# Consolidated morphology

Gaëtan Lehmann[1] and Richard Beare[2]

September 17, 2006

[1]INRA, UMR 1198; ENVA; CNRS, FRE 2857, Biologie du Développement et Reproduction, Jouy en Josas, F-78350, France
[2]Department of Medicine, Monash University, Australia

**Abstract**

Grayscale dilation and erosion are basic transformations of mathematical morphology. Used together or with other transformations, they are very useful tools for image analysis. However, they can be very time consuming, especially with 3D images, and with large structuring elements. Several algorithms have been created to decrease the computation time, some of them with some limitations of shape of structuring element. We have implemented several algorithms, studied their performance in different conditions, and shown that all of them are more efficients than the others in certain conditions.

We finally introduce a new structuring element class and a some meta filter designed to select the best algorithm depending on the image and the structuring element, and to smoothely integrate the different algorithms available in the toolkit.

*This article is an early version made to describe the FlatStructuringElement class. The other parts will come later. Be aware that other classes than FlatStructuringElement are still in development and may produce incorrect results.*

## Contents

## 1 Introduction

Several classes can currently be used as structuring element in ITK: `itk::Neighborhood` and some subclass of `itk::Neighborhood`. There is several problems with that:

- Some class like `itk::BinaryBallStructuringElement` contains some code to generate a structuring element of particular shape (a ball in that case), but the required call to the method `CreateStructuringElement()` make it particulary confusing for the beginner
- `TPixel`, the first template parameter of the `itk::Neighborhood` class and its subclass is meaning less in that case. It can be safely set to a different type than the pixel type of the image, and store only a binary information: inside the neighborhood or outside the neighborhood.
- ITK doesn't offer convenient code to visualize the structuring element, or generate a structuring element from an image

The `itk::FlatStructuringElement` class provided with this article aims to solve those problems, and, with the meta filters, to make the integration of the van Herk / Gil Werman and Anchor algorithms as transparent as possible.

# 2 Algorithms

## 2.1 Basic algorithm

## 2.2 Moving histogram algorithm

## 2.3 van Herk / Gil Werman algorithm

## 2.4 Anchor algorithm

# 3 Performance

# 4 Implementation

## 4.1 Basic algorithm

## 4.2 Moving histogram algorithm

## 4.3 van Herk / Gil Werman algorithm

## 4.4 Anchor algorithm

## 4.5 FlatStructuringElement

The FlatStructuringElement is implemented as a subclass of `itk::Neighborhood`, in order to keep backward compatibility. However, only the `VDimension` template parameter has been kept. The meaning less `TPixel` of the `itk::Neighborhood` class is set to `bool` and hidden to the user.

`itk::FlatStructuringElement` provides several static methods. Each of them let the user produce a stucturing element with a given shape:

- `Ball()` generate a ball structuring element. It takes the radius of the structuring element as parameter.
- `Box()` produce a box structuring element. It takes the radius of the structuring element as parameter.
- `FromImage()` produce a structuring element from an image. The image is passed as parameter to the method. A optional parameter can also be passed: the pixel value to be considered as the foreground value in the image. This value defaults to `NumericTraits< PixelType >::max()`.

The number of available shapes is currently limited. However, the most useful ones are there, and the ability to generate a structuring element from an image should let the user easily create the ones he may need. Also, the infrastructure is there and other methods to generate others shape can be easily added.

The `GetImage()` method in `itk::FlatStructuringElement` class let the user generate an image from its structuring element, and thus easily visualize its real shape. `GetImage()` is templated over the image type to produce. By default, the pixels outside the structuring element are set to `NumericTraits< PixelType >::Zero`, and the pixels inside the structuring element to `NumericTraits< PixelType >::max()`.

The method `GetImage()` and `FromImage()` are templated methods. This particularity make them not available when wrapped in `Python`, `Tcl` or `Java`. For this usage, `GetImageUC()` and `FromImageUC()` has been created. They provide the same fonctionality that `GetImage()` and `FromImage()`, but are limited to the pixel type `unsigned char`, so they can be not templated. This methods are only there to be used with wrappers and shouldn't be used in `C++`.

### 4.6 Meta filters

## 5 Usage

## 6 Conclusion

## 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.