Iterative Smoothing of Field Data in Spherical Meshes

Release 1.10

Luis Ibanez[1], B. T. Thomas Yeo[2], Polina Golland[2]

July 16, 2009

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

Abstract

This document describes a contribution to the Insight Toolkit intended to smooth the values of Field data associated with the nodes of a Spherical Mesh. The Mesh Smoothing filters contributed here do not modify the geometry or the topology of the Mesh. They act only upon the pixel data values associated with the nodes. Two filters are presented, one that smooths scalar field data, and a second one that smooths vector field data.

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

A family of classes has been added recently to the Insight Toolkit for represented oriented 2D manifolds embedded in a 3D space, objects colloquially known as "Surfaces". These classes use the QuadEdge representation in order to ensure the consistency of the orientation all over the surface.

Most of the filters available for this type of surface representation act upon the geometry and topology of the Meshes. However, the Meshes are also capable of carrying data associated with their nodes and therefore there is a need for filters that process such associated data. The collection of values associated with every node of a Mesh is usually called a "Field". In the Insight Toolkit we have commonly referred to it as "PixelData" which is a bit less intuitive but more software-oriented. Following the first terminology, in this report we refer to the point data as a "Field". This terminology is used in the Visualization Toolkit **VTK** as well.

In this paper we contribute new classes that can be used for smoothing the values of field data over the surface of the Mesh. The Field data can be Scalar or Vector. Specific smoothing filters are provided for each case.

This paper is the second on a series related to implementing support for deformable registration of Meshes according the method described in [1]. This reference also provide a detailed description of the algorithm implemented here. This report will focus on the software implementation and its usage.

2 Overview

The filters described in this report operate on the Field data, or Pixel Data, of a Mesh. The Meshes are expected to be of type `itk::QuadEdgeMesh`.

2.1 Algorithm

The basic algorithm implemented in these classes is described in [1]. In summary the process involve visiting all the node of the Mesh, and for each one of them, computing a new scalar value by using a weighted average of the scalar values of the immediate neighbors.

Weighted Average

A simply scheme of two-weights is used. The weight associated to the central value is:

λ	Neighbor Weight	Central Weight
0.03125	0.00000	1.00000
0.06250	0.00033	0.99799
0.12500	0.01650	0.90099
0.25000	0.07469	0.55187
0.50000	0.11470	0.31179
1.00000	0.13074	0.21556
2.00000	0.13729	0.17628
4.00000	0.14019	0.15886
8.00000	0.14155	0.15068

Table 1: Effect on Lambda on the weights used for computing the weighted-average of neighbors in a node with six neighbors.

$$\frac{1}{1+|N_i|\exp\left(-\frac{1}{2\lambda}\right)} \tag{1}$$

While of the neighbors values will use weights equal to

$$\frac{\exp\left(-\frac{1}{2\lambda}\right)}{1+|N_i|\exp\left(-\frac{1}{2\lambda}\right)} \tag{2}$$

Where N_i is the number of neighbors of this i-th node, and λ is a user-provided constant that applies to the entire mesh.

This process of local weighted average is applied on all the nodes and this is done in iteratively until reaching a use-provided number of iterations. The effect of the λ parameter is such that the field data is smoothed more when the values of λ are large. Typical values of λ are in the range $[0.10, 10.0]$, although they are not restricted to that range at all.

As an illustration, Table1 presents the weight values of the central node and its neighbors for different values of λ, on the typical case of a neighborhood of six nodes.

As the table shows, 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.

Iterations

The smoothing filter does not have a stopping criteria based on convergence. It simply runs as many iterations as specified by the user. It is the user's responsibility to choose how many iterations must be run.

2.2 Smoothing Scalars

Smoothing of Meshes with a Scalars field is implemented in the filter

itk::QuadEdgeMeshScalarPixelValuesSmoothingFilter

This filter is suitable for a QuadEdgeMesh whose PixelType has a single component. Typical examples will use float or double as pixel types.

The global use of the filter requires the user to

- Connect an input QuadEdgeMesh of Pixel Scalar type
- Set the values of the λ parameter
- Set the number of iterations.

2.3 Smoothing Vectors

Smoothing of Meshes with a Vector field is implemented in the filter

itk::QuadEdgeMeshVectorPixelValuesSmoothingFilter

This filter is suitable for a QuadEdgeMesh whose PixelType has a exactly three component. The geometrical implementation of the smoothing process is very specific to Vectors and will not be appropriate for arbitrary arrays that are not representing a geometrical Vector in space. For example, this method will not be appropriate for a Mesh whose pixel data are RGB color triplets.

The particularity of this filter is that before the weighted average of the Vectors is computed, each one of the vectors from neighbor nodes is parallel-transported to the central node. The reason for doing this is that vectors located at different points in the mesh are not comparable specially when the mesh has non-zero curvature. The process of Parallel-Transform is performed under the assumption that the surface has a Spherical geometry. Therefore, this filter is only appropriate for Spherical surfaces.

Parallel Transport from point p_1 to point p_2 of the surface is implemented by sliding the Vector along the greatest circle that connects points p_1 and p_2. The filter expects the user to provide the coordinates of the Sphere center as well as its radius. No verification if performed on these values.

In summary, in order to use this filter, the user should

- Connect an input QuadEdgeMesh of Vector Scalar type
- Set the values of the λ parameter
- Set the number of iterations.
- Set the coordinates of the Sphere center
- Set the Sphere radius

3 How to Build

This contribution includes

- Source code of the Scalar and Vector Mesh smoothing filters
- Source code of a Mesh Writer (to .vtk legacy format)
- Tests for the filters and the writer
- Examples on how to smooth a Scalar Mesh and a Vector Mesh
- All the LaTeX source files of this paper

3.1 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 QuadEdgeMeshFieldSmoothingFiltersSourceDirectory`
- `make`
- `ctest`

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

3.2 Building this Report

Once the tests and examples run, the images shown in this report can be generated by doing:

- `ccmake QuadEdgeMeshFieldSmoothingFiltersSourceDirectory`
- Turn ON the CMake variables
  - `USE_VTK`
  - `GENERATE_PAPER_SCREENSHOTS`
- `make`
- `ctest`

This run of ctest will generate a collection of screen shots that will be saved as PNG files in the `Testing/Temporary` subdirectory.

Finally, in order to build this report you can do

- `ccmake QuadEdgeMeshFieldSmoothingFiltersSourceDirectory`
- Turn ON the CMake variable
  - `GENERATE_REPORT`
- `make`

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

4 How to Use the Filters

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.

4.1 Scalar Mesh

Source Code Example

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

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

Then we declare the specific instantiations of each one of these types. Note that the pixel type used for the QuadEdgeMesh is a scalar type in this case.

```
36   const unsigned int Dimension = 3;
37   typedef float      MeshPixelType;
38
39   typedef itk::QuadEdgeMesh< MeshPixelType, Dimension >   InputMeshType;
40   typedef itk::QuadEdgeMesh< MeshPixelType, Dimension >   OutputMeshType;
```

We declare the type of the smoothing filter, and construct and instance of it.

```
48   typedef itk::QuadEdgeMeshScalarPixelValuesSmoothingFilter<
49     InputMeshType, OutputMeshType >                       FilterType;
50
51   FilterType::Pointer filter = FilterType::New();
```

Then we set its parameters: lambda and number of iterations.

```
53   const double lambda = atof( argv[3] );
54   const unsigned int numberOfIterations = atoi( argv[4] );
55
56   filter->SetLambda( lambda );
57   filter->SetMaximumNumberOfIterations( numberOfIterations );
```

Typical pipeline connections are set by calling SetInput() and GetOutput() methods. In this case we simply read the mesh and write its smoothed version.

```
63   filter->SetInput( reader->GetOutput() );
64   writer->SetInput( filter->GetOutput() );
65
66   filter->Update();
```

How to Run

The Subdirectory `Examples` contain the source file

- SmoothingMeshWithScalars.cxx

When building the project this file generates an executable called

- SmoothingMeshWithScalars

This executable expects the following four command line arguments

1. InputMeshFileName (.vtk legacy format)
2. OutputMeshFileName (.vtk legacy format)
3. lambda (real value)
4. numberOfIterations (integer value)

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

```
SmoothingMeshWithScalars meshWithScalarsIC1.vtk  smoothed.vtk 2.0  5
```

Which means that a value of lambda equals to 2.0 was used along a number of iterations equal to five.

Results

The diagram in Figure 1 presents the results of running the smoothing filters for multiple values of lambda and multiple iterations on a surface generated from the Icosahedron after one subdivision (IC1).

The diagram in Figure 2 presents the results of running the smoothing filters for multiple values of lambda and multiple iterations on a surface generated from the Icosahedron after four subdivision (IC4).

In both cases, the initial surface has all its point values set to zero except for a single node that is set to the value 10.0.

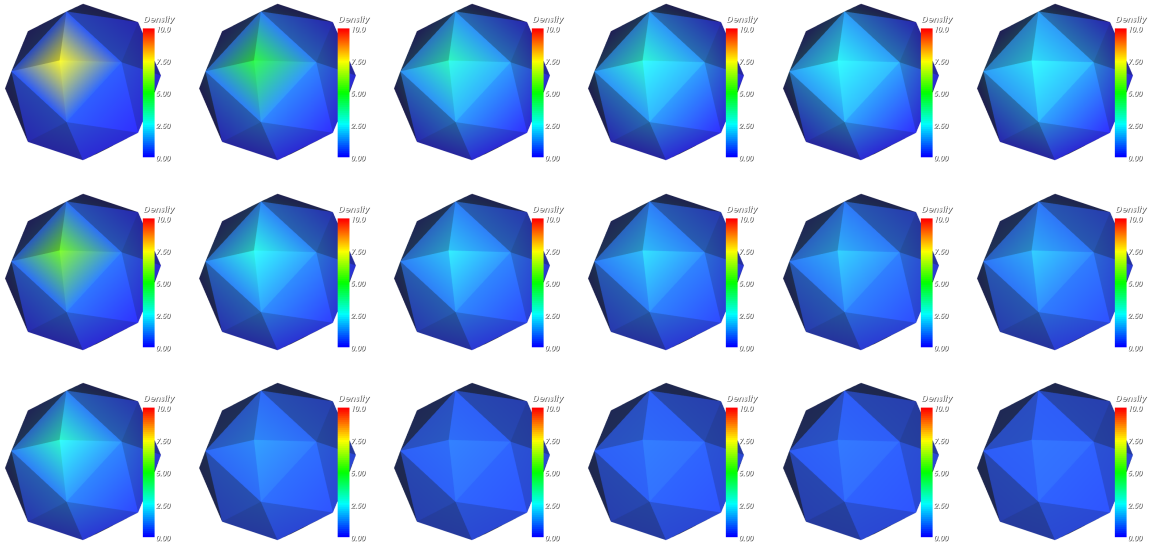

Figure 1: Results of running the Smoothing filter in an IC1 surface for multiple values of lambda and number of iterations. From left to right each column was executed with lambda values 0.2, 0.4, 0.8, 1.0, 2.0 and 4.0 respectively. From top to bottom each row was executed with a number of iterations equal to 1,2 and 5 respectively.

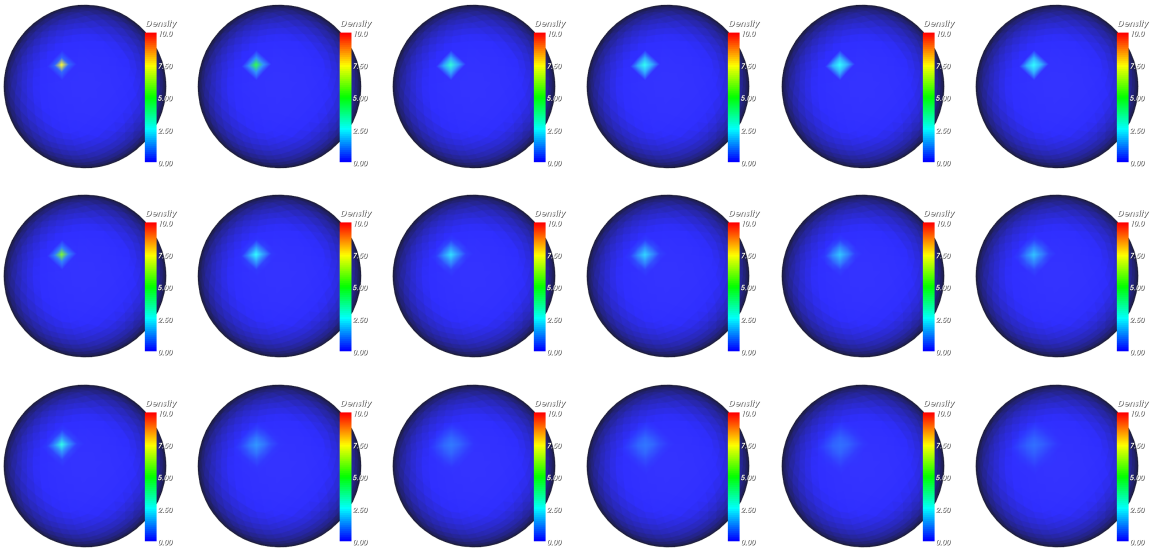

Figure 2: Results of running the Smoothing filter in an IC4 surface for multiple values of lambda and number of iterations. From left to right each column was executed with lambda values 0.2, 0.4, 0.8, 1.0, 2.0 and 4.0 respectively. From top to bottom each row was executed with a number of iterations equal to 1,2 and 5 respectively.

4.2 Vector Mesh

Source Code Example

In order to use this filter we should start by including headers for the smoothing filter, the reader and writer types and the `itk::QuadEdgeMesh` itself. Note that a different version of the PolyData writer class is needed here in order to manage writing of meshes whose pixel types are vectors.

```
#include "itkQuadEdgeMeshVectorPixelValuesSmoothingFilter.h"
#include "itkQuadEdgeMeshVectorDataVTKPolyDataWriter.h"
#include "itkQuadEdgeMeshVTKPolyDataReader.h"
#include "itkQuadEdgeMesh.h"
```

Then we declare the specific instantiations of each one of these types. Note that the pixel type used for the QuadEdgeMesh is a Vector type in this case. The `itk::Vector` class has a very specific meaning in ITK. It is the class that represents the Geometrical concept of the subtraction between to points in space. This is particularly important for the operation of parallel transport that is performed internally as part of the execution of this smoothing filter.

```
  const unsigned int Dimension = 3;
  typedef itk::Vector< float, Dimension >   MeshPixelType;

  typedef itk::QuadEdgeMesh< MeshPixelType, Dimension >   InputMeshType;
  typedef itk::QuadEdgeMesh< MeshPixelType, Dimension >   OutputMeshType;
```

We declare the type of the smoothing filter, and construct and instance of it.

```
  typedef itk::QuadEdgeMeshVectorPixelValuesSmoothingFilter<
    InputMeshType, OutputMeshType >                       FilterType;

  FilterType::Pointer filter = FilterType::New();
```

The filter expects an input surface with a spherical geometry. The user must provide the center and radius of this spherical mesh surface by calling the methods `SetSphereCenter()` and `SetSphereRadius()`. [1]

```
  sphereCenter.Fill( 0.0 );

  const double sphereRadius = 100.0;

  filter->SetSphereCenter( sphereCenter );
```

Then we set its parameters: lambda and number of iterations.

[1] **Implementation Note:** This filter could have computed and estimation of the radius and the location of the sphere center, but such estimation would require a non-negligible amount of computation time. It may still be desirable to have such implementation as one possible mode of operation for the filter.

```
60  const double lambdaValue = atof( argv[3] );
61  const unsigned int numberOfIterations = atoi( argv[4] );
62
63  filter->SetLambda( lambdaValue );
64  filter->SetMaximumNumberOfIterations( numberOfIterations );
```

The type of the mesh writer that is capable of managing vectors as point data, is defined and one instance is created.

```
66  typedef itk::QuadEdgeMeshVectorDataVTKPolyDataWriter< OutputMeshType >   WriterType;
67
68  WriterType::Pointer writer = WriterType::New();
69  writer->SetFileName( argv[2] );
```

Typical pipeline connections are set by calling SetInput() and GetOutput() methods. In this case we simply read the mesh and write its smoothed version.

```
71  filter->SetInput( reader->GetOutput() );
72  writer->SetInput( filter->GetOutput() );
```

and the execution of the pipeline is triggered by calling the `Update()` method on the filter at the end of the pipeline. In this case, the writer.

```
76  try
77    {
78    writer->Update();
79    }
80  catch( itk::ExceptionObject & excp )
81    {
82    std::cerr << excp << std::endl;
83    return EXIT_FAILURE;
84    }
```

How to Run

The Subdirectory `Examples` contain the source file

- SmoothingMeshWithVectors.cxx

When building the project this file generates an executable called

- SmoothingMeshWithVectors

This executable expects the following four command line arguments

1. InputMeshFileName (.vtk legacy format)
2. OutputMeshFileName (.vtk legacy format)
3. lambda (real value)
4. numberOfIterations (integer value)

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

```
SmoothingMeshWithVectors meshWithVectorsIC1.vtk  smoothed.vtk 2.0  5
```

Which means that a value of lambda equals to 2.0 was used along a number of iterations equal to five.

Results

The diagram in Figure 3 presents the results of running the smoothing filters for multiple values of lambda and multiple iterations on a surface generated from the Icosahedron after one subdivision (IC1).

The diagram in Figure 4 presents the results of running the smoothing filters for multiple values of lambda and multiple iterations on a surface generated from the Icosahedron after four subdivision (IC4).

In both cases, the initial surface has all its point values set to Vectors of three components, with all their components set to zero, except for a single node that has an axis aligned vector with a magnitude equal to 10. The radius of the sphere is equal to 100.0, and it is centered at the origin of coordinates. The figures show how the smoothing filter propagates the Vector across the surface, producing smaller vectors that are parallel-transported from the original single non-null vector.

Figure 5 presents the results of running the Vector smoothing for a lambda value of 1.0, and longer number of iterations. In this figure we can observe how the vectors are parallel-transported as their values propagate to neighbor nodes through the iterations of the filter.

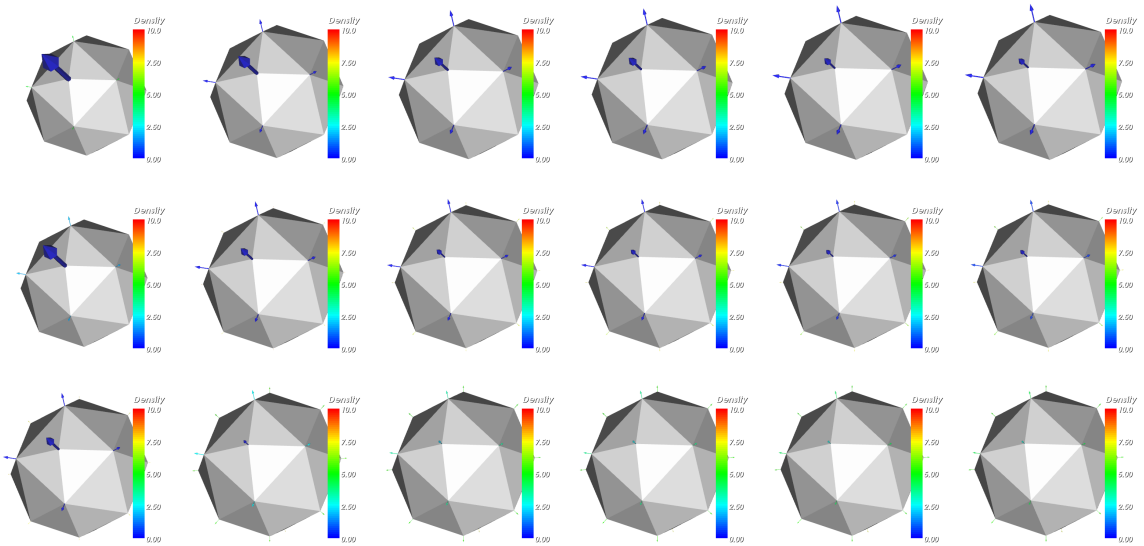

Figure 3: Results of running the Smoothing filter in an IC1 surface for multiple values of lambda and number of iterations. From left to right each column was executed with lambda values 0.2, 0.4, 0.8, 1.0, 2.0 and 4.0 respectively. From top to bottom each row was executed with a number of iterations equal to 1,2 and 5 respectively.

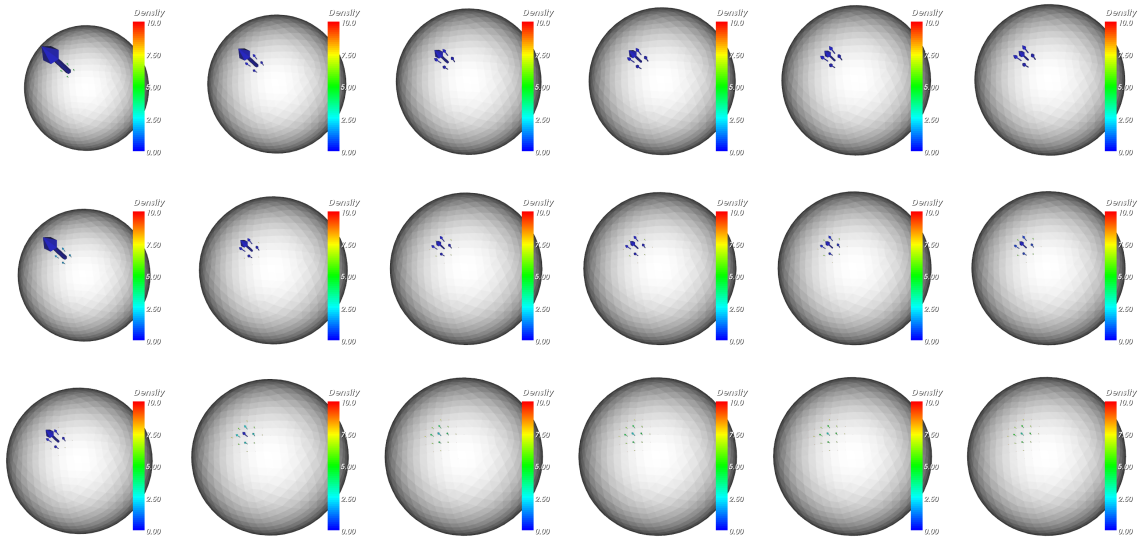

Figure 4: Results of running the Smoothing filter in an IC4 surface for multiple values of lambda and number of iterations. From left to right each column was executed with lambda values 0.2, 0.4, 0.8, 1.0, 2.0 and 4.0 respectively. From top to bottom each row was executed with a number of iterations equal to 1,2 and 5 respectively.

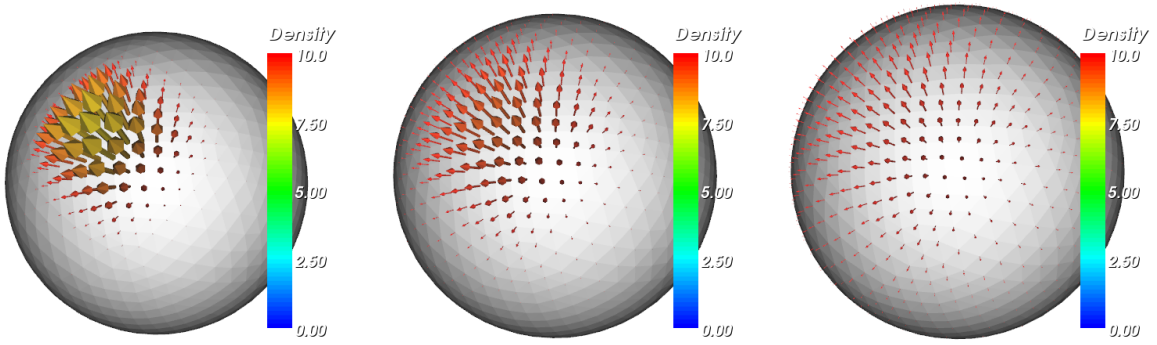

Figure 5: Results of running the Smoothing filter in an IC4 surface for a lambda value of 1.0 and number of iterations (from left to right): 20, 40 and 80. The vectors are scaled by a factor of x400 to make them visible.

References

[1] 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. 1, 2.1