

# ITK Workshop

*A bird's eye view on developing ITK applications*



Alp Kucukelbir  
<http://proditus.com>

Yale University

June 3, 2010

# Outline

## Introduction

- What is ITK?
- Installing ITK
- CMake
- Using an IDE with ITK

## The Basics

- Generic Programming
- Smart Pointers and Object Factories
- Filters
- The Pipeline
- Recap (Thinking in ITK)

## Basic Filtering

- Reading Images
- Gradient Magnitude
- Writing Images

## Segmentation

- Pre-processing an Image
- Initializing a Level Set
- Level Set Segmentation

## Registration

- The Framework
- Metrics and Transforms
- Translation Registration in 2D

# Introduction

# What is ITK?

## ■ ITK is...

- Image Processing
- Segmentation
- Registration
- Image Statistics
- Open Source
- C++
- Templated

## ■ ITK is **not**...

- Visualization
- Graphics or User Interfaces (GUIs)
- MATLAB
- VTK
- Easy for the novice programmer
- A fiend out to get you
- A joke

# What is ITK?

## ■ ITK is...

- Image Processing
- Segmentation
- Registration
- Image Statistics
- Open Source
- C++
- Templated

## ■ ITK is **not**...

- Visualization
- Graphics or User Interfaces (GUIs)
- MATLAB
- VTK
- Easy for the novice programmer
- A fiend out to get you
- A joke

# What do I need to Install ITK?

## ■ A Computer

- Linux
- Mac OS X
- Windows (Visual Studio)
- Windows (Cygwin)

## ■ A Recent C++ Compiler

- GCC (2.95+)
- Visual Studio (6.0+)
- Intel (7.1+)
- Borland (5.5)

## ■ A Cross-Platform Build System

- Need to compile and build your ITK application.
- Each compiler/platform has its own way.
- ITK is complex. Want to make this as easy as possible.

## ■ CMake

- A cross-platform solution.
- Also written and provided by Kitware.

# What do I need to Install ITK?

- A Computer
  - Linux
  - Mac OS X
  - Windows (Visual Studio)
  - Windows (Cygwin)
- A Recent C++ Compiler
  - GCC (2.95+)
  - Visual Studio (6.0+)
  - Intel (7.1+)
  - Borland (5.5)
- A Cross-Platform Build System
  - Need to compile and build your ITK application.
  - Each compiler/platform has its own way.
  - ITK is complex. Want to make this as easy as possible.
- CMake
  - A cross-platform solution.
  - Also written and provided by Kitware.

# CMake (<http://www.cmake.org>)

- CMake is ...
  - A program you also need to install.
  - Cross-platform (Linux, Mac, Windows).
  - A build system for your ITK applications.
- CMake uses ...
  - Platform and compiler independent configuration files.
  - (Write it once.)
  - (Compile everywhere.)
- CMake generates ...
  - Native makefiles and workspaces:
    - make/gcc
    - Visual Studio
    - Eclipse
- CMake is ...
  - Necessary to build ITK itself!

# CMake (<http://www.cmake.org>)

- CMake is ...
  - A program you also need to install.
  - Cross-platform (Linux, Mac, Windows).
  - A build system for your ITK applications.
- CMake uses ...
  - Platform and compiler independent configuration files.
  - (Write it once.)
  - (Compile everywhere.)
- CMake generates ...
  - Native makefiles and workspaces:
    - make/gcc
    - Visual Studio
    - Eclipse
- CMake is ...
  - Necessary to build ITK itself!

# Using an IDE with ITK

- First: Install ITK
  - Configure CMake.
  - Generate make/workspace files for ITK.
  - Build ITK.
  - See Ch.2. of ITK Guide or the first tutorial for ITK.
- (... or ...)
  - Switch to Ubuntu and download binaries.
  - Check out Paul Novotny's website. (<http://www.paulnovo.org>)
- Second: Set-up your own ITK application
  - Write your .h and .cxx/.ttx files.
  - Write a CMake file.
  - Run CMake on your project.
  - Import project files into...
    - Visual Studio (Windows)
    - Eclipse (Linux/Mac)
- See (<http://proditus.com/itk>) for resources mentioned in this slide.

# Using an IDE with ITK

- First: Install ITK
  - Configure CMake.
  - Generate make/workspace files for ITK.
  - Build ITK.
  - See Ch.2. of ITK Guide or the first tutorial for ITK.
- (... or ...)
  - Switch to Ubuntu and download binaries.
  - Check out Paul Novotny's website. (<http://www.paulnovo.org>)
- Second: Set-up your own ITK application
  - Write your .h and .cxx/.txx files.
  - Write a CMake file.
  - Run CMake on your project.
  - Import project files into...
    - Visual Studio (Windows)
    - Eclipse (Linux/Mac)
- See (<http://proditus.com/itk>) for resources mentioned in this slide.

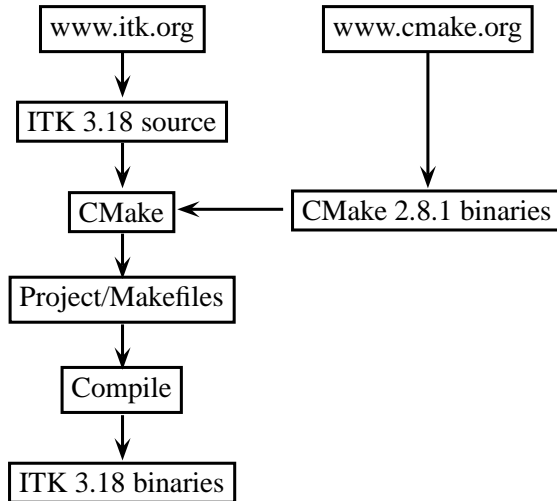


Figure: Installing ITK

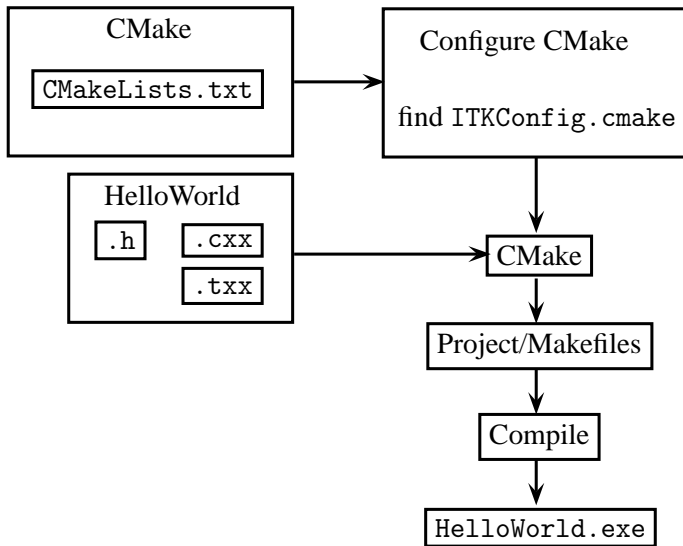


Figure: First ITK Program

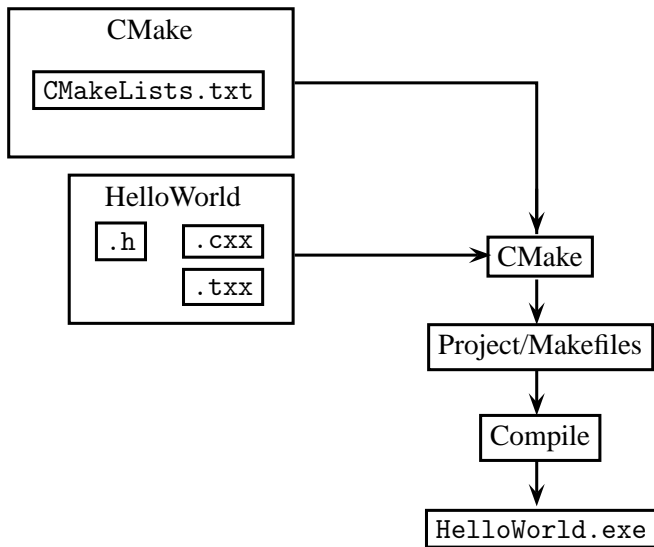


Figure: Following ITK Programs

# Things I left out

- CMakeLists.txt files

- Used to set up project.
- Generate make/workspace files for your program.
- Quite simple for basic projects.
- Will be provided along with examples later.

- ITK specific header files

- See full example files later.

- IDE specific tasks

- You should use whatever is most comfortable.
- Common choices are:
  - Visual Studio (Windows)
  - Eclipse (Linux/Mac)
- Follow basic tutorials to learn.

# Things I left out

- CMakeLists.txt files
  - Used to set up project.
  - Generate make/workspace files for your program.
  - Quite simple for basic projects.
  - Will be provided along with examples later.
- ITK specific header files
  - See full example files later.
- IDE specific tasks
  - You should use whatever is most comfortable.
  - Common choices are:
    - Visual Studio (Windows)
    - Eclipse (Linux/Mac)
  - Follow basic tutorials to learn.

# The Basics

# Generic Programming

- Generic Programming Ideas

- Generic Algorithms
- Adaptive and Efficient Plug-and-play
- *Containers, Iterators, etc.*

- C++

- Templates
- Standard Template Library (STL)

- C++ Templating

- Write code in terms of unknown template T.
- T can be:
  - A Native Type (int, float)
  - A User-Defined Type (myClass, myImage)
- Templates are resolved at **compile-time**.

# Generic Programming

## ■ Generic Programming Ideas

- Generic Algorithms
- Adaptive and Efficient Plug-and-play
- *Containers, Iterators, etc.*

## ■ C++

- Templates
- Standard Template Library (STL)

## ■ C++ Templating

- Write code in terms of unknown template T.
- T can be:
  - A Native Type (int, float)
  - A User-Defined Type (myClass, myImage)
- Templates are resolved at **compile-time**.

# Generic Programming

- Generic Programming Ideas

- Generic Algorithms
- Adaptive and Efficient Plug-and-play
- *Containers, Iterators, etc.*

- C++

- Templates
- Standard Template Library (STL)

- C++ Templating

- Write code in terms of unknown template T.
- T can be:
  - A Native Type (`int`, `float`)
  - A User-Defined Type (`myClass`, `myImage`)
- Templates are resolved at **compile-time**.

## Generic Programming (STL Example)

- STL Vector

*// Create a vector with 3 entries of 100 as integers*

```
vector< int > myIntegerVector (3, 100);
```

*// Create a vector with 3 entries of 3.14 as floats*

```
vector< float > myFloatVector (3, 3.14);
```

## Generic Programming (ITK Example)

- ITK Image

```
// Define custom type PixelType [0-255]  
typedef unsigned char PixelType;  
  
// Define Dimension  
const int Dimension = 2;  
  
// Define an ITK image type  
typedef itk::Image< PixelType, Dimension > ImageType;
```

# Generic Programming

## ■ Advantages

- Highly adaptive code.
- True Plug-and-play.
- Compile-time feedback.

## ■ Disadvantages

- Code can look complicated.
- Bad coding practices can make syntax look awful.
- Challenging to wrap templated C++ code.

## ■ A good reference book:

- M. H. Austern. *Generic Programming and the STL*. Professional Computing Series. Addison-Wesley, 1999.

# Generic Programming

## ■ Advantages

- Highly adaptive code.
- True Plug-and-play.
- Compile-time feedback.

## ■ Disadvantages

- Code can look complicated.
- Bad coding practices can make syntax look awful.
- Challenging to wrap templated C++ code.

## ■ A good reference book:

- M. H. Austern. *Generic Programming and the STL*. Professional Computing Series. Addison-Wesley, 1999.

# Generic Programming

## ■ Advantages

- Highly adaptive code.
- True Plug-and-play.
- Compile-time feedback.

## ■ A good reference book:

- M. H. Austern. *Generic Programming and the STL*. Professional Computing Series. Addison-Wesley, 1999.

## ■ Disadvantages

- Code can look complicated.
- Bad coding practices can make syntax look awful.
- Challenging to wrap templated C++ code.

# Smart Pointers and Object Factories

## ■ Smart Pointers

- ITK likes to take care of its own memory.
- Many ways to automate memory management.
- ITK uses reference counting.

## ■ Reference Counting

- No need to call `Delete()`.
- Pointers exist within scope (like regular variables).
- Unlike Garbage Collection, the coder has control.

## ■ Object Factories

- ITK classes are instantiated through an *object factory* mechanism.
- Must use ITK's Smart Pointers.
- Must use ITK's static `New()` class.
- Cannot (usually) construct anything on the heap.
- Most of the time, this is totally transparent.

# Smart Pointers and Object Factories

## ■ Smart Pointers

- ITK likes to take care of its own memory.
- Many ways to automate memory management.
- ITK uses reference counting.

## ■ Reference Counting

- No need to call `Delete()`.
- Pointers exist within scope (like regular variables).
- Unlike Garbage Collection, the coder has control.

## ■ Object Factories

- ITK classes are instantiated through an *object factory* mechanism.
- Must use ITK's Smart Pointers.
- Must use ITK's static `New()` class.
- Cannot (usually) construct anything on the heap.
- Most of the time, this is totally transparent.

# Smart Pointers and Object Factories

## ■ Smart Pointers

- ITK likes to take care of its own memory.
- Many ways to automate memory management.
- ITK uses reference counting.

## ■ Reference Counting

- No need to call `Delete()`.
- Pointers exist within scope (like regular variables).
- Unlike Garbage Collection, the coder has control.

## ■ Object Factories

- ITK classes are instantiated through an *object factory* mechanism.
- Must use ITK's Smart Pointers.
- Must use ITK's static `New()` class.
- Cannot (usually) construct anything on the heap.
- Most of the time, this is totally transparent.

## Smart Pointers and Object Factories (Example)

- ITK Image (*continued*)

```
// Define an ITK image type
typedef itk::Image< PixelType, Dimension > ImageType;

try
{
    // Instantiate a new image
    ImageType::Pointer myImage = ImageType::New();
}
catch ( itk::ExceptionObject exp )
{
    // Catch an ITK exception here
}
```

# Filters

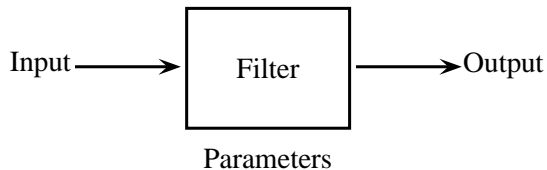


Figure: ITK Filter Structure

## ■ Signals and Systems

- ITK filters are *process objects*.
- Can think of in terms of block diagrams.
- Can think of as (literal) tools.

## ■ Filters can be...

- Image Readers/Writers
- Image Processors (e.g. Thresholding)
- Transforms
- *etc.*

# Filters

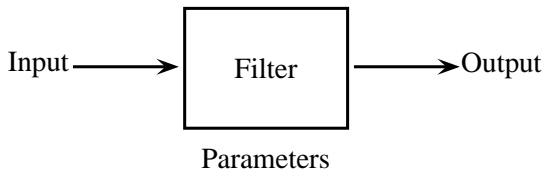


Figure: ITK Filter Structure

## ■ Signals and Systems

- ITK filters are *process objects*.
- Can think of in terms of block diagrams.
- Can think of as (literal) tools.

## ■ Filters can be...

- Image Readers/Writers
- Image Processors (e.g. Thresholding)
- Transforms
- *etc.*

## Filters (Example)

- ITK Image Reader

```
typedef itk::Image< PixelType, Dimension > ImageType;

// Define an ITK image reader type
typedef itk::ImageFileReader< ImageType > ReaderType;

// Instantiate image and image reader
ImageType::Pointer    myImage    = ImageType::New();
ReaderType::Pointer  myReader    = ReaderType::New();

// Set inputs, outputs, and parameters
myReader->SetFileName( "someImage.tiff" );

myImage = myReader->GetOutput();
```

## The Pipeline

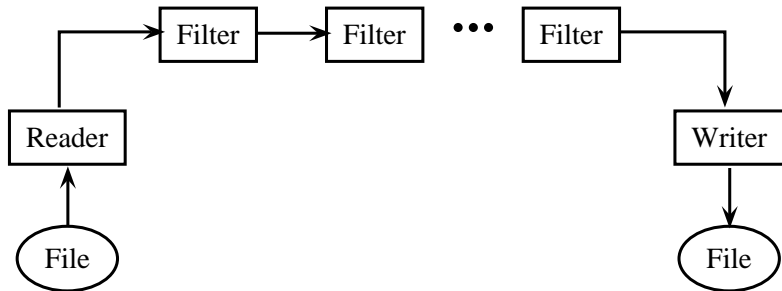


Figure: ITK Pipeline

- The "glue" that brings together all the components of your program.
- Coarsely looks like the above for most image processing tasks.

# Recap (Thinking in ITK)

## ■ Things you need

- ITK binaries.
- CMake for projects.
- C++ compiler/IDE.

## ■ Generic Programming

- Use types and typedef.
- Use custom iterators and containers.
- Use ITK consistent notation/syntax.

## ■ Smart Pointers and Object Factories

- Use ITK's Smart Pointers and static New() class.
- Be careful about typos with types here.

## ■ Filters and the Pipeline

- Filters are your tools.
- The Pipeline is your work bench.
- Signals and Systems (Block Diagrams) thought process.

# Recap (Thinking in ITK)

## ■ Things you need

- ITK binaries.
- CMake for projects.
- C++ compiler/IDE.

## ■ Generic Programming

- Use types and typedef.
- Use custom iterators and containers.
- Use ITK consistent notation/syntax.

## ■ Smart Pointers and Object Factories

- Use ITK's Smart Pointers and static New() class.
- Be careful about typos with types here.

## ■ Filters and the Pipeline

- Filters are your tools.
- The Pipeline is your work bench.
- Signals and Systems (Block Diagrams) thought process.

# Recap (Thinking in ITK)

## ■ Things you need

- ITK binaries.
- CMake for projects.
- C++ compiler/IDE.

## ■ Generic Programming

- Use types and typedef.
- Use custom iterators and containers.
- Use ITK consistent notation/syntax.

## ■ Smart Pointers and Object Factories

- Use ITK's Smart Pointers and static New() class.
- Be careful about typos with types here.

## ■ Filters and the Pipeline

- Filters are your tools.
- The Pipeline is your work bench.
- Signals and Systems (Block Diagrams) thought process.

## Recap (Thinking in ITK)

- Things you need
  - ITK binaries.
  - CMake for projects.
  - C++ compiler/IDE.
- Generic Programming
  - Use types and typedef.
  - Use custom iterators and containers.
  - Use ITK consistent notation/syntax.
- Smart Pointers and Object Factories
  - Use ITK's Smart Pointers and static New() class.
  - Be careful about typos with types here.
- Filters and the Pipeline
  - Filters are your tools.
  - The Pipeline is your work bench.
  - Signals and Systems (Block Diagrams) thought process.

# Basic Filtering

## Gradient Magnitude (Pipeline)

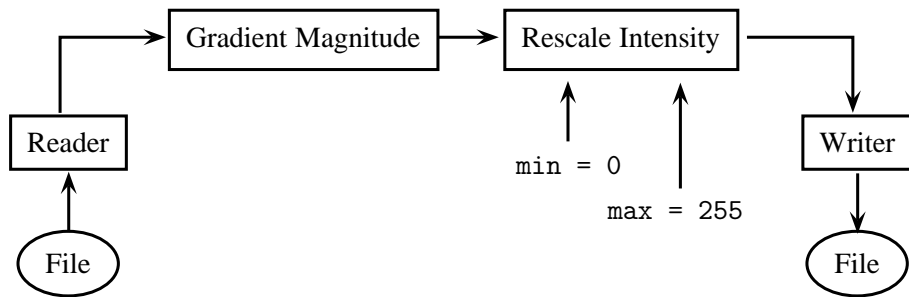


Figure: Gradient Magnitude Pipeline Diagram

## Gradient Magnitude (Code)

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkGradientMagnitudeImageFilter.h"

int main( int argc, char * argv[] )
{
    if( argc < 3 )
    {
        std::cerr << "Usage: " << std::endl;
        std::cerr << argv[0] << "  inputImageFile  outputImageFile " << std::endl;
        return EXIT_FAILURE;
    }
}
```

```
/*=====Type Definitions=====*/
// Pixel type definitions
typedef float InputPixelType;
typedef float OutputPixelType;
typedef unsigned char WritePixelType;

// Define dimension of image to be processed.
const unsigned int Dimension = 2;

// The image types are defined using their respective pixel types.
typedef itk::Image< InputPixelType, Dimension > InputImageType;
typedef itk::Image< OutputPixelType, Dimension > OutputImageType;
typedef itk::Image< WritePixelType, Dimension > WriteImageType;

// Define the image reader type.
typedef itk::ImageFileReader< InputImageType > ReaderType;
```

```
// The type of the gradient magnitude filter is defined by the  
// input image and the output image types.  
typedef itk::GradientMagnitudeImageFilter<  
    InputImageType, OutputImageType >    GMFilterType;  
  
// Define the image rescaler filter type.  
typedef itk::RescaleIntensityImageFilter<  
    OutputImageType, WriteImageType >    RescaleFilterType;  
  
// Define the image writer type.  
typedef itk::ImageFileWriter< WriteImageType >    WriterType;
```

```
/*=====Filter Instantiations and Set-up=====*/

// Instantiate reader and set the input filename (from command line input).
ReaderType::Pointer reader = ReaderType::New();
reader->SetFileName( argv[1] );

// Instantiate gradient magnitude filter.
GMFilterType::Pointer gradMagFilter = GMFilterType::New();

// Instantiate rescaler filter and set range of output intensities.
RescaleFilterType::Pointer rescaler = RescaleFilterType::New();
rescaler->SetOutputMinimum( 0 );
rescaler->SetOutputMaximum( 255 );

// Instantiate writer and set the output filename (from command line input).
WriterType::Pointer writer = WriterType::New();
writer->SetFileName( argv[2] );
```

```
/*=====Pipeline Set-up=====*/
```

```
// Connect the filters together.
```

```
gradMagFilter->SetInput( reader->GetOutput() );
```

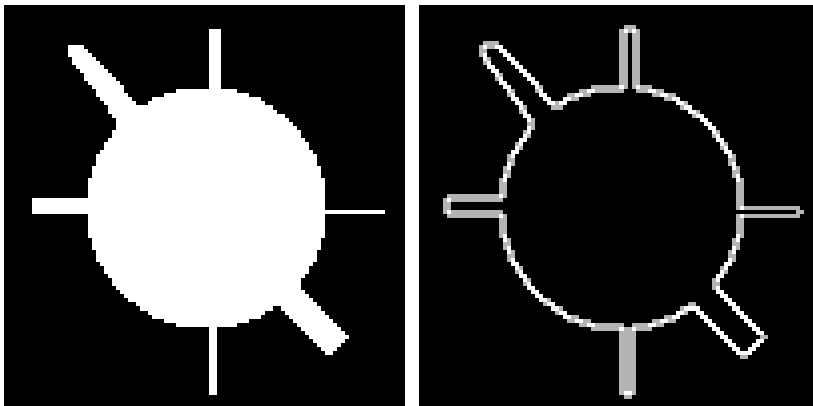
```
rescaler->SetInput( gradMagFilter->GetOutput() );
```

```
writer->SetInput( rescaler->GetOutput() );
```

```
/*=====Evoke Pipeline ( Update() )=====*/
try
{
    writer->Update();
}
catch( itk::ExceptionObject & excep )
{
    std::cerr << "Exception caught!" << std::endl;
    std::cerr << excep << std::endl;
}

return EXIT_SUCCESS;
}
```

## Gradient Magnitude (Results)



(a) Input

(b) Output

Figure: Gradient Magnitude Program Results

# Segmentation (Level-Set)

## Segmentation (Pipeline)

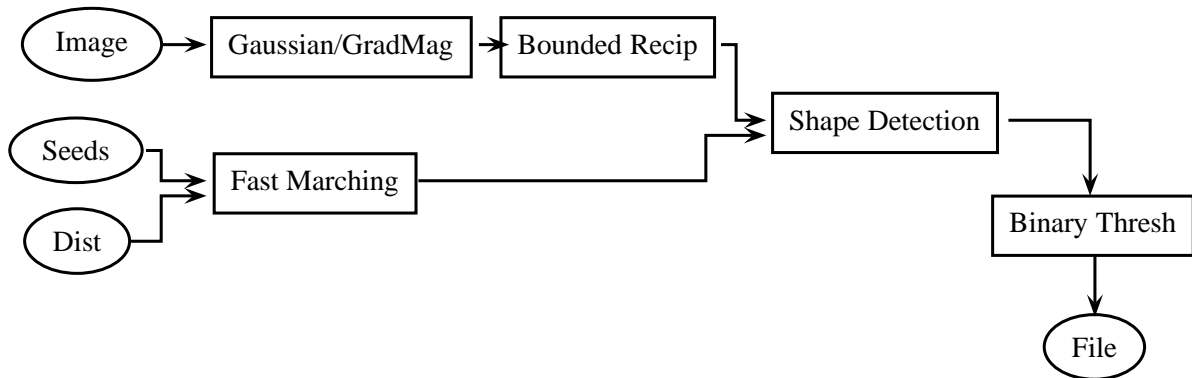


Figure: Segmentation Pipeline Diagram

## Segmentation (Code)

```
#include "itkImage.h"  
#include "itkImageFileReader.h"  
#include "itkImageFileWriter.h"  
  
#include "itkGradientMagnitudeRecursiveGaussianImageFilter.h"  
#include "itkBoundedReciprocalImageFilter.h"  
  
#include "itkFastMarchingImageFilter.h"  
  
#include "itkShapeDetectionLevelSetImageFilter.h"  
  
#include "itkBinaryThresholdImageFilter.h"  
#include "itkRescaleIntensityImageFilter.h"
```

```
int main( int argc, char *argv[] )
{
    if( argc < 6 )
    {
        std::cerr << "Missing Parameters " << std::endl;
        std::cerr << "Usage: " << argv[0];
        std::cerr << " inputImage  outputImage";
        std::cerr << " seedX seedY InitialDistance" << std::endl;
        return EXIT_FAILURE;
    }
}
```

```
/*=====Type Definitions=====*/

// Pixel type definitions
typedef float InternalPixelType;
typedef unsigned char OutputPixelType;

// Define dimension of image to be processed.
const unsigned int Dimension = 2;

// The image types are defined using their respective pixel types.
typedef itk::Image< InternalPixelType, Dimension > InternalImageType;
typedef itk::Image< OutputPixelType, Dimension > OutputImageType;

// Define the image reader and writer type.
typedef itk::ImageFileReader< InternalImageType > ReaderType;
typedef itk::ImageFileWriter< OutputImageType > WriterType;
```

```
// Define gradient magnitude recursive gaussian filter type.
typedef itk::GradientMagnitudeRecursiveGaussianImageFilter <
    InternalImageType, InternalImageType > GradientFilterType;

// Define bounded reciprocal filter type.
typedef itk::BoundedReciprocalImageFilter <
    InternalImageType, InternalImageType > ReciprocalFilterType;

// Define fast marching (initial level-set) filter type.
typedef itk::FastMarchingImageFilter <
    InternalImageType, InternalImageType > FastMarchingFilterType;
```

```
// Define shape detection (level-set segmentation) filter type.
typedef itk::ShapeDetectionLevelSetImageFilter<
    InternalImageType, InternalImageType > ShapeDetectionFilterType;

// Define binary thresholding filter type.
typedef itk::BinaryThresholdImageFilter<
    InternalImageType, OutputImageType > ThresholdingFilterType;
```

```
/*=====Parameters=====*/  
// Sigma parameter for Gradient Magnitude Recursive Gaussian Filter  
const double sigma = 0.5;  
  
// Curvature and Propagation scaling parameters for Shape Detection Filter  
const double curvatureScaling = 0.01;  
const double propagationScaling = 1.0;  
  
// Stopping rule and maximum number of iterations for segmentation.  
const double RMSE = 0.04;  
const unsigned int maxIter = 800;
```

```
/*=====Filter Instantiations and Set-up=====*/  
  
// Instantiate reader and set the input filename (from command line input).  
ReaderType::Pointer reader = ReaderType::New();  
    reader->SetFileName( argv[1] );  
  
// Instantiate writer and set the output filename (from command line input).  
WriterType::Pointer writer = WriterType::New();  
    writer->SetFileName( argv[2] );
```

```
// Instantiate Gradient Magnitude filter and set its sigma parameter.  
GradientFilterType::Pointer gradientMagnitude = GradientFilterType::New();  
    gradientMagnitude->SetSigma( sigma );  
  
// Instantiate Bounded Reciprocal filter.  
ReciprocalFilterType::Pointer reciprocal = ReciprocalFilterType::New();
```

```
// Instantiate Fast Marching filter.
FastMarchingFilterType::Pointer fastMarching = FastMarchingFilterType::New();

// Define seed input to fastMarching Filter
typedef FastMarchingFilterType::NodeContainer NodeContainer;
typedef FastMarchingFilterType::NodeType NodeType;
NodeContainer::Pointer seeds = NodeContainer::New();

// Define seed position as (x,y) if in 2D
InternalImageType::IndexType seedPosition;

// Grab initial seed position and distance from input
seedPosition[0] = atoi( argv[3] );
seedPosition[1] = atoi( argv[4] );
const double initialDistance = atof( argv[5] );
```

```
NodeType node;
const double seedValue = - initialDistance;
node.SetValue( seedValue );
node.SetIndex( seedPosition );
seeds->Initialize();
seeds->InsertElement( 0, node );

// Feed seeds into fastMarching filter
fastMarching->SetTrialPoints( seeds );

// Configure fastMarching filter as a distance map generator
fastMarching->SetSpeedConstant( 1.0 );

// Configure image size (only valid after explicit Update())
reader->Update();
fastMarching->SetOutputSize(
    reader->GetOutput()->GetBufferedRegion().GetSize() );
```

```
// Initialize level-set segmentation filter and set parameters.
ShapeDetectionFilterType::Pointer
  shapeDetection = ShapeDetectionFilterType::New();

// Set propagation and curvature scaling parameters.
shapeDetection->SetPropagationScaling(  propagationScaling );
shapeDetection->SetCurvatureScaling(   curvatureScaling   );

// Set RMSE and maximum number of iterations.
shapeDetection->SetMaximumRMSError(    RMSE    );
shapeDetection->SetNumberOfIterations( maxIter );
```

```
// Initialize Binary Thresholding filter and set parameters.  
ThresholdingFilterType::Pointer thresholder = ThresholdingFilterType::New();  
  
// The upper threshold of the BinaryThresholdImageFilter is set  
// to 0.0 in order to display the zero set of the resulting level  
// set. The lower threshold is set to a large negative number in order to  
// ensure that the interior of the segmented object will appear  
// inside the binary region.  
thresholder->SetLowerThreshold( -1000.0 );  
thresholder->SetUpperThreshold( 0.0 );  
  
thresholder->SetOutsideValue( 0 );  
thresholder->SetInsideValue( 255 );
```

```
/*=====Pipeline Set-up=====*/
```

```
// Create the Edge Potential Image (output of 'reciprocal')
```

```
gradientMagnitude->SetInput( reader->GetOutput() );
```

```
reciprocal->SetInput( gradientMagnitude->GetOutput() );
```

```
// Initial Level-set already computed (output of 'fastMarching')
```

```
// Plug both into level-set segmentation algorithm ('shapeDetection')
```

```
shapeDetection->SetFeatureImage( reciprocal->GetOutput() );
```

```
shapeDetection->SetInput( fastMarching->GetOutput() );
```

```
// Connect segmentation output to binary thresholder ('thresholder')
```

```
thresholder->SetInput( shapeDetection->GetOutput() );
```

```
// Connect thresholder output to file writer ('writer')
```

```
writer->SetInput( thresholder->GetOutput() );
```

```
/*=====Evoke Pipeline ( Update() )=====*/  
try  
  {  
    writer->Update();  
  }  
catch( itk::ExceptionObject & excep )  
  {  
    std::cerr << "Exception caught !" << std::endl;  
    std::cerr << excep << std::endl;  
  }
```

```
// Print out some useful information
std::cout << std::endl;
std::cout << "Max. no. iterations: ";
std::cout << shapeDetection->GetNumberOfIterations() << std::endl;
std::cout << "Max. RMS error: ";
std::cout << shapeDetection->GetMaximumRMSError() << std::endl;
std::cout << std::endl;
std::cout << "No. elapsed iterations: ";
std::cout << shapeDetection->GetElapsedIterations() << std::endl;
std::cout << "RMS change: " << shapeDetection->GetRMSChange() << std::endl;

return EXIT_SUCCESS;
}
```

## Segmentation (Results)



(a) Input

(b) Output

**Figure:** Level-Set Segmentation Results

# Registration

## Registration (Framework)

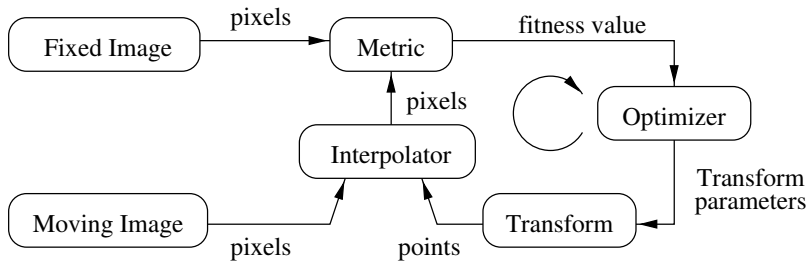


Figure: Registration Framework (Figure 8.2 from the ITK Software Guide)

# Metrics

## ■ Metrics

- Mean Squares
- Normalized Correlation
- Mutual Information (Wells or Mattes)
- Kullback Lieber
- Correlation Coefficient
- Kappa Statistics
- Gradient Difference

## ■ Transforms

- Translation
- Scale
- Rigid (Euler or Centered) (2D or 3D)
- Rigid + Projection (3D)
- Affine (2D or 3D)
- BSpline Deformable
- Kernel

# Metrics

## ■ Metrics

- Mean Squares
- Normalized Correlation
- Mutual Information (Wells or Mattes)
- Kullback Lieber
- Correlation Coefficient
- Kappa Statistics
- Gradient Difference

## ■ Transforms

- Translation
- Scale
- Rigid (Euler or Centered) (2D or 3D)
- Rigid + Projection (3D)
- Affine (2D or 3D)
- BSpline Deformable
- Kernel

## Registration (Code)

```
// Basic ITK image libraries  
#include "itkImage.h"  
#include "itkImageFileReader.h"  
#include "itkImageFileWriter.h"  
  
// Libraries for registration  
#include "itkImageRegistrationMethod.h"  
#include "itkTranslationTransform.h"  
#include "itkMeanSquaresImageToImageMetric.h"  
#include "itkLinearInterpolateImageFunction.h"  
#include "itkRegularStepGradientDescentOptimizer.h"  
  
// Libraries for post-processing of output images  
#include "itkResampleImageFilter.h"  
#include "itkCastImageFilter.h"
```

```
int main( int argc, char *argv[] )
{
    if( argc < 4 )
    {
        std::cerr << "Missing Parameters " << std::endl;
        std::cerr << "Usage: " << argv[0];
        std::cerr << " fixedImageFile  movingImageFile";
        std::cerr << " outputImagefile" << std::endl;
        return EXIT_FAILURE;
    }
}
```

```
/*=====Type Definitions=====*/

// Dimension and Pixel type definitions
const   unsigned int   Dimension = 2;
typedef float          PixelType;

// The image types are defined with the same pixel type.
typedef itk::Image< PixelType, Dimension > FixedImageType;
typedef itk::Image< PixelType, Dimension > MovingImageType;

// The image readers are defined.
typedef itk::ImageFileReader< FixedImageType > FixedImageReaderType;
typedef itk::ImageFileReader< MovingImageType > MovingImageReaderType;
```

```
// The transform that will map the fixed image space into the moving image  
// space is defined.  
typedef itk::TranslationTransform< double, Dimension > TransformType;  
  
// An optimizer is required to explore the parameter space of the transform  
// in search of optimal values of the metric.  
typedef itk::RegularStepGradientDescentOptimizer OptimizerType;  
  
// The metric will compare how well the two images match each other. Metric  
// types are usually parameterized by the image types as it can be seen in  
// the following type declaration.  
typedef itk::MeanSquaresImageToImageMetric<  
    FixedImageType, MovingImageType > MetricType;
```

```
// Finally, the type of the interpolator is declared. The interpolator will  
// evaluate the intensities of the moving image at non-grid positions.  
typedef itk::LinearInterpolateImageFunction<  
    MovingImageType, double > InterpolatorType;  
  
// The registration method type is instantiated using the types of the  
// fixed and moving images. This class is responsible for interconnecting  
// all the components that we have described so far.  
typedef itk::ImageRegistrationMethod<  
    FixedImageType, MovingImageType > RegistrationType;
```

```
/*=====Filter Instantiations and Set-up=====*/
```

```
// Instantiate fixed image reader and set the input filename.
```

```
FixedImageReaderType::Pointer fixedImageReader =  
    FixedImageReaderType::New();  
fixedImageReader->SetFileName( argv[1] );
```

```
// Instantiate moving image reader and set the input filename.
```

```
MovingImageReaderType::Pointer movingImageReader =  
    MovingImageReaderType::New();  
movingImageReader->SetFileName( argv[2] );
```

*// Instantiate the filters defined in the previous section*

```
MetricType::Pointer      metric      = MetricType::New();  
TransformType::Pointer  transform    = TransformType::New();  
OptimizerType::Pointer  optimizer    = OptimizerType::New();  
InterpolatorType::Pointer interpolator = InterpolatorType::New();  
RegistrationType::Pointer registration = RegistrationType::New();
```

*// Connect each component to the instance of the registration method.*

```
registration->SetMetric(      metric      );  
registration->SetOptimizer(   optimizer   );  
registration->SetTransform(   transform   );  
registration->SetInterpolator( interpolator );
```

*// Connect the image readers to the registration method.*

```
registration->SetFixedImage(    fixedImageReader->GetOutput()    );
```

```
registration->SetMovingImage(    movingImageReader->GetOutput()    );
```

*// Set image region for registration*

```
fixedImageReader->Update();
```

```
registration->SetFixedImageRegion(  
    fixedImageReader->GetOutput()->GetBufferedRegion() );
```

```
// The parameters of the transform are initialized by passing them in an  
// array. This can be used to setup an initial known correction of the  
// misalignment. In this particular case, a translation transform is  
// being used for the registration.
```

```
typedef RegistrationType::ParametersType ParametersType;  
ParametersType initialParameters( transform->GetNumberOfParameters() );
```

```
initialParameters[0] = 0.0; // Initial offset in mm along X  
initialParameters[1] = 0.0; // Initial offset in mm along Y
```

```
registration->SetInitialTransformParameters( initialParameters );
```

```
// Set-up optimizer parameters
```

```
optimizer->SetMaximumStepLength( 4.00 );  
optimizer->SetMinimumStepLength( 0.01 );  
optimizer->SetNumberOfIterations( 200 );
```

```
/*=====Evoked Registration ( Update() )=====*/
try
{
  registration->Update();
}
catch( itk::ExceptionObject & err )
{
  std::cerr << "ExceptionObject caught !" << std::endl;
  std::cerr << err << std::endl;
  return EXIT_FAILURE;
}
```

```
// The result of the registration process is an array of parameters that  
// defines the spatial transformation in an unique way.  
ParametersType finalParameters = registration->GetLastTransformParameters();  
const double TranslationAlongX = finalParameters[0];  
const double TranslationAlongY = finalParameters[1];  
const unsigned int numberOfIterations = optimizer->GetCurrentIteration();  
const double bestValue = optimizer->GetValue();  
  
// Print out results  
std::cout << "Result = " << std::endl;  
std::cout << " Translation X = " << TranslationAlongX << std::endl;  
std::cout << " Translation Y = " << TranslationAlongY << std::endl;  
std::cout << " Iterations      = " << numberOfIterations << std::endl;  
std::cout << " Metric value   = " << bestValue << std::endl;
```

```
/*===== (Output) Type Definitions =====*/
```

```
// Define image resampler filter type.
```

```
typedef itk::ResampleImageFilter<  
    MovingImageType, FixedImageType >    ResampleFilterType;
```

```
// Define output pixel type and image type
```

```
typedef unsigned char    OutputPixelType;  
typedef itk::Image< OutputPixelType, Dimension >    OutputImageType;
```

```
// Define cast image filter and image writer.
```

```
typedef itk::CastImageFilter<  
    FixedImageType, OutputImageType >    CastFilterType;  
typedef itk::ImageFileWriter< OutputImageType >    WriterType;
```

```
/*===== (Output) Filter Instantiations and Set-up=====*/
```

```
// Instantiate image resampler.
```

```
ResampleFilterType::Pointer resampler = ResampleFilterType::New();
```

```
// Set resampler input and transform to be applied.
```

```
resampler->SetInput(      movingImageReader->GetOutput() );
```

```
resampler->SetTransform( registration->GetOutput()->Get() );
```

```
// Set other necessary parameters for the resampler.
```

```
FixedImageType::Pointer fixedImage = fixedImageReader->GetOutput();
```

```
resampler->SetSize( fixedImage->GetLargestPossibleRegion().GetSize() );
```

```
resampler->SetOutputOrigin( fixedImage->GetOrigin() );
```

```
resampler->SetOutputSpacing( fixedImage->GetSpacing() );
```

```
resampler->SetOutputDirection( fixedImage->GetDirection() );
```

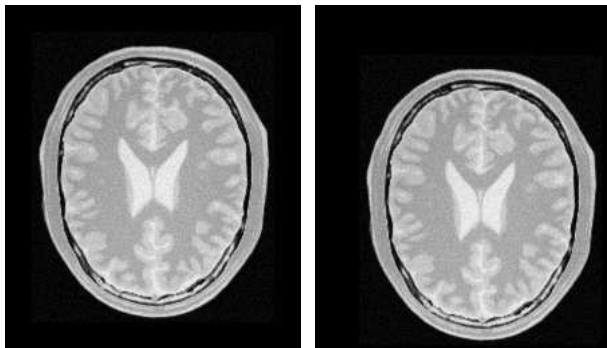
```
resampler->SetDefaultPixelValue( 100 );
```

```
// Instantiate image caster and writer.  
CastFilterType::Pointer  caster = CastFilterType::New();  
WriterType::Pointer      writer = WriterType::New();  
  
// Connect the output (mini) pipeline  
writer->SetFileName( argv[3] );  
caster->SetInput( resampler->GetOutput() );  
writer->SetInput( caster->GetOutput() );
```

```
/*=====Evoked Output Pipeline ( Update() )=====*/
try
{
    writer->Update();
}
catch( itk::ExceptionObject & excep )
{
    std::cerr << "Exception caught !" << std::endl;
    std::cerr << excep << std::endl;
}

return EXIT_SUCCESS;
}
```

## Registration (Set-up)



(a) Fixed Image

(b) Moving Image

Figure: Registration Set-up

## Registration (Results)

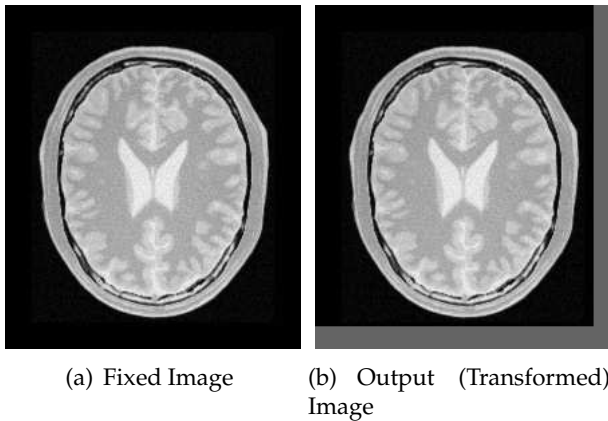


Figure: Registration Results

## Registration (Results)

Result =

Translation X = 12.9959 (ground truth = 13.0)

Translation Y = 17.0001 (ground truth = 17.0)

Iterations = 18

Metric value = 0.00745293

# References

- ITK (please see <http://proditus.com/itk> for links)
  - The ITK Software Guide
  - ITK Doxygen/API Documentation
  - ITK/CMake/Eclipse
  - ITK Examples (Installed along with ITK)
  
- C++
  - Cplusplus.com (<http://www.cplusplus.com>)
  
- Eclipse
  - Eclipse IDE for C/C++ (<http://www.eclipse.org/downloads>)

# ITK Workshop

*A bird's eye view on developing ITK applications*



Alp Kucukelbir  
<http://proditus.com>

Yale University

June 3, 2010