Slice by slice filtering with ITK

Gaëtan Lehmann

November 13, 2006

INRA, UMR 1198; ENVA; CNRS, FRE 2857, Biologie du Développement et Reproduction, Jouy en Josas,
F-78350, France

Abstract

While filtering in N dimensions is a main feature of ITK, filtering an image in N-1 dimensions, slice by slice, can be very useful in many cases. Currently, this operation require a consequent amount of work to be done with ITK. A new filter is provided with this article to perform this operation with only a few lines of code.

1 Description and code example

This filter is better descibed by a simple example. For example, suppose we want to perform a median filtering on all the slices of an image[1]

We first do the standard includes, and check the command line arguments.

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkSimpleFilterWatcher.h"

#include "itkMedianImageFilter.h"
#include "itkSliceBySliceImageFilter.h"


int main(int argc, char * argv[])
{

  if( argc != 3 )
    {
    std::cerr << "usage: " << argv[0] << " input output" << std::endl;
    exit(1);
    }
```

The dimension of the image, the pixel type, and the image type are declared. A file reader is created.

[1] SliceBySliceImageFilter is not required in that case: the most simple solution is to set the radius to 0 on one dimension - that's only an example.

```
  const int dim = 3;
  typedef unsigned char PType;
  typedef itk::Image< PType, dim > IType;

 typedef itk::ImageFileReader< IType > ReaderType;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName( argv[1] );
```

We then declare the type of the SliceBySliceImageFilter, instantiate a filter, and set the image from the reader as input. At this point, the filter can't do anything: the developper have to give him a filter which will be used internally to perform the transform on all the classes.

```
  typedef itk::SliceBySliceImageFilter< IType, IType > FilterType;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput( reader->GetOutput() );
```

We declare the type of the internal filter - a median - using the type defined in the SliceBySliceImageFilter, instantiate it, and set the options correctly. InternalInputImageType, and InternalOutputImageType are the same type than the input and output image type of the SliceBySliceImageFilter, but decreased of one dimension - here both are `itk::Image< unsigned char, 2 >`.

```
  typedef itk::MedianImageFilter< FilterType::InternalInputImageType,
                                  FilterType::InternalOutputImageType > MedianType;
  MedianType::Pointer median = MedianType::New();
  MedianType::InputSizeType rad;
  rad.Fill( 5 );
  median->SetRadius( rad );
```

The median is passed to the slice by slice filter using `SetFilter()`.

```
  filter->SetFilter( median );
```

Finally, the output is wrote to a file. When `Update()` is called, the slice by slice filter runs the median filter on all the slices of the image, and store the result in its output image. The dimension reduced to pass to dimension N-1 can be selected with `SetDimension()` and defaults to the highest one.

```
  itk::SimpleFilterWatcher watcher(filter, "filter");

  typedef itk::ImageFileWriter< IType > WriterType;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput( filter->GetOutput() );
  writer->SetFileName( argv[2] );
  writer->Update();

  return 0;
}
```

2 Conclusion

The new provided class gives an easy way to perform a slice by slice transform of an image.

References

[1] L. Ibanez and W. Schroeder. *The ITK Software Guide*. Kitware, Inc. ISBN 1-930934-10-6, http://www.itk.org/ItkSoftwareGuide.pdf, 2003.