Spherical Demons Registration of Spherical Surfaces

Release 1.00

Luis Ibanez[1], Michel Audette[1], B. T. Thomas Yeo[1], Polina Goland[2]

August 4, 2009

[1]Kitware Inc., Clifton Park, NY
[2]CSAIL MIT, Boston, MA

Abstract

This document describes a contribution to the Insight Toolkit intended to support the process of performing deformable registration on two Meshes. The method implemented here is restricted to Meshes with a Spherical geometry and topology, and with scalar values associated to their nodes. The code described here is an implementation of the paper "*Spherical Demons: Fast Diffeomorphic Landmark-Free Surface Registration*" by Yeo, Sabuncu, Vercauteren, Ayache, Fischl and Golland [3, 4].

This paper is accompanied with the source code, input data, parameters and output data that we used for validating the algorithm described in this paper. This adheres to the fundamental principle that scientific publications must facilitate **reproducibility** of the reported results.

Contents

1 Introduction **2**

2 How to Build **2**
2.1 Building Executables and Tests 2
2.2 Building this Report 3

3 How to Use the Filter **3**
3.1 Basic Registration Source Code Example 3
3.2 How to Run 6

4 Results **7**

1 Introduction

The method described in this paper is the Mesh equivalent of the Demons Deformable registration method implemented for images that is currently available in the Insight Toolkit.

For example

- itk::DemonsRegistrationFilter
- itk::FastSymmetricForcesDemonsRegistrationFunction
- itk::DiffeomorphicDemonsRegistrationFilter

The method implemented here is restricted to the case of two Meshes with spherical geometry and topology, with scalar values associated to their nodes. The registration operates on the scalar values, not on the geometry of the mesh surface.

This contribution is the third on a series of papers related to improving support for mesh registration in the Insight Toolkit. Previous papers have covered the topics of rigid registration on Meshes and iterative smoothing of meshes with scalar and vector field values [2, 1]

2 How to Build

This contribution includes

- Source code of the Spherical Diffeomorphic Demons filter
- Tests for the filter
- Examples on how to use the filter
- All the LaTeX source files of this paper

2.1 Building Executables and Tests

In order to build the whole, it is enough to configure the directory with CMake. As usual, an out-of-source build is the recommended method.

In a Linux environment it should be enough to do the following:

- `ccmake SOURCE_DIRECTORY`
- `make`
- `ctest`

Where SOURCE_DIRECTORY is the directory where you have expanded the source code that accompanies this paper.

This will configure the project, build the executables, and run the tests and examples.

2.2 Building this Report

In order to build this report you can do

- `ccmake SOURCE_DIRECTORY`
- Turn ON the CMake variable
  - `BUILD_REPORTS`
- `make`

This should produce a PDF file in the binary directory, under the subdirectory `Documents/Report001`.

3 How to Use the Filter

This section illustrates the minimum operations required for running these two filters. The code shown here is available in the Examples directory of the code that accompanies this paper. You can download the entire set of files from the Insight Journal web site.

It is assumed that you would have already performed Rigid registration between the two meshes before you attempt to perform Deformable registration using the filter described below.

3.1 Basic Registration Source Code Example

The source code presented in this section can be found in the `Examples` directory under the filename

- QuadEdgeMeshSphericalDiffeomorphicDemonsFilter1

In order to use this filter we should start by including headers for the Demons registration filter, the reader and writer types and the `itk::QuadEdgeMesh` itself.

```
21 #include "itkQuadEdgeMeshSphericalDiffeomorphicDemonsFilter.h"
22 #include "itkQuadEdgeMeshScalarDataVTKPolyDataWriter.h"
23 #include "itkQuadEdgeMeshVTKPolyDataReader.h"
24 #include "itkQuadEdgeMesh.h"
```

The Scalar type associated with the nodes in the mesh, and the mesh dimension are defined in order to declare the Mesh type

```
39  typedef float       MeshPixelType;
40  const unsigned int Dimension = 3;
41
42  typedef itk::QuadEdgeMesh< MeshPixelType, Dimension >   FixedMeshType;
43  typedef itk::QuadEdgeMesh< MeshPixelType, Dimension >   MovingMeshType;
44  typedef itk::QuadEdgeMesh< MeshPixelType, Dimension >   RegisteredMeshType;
```

We declare the type of the registration filter and instantiate it.

```
46  typedef itk::QuadEdgeMeshSphericalDiffeomorphicDemonsFilter<
47    FixedMeshType, MovingMeshType, RegisteredMeshType >   DemonsFilterType;
48
49  DemonsFilterType::Pointer demonsFilter = DemonsFilterType::New();
```

In order to read the input mesh we declare a reader types for both the Fixed and Moving meshes, and create one instance of each one.

```
51  typedef itk::QuadEdgeMeshVTKPolyDataReader< FixedMeshType >   FixedReaderType;
52  typedef itk::QuadEdgeMeshVTKPolyDataReader< MovingMeshType >  MovingReaderType;
53
54  FixedReaderType::Pointer fixedReader = FixedReaderType::New();
55  fixedReader->SetFileName( argv[1] );
56
57  MovingReaderType::Pointer movingReader = MovingReaderType::New();
58  movingReader->SetFileName( argv[2] );
```

The output of the readers is passed as input to the mesh deformable registration filter.

```
72  demonsFilter->SetFixedMesh( fixedReader->GetOutput() );
73  demonsFilter->SetMovingMesh( movingReader->GetOutput() );
```

As described in the introduction, this filter is designed to operate on meshes representing a spherical geometry and spherical topology. In order to perform consistent computations, the filter requires the user to provide the coordinates of the sphere center as well as the sphere radius. This may seem to be redundant, since, obviously the filter could have estimated these parameters from the population of points in the mesh, however, such approach would require to invest the computation time of estimating those values, without the guaranty that the resulting estimates will be satisfactory.

In the current API of the filter, the user should provide the sphere center and the sphere radius by calling the methods `SetCenter()` and `SetRadius()` respectively. The values provided *must* correspond to the real parameters of the spherical meshes passed as input, otherwise all the deformation calculations will be incorrect. It is also a requirement that both the Fixed and Moving mesh will have exactly the same center and same radius.

```
78  demonsFilter->SetSphereCenter( center );
79  demonsFilter->SetSphereRadius( 100.0 );
```

Note that the type of the `center` variable can be taken as a trait of the filter type.

```
75  DemonsFilterType::PointType center;
```

As described in [3, 4] the demons filter has a set of parameters that control the behavior of the deformation field. The main parameters are

- Gamma
- SigmaX
- Number of Iterations
- Lambda
- Number of Smoothing Iterations

The parameter Gamma is the coefficient that multiplies the $2x2$ identity matrix in the Levenberg Marquardt modification of the Newton method. This parameter adds stability to the solver. The larger this value is, the smaller the deformations will be.

The parameter SigmaX is used to divide the Jacobian matrix term in the computation of the velocity field. The larger this parameter is, the larger will be the deformations of the resulting field.

The "Number of Iterations" parameter corresponds to the main iterative loop of the solver that computes updates of the velocity field and that compose them with the current deformation. The total computation time of the filter will be linearly proportional to this value.

The Lambda parameter is used in the smoothing of the deformation field. It affects the weights that will be used when computing the weighted average of the first ring of neighbors at every node. The effect of the Lambda value has been discussed in [2]. For small values of λ, for example, less than 0.1, the weight of the central pixels dominates the computation, and therefore a subtle smoothing is applied. On the other hand, for values of λ above 10.0 the weights of central value is almost the same as the weight of any of the neighbors which make the filter behave as a iterative average filter and produce a stronger smoothing effect.

The "Number of Smoothing Iterations" parameter is also discussed in [2]. The smoothing is performed iteratively by visiting all the nodes and computing a weighted average of the first ring neighbors. The more iterations are applied, the stronger the smoothing will be.

These parameters described above are set with the methods

- `SetGamma()`
- `SetSigmaX()`
- `SetMaximumNumberOfIterations()`
- `SetLambda()`
- `SetMaximumNumberOfSmoothingIterations()`

As shown in the following code

```
87  demonsFilter->SetGamma( gamma );
88  demonsFilter->SetSigmaX( sigmaX );
89  demonsFilter->SetMaximumNumberOfIterations( maximumNumberOfIterations );
90
91  demonsFilter->SetLambda( lambda );
92  demonsFilter->SetMaximumNumberOfSmoothingIterations( maximumNumberOfSmoothingIterations );
```

The execution of the filter can be triggered by calling the `Update()` method. This should typically be done inside a `try/catch` block, since it is possible that error conditions may generate exceptions.

```
95  try
96    {
97    demonsFilter->Update( );
98    }
99  catch( itk::ExceptionObject & exp )
100   {
101   std::cerr << exp << std::endl;
102   return EXIT_FAILURE;
103   }
```

Finally, the result of mapping the values of the Moving mesh onto the geometry of the Fixed mesh by using the deformation field that maps points from the Fixed Mesh into points of the Moving Mesh is obtained as the Output of the filter, and can be passed to a Mesh Writer.

```
106 typedef itk::QuadEdgeMeshScalarDataVTKPolyDataWriter< FixedMeshType >   WriterType;
107
108 WriterType::Pointer writer = WriterType::New();
109 writer->SetFileName( argv[3] );
110 writer->SetInput( demonsFilter->GetOutput() );
111
112
113 try
114   {
115   writer->Update();
116   }
117 catch( itk::ExceptionObject & excp )
118   {
119   std::cerr << excp << std::endl;
120   return EXIT_FAILURE;
121   }
```

3.2 How to Run

Once you have compiled the source code described in the previous section, you can run it from a command line of a console such as

- Unix shell
- GNU/Linux shell
- MS-DOS console window
- Visual Studio Command Prompt

In order to follow the rest of this section, you should be familiar with the command line operations of your platform.

The Subdirectory `Examples` contain the source file

- QuadEdgeMeshSphericalDiffeomorphicDemonsFilter1.cxx

When building the project this file generates an executable called

- QuadEdgeMeshSphericalDiffeomorphicDemonsFilter1

This executable expects the following four command line arguments

1. Input FixedMeshFileName (.vtk legacy format)
2. Input MovingMeshFileName (.vtk legacy format)
3. Output ResampledMeshFileName (.vtk legacy format)
4. gamma (real value)
5. sigmaX (real value)
6. lambda (real value)
7. numberOfSmoothingIterations (integer value)
8. numberOfIterations (integer value)

The results in the following section were generated with calls similar to

```
SmoothingMeshWithScalars fixedMeshWithScalarsIC1.vtk movingMeshWithScalarsIC1.vtk \
resampledMesh.vtk  1.0 1.0 1.0 1 300
```

4 Results

Figure 1 illustrates the two meshes passes as input to this test. Both meshes have radius of 100.0 units and are centered at the origin of coordinates. The scalar function on the surface of the mesh is a Gaussian of the angle ϕ scaled by a constant. In the case of the Fixed Mesh on the left, the constant used was 3.0 while in the case of the Moving Mesh a constant of 2.0 was used.

Figure 2 shows the output of the Demons registration filter. On the left, the values of the Moving mesh after being mapped to the geometry of the Fixed mesh by using the deformation field. On the right, the deformation field computed by the filter.

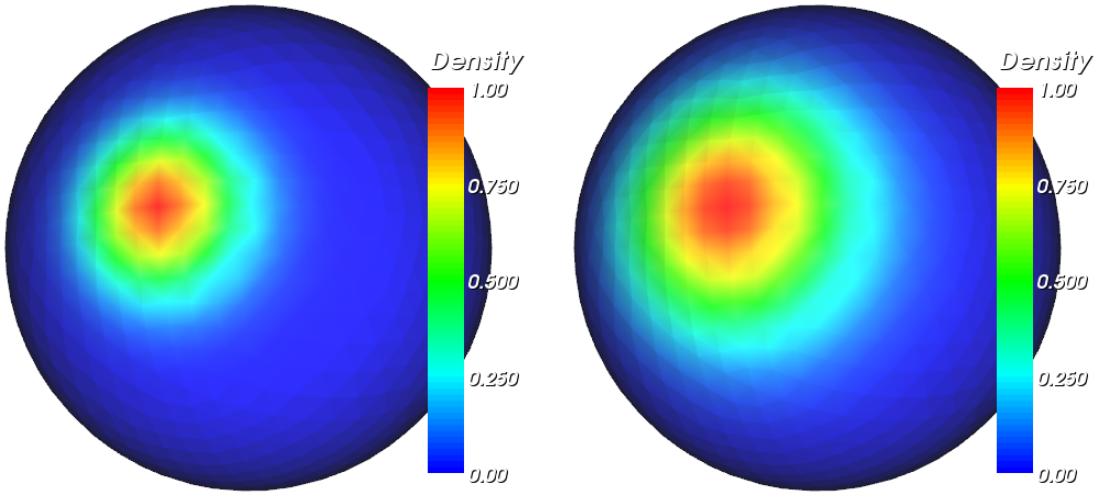

Figure 1: Fixed Mesh (left) and Moving Mesh (right) passed as input to the Demons filter.

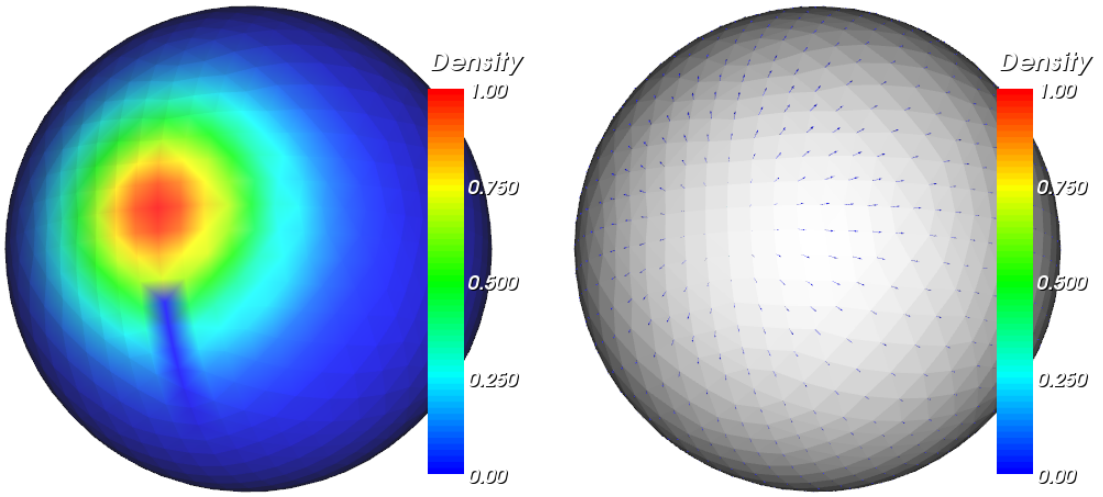

Figure 2: Registered Mesh (left) and Deformation field (right) produced as output by the Demons filter.

References

[1] L. Ibanez, M. Audette, B. T. T. Yeo, and P. Golland. Rotational registration of spherical surfaces represented as quadedge meshes. *Insight Journal*, 2009. 1

[2] L. Ibanez, B. T. T. Yeo, and P. Golland. Iterative smoothing of field data in spherical meshes. *Insight Journal*, 2009. 1, 3.1

[3] B. T. T. Yeo, M. Sabuncu, T. Vercauteren, N. Ayache, B. Fisch, and P. Golland. Spherical demons: Fast surface registration. In *MICCAI'2008 International Conference on Medical Image Computing and Computer-Assisted Intervention*, Lecture Notes in Computer Science, pages 745–753, September 2008. (document), 3.1

[4] B. T. T. Yeo, M. Sabuncu, T. Vercauteren, N. Ayache, B. Fisch, and P. Golland. Spherical demons: Fast diffeomorphic landmark-free surface registration. *In Review: IEEE TMI*, 2009. (document), 3.1