An implementation of TDFFD and LDFFD algorithms

Release 0.00

Mathieu De Craene[1,2], Gemma Piella[2]

March 8, 2012

[1]Philips Research, Medisys, Paris.
[2]DTIC, Universitat Pompeu Fabra, Barcelona.

Abstract

This paper describes the implementation of the LDFFD and TDFFD algorithms. Both quantify motion and deformation from an input image sequence by taking profit of the temporal dimension of velocity fields in the diffeomorphic registration framework to process the input image sequence as a single 3D+t object. The rationale behind these approaches is to formulate motion estimation in an image sequence as a multi-channel registration problem. This paper describes our implementation, currently following the original ITK registration framework without multi-threading and diffeomorphic registration framework. Migration to ITKv4 will be addressed in future work. We also provide code for computing strain on a segmentation represented as a VTK polydata mesh. Program usages for LDFFD and TDFFD as well as strain quantification results are given for one 3D ultrasound image sequence acquired on a healthy volunteer.

Latest version available at the Insight Journal [http://hdl.handle.net/10380/XXXXX]

Contents

1 Introduction

Diffeomorphic registration algorithms were originally designed to solve for large invertible deformations between a pair of images. They solved this problem by evolving from a displacement-based representation of the transformation to a velocity-based representation. In this framework, the mapping between the two images is obtained by following the flow of a velocity field over time. The composition of - theoretically - infinitesimal time steps guarantees to generate invertible transforms, provided that the velocity is a continuous function.

Since diffeomorphic registration algorithms introduced a temporal component to the transformation, several researches have studied the implications of extending such framework to problems that naturally involve time. Khan *et al.* [8] extended the Large Deformation Diffeomorphic Metric Mapping (LDDMM) [1] registration framework to quantify longitudinal shape changes in a sequence of images. They used a dense representation of the velocity field and smoothed at every iteration, the velocity field by a Gaussian kernel in the spatial dimensions. Qui *et al.* [9] used time sequence diffeomorphic mappings to track anatomical shape changes in serial images for a collection of subjects. They then map the trajectories using parallel transport to a reference space to compare different subjects. Durrleman *et al.* [7] followed a similar concept for analyzing the anatomical variability of a set of longitudinal data.

In this paper, we provided the implementation and some basic examples of applications for the Large Diffeomorphic Free Form Deformation (LDFFD) and Time Diffeomorphic Free Form Deformation (TDFFD) algorithms [2, 5, 4, 6]. The key idea of the two algorithms is to represent the temporal transformation from a reference frame to all frames of an image sequence using the composition of small time steps, represented using a velocity field.

The two approaches differ in the representation of the velocity field. In the LDFFD approach, the transformation is encoded as a vector of FFD transformations. In this setup, the time dimension is only represented through the concatenation of small transformations. Invertibility of the transformation is guaranteed by imposing an upper bound [10] on the magnitude of each step. If this bound is reached during the optimization process, the time step exceeding the upper bound is split by finding the closest approximation to its square root [3].

In the TDFFD framework, the velocity is represented as a continuous function over space *and* time by using kernels in the 4D space. Our implementation uses a fixed time step and checks the positivity of Jacobians over each trajectory to ensure that the transformation is indeed invertible. The temporal smoothness in time of the velocity field guarantees to recover temporally smooth transformation. Another advantage is that the trajectories can be evaluated for any continuous time and are not restricted to the discrete set of imaging time points.

In both approaches, some *coupling* appears in the expression of the metric derivatives. By coupling, we mean that the optimization of the velocity field at one time step evaluates image intensities not only at the

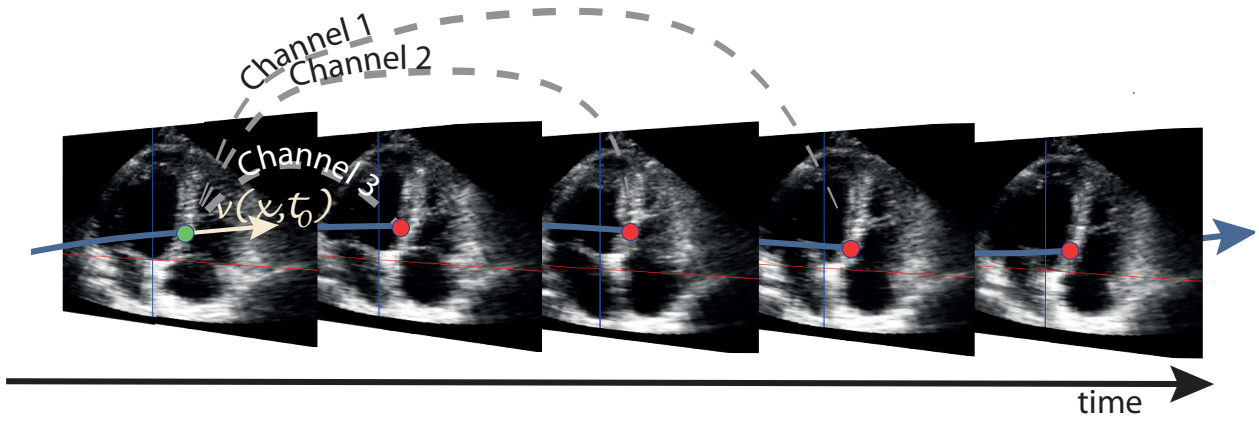

Figure 1: Coupling: the optimization of velocity at one time step involves not only the images adjacent to this time step but potentially all pairs of images in the sequence. Each pair of images can be seen as a channel for optimizing velocity.

adjacent frames but through all the sequence. An analogy to this is the estimation of the slope of a linear function from noisy input data. If two adjacent point are considered for estimating the slope, this estimate will amplify the noise and give an inaccurate slope value. If instead, all points of the dataset are used, the noise can be compensated in a more robust way. This is illustrated in Fig. 1. A variation of velocity at time t_0 has influence on the transported coordinate at any frame of the sequence. The comparison of intensities between the closest frame to t_0 and any frame in the sequence can be seen as one observation channel for optimizing the velocity field. This inter-dependance is expected to increase the robustness of motion quantification algorithms. This was shown for synthetic 3D ultrasound sequences in [4].

2 Implementation

2.1 TDFFD

In the TDFFD approach, time is represented as the last coordinate in the pixel stack of the input image. Since the velocity field is continuous in time in this approach, both velocity and input images are expressed in a 4D space for 3D image sequences. The input sequence is represented as a single object rather than the concatenation of spatial images.

We reimplemented metrics and transform objects, copying the existing ITK registration framework and extending it to handle temporal image sequences as single objects. It is currently detached from the existing ITK framework and does not take advantage of all sampling and multi-threading strategies implemented in ITKv4. Fig. 2 gives the main classes, instances and inheritances that are used in our implementation. Taking the `BSplineDeformableTransform` as starting point, we created a class called `BSplineField` that interpolates control point velocities to compute the velocity value at any spatiotemporal location. We could not use `BSplineDeformableTransform` as it is because it does not allow for different input and output dimensions (for a 3D image sequence, velocities are three dimensional vectors in a four dimensional space). The basic functionalities of a temporal diffeomorphic transformation were then implemented in the `TimeDiffeomorphicTransform` class. This leaves the possibility to work with other representations of the

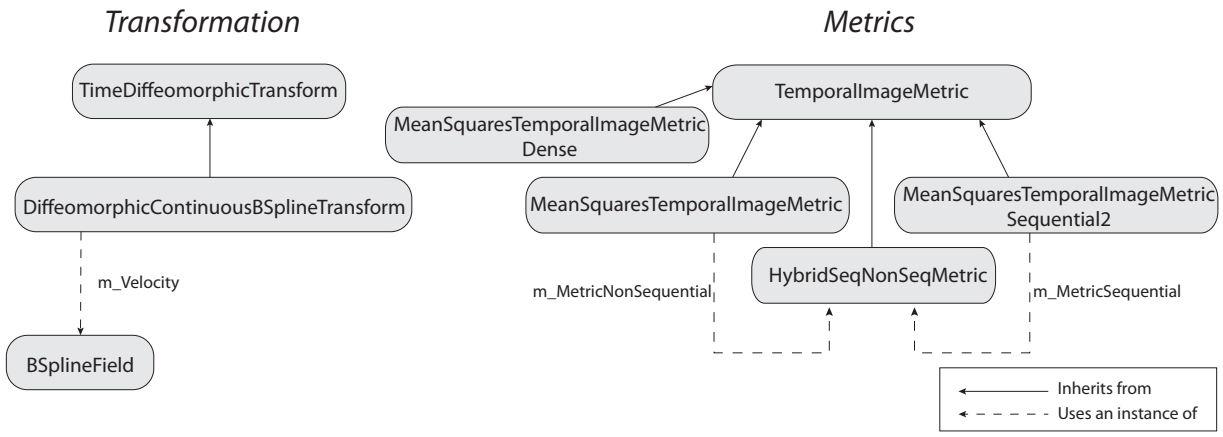

Figure 2: Diagram of classes for the TDFFD registration framework.

velocity field or with another composition strategy for generating diffeomorphic mappings.

Regarding the image similarity metric, we implemented mean squared errors with fixed or floating reference. A fixed reference frame, as implemented in the `MeanSquaresTemporalImageMetric` class, has the advantage of being more robust to drift since all trajectories are integrated over time and intensities are compared to the reference frame. Although this accumulation avoids that the composition of small errors generates important drift, it is expected to be less accurate to detect small frame-to-frame deformations than quantifying similarity in a sequential way [11], as implemented in the `MeanSquaresTemporalImageMetricSequential` class. To get the best of these two choices, we built an hybrid metric summing both fixed reference and sequential terms. This "hybrid" metric was implemented in the `HybridSeqNonSeqMetric`. The weight between the two terms is adjusted by default so that the two terms have an equal contribution to the metric for the initial set of parameters. Alternative ways of combining sequential and fixed reference similarity terms for the TDFFD framework have been proposed by Zhang *et al.*[12].

Tables 1 and 2 give a list of the main functions and members implemented in each class.

2.2 LDFFD

In the case of LDFFD, an image sequence is represented as a vector of 3D images. Unlike TDFFD, there is no temporal continuity embedded in this approach. Hence, each 3D image is associate to a time index. At initialization, all images in the sequence are incrementally indexed from 0 to $N-1$, where N is the number of images in the sequence. Each time index is associated to the corresponding index in a vector of 3D transformations. When one of these transformations exceeds in magnitude a threshold that ensures invertibility, the corresponding transformation is split in two and all subsequent time indexes in the image sequence are incremented.

The diffeomorphic transformation object is called `DiffeomorphicBSplineTransform`. It contains a vector of parameters as any ITK transform. Internally, this vector of parameters is the concatenation of internal transforms parameters. Every internal transformation is a BSpline deformable transformation, as implemented by the `BSplineDeformableTransformOpt`. The only additional method in `BSplineDeformableTransformOpt` with respect to the ITK class, is `GetClassicalJacobian` that com-

Table 1: Main methods of the TDFFD implementation.

BSplineField	
SetGridRegion	Specify the B-Spline region defining the 3D+t (or 2D+t) velocity field. Regions are specified in the standard ITK way and contain an index and a size. The region type is defined in RegionType.
SetGridSpacing	Specify the B-Spline grid spacing between control points. Expected type defined in SpacingType.
SetGridOrigin	Specify the B-Spline grid origin. Expected type defined in OriginType.
GetPointVelocity	Compute the velocity field at a given spatiotemporal location. Takes as input a 3D+t or 2D+t point. Besides returning the velocity, this function also outputs the interpolation weights, the affected indexes in the coefficient images and a flag specifying if the points were inside the region. These concepts are implemented in a similar way as in the BSplineDeformableTransform class.
GetClassicalJacobian	Computes the spatial derivative of the velocity field. This is related to strain rate when quantifying cardiac deformation.
GetVelocityJacobian	Computes the derivative of the output velocity field with respect to the parameters. In our case, this function returns the interpolation weights as the output velocity is a linear combination of these and the parameters.
TimeDiffeomorphicTransform	
SetParameters	Set parameters (passing a reference).
SetParametersByValue	Set parameters (by value).
GetJacobian	Get the derivative of the transformed point coordinates with respect to transformation parameters in the standard ITK way.
GetSparseJacobian	Returns the same derivative as GetJacobian but using a sparse representation for the Jacobian. We use as SparseJacobianType an std::map containing vnl_vector. Each vnl_vector represents one column of the full Jacobian matrix.
GetIncrementalSparseJacobian	In the case of a temporal diffeomorphic transformation, there is an accumulation mechanism that propagates the derivative at further time points. Therefore, one can use computations performed at earlier time steps when moving along the sequence in time. This function does not "reset" the Jacobian for avoiding to redo the same computations.
DiffeomorphicContinuousBSplineTransform	
Set/GetTimeStep	Set/Get the time step used for following the flow of the velocity field over time.
Set/GetVelocity	Set/Get the velocity field (implemented in the BSplineField object).

Table 2: Main methods of the TDFFD implementation (continued).

TemporalImageMetric	
Set/GetTransform	Set/Get the transform (of type TimeDiffeomorphicTransform).
Set/GetTransform	Set/Get the input image sequence.
Set/GetInterpolator	Set/Get the interpolator. It will be used for computing image derivative if it is B-Spline based (similarly to Mattes Metric).
Set/GetImageRegion	Set/Get the fixed image region (including the temporal dimension).
Set/GetSpaceImageMask	Set/Get the spatial mask for randomly taking samples. The dimension of the mask is the one of the input image minus 1.
Set/GetSpaceMovingImageMask	Set/Get the moving mask. Samples falling outside of this mask will be ignored.
SetTransformParameters	Pass parameters to the transform.
Initialize	Check all components and sample image domain.
SampleReferenceImageDomain	Sample the reference domain. In cases of image metrics with a fixed reference, samples will be taken at the corresponding time index only. In case of reference free metrics, samples will be taken in the entire sequence.
Set/GetNumberOfSamples	Set/Get the number of samples.
MeanSquaresTemporalImageMetric	
GetValueAndDerivative	Compute the metric value and derivative. This method must be reimplemented for each metric variant.
HybridSeqNonSeqMetric	
Set/GetMetricNonSequential	Set Non sequential term of the metric.
Set/GetMetricSequential	Set sequential term of the metric.
Set/GetWeight	Set weight between sequential and non sequential terms.
ComputeWeight	Compute weight to give the same contribution to sequential and non sequential terms at the initialization.

putes at a given spatial location the linear approximation of the transformation. This volume change propagates the Jacobian of one transformation to further time steps in a similar way as illustrated in Figure 1, as detailed in [4].

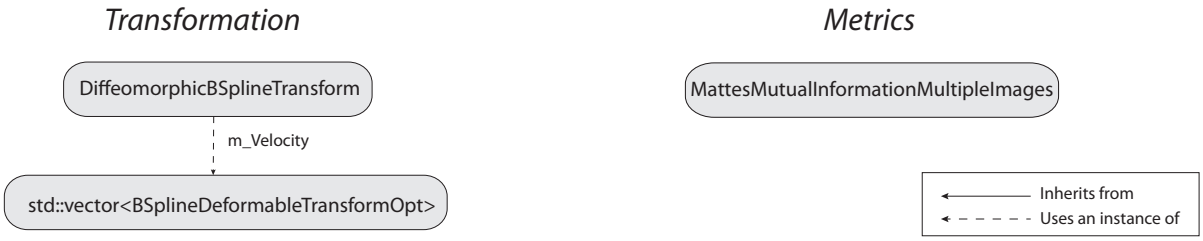

Figure 3: Diagram of classes for the LDFFD registration framework.

3 Experiments

3.1 Launching TDFFD

When launching the tdffd command with no arguments, the following help is provided

```
Problem encountered while parsing arguments.
Usage : inputImage outputImageFilePrefix outputTransformFile
Optional arguments :
 -inputTransformation transformFile
 -splitTransformation
 -timeMultiRes
 -regionSize SizeX SizeY SizeZ SizeT
 -inputMask MaskFileName
 -inputMovingMask MaskMovingFileName
 -imageRegion indexX indexY indexZ indexT sizeX sizeY sizeZ sizeT
 -MetricSequential
 -MetricDense
 -weightSequential weight
 -referenceTime refTime
 -numSamples numberOfSamples
 -maxv maxVelocityValueInsideMask
 -maxvx maxVelocityValueInsideMaskX
 -maxvy maxVelocityValueInsideMaskY
 -maxvz maxVelocityValueInsideMaskZ
 -timeStep timeStep
 -minTimeStep minTimeStep
 -padding numberOfVoxels
 -paddingx numberOfVoxelsInX
 -paddingy numberOfVoxelsInY
 -paddingz numberOfVoxelsInZ
```

Table 3: Main methods of the LDFFD implementation.

BSplineDeformableTransformOpt	
GetClassicalJacobian	Computes the spatial derivative of one BSpline transformation in the chain using the BSplineInterpolationWeightFunctionDerivative methods.
DiffeomorphicBSplineTransform	
SetNumberOfTimeSteps	Set the number of transforms in the chain.
Set/GetGridSpacing	Set/Get the BSpline grid spacing of all transformations.
Set/GetGridSpacing	Set/Get the BSpline grid origin of all transformations.
Set/GetGridRegion	Set/Get the BSpline region of all transformations.
Set/GetGridRegion	Set/Get the BSpline region of all transformations.
GetJacobian	Compute the derivatives of the transformed spatial coordinates with respect to the transformation parameters. This method takes as parameters the input point, and a flag accumulate indicating if the physical jacobians at anterior times have to be recomputed or not. The last parameter, passed by reference, stores the indexes of the non-null columns in the Jacobian matrices.
AccumulatePhysical JacobiansInTimeInterval	Accumulates physical Jacobians between two time steps (passed as parameters). This accumulation multiplies physical jacobian downstream in time as described in [2] (Equation 4).
MattesMutualInformationMultipleImages	
SetNumberOfMovingImages	Set the number of moving images in the sequence.
SetMovingImage	Insert a moving image at position pos in the vector. The time stamp of the moving images is the last parameters (time).

```
-dispFileName outputDispFilePrefix
```

The first three options are useful for multi resolution or to perform a second run of the algorithm from a previous transformation result. The first option (`inputTransformation`) permits to specify a transformation as input. The second (`splitTransformation`) refines the resolution of the input transformation by a factor 2. Note that by default, this refinement is performed in all spatial dimensions. In case the `-timeMultiRes` option is specified, the refinement is performed in the temporal dimension as well.

The `regionSize` option permits to specify the number of control points in each dimension. For 4D images, it should be followed by 4 numbers indicating the number of control points in each dimension.

The three next options are related to specifying image regions for computing the metric. Argument `inputMask` permits to specify a fixed image mask specifying the domain on which the similarity metric is computed. Argument `inputMovingMask` specifies a moving image mask, i.e. all samples falling outside of the mask after applying the transformation will be ignored in the metric computation. Finally, `imageRegion` is equivalent to `inputMask` but specified as an image region.

The next set of parameters is related to the metric. The most important parameter is `numberOfSamples` that sets the number of samples for the metric computation. By default, the algorithm uses a fixed reference metric. The reference frame will be the first frame by default, but can be adjusted using `referenceTime`. With the option `MetricSequential`, the algorithm optimizes a sequential metric without any reference. This generates a lot of drift effect in our experience. The combined metric weighting fixed reference and sequential metric is obtained by using the `MetricDense` option. By default, the two terms are weighted to have an equal contribution at the initialization, but the weight can be modified using `weightSequential`.

The last set of parameters can be used to parametrize the transformation. Argument `maxv` bounds the amplitude of velocity in all dimensions while `maxvx|y|z` does the same in a specific dimension. The time step for forward eulerian integration can be adjusted using `timeStep` and `minTimeStep`. We use the padding option when the image domain is very adjusted on the object of interest. By padding, we mean that the bounds of the velocity control points go beyond the image bounds by a certain amount of voxels. One can specify this amount for all spatial dimensions using `padding` or in a specific spatial dimension using `padding x|y|z`.

Finally, the `dispFileName` option permits to save the output displacement fields towards the first frame as vector images. This can be useful for visualizing the displacement field using glyphs, for example using Paraview.

3.2 Launching LDFFD

The LDFFD program takes a single argument indicating the name of a configuration file. If no argument is given, the program returns the following error:

```
could not open file: --help
Error while reading configuration file
```

A minimalist configuration file contains the following lines

```
-fixedImageFileName fixedImage.mhd
-movingImageFileName movingImage1.mhd
-movingImageFileName movingImage2.mhd
```

```
-movingImageFileName movingImage3.mhd
...
-outputImageFileNamePrefix Prefix
```

All moving images have to be specified in the order of the sequence. The first image of the sequence (`-fixedImageFileName`) is defined as fixed image. All other images are passed with the `-movingImageFileName` argument. By default, each image receives as time index equal to its position in the sequence. In the case one time step is split, as described in [3], the time indexes of all subsequent images will be shifted.

3.3 Strain quantification

The strain is estimated from the spatial derivative of the resulting diffeomorphic transformation. We provide the functions `tdffd_strain` and `ldffd_strain` to compute the strain.

When launching the `tdffd_strain` command with no arguments, the following help is provided

```
Usage: tddfd_strain inputPolyData.vtk inputTransform.dof numberOfTimeSteps
outputMeshPrefix [-la 0 0 1]
```

The fourth first arguments are required and the last one takes default value `0 0 1`. The argument `inputPolyData` corresponds to the mesh in the space of coordinates of the first time point. The second argument `inputTransform` is the output transform obtained from launching `tdffd` program. The argument `numberOfTimeSteps` is the number of time points on which the strain is going to be computed and `outputMeshPrefix` the name of the output deformed meshed. Last argument $-la$ specifies the long axis direction of the mesh and has by default value `0 0 1` (i.e., long axis lies on the z-axis). Note that for the TDFFD strain computation, we allow to pass to the `tdffd_strain` executable a volumetric mesh using the `-ug` option. In that case, the transformation is applied to every node of the volumetric mesh. The surface mesh specified as first argument will then provide the reference surface for computing the normal direction. For every, point of the volumetric mesh, the normal of the closest point on the surface mesh will be considered as radial direction.

Once strain has been computed for each node of the mesh (either surfacic, either volumetric), temporal strain curves can be reconstructed for each point or each segment, if a node array defines them for the mesh.

The computation of strain for `ldffdd_strain` used an approximation for computing the strain spatial derivatives. Instead of using the analytical derivatives of the transformation, we used the linear shape function defined on the VTK mesh. This is equivalent to approximating the transformation to a linear mapping at each time step and for each triangle of the mesh. The option of the `ldffdd_strain` executable are similar to the TDFFD one: the first argument specifies the input mesh (only surface mesh i.e. `vtkPolyData` can be specified as input in this this case), the second argument gives the transformation file name and the last argument gives the prefix for saving output meshes.

4 Processing a 3D ultrasound image sequence

In this Section, we compare the strain quantification obtained by TDFFD and LDFFD on the dataset images given in the `Data/` folder. This folder contains one 3D ultrasound image sequence. The original data is

stored in VTK format in the `Image3D/` subfolder. The data was smoothed with a Gaussian filter ($\sigma = 1.5$) and written as a single 4D image file in the `Image4D/` subfolder. We also provide fixed and moving image masks in the `Masks/` subfolder. The fixed image mask follows the myocardial boundaries while the moving image mask is set to the conical field of view of the 3D ultrasound image. A mesh of the left ventricular geometry can be found in the `Mesh/` subfolder and will be used for strain computation.

4.1 LDFFD

All parameters for the TDFFD processing are given in the `config.txt` configuration file. As output, the algorithm provides two sets of files: the output transformation and output images. The output images contain the image sequence resampled to the space of coordinates of the first frame, as shown in Fig. 4. The output transformation can then be used to propagate the mesh and compute strain.

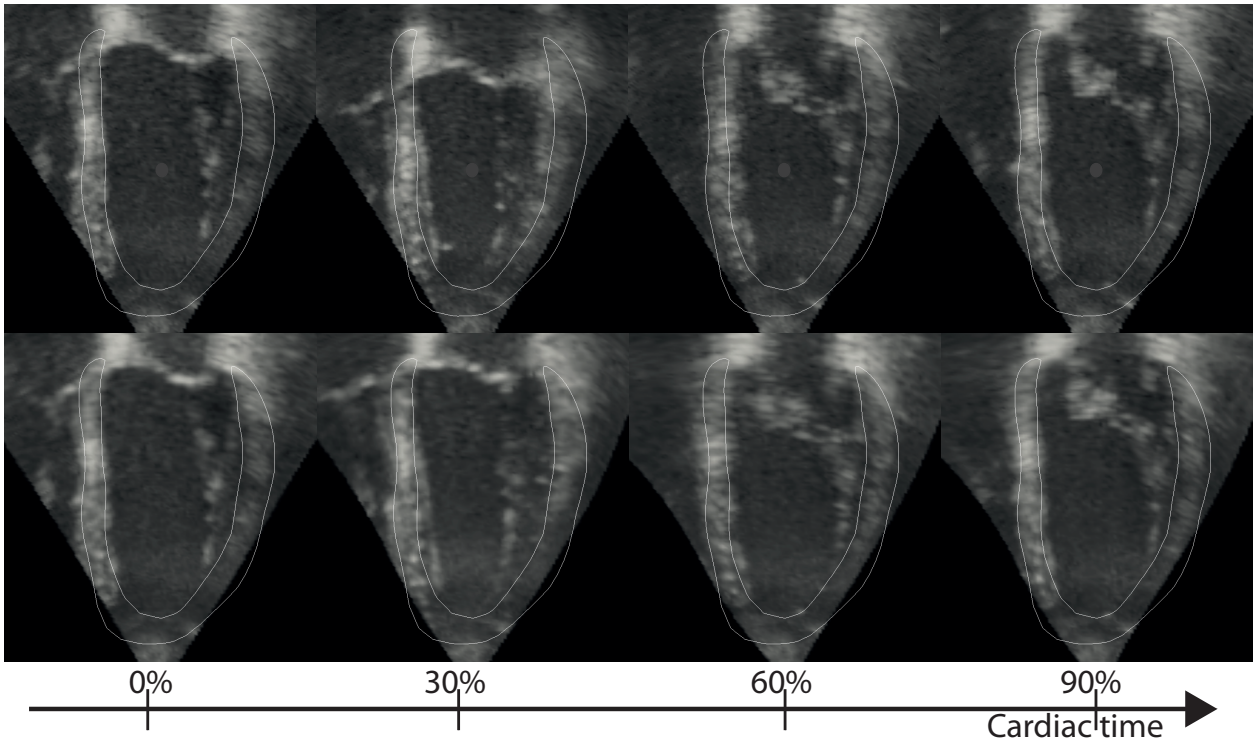

Figure 4: Resampling of one input image sequence to the reference frame using the LDFFD algorithm. The top row shows the image sequence before resampling. The bottom row show the same sequence after resampling to the first image. The contour overlaid on the image is the myocardial border segmented from the first frame.

4.2 TDFFD

The command line for launching the TDFFD code can be found in the `tdffd.pl` script. In this example, the resolution grid had a resolution of 5 control points in the short axis plane and 5 in the longitudinal direction. For the temporal dimension, we typically take a number of control points equal to the number of frames in the sequence. The result of the TDFFD algorithm is a 4D transformation that can be used

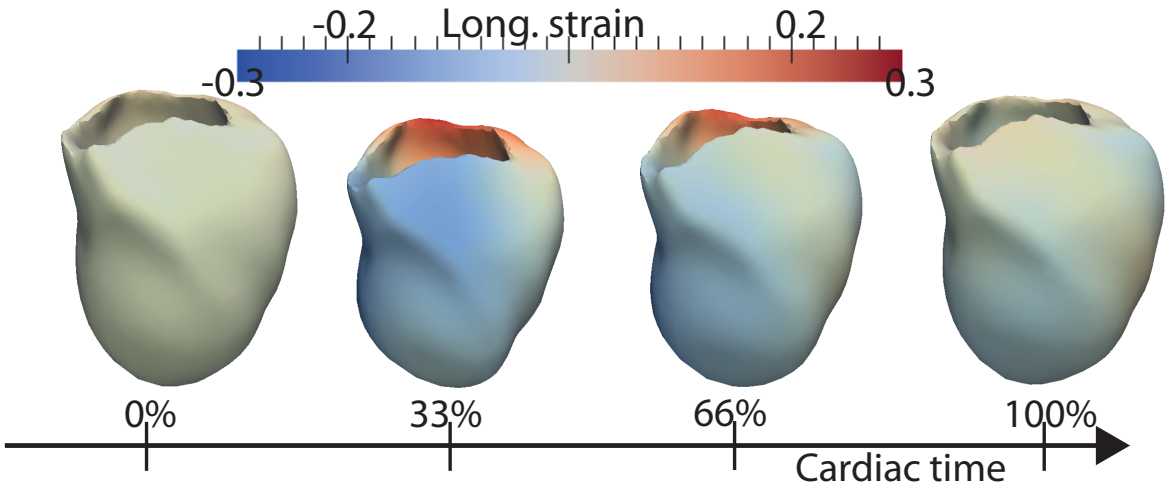

Figure 5: Visualization of longitudinal strain as a color map.

to compute trajectories between any pair of frames from the input sequence. Traditionally, since strain is computed with respect to the end of diastole, we are interested to propagate points from the first frame to any other time point in the cardiac cycle. Strain can then be obtained by computing the volume change associate to this trajectory. The resulting strain tensor can further be projected onto a local cardiac coordinate system to compute the deformation in the radial, circumferential and longitudinal directions. An example of the strain computation is given in the `tdffd_strain` script. After running this script, a collection of polydatas files is generated in VTK format. The meshes correspond to the propagation of the input surface mesh to each frame of the sequence. Additionally, each mesh contains point scalar arrays corresponding to radial, circumferential and longitudinal strains. Strain can easily be visualized using Paraview by applying a colormap, as shown in Fig. 5. The same information can also be saved to a text file and plotted as function of time per AHA segment. For this example, the AHA segmentscan further be projected onto a local cardiac coordinate system to compute the deformation in the radial, circumferential and longitudinal directions are available as another point array in the input mesh. The resulting strain curves are plotted in Fig. 6.

5 Conclusions

In this paper, we presented an implementation of LDFFD and TDFFD algorithms. Both methods attempt to extend current image registration algorithms to handle temporal sequences by integrating a non-stationary velocity field. They differ by the representation of the velocity field: sequential in LDFFD and continuous in TDFFD. We hope that these implementations will help interested researchers in the community to develop diffeomorphic temporal registration algorithms.

References

[1] M. Faisal Beg, Michael I. Miller, Alain Trouvé, and Laurent Younes. Computing Large Deformation Metric Mappings via Geodesic Flows of Diffeomorphisms. *International Journal of Computer Vision*, 61(2):139–157, February 2005. 1

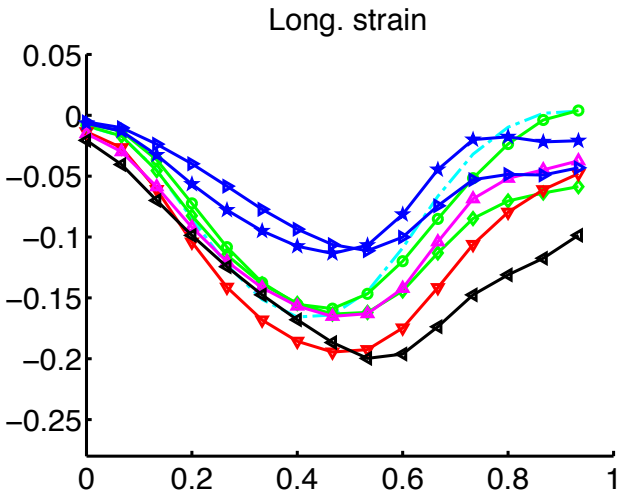

Figure 6: Visualization of longitudinal strain as temporal curve per AHA segment (each curve represents a different segment). Only segments that were totally included in the field of view of the 3D ultrasound image sequence are shown.

[2] M. De Craene, O. Camara, B.H. Bijnens, and A. F. Frangi. Diffeomorphic ffd registration for motion and strain quantification from 3d-us sequences. In *FIMH 2009 (Functional Imaging and Modeling of the Heart)*, 2009. 1, 3

[3] M. De Craene, O. Camara, B.H. Bijnens, and A. F. Frangi. Non-stationary diffeomorphic registration: application to endo-vascular treatment monitoring. volume 7259, page 72591F. SPIE, 2009. 1, 3.2

[4] M. De Craene, G. Piella, O. Camara, N. Duchateau, E. Silva, A. Doltra, J. D'hooge, J. Brugada, M. Sitges, and A.F. Frangi. Temporal diffeomorphic free-form deformation: Application to motion and strain estimation from 3d echocardiography. *Medical Image Analysis*, 2012. 1, 1, 2.2

[5] M. De Craene, G. Piella, N. Duchateau, E. Silva, A. Doltra, H. Gao, J. D'hooge, O. Camara, J Brugada, M. Sitges, and A.F. Frangi. Temporal diffeomorphic free-form deformation for strain quantification in 3d-us images. In *13th Medical Image Computing and Computer-Assisted Intervention*, Lecture Notes in Computer Science. Springer, 2010. Beijing. 1

[6] M. De Craene, C. Tobon-Gomez, C. Butakoff, N. Duchateau, G. Piella, K.S. Rhode, and A.F. Frangi. Temporal diffeomorphic free form deformation (tdffd) applied to motion and deformation quantification of tagged mri sequences. In *Proc. STACOM 2011, workshop held in Conjunction with MICCAI 2011*. Springer, 2011. Toronto. 1

[7] Stanley Durrleman, Xavier Pennec, Alain Trouvé, Guido Gerig, and Nicholas Ayache. Spatiotemporal atlas estimation for developmental delay detection in longitudinal datasets. *Medical image computing and computer-assisted intervention : MICCAI ... International Conference on Medical Image Computing and Computer-Assisted Intervention*, 12(Pt 1):297–304, January 2009. 1

[8] Ali R. Khan. Representation of time-varying shapes in the large deformation diffeomorphic framework. In *IEEE International Symposium on Biomedical Imaging*, number 1, pages 1521–1524, 2008. 1

[9] A Qiu, M Albert, L Younes, and M I Miller. Time sequence diffeomorphic metric mapping and parallel transport track time-dependent shape changes. *NeuroImage*, 45(1, Supplement 1):S51 – S60, 2009. 1

[10] D Rueckert, P Aljabar, R A Heckemann, J V Hajnal, and A Hammers. Diffeomorphic Registration using B-Splines. In *Medical Image Computing and Computer-Assisted Intervention*, volume 4191 of *LNCS*, pages 702–709. Springer, October 2006. 1

[11] Z. Zhang, D. Sahn, and X. Song. Frame to Frame Diffeomorphic Motion Analysis from Echocardiographic Sequences. In Xavier Pennec, Sarang Joshi, and Mads Nielsen, editors, *Proceedings of the Third International Workshop on Mathematical Foundations of Computational Anatomy - Geometrical and Statistical Methods for Modelling Biological Shape Variability*, pages 15–24, Toronto, Canada, September 2011. 2.1

[12] Z. Zhang, D.J. Sahn, and X. Song. Temporal diffeomorphic motion analysis from echocardiographic sequences by using intensity transitivity consistency. In *Proc. STACOM 2011, workshop held in Conjunction with MICCAI 2011*. Springer, 2011. Toronto. 2.1