Robust Scattered Data Points Approximation Using Finite Element Biomechanical Model

Yixun Liu, Nikos Chrisochoides

CRTC Lab and Computer Science Department,
Old Dominion University, Norfolk, VA 23529
{yxliuwm, npchris}@gmail.com

Abstract Many enabling technologies like non-rigid registration in medical image computing rely on the construction of a function by interpolating scattered points; however, the outliers contained in the data and the approximation error make the robust and accurate estimation difficult. This paper presents an ITK implementation of a robust Finite Element (FE) solver, which can effectively deal with the above difficulties. The experiment results of synthetic data and real cases demonstrate the characteristics of the robust solver and its typical application.

1 Introduction

In this paper, we present an ITK implementation of a robust Finite Element solver [1], which is characterized by 1) approximation to interpolation, and 2) robust against outliers. The robust solver uses a parameterized piece-wise linear polynomials to represent the unknown function. In future, we plant to provide an option for the user to choose different higher order polynomials. The parameters are estimated by approximating scattered data points. To deal with sparsity of the data, the parameter estimation is regularized by a biomechanical model, which is capable of describing the entire behavior of the system based on quite few data, i.e., the boundary condition. To make the estimation robust again outliers, this solver performs the parameter estimation as a Least Trimmed Squares (LTS) regression [3]. More specifically, at each iteration, estimate parameters first without any outliers, then identify the points with larger error as outliers, finally remove outliers from the data and re-estimate the parameters. To reduce the approximation error, this solver allows the mesh to iteratively approach the scattered data. An additional external force is applied to the FE model to reset or counteract the strain energy of the model in order to enable the model to be further deformed. In this paper, we first describe the principle of the robust solver. Then, we will present its ITK implementation of two filters: itk::fem::RobustSolver and itk::fem::FEMScatteredDataPointSetToImageFilter. RobustSolver undertakes concrete work, and FEMScatteredDataPointSetToImageFilter undertakes some “dirty” work to facilitate the use of the RobustSolver. At last, the experiments on both synthetic and real data are provided along with corresponding ITK codes.

The reimplementation of the solver not only inherits the coherent characteristics of the original solver, i.e., approximation to interpolation and robustness, but also enables the original solver to be easily adapted to different geometry domains and physical problems, owing to the flexible implementation of the itk::fem library. Currently, except built-in 2D quadrilateral and 3D hexahedral meshes, the RobustSolver supports 2D triangle and 3D tetrahedral meshes as input in dealing with the linear elasticity problem. It is easy to extend this RobustSolver to other geometry domains and physical problems by choosing appropriate FE elements (see itk::fem::Element).

2. Method

Given scattered point set $S = \{s_i\}_{i=1}^{p} \in R^D, D = 2,3$ and the displacement d_i associated with s_i, the scattered data approximation is formalized as a minimization of the following energy function:

$$W = \int_{\Omega} \sigma(u)\varepsilon(u)d\Omega + \lambda \sum_{i=1}^{p} \left\| u(s_i) - d_i \right\|^2 \tag{1}$$

where u is the unknown function. The first term, regularization term, describes the stain energy of a linear elastic biomechanical model, and the second terms describes the degree of the matching between the estimated data and the observation. λ controls the balance of these two terms. It is difficult to find the analytical solution of equation (1). Finite Element Method is used to approximate u with $\sum_{i=0}^{M} N_i U_i$, where M is the number of the vertices of the finite element, N_i is the shape function associated with node i, and U_i is node displacement vector. As a result, equation (1) can be discretized as:

$$W = U^T KU + (HU - D)^T S(HU - D) \tag{2}$$

where U is the vector of the concatenation of U_i with a size of $3n$ for 3D space. n is the number of vertices of the mesh. D is the vector of the concatenation of d_i . K is the mesh stiffness matrix of size $3n \times 3n$. The building of K has been well documented in [2]. H is the linear interpolation matrix of size $3p \times 3n$. Each registration point o_k with number k contained in the tetrahedron with vertex number c_i, $i \in [0:3]$ has contribution to four 3×3 submatrices: $[H]_{kc_0}, [H]_{kc_1}, [H]_{kc_2}, [H]_{kc_3}$ **.** The diagonal matrix $[H]_{kc_i}$ is defined as: $[H]_{kc_i} = diag(h_i, h_i, h_i)$. The linear interpolation factor h_i can be obtained by

$$\begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix} = \begin{bmatrix} v_{c0}^x & v_{c1}^x & v_{c2}^x & v_{c3}^x \\ v_{c0}^y & v_{c1}^y & v_{c2}^y & v_{c3}^y \\ v_{c0}^z & v_{c1}^z & v_{c2}^z & v_{c3}^z \\ 1 & 1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} o_k^x \\ o_k^y \\ o_k^z \\ 1 \end{bmatrix} \tag{3}$$

where v_{c_i} is the vertex with number c_i. S is the matching stiffness matrix of size 3p × 3p. S is an extension to the classical diagonal stiffness matrix, taking into account the matching confidence and the local structure distribution [1]. These measures are introduced through the matrix S , whose 3 × 3 sub-matrix S_k corresponding to registration point k is defined as:

$$S_k = \lambda \frac{n}{p} c_k T_k \tag{4}$$

where T_k is tensor structure of the block surrounding the point, which allows us to only consider the matching direction collinear to the orientation of the intensity gradient in the block [1]. In ITK implementation, point set of registration point $S = \{s_i\}_{i=1}^{p} \in R^D$ is a mandatory input, and point set of correspondence $C = \{c_i\}_{i=1}^{p} \in R^D$ and point set of tensor $T = \{t_i\}_{i=1}^{p} \in R^D$ are optional.

The equation (2) can be solved by

$$\frac{\partial W}{\partial U} = [K + H^T SH]U - H^T SD = 0 \tag{5}$$

leading to the linear system:

$$[K + H^T SH]U = H^T SD \tag{6}$$

The above approximation formulation performs well in the presence of outliers but suffers from a systematic error. Alternatively, solving the exact interpolation problem based on noisy data is not adequate. The robust solve can take advantage of both approximation and interpolation to iteratively estimate the deformation from the approximation to the interpolation while rejecting outliers. The gradual convergence to the interpolation solution is achieved through the use of an external force F added to the approximation formulation of Equation (6), which balances the internal mesh stress:

$$[K + H^T SH]U = H^T SD + F \tag{7}$$

This force F is computed at each iteration i to balance the mesh internal force KU_i, which leads to the iterative scheme:

$$\begin{aligned} F_i &\Leftarrow KU_i \\ U_{i+1} &\Leftarrow [K + H^T SH]^{-1}[H^T SD + F_i] \end{aligned} \tag{8}$$

In ITK implementation, the number of the approximation to interpolation steps is specified by the user. Due to potential outliers, Least Trimmed Squares robust regression is used to reject a fraction of the total registration points based on an error function,

$$\xi_k = \left\| S_k[(HU)_k - D_k] \right\| \tag{9}$$

which measures the error between the estimated displacement and the real displacement. To make the error independent of the current estimated displacement, error function (9) is normalized as,

$$\frac{\xi_k = \left\| S_k[(HU)_k - D_k] \right\|}{\lambda \left\| (HU)_k \right\| + 1} \tag{10}$$

The number of rejection steps based on this error function and the fraction of rejection per iteration are defined by the user.

3. ITK Implementation

FEMScatteredDataPointSetToImageFilter and RobustSolver implement the solver presented in [1]. FEMScatteredDataPointSetToImageFilter is a wrapper of RobustSolver.

FEMScatteredDataPointSetToImageFilter is used to facilitate the use of RobustSolver by converting natural inputs such as mesh and feature points into specific FEMObject, providing built-in 2D and 3D rectilinear meshes, invoking RobustSolver to resolve the solution to produce a deformed FEMObject, and converting the deformed FEMObject into a deformation field. FEMRobustSolver takes a FEMObject as input, then iteratively approximates the data (displacement) associated with the feature points while rejecting outliers, and finally outputs a deformed FEMObject.

3.1 FEMScatteredDataPointSetToImageFilter

Figure 1 shows the flow chart and the inheritance diagram of this filter, respectively. FEMScatteredDataPointSetToImageFilter provides a built-in 2D quadrilateral and 3D hexahedron mesh if the input mesh is not available. Otherwise, just simply passes the input mesh to the converter. The natural inputs of the RobustSolver are mesh and point sets including mandatory feature points and optional confidence and tensor. itk FEM library requires a FEMObject as input. FEMScatteredDataPointSetToImageFilter converts the mesh and point sets into a FEMObject, which is undertaken by a member function:

```
InitializeFEMObject(FEMObjectType * femObject)
{
  this->InitializeMaterials(femObject);
  this->InitializeNodes(femObject);
  this->InitializeElements(femObject);
  this->InitializeLoads(femObject);

  // produce DOF
  femObject->FinalizeMesh();
}
```

The material properties of the biomechanical model such as Young modulus and Poisson’s ratio are specified in InitializeMaterials. InitializeNodes and InitializeElements are used to store the nodes and the elements of the mesh into containers of the FEMObject. The displacement assocaited with the featrue points are stored as loads in the FEMObject by InitializeLoads, in which the correspondence and tensor will be stored too if they are provided by users. After initialization, FinalizeMesh should be invoked to produce Degree of Freedoms (DOF) for the building of stiffness matrix K. After converting to FEMObject, FEMRobustSolver is invoked to construct linear system of equations described by equation (7), resolve U of the linear system, and output a deformed FEMObject, which is used by DeformationFieldGenerator to produce a deformation field. The following codes show its typical usage.

Typical usage:

```
const unsigned int ParametricDimension = 2;
const unsigned int DataDimension = 2;

typedef     short                                                PixelType;
typedef     double                                               RealType;
typedef itk::Image<PixelType, ParametricDimension>               ImageType;
typedef itk::Vector<RealType, DataDimension>                     VectorType;
typedef itk::Matrix<RealType, DataDimension, DataDimension>      MatrixType;
typedef itk::Image<VectorType, ParametricDimension>              DeformationFieldType;
typedef itk::PointSet <VectorType, ParametricDimension>          PointSetType;

typedef itk::PointSet <MatrixType, ParametricDimension>          TensorPointSetType;
typedef itk::PointSet <RealType, ParametricDimension>            ConfidencePointSetType;
typedef itk::Mesh< VectorType, ParametricDimension>              MeshType;

typedef itk::FEMScatteredDataPointSetToImageFilter
<PointSetType, MeshType, DeformationFieldType,
ConfidencePointSetType, TensorPointSetType>                      FilterType;

FilterType::Pointer filter = FilterType::New();
PointSetType::Pointer featurePoints = PointSetType::New(); // feature points associated with displacement
MeshType::Pointer mesh = MeshType::New(); // 2D triangle/rectilinear or 3D tetrahedral/hexahedral mesh
ConfidencePointSetType::Pointer confidence = ConfidencePointSetType::New();
TensorPointSetType::Pointer tensor = TensorPointSetType::New();
```

```
filter->SetInput(featurePoints);
filter->SetConfidencePointSet(confidence); //optional
filter->SetTensorPointSet(tensor); //optional
filter->SetMesh(mesh); // optional
filter->Updata();

DeformationFieldType::Pointer field = filter->GetOutput();
```

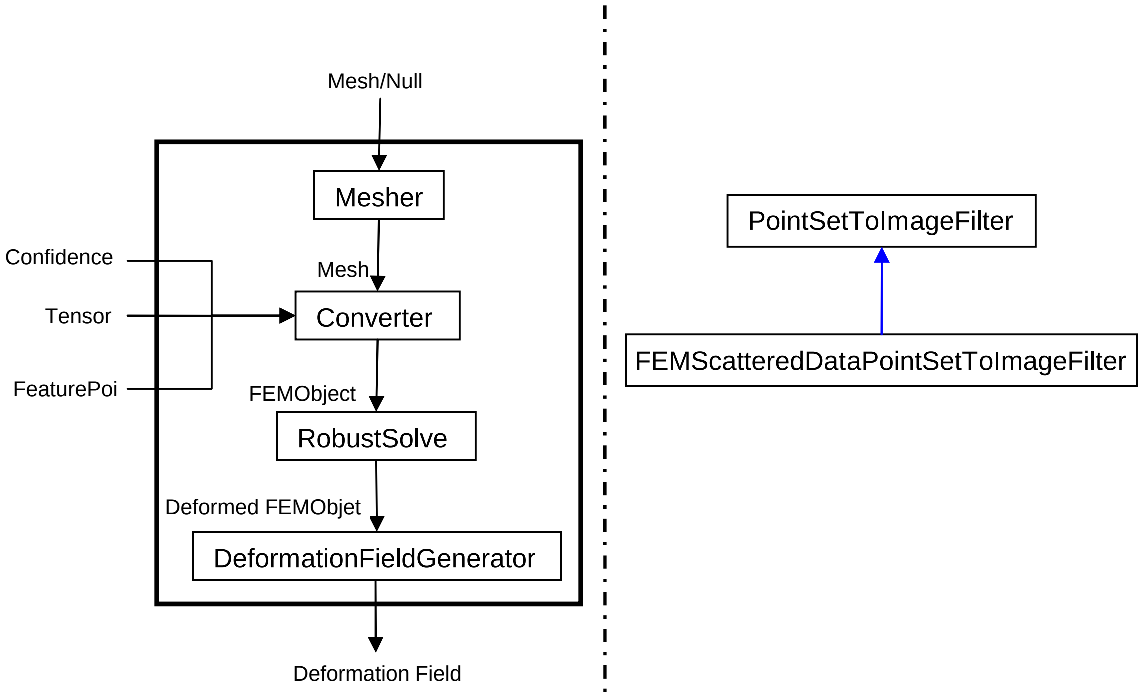

Figure 1. The flow chart (left) and the inheritance diagram (right) of FEMScatteredDataPointSetToImageFilter. FEMScatteredDataPointSetToImageFilter takes mesh, feature points, confidence and structural tensor as inputs. Converter first converts these input into a FEMObject, and then invokes RobustSolver to produce a deformed FEMObject. This deformed Object is converted into the deformation filed by DeformationFildGenerator.

3.2 RobustSolver

Given a 2- or 3-D scattered and noisy point set, in which each point is associated with a 2-D or 3-D displacement, RobustSolver is able to approximate the data while rejecting outliers, advance toward interpolation, and finally output a deformed FEMObject. The flow chart and inheritance diagram are described in Figure 2 and Figure 3, respectively.
RobustSolver also takes into account two optional point sets: the confidence and structural tensor. Confidence point set describes our confidence for each feature point using a value between 0 and 1 (0: not trustful, 1: completely trustful), which will make the solver behavior like a weighted Least Square. Tensor point set describes the distribution of the edge direction within a small block surrounding the feature point, which is used to avoid the aperture problem [4, 5]. The following codes show the typical usage of the RobustSolver.

Typical usage:

```
typedef itk::fem::FEMObject<2>     FEMObjectType;
FEMObjectType::Pointer underformedFEMObject = FEMObjectType::New();

// initialize underformedFEMObject  here or use FEMScatteredDataPointSetToImageFilter, which will undertake the initialization.
```

```
typedef itk::fem::RobustSolver<2>    FEMSolverType;
FEMSolverType::Pointer solver = FEMSolverType::New();

solver->SetInput(underformedFEMObject);
solver->Update( );
FEMObjectType::Pointer deformedFEMObject = solver->GetOutput( );
```

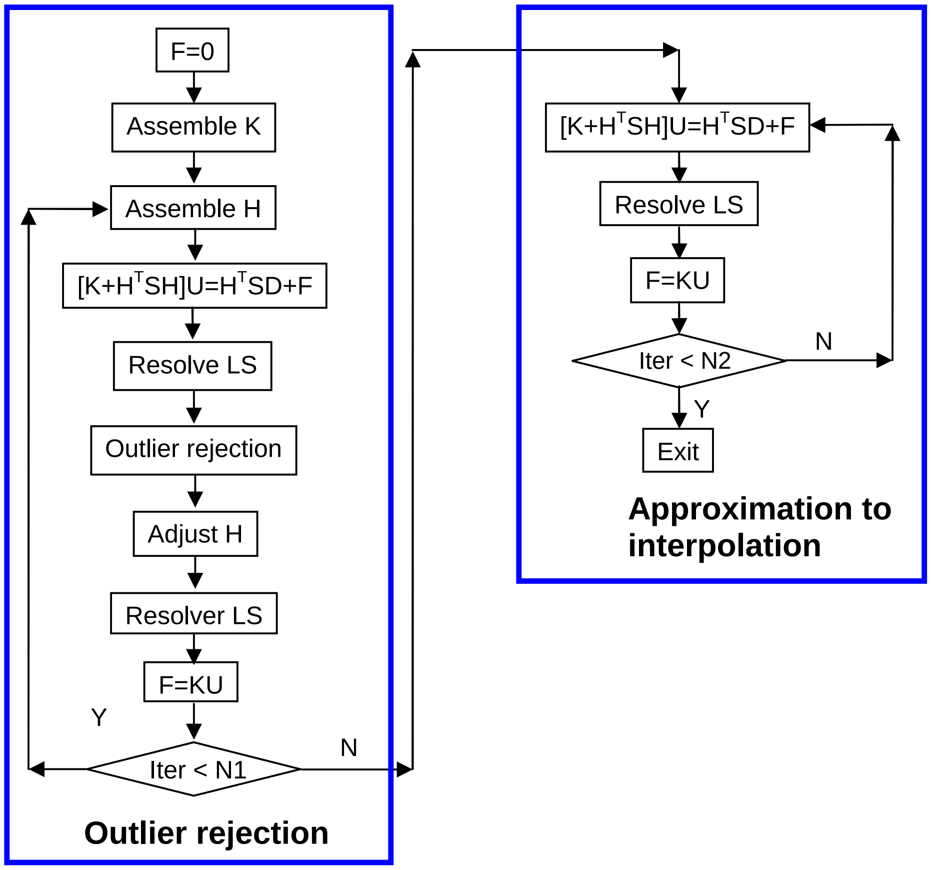

Figure 2. The flow chart of RobustSolver. RobustSolver includes two parts: outlier rejection and approximation to interpolation. Outlier rejection proceeds as a LTS regression: resolve U first, then detect outliers based on error function (10), remove outliers and resolve U again. The F is used to reset the strain energy to enable the mesh to be deformed further. The difference between the two parts is there is no outlier rejection in the approximation to interpolation part.

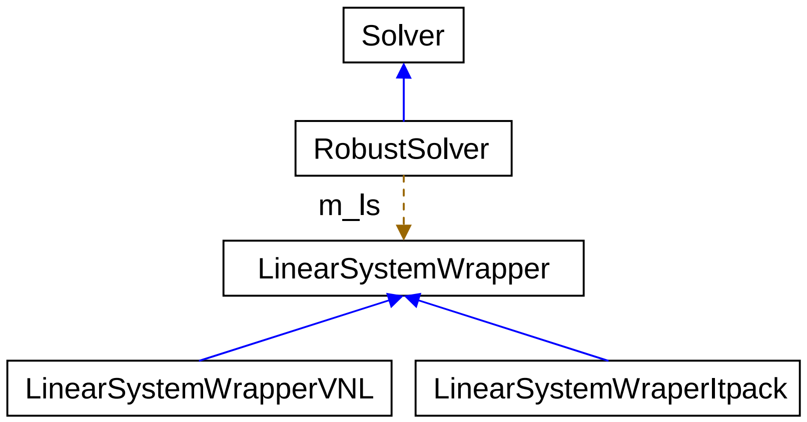

Figure 3. The inheritance diagram of RobustSolver. RobustSolver supports both VNL solver and Itpack solver to resolve the linear system of equations. Compared to VNL solver, Itpacks runs faster, which is the default LS solver in RobustSolver.

4. Experiments

We performed experiments on the synthetic data to evaluate two features of FEMScatteredDataPointSetToImageFilter : approximation to interpolation and robustness, and performed experiments on the real MRI image to show its typical application on interpolating deformation filed and registration.

4.1 Approximation to interpolation

A 7×7 grid is produced to simulate an image. 14 landmarks (red points in Fig. 4a) are produced. One landmark is associated with a unit positive displacement and the others are fixed. The mesh is a 3 × 3 (spacing: 2 × 2) quadrilateral, which is not shown in the figure. After one approximation (Fig. 4b), there is a large approximation error associated with the left corner landmark. After two (Fig. 4c) and three approximation steps (Fig. 4d), the approximation error is reduced, which demonstrates that the solver is able to gradually advance from approximation to interpolation. The typical codes are:

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

filter->GetFEMSolver()->SetApproximationSteps(3);

// no outlier rejection
filter->GetFEMSolver()->SetOutlierRejectionSteps(0);
```

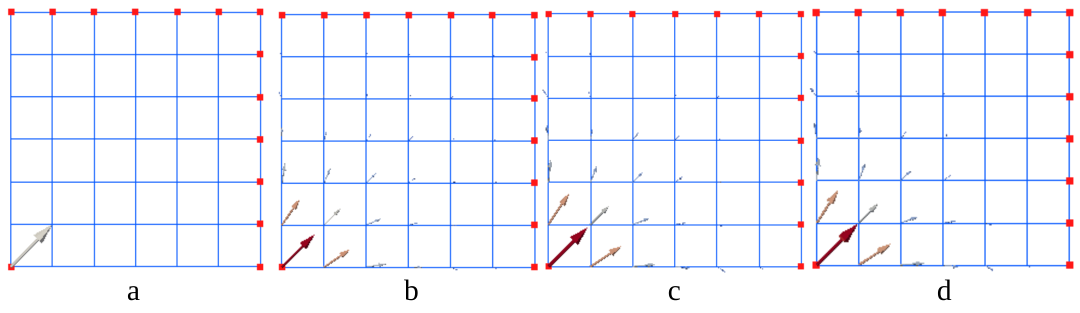

Figure 4. Approximation to interpolation. a shows the 7 × 7 grid and 14 landmarks. b, c and d show the deformation fields after one, two and three approxiamtion steps, respecively.

4.2 Robustness (outlier rejection)

To evaluate the robustness of the solver, two landmarks (red color points in Fig. 5a) are selected from the two fixed boundaries. A negative unit displacement is assigned to these two outliers. Without outlier rejection, the resulting deformation field is shown as Fig. 5b, where many pixels have a wrong negative displacement induced by the outliers. With outlier rejection (Fig. 5c), the resulting deformation is quite same with the ground truth (Fig. 4d), which demonstrates that the solver is able to remove outliers. The typical codes are:

```
// setting for outlier rejection. Set OutlierRejectionSteps to 0 to disable outlier
// rejectioin.
filter->GetFEMSolver()->SetOutlierRejectionSteps(1);
filter->GetFEMSolver()->SetFractionErrorRejected(0.2);
```

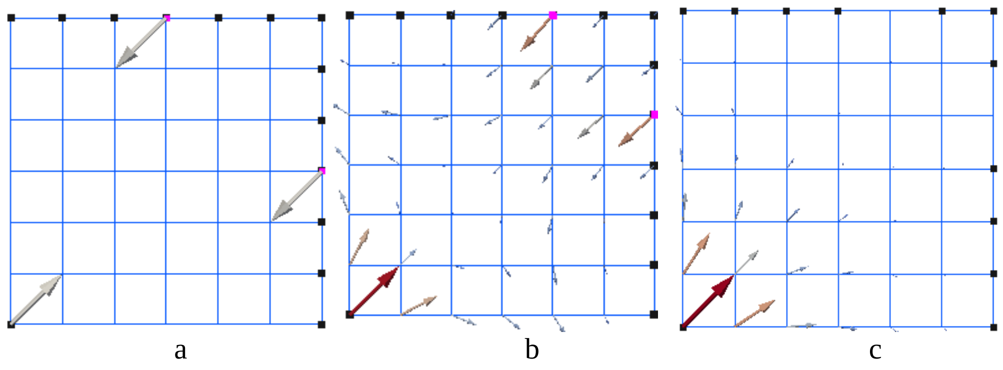

Figure 5. Outlier rejection. a shows two outliers. b is the deformation field without outlier rejection, and c is the deformation field with outlier rejection.

4.3 Application

A typical application of FEMScatteredDataPointSetToImageFilter is to estimate the entire deformation field based on sparse deformation field. The approximated deformation filed can be further used with itk::WarperFilter to produce an aligned image. To produce a sparse deformation field, first, we perform deformable registration on the lung images of rat (see Figure 6) using itk::BSplineDeformableTransform. The resulting deformation field is shown in the left image of Figure 7. Then, we perform edge detection in the fixed image (left image of Figure 6) to produce the edge image (right image of Figure 6). At last, for all edge points perform interpolation in the deformation filed to produce a sparse deformation field, which is represent by itk::PointSet. Since the edge detection is performed on the fixed image, which has the same origin, spacing and size with the deformation field. The displacement associated with the edge point can be directly obtained. The following codes demonstrate how to produce the sparse deformation field from an edge image and a deformation field. Note that we focus on the assessment of the FEMScatteredDataPointSetToImageFilter in estimating the entire deformation field from a sparse deformation field rather than on how to produce the input sparse deformation field. Users can use the tools they have to produce the sparse deformation field, no necessary following the procedures presented in this paper.

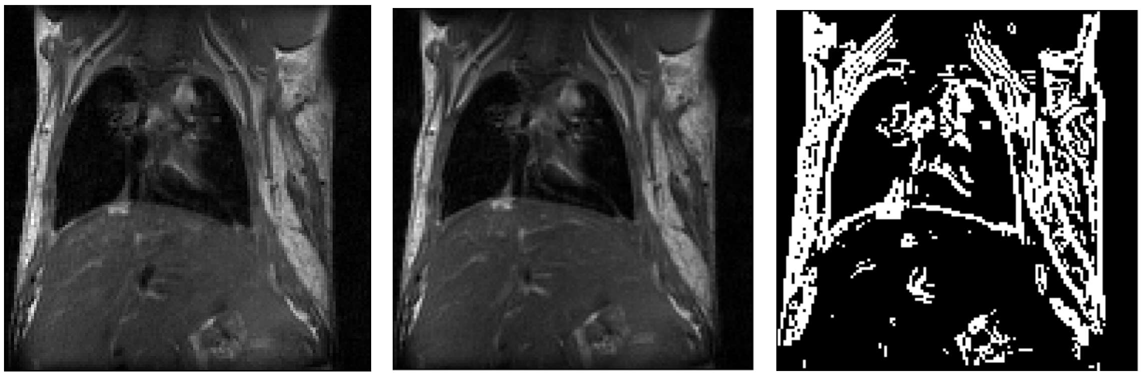
Figure 6: Fixed image, moving image and edges detected in the fixed image.

```
const unsigned int ParametricDimension = 2;
const unsigned int DataDimension = 2;

typedef       unsigned char                                             PixelType;
typedef       double                                                    RealType;
typedef itk::Vector<RealType, DataDimension>                            VectorType;
typedef itk::Matrix<RealType, DataDimension, DataDimension>             MatrixType;

typedef itk::Image<PixelType, ParametricDimension>                      ImageType;
typedef itk::Image<VectorType, ParametricDimension>                     DeformationFieldType;

typedef itk::ImageFileReader<ImageType>                    FeatureReaderType;
typedef itk::ImageFileReader <DeformationFieldType>        DeformationFieldReaderType;
typedef itk::ImageFileWriter <DeformationFieldType>        DeformationFieldWriterType

typedef itk::PointSet<VectorType, ParametricDimension >    FeaturePointSetType ;

ImageType::SizeType size;
ImageType::SpacingType spacing;
ImageType::RegionType region;
ImageType::IndexType index;
ImageType::PointType origin;
ImageType::DirectionType direction;

// load edge image
FeatureReaderType::Pointer featureReader = FeatureReaderType::New();
featureReader->SetFileName(edgeImageFilename);
featureReader->Update();
ImageType::Pointer featureImage = edgeReader ->GetOutput();

region = featureImage ->GetLargestPossibleRegion();

// load deformation field
DeformationFieldReaderType::Pointer deformationFieldReader = DeformationFieldReaderType::New();
deformationFieldReader->SetFileName(inputDeformationFieldFilename);
deformationFieldReader->Update();
DeformationFieldType::Pointer inputField = deformationFieldReader->GetOutput();

// interpolate deformation field for all edge points
itk::ImageRegionIteratorWithIndex <ImageType>   itFeaturePoint (featureImage, region);
itk::ImageRegionIteratorWithIndex <DeformationFieldType>   itField (inputField, region);

FeaturePointSetType::Pointer featurePoints = FeaturePointSetType::New();

itFeaturePoint.GoToBegin();
itField.GoToBegin();

while( !itFeaturePoint.IsAtEnd() )
  {
  if(itFeaturePoint.Get() != itk:: NumericTraits <PixelType >::Zero)
    {
```

```
  FeaturePointSetType::PointType point;
  featureImage->TransformIndexToPhysicalPoint(itFeaturePoint.GetIndex(), point);
  unsigned long i = featurePoints -> GetNumberOfPoints ();
  featurePoints->SetPoint(i, point);

  FeaturePointSetType::PixelType displacement(DataDimension);
  displacement = itField.Get();
  featurePoints->SetPointData(i, displacement);

  }

  ++itFeaturePoint;
  ++itField;
 }
```

Based on the sparse deformation field, FEMScatteredDataPointSetToImageFilter is able to interpolate the entire deformation field, represented by an itk::Image. This filter takes a feature point set, mesh, confidence point set and tensor point set as inputs, and outputs a deformation field. The feature point set defines the sparse deformation filed. The mesh can be 2D triangular/quadrilateral or 3D tetrahedral/hexahedral mesh. For convenience, default 2D/3D rectilinear mesh, confidence and tensor point sets are provided.
Figure 7 shows the comparison between original deformation field and the estimated deformation field. The estimated field has the same range of the magnitude with the original field as shown in the scalar bars. Usually, the more edge points we have, the better the estimated deformation field is. In the data related to this paper, except the binary edge image and the input deformation field, we also provide a gray edge image. The binary edge image used in this paper is produced by setting the threshold to [100, 255]. This gray edge image allows users to play with different thresholds using Slicer4 to produce different binary edge images in order to observe the influence of the number of the edge points on the estimated deformation field.
The following codes show how to use FEMScatteredDataPointSetToImageFilter to interpolate the entire deformation field from the sparse deformation field.

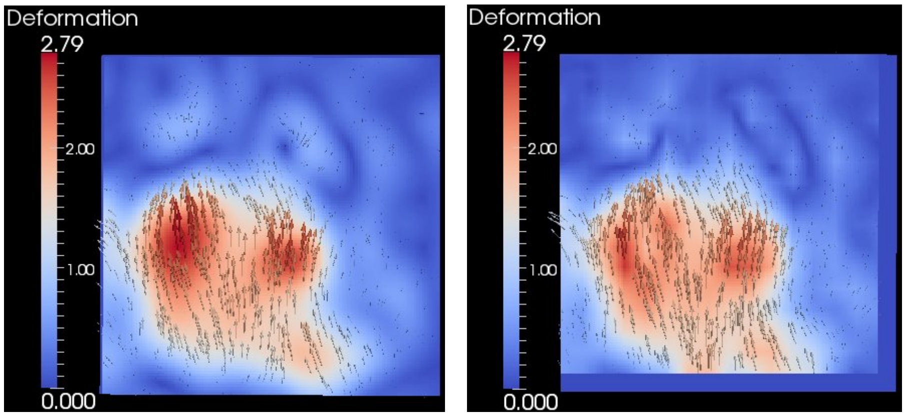

Figure 7: Original deformation field and the approximated deformation field. The estimated deformation field shows the same range of the magnitude of the deformation and quite similar deformation distribution. The figures are produced by ParaView 3.12 with the following steps: 1) load deformation field, 2) input “MetaImage_X*iHat+MetaImage_Y*jHat” in the calculator filter, and 3) visualize Glyphs for 1000 points.

```
typedef itk::Mesh< VectorType, ParametricDimension >          MeshType;
typedef itk::PointSet<RealType, ParametricDimension>          ConfidencePointSetType;
typedef itk::PointSet<MatrixType, ParametricDimension>          TensorPointSetType;

typedef itk::fem::FEMScatteredDataPointSetToImageFilter
<FeaturePointSetType, MeshType, DeformationFieldType, ConfidencePointSetType,
TensorPointSetType>                         ScatteredDataPointSetToImageFilterType;

ScatteredDataPointSetToImageFilterType::Pointer scatteredPointSetToImage = ScatteredDataPointSetToImageFilterType::New();
scatteredPointSetToImage->SetInput(featurePoints);

// set the parameters for a rectilinear mesh. Ingore this setting if users provide a mesh
DeformationFieldType::SpacingType elementSpacing;
elementSpacing[0] = 8.0;
elementSpacing[1] = 8.0;
scatteredPointSetToImage->SetElementSpacing(elementSpacing);

// set the confidence of the feature points.
// not necessary, if all feature points are trustful
//ConfidencePointSetType::Pointer confidence = ConfidencePointSetType::New();

// set the tensor of the feature points.
// not necessary due to no tensor assocaited with the feature point.
// If the feature points come from itkMaskFeaturePointSelectionFilter,
// the tensor produced by itkMaskFeaturePointSelectionFilter need to be set here
//TensorPointSetType::Pointer tensor = TensorPointSetType::New();

size = region.GetSize();
origin = featureImage->GetOrigin();
spacing = featureImage->GetSpacing();

// set output deformation field, which has the same setting with the feature image,
// as well as the fixed image, since the edge detection is performed on the fixed image.
scatteredPointSetToImage->SetSize(size);
scatteredPointSetToImage->SetSpacing(spacing);
scatteredPointSetToImage->SetOrigin(origin);

// set parameters for FEM solver. Usually, the default setting works well.
//scatteredPointSetToImage->GetFEMSolver()->SetTradeOffImageMeshEnergy(1.0);
//scatteredPointSetToImage->GetFEMSolver()->SetApproximationSteps(10);
//scatteredPointSetToImage->GetFEMSolver()->SetOutlierRejectionSteps(5);

// no outlier rejection. We trust all edge points
scatteredPointSetToImage->GetFEMSolver()->SetFractionErrorRejected(0.0);

scatteredPointSetToImage->Update();

DeformationFieldType::Pointer outputDeformationField = scatteredPointSetToImage->GetOutput();

DeformationFieldWriterType::Pointer deformationFieldWriter = DeformationFieldWriterType::New();
deformationFieldWriter->SetFileName("outputDeformationFieldFilename.mha");
deformationFieldWriter->SetInput(outputDeformationField);
try
  {
  deformationFieldWriter->Update();
  }
catch( itk::ExceptionObject & err )
  {
  std::cerr << "ExceptionObject caught !" << std::endl;
  std::cerr << err << std::endl;
  return EXIT_FAILURE;
  }
```

The estimated deformation field can be used with itk::WarpImageFilter to produce an aligned moving image. Figure 8 shows the Checkerboard comparison before and after registration. The corresponding codes are as follows.

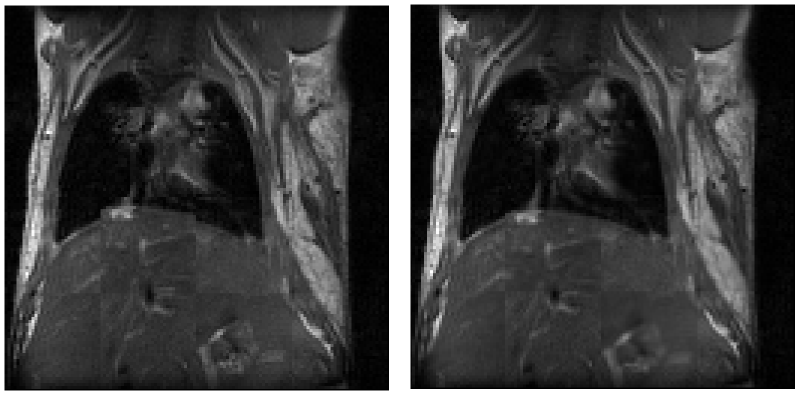
Figure 8: Checkerboard comparison before and after registration.

```
typedef itk::WarpImageFilter<ImageType, ImageType,  DeformationFieldType>     WarperType;

typedef itk::LinearInterpolateImageFunction<ImageType, RealType>              InterpolatorType;

typedef itk::ImageFileReader<ImageType>                                       ReaderType;
typedef itk::ImageFileWriter<ImageType>                                       WriterType;

typedef itk::CheckerBoardImageFilter< ImageType >                             CheckerBoardFilterType;

// load fixed image
ReaderType::Pointer fixedReader = ReaderType::New();
fixedReader->SetFileName(fixedImageFilename);
fixedReader->Update();
ImageType::Pointer fixedImage = fixedReader->GetOutput();

// load moving image
ReaderType::Pointer movingReader = ReaderType::New();
movingReader->SetFileName(movingImageFilename);
movingReader->Update();
ImageType::Pointer movingImage = movingReader->GetOutput();

// warp image
WarperType::Pointer warper = WarperType::New();
InterpolatorType::Pointer interpolator = InterpolatorType::New();
warper->SetInput( movingImage );
warper->SetInterpolator( interpolator );
warper->SetOutputSpacing( spacing );
warper->SetOutputOrigin( origin );
warper->SetDeformationField(outputDeformationField);
warper->Update();

CheckerBoardFilterType::Pointer checkerBoardFilter = CheckerBoardFilterType::New();
checkerBoardFilter->SetInput1(fixedImage);
checkerBoardFilter->SetInput2(warper->GetOutput());
checkerBoardFilter->Update();

WriterType::Pointer writer = WriterType::New();
writer->SetFileName("checkerBoardAfterRegistration.mha");
writer->SetInput(checkerBoardFilter->GetOutput());
try
  {
  writer->Update();
  }
catch( itk::ExceptionObject & err )
  {
  std::cerr << "ExceptionObject caught !" << std::endl;
  std::cerr << err << std::endl;
  return 0;
  }
```

5. Conclusion

We present an ITK implementation of a robust FEM solver. This solver can iteratively reject potential outliers and approaches interpolation. The synthetic experiments demonstrate these two characteristics, and the real experiments demonstrate its typical application on estimation of the entire deformation field based on a sparse field. The estimated field can be further used with the itk::WarpImageFilter to produce an aligned image. In this paper, we produce a sparse deformation field by interpolating the original field at the edge points. In practice, the sparse deformation field can be produced by feature point detection filter: itkMaskFeaturePointSelectionFilter and block matching filter: itkBlockMatchingImageFilter. Combined with these two filters, not limited to these filters, FEMScatteredDataPointSetToImageFilter can be used to perform physics-based non-rigid registration.

6. Acknowledgements

This work is funded mainly by the ARRA funds for the ITK-v4 implementation with grant number: NLM A2D2 201000586P. In addition, this work is supported in part by NSF grants: CCF-1139864, CCF-1136538, and CSI-1136536 and by the John Simon Guggenheim Foundation and the Richard T. Cheng Endowment. We would like to thank Luis Ibanez, Matt McCormick, Arnaud Gelas and Andinet Enquobahrie for their help on ITK coding style. In addition, Andriy Fedorov and Andriy Kot, they were instrumental in running the old BioFem code which helped us to better understand the method in [1]. Of course, Olivier Clatz for insightful discussions on this method [1] during his week-long 2006 visit in the Center for Real Time Computing at William and Mary. At last but not least Fotis Drakopoulos for fixing several bugs while he used this filter for non-rigid registration of brain MRI images.

References

[1] O. Clatz, H. Delingette, I.-F. Talos, A. Golby, R. Kikinis, F. Jolesz, N. Ayache, and S. Warfield, "Robust non-rigid registration to capture brain shift from intra-operative MRI", IEEE Trans. Med. Imag., 24(11);1417-27, 2005.
[2] K. Bathe, “Finite Element Procedure”, Prentice-Hall, 1996.
[3] P. J. Rousseeuw and A. M. Leroy, “Robust regression and outlier detection”, John Wiley & Sons, Inc., New York, NY, USA, 1987.
[4] S. Shimojo, G. H. Silverman, and K. Nakayama, “Occlusion and the solution to the aperture problem for motion”, Vision Research, 29, 619-626, 1989
[5] T. Poggio, V. Torre, and C. Koch, “Computational vision and regularization theory,” Nature, vol. 317, pp. 314–319, Oct. 1985.