Helper classes for the itk::BSplineDeformableTransform

Release 0.00

Marius Staring[1] and Stefan Klein[2]

July 9, 2008

[1]Leiden University Medical Center, The Netherlands
[2]Biomedical Imaging Group Rotterdam, Erasmus MC, Rotterdam, The Netherlands.

Abstract

This document describes two new classes that facilitate the use of the itk::BSplineDeformableTransform in multiresolution registration algorithms. The first class, itk::GridScheduleComputer, defines the B-spline grid, based on a user-specified grid spacing and an input image (the fixed image in registration). A different grid is determined for each resolution level of the registration, given a multiresolution schedule similar to that of the image pyramids. The second class, itk::UpsampleBSplineParametersFilter, resamples (usually upsamples) the B-spline coefficients to a new control point grid having a different spacing/origin/size. This can be used to transfer the result from one resolution level to that of the next level. Summarising, suitable grid definitions follow from the first class, and are used as input for the second class, which performs the actual resampling.

The classes are implemented using the Insight Toolkit www.itk.org. This paper is accompanied with the source code, including a test program.

Latest version available at the Insight Journal [http://hdl.handle.net/1926/????]

Contents

1 Introduction **2**

2 The GridScheduleComputer class **2**

3 The UpsampleBSplineParametersFilter class **3**

4 Conclusion **4**

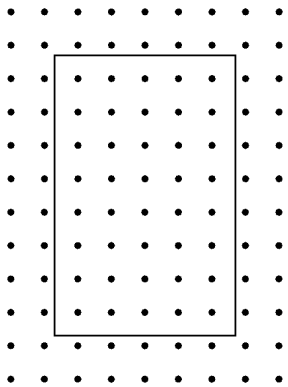

Figure 1: Illustration of the control point grid placement. The square represents the image. The dots are the B-spline control points. The example is for a 3rd order B-spline basis function. Note that we chose to center the grid over the image.

1 Introduction

Nonrigid image registration often requires a multiresolution strategy not only for the images, but also for the transformation. In case of a transformation modelled by B-splines, it is for example common to start with a coarse B-spline grid, which takes care of the more global deformations. The grid is gradually refined, to enable matching of finer structures.

The B-spline grid is defined by the spacing between the control points and the position of the 'first' control point (the grid origin). Some care has to be taken to place a band of control points outside the fixed image domain, such that the transformation is valid for all voxels. The grid definition is therefore also related to the support region of the B-spline basis functions. Since the grid needs to be defined for every resolution level, the functionality is needed regularly. A dedicated class simplifies the procedure. The new class `itk::GridScheduleComputer` takes care of computing the required information for all resolution levels.

Another aspect of multiresolution registration is the initialisation of the transformation in one resolution level, based on the result of the previous resolution level. This requires evaluation of the transformation at the control point locations of the new (finer-scale) grid, followed by a B-spline decomposition. The new class `itk::UpsampleBSplineParametersFilter` implements this procedure.

Included with the source code is a program that tests all functionality of the two classes.

2 The GridScheduleComputer class

Figure 1 illustrates the grid placement strategy implemented by the `itk::GridScheduleComputer` class. We chose to place the B-spline grid such that the image is in the centre. Note the band of control points outside the image. More details on the exact implementation can be found in the `ComputeBSplineGrid()`-function of the class.

In many registration problems, the nonrigid B-spline registration is preceded by a rigid or affine registration. The transformations could be simply summed, in which case there is no problem. In case the transformations are combined by *composition*, the grid placement algorithm needs to be modified. Imagine, for example, that the result of rigid registration is a global translation of 10 mm in the x-direction. Consider a point close to the rightmost boundary of the fixed image. The voxel is first translated 10 mm to the right and

thus possibly falls outside the valid B-spline grid region. Therefore, the B-spline grid should have been placed 10 mm to the right. To facilitate this, the itk::GridScheduleComputer has an optional input: SetInitialTransform().

The itk::GridScheduleComputer requires four user inputs: the BSplineOrder, the FinalGridSpacing, the GridSpacingSchedule, and the domain of the fixed image (given by ImageOrigin, ImageSpacing, and ImageRegion). The BSplineOrder is used to compute the width of the band of control points outside the image. The FinalGridSpacing denotes the control point grid spacing at the finest resolution level. The GridSpacingSchedule defines the downsampling scheme used to determine the grid at lower resolution levels. The schedule has a similar meaning as in the itk::MultiResolutionPyramidImageFilter. The schedule can be supplied in two ways. The first way has the most freedom: the user supplies a vector of a vector of downsampling factors. For every resolution and for every dimension the downsampling factor can thus be specified. The second, simplified way supplies the number of resolution levels and a single downsample factor. For example, a factor 2.0 will halve the grid spacing every resolution, for each dimension. Both schedules end with the final grid spacing.

An example of usage is given below:

```
typedef GridScheduleComputer<CoordRepType,Dimension> GridScheduleComputerType;

GridScheduleComputerType::Pointer gridScheduleComputer
  = GridScheduleComputerType::New();
gridScheduleComputer->SetImageOrigin(  fixedImage()->GetOrigin() );
gridScheduleComputer->SetImageSpacing( fixedImage()->GetSpacing() );
gridScheduleComputer->SetImageRegion(  fixedImage()->GetLargestPossibleRegion() );
gridScheduleComputer->SetInitialTransform( initialTransform ); // optional
gridScheduleComputer->SetFinalGridSpacing( finalGridSpacing );
gridScheduleComputer->SetSchedule( gridSchedule );

/** This function does the work. */
gridScheduleComputer->ComputeBSplineGrid();

/** Obtain the grid definition for a certain resolution level. */
gridScheduleComputer->GetBSplineGrid( resolutionLevel,
  gridRegion, gridSpacing, gridOrigin );
```

3 The UpsampleBSplineParametersFilter class

This filter is based on the code in DeformableRegistration6.cxx, found in the ITK repository (/Examples/Registration). The filter resamples a B-spline coefficient image on a new grid. Usually, it will be used to *upsample* the grid, from one resolution level to the next.

The filter takes as input the old (low-resolution) B-spline grid (origin, spacing, region) and the corresponding coefficients (obtained by bSplineTransform->GetParameters()). The coefficients are resampled on a new, user-specified "required grid". Computation is done by resampling the old grid, using the itk::BSplineResampleImageFunction, followed by B-spline decomposition using the itk::BSplineDecompositionImageFilter. More details can be found in the UpsampleParameters()-function of the class.

An example of usage is given below:

```
typedef UpsampleBSplineParametersFilter<
  ParametersType, CoefficientImageType >  GridUpsamplerType;

GridUpsamplerType::Pointer gridUpsampler
  = GridUpsamplerType::New();

OriginType  currentGridOrigin  = bSplineTransform->GetGridOrigin();
SpacingType currentGridSpacing = bSplineTransform->GetGridSpacing();
RegionType  currentGridRegion  = bSplineTransform->GetGridRegion();
gridUpsampler->SetCurrentGridOrigin( currentGridOrigin );
gridUpsampler->SetCurrentGridSpacing( currentGridSpacing );
gridUpsampler->SetCurrentGridRegion( currentGridRegion );

/** Use the previously described itk::GridScheduleComputer to
 * get the grid domain definition of the next resolution level. */
OriginType  requiredGridOrigin;
SpacingType requiredGridSpacing;
RegionType  requiredGridRegion;
gridScheduleComputer->GetBSplineGrid( newResolutionLevel,
  requiredGridRegion, requiredGridSpacing, requiredGridOrigin );

gridUpsampler->SetRequiredGridOrigin( requiredGridOrigin );
gridUpsampler->SetRequiredGridSpacing( requiredGridSpacing );
gridUpsampler->SetRequiredGridRegion( requiredGridRegion );

/** Get the parameters resulting from the last resolution level.
 * It is assumed in this example that the
 * itk::MultiResolutionImageRegistrationMethod is used. */
ParametersType latestParameters =
  registration->GetLastTransformParameters();
ParametersType upsampledParameters;
gridUpsampler->UpsampleParameters( latestParameters, upsampledParameters );

bSplineTransform->SetGridOrigin( requiredGridOrigin );
bSplineTransform->SetGridSpacing( requiredGridSpacing );
bSplineTransform->SetGridRegion( requiredGridRegion );

registration->SetInitialTransformParametersOfNextLevel( upsampledParameters );
```

4 Conclusion

Two classes were introduced that facilitate multiresolution image registration, using the B-spline transformation. The classes implement commonly used functionality, thereby simplifying integration in the registration framework.

Future work includes integration with the `itk::BSplineDeformableTransformInitializer`, see `http://hdl.handle.net/1926/1338`. That class requires less user interaction, since it directly acts on the transform object. It also takes into account the direction cosines, which we have neglected. An advantage of our class is that it centers the control point grid on the image domain.