Improving features and performance of binary erode and dilate filters

Gaëtan Lehmann

February 19, 2006

Unité de Biologie du Développement et de la Reproduction, Institut National de la Recherche Agronomique, 78350 Jouy-en-Josas, France

Abstract

Binary erosion and dilation are the base filters of binary mathematical morphology. In ITK 2.4.1 `BinaryDilateImageFilter` and `BinaryErodeImageFilter` implement an efficient algorithm described in [2]. However, those filter lack support for boundary values, have a poor progress report, and can be quite inefficient for complex 3D images.

1 Implementation

Thread support have been completely removed: only a small part of the filter is treadable, and sadly, it's the creation of the output image during the flooding stage which take the most time to run, and it can't be threaded.

Adding boundary support was quite easy: the temporary image used internally is created with an extra pixel on all borders, and the value of those pixel is set to foreground or background value, depending of the option set by the user.

To improve the progress report, the output pixels are set during the burning stage, so there is no need to create a list with an unknown size.

To improve the performance, the output image is no more created with an output iterator - the iterator is really inefficient in that case, because it copy all the neighbor pixels in a buffer, while the filter need to access a small number of those pixels.

2 Performance

A timing test comparing performance on a $371 \times 371 \times 34$ image with a binary ball structuring element of size $20 \times 20 \times 7$ shows that the new filters give better results than the old ones.

border is foreground	foreground value	new erode	old erode	new dilate	old dilate
no	100	18.4086 s	/	11.7803 s	59.7689 s
no	200	13.5472 s	/	2.65948 s	8.28261 s
yes	100	16.2584 s	58.4875 s	19.8128 s	/
yes	200	11.7388 s	17.3176 s	10.7766 s	/

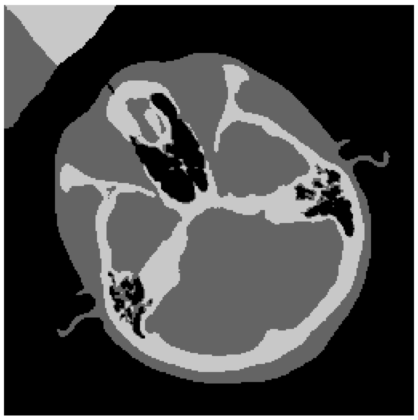
Table 1: Execution times.

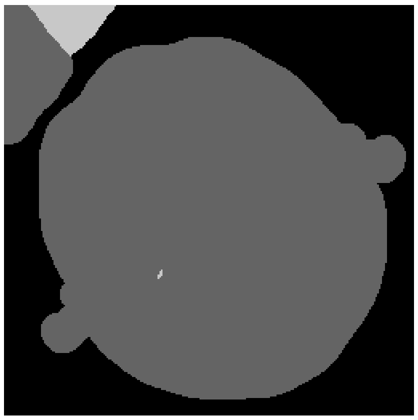
3 Examples

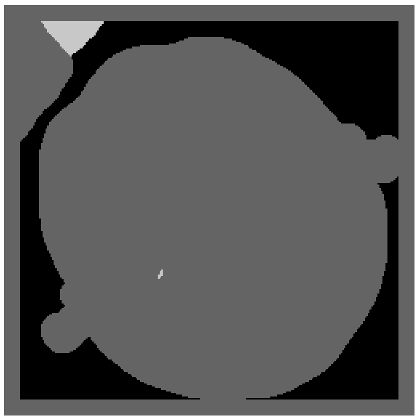
Figures 1, 2 and 3 show some examples of of dilation and erosion with different structuring elements, and different parameters.

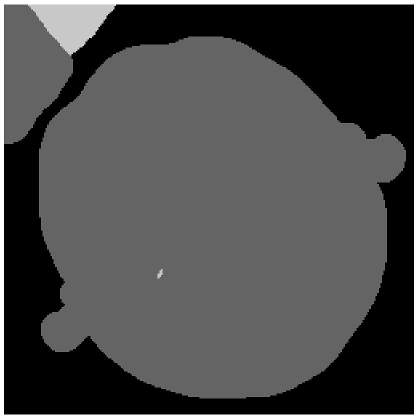
4 Code sample

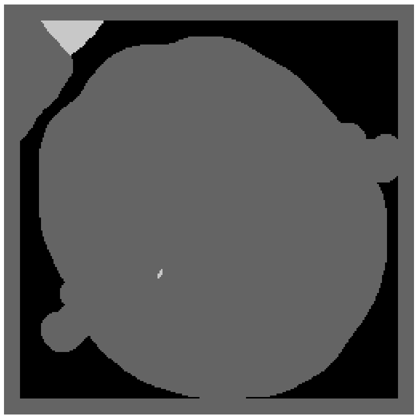
```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkCommand.h"
#include "itkSimpleFilterWatcher.h"

#include "itkBinaryDilateImageFilter.h"
#include "itkBinaryBallStructuringElement.h"
#include "itkNeighborhood.h"


int main(int, char * argv[])
{
  const int dim = 2;

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

  typedef itk::ImageFileReader< IType > ReaderType;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName( argv[4] );

  typedef itk::BinaryBallStructuringElement< PType, dim > SRType;
  SRType kernel;
  kernel.SetRadius( 10 );
  kernel.CreateStructuringElement();

  typedef itk::BinaryDilateImageFilter< IType, IType, SRType > FilterType;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput( reader->GetOutput() );
  filter->SetKernel( kernel );
  filter->SetForegroundValue( atoi(argv[1]) );
  filter->SetBackgroundValue( atoi(argv[2]) );
  filter->SetBoundaryIsForeground( atoi(argv[3]) );

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

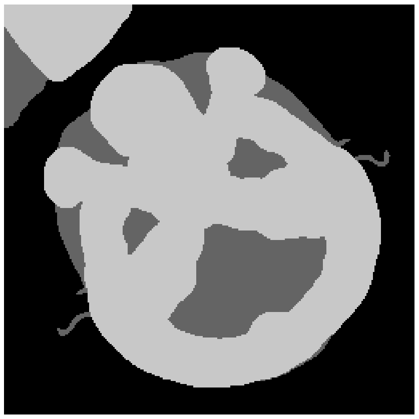

(a) input image

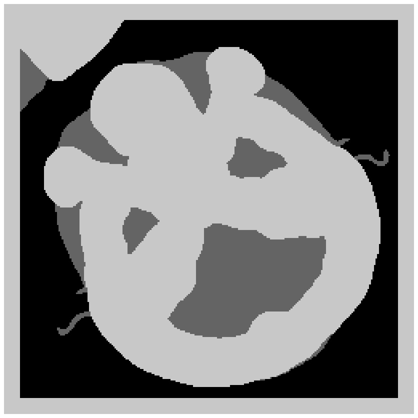

(b) foreground = 100, background = 0, border is background

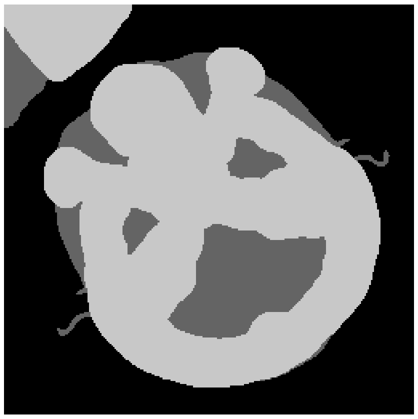

(c) foreground = 100, background = 0, border is foreground


(d) foreground = 100, background = 150, border is background


(e) foreground = 100, background = 150, border is foreground


(f) foreground = 200, background = 0, border is background


(g) foreground = 200, background = 0, border is foreground


(h) foreground = 200, background = 150, border is background

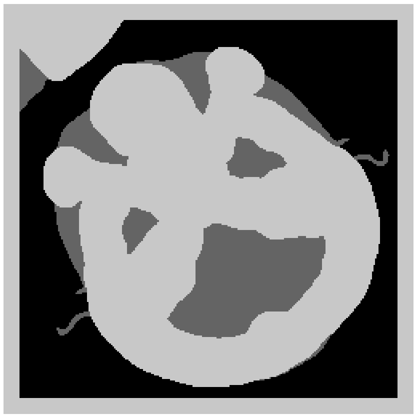
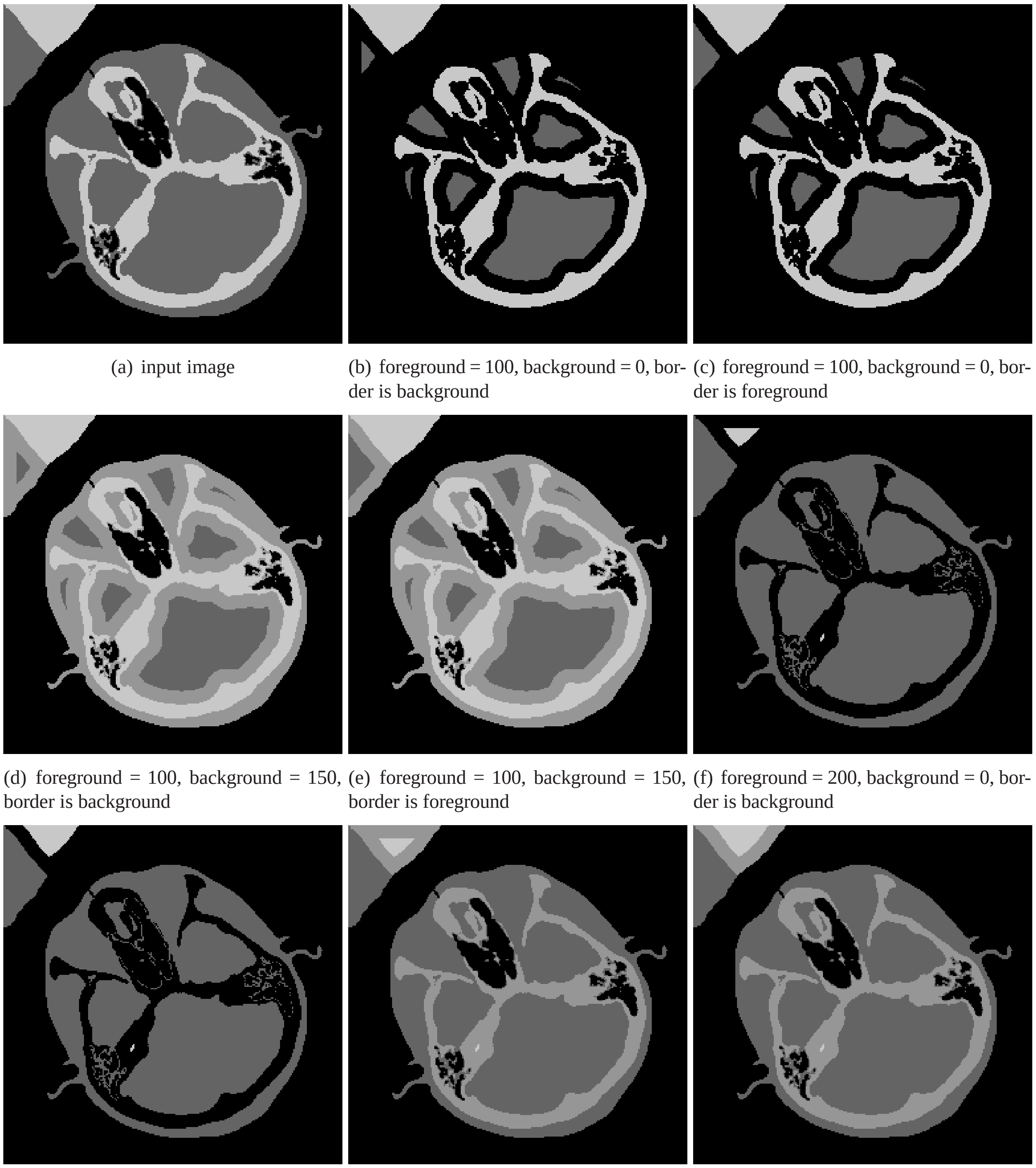

(i) foreground = 200, background = 150, border is foreground

Figure 1: Dilation with a binary ball structuring element of radius 10.



(a) input image

(b) foreground = 100, background = 0, border is background

(c) foreground = 100, background = 0, border is foreground

(d) foreground = 100, background = 150, border is background

(e) foreground = 100, background = 150, border is foreground

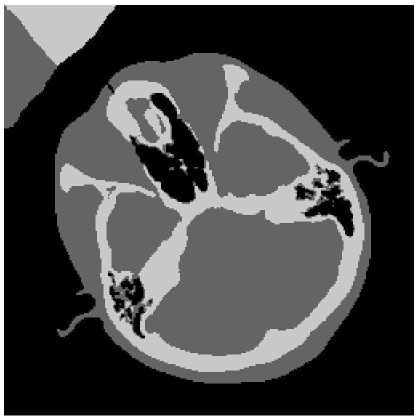
(f) foreground = 200, background = 0, border is background

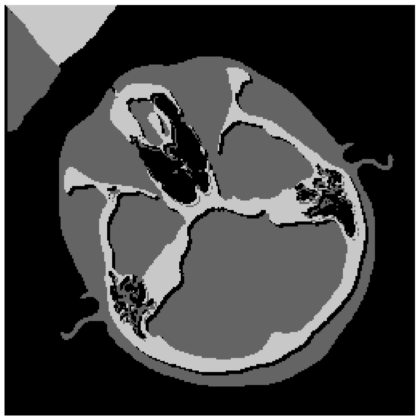
(g) foreground = 200, background = 0, border is foreground

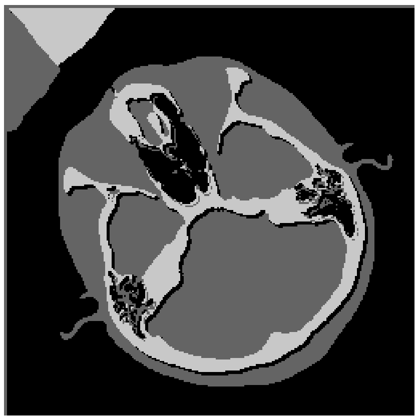
(h) foreground = 200, background = 150, border is background

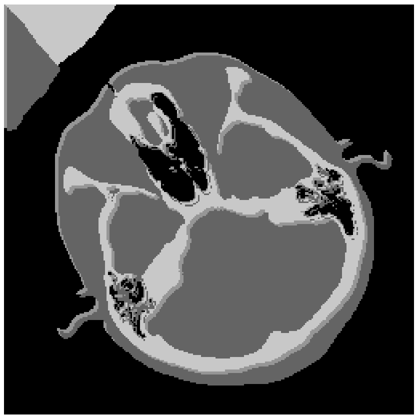
(i) foreground = 200, background = 150, border is foreground

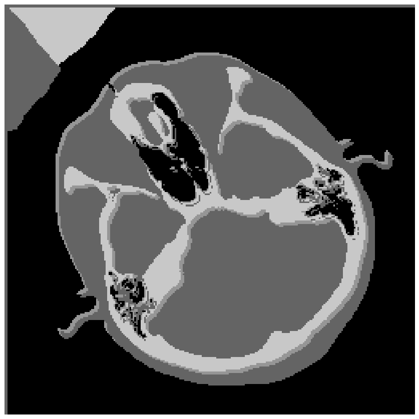
Figure 2: Erosion with a binary ball structuring element of radius 10.

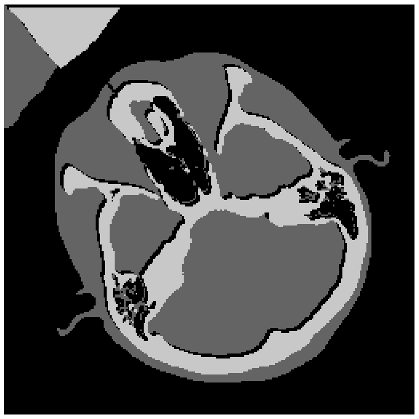

(a) input image

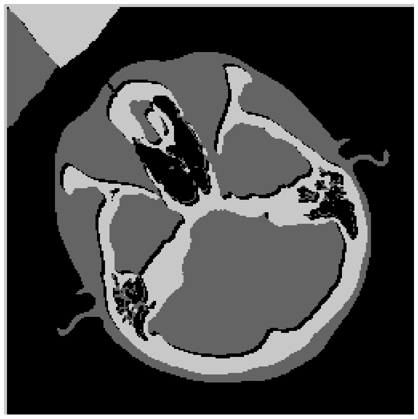

(b) foreground = 100, background = 0, border is background

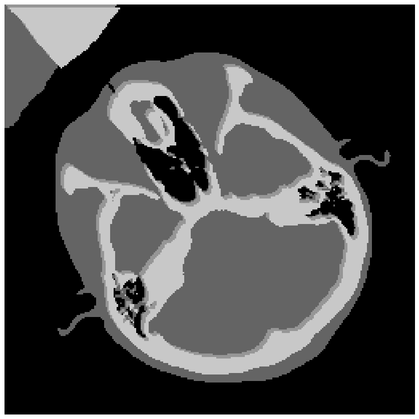

(c) foreground = 100, background = 0, border is foreground


(d) foreground = 100, background = 150, border is background


(e) foreground = 100, background = 150, border is foreground


(f) foreground = 200, background = 0, border is background


(g) foreground = 200, background = 0, border is foreground


(h) foreground = 200, background = 150, border is background

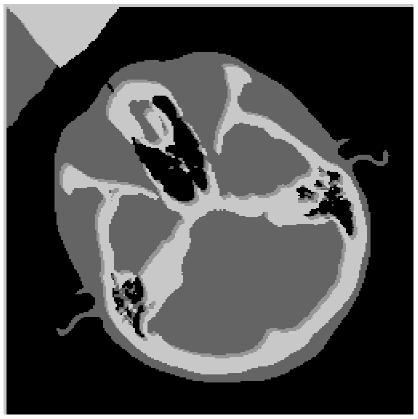

(i) foreground = 200, background = 150, border is foreground

Figure 3: Dilation and erosion with a structuring element with only one point.

```
  typedef itk::ImageFileWriter< IType > WriterType;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput( filter->GetOutput() );
  writer->SetFileName( argv[5] );

  writer->Update();

  return 0;
}
```

5 Conclusion

The new proposed filters outperform the old ones, and provide more features. However, it is surely possible to improve the performance of the new filters. The creation of the output image still use lots of time. An iterator which put only the neighbors in a buffer may be a great help in that case.

6 Acknowledgments

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

I also thank Jerome SCHMID for his explanations, and for his first work on those filters.

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.

[2] N. Nikopoulos. An efficient algorithm for 3d binary morphological transformations with 3d structuring elements or arbitrary size and shape. *IEEE Transactions on Image Processing*, 9(3):283–286, 2000. (document)