An image sampling framework for the ITK

Release 1.0

Marius Staring[1] and Stefan Klein[2]

August 12, 2010

[1]Division of Image Processing, Leiden University Medical Center, Leiden, The Netherlands
[2]Biomedical Imaging Group Rotterdam, Departments of Radiology & Medical Informatics, Erasmus MC, Rotterdam, The Netherlands

Abstract

This document describes the implementation of image samplers using the Insight Toolkit ITK www.itk.org. Image samplers take a set of 'picks' from an image and store them in an array. A sample consists of the location of the pick (a point) and the corresponding image intensity (a value). Image samplers are useful for image registration, where samples are drawn from the fixed image in order to compute the similarity measure. Together with an image sampler base class, we introduce the following image samplers: 1) a full sampler that draws all voxel coordinates from the input image, 2) a grid sampler that draws samples from a user-specified regular voxel grid, 3 and 4) two random samplers that uniformly draw a user-specified number of samples from the input image.

This paper is accompanied with the source code, input data, parameters and output data that the authors used for validating the algorithm described in this paper.

Latest version available at the Insight Journal [http://hdl.handle.net/1926/1338]

Contents

1 Definitions

An input image is denoted as $I(\boldsymbol{x})$ with spatial position $\boldsymbol{x} = (x_1, \ldots, x_d)$, with d the dimension of the input image. We assume that we have scalar data: $I(\boldsymbol{x}) \in \mathbb{R}$. Let the image size be $n_1 \times \cdots \times n_d$.

A sample is defined as the pair $s = \{\boldsymbol{x}, I(\boldsymbol{x})\}$ containing the spatial position and the intensity at that position. A sample set, or the image samples, are a collection of pairs $\mathcal{S} = \{s_i | i = 1, \ldots, N\}$, with N the number of samples that are drawn.

We define the following samplers:

ImageFullSampler: Every voxel from I is put in the sample set, which therefore has size $\prod_{i=1}^{d} n_i$. This sample set is basically another way of storing the input image.

ImageGridSampler: The user specifies a regular grid on the input image, by supplying a downsampling factor for each dimension. The grid sampler then only pick samples from this grid. Effectively, this sampler downsamples the input image and subsequently performs full sampling.

ImageRandomSampler: The random sampler selects a user-specified number of samples N from the input image. Every voxel in the input image has an equal probability of being selected. New samples are added to the sample set until its size is N. A voxel from the input image is not necessarily selected only once.

ImageRandomCoordinateSampler: All previous image samplers only select from the voxel grid. The ImageRandomCoordinateSampler is not limited to this grid, but can also select coordinates between voxels. The intensity $I(\boldsymbol{x})$ must then be obtained by interpolation. Other than that it is equal to the ImageRandomSampler.

While at first glance the full sampler seems the most obvious choice for image registration, in practice it is not always smart to do so. The computational complexity of a similarity metric in image registration is directly related to the size of the sample set, which can grow large for large images when using full sampling. Random samplers can be quite useful in this case in combination with a stochastic optimisation routine [2].

In addition to sampling from the entire image, it is possible to sample from a subregion of an image. This is useful in image registration when the input images have spatially varying intensity distributions (of which MRI data is an example). Local image metrics like Local Mutual Information (LMI) are described in more detail in [3].

2 Implementation

The image sample class is quite simple, it simply stores $\boldsymbol{x}$ as an `itk::Point` and $I(\boldsymbol{x})$ as a `RealType`, both as public member variables. The sample set is implemented by a `itk::VectorDataContainer`, a derived class of `std::vector` and `itk::DataObject`. In addition, a `itk::VectorContainerSource` was created from which an `itk::ImageToVectorContainerFilter` inherits, from which all image samplers inherit.

The ImageSampler base class allows to set the input image, an input region, and optionally a mask:

```
SetInput()
SetInputImageRegion()
SetMask()
```

Furthermore, a method is defined to force the sampler to regenerate the sample set on calling Update(). This makes of course only sense for the Random samplers.

```
SelectNewSamplesOnUpdate();
```

Also a function is defined that returns whether the sampler supports SelectNewSamplesOnUpdate():

```
bool SelectingNewSamplesOnUpdateSupported()
```

which returns true in case the sampler is stochastic.

In more detail, the image samplers work as follows:

ImageFullSampler: We iterate over the sample region until the iterator is at the end, and add $\boldsymbol{x}$ and $I(\boldsymbol{x})$ to the image sample container, when $\boldsymbol{x}$ is within the supplied mask.

ImageGridSampler: The user can supply a grid spacing schedule or the number of desired samples N. In the latter case the grid sampling schedule is derived from N, such that approximately N samples are drawn. We iterate over the grid and add the pair $(\boldsymbol{x}, I(\boldsymbol{x}))$ to the image sample container, when $\boldsymbol{x}$ is within the supplied mask.

ImageRandomSampler: Iterate over the sample region using an `itk::ImageRandomConstIteratorWithIndex` until we have drawn N valid (= within mask) samples.

ImageRandomCoordinateSampler: The `itk::MersenneTwisterRandomVariateGenerator` is used to randomly draw (continuous) coordinates within the appropriate image region. Continue until N valid samples are drawn.

ImageRandomSamplerSparseMask: This image sampler is useful in combination with small masks. The above two random samplers may not find enough valid samples in a reasonable amount of time in combination with small masks. Therefore the sparse mask sampler first performs a full sampling over the masks and stores this intermediate result. Subsequently, it randomly draws samples from the intermediate result until N.

The ImageRandomCoordinateSampler supports drawing samples from a randomly chosen subregion. The user supplies a region size (the size of the box). A region center is drawn randomly in such a way that the box fits completely within the input image. Subsequently, N samples are drawn from within the box. This could be implemented for other image samplers as well of course.

Example of usage:

```
// Some typedefs
typedef itk::Image< short, 3 >                 ImageType;
typedef itk::ImageRandomSampler< ImageType >   SamplerType;
typedef SamplerType::ImageSampleContainerType  SampleContainerType;
typedef SamplerType::ImageSampleType           SampleType;

// read the input image ...

// Create, setup and run the image sampler
```

```
typename SamplerType::Pointer sampler = SamplerType::New();
sampler->SetInput( inputImage );
sampler->SetInputImageRegion( inputImage->GetBufferedRegion() );
sampler->SetNumberOfSamples( 2000 );
sampler->Update();

// Get sample container */
SampleContainerType::Pointer samples = sampler->GetOutput();
SampleType sample;

// Print first 10 samples
std::cout << "First 10 samples:" << std::endl;
std::cout << "Sample\tValue\tCoordinates" << std::endl;
for ( unsigned int i = 0; i < 10; i++ )
{
  sample = samples->ElementAt( i );
  std::cout << i << "\t" << sample.m_ImageValue
    << "\t" << sample.m_ImageCoordinates << std::endl;
}
```

3 Experiments

In Figure 1 we illustrate the behaviour of the image samplers.

4 Use in image registration

A said, a possible application of image samplers is in the context of image registration. This has been implemented in `elastix` [1] already. In the following we sketch how to adapt the `itk::ImageToImageMetric`'s to use image samplers.

Add some typedefs and methods to enable setting/getting the image sampler in the metric:

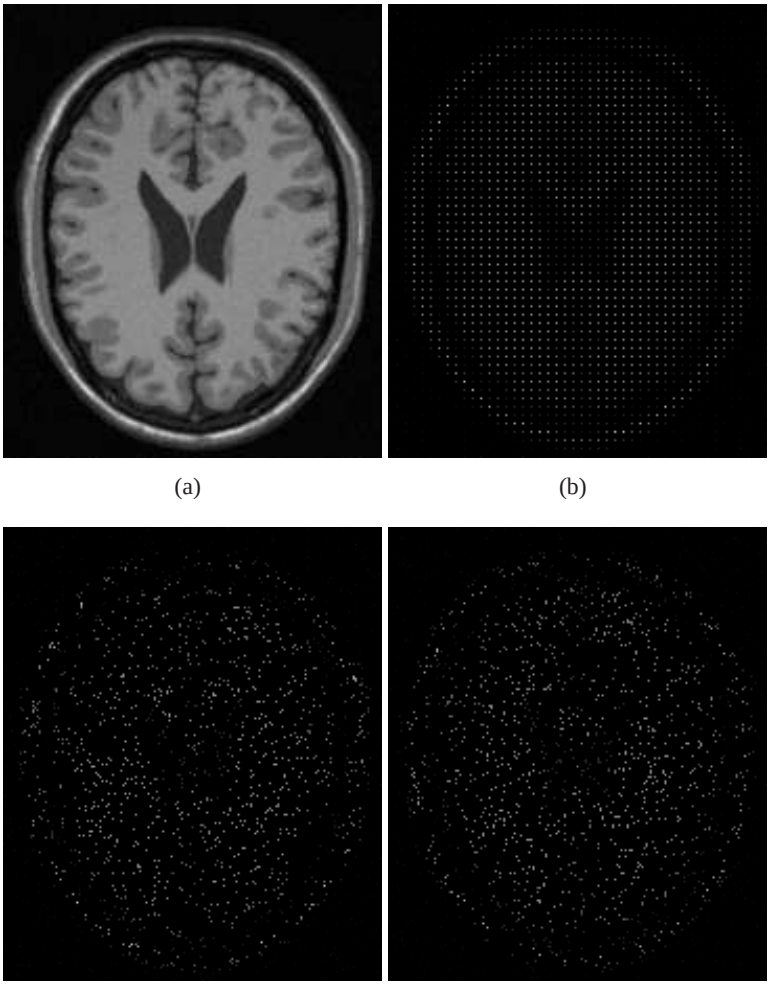
```
typedef itk::ImageSampler< FixedImageType > ImageSamplerType;
itkSetObjectMacro( ImageSampler, ImageSamplerType );
itkGetConstObjectMacro( ImageSampler, ImageSamplerType );
```

In the functions `GetValue()`, `GetDerivative()` and `GetValueAndDerivative()`, instead of looping over the fixed image, we need to loop over the sample container:

```
  /** Update the imageSampler and get a handle to the sample container. */
  this->GetImageSampler()->Update();
  ImageSampleContainerPointer sampleContainer = this->GetImageSampler()->GetOutput();

  /** Create iterator over the sample container. */
  typename ImageSampleContainerType::ConstIterator fiter;
  typename ImageSampleContainerType::ConstIterator fbegin = sampleContainer->Begin();
  typename ImageSampleContainerType::ConstIterator fend = sampleContainer->End();

  /** Loop over the fixed image samples to calculate ... */
  for ( fiter = fbegin; fiter != fend; ++fiter )
```



(a) (b)

(c) (d)

Figure 1: Illustrating the sampling mechanism. (a) is the 2D input image. Samples are drawn from it ($N = 2000$) and displayed by a white dot in (b)-(d), with (b) the grid sampler, (c) the random sampler, and (d) the random coordinate sampler.

```
{
  const RealType & fixedImageValue = static_cast<RealType>( (*fiter).Value().m_ImageValue );
  const FixedImagePointType & fixedPoint = (*fiter).Value().m_ImageCoordinates;

  ... do something with it ...
}
```

We have already done this in elastix (check out http://elastix.isi.uu.nl), resulting in the itk::AdvancedImageToImageMetric, and adapted versions of the itk::MeanSquaresImageToImageMetric, itk::NormalizedCorrelationImageToImageMetric, itk::MattesMutualInformationImageToImageMetric, itk::KappaStatisticImageToImageMetric and some more. The big advantage of this implementation is that the image sampling strategies get separated from the metric definition. Currently, in the ITK some choices for sampling are given, but they are all implemented in the itk::ImageToImageMetric, greatly reducing code readability. The proposed image samplers framework also makes it straightforward for developers to implement new sampling strategies (for example, sampling on edges) and use these in existing metrics.

5 Conclusion

We propose a new image sampling framework for the ITK, especially useful for the image registration framework. The upcoming ITK4 refactoring seems like a great opportunity to modify the current implementation.

References

[1] S. Klein, M. Staring, K. Murphy, M.A. Viergever, and J.P.W. Pluim. elastix: a toolbox for intensity-based medical image registration. *IEEE Transactions on Medical Imaging*, 29(1):196 – 205, 2010. 4

[2] S. Klein, M. Staring, and J. P. W. Pluim. Evaluation of optimization methods for nonrigid medical image registration using mutual information and B-splines. *IEEE Transactions on Image Processing*, 16(12):2879 – 2890, December 2007. 1

[3] S. Klein, U. A. van der Heide, and I. M. Lips *et al.* Automatic segmentation of the prostate in 3D MR images by atlas matching using localized mutual information. *Medical Physics*, 35(4):1407 – 1417, April 2008. 1