Combining labeled images with ITK

Gaëtan Lehmann[2]

April 15, 2007

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

Abstract

Combining some labeled images is a quite common and useful task in image analysis. That's something we may want to do to prepare the markers for a watershed from markers for example, or to write the output of several segmentations in a single file. This article comes with three new filters to make the combination of labeled images easier with ITK. This contribution also try to provide an implementation better integrated with the ITK pipeline model than the itkMultipleUnlabeledImagesToLabeledImage-Filter contribution.

1 How to combine the labels from several images

There are two major problems while combining the labels from several images:

- avoiding the label collision. The input images may or may not compatibles labels. The labels may be modified to avoid the collision, but it might be necessary to know there exact value after the combination.
  - The labels must be kept unchanged. The user must take care to avoid the collisions by himself.
  - The labels can be modified, but the user must be able to identify them easily. The simplest solution is to shift the label values according to the image number.
  - The labels can be fully modified. The images will be relabeled, all the first labels will come from the first image, the next ones from the second image, etc. This method also produce consecutive labels - often a desirable feature.
- avoiding the labeled region collision. This is a much simple problem: an exception can be throws when a collision occure, or we can choose to give a higher priority to a label than the others. For this last point, the value of the label don't seem to be a good choice because the labels are not supposed to be ordered. A better choice seems to be the image number. By convention, let say that's the label from the last image.

2 Class description an usage examples

All the new filters provided with that article are subclassing the NaryFunctorImageFilter, or are implementing a similar interface. They all take one or more images as input, and produce a single output image.

They also provide the SetIgnoreCollision(bool) method which let the user choose to ignore the labeled region collision or not. By default, they are ignored.

Finally, they all provide the SetBackgroundValue(OutputPixelType) wich let the user set the value of the background label.

2.1 NaryLabelImageFilter

This filter take all the labels which are not background pixels in all the input images, and put them in the output image. The labels may be shifted on a per image basis, by calling SetShift(OutputPixelType) with a value different of 0. By default, the labels are not shifted.

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

#include "itkNaryLabelImageFilter.h"


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

  if( argc != 7 )
    {
    std::cerr << "usage: " << argv[0] << " input1 input2 output bg shift collision" << std::endl;
    // std::cerr << "  : " << std::endl;
    exit(1);
    }

  const int dim = 2;

  typedef unsigned char PType;
  typedef itk::Image< PType, dim > IType;

  typedef itk::ImageFileReader< IType > ReaderType;
  ReaderType::Pointer reader1 = ReaderType::New();
  reader1->SetFileName( argv[1] );

  ReaderType::Pointer reader2 = ReaderType::New();
  reader2->SetFileName( argv[2] );

  typedef itk::NaryLabelImageFilter< IType, IType > FilterType;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput( 0, reader1->GetOutput() );
  filter->SetInput( 1, reader2->GetOutput() );
  filter->SetBackgroundValue( atoi(argv[4]) );
```

```
  filter->SetShift( atoi(argv[5]) );
  filter->SetIgnoreCollision( atoi(argv[6]) );

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

  typedef itk::ImageFileWriter< IType > WriterType;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput( filter->GetOutput() );
  writer->SetFileName( argv[3] );
  writer->Update();

  return 0;
}
```

2.2 NaryRelabelImageFilter

This filter search all the label in the image, and give them a new value so they are all consecutive. It then do the same with the second image, and give them the labels immediately after the ones of the first image. It then do the same with the third image, etc. The new labels are then copied to the output image. Contrary to the previous filter, the user can't easily identify a label in the input image. However, he is sure to get no label collision, and only consecutive labels in the output image (if there is no labeled region collision).

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

#include "itkNaryRelabelImageFilter.h"


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

  if( argc != 6 )
    {
    std::cerr << "usage: " << argv[0] << " input1 input2 output bg collision" << std::endl;
    // std::cerr << "  : " << std::endl;
    exit(1);
    }

  const int dim = 2;

  typedef unsigned char PType;
  typedef itk::Image< PType, dim > IType;

  typedef itk::ImageFileReader< IType > ReaderType;
  ReaderType::Pointer reader1 = ReaderType::New();
  reader1->SetFileName( argv[1] );

  ReaderType::Pointer reader2 = ReaderType::New();
  reader2->SetFileName( argv[2] );

  typedef itk::NaryRelabelImageFilter< IType, IType > FilterType;
```

```
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput( 0, reader1->GetOutput() );
  filter->SetInput( 1, reader2->GetOutput() );
  filter->SetBackgroundValue( atoi(argv[4]) );
  filter->SetIgnoreCollision( atoi(argv[5]) );

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

  typedef itk::ImageFileWriter< IType > WriterType;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput( filter->GetOutput() );
  writer->SetFileName( argv[3] );
  writer->Update();

  return 0;
}
```

2.3 NaryBinaryToLabelImageFilter

This filter in very similar to the previous one, but for binary images. Only the pixels with the value set with SetForegroundValue(InputImageType) are considered to be an object. The other are background.

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

#include "itkNaryBinaryToLabelImageFilter.h"


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

  if( argc != 7 )
    {
    std::cerr << "usage: " << argv[0] << " input1 input2 output bg fg collision" << std::endl;
    // std::cerr << "  : " << std::endl;
    exit(1);
    }

  const int dim = 2;

  typedef unsigned char PType;
  typedef itk::Image< PType, dim > IType;

  typedef itk::ImageFileReader< IType > ReaderType;
  ReaderType::Pointer reader1 = ReaderType::New();
  reader1->SetFileName( argv[1] );

  ReaderType::Pointer reader2 = ReaderType::New();
  reader2->SetFileName( argv[2] );

  typedef itk::NaryBinaryToLabelImageFilter< IType, IType > FilterType;
```

```
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput( 0, reader1->GetOutput() );
  filter->SetInput( 1, reader2->GetOutput() );
  filter->SetBackgroundValue( atoi(argv[4]) );
  filter->SetForegroundValue( atoi(argv[5]) );
  filter->SetIgnoreCollision( atoi(argv[6]) );

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

  typedef itk::ImageFileWriter< IType > WriterType;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput( filter->GetOutput() );
  writer->SetFileName( argv[3] );
  writer->Update();

  return 0;
}
```

3 Wrappers

As usual, the wrappers for WrapITK are provided with the sources, and have been sucessfully tested with python.

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.