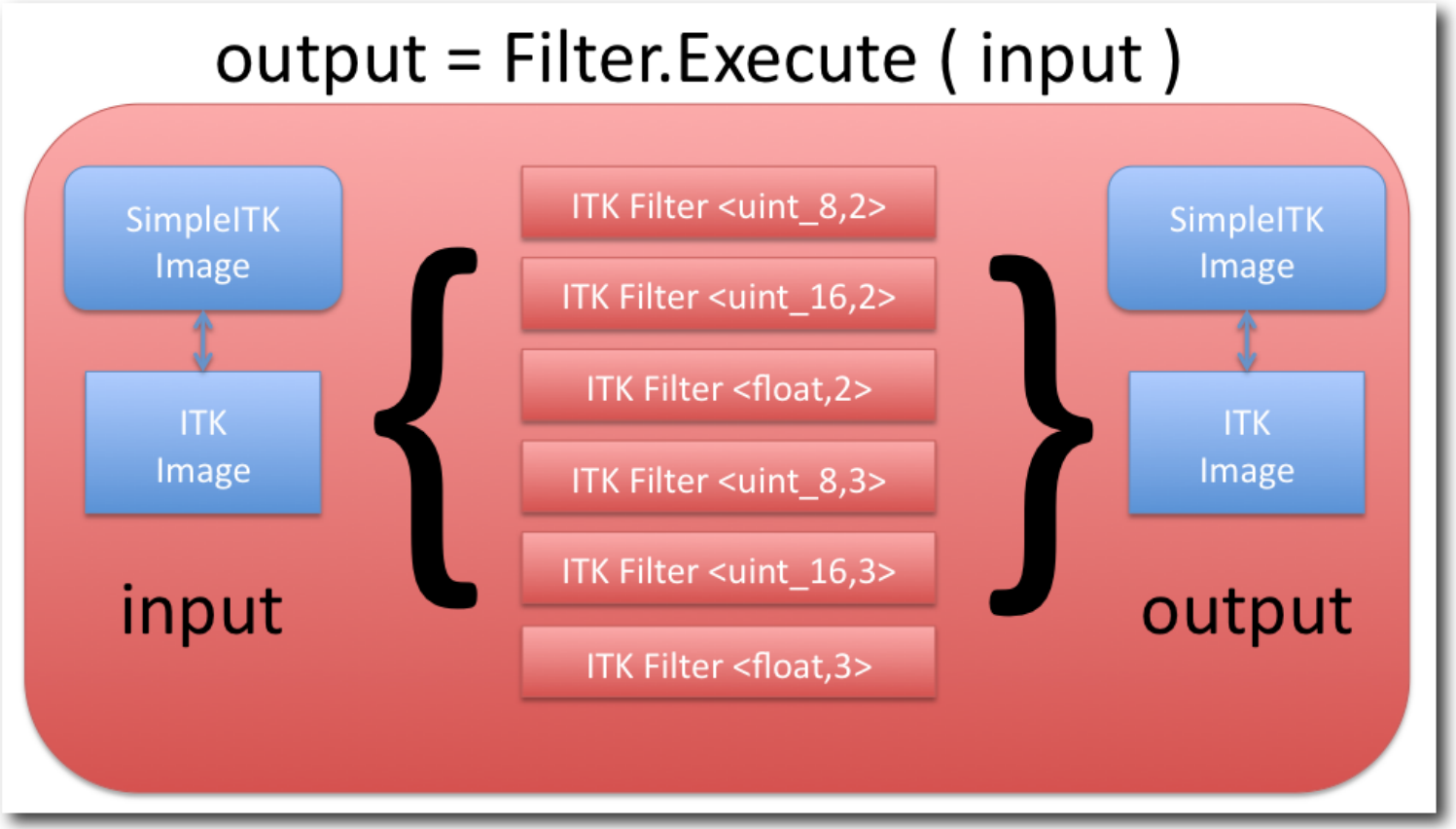


- SimpleITK filters create ITK filters
- Templated based on input type
- Output type is usually the same as input type
- Instantiated for many possible image types

Image and Filter Types

- Dimensions
  - 2 dimensional
  - 3 dimensional
- Scalar types
  - *int*8_*t*
  - *uint*8_*t*
  - *int*16_*t*
  - *uint*16_*t*
  - *int*32_*t*
  - *uint*32_*t*
  - *float*
  - *double*
  - *std* :: *complex* < *float* >
  - *std* :: *complex* < *double* >
- Vector Types
  - *int*8_*t*
  - *uint*8_*t*
  - *int*16_*t*
  - *uint*16_*t*
  - *int*32_*t*
  - *uint*32_*t*
  - *float*
  - *double*
- Label Types
  - *uint*8_*t*
  - *uint*16_*t*
  - *uint*32_*t*

Filter Anatomy



- Filter interrogates *input*
- Instantiates proper ITK filter
- Executes ITK filter
- Constructs *output* from ITK image

Using Filters

Object Paradigm (C++)

```
#include <SimpleITK.h>
namespace sitk = itk::simple;
...
// Create a smoothing filter
sitk::SmoothingRecursiveGaussianImageFilter gaussian;

// Set a parameter
gaussian.SetSigma ( 2.0 );

// "Execute" the Filter
sitk::Image blurredImage = gaussian.Execute ( image );
```

Object Paradigm (C++)

Flexibility

```
1 #include <SimpleITK.h>
2 namespace sitk = itk::simple;
3 ...
4 // Create a smoothing filter
5 sitk::SmoothingRecursiveGaussianImageFilter gaussian;
6
7 // Set parameter(s), then execute
8 sitk::Image blurredImage = gaussian
9                              .SetSigma ( 2.0 )
10                             .Execute ( image );
```

Object Paradigm (C++)

```
#include <SimpleITK.h>
namespace sitk = itk::simple;
...
blurredImage = sitk::SmoothingRecursiveGaussianImageFilter()
                          .SetSigma ( 2.0 )
                          .SetRadius ( 5 )
                          .Execute ( image );
```

One line: create anonymous filter, set parameters, and execute

"Function" Paradigm (C++)

```
#include <SimpleITK.h>
namespace sitk = itk::simple;
...
// Call the function version
// NB: Drop the "ImageFilter"!
// Signature:
/*
    sitk::Image SmoothingRecursiveGaussian (
            const Image&,
            double inSigma = 1.0,
            bool inNormalizeAcrossScale = false );
*/
sitk::Image blurredImage = sitk::SmoothingRecursiveGaussian (
                                image,
                                2.0,
                                false );
```

Mix & Match (C++)

```
#include <SimpleITK.h>
namespace sitk = itk::simple;
...
// Get our gaussian ready
sitk::SmoothingRecursiveGaussianImageFilter gaussian;
gaussian.SetSigma ( 2.0 );

// What is the effect on the image
sitk::Image difference = sitk::Subtract (
                           image,
                           gaussian.Execute ( image )
                           );
sitk::Image difference2 = sitk::Subtract (
                        image,
                        sitk::SmoothingRecursiveGaussian (
                          image, 2.0
                          )
                        );
```

Code Philosophy

Filter Class Overview (C++)

```
1 class SmoothingRecursiveGaussianImageFilter :
2     public ImageFilter {
3   typedef SmoothingRecursiveGaussianImageFilter Self;
4
5   /** Default Constructor that takes no arguments
6       and initializes default parameters */
7   SmoothingRecursiveGaussianImageFilter();
```

- In line 1, we declare a subclass of ImageFilter
- Line 3 creates a special typedef for use later
- The default constructor is line 7 (never any parameters)

Filter Class Overview (C++) Continued

```
1  /** Define the pixels types supported by this filter */
2  typedef BasicPixelIDTypeList  PixelIDTypeList;
```

- Notice *PixelIDTypeList* in line 2
- Used to instantiate ITK filters
- Determines valid input image types
- *BasicPixelIDTypeList* expands to:
  - *int*8_*t*, *uint*8_*t*
  - *int*16_*t*, *uint*16_*t*
  - *int*32_*t*, *uint*32_*t*
  - *float*, *double*

Filter Class Overview (C++) Continued

```
1  Self& SetSigma ( double t ) { ... return *this; }
2  double GetSigma() { return this->m_Sigma; }
3
4  Self& SetNormalizeAcrossScale ( bool t ) { ... }
5  Self& NormalizeAcrossScaleOn() { ... }
6  Self& NormalizeAcrossScaleOff() { ... }
7
8  bool GetNormalizeAcrossScale() { ... }
```

- Get/Set parameters
- Set methods always return *Self*& (more later)
- Generally, a direct mapping to ITK
- Boolean parameters generate *On* and *Off* methods

Filter Class Overview (C++) Continued

```
1  /** Name of this class */
2  std::string GetName() const { ... }
3
4  /** Print ourselves out */
5  std::string ToString() const;
```

- Return the name and description of the filter

Filter Class Overview (C++) Continued

```
1   /** Execute the filter on the input image */
2   Image Execute ( const Image & );
3
4   /** Execute the filter with parameters */
5   Image Execute ( const Image &,
6     double inSigma,
7     bool inNormalizeAcrossScale );
8 };  /* End of class SmoothingRecursiveGaussian */
9
10 Image SmoothingRecursiveGaussian ( const Image& ,
11   double inSigma = 1.0,
12   bool inNormalizeAcrossScale = false );
```

- Run the filter on an image and return the result
- Notice extra function (line 10), adds flexibility
- Drop *ImageFilter* from class name to get function name

Questions?

Using ITK with SimpleITK

Problem: Use ITK from SimpleITK (or vice versa)

```
./ToITK input.nii output.nii
```

Steps:

- Load image using SimpleITK
- Filter using ITK
- Save using OpenCV

To ITK

Starting code: `ToITK/ToITK.cxx`

Directory:

`SimpleITK-MICCAI-2011-Tutorial/Examples/AdvancedTutorial`

```
1  namespace sitk = itk::simple;
2  ...
3    // Load the image via SimpleITK
4    sitk::Image sitkImage = sitk::ReadImage ( inputFilename );
5
6    // Construct the ITK Pipeline
7    // Link pipeline to SimpleITK
8    // Update pipeline
9    // Create output SimpleITK image
10   // Save image via SimpleITK
11   sitk::WriteImage ( sOutput, outputFilename );
12   return EXIT_SUCCESS;
```

ToITK – Step 1: Construct the ITK Pipeline

```
1   // Construct the ITK Pipeline
2   typedef itk::Image<float,3> ImageType;
3   typedef itk::MirrorPadImageFilter<ImageType,ImageType> PadFilterType;
4   PadFilterType::SizeType upperBound, lowerBound;
5
6   PadFilterType::Pointer pad = PadFilterType::New();
7   for ( unsigned int i = 0; i < 3; i++ )
8     {
9       upperBound[i] = sitkImage.GetSize()[i];
10      lowerBound[i] = sitkImage.GetSize()[i];
11    }
12  pad->SetPadUpperBound ( upperBound );
13  pad->SetPadLowerBound ( lowerBound );
```

ToITK – Step 2: Link pipeline to SimpleITK

```
1  // Link pipeline to SimpleITK
2  ImageType::Pointer inputImage = (ImageType*) sitkImage.GetImageBase();
3  pad->SetInput ( inputImage );
```

ToITK – Step 3: Update ITK Pipeline

```
1  // Update pipeline
2  pad->Update();
```

ToITK – Step 4: Create the SimpleITK output image

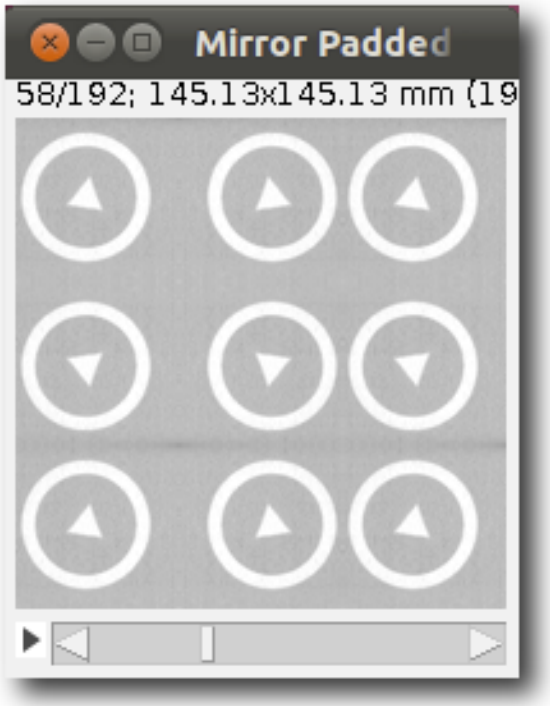
```
1  // Create output SimpleITK image
2  sitk::Image sOutput ( pad->GetOutput() );
```

ToITK – (Optional) Step 5: Show

```
1  // (Optional) Show the results
2  sitk::Show ( sOutput );
```

To ITK Solution

```
~/Source/AdvancedTutorial-build/ToITK/ToITKSolution \
~/Source/SimpleITK/Testing/Data/Input/RA-Float.nrrd \
/tmp/foo.nii
```



Simple Gaussian in 8 languages

`AdvancedTutorial/SimpleGaussian/SimpleGaussian.*`

```cpp
#include <SimpleITK.h>
namespace sitk = itk::simple;
int main ( int argc, char* argv[] ) {
  // Read the image file
  sitk::ImageFileReader reader;
  reader.SetFileName ( std::string ( argv[1] ) );
  sitk::Image image = reader.Execute();

  // This filters perform a gaussian bluring with sigma in physical
  // space. The output image will be of real type.
  sitk::SmoothingRecursiveGaussianImageFilter gaussian;
  gaussian.SetSigma ( atof ( argv[2] ) );
  sitk::Image blurredImage = gaussian.Execute ( image );

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

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

  return 0;
}
```

Ruby

```ruby
require 'simpleitk'

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

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

inputPixelType = image.get_pixel_idvalue

gaussian = Simpleitk::SmoothingRecursiveGaussianImageFilter.new
gaussian.set_sigma ARGV[1].to_f
image = gaussian.execute image;

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

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

R

```
# Run with:
#
# Rscript --vanilla SimpleGaussian.R  input sigma output
#

library(SimpleITK)

args <- commandArgs( TRUE )

myreader <- ImageFileReader()
myreader <- ImageFileReader_SetFileName( myreader, args[[1]] )
myimage <- ImageFileReader_Execute( myreader )

pixeltype <- Image_GetPixelIDValue( myimage )

myfilter <- SmoothingRecursiveGaussianImageFilter()
myfilter <- SmoothingRecursiveGaussianImageFilter_SetSigma( myfilter, as.real(args[2]) )
smoothedimage <- SmoothingRecursiveGaussianImageFilter_Execute( myfilter, myimage )

mycaster <- CastImageFilter()
mycaster <- CastImageFilter_SetOutputPixelType( mycaster, pixeltype )
castedimage <- CastImageFilter_Execute( mycaster, soothedimage )

mywriter <- ImageFileWriter()
mywriter <- ImageFileWriter_SetFileName( mywriter, args[[3]] )
mywriter <- ImageFileWriter_Execute( mywriter, castedimage )
```

C#

```
using System;
using itk.simple;

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

                // Execute Gaussian smoothing filter
                SmoothingRecursiveGaussianImageFilter gaussian = new SmoothingRecursiveG
                gaussian.SetSigma(Double.Parse(args[1]));
                image = gaussian.Execute(image);

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

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

Java

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

class SimpleGaussian {

  public static void main(String argv[]) {

    if ( argv.length < 3 ) {
      System.out.println("Usage: java SimpleGaussian <input> <sigma> <output>");
      return;
    }

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

    SmoothingRecursiveGaussianImageFilter filter =
      new SmoothingRecursiveGaussianImageFilter();
    filter.setSigma( Double.valueOf( argv[1] ).doubleValue() );
    Image blurredImg = filter.execute(img);

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

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

  }

}
```

Lua

```
1
2
3 if #arg < 3 then
4   print ( "Usage: SimpleGaussian <input> <sigma> <output>" )
5   os.exit ( 1 )
6 end
7
8 reader = SimpleITK.ImageFileReader()
9 -- Remember that Lua arrays are 1-based,
10 -- and that arg does not contain the application name!
11 reader:SetFileName ( arg[1] )
12 image = reader:Execute();
13
14 inputPixelType = image:GetPixelIDValue()
15
16 gaussian = SimpleITK.SmoothingRecursiveGaussianImageFilter()
17 gaussian:SetSigma ( arg[2] )
18 image = gaussian:Execute ( image );
19
20 caster = SimpleITK.CastImageFilter();
21 caster:SetOutputPixelType( inputPixelType );
22 image = caster:Execute( image )
23
24 writer = SimpleITK.ImageFileWriter()
25 writer:SetFileName ( arg[3] )
26 writer:Execute ( image );
```

Python

```
1
2 import SimpleITK as sitk
3 import sys
4
5 if len ( sys.argv ) < 4:
6     print "Usage: SimpleGaussian <input> <sigma> <output>";
7     sys.exit ( 1 )
8
9
10 reader = sitk.ImageFileReader()
11 reader.SetFileName ( sys.argv[1] )
12 image = reader.Execute()
13
14 pixelID = image.GetPixelIDValue()
15
16 gaussian = sitk.SmoothingRecursiveGaussianImageFilter()
17 gaussian.SetSigma ( float ( sys.argv[2] ) )
18 image = gaussian.Execute ( image )
19
20 caster = sitk.CastImageFilter()
21 caster.SetOutputPixelType( pixelID )
22 image = caster.Execute( image )
23
24 writer = sitk.ImageFileWriter()
25 writer.SetFileName ( sys.argv[3] )
26 writer.Execute ( image );
```

Tcl

```
1
2 if { $argc < 3 } {
3     puts "Usage: SimpleGaussian <input> <sigma> <output>"
4     exit 1
5 }
6
7 ImageFileReader reader
8 reader SetFileName [ lindex $argv 0 ]
9 set image [ reader Execute ]
10
11 set pixelID [ $image GetPixelIDValue ]
12
13 SmoothingRecursiveGaussianImageFilter gaussian
14 gaussian SetSigma [ lindex $argv 1 ]
15 set image [ gaussian Execute $image ]
16
17 CastImageFilter caster
18 caster SetOutputPixelType $pixelID
19 set image [ caster Execute $image ]
20
21 ImageFileWriter writer
22 writer SetFileName [ lindex $argv 2]
23 writer Execute $image
24
25 # Tcl requires explicit cleanup Cleanup
26 reader -delete
27 gaussian -delete
28 caster -delete
29 $image -delete
30 writer -delete
```

Conclusion

Wrap-up

- Gentle introduction to ITK
- Introduce SimpleITK
- Provide hands-on experience
- Problem solving, not direction following

Where to go next

Some resources for using and extending SimpleITK

- Home Page
  - `http://www.simpleitk.org`
- Documentation
  - `http://www.itk.org/SimpleITKDoxygen/html/`
- Conventions
  - `http://www.itk.org/SimpleITKDoxygen/html/Conventions.html`
- Contributions
  - `http://www.itk.org/SimpleITKDoxygen/html/Developer.html`