Viscous Fluid Model for Large Deformation Image Registration

Igor Yanovsky and Luminita Vese
Department of Mathematics, UCLA

Center for Domain-Specific Computing
March 10, 2010

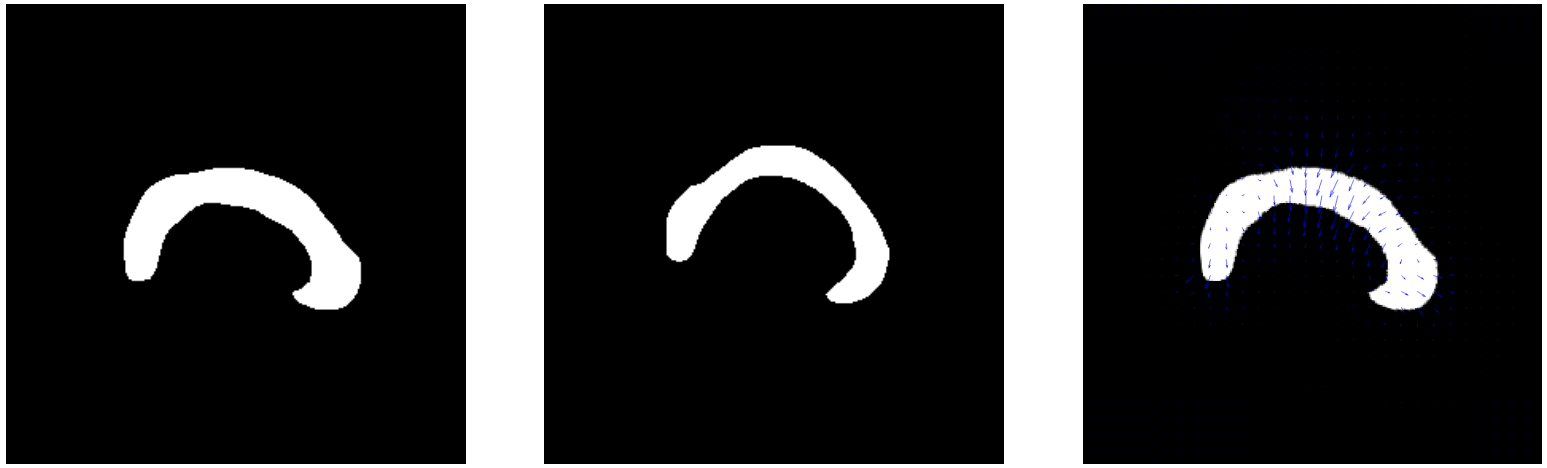
UCLA
Department of Mathematics

Image Registration Problem

Given the target image I_1 and the source image I_2, find the transformation **g** that maps I_2 into correspondence with I_1.

Equivalently, find the displacement field **u**, defined in terms of the deformation **g**, by **g** = **id** – **u**.

I_1 I_2 $I_2 \circ \mathbf{g}$



Variational Framework

In general, nonlinear image registration models can be formulated in a variational framework.

- An appropriate fidelity term F indicates how accurately the deformed image is aligned with the target image.
- Regularizing constraint R is imposed in order to construct a deformation that is one-to-one and differentiable.

Minimization Problem:

$$\inf_{\mathbf{u}} \left\{ E(\mathbf{u}) = F(\mathbf{u}) + \lambda R(\mathbf{u}) \right\},$$

where F is the fidelity term, R is the regularizer,

and $\lambda > 0$ is a weighting parameter.

Minimization Problem

Minimization Problem:

$$\inf_{\mathbf{u}} \{E(\mathbf{u}) = F(\mathbf{u}) + \lambda R(\mathbf{u})\},$$

where F is the fidelity term, R is the regularizer,

and $\lambda > 0$ is a weighting parameter.

A necessary condition for a minimizer $\mathbf{u}$ is that the Gâteaux derivative of E vanishes for all suitable perturbations.

First variation:

$$dE(\mathbf{u}, \boldsymbol{\eta}) = \langle \partial_{\mathbf{u}} E, \boldsymbol{\eta} \rangle = 0.$$

Notation:

$$\partial_{\mathbf{u}} E(\mathbf{u}) = \big(\partial_{u_1} E(\mathbf{u}), \ldots, \partial_{u_n} E(\mathbf{u})\big)^T.$$

Registration Metrics

L^2-norm (the sum of squared intensity differences):

- The *L2-norm matching functional* is suitable when the images have been acquired through similar sensors and thus are expected to present the same intensity range and distribution.

Given the displacement field $\mathbf{u}$, the L^2 norm of the difference between $I_1(\mathbf{x}-\mathbf{u})$ and $I_2(\mathbf{x})$ is

$$F_{L^2}(I_1, I_2, \mathbf{u}) = \frac{1}{2}\int_{\Omega} \left(I_2(\mathbf{x}-\mathbf{u}) - I_1(\mathbf{x})\right)^2 d\mathbf{x}.$$

The force field is defined as:

$$\mathbf{f}(\mathbf{x}, \mathbf{u}(\mathbf{x})) = -\partial F_{L^2}(I_1, I_2, \mathbf{u}) = \left[I_2(\mathbf{x}-\mathbf{u}(\mathbf{x})) - I_1(\mathbf{x})\right]\nabla I_2(\mathbf{x}-\mathbf{u}(\mathbf{x})).$$

Registration Metrics

Mutual Information similarity function:

- Register images of different modalities.

Let p^{I_1} and $p_{\mathbf{u}}^{I_2}$ be intensity distributions estimated from $I_1(\mathbf{x})$ and $I_2(\mathbf{x}-\mathbf{u})$, respectively, and $p_{\mathbf{u}}^{I_1,I_2}$ is an estimate of the joint intensity distribution. Let $i_1 = I_1(\mathbf{x})$, $i_2 = I_2(\mathbf{x}-\mathbf{u}(\mathbf{x}))$ denote intensity values. Given the displacement field $\mathbf{u}$, the mutual information computed from I_1 and I_2 is

$$MI_{\mathbf{u}}^{I_1,I_2} = \frac{1}{2}\int_{\mathbf{R}^2} p_{\mathbf{u}}^{I_1,I_2}(i_1,i_2)\log\frac{p_{\mathbf{u}}^{I_1,I_2}(i_1,i_2)}{p^{I_1}(i_1)p_{\mathbf{u}}^{I_2}(i_2)}\,di_1\,di_2.$$

Hence, $F_{MI}(I_1,I_2,\mathbf{u}) = -MI_{\mathbf{u}}^{I_1,I_2}$.

The force field is defined as:

$$\mathbf{f}(\mathbf{x},\mathbf{u}(\mathbf{x})) = -\partial_{\mathbf{u}}F_{MI}(\mathbf{u}).$$

Regularizers

- Diffusion Registration

$$R_{DIF}(\mathbf{u}) = \frac{1}{2}\sum_{k=1}^{n}\int_{\Omega} ||\nabla u_k||^2 d\mathbf{x}$$

$$\partial_{\mathbf{u}} R_{DIF}(\mathbf{u}) = -\triangle \mathbf{u}$$

$$\lambda \triangle \mathbf{u} + \mathbf{f} = 0$$

- Biharmonic Registration

$$R_{BH}(\mathbf{u}) = \frac{1}{2}\sum_{k=1}^{n}\int_{\Omega} (\triangle u_k)^2 d\mathbf{x}$$

$$\partial_{\mathbf{u}} R_{BH}(\mathbf{u}) = \triangle^2 \mathbf{u}$$

$$\lambda \triangle^2 \mathbf{u} - \mathbf{f} = 0$$

- The expression of the force field **f** depends on the similarity measure
- Easy to implement numerically
- Depend only on one parameter
- Small deformations are allowed

Regularizers - Physical Models

The deforming image is considered to be embedded in a three-dimensional deformable medium, which is either:

- elastic material

  Linear Elastic Model (Broit, 1991)

- viscous fluid

  Viscous Fluid Model (Christensen, Rabbitt, Miller, 1996)

Regularizers - Physical Models

- Linear Elastic Model (Broit, 1991)
  - The theory of linear elasticity is based on notions of stress and strain.
  - Navier-Cauchy linear elastic PDE:

$$\mu \triangle \mathbf{u} + (\mu + \nu)\nabla(\nabla \cdot \mathbf{u}) + \mathbf{f}(\mathbf{x}) = 0.$$

  - Equivalently, in variational framework:

$$R_{LE}(\mathbf{u}) = \frac{1}{2}\int_{\Omega}\left(\nu(\nabla \cdot \mathbf{u})^2 + 2\mu \sum_{i,j=1}^{n}(\varepsilon_{ij}(\mathbf{u}))^2\right) d\mathbf{x}$$

where $\varepsilon_{ij}(\mathbf{u}) = \frac{1}{2}(\partial_j u_i + \partial_i u_j)$ is the linear strain.

$$\partial_{\mathbf{u}} R_{LE}(\mathbf{u}) = -\mu \triangle \mathbf{u} - (\mu + \nu)\nabla(\nabla \cdot \mathbf{u}).$$

  - Small displacement fields.

Regularizers - Physical Models

- Viscous Fluid Model (Christensen, Rabbitt, Miller, 1996)
  - Eulerian reference frame; Material derivative:

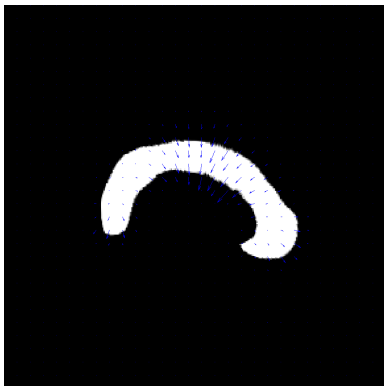
$$\mathbf{v} = \frac{D\mathbf{u}}{Dt} = \frac{\partial \mathbf{u}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{u}.$$

  - Navier-Stokes equation for a viscous fluid:

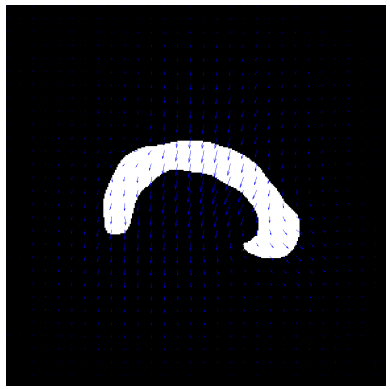
$$\mu \triangle \mathbf{v} + (\mu + \nu) \nabla (\nabla \cdot \mathbf{v}) + \mathbf{f}(\mathbf{x}, \mathbf{u}) = 0.$$

  - Alternatively, $\mathbf{v} = G_\sigma * \mathbf{f}(\mathbf{x}, \mathbf{u})$.
  - Allows long distance, nonlinear deformations.

$\mathbf{v} = G_\sigma * \mathbf{f}(\mathbf{x}, \mathbf{u})$.



$\mu \triangle \mathbf{v} + (\mu + \nu) \nabla (\nabla \cdot \mathbf{v}) + \mathbf{f}(\mathbf{x}, \mathbf{u}) = 0.$



Fluid Registration

Major operations:

1. Given **u**, interpolate $I_2(\mathbf{x})$ to obtain $I_2(\mathbf{x}-\mathbf{u})$.
2. Calculate the force field **f**.
3. Evaluate the velocity field **v** via solving Navier-Stokes equation.
4. Solve for displacement **u**.

Fluid Registration

- **1)** Given **u**, interpolate I_2(**x**) to obtain I_2(**x-u**):

- **Code:**

```
Registration3D::linearInterpolation( I2, u1, u2, u3,...);
In: I2,(u1,u2,u3)
Out: I2(x-u)
```

Fluid Registration

- **2)** Calculate the **force field f**:

$$\mathbf{f}(\mathbf{x}, \mathbf{u}(\mathbf{x})) = [I_2(\mathbf{x} - \mathbf{u}(\mathbf{x})) - I_1(\mathbf{x})] \nabla I_2(\mathbf{x} - \mathbf{u}(\mathbf{x}))$$

or, equivalently

$$\mathbf{f}(\mathbf{x}, \mathbf{u}(\mathbf{x})) = [I_2(\mathbf{x} - \mathbf{u}(\mathbf{x})) - I_1(\mathbf{x})] \begin{bmatrix} \frac{\partial I_2(\mathbf{x} - \mathbf{u}(\mathbf{x}))}{\partial x} \\ \frac{\partial I_2(\mathbf{x} - \mathbf{u}(\mathbf{x}))}{\partial y} \\ \frac{\partial I_2(\mathbf{x} - \mathbf{u}(\mathbf{x}))}{\partial z} \end{bmatrix}$$

- **Code:**

```
Registration3D::evaluate_f_L2( I1, I2, u1, u2, u3,
                               f1, f2, f3, ... );
```

In: I1, I2, (u1,u2,u3)

Out: (f1,f2,f3)

Fluid Registration

- **3a)** Evaluate the **velocity field** **v** via solving Navier-Stokes equation:

$$\mu \triangle \mathbf{v} + (\mu + \nu)\nabla(\nabla \cdot \mathbf{v}) + \mathbf{f}(\mathbf{x}, \mathbf{u}) = 0.$$

- **Code:**

```
Registration3D::evaluate_v( f1, f2, f3, ... );
In: (f1,f2,f3)
Out: (f1,f2,f3), which is (v1,v2,v3)
```

PDE_System3D is a **virtual** class which allows for defining a particular system of general elliptic PDEs as well as the functionality for obtaining a numerical solution to this system.

```
class PDE_System3D
{

public:

     long problemNumber;       // problem identifier

     virtual void operatorL(   const DoubleArray3D& v1, const DoubleArray3D& v2,
                               const DoubleArray3D& v3,
                               DoubleArray3D& Lop1, DoubleArray3D& Lop2,
                               DoubleArray3D& Lop3,
                               const double& dx )            = 0;

     virtual void solveExactly(    DoubleArray3D& v1, DoubleArray3D& v2,
                                   DoubleArray3D& v3,
                                   const DoubleArray3D& f1, const DoubleArray3D& f2,
                                   const DoubleArray3D& f3,
                                   const double& dx )        = 0;

     virtual void applyRelaxationGS(   DoubleArray3D& v1, DoubleArray3D& v2,
                                       DoubleArray3D& v3,
                                       const DoubleArray3D& f1, const DoubleArray3D& f2,
                                       const DoubleArray3D& f3,
                                       const double& dx )    = 0;

     virtual void findResidual( const DoubleArray3D& v1, const DoubleArray3D& v2,
                                const DoubleArray3D& v3,
                                const DoubleArray3D& f1, const DoubleArray3D& f2,
                                const DoubleArray3D& f3,
                                DoubleArray3D& R1, DoubleArray3D& R2,
                                DoubleArray3D& R3,
                                const double& dx )         = 0;
};
```

StationaryNS3D class inherits from PDE_System3D and defines the system of Navier-Stokes equations as well as the functionality for obtaining a numerical solution to this system.

```
class StationaryNS3D : public PDE_System3D
{

public:

     double mu;
     double lambda;

     StationaryNS3D();

     void operatorL( const DoubleArray3D& v1, const DoubleArray3D& v2, const DoubleArray3D& v3,
         DoubleArray3D& Lop1, DoubleArray3D& Lop2, DoubleArray3D& Lop3, const double& dx );

     void solveExactly( DoubleArray3D& v1,DoubleArray3D& v2, DoubleArray3D& v3,
         const DoubleArray3D& f1, const DoubleArray3D& f2, const DoubleArray3D& f3,
         const double& dx );

     void applyRelaxationGS( DoubleArray3D& v1, DoubleArray3D& v2, DoubleArray3D& v3,
         const DoubleArray3D& f1, const DoubleArray3D& f2, const DoubleArray3D& f3,
         const double& dx );

     void findResidual( const DoubleArray3D& v1, const DoubleArray3D& v2, const DoubleArray3D& v3,
         const DoubleArray3D& f1, const DoubleArray3D& f2, const DoubleArray3D& f3,
         DoubleArray3D& R1, DoubleArray3D& R2, DoubleArray3D& R3,
         const double& dx );

};
```

MGsystem3D class solves a system defined by PDE_System3D using fast Multigrid solver.

```
class MGsystem3D
{

private:

      long nr;  // Number of  pre-smoothing relaxations,
          // i.e. while restricting the residual to coarse grid.

      long ns;  // Number of smoothing relaxations.
                    // Appropriate for two-grid, when numerous relaxations done on a coarse grid.

      long np;  // Number of post-smoothing relaxations,
                    // i.e. while prolongating the error to fine grid.

      long numVcyclesFMG; // Number of V-cycles per level for FMG.

      PDE_System3D* pPDE; // virtual

      void restriction( const DoubleArray3D& R, DoubleArray3D& R2, const long& order );

      void prolongation( const DoubleArray3D& V2, DoubleArray3D& V, const long& order );

      void correct( const DoubleArray3D& V, DoubleArray3D& u );


public:

      void setPDE( PDE_System3D* pde ) { pPDE = pde; }

      long U_RestrictionOrder;
      long R_RestrictionOrder;

      long solver;   // identifier for GS, Two-Grid, or Multigrid.

      MGsystem3D();

      void setNumberGSRelaxations( void );
      void setNumberGSRelaxations( long preSmooth, long Smooth, long postSmooth );

      void initialize( DoubleArray3D& A, const Grid3D& grid, const long& choice );

      void MultiGridCycle( DoubleArray3D& u1, DoubleArray3D& u2,DoubleArray3D& u3,
                const DoubleArray3D& f1, const DoubleArray3D& f2, const DoubleArray3D& f3, const double& dx );

      void MultiGridCycle_FAS( DoubleArray3D& u1, DoubleArray3D& u2, DoubleArray3D& u3,
                const DoubleArray3D& f1, const DoubleArray3D& f2, const DoubleArray3D& f3, const double& dx );

      void outputParameters( const double& accuracy, const Grid3D& grid,
                const long& TimeSteps, const long& outputCount, const double& timeTaken );
};
```

```
void Registration3D::evaluate_v( ... )
{
    ......

    StationaryNS3D PDEsystem;
    MGsystem3D MG;

    MG.setPDE(&PDEsystem);// passing in a reference (e.g. a pointer value)
                          // to the derived class
    ......
}
```

A pointer to PDE_System3D class would point to the address of PDEsystem (an object of StationaryNS3D), and would use this object's member functions and member variables.

Fluid Registration

- **3b) Velocity field v** can also be obtained from convolution of **f** with the Gaussian kernel:

$$\mathbf{v} = G_\sigma * \mathbf{f}(\mathbf{x}, \mathbf{u}).$$

  - Convolution is performed using FFTW.

- **Code:**

```
Registration3D::evaluate_v( f1, f2, f3, ... );
In: (f1,f2,f3)
Out: (f1,f2,f3), which is (v1,v2,v3)
```

Fluid Registration

- **4)** Solve for **displacement u**:

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{v} - \mathbf{v} \cdot \nabla \mathbf{u}$$

or, equivalently

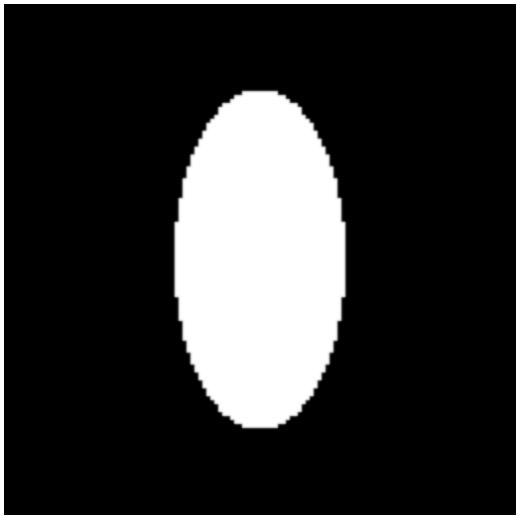
$$\frac{\partial \mathbf{u}}{\partial t} = \begin{bmatrix} v_1 - v_1 \frac{\partial u_1}{\partial x} - v_2 \frac{\partial u_1}{\partial y} - v_3 \frac{\partial u_1}{\partial z} \\ v_2 - v_1 \frac{\partial u_2}{\partial x} - v_2 \frac{\partial u_2}{\partial y} - v_3 \frac{\partial u_2}{\partial z} \\ v_3 - v_1 \frac{\partial u_3}{\partial x} - v_2 \frac{\partial u_3}{\partial y} - v_3 \frac{\partial u_3}{\partial z} \end{bmatrix}$$

- **Code:**

```
Registration3D::update_U( u1, u2, u3, v1, v2, v3, ... );
In: (v1,v2,v3)
Out: (u1,u2,u3)
```

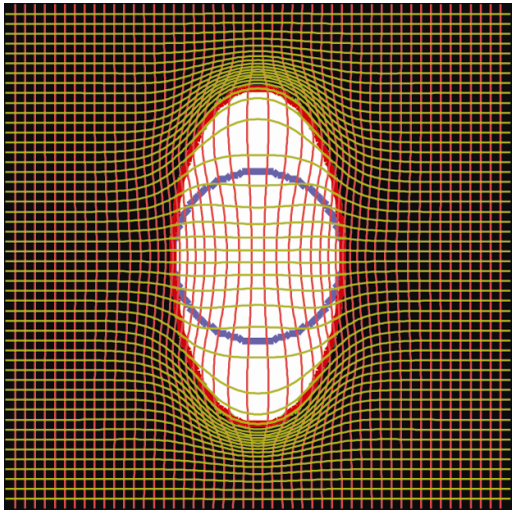
Disk to Ellipsoid Example

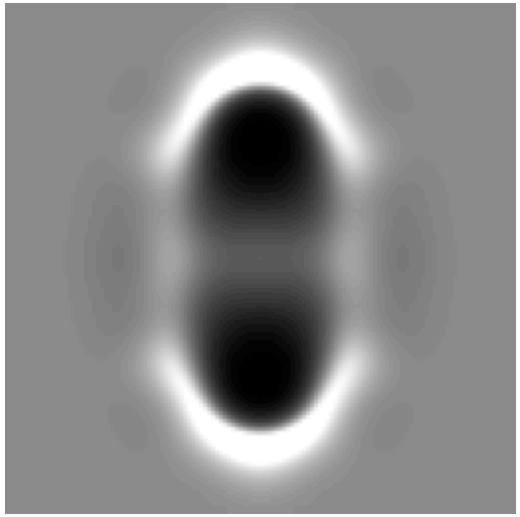
I_1



I_2

$I_2 \circ g$ and Deformed Grid

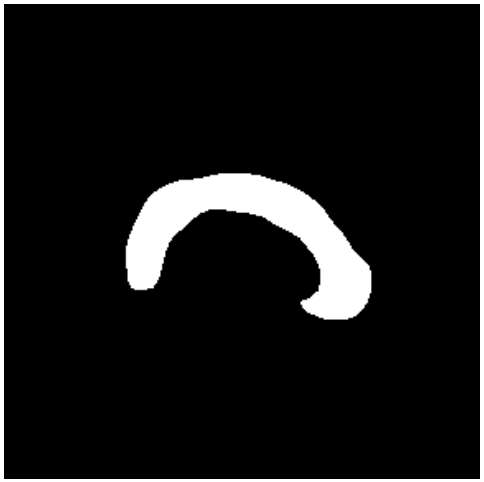


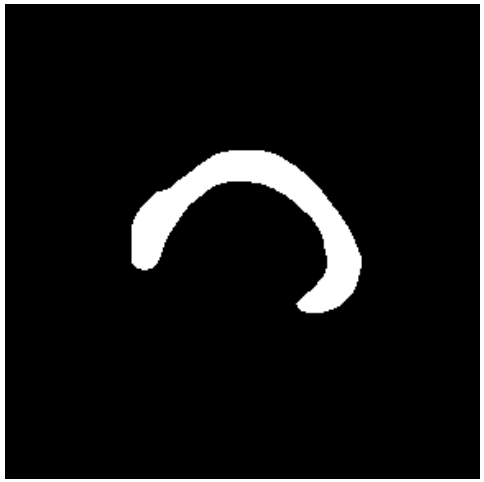


Jacobian map

Corpus Callosum Example

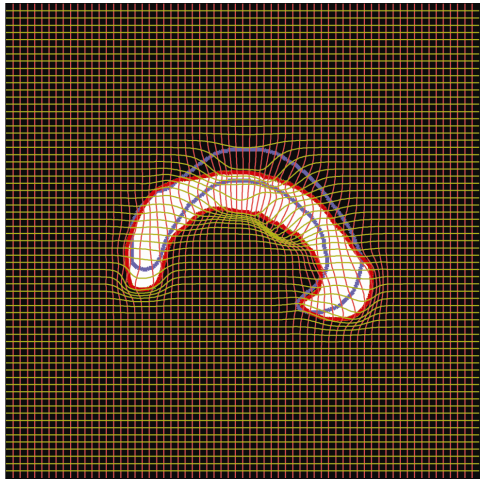
I_1





I_2

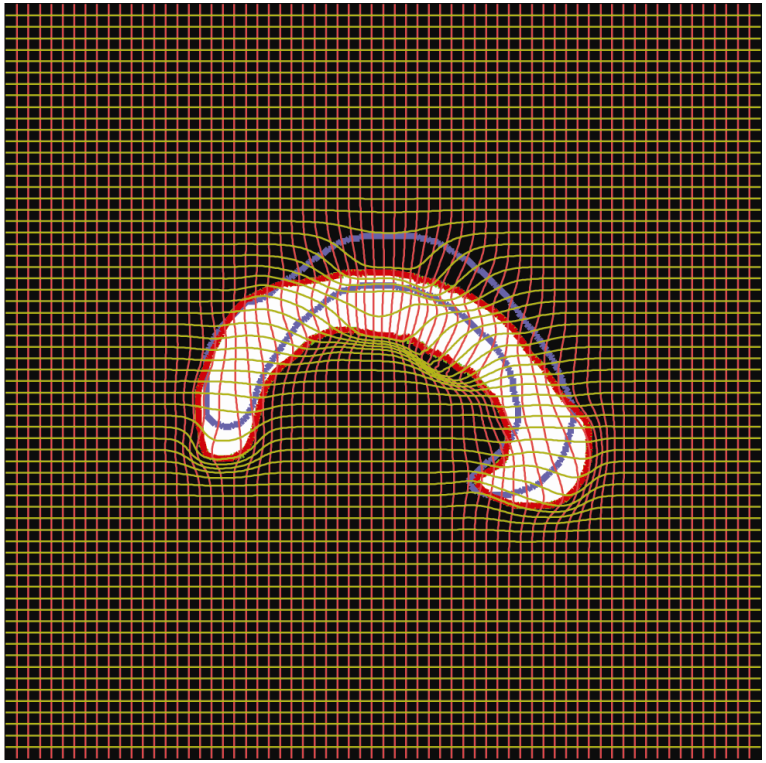
$I_2 \circ g$



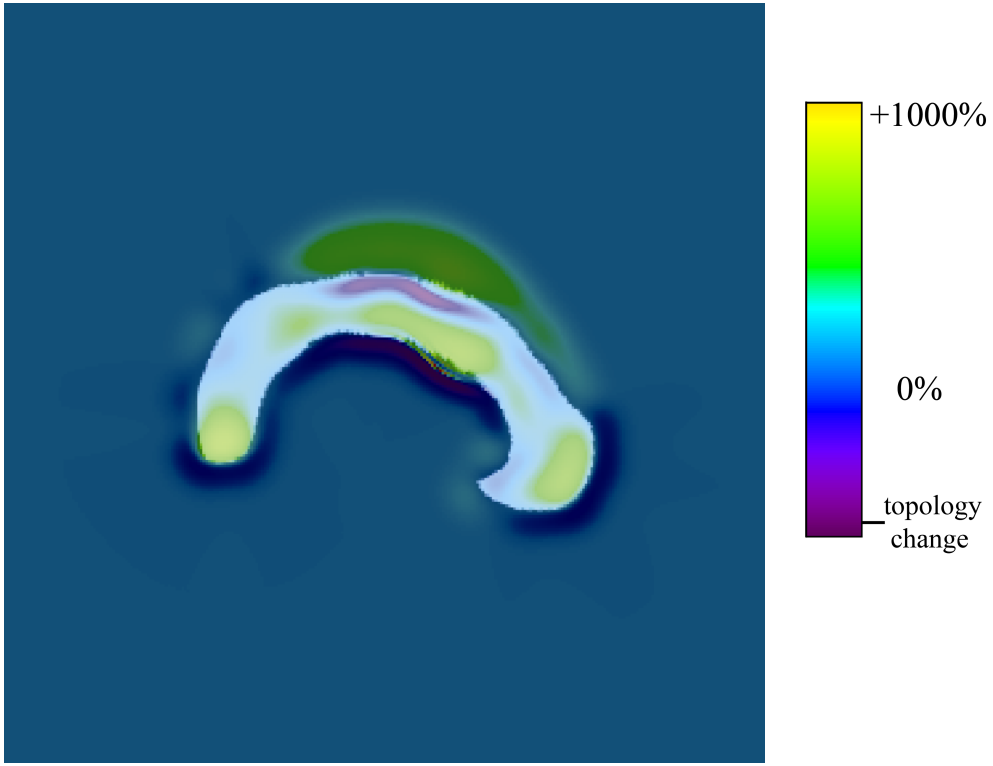
Deformed Grid

Corpus Callosum Example

Deformed Grid

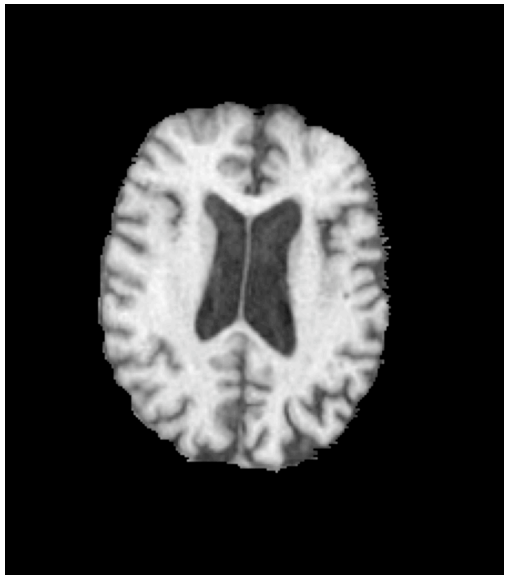


Jacobian Map



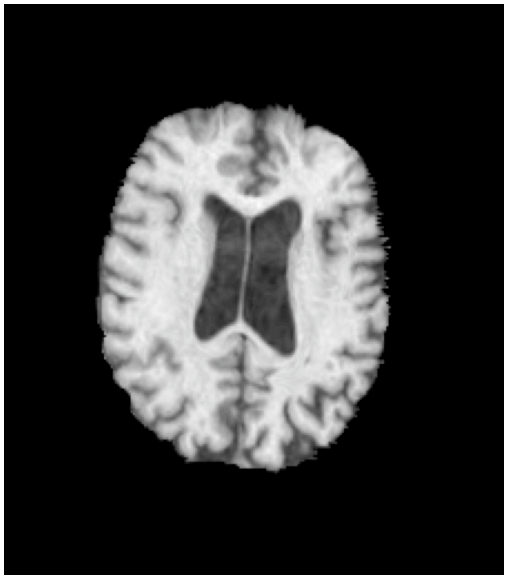
MRI Example

I_1



I_2

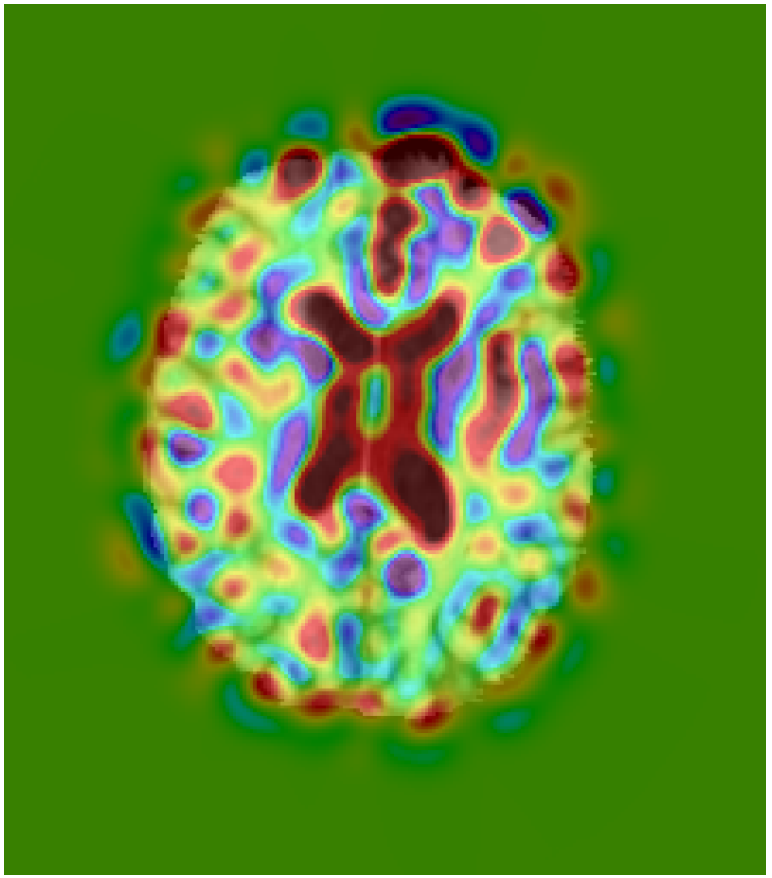
$I_2 \circ g$, L_2 - Fluid



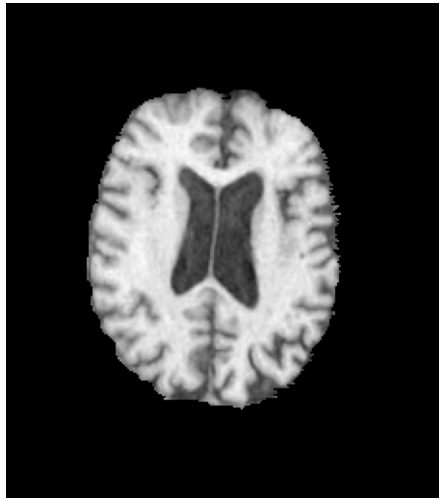
MRI Example

Deformed Grid

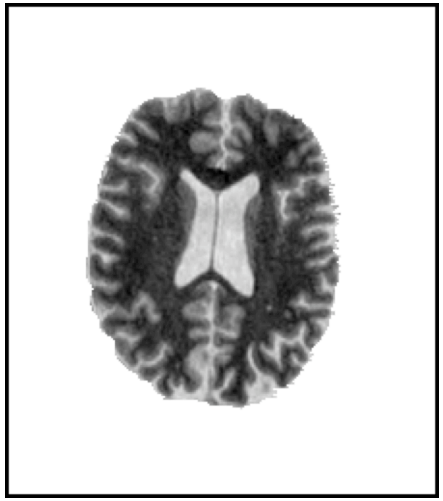
Jacobian Map



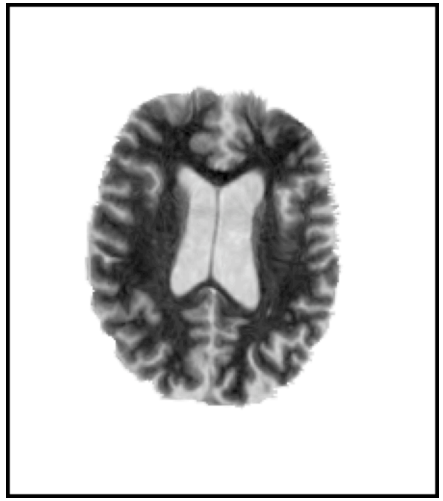
Mutual Information Matching Example 1



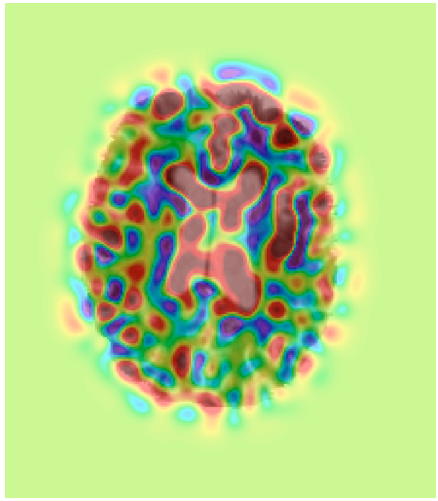
I_1



I_2

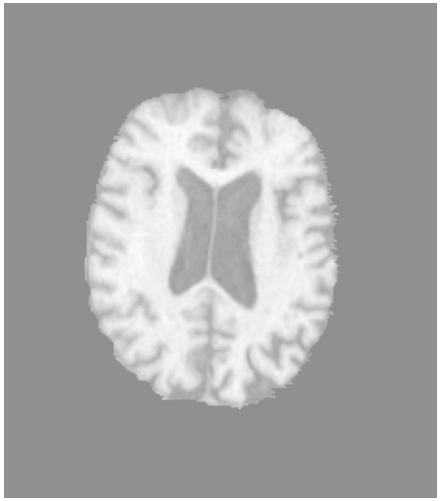


$I_2 \circ g$, *MI - Fluid*

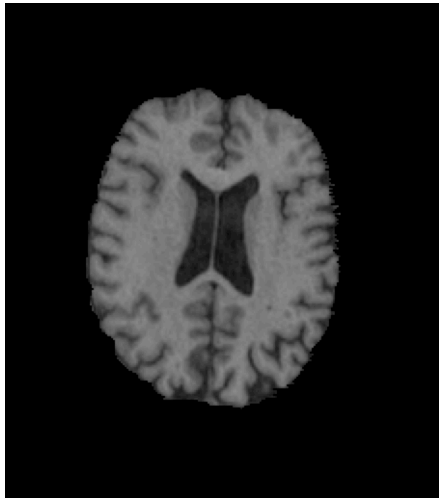


Jacobian map

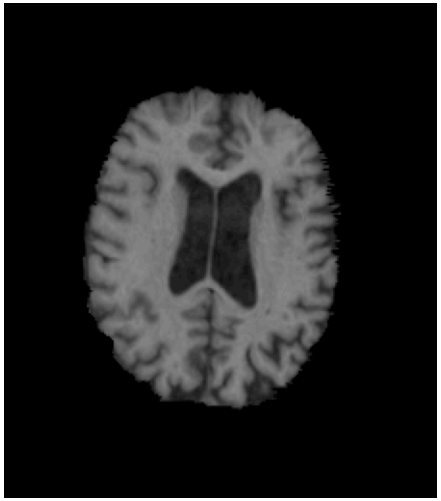
Mutual Information Matching Example 2



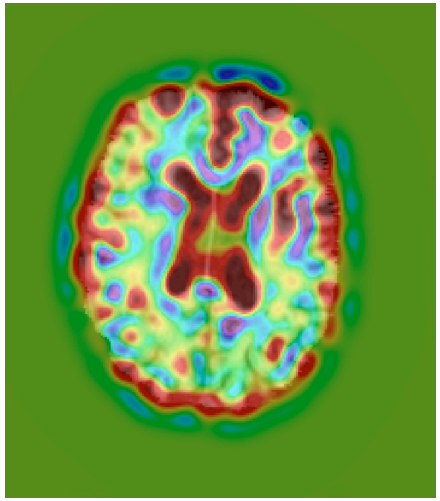
I_1



I_2



$I_2 \circ g$, *MI - Fluid*



Jacobian map

Thank You!