Olena Reference Manual

EPITA Reserch and Development Laboratory

FIXME:

This manual documents Olena, a generic image processing library.

Chapter 1

Introduction

This reference manual will eventually document any public class and functions available in Olena. Sadly, it only covers the morphological processing presently.

The `demo/` directory contains a few sample programs that may be worth looking at before digging the source or sending us an email (`olena@lrde.epita.fr`).

Chapter 2

Processings

2.1 Morphological processings

Soille refers to *P. Soille, morphological Image Analysis – Principals and Applications*. Springer 1998.

2.1.1 morpho::beucher_gradient

Purpose

Morphological Beucher Gradient.

Prototype

```
#include <oln/morpho/gradient.hh>

mute<I, typename convoutput<C, Value(I)>::ret>::ret
morpho::beucher_gradient (const conversion<C>& c,
const image<I>& input, const struct_elt<E>& se);

mute<I, typename convoutput<C, Value(I)>::ret>::ret
morpho::fast::beucher_gradient (const conversion<C>& c,
const image<I>& input, const struct_elt<E>& se);

Concrete(I) morpho::beucher_gradient
(const image<I>& input, const struct_elt<E>& se);

Concrete(I) morpho::fast::beucher_gradient
(const image<I>& input, const struct_elt<E>& se);
```

Parameters

c	IN	conversion object
input	IN	input image
se	IN	structural element

Description

Compute the arithmetic difference between the diltation and the erosion of *input* using *se* as structural element. Soille, p67.

See also

morpho::erosion, §2.1.5, p.9,
morpho::dilation, §2.1.4, p.8,
morpho::external_gradient, §2.1.6, p.10,
morpho::internal_gradient, §2.1.18, p.21.

Example

```
image2d<int_u8> im = load("lena256.pgm");
save(morpho::beucher_gradient(im, win_c8p()), "out.pgm");
```

Figure 2.1: lena256.pgm

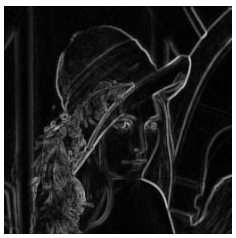

Figure 2.2: out.pgm

2.1.2 morpho::black_top_hat

Purpose

Black top hat.

Prototype

```
#include <oln/morpho/top_hat.hh>

typename mute<I, typename convoutput<C,
Value(I)>::ret>::ret
morpho::black_top_hat (const conversion<C>& c,
const image<I>& input, const struct_elt<E>& se);

typename mute<I, typename convoutput<C,
Value(I)>::ret>::ret
morpho::fast::black_top_hat (const conversion<C>& c,
const image<I>& input, const struct_elt<E>& se);
```

Parameters

c	IN	conversion object
input	IN	input image
se	IN	structural element

Description

Compute black top hat of *input* using *se* as structural element. Soille p.105.

See also
morpho::closing, §2.1.3, p.7.

Example
```
image2d<int_u8> im = load("lena256.pgm");
save(morpho::black_top_hat(im, win_c8p()), "out.pgm");
```

Figure 2.3: lena256.pgm

Figure 2.4: out.pgm

2.1.3 morpho::closing

Purpose
Morphological closing.

Prototype
```
#include <oln/morpho/closing.hh>

Concrete(I) morpho::closing (const image<I>& input,
const struct_elt<E>& se);

Concrete(I) morpho::fast::closing (const image<I>& input,
const struct_elt<E>& se);
```

Parameters

input	IN	input image
se	IN	structural element

Description
Compute the morphological closing of *input* using *se* as structural element.

See also
morpho::erosion, §2.1.5, p.9,
morpho::dilation, §2.1.4, p.8,
morpho::closing, §2.1.3, p.7.

Example
```
image2d<bin> im = load("object.pbm");
save(morpho::closing(im, win_c8p()), "out.pbm");
```

Figure 2.5: object.pbm

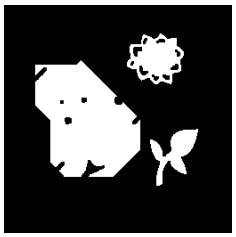

Figure 2.6: out.pbm

2.1.4 morpho::dilation

Purpose

Morphological dilation.

Prototype

```
#include <oln/morpho/dilation.hh>

Concrete(I) morpho::dilation (const image<I>& input ,
const struct_elt<E>& se );

Concrete(I) morpho::fast::dilation (const image<I>& input ,
const struct_elt<E>& se );
```

Parameters

input	IN	input image
se	IN	structural element

Description

Compute the morphological dilation of *input* using *se* as structural element.

On grey-scale images, each point is replaced by the maximum value of its neighbors, as indicated by *se*. On binary images, a logical `or` is performed between neighbors.

The `morpho::fast` version of this function use a different

See also

morpho::n_dilation, §2.1.20, p.23,
morpho::erosion, §2.1.5, p.9.

Example

```
image2d<bin> im = load("object.pbm");
save(morpho::dilation(im, win_c8p()), "out.pbm");
```

Figure 2.7: object.pbm

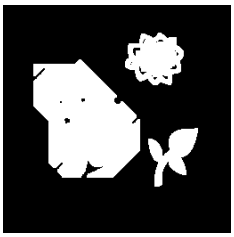

Figure 2.8: out.pbm

2.1.5 morpho::erosion

Purpose

Morphological erosion.

Prototype

```
#include <oln/morpho/erosion.hh>

Concrete(I) morpho::erosion (const image<I>& input,
const struct_elt<E>& se);

Concrete(I) morpho::fast::erosion (const image<I>& input,
const struct_elt<E>& se);
```

Parameters

input	IN	input image
se	IN	structural element

Description

Compute the morphological erosion of *input* using *se* as structural element.

On grey-scale images, each point is replaced by the minimum value of its neighbors, as indicated by *se*. On binary images, a logical `and` is performed between neighbors. The `morpho::fast` version of this function use a different

See also

morpho::n_erosion, §2.1.21, p.23,
morpho::dilation, §2.1.4, p.8.

Example

```
image2d<bin> im = load("object.pbm");
save(morpho::erosion(im, win_c8p()), "out.pbm");
```

Figure 2.9: object.pbm

Figure 2.10: out.pbm

2.1.6 morpho::external_gradient

Purpose

Morphological External Gradient.

Prototype

```
#include <oln/morpho/gradient.hh>

mute<I, typename convoutput<C, Value(I)>::ret>::ret
morpho::external_gradient (const conversion<C>& c,
const image<I>& input, const struct_elt<E>& se);

mute<I, typename convoutput<C, Value(I)>::ret>::ret
morpho::fast::external_gradient (const conversion<C>& c,
const image<I>& input, const struct_elt<E>& se);

Concrete(I) morpho::external_gradient
(const image<I>& input, const struct_elt<E>& se);

Concrete(I) morpho::fast::external_gradient
(const image<I>& input, const struct_elt<E>& se);
```

Parameters

c	IN	conversion object
input	IN	input image
se	IN	structural element

Description

Compute the arithmetic difference between and the dilatation of *input* using *se* as structural element, and the original image *input*. Soille, p67.

See also

morpho::beucher_gradient, §2.1.1, p.5,
morpho::internal_gradient, §2.1.18, p.21,
morpho::dilation, §2.1.4, p.8.

Example

```
image2d<int_u8> im = load("lena256.pgm");
save(morpho::external_gradient(im, win_c8p()), "out.pgm");
```

Figure 2.11: lena256.pgm

Figure 2.12: out.pgm

2.1.7 morpho::geodesic_dilation

Purpose

Geodesic dilation.

Prototype

```
#include <oln/morpho/geodesic_dilation.hh>

Concrete(I1) morpho::geodesic_dilation
(const image<I1>& marker, const image<I2>& mask,
const struct_elt<E>& se);
```

Parameters

marker	IN	marker image
mask	IN	mask image
se	IN	structural element

Description

Compute the geodesic dilation of *marker* with respect to the mask *mask* image using *se* as structural element. Soille p.156. Note mask must be greater or equal than marker.

See also

morpho::simple_geodesic_dilation, §2.1.28, p.29.

Example

```
image2d<int_u8> light = load("light.pgm");
image2d<int_u8> dark = load("dark.pgm");
save(morpho::geodesic_dilation(dark, light, win_c8p()), "out.pgm");
```

2.1.8 morpho::geodesic_erosion

Purpose

Geodesic erosion.

Prototype

```
#include <oln/morpho/geodesic_erosion.hh>
```

```
Concrete(I1) morpho::geodesic_erosion
(const image<I1>& marker, const image<I2>& mask,
const struct_elt<E>& se);
```

Parameters

marker	IN	marker image
mask	IN	mask image
se	IN	structural element

Description

Compute the geodesic erosion of *marker* with respect to the mask *mask* image using *se* as structural element. Soille p.158. Note marker must be greater or equal than mask.

See also

morpho::simple_geodesic_dilation, §2.1.28, p.29.

Example

```
image2d<int_u8> light = load("light.pgm");
image2d<int_u8> dark = load("dark.pgm");
save(morpho::geodesic_erosion(light, dark, win_c8p()), "out.pgm");
```

2.1.9 morpho::hit_or_miss

Purpose

Hit_or_Miss Transform.

Prototype

```
#include <oln/morpho/hit_or_miss.hh>

typename mute<_I, typename convoutput<C,
Value(_I)>::ret>::ret
morpho::hit_or_miss (const conversion<C>& c,
const image<I>& input, const struct_elt<E>& se1,
const struct_elt<E>& se2);

typename mute<_I, typename convoutput<C,
Value(_I)>::ret>::ret
morpho::fast::hit_or_miss (const conversion<C>& c,
const image<I>& input, const struct_elt<E>& se1,
const struct_elt<E>& se2);

Concrete(I) morpho::hit_or_miss (const image<I>& input,
const struct_elt<E>& se1, const struct_elt<E>& se2);

Concrete(I) morpho::fast::hit_or_miss (const image<I>& input,
const struct_elt<E>& se1, const struct_elt<E>& se2);
```

Parameters

c	IN	conversion object
input	IN	input image
se1	IN	structural element
se2	IN	structural element

Description

Compute the hit_or_miss transform of *input* by the composite structural element (*se1*, *se2*). Soille p.131.

By definition *se1* and *se2* must have the same origin, and need to be disjoint. This algorithm has been extended to every data types (althought it is not increasing). Beware the result depends upon the image data type if it is not `bin`.

Example

```
image2d<bin> im = load("object.pbm");
window2d mywin;
mywin
 .add(-3,-2).add(-3,-1).add(-3,0).add(-3,1).add(-3,2)
 .add(-2,-1).add(-2,0).add(-2,1)
 .add(-1,0);
window2d mywin2 = - mywin;
save(morpho::fast::hit_or_miss(convert::bound<int_u8>(),
                               im, mywin, mywin2), "out.pgm");
```

Figure 2.13: object.pbm

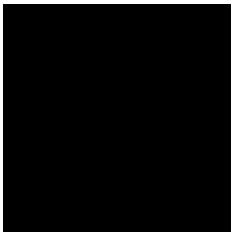

Figure 2.14: out.pgm

2.1.10 morpho::hit_or_miss_closing

Purpose

Hit_or_Miss closing.

Prototype

`#include <oln/`*morpho/hit_or_miss.hh*`>`

```
Concrete(I) morpho::hit_or_miss_closing
(const image<I>& input, const struct_elt<E>& se1,
const struct_elt<E>& se2);

Concrete(I) morpho::fast::hit_or_miss_closing
(const image<I>& input, const struct_elt<E>& se1,
const struct_elt<E>& se2);
```

Parameters

input	IN	input image
se1	IN	structural element
se2	IN	structural element

Description

Compute the hit_or_miss closing of *input* by the composite structural element (*se1*, *se2*). This is the dual transformation of hit-or-miss opening with respect to set complementation. Soille p.135.

By definition *se1* and *se2* must have the same origin, and need to be disjoint. This algorithm has been extended to every data types (althought it is not increasing). Beware the result depends upon the image data type if it is not `bin`.

See also

morpho::hit_or_miss, §2.1.9, p.12,
morpho::hit_or_miss_closing_bg, §2.1.11, p.14,
morpho::hit_or_miss_opening, §2.1.12, p.16,
morpho::hit_or_miss_opening_bg, §2.1.13, p.17.

Example

```
 image2d<bin> im = load("object.pbm");
window2d mywin;
mywin
.add(-3,-2).add(-3,-1).add(-3,0).add(-3,1).add(-3,2)
.add(-2,-1).add(-2,0).add(-2,1)
.add(-1,0);
window2d mywin2 = - mywin;
 save(morpho::hit_or_miss_closing(im, mywin, mywin2), "out.pbm");
```

2.1.11 morpho::hit_or_miss_closing_bg

Purpose

Hit_or_Miss closing of background.

Prototype

`#include <oln/`*morpho/hit_or_miss.hh*`>`

Figure 2.15: object.pbm

Figure 2.16: out.pbm

```
Concrete(I) morpho::hit_or_miss_closing_bg
(const image<I>& input, const struct_elt<E>& se1,
const struct_elt<E>& se2);
```

```
Concrete(I) morpho::fast::hit_or_miss_closing_bg
(const image<I>& input, const struct_elt<E>& se1,
const struct_elt<E>& se2);
```

Parameters

input	IN	input image
se1	IN	structural element
se2	IN	structural element

Description

Compute the hit_or_miss closing of the background of *input* by the composite structural element (*se1*, *se2*). This is the dual transformation of hit-or-miss opening with respect to set complementation. Soille p.135.

By definition *se1* and *se2* must have the same origin, and need to be disjoint. This algorithm has been extended to every data types (althought it is not increasing). Beware the result depends upon the image data type if it is not `bin`.

See also

morpho::hit_or_miss, §2.1.9, p.12,
morpho::hit_or_miss_closing, §2.1.10, p.13,
morpho::hit_or_miss_opening, §2.1.12, p.16,
morpho::hit_or_miss_opening_bg, §2.1.13, p.17.

Example

```
 image2d<bin> im = load("object.pbm");
window2d mywin;
mywin
.add(-3,-2).add(-3,-1).add(-3,0).add(-3,1).add(-3,2)
.add(-2,-1).add(-2,0).add(-2,1)
.add(-1,0);
window2d mywin2 = - mywin;
```

```
save(morpho::hit_or_miss_closing_bg(im, mywin, mywin2), "out.pbm");
```

Figure 2.17: object.pbm

Figure 2.18: out.pbm

2.1.12 morpho::hit_or_miss_opening

Purpose

Hit_or_Miss opening.

Prototype

`#include <oln/`*morpho/hit_or_miss.hh*`>`

`Concrete(I)` **morpho::hit_or_miss_opening**
`(const image<I>&` *input*`, const struct_elt<E>&` *se1*`,`
`const struct_elt<E>&` *se2*`);`

`Concrete(I)` **morpho::fast::hit_or_miss_opening**
`(const image<I>&` *input*`, const struct_elt<E>&` *se1*`,`
`const struct_elt<E>&` *se2*`);`

Parameters

input	IN	input image
se1	IN	structural element
se2	IN	structural element

Description

Compute the hit_or_miss opening of *input* by the composite structural element (*se1*, *se2*). Soille p.134.

By definition *se1* and *se2* must have the same origin, and need to be disjoint. This algorithm has been extended to every data types (althought it is not increasing). Beware the result depends upon the image data type if it is not `bin`.

See also

morpho::hit_or_miss, §2.1.9, p.12,
morpho::hit_or_miss_closing, §2.1.10, p.13,
morpho::hit_or_miss_closing_bg, §2.1.11, p.14,
morpho::hit_or_miss_opening_bg, §2.1.13, p.17.

Example

```
 image2d<bin> im = load("object.pbm");
window2d mywin;
mywin
.add(-3,-2).add(-3,-1).add(-3,0).add(-3,1).add(-3,2)
.add(-2,-1).add(-2,0).add(-2,1)
.add(-1,0);
window2d mywin2 = - mywin;
 save(morpho::hit_or_miss_opening(im, mywin, mywin2), "out.pbm");
```

Figure 2.19: object.pbm

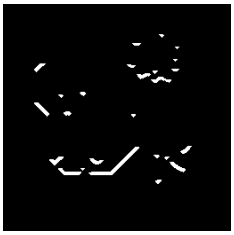

Figure 2.20: out.pbm

2.1.13 morpho::hit_or_miss_opening_bg

Purpose

Hit_or_Miss opening of background.

Prototype

```
#include <oln/morpho/hit_or_miss.hh>

Concrete(I) morpho::hit_or_miss_opening_bg
(const image<I>& input, const struct_elt<E>& se1,
const struct_elt<E>& se2);

Concrete(I) morpho::fast::hit_or_miss_opening_bg
(const image<I>& input, const struct_elt<E>& se1,
const struct_elt<E>& se2);
```

Parameters

input	IN	input image
se1	IN	structural element
se2	IN	structural element

Description

Compute the hit_or_miss opening of the background of *input* by the composite structural element (*se1*, *se2*). Soille p.135.

By definition *se1* and *se2* must have the same origin, and need to be disjoint. This algorithm has been extended to every data types (althought it is not increasing). Beware the result depends upon the image data type if it is not `bin`.

See also

morpho::hit_or_miss, §2.1.9, p.12,
morpho::hit_or_miss_closing, §2.1.10, p.13,
morpho::hit_or_miss_closing_bg, §2.1.11, p.14,
morpho::hit_or_miss_opening, §2.1.12, p.16.

Example

```
 image2d<bin> im = load("object.pbm");
window2d mywin;
mywin
.add(-3,-2).add(-3,-1).add(-3,0).add(-3,1).add(-3,2)
.add(-2,-1).add(-2,0).add(-2,1)
.add(-1,0);
window2d mywin2 = - mywin;
 save(morpho::hit_or_miss_opening_bg(im, mywin, mywin2), "out.pbm");
```

Figure 2.21: object.pbm

Figure 2.22: out.pbm

2.1.14 morpho::hybrid_geodesic_reconstruction_dilation

Purpose

Geodesic reconstruction by dilation.

Prototype

`#include <oln/`*morpho/reconstruction.hh*`>`

`Concrete(I1)` **morpho::hybrid_geodesic_reconstruction_dilation**
`(const image<I1>&` *marker*`, const image<I2>&` *mask*`,`
`const struct_elt<E>&` *se*`);`

Parameters

marker	IN	marker image
mask	IN	mask image
se	IN	structural element

Description

Compute the reconstruction by dilation of *marker* with respect to the mask *mask* image using *se* as structural element. Soille p.160. The algorithm used is the one defined as hybrid in Vincent(1993), Morphological grayscale reconstruction in image analysis: applications and efficient algorithms, itip, 2(2), 176–201.

See also

morpho::simple_geodesic_dilation, §2.1.28, p.29.

Example

```
image2d<int_u8> light = load("light.pgm");
image2d<int_u8> dark = load("dark.pgm");
save(morpho::hybrid_geodesic_reconstruction_dilation(light, dark, win_c8p()), "out.pgm");
```

2.1.15 morpho::hybrid_geodesic_reconstruction_erosion

Purpose

Geodesic reconstruction by erosion.

Prototype

```
#include <oln/morpho/reconstruction.hh>

Concrete(I1) morpho::hybrid_geodesic_reconstruction_erosion
(const image<I1>& marker, const image<I2>& mask,
const struct_elt<E>& se);
```

Parameters

marker	IN	marker image
mask	IN	mask image
se	IN	structural element

Description

Compute the reconstruction by erosion of *marker* with respect to the mask *mask* image using *se* as structural element. Soille p.160. The algorithm used is the one defined as hybrid in Vincent(1993), Morphological grayscale reconstruction in image analysis: applications and efficient algorithms, itip, 2(2), 176–201.

See also

morpho::simple_geodesic_erosion, §2.1.29, p.29.

Example

```
image2d<int_u8> light = load("light.pgm");
image2d<int_u8> dark = load("dark.pgm");
save(morpho::sequential_geodesic_reconstruction_erosion(light, dark, win_c8p()),
```

2.1.16 morpho::hybrid_minima_imposition

Purpose

Minima Imposition.

Prototype

```
#include <oln/morpho/extrema.hh>

Concrete(_I) morpho::hybrid_minima_imposition
(const image<I1>& input, const image<I2>& minima_map,
const struct_elt<E>& se);
```

Parameters

input	IN	input image
minima_map	IN	bin image
se	IN	structural element

Description

Impose minima defined by *minima_map* on *input* using *se* as structural element. Soille p.172. *minima_map* must be a bin image (true for a minimum, false for a non minimum). The algorithm uses hybrid_geodesic_reconstruction_erosion.

See also

morpho::hybrid_geodesic_reconstruction_erosion, §2.1.15, p.19.

Example

```
image2d<int_u8> light = load("light.pgm");
image2d<bin> minima = load("minima.pbm");
save(morpho::hybrid_minima_imposition(light, minima, win_c8p()), "out.pgm");
```

2.1.17 morpho::hybrid_regional_minima

Purpose

Regional minima.

Prototype

```
#include <oln/morpho/extrema.hh>

Concrete(_I) morpho::hybrid_regional_minima
(const image<I1>& input, const struct_elt<E>& se);
```

Parameters

input	IN	input image
se	IN	structural element

Description

Extract regional minima of *input* using *se* as structural element. Soille p.169. The algorithm uses hybrid_geodesic_reconstruction_erosion.

See also

morpho::hybrid_geodesic_reconstruction_erosion, §2.1.15, p.19.

Example

```
image2d<int_u8> light = load("light.pgm");
save(morpho::hybrid_minima_imposition(light, win_c8p()), "out.pgm");
```

2.1.18 morpho::internal_gradient

Purpose

Morphological Internal Gradient.

Prototype

```
#include <oln/morpho/gradient.hh>

mute<I, typename convoutput<C, Value(I)>::ret>::ret
morpho::internal_gradient (const conversion<C>& c,
const image<I>& input, const struct_elt<E>& se);

mute<I, typename convoutput<C, Value(I)>::ret>::ret
morpho::fast::internal_gradient (const conversion<C>& c,
const image<I>& input, const struct_elt<E>& se);

Concrete(I) morpho::internal_gradient
(const image<I>& input, const struct_elt<E>& se);

Concrete(I) morpho::fast::internal_gradient
(const image<I>& input, const struct_elt<E>& se);
```

Parameters

c	IN	conversion object
input	IN	input image
se	IN	structural element

Description

Compute the arithmetic difference between the original image *input* and the erosion of *input* using *se* as structural element. Soille, p67.

See also

morpho::beucher_gradient, §2.1.1, p.5,
morpho::external_gradient, §2.1.6, p.10,
morpho::erosion, §2.1.5, p.9.

Example

```
image2d<int_u8> im = load("lena256.pgm");
save(morpho::internal_gradient(im, win_c8p()), "out.pgm");
```

Figure 2.23: lena256.pgm

Figure 2.24: out.pgm

2.1.19 morpho::laplacian

Purpose

Laplacian.

Prototype

```
#include <oln/morpho/laplacian.hh>

typename mute<I, typename convoutput<C,
Value(I)>::ret>::ret
morpho::laplacian (const conversion<C>& c,
const image<I>& input, const struct_elt<E>& se);

typename mute<I, typename convoutput<C,
Value(I)>::ret>::ret
morpho::fast::laplacian (const conversion<C>& c,
const image<I>& input, const struct_elt<E>& se);

typename mute<I, Value(I)::slarger_t>::ret
morpho::laplacian (const image<I>& input,
const struct_elt<E>& se);

typename mute<I, Value(I)::slarger_t>::ret
morpho::fast::laplacian (const image<I>& input,
const struct_elt<E>& se);
```

Parameters

c	IN	conversion object
input	IN	input image
se	IN	structural element

Description

Compute the laplacian of *input* using *se* as structural element.

See also

morpho::dilation, §2.1.4, p.8,
morpho::erosion, §2.1.5, p.9.

Example

```
image2d<int_u8> im = load("lena256.pgm");
save(morpho::laplacian(convert::bound<int_u8>(), im, win_c8p()), "out.pgm");
```

Figure 2.25: lena256.pgm

Figure 2.26: out.pgm

2.1.20 morpho::n_dilation

Purpose

Morphological dilation itered n times.

Prototype

```
#include <oln/morpho/dilation.hh>

Concrete(I) morpho::n_dilation (const image<I>& input,
const struct_elt<E>& se, unsigned n);
```

Parameters

input	IN	input image
se	IN	structural element
n	IN	number of iterations

Description

Apply `morpho::dilation` n times.

See also

morpho::dilation, §2.1.4, p.8,
morpho::n_erosion, §2.1.21, p.23.

2.1.21 morpho::n_erosion

Purpose

Morphological erosion itered n times.

Prototype

```
#include <oln/morpho/erosion.hh>

Concrete(I) morpho::n_erosion (const image<I>& input,
const struct_elt<E>& se, unsigned n);
```

Parameters

input	IN	input image
se	IN	structural element
n	IN	number of iterations

Description

Apply `morpho::erosion` n times.

See also

morpho::erosion, §2.1.5, p.9,
morpho::n_dilation, §2.1.20, p.23.

2.1.22 morpho::opening

Purpose

Morphological opening.

Prototype

```
#include <oln/morpho/opening.hh>

Concrete(I) morpho::opening (const image<I>& input,
const struct_elt<E>& se);

Concrete(I) morpho::fast::opening (const image<I>& input,
const struct_elt<E>& se);
```

Parameters

input	IN	input image
se	IN	structural element

Description

Compute the morphological opening of *input* using *se* as structural element.

See also

morpho::erosion, §2.1.5, p.9,
morpho::dilation, §2.1.4, p.8,
morpho::closing, §2.1.3, p.7.

Example

```
image2d<bin> im = load("object.pbm");
save(morpho::opening(im, win_c8p()), "out.pbm");
```

Figure 2.27: object.pbm

Figure 2.28: out.pbm

2.1.23 morpho::self_complementary_top_hat

Purpose

Self complementary top hat.

Prototype

```
#include <oln/morpho/top_hat.hh>

typename mute<I, typename convoutput<C,
Value(I)>::ret>::ret
morpho::self_complementary_top_hat (const conversion<C>& c,
const image<I>& input, const struct_elt<E>& se);

typename mute<I, typename convoutput<C,
Value(I)>::ret>::ret
morpho::fast::self_complementary_top_hat
(const conversion<C>& c, const image<I>& input,
const struct_elt<E>& se);

typename mute<I, typename convoutput<C,
Value(I)>::ret>::ret
morpho::self_complementary_top_hat (const image<I>& input,
const struct_elt<E>& se);

typename mute<I, typename convoutput<C,
Value(I)>::ret>::ret
morpho::fast::self_complementary_top_hat
(const image<I>& input, const struct_elt<E>& se);
```

Parameters

c	IN	conversion object
input	IN	input image
se	IN	structural element

Description

Compute self complementary top hat of *input* using *se* as structural element. Soille p.106.

See also

morpho::closing, §2.1.3, p.7,
morpho::opening, §2.1.22, p.24.

Example

```
image2d<int_u8> im = load("lena256.pgm");
save(morpho::self_complementary_top_hat(im, win_c8p()), "out.pgm");
```

Figure 2.29: lena256.pgm

Figure 2.30: out.pgm

2.1.24 morpho::sequential_geodesic_reconstruction_dilation

Purpose

Geodesic reconstruction by dilation.

Prototype

```
#include <oln/morpho/reconstruction.hh>

Concrete(I1)
morpho::sequential_geodesic_reconstruction_dilation
(const image<I1>& marker, const image<I2>& mask,
const struct_elt<E>& se);
```

Parameters

marker	IN	marker image
mask	IN	mask image
se	IN	structural element

Description

Compute the reconstruction by dilation of *marker* with respect to the mask *mask* image using *se* as structural element. Soille p.160. The algorithm used is the one defined as sequential in Vincent(1993), Morphological grayscale reconstruction in image analysis: applications and efficient algorithms, itip, 2(2), 176–201.

See also

morpho::simple_geodesic_dilation, §2.1.28, p.29.

Example

```
image2d<int_u8> light = load("light.pgm");
image2d<int_u8> dark = load("dark.pgm");
save(morpho::sequential_geodesic_reconstruction_dilation(light, dark, win_c8p()), "out.pgm"
```

2.1.25 morpho::sequential_geodesic_reconstruction_erosion

Purpose

Geodesic reconstruction by erosion.

Prototype

```
#include <oln/morpho/reconstruction.hh>

Concrete(I1)
morpho::sequential_geodesic_reconstruction_erosion
(const image<I1>& marker, const image<I2>& mask,
const struct_elt<E>& se);
```

Parameters

marker	IN	marker image
mask	IN	mask image
se	IN	structural element

Description

Compute the reconstruction by erosion of *marker* with respect to the mask *mask* image using *se* as structural element. Soille p.160. The algorithm used is the one defined as sequential in Vincent(1993), Morphological grayscale reconstruction in image analysis: applications and efficient algorithms, itip, 2(2), 176–201.

See also

morpho::simple_geodesic_erosion, §2.1.29, p.29.

Example

```
image2d<int_u8> light = load("light.pgm");
image2d<int_u8> dark = load("dark.pgm");
save(morpho::sequential_geodesic_reconstruction_erosion(light, dark, win_c8p()), "out.pgm")
```

2.1.26 morpho::sequential_minima_imposition

Purpose

Minima Imposition.

Prototype

```
#include <oln/morpho/extrema.hh>
```

```
Concrete(_I) morpho::sequential_minima_imposition
(const image<I1>& input, const image<I2>& minima_map,
const struct_elt<E>& se);
```

Parameters

input	IN	input image
minima_map	IN	bin image
se	IN	structural element

Description

Impose minima defined by *minima_map* on *input* using *se* as structural element. Soille p.172. The algorithm uses sequential_geodesic_reconstruction_erosion. *minima_map* must be a bin image (true for a minimum, false for a non minimum).

See also

morpho::sequential_geodesic_reconstruction_erosion, §2.1.25, p.27.

Example

```
image2d<int_u8> light = load("light.pgm");
image2d<bin> minima = load("minima.pbm");
save(morpho::sequential_minima_imposition(light, minima, win_c8p()), "out.pgm");
```

2.1.27 morpho::sequential_regional_minima

Purpose

Regional minima.

Prototype

```
#include <oln/morpho/extrema.hh>

Concrete(_I) morpho::sequential_regional_minima
(const image<I1>& input, const struct_elt<E>& se);
```

Parameters

input	IN	input image
se	IN	structural element

Description

Extract regional minima of *input* using *se* as structural element. Soille p.169. The algorithm uses sequential_geodesic_reconstruction_erosion.

See also

morpho::sequential_geodesic_reconstruction_erosion, §2.1.25, p.27.

Example

```
image2d<int_u8> light = load("light.pgm");
save(morpho::sequential_minima_imposition(light, win_c8p()), "out.pgm");
```

2.1.28 morpho::simple_geodesic_dilation

Purpose

Geodesic dilation.

Prototype

```
#include <oln/morpho/geodesic_dilation.hh>

Concrete(I1) morpho::simple_geodesic_dilation
(const image<I1>& marker, const image<I2>& mask,
const struct_elt<E>& se);
```

Parameters

marker	IN	marker image
mask	IN	mask image
se	IN	structural element

Description

Compute the geodesic dilation of *marker* with respect to the mask *mask* image using *se* as structural element. Soille p.156. Computation is performed by hand (i.e without calling dilation). Note mask must be greater or equal than marker.

See also

morpho::simple_geodesic_dilation, §2.1.28, p.29.

Example

```
image2d<int_u8> light = load("light.pgm");
image2d<int_u8> dark = load("dark.pgm");
save(morpho::simple_geodesic_dilation(dark, light,
                                      win_c8p()), "out.pgm");
```

2.1.29 morpho::simple_geodesic_erosion

Purpose

Geodesic erosion.

Prototype

```
#include <oln/morpho/geodesic_erosion.hh>

Concrete(I1) morpho::simple_geodesic_erosion
(const image<I1>& marker, const image<I2>& mask,
const struct_elt<E>& se);
```

Parameters

marker	IN	marker image
mask	IN	mask image
se	IN	structural element

Description

Compute the geodesic erosion of *marker* with respect to the mask *mask* image using *se* as structural element. Soille p.156. Computation is performed by hand (i.e without calling dilation). Note marker must be greater or equal than mask.

See also

morpho::simple_geodesic_dilation, §2.1.28, p.29.

Example

```
image2d<int_u8> light = load("light.pgm");
image2d<int_u8> dark = load("dark.pgm");
save(morpho::geodesic_erosion(light, dark, win_c8p()), "out.pgm");
```

2.1.30 morpho::sure_geodesic_reconstruction_dilation

Purpose

Geodesic reconstruction by dilation.

Prototype

`#include <oln/`*morpho/reconstruction.hh*`>`

`Concrete(I1)` **morpho::sure_geodesic_reconstruction_dilation**
`(const image<I1>&` *marker*`, const image<I2>&` *mask*`,`
`const struct_elt<E>&` *se*`);`

Parameters

marker	IN	marker image
mask	IN	mask image
se	IN	structural element

Description

Compute the reconstruction by dilation of *marker* with respect to the mask *mask* image using *se* as structural element. Soille p.160. This is the simplest algorithm: iteration is performed until stability.

See also

morpho::simple_geodesic_dilation, §2.1.28, p.29.

Example

```
image2d<int_u8> light = load("light.pgm");
image2d<int_u8> dark = load("dark.pgm");
save(morpho::sure_geodesic_reconstruction_dilation(light, dark, win_c8p()), "out.
```

2.1.31 morpho::sure_geodesic_reconstruction_erosion

Purpose

Geodesic reconstruction by erosion.

Prototype

```
#include <oln/morpho/reconstruction.hh>

Concrete(I1) morpho::sure_geodesic_reconstruction_erosion
(const image<I1>& marker, const image<I2>& mask,
const struct_elt<E>& se);
```

Parameters

marker	IN	marker image
mask	IN	mask image
se	IN	structural element

Description

Compute the reconstruction by erosion of *marker* with respect to the mask *mask* image using *se* as structural element. Soille p.160. This is the simplest algorithm : iteration is performed until stability.

See also

morpho::simple_geodesic_erosion, §2.1.29, p.29.

Example

```
image2d<int_u8> light = load("light.pgm");
image2d<int_u8> dark = load("dark.pgm");
save(morpho::sure_geodesic_reconstruction_erosion(light, dark, win_c8p()), "out.pgm");
```

2.1.32 morpho::sure_minima_imposition

Purpose

Minima Imposition.

Prototype

```
#include <oln/morpho/extrema.hh>

Concrete(_I) morpho::sure_minima_imposition
(const image<I1>& input, const image<I2>& minima_map,
const struct_elt<E>& se);
```

Parameters

input	IN	input image
minima_map	IN	bin image
se	IN	structural element

Description

Impose minima defined by *minima_map* on *input* using *se* as structural element. Soille p.172. *minima_map* must be a bin image (true for a minimum, false for a non minimum). The algorithm uses sure_geodesic_reconstruction_erosion.

See also

morpho::sure_geodesic_reconstruction_erosion, §2.1.31, p.31.

Example

```
image2d<int_u8> light = load("light.pgm");
image2d<bin> minima = load("minima.pbm");
save(morpho::sure_minima_imposition(light, minima, win_c8p()), "out.pgm");
```

2.1.33 morpho::sure_regional_minima

Purpose

Regional minima.

Prototype

```
#include <oln/morpho/extrema.hh>

Concrete(_I) morpho::sure_regional_minima
(const image<I1>& input, const struct_elt<E>& se);
```

Parameters

input	IN	input image
se	IN	structural element

Description

Extract regional minima of *input* using *se* as structural element. Soille p.169. The algorithm uses sure_geodesic_reconstruction_erosion.

See also

morpho::sure_geodesic_reconstruction_erosion, §2.1.31, p.31.

Example

```
image2d<int_u8> light = load("light.pgm");
save(morpho::sure_minima_imposition(light, win_c8p()), "out.pgm");
```

2.1.34 morpho::top_hat_contrast_op

Purpose

Top hat contrastor operator.

Prototype

```
#include <oln/morpho/top_hat.hh>
```

```
typename mute<I, typename convoutput<C,
Value(I)>::ret>::ret
morpho::top_hat_contrast_op (const conversion<C>& c,
const image<I>& input, const struct_elt<E>& se);

typename mute<I, typename convoutput<C,
Value(I)>::ret>::ret
morpho::fast::top_hat_contrast_op (const conversion<C>& c,
const image<I>& input, const struct_elt<E>& se);

typename mute<I, typename convoutput<C,
Value(I)>::ret>::ret
morpho::top_hat_contrast_op (const image<I>& input,
const struct_elt<E>& se);

typename mute<I, typename convoutput<C,
Value(I)>::ret>::ret
morpho::fast::top_hat_contrast_op (const image<I>& input,
const struct_elt<E>& se);
```

Parameters

c	IN	conversion object
input	IN	input image
se	IN	structural element

Description

Enhance contrast *input* by adding the white top hat, then substracting the black top hat to *input*. Top hats are computed using *se* as structural element. Soille p.109.

See also

morpho::white_top_hat, §2.1.38, p.35,
morpho::black_top_hat, §2.1.2, p.6.

Example

```
 image2d<int_u8> im = load("lena256.pgm");
 save(morpho::top_hat_contrast_op(convert::bound<int_u8>(),
im, win_c8p()), "out.pgm");
```

2.1.35 morpho::watershed_con

Purpose

Connected Watershed.

Prototype

`#include <oln/`*morpho/watershed.hh*`>`

Figure 2.31: lena256.pgm

Figure 2.32: out.pgm

```
typename mute<I, DestValue>::ret
morpho::watershed_contexttt¡class DestValue¿
(const image<I>& im, const neighborhood<N>& ng);
```

Parameters

DestValue		type of output labels
im	IN	image of levels
ng	IN	neighborhood to consider

Description

Compute the connected watershed for image *im* using neighborhood *ng*.

`watershed_con` creates an ouput image whose values have type *DestValue* (which should be discrete). In this output all basins are labeled using values from `DestValue::min()` to `DestValue::max() - 4` (the remaining values are used internally by the algorithm).

When there are more basins than `DestValue` can hold, wrapping occurs (i.e., the same label is used for several basin). This is potentially harmful, because if two connected basins are labeled with the same value they will appear as one basin.

2.1.36 morpho::watershed_seg

Purpose

Segmented Watershed.

Prototype

```
#include <oln/morpho/watershed.hh>

typename mute<I, DestValue>::ret
morpho::watershed_segtexttt¡class DestValue¿
(const image<I>& im, const neighborhood<N>& ng);
```

Parameters

DestValue		type of output labels
im	IN	image of levels
ng	IN	neighborhood to consider

Description

Compute the segmented watershed for image *im* using neighborhood *ng*.

`watershed_seg` creates an ouput image whose values have type *DestValue* (which should be discrete). In this output image, `DestValue::max()` indicates a watershed, and all basins are labeled using values from `DestValue::min()` to `DestValue::max() - 4` (the remaining values are used internally by the algorithm).

When there are more basins than `DestValue` can hold, wrapping occurs (i.e., the same label is used for several basin).

2.1.37 morpho::watershed_seg_or

Purpose

Segmented Watershed with user-supplied starting points.

Prototype

```
#include <oln/morpho/watershed.hh>

Concrete(I2)& morpho::watershed_seg_or
(const image<I1>& levels, image<I2>& markers,
const neighborhood<N>& ng);
```

Parameters

levels	IN		image of levels
markers	IN	OUT	image of markers
ng	IN		neighborhood to consider

Description

Compute a segmented watershed for image *levels* using neighborhood *ng*, and *markers* as starting point for the flooding algorithm.

markers is an image of the same size as *levels* and containing discrete values indicating label associated to each basin. On input, fill *markers* with `Value(I2)::min()` (this is the *unknown* label) and mark the starting points or regions (usually these are minima in *levels*) using a value between `Value(I2)::min()+1` and `Value(I2)::max()-1`.

`watershed_seg_or` will flood *levels* from these non-*unknown* starting points, labeling basins using the value you assigned to them, and markining watershed lines with `Value(I2)::max()`. *markers* should not contains any `Value(I2)::min()` value on output.

2.1.38 morpho::white_top_hat

Purpose

White top hat.

Prototype

```
#include <oln/morpho/top_hat.hh>

typename mute<I, typename convoutput<C,
Value(I)>::ret>::ret
morpho::white_top_hat (const conversion<C>& c,
const image<I>& input, const struct_elt<E>& se);

typename mute<I, typename convoutput<C,
Value(I)>::ret>::ret
morpho::fast::white_top_hat (const conversion<C>& c,
const image<I>& input, const struct_elt<E>& se);

Concrete(I) morpho::white_top_hat (const image<I>& input,
const struct_elt<E>& se);

Concrete(I) morpho::fast::white_top_hat
(const image<I>& input, const struct_elt<E>& se);
```

Parameters

c	IN	conversion object
input	IN	input image
se	IN	structural element

Description

Compute white top hat of *input* using *se* as structural element. Soille p.105.

See also

morpho::opening, §2.1.22, p.24.

Example

```
image2d<int_u8> im = load("lena256.pgm");
save(morpho::white_top_hat(im, win_c8p()), "out.pgm");
```

Figure 2.33: lena256.pgm

Figure 2.34: out.pgm

2.2 Level processings

2.2.1 level::connected_component

Purpose

Connected Component.

Prototype

```
#include <oln/level/connected.hh>

typename mute<_I, DestType>::ret
level::connected_component (const image<I1>& marker,
const struct_elt<E>& se);
```

Parameters

marker	IN	marker image
se	IN	structural element

Description

It removes the small (in area) connected components of the upper level sets of *input* using *se* as structural element. The implementation comes from *Cocquerez et Philipp, Analyse d'images, filtrages et segmentations* p.62.

See also

level::frontp_connected_component, §2.2.4, p.39.

Example

```
image2d<int_u8> light = load("light.pgm");
save(level::connected_component<int_u8>(light, win_c8p()), "out.pgm");
```

2.2.2 level::fast_maxima_killer

Purpose

Maxima killer.

Prototype

```
#include <oln/level/extrema_killer.hh>

Concrete(_I1) level::fast_maxima_killer
(const image<I1>& marker, const unsigned int area area,
const neighborhood<_N>& Ng);
```

Parameters

marker	IN	marker image
area	IN	area
Ng	IN	neighboorhood

Description

It removes the small (in area) connected components of the upper level sets of *input* using *Ng* as neighboorhood. The implementation is based on stak. Guichard and Morel, Image iterative smoothing and PDE's. Book in preparation. p 265.

See also

level::sure_maxima_killer, §2.2.5, p.39.

Example

```
image2d<int_u8> light = load("light.pgm");
save(level::fast_maxima_killer(light, 20, win_c8p()), "out.pgm");
```

2.2.3 level::fast_minima_killer

Purpose

Minima killer.

Prototype

```
#include <oln/level/extrema_killer.hh>

Concrete(_I1) level::fast_minima_killer
(const image<I1>& marker, const unsigned int area area,
const neighborhood<_N>& Ng);
```

Parameters

marker	IN	marker image
area	IN	area
Ng	IN	neighboorhood

Description

It removes the small (in area) connected components of the lower level sets of *input* using *Ng* as neighboorhood. The implementation is based on stak. Guichard and Morel, Image iterative smoothing and PDE's. Book in preparation. p 265.

See also

level::sure_minima_killer, §2.2.6, p.40.

Example

```
image2d<int_u8> light = load("light.pgm");
save(level::fast_minima_killer(light, 20, win_c8p()), "out.pgm");
```

2.2.4 level::frontp_connected_component

Purpose

Connected Component.

Prototype

```
#include <oln/level/cc.hh>

typename mute<I, DestType>::ret
level::frontp_connected_component (const image<I1>& marker,
const neighborhood<E>& se);
```

Parameters

marker	IN	marker image
se	IN	neighbourhood

Description

It removes the small (in area) connected components of the upper level sets of *input* using *se* as structural element. The implementation uses front propagation.

See also

level::connected_component, §2.2.1, p.37.

Example

```
image2d<int_u8> light = load("light.pgm");
save(level::frontp_connected_component<int_u16>(light, win_c8p()), "out.pgm");
```

2.2.5 level::sure_maxima_killer

Purpose

Maxima killer.

Prototype

```
#include <oln/level/extrema_killer.hh>

Concrete(I1) level::sure_maxima_killer
(const image<I1>& marker, const unsigned int area area,
const struct_elt<E>& se);
```

Parameters

marker	IN	marker image
area	IN	area
se	IN	structural element

Description

It removes the small (in area) connected components of the upper level sets of *input* using *se* as structual element. The implementation uses the threshold superposition principle; so it is very slow ! it works only for int_u8 images.

See also

level::fast_maxima_killer, §2.2.2, p.37.

Example

```
image2d<int_u8> light = load("light.pgm");
save(level::sure_maxima_killer(light, 20, win_c8p()), "out.pgm");
```

2.2.6 level::sure_minima_killer

Purpose

Minima killer.

Prototype

```
#include <oln/level/extrema_killer.hh>

Concrete(I1) level::sure_minima_killer
(const image<I1>& marker, const unsigned int area area,
const struct_elt<E>& se);
```

Parameters

marker	IN	marker image
area	IN	area
se	IN	structural element

Description

It removes the small (in area) connected components of the lower level sets of *input* using *se* as structual element. The implementation uses the threshold superposition principle; so it is very slow ! it works only for int_u8 images.

See also

level::fast_maxima_killer, §2.2.2, p.37.

Example

```
image2d<int_u8> light = load("light.pgm");
save(level::sure_minima_killer(light, 20, win_c8p()), "out.pgm");
```