Efficient border detection in binary and label images

Gaëtan Lehmann[1]

April 23, 2008

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

Abstract

Currently in ITK, the only way to find the border of the objects in a binary image is to use the `itk::BinaryErodeImageFilter`, with a kernel of radius 1. This filter is a generic filter, made to support any shape and size of structuring element, and thus is not optimized for the particular case needed to detect the borders. Moreover, that filter is not multithreaded, so it can't get the performance improvements allowed by the multiprocessor systems. As a result, the border detection can be quite time consuming currently – for example, in `itk::SignedMaurerDistanceMapImageFilter`, the border detection takes about 33% of the execution time.

This contribution comes with a new filter which highly improve the performance of the border detection in the binary image, and a second filter which allow the detection of the borders in label images with similar performance.

1 Algorithm and implementation

The `itk::BinaryBorderImageFilter` and the `itk::LabelBorderImageFilter` are based on `itk::ConnectedComponentImageFilter`[1] from [1]. The images is first transformed in a set of lines in the foreground and in the background. The foreground lines which are touching the background are found in the second step, as well as the determination of the exact part of the line on the border of the object. That way, the image is visited only in raster order, and the comparisons made on blocks of pixels.

Both filters can be run in place.

2 Binary and label images

`itk::BinaryBorderImageFilter` performs a border detection on a single label, and preserves the other (background) values. `itk::LabelBorderImageFilter` a detection on all the labels in the image, excepted the background label. Because `itk::BinaryBorderImageFilter` works only on two kinds of pixels, it can be much optimized than `itk::LabelBorderImageFilter`. The performance difference are not that

[1]More precisely, on a multithreaded version of `itk::ConnectedComponentImageFilter`, available at `http://voxel.jouy.inra.fr/darcs/contrib-itk/watershed/`

important though, because most of the execution time is spent in reading the input image, and writing the output one.

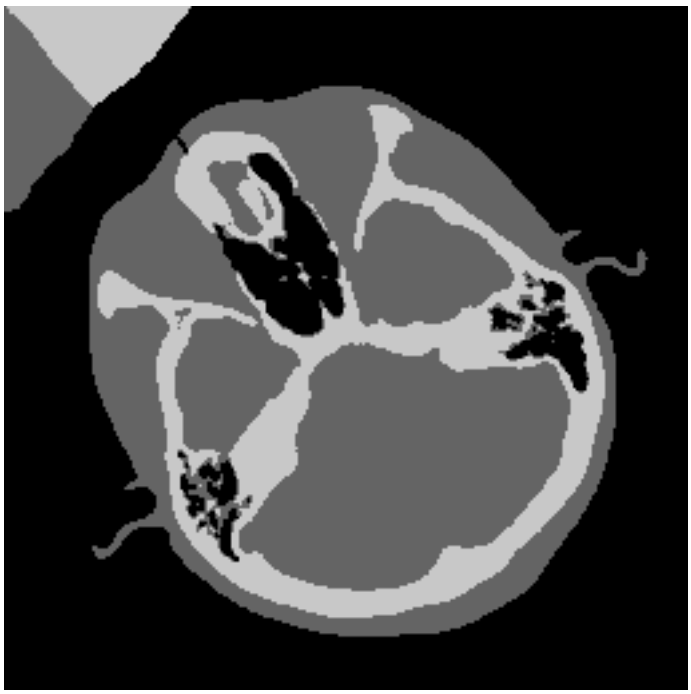

Figure 1: The input image.

3 Performance

The performance tests have been run on a Intel® Pentium® 4 CPU at 3.20GHz with 2GB of RAM running linux 2.6.12, with the command `./perf_threads image/big.nrrd`. The size of the image used for the test is $1024 \times 1024 \times 89$.

The execution time is 1.29 seconds for the binary case, 1.42 seconds for the label case, and 67.6 seconds for the morphological erosion. The speed up is 52.4 for the binary case.

4 Examples

4.1 Binary case

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

#include "itkBinaryBorderImageFilter.h"


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

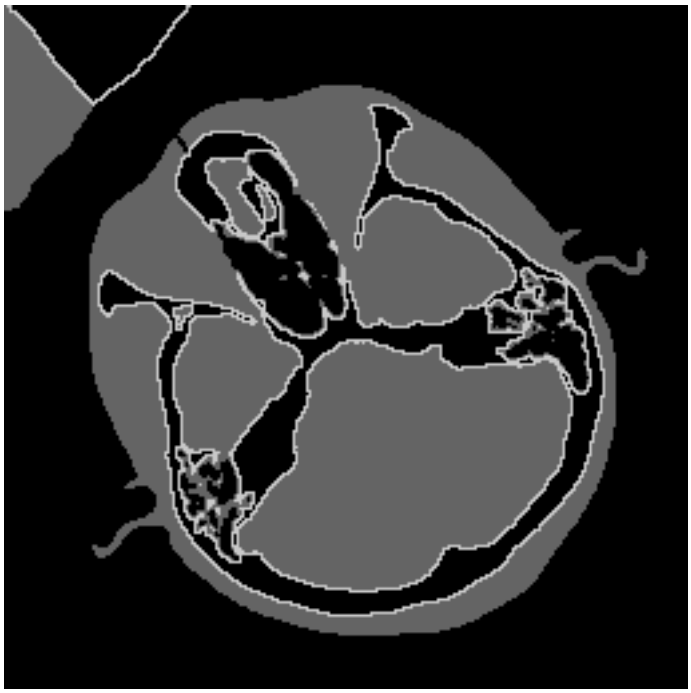

Figure 2: Binary border detection (foreground value is 200). The background is unchanged.

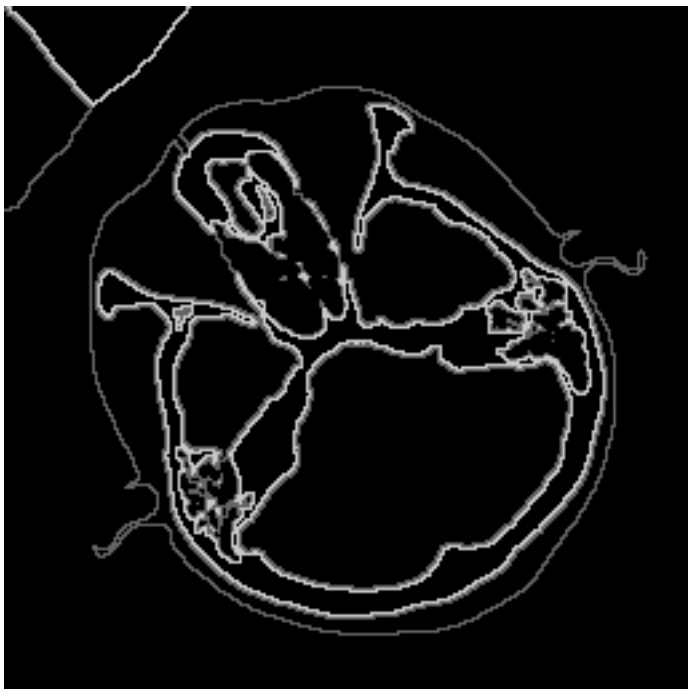

Figure 3: Label border detection. All the label in the image are changed.

```
  if( argc != 6 )
    {
    std::cerr << "usage: " << argv[0] << " intput output fullyConnected fg bg" << std::endl;
    std::cerr << " input: the input image" << std::endl;
    std::cerr << " output: the output image" << std::endl;
    std::cerr << " fullyConnected: 0 or 1" << std::endl;
    exit(1);
    }

//   itk::MultiThreader::SetGlobalMaximumNumberOfThreads(1);
  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] );

  typedef itk::BinaryBorderImageFilter< IType, IType > FilterType;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput( reader->GetOutput() );
  filter->SetFullyConnected( atoi(argv[3]) );
  filter->SetForegroundValue( atoi(argv[4]) );
  filter->SetBackgroundValue( 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[2] );
  writer->Update();

  return 0;
}
```

4.2 Label case

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

#include "itkLabelBorderImageFilter.h"


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

  if( argc != 5 )
    {
    std::cerr << "usage: " << argv[0] << " intput output fullyConnected bg" << std::endl;
```

```
    std::cerr << " input: the input image" << std::endl;
    std::cerr << " output: the output image" << std::endl;
    std::cerr << " fullyConnected: 0 or 1" << std::endl;
    exit(1);
    }

//   itk::MultiThreader::SetGlobalMaximumNumberOfThreads(1);
  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] );

  typedef itk::LabelBorderImageFilter< IType, IType > FilterType;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput( reader->GetOutput() );
  filter->SetFullyConnected( atoi(argv[3]) );
  filter->SetBackgroundValue( atoi(argv[4]) );

  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;
}
```

5 Acknowledgments

I thank Dr Pierre Adenot and MIMA2 confocal facilities (http://mima2.jouy.inra.fr) for providing the 3D test image.

6 Conclusion

That contribution can highly enhance the performance of the algorithms where the border of the objects must be detected. Use in `itk::SignedMaurerDistanceMapImageFilter`[2] instead of `itk::BinaryErodeImageFilter`, it can speedup the execution time by about 30%.

References

[1] Richard Beare. Optimization of connected component labelling. *Insight Journal*, 2006. 1

[2] A multithreaded version of `itk::SignedMaurerDistanceMapImageFilter` which use that contribution is available at http://voxel.jouy.inra.fr/darcs/contrib-itk/watershed/