# *Lecture 7*
# *Registration with ITK*

Methods in Medical Image Analysis - Spring 2024
16-725 (CMU RI) : BioE 2630 (Pitt)
Dr. John Galeotti

Based in part on Damion Shelton's slides from 2006

# For more info/gory detail…

- Please see the following for exhaustive detail:
  - Chapter 3 in the ITK Software Guide Book 2
  - *Insight into Images*
  - ITK Source Tree
    - Examples/RegistrationITKv4/
    - E.g. Examples/RegistrationITKv4/ImageRegistration1.cxx
  - ITK Doxygen
    - http://www.itk.org/Doxygen53/html/group__Group-Registration.html
    - https://itk.org/Doxygen53/html/group__ITKRegistrationMethodsv4.html
    - http://www.itk.org/Doxygen53/html/group__Group-Numerics.html
  - SimpleITK:
    - http://insightsoftwareconsortium.github.io/SimpleITK-Notebooks/
      - See all the Python Registration (6x) notebooks, especially:
      - http://insightsoftwareconsortium.github.io/SimpleITK-Notebooks/Python_html/60_Registration_Introduction.html
      - https://simpleitk.org/doxygen/v2_3/html/classitk_1_1simple_1_1ImageRegistrationMethod.html

# What is registration?

- The process of aligning a target image to a source image
- More generally, determining the transform that maps points in the target image to points in the source image

# Transform types

- Rigid (rotate, translate)
- Affine (rigid + scale & shear)
- Deformable = non-rigid (affine + vector field)
- Many others

# Registration in ITK

- ITK uses an extensible registration framework
- Various interchangeable classes exist
- Relatively easy to “twiddle” the part you’re interested in while recycling prior work

- The newer ITKv4 Registration framework is separate from the legacy framework.
  - The legacy framework follows traditional practice
  - Version 4 registration is more flexible and thus more complex
  - Use the v4 framework whenever practical

- SimpleITK  also supports registration

- For “simplified” complex registration, consider using ANTS instead:
  - http://picsl.upenn.edu/software/ants/
  - http://stnava.github.io/ANTs/

# New since ITKv4 (ImageRegistrationMethodv4, etc.)

- New unified, improved, and fully multi-threaded optimization and registration framework (including multi-threaded metrics)
- Dense deformation fields (including a new transform that encapsulates a dense deformation field)
- Point Set registration methods (landmark or label guided registration)
- Automatic parameter scale estimation for transforms
- Automatic step-size selection for gradient-based registration optimizers
- Composite Transforms (grouping multiple transforms into a single one)
- Symmetric registration (where the Fixed and Moving images make unbiased contributions to the registration)
- New metrics for Demons and Mutual Information
- Diffeomorphic (velocity field) deformable registration
- Additional evolutionary optimizers
- Improved B-Spline registration approach available and bug fixes to old framework
- Accurately transform and reorient covariant tensors and vectors

List taken from http://www.itk.org/Wiki/ITK_Release_4/Why_Switch_to_ITKv4 and http://www.itk.org/Wiki/ITK_Release_4/Migration_Plan/Release_Notes/ITKv4_Final_Release_Notes

# ITKv4 Registration

- Uses a different framework than “traditional” ITK registration. The new framework is designated with a “v4” suffix.
- You must use a v4 metric and a v4 optimizer when doing a v4 registration!

- Take a look here:

  http://www.itk.org/Doxygen53/html/group__ITKRegistrationMethodsv4.html
  http://www.itk.org/Doxygen53/html/group__ITKMetricsv4.html
  http://www.itk.org/Doxygen53/html/group__ITKOptimizersv4.html
  ITK source code: Modules/Registration/RegistrationMethodsv4/include/
  ITK source code: Modules/Registration/Metricsv4/include/
  ITK source code: Modules/Numerics/Optimizersv4/include/

- Pay special attention to:
  - MattesMutualInformationImageToImageMetricv4
  - DemonsImageToImageMetricv4
  - QuasiNewtonOptimizerv4 (an improved gradient descent)

# Typical registration terminology

- Fixed image *f(x)* - stationary in space
- Moving image *m(x)* - the fixed image with an unknown transform applied
- Goal: recover the transform *T(x)* which maps points in *f(x)* to *m(x)*

# Typical registration framework

- 2 input images, fixed and moving
- Metric - determines the “fitness” of the current registration iteration
- Optimizer - adjusts the transform in an attempt to improve the metric
- Interpolator - applies transform to image and computes sub-pixel values

# Typical registration flowchart

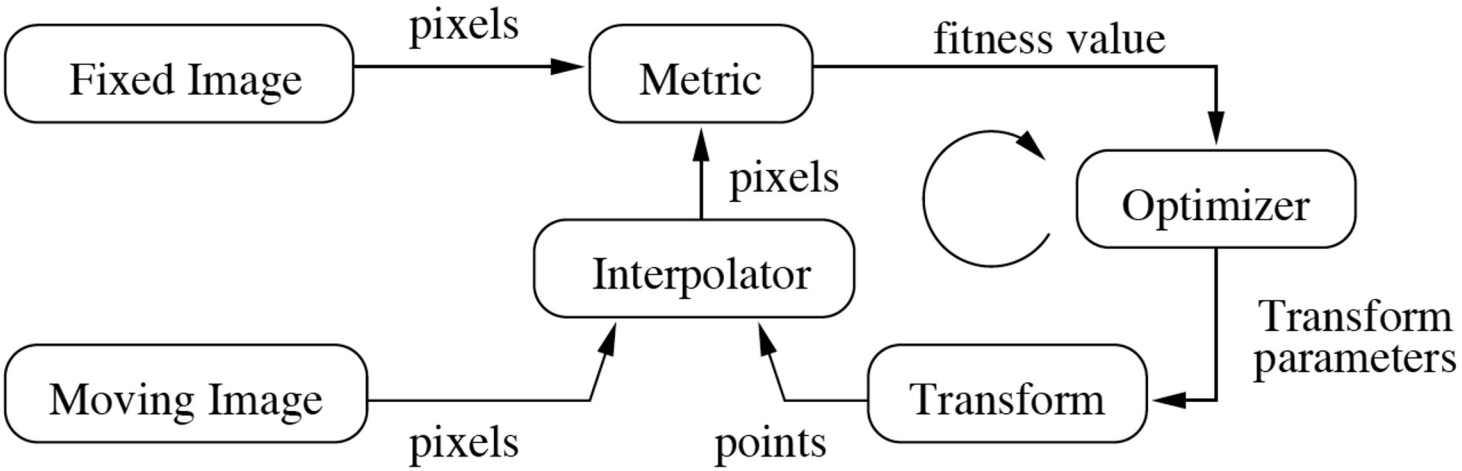



Figure 8.2 from the ITK Software Guide v 2.4, by Luis Ibáñez, et al.

# ITK v4 registration flowchart

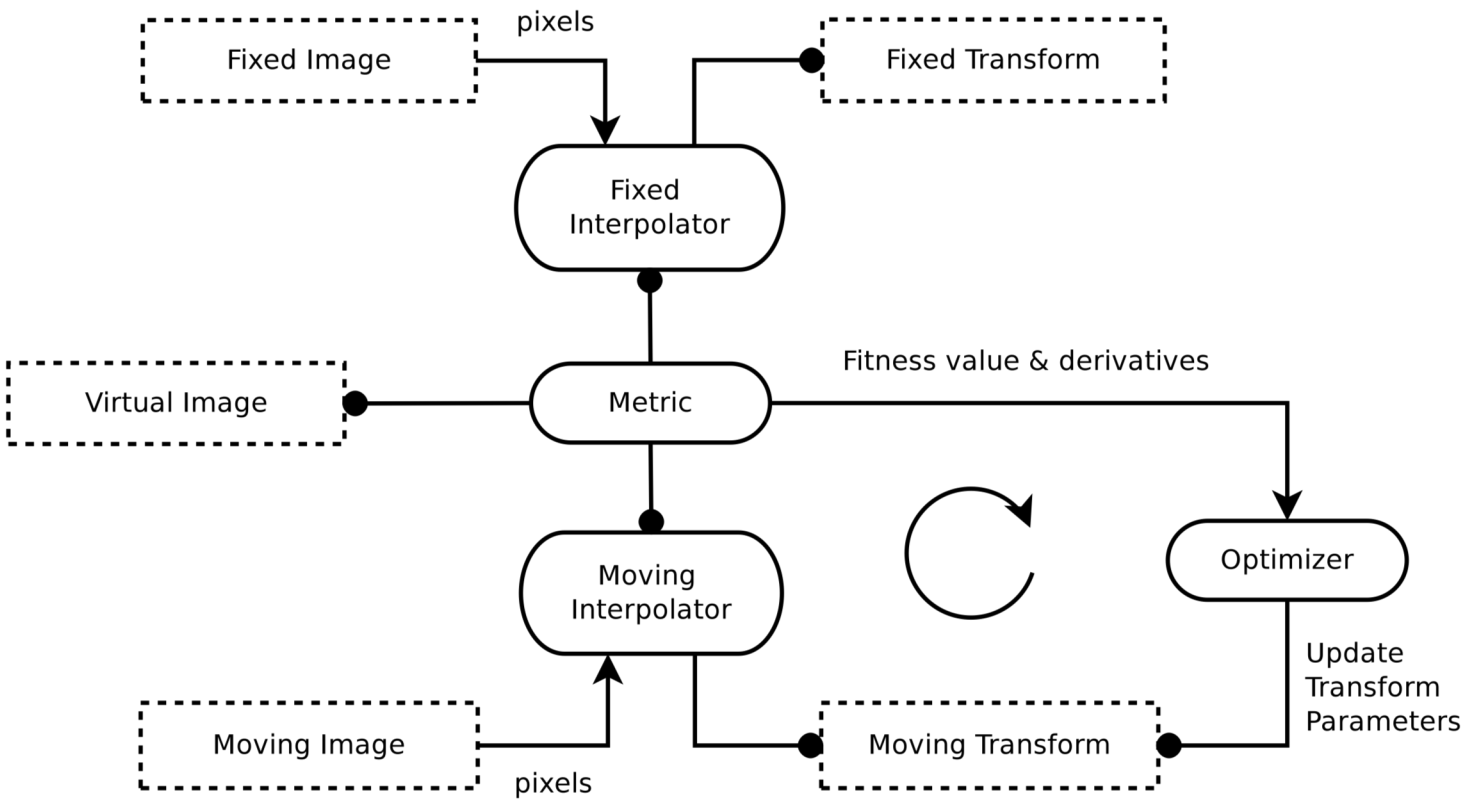



Figure 3.3 from the ITK Software Guide Book 2, Fourth Edition, by Hans J. Johnson, et al.

# ITK v4: key differences

- Both input images are transformed into a common virtual domain, which determines:
  - The output resampled-image dimensions and spacing
  - The sampling grid (not necessarily a uniform grid)
  - Defaults to the fixed image domain
- Only the Moving Transform is Optimized
- Fixed Transform defaults to identity transform
  - But it could be set to the result of a previous registration, etc.

# ITK v4 Virtual Domain

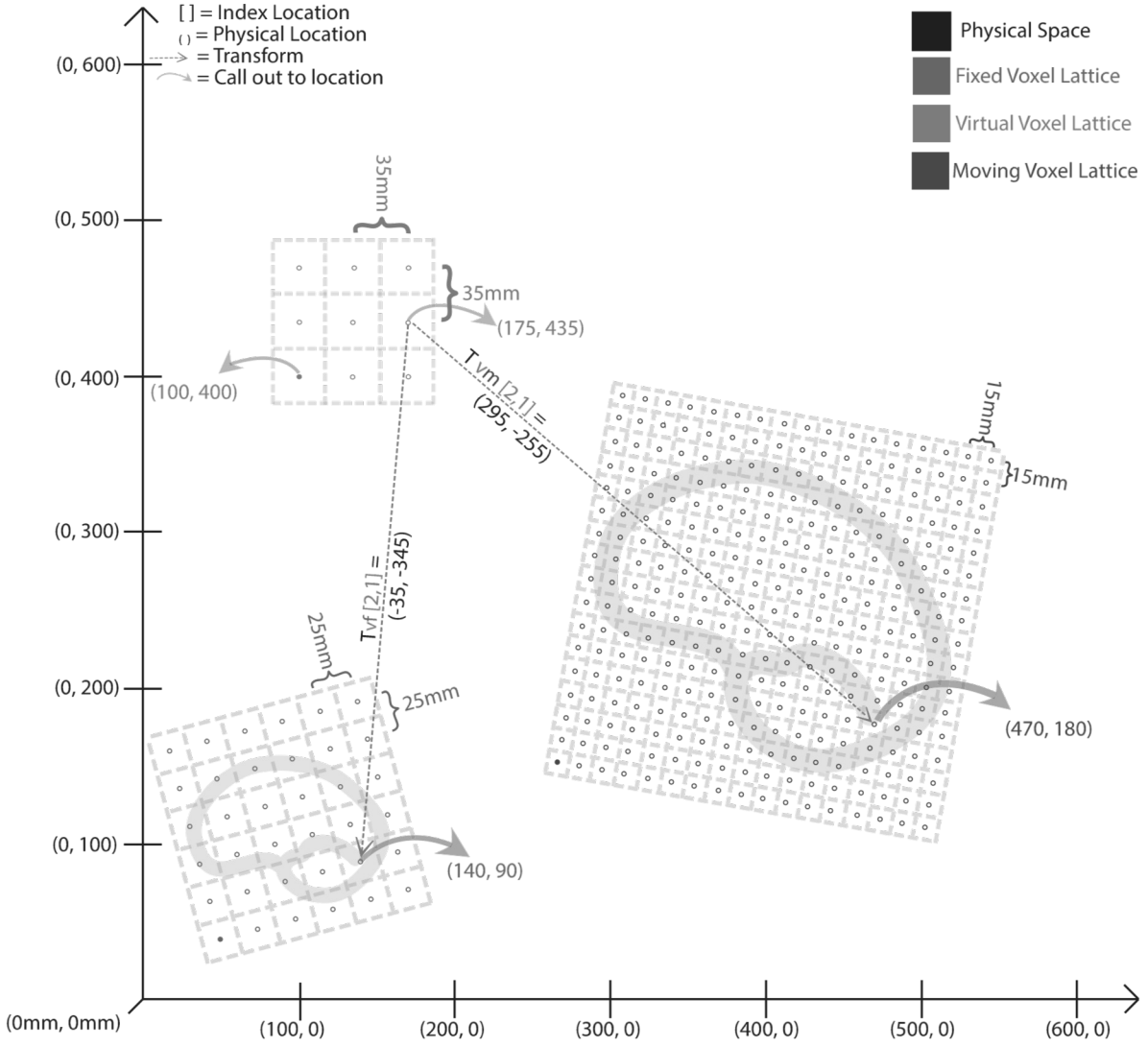



Figure 3.8 from the ITK Software Guide Book 2, Fourth Edition, by Hans J. Johnson, et al.

# ITK's "Hello world" registration example

- Uses ITK's v4 framework, but in the "typical" traditional style
- Please see the software guide (Book 2, Section 3.2) for code specifics
- I am going to cover what each piece does, not look at code per se

# ITK's "Hello World" Example: Flow Chart for Everything

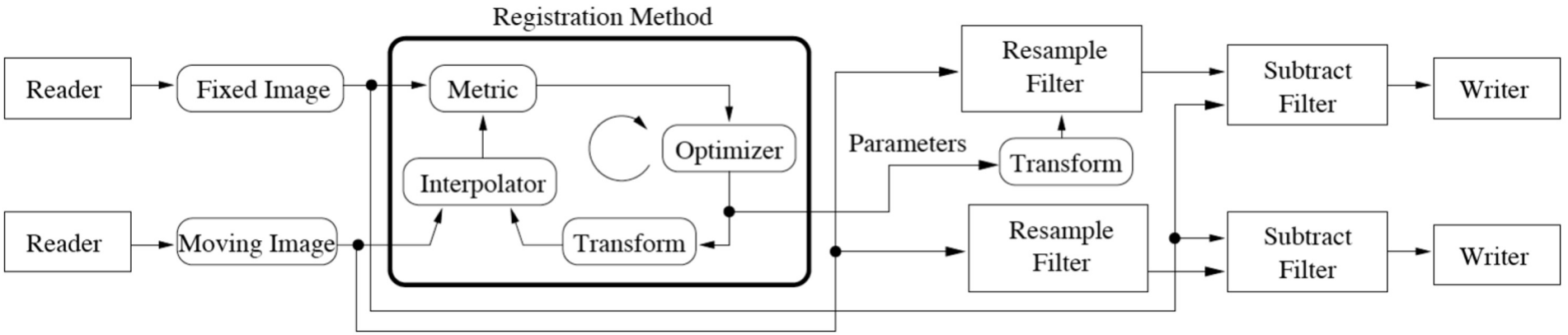



Figure 8.5 from the ITK Software Guide v 2.4, by Luis Ibáñez, et al.

# Input images

- 2D floating point
- Floating point avoids loss of precision problems with integer pixel types

# Transform

- TranslationTransform
- Permits *translation only* in 2D

- ITKv4 still uses the same legacy transforms
- ITKv4 also supports new *composite transforms*:
  - Two or more successive transforms…
  - Combined into a single transform object
  - Can initialize with one transform and optimize another

# Metric

- MeanSquaresImageToImageMetricv4
- Sum of squared differences between 2 images on a “pixel-by-pixel” basis
  - Remember that both images are transformed to the virtual domain before doing the comparisons
- A bit naïve
- Works for 2 images that were acquired with the same imaging modality

# Optimizer

- RegularStepGradientDescentOptimizerv4
- Follows the derivative of the metric
- Step size depends on rapid changes in the gradient's direction
- Step size eventually reaches a user-defined value that determines convergence

# Interpolator

- LinearInterpolateImageFunction
- Fast and conceptually simple

# Wrapper

- ImageRegistrationMethodv4
- Combines all of the previous classes into a master class

```
registration->SetMetric( metric );
registration->SetOptimizer( optimizer );
metric->SetFixedInterpolator( FixedInterpolator);
metric->SetMovingInterpolator( MovingInterpolator);
```

- Registration method automatically instantiates its own internal transform
  - Based on its template parameters

# Other steps

- Read the input images
- Setup the virtual domain
  - Defaults to the fixed image
- Set the region of the fixed image the registration will operate on
  - Useful for ignoring bad data
- Initialize the transforms
  - Fixed transform defaults to identity
- Setup multi-level registration
  - Like image-pyramids, but better
  - Defaults to a single level
- Use a C++ try/catch block to avoid crashing on errors
- Twiddle the optimizer for best performance*

*may involve pain and suffering

# Hello world input

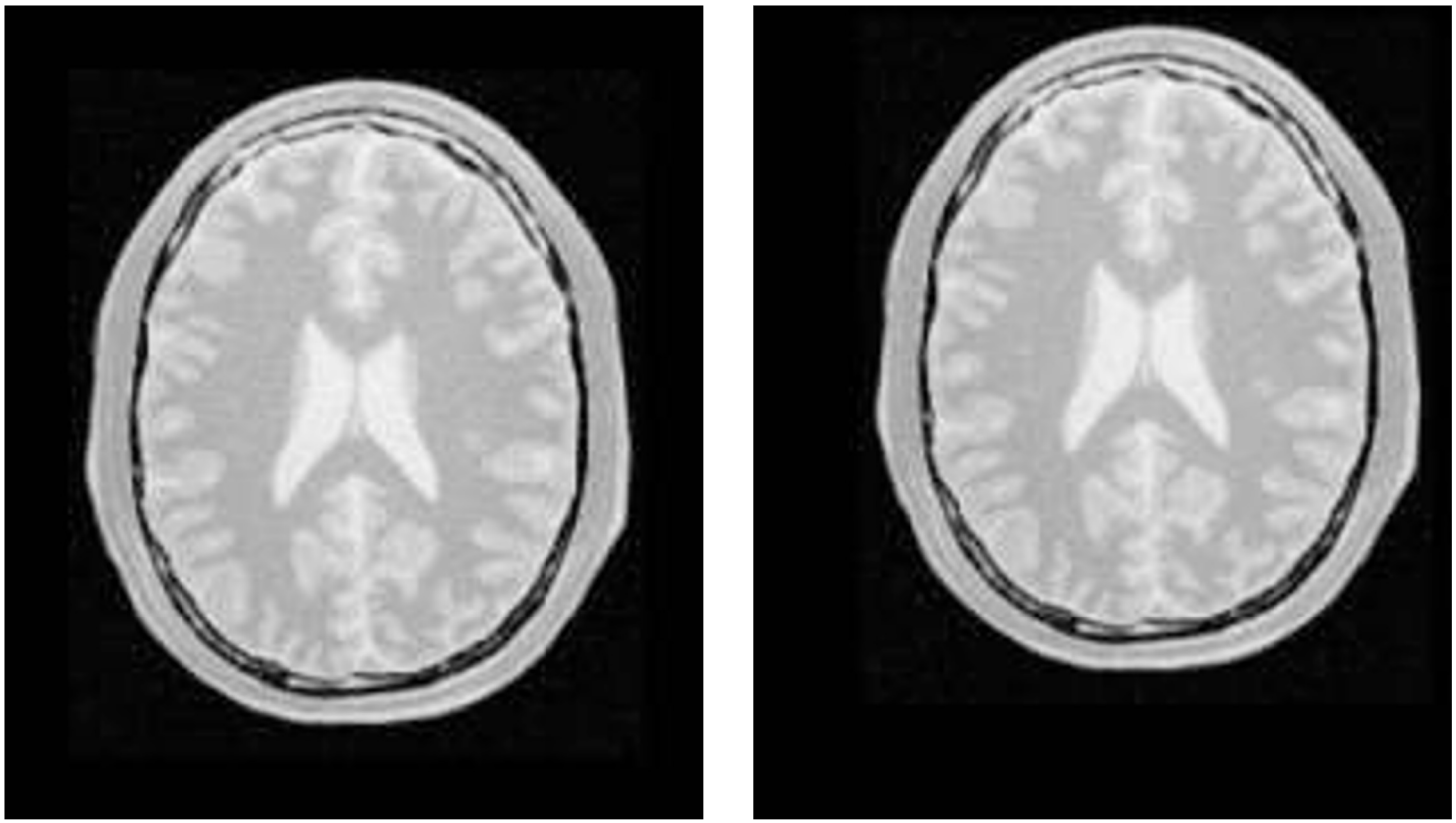


Figure 8.3 from the ITK Software Guide v 2.4, by Luis Ibáñez, et al.

# X & Y translation vs. time

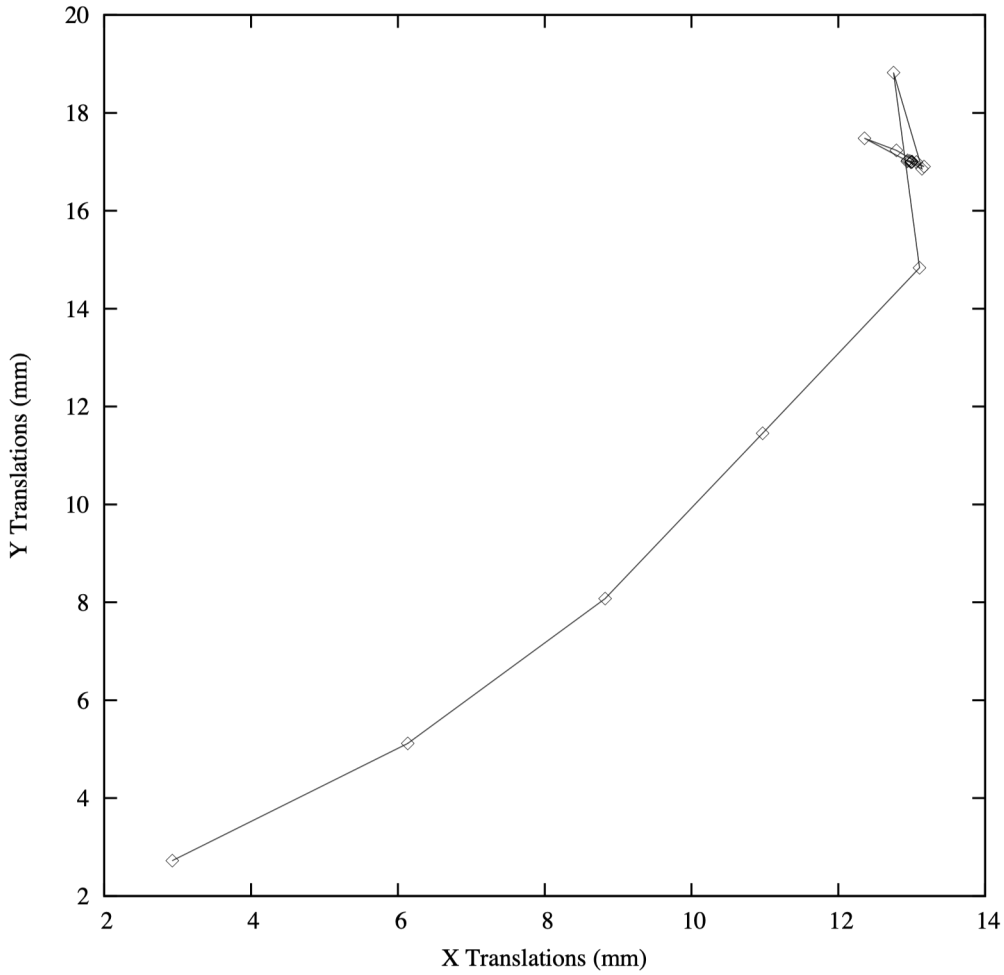



Figure 3.7 (left) from the ITK Software Guide Book 2, Fourth Edition, by Hans J. Johnson, et al.

# Metric vs. time

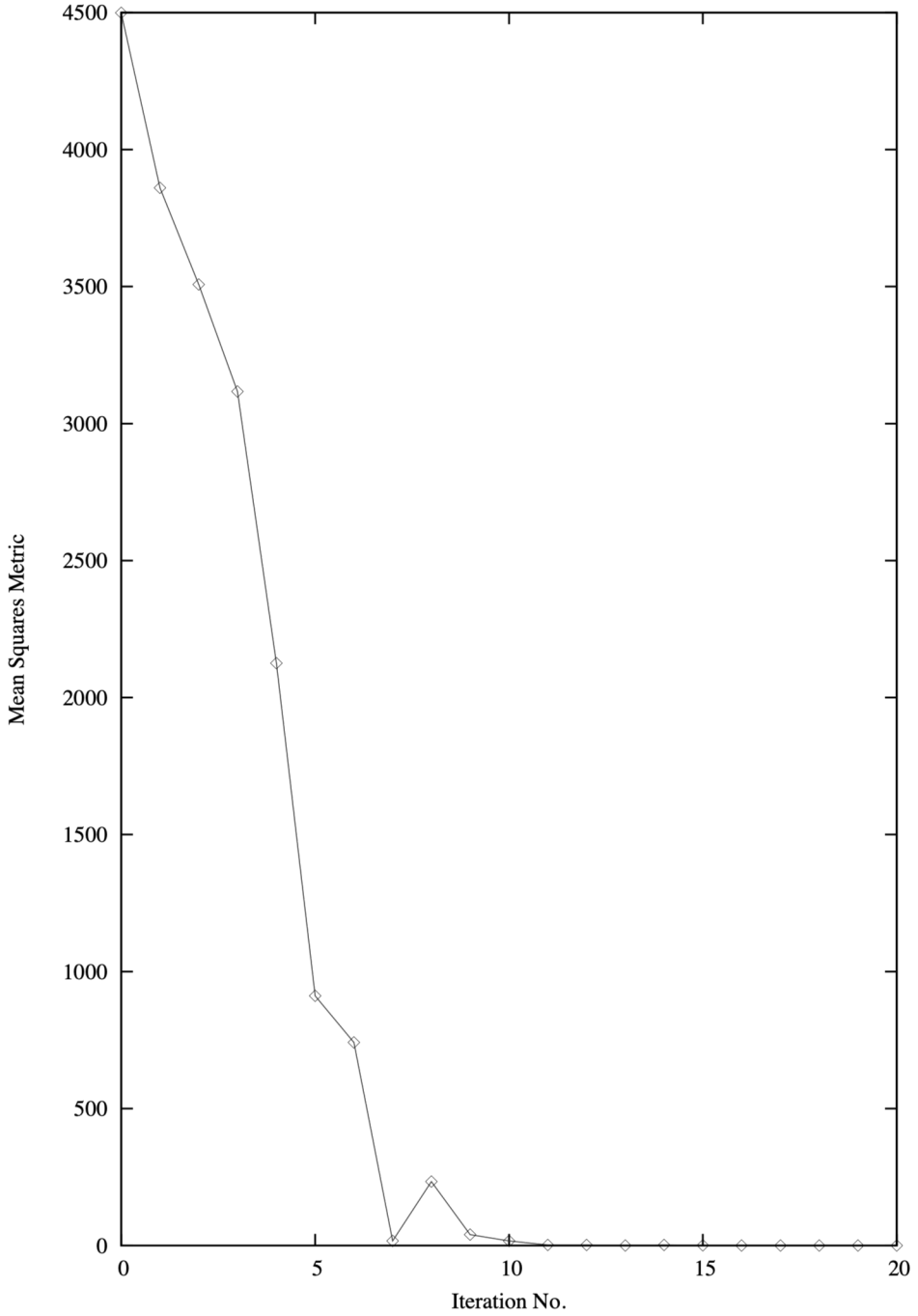



Figure 3.7 (left) from the ITK Software Guide Book 2, Fourth Edition, by Hans J. Johnson, et al.

# Registration results

- After registration converges/terminates, you recover the optimized transform with:

```
registration->GetTransform()
```

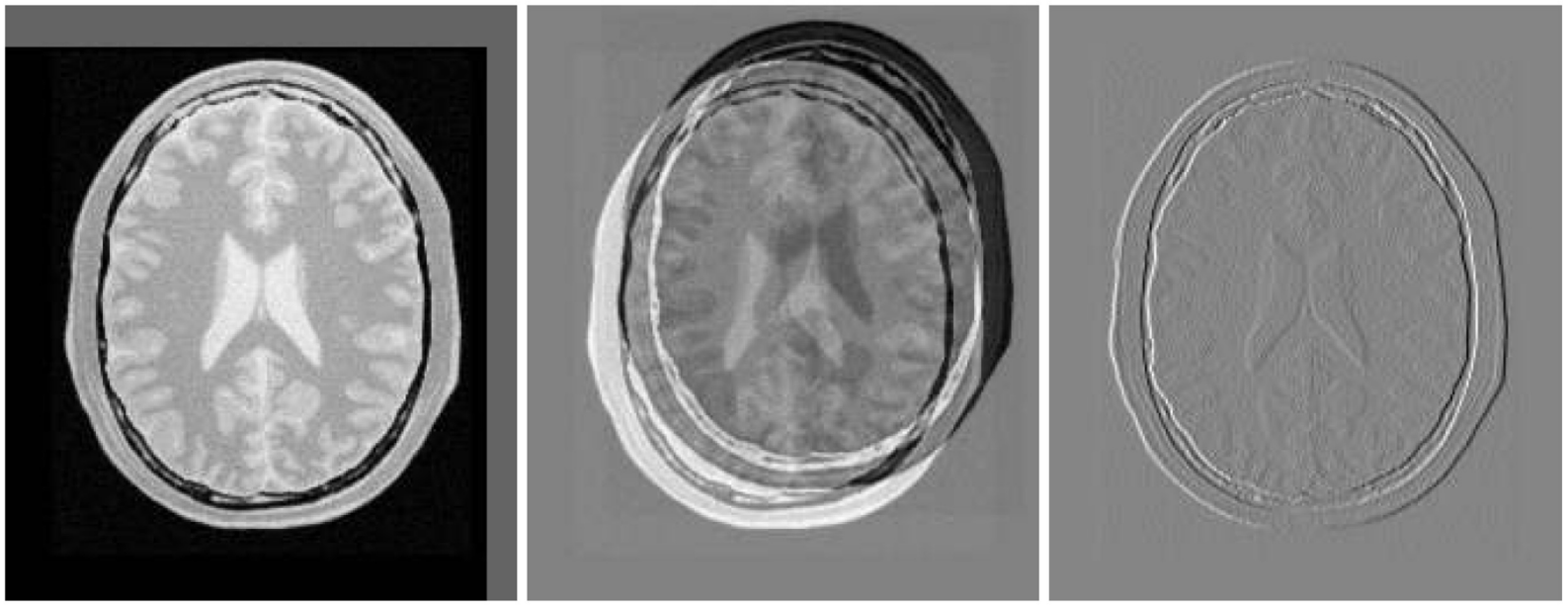
- For the Hello World example there are 2 parameters, X & Y translation
- If you used a separate initial moving transform, create a composite to get the total transform:

```
outputCompositeTransform->AddTransform(
    movingInitialTransform);
outputCompositeTransform->AddTransform(
    registration->GetModifiableTransform());
```

# Double checking results

- Use ResampleImageFilter to apply the transform for the fixed and moving images
- Take the outputs, and compute their difference
- In this case, just subtract the registered images
  - Good registration results in nothing much to see

# Image comparison



Registered moving image

Difference before registration

Difference after registration

Figure 8.4 from the ITK Software Guide v 2.4, by Luis Ibáñez, et al.

# Keeping tabs on registration

- Registration is often time consuming
- It's nice to know that your algorithm isn't just spinning it's wheels
- Use the observer (**itk::Command**) mechanism in ITK to monitor progress
  - ITK software guide, book 1: 3.2.6 and book 2: 3.4
- We'll see this again later, when we discuss how to write your own ITK filters
  - **itk::ProgressEvent** is one example

# Observer steps

- Write an observer class that will process "iteration" events
  - (Just copy some code from an example)
- Add the observer to the optimizer
  - As a generic note, observers can observe any class derived from **itk::Object**
- Start registration as usual

# Things observers can do

- Print debugging info
- Update GUI
- Other small management functions
- *Should not* do anything too processor intensive

# ITK v4 Registration Observer

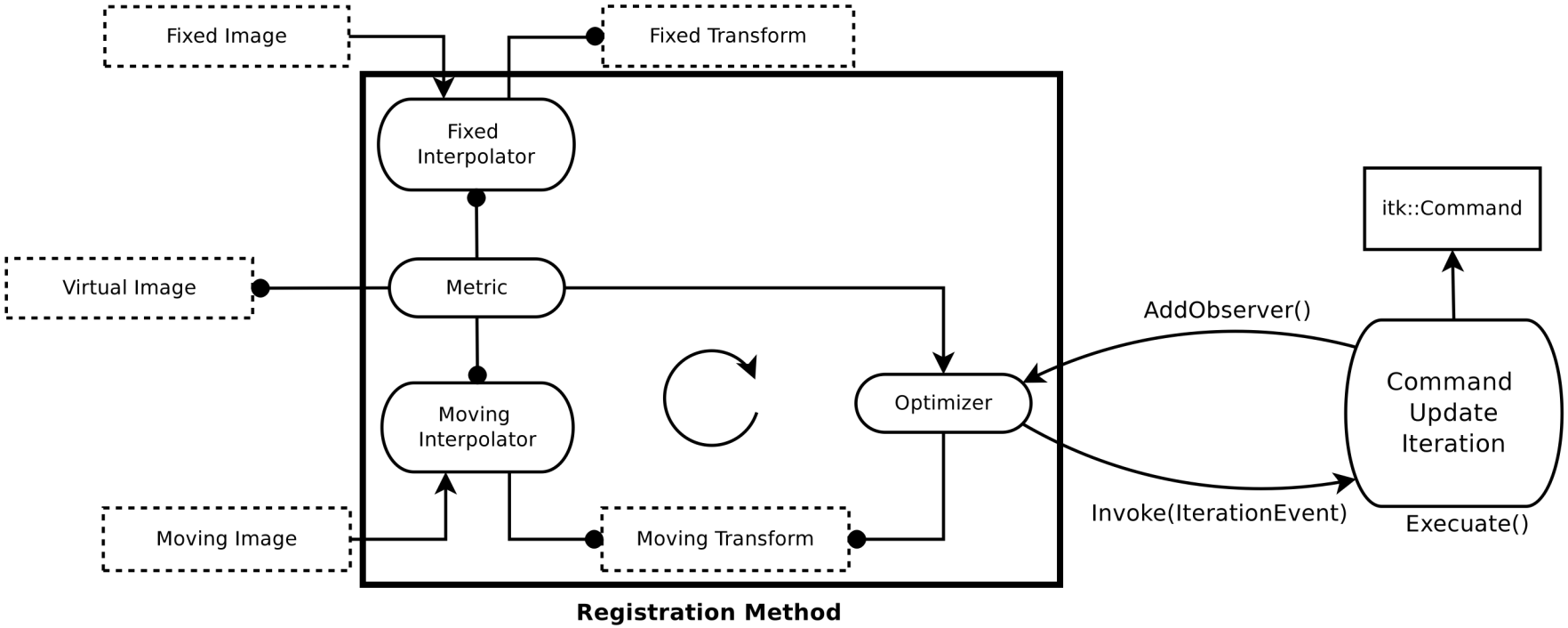



Figure 3.9 from the ITK Software Guide Book 2, Fourth Edition, by Hans J. Johnson, et al.

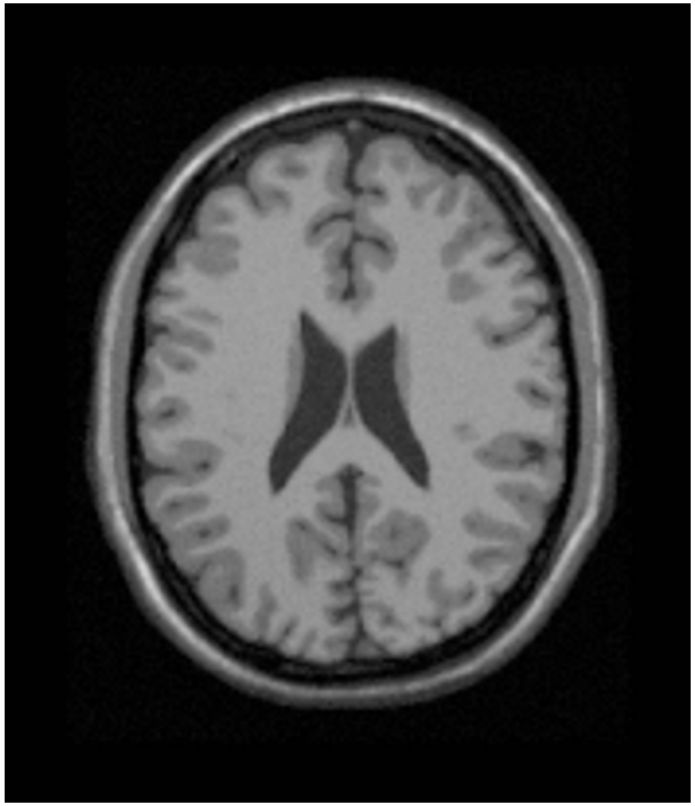
# Multi-modality registration

- Remember how I said sum-of-squares difference is relatively naïve?
- Mutual information helps overcome this problem
- Section 3.5 shows how to implement a simple MI registration
  - Note that Mattes MI is usually easier to use than Viola-Wells MI

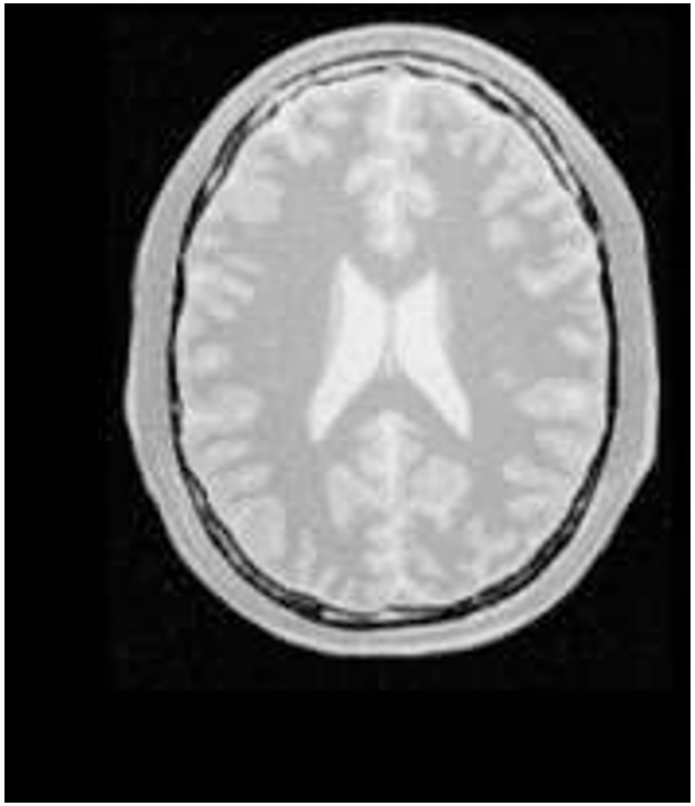
# Notes about the MI example

- Significantly, largely the same piece of code as Hello World
- Mutual Information is a *metric*, so we can keep the optimizer, the interpolator, and so on
- Majority of differences are in tweaking the metric, not in rewriting code

# MI Inputs



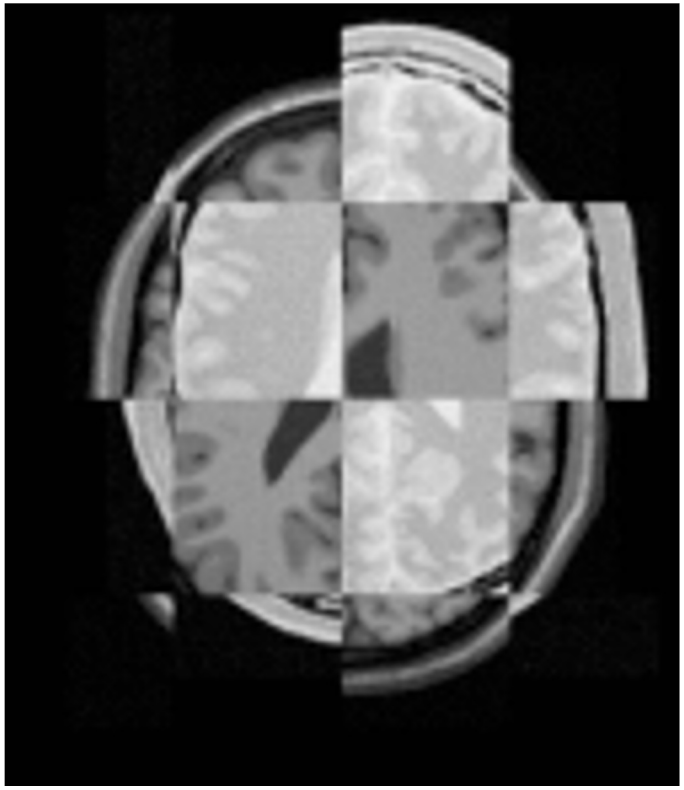
T1 MRI



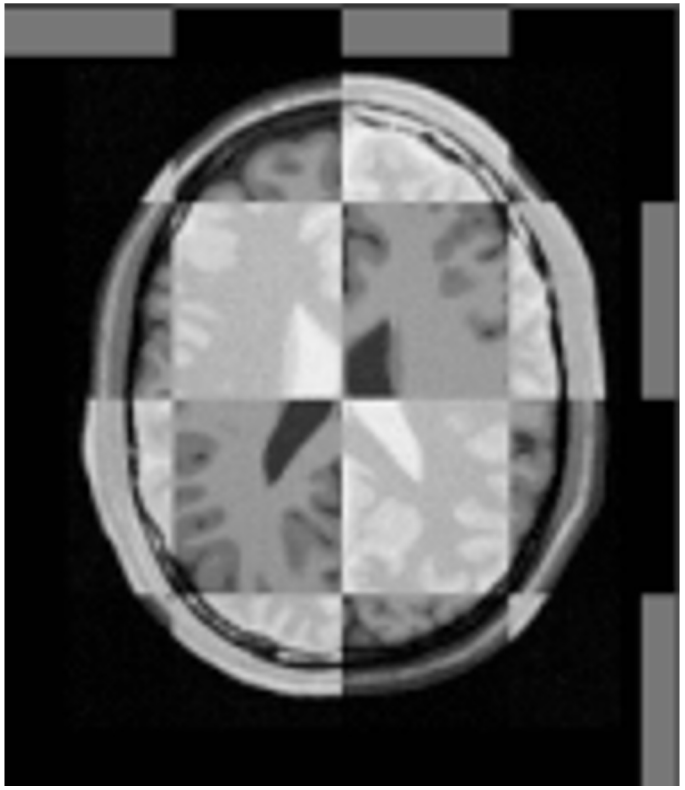
Proton density MRI

Figure 8.9 from the ITK Software Guide v 2.4, by Luis Ibáñez, et al.

# MI Output: Image Comparison



Before



After

This is an example of a checkerboard visualization

Taken from Figure 8.10 of the ITK Software Guide v 2.4, by Luis Ibáñez, et al.

# Centered transforms

- More natural (arguably) reference frame than having the origin at the corner of the image
- Big picture is not appreciably different from other rigid registrations
- But, for the moment there are implementation complexities and differences, see 3.6

# An aside: “Twiddling”

- A common criticism of many/most registration techniques is their number of parameters
- A successful registration often depends on a very specific fine-tuning of the algorithm
- “Generalized” registration is an open problem

# Multi-Resolution registration

- Useful to think of this as algorithmic "squinting" by using image pyramids
- Start with something simple and low-res
- Use low-res registration to seed the next higher step
- Eventually run registration at high-res
- Also called "coarse to fine"

# Multi-resolution idea

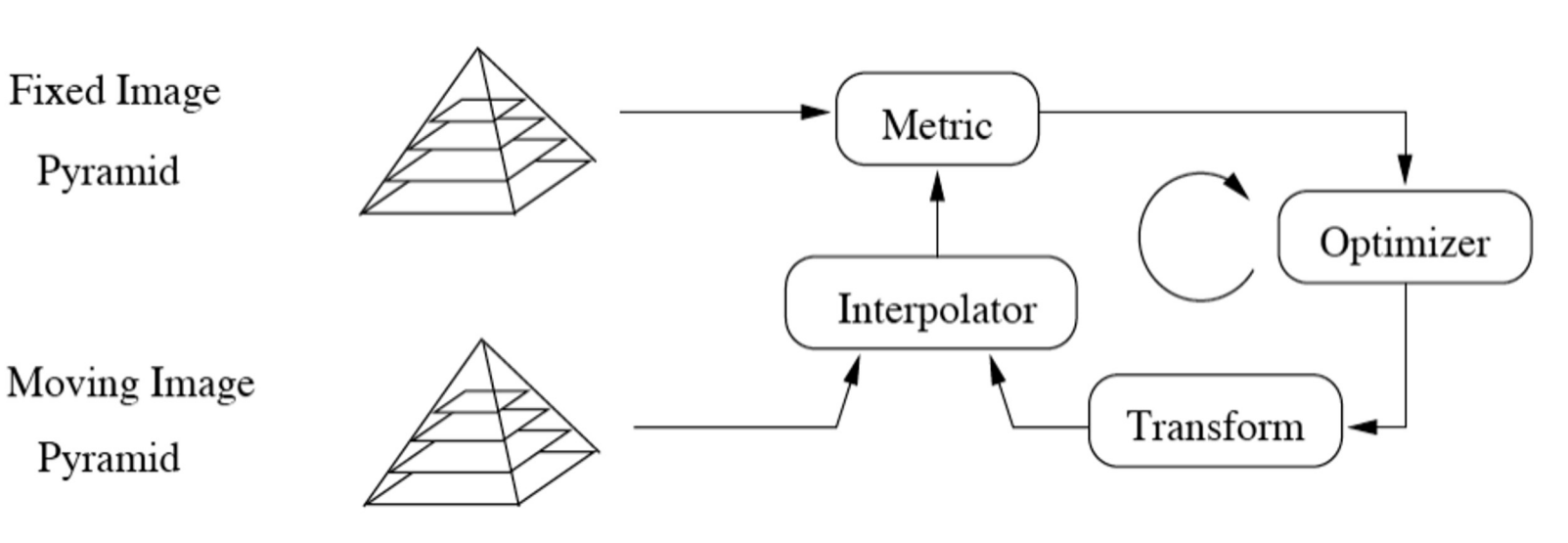



Figure 8.36 from the ITK Software Guide v 2.4, by Luis Ibáñez, et al.

# Image pyramids

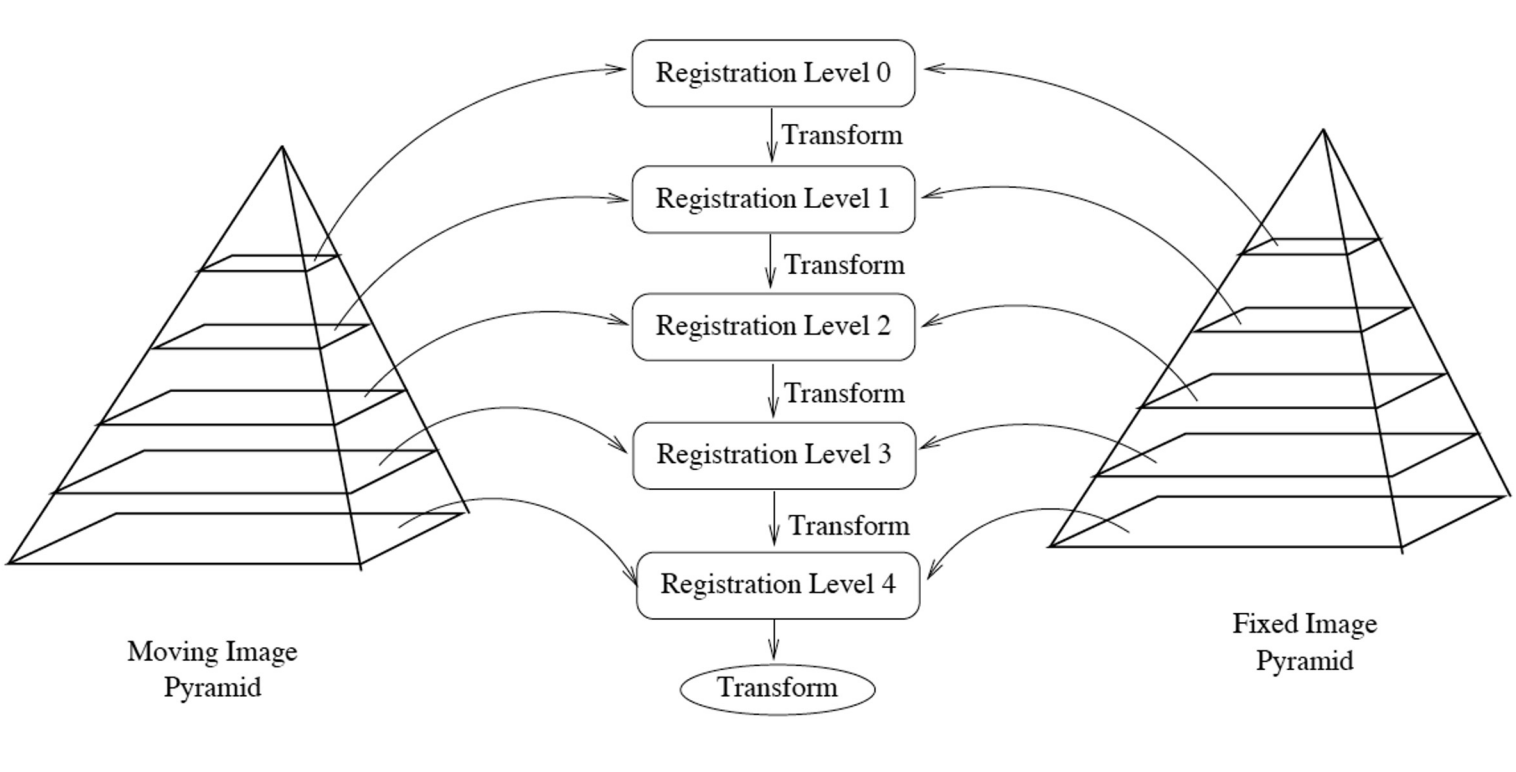



Figure 8.37 from the ITK Software Guide v 2.4, by Luis Ibáñez, et al.

# Optimization

- Parameter dependency rears its ugly head
- You often/usually need to adjust optimizer parameters as you move through the pyramid
- You can do this using the Observer mechanism

# Multi-resolution example

- Again, mostly the same code as Hello World
- Multi-Resolution is now built into all of ITKv4 registration, so no need for extra classes or image pyramids

# Benefits of multi-resolution

- Often faster
- More tolerant of noise (from “squinting”)
- Minimizes initialization problems to a certain extent, though not perfect

# Multi-resolution

- Remember, at large (high) scale only large objects are visible
- Higher scale is higher in the image pyramid
  - So higher scale has lower resolution
- Lower scale is lower in the image pyramid
  - So lower scale has higher resolution

# See the software guide for…

- Detailed list of:
  - Transforms
  - Optimizers
  - Interpolation methods
- You're encouraged to mix and match!

# Deformable registration

- Three common techniques:
  - Finite element: treat small image regions as having physical properties that control deformation
  - Bsplines:  deform a mapping grid
  - Demons: images are assumed to have iso-intensity contours (isophotes); image deformations occur by pushing on these contours

# Model based registration

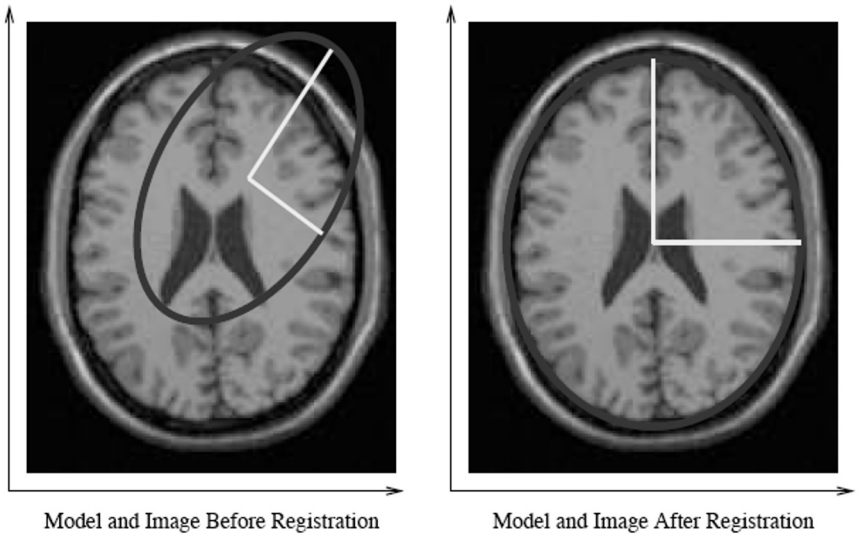
- Software guide, book 2, ch. 3, section 16.

- Build a simplified geometric model from a training set
- Identify parameters that control the characteristics of the model
- Register the model to a target image to adapt to a particular patient

# Model based, cont.

- Uses the Spatial Objects framework for representing geometry
- Useful because it derives analytical data from the registration process, not just a pixel-to-pixel mapping

# Model-based example




Note: This is what we want, NOT the output of an actual registration

Figure 8.60 from the ITK Software Guide v 2.4, by Luis Ibáñez, et al.

# Model-based reg. schematic

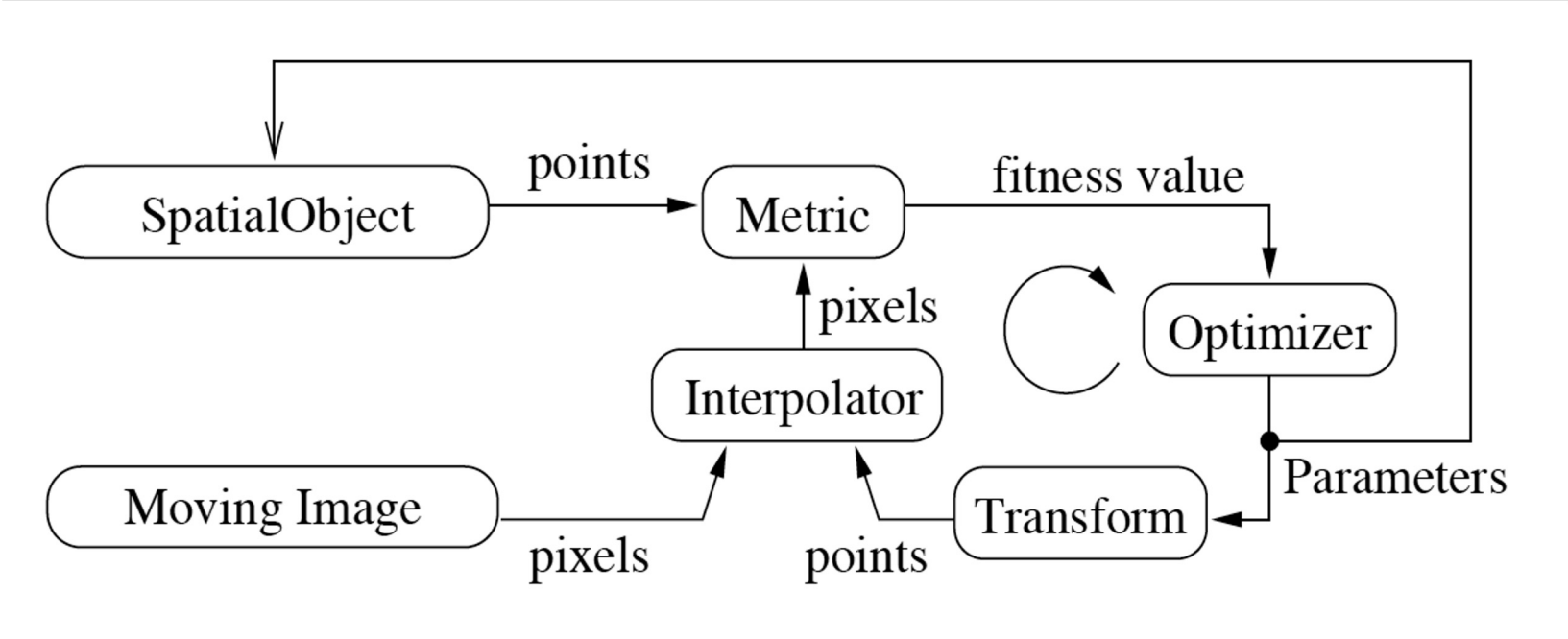



Figure 8.59 from the ITK Software Guide v 2.4, by Luis Ibáñez, et al.

# Model-based registration: Warning!

- ITK does not yet directly support generic model-based registration “out of the box”
- ITKv4 does support point-set to image registration
- Otherwise, model-based reg. requires writing your own custom ITK transform, with new parameters
  - Transform’s new parameters → Spatial Object parameters
  - *You* must individually map your custom transform’s new parameters to the specific spatial object parameters you want to allow registration to adjust
  - This isn’t too complicated if you know what you’re doing
  - Search Insight Journal for examples

# Speed issues

- Execution time can vary wildly
  - Optimizer (more naïve = faster)
  - Image dimensionality (fewer = faster)
  - Transform (fewer DOF = faster)
  - Interpolator (less precise = faster)

# Take home messages

- Exactly what parameters do what is not always obvious, even if you are familiar with the code
- Successful registrations can be something of an art form
- Multi-resolution techniques can help
- Work within the framework!