A generic interpolator for multi-label images

Release 1.00

Joël Schaerer, Florent Roche and Boubakeur Belaroussi

Nov 24, 2014

BioClinica, Lyon, France

Abstract

We present a generic interpolator for label images. The basic idea is to interpolate each label with an ordinary image interpolator, and return the label with the highest value. This is the idea used by the itk::LabelImageGaussianInterpolateImageFunction interpolator. Unfortunately, this class is currently limited to Gaussian interpolation. Using generic programming, our proposed interpolator extends this idea to any image interpolator. Combined with linear interpolation, this results in similar or better accuracy and much improved computation speeds on a test image.

Contents

Label images are an extension of binary masks, where multiple label values are used to annotate different anatomical regions. Since averaging categorical data makes no sense, most usual interpolators do not work out of the box for label images. As a workaround, many people have been using nearest neighbor interpolation for this taks.

Recently, Yushkevich and Tustison [1] have proposed the `itk::LabelImageGaussianInterpolateImageFunction`, which improves on nearest neighbor interpolation. The basic idea is to interpolate each label independently as if it were a binary mask, and return the label with the maximum interpolated value:

$$I(x) = arg\ \max_{L \in labels} \left(I_L(x) \right)$$

Where I_L is any standard image interpolator:

$$I_L(x): \mathbb{R}^3 \rightarrow \mathbb{R}$$

This allows taking the entire neighborhood into account when interpolating, as opposed to the nearest neighbor only. In contrast with usual iterators, it respects label values by avoiding operations such as averaging on these values.

Unfortunately, the current implementation is limited to using the Gaussian interpolator as the underlying interpolator. We propose to lift this restriction, allowing any ITK iterator to be used for label image interpolation.

1 Generic label interpolation

Our `itk::LabelImageGenericInterpolateImageFunction` takes three template parameters: the image type, the underlying interpolator, and (optionally), the coordinate type.

```
template <typename TInputImage,template<class, typename> class TInterpolator, typename
TCoordRep=double >

class LabelImageGenericInterpolateImageFunction :

  public InterpolateImageFunction<TInputImage, TcoordRep>
```

To avoid duplicating the image for each label, we use a `LabelSelectionAdaptor` which presents the image as a binary mask for each label. To avoid race conditions in multithreaded filters, we create one interpolator and one adaptor for each label when the image is set. Then, the `EvaluateAtContinuousFunction` function calls each interpolator in turn, and returns the label which yields the maximum value.

The `itk::LabelImageGenericInterpolateImageFunction` can be used as any other standard interpolator. The only limitation is that the underlying interpolator must have exactly two template arguments: the image type and the coordinate type. If this is not the case, it should be possible to wrap the interpolator in another class that respects this convention. For example, for the Bspline interpolator, we create this dummy class:

```
template<class TImage,typename TCoordRep> class BSplineInterpolator : public
itk::BSplineInterpolateImageFunction<TImage,TCoordRep> {};
```

It can then be used as follows:

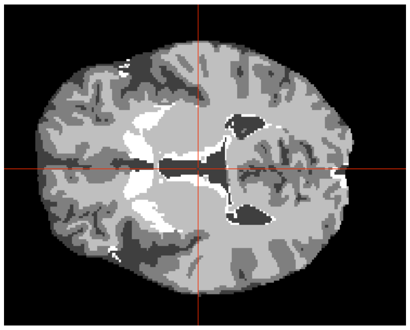
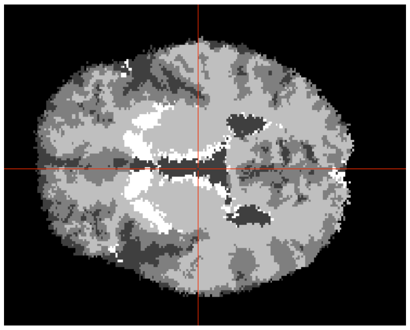
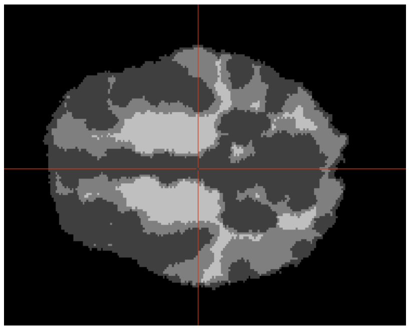
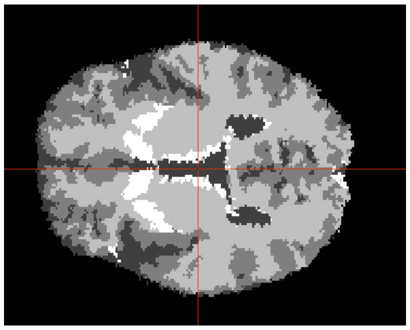
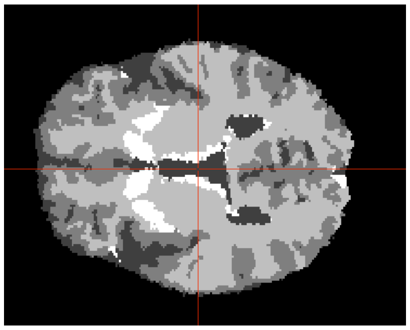
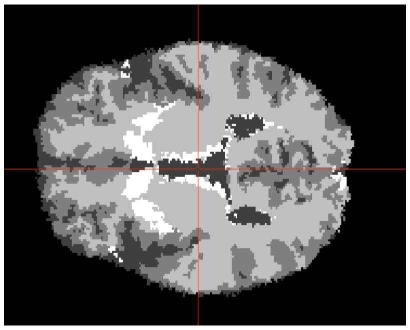
```
typedef itk::LabelImageGenericInterpolateImageFunction<ImageType,BSplineInterpolator>
GBSInterpolatorType;

GBSInterpolatorType::Pointer gbs_interp = GBSInterpolatorType::New();
```

2 Test using a brain classification image

1. Visual assesment

To evaluate the various possible label image interpolators, we used a sample brain classification label map, with four labels: background, cerebro-spinal fluid, gray matter and white matter. We proceed to make it perform a full rotation around the z axis in a given number of steps, resampling the image after each step. If the interpolator performs correctly, the final image should be in the same position as the original image, and hopefully not too distorted. The results are the following:

 a) Original image	 b) Nearest neighbor	 c) Linear interpolation
 d) LabelImageGaussian	 e) LabelImageLinear	 f) LabelImageBSpline

Spline order was set to 3. The standard deviation for the Gaussian interpolator was set to 0.3, which was the value found to give the best results.

As expected, vanilla linear interpolation (1-c) performs very poorly. Nearest neighbor interpolation (1-b) respects label values, but still distorts the contours a lot. The three label image interpolators (1-d,e,f) unsurprisingly give the best results, with a slight edge for the generic interpolator using linear interpolation as the underlying interpolator.

2. Quantitative results

We reproduced the previously described test with 3, 5 and 7 rotation steps. To get a quantitative estimate of each interpolator's error rate, we count the number of pixels that differ from the original image after the full rotation. In addition to the previously shown interpolators, we add the generic label interpolator with nearest neighbor and gaussian interopolation. The results are given in Table 1. Again, the generic label interpolator with linear interpolation gives the best results. It also has a much faster computation time than more complex interpolators.

As expected, the generic versions of nearest neighbor and Gaussian interpolation give exactly the same results as the plain nearest neighbor and LabelImageGaussian interpolators, at a cost in computational time.

Interpolator	**Rotations**	**# Differences**	**# Differences (%)**	**Computation time**
Nearest neighbor interpolator	3	86376	0.79%	0.14s
Linear interpolator	3	629094	5.78%	0.20s
Label Gaussian interpolator type	3	70097	0.64%	5.10s
Generic label interpolator with nearest neighbor	3	86376	0.79%	0.71s
Generic label interpolator with linear interpolation	**3**	**65826**	**0.61%**	**1.80s**
Generic label interpolator with B-Spline	3	69088	0.64%	22.17s
Generic label interpolator with Gaussian	3	70097	0.64%	20.38s
Nearest neighbor interpolator	5	167969	1.54%	0.08s
Linear interpolator	5	862401	7.93%	0.22s
Label Gaussian interpolator type	5	131776	1.21%	9.77s
Generic label interpolator with nearest neighbor	5	167969	1.54%	1.87s
Generic label interpolator with linear interpolation	**5**	**115469**	**1.06%**	**3.16s**
Generic label interpolator with B-Spline	5	124410	1.14%	38.09s
Generic label interpolator with Gaussian	5	131776	1.21%	39.01s
Nearest neighbor interpolator	7	208304	1.91%	0.28s
Linear interpolator	7	996401	9.16%	2.48s
Label Gaussian interpolator type	7	163821	1.51%	11.29s
Generic label interpolator with nearest neighbor	7	208304	1.91%	2.69s
Generic label interpolator with linear interpolation	**7**	**148282**	**1.36%**	**4.63s**
Generic label interpolator with B-Spline	7	161522	1.48%	51.06s
Generic label interpolator with Gaussian	7	163821	1.51%	48.95s

Table 1: Measured accuracy and computation times for various interpolators

3 Conclusions

We proposed a new interpolator for multi-label images, which is able to transform any image interpolator into a multi-label image interpolator, re-using the idea from [1]. With a simple test using a sample brain classification, we find that the new interpolator gives the best results when used with linear interpolation. It also has a very competitive computation time.

References

[1] Yushkevich and Tustison, itk::LabelImageGaussianInterpolateImageFunction, http://www.itk.org/Doxygen/html/classitk_1_1LabelImageGaussianInterpolateImageFunction.html