Olena Reference Manual

A generic image processing library.

Table of Contents

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`).

2 Processings

2.1 Morphological processings

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

morpho::beucher_gradient

Purpose Morphological Beucher Gradient.

Prototype

`#include "`*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
mute<I, typename convoutput<C, Value(I)>::ret>::ret
morpho::fast::beucher_gradient (const conversion<C>& *c*, const image<I>& *input*, const struct_
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

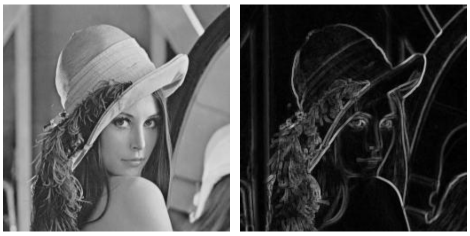
Description

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

See also [morpho.erosion], page 5,
[morpho.dilation], page 4,
[morpho.external_gradient], page 6,
[morpho.internal_gradient], page 13.

Example

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



morpho::black_top_hat

Purpose Black top hat.

Prototype

`#include "`*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>&
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<

PARAMETERS

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

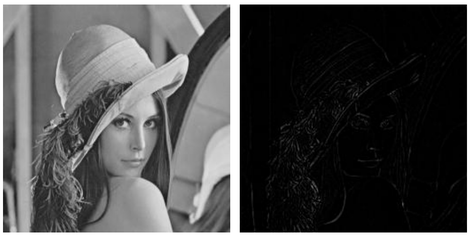
DESCRIPTION

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

SEE ALSO [morpho.closing], page 4.

EXAMPLE

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



morpho::closing

PURPOSE Morphological closing.

PROTOTYPE

`#include "`*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

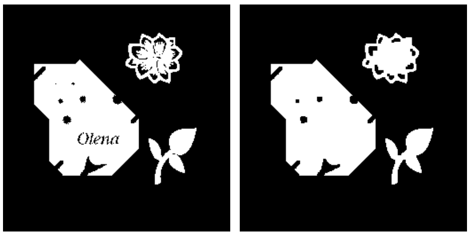
DESCRIPTION

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

SEE ALSO [morpho.erosion], page 5,
[morpho.dilation], page 4,
[morpho.closing], page 4.

EXAMPLE

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



morpho::dilation

PURPOSE Morphological dilation.

PROTOTYPE

`#include "`*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.

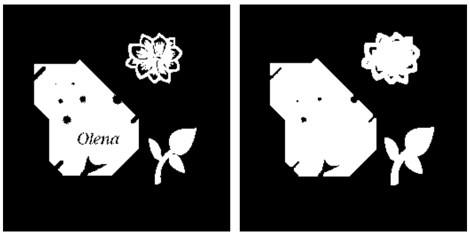
On grey-scale images, each point is replaced by the maximum value of its neighbors, as indicated by *se*. On binary images, a logical `or` is performed between neighbors.

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

See also [morpho.n_dilation], page 14,
[morpho.erosion], page 5.

Example

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



morpho::erosion

Purpose Morphological erosion.

Prototype

`#include "`*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

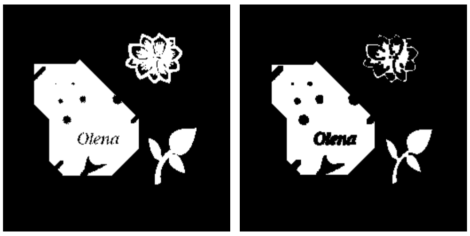
Compute the morphological erosion of *input* using *se* as structural element.

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], page 14,
[morpho.dilation], page 4.

Example

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



morpho::external_gradient

PURPOSE Morphological External Gradient.

PROTOTYPE

`#include "`*`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
mute<I, typename convoutput<C, Value(I)>::ret>::ret
morpho::fast::external_gradient (const conversion<C>& *c*, const image<I>& *input*, const struct_
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

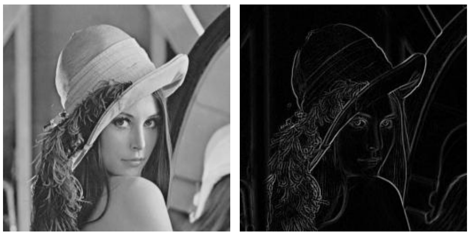
DESCRIPTION

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], page 3,
[morpho.internal_gradient], page 13,
[morpho.dilation], page 4.

EXAMPLE

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



morpho::geodesic_dilation

PURPOSE Geodesic dilation.

PROTOTYPE

`#include "`*`morpho/geodesic_dilation.hh`*`"`

Concrete(I1) **morpho::geodesic_dilation** (const image<I1>& *marker*, const image<I2>& *mask*, con

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], page 18.

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");
```

morpho::geodesic_erosion

Purpose Geodesic erosion.

Prototype

`#include "`*morpho/geodesic_erosion.hh*`"`

Concrete(I1) **morpho::geodesic_erosion** (const image<I1>& *marker*, const image<I2>& *mask*, con

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], page 18.

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");
```

morpho::hit_or_miss

Purpose Hit_or_Miss Transform.

Prototype

`#include "`*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>& *se*
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>
Concrete(I) **morpho::hit_or_miss** (const image<I>& *input*, const struct_elt<E>& *se1*, const struct
Concrete(I) **morpho::fast::hit_or_miss** (const image<I>& *input*, const struct_elt<E>& *se1*, const s

Parameters

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

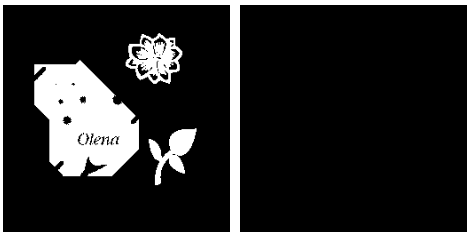
Description

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");
```



morpho::hit_or_miss_closing

Purpose Hit_or_Miss closing.

Prototype

`#include "`*morpho/hit_or_miss.hh*`"`

Concrete(I) **morpho::hit_or_miss_closing** (const image<I>& *input*, const struct_elt<E>& *se1*, cons
Concrete(I) **morpho::fast::hit_or_miss_closing** (const image<I>& *input*, const struct_elt<E>& *se1*,

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.

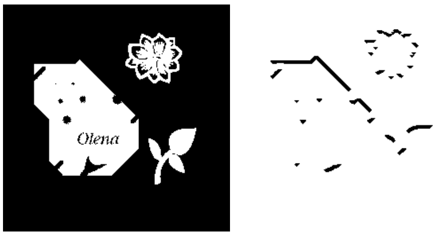
By definition *se1* and *se2* must have the same origin, and need to be disjoint. This algorithm has been extended to every data types (althought it is not increasing). Beware the result depends upon the image data type if it is not `bin`.

See also [morpho.hit_or_miss], page 7,
[morpho.hit_or_miss_closing_bg], page 9,
[morpho.hit_or_miss_opening], page 10,
[morpho.hit_or_miss_opening_bg], page 10.

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");
```



morpho::hit_or_miss_closing_bg

PURPOSE Hit_or_Miss closing of background.

PROTOTYPE

`#include "morpho/hit_or_miss.hh"`

Concrete(I) **morpho::hit_or_miss_closing_bg** (const image<I>& *input*, const struct_elt<E>& *se1*, c
Concrete(I) **morpho::fast::hit_or_miss_closing_bg** (const image<I>& *input*, const struct_elt<E>& *s*

PARAMETERS

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

DESCRIPTION

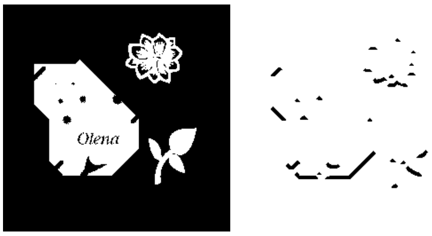
Compute the hit_or_miss closing of the background of *input* by the composite structural element (*se1*, *se2*). This is the dual transformation of hit-or-miss opening with respect to set complementation. Soille p.135.

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], page 7,
[morpho.hit_or_miss_closing], page 8,
[morpho.hit_or_miss_opening], page 10,
[morpho.hit_or_miss_opening_bg], page 10.

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");
```



morpho::hit_or_miss_opening

Purpose Hit_or_Miss opening.

Prototype

`#include "`*morpho/hit_or_miss.hh*`"`

Concrete(I) **morpho::hit_or_miss_opening** (const image<I>& *input*, const struct_elt<E>& *se1*, con
Concrete(I) **morpho::fast::hit_or_miss_opening** (const image<I>& *input*, const struct_elt<E>& *se*

Parameters

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

Description

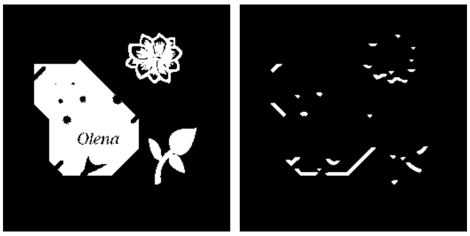
Compute the hit_or_miss opening of *input* by the composite structural element (*se1*, *se2*). Soille p.134.

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], page 7,
[morpho.hit_or_miss_closing], page 8,
[morpho.hit_or_miss_closing_bg], page 9,
[morpho.hit_or_miss_opening_bg], page 10.

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");
```



morpho::hit_or_miss_opening_bg

Purpose Hit_or_Miss opening of background.

Prototype

`#include "`*morpho/hit_or_miss.hh*`"`

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

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.

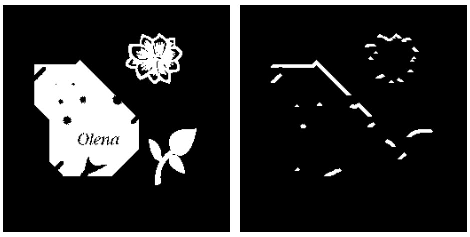
By definition *se1* and *se2* must have the same origin, and need to be disjoint. This algorithm has been extended to every data types (althought it is not increasing). Beware the result depends upon the image data type if it is not `bin`.

See also

[morpho.hit_or_miss], page 7,
[morpho.hit_or_miss_closing], page 8,
[morpho.hit_or_miss_closing_bg], page 9,
[morpho.hit_or_miss_opening], page 10.

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");
```



morpho::hybrid_geodesic_reconstruction_dilation

Purpose

Geodesic reconstruction by dilation.

Prototype

`#include "`*morpho/reconstruction.hh*`"`

Concrete(I1) **morpho::hybrid_geodesic_reconstruction_dilation** (const image<I1>& *marker*, const

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], page 18.

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
```

morpho::hybrid_geodesic_reconstruction_erosion

Purpose Geodesic reconstruction by erosion.

Prototype

`#include "`*morpho/reconstruction.hh*`"`

Concrete(I1) **morpho::hybrid_geodesic_reconstruction_erosion** (const image<I1>& *marker*, const

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], page 18.

Example

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

morpho::hybrid_minima_imposition

Purpose Minima Imposition.

Prototype

`#include "`*morpho/extrema.hh*`"`

Concrete(_I) **morpho::hybrid_minima_imposition** (const image<I1>& *input*, const image<I2>& *m*

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], page 12.

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
```

morpho::hybrid_regional_minima

Purpose Regional minima.

Prototype

`#include "`*morpho/extrema.hh*`"`

Concrete(_I) **morpho::hybrid_regional_minima** (const image<I1>& *input*, const struct_elt<E>& s

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], page 12.

Example

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

morpho::internal_gradient

Purpose Morphological Internal Gradient.

Prototype

`#include "`*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
mute<I, typename convoutput<C, Value(I)>::ret>::ret
morpho::fast::internal_gradient (const conversion<C>& *c*, const image<I>& *input*, const struct_e
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

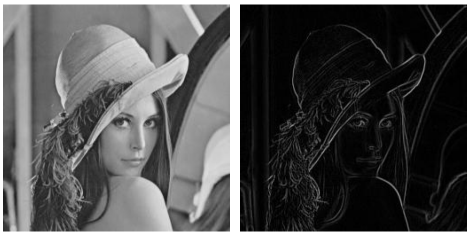
Description

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], page 3,
[morpho.external_gradient], page 6,
[morpho.erosion], page 5.

Example

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



morpho::laplacian

Purpose Laplacian.

PROTOTYPE

`#include "`*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>&
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

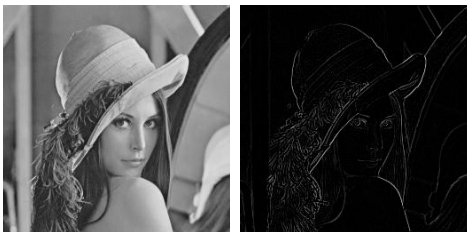
DESCRIPTION

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

SEE ALSO [morpho.dilation], page 4,
[morpho.erosion], page 5.

EXAMPLE

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



morpho::n_dilation

PURPOSE Morphological dilation itered n times.

PROTOTYPE

`#include "`*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], page 4,
[morpho.n_erosion], page 14.

morpho::n_erosion

PURPOSE Morphological erosion itered n times.

PROTOTYPE

`#include "`*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], page 5,
[morpho.n_dilation], page 14.

morpho::opening

PURPOSE Morphological opening.

PROTOTYPE

`#include` "*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

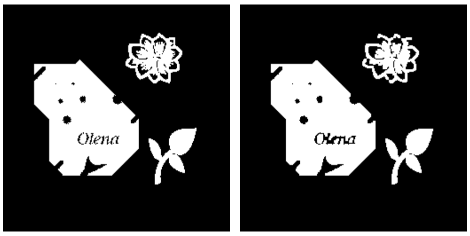
DESCRIPTION

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

SEE ALSO [morpho.erosion], page 5,
[morpho.dilation], page 4,
[morpho.closing], page 4.

EXAMPLE

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



morpho::self_complementary_top_hat

PURPOSE Self complementary top hat.

PROTOTYPE

`#include` "*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 st
typename mute<I, typename convoutput<C, Value(I)>::ret>::ret
morpho::fast::self_complementary_top_hat (const conversion<C>& *c*, const image<I>& *input*, co
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

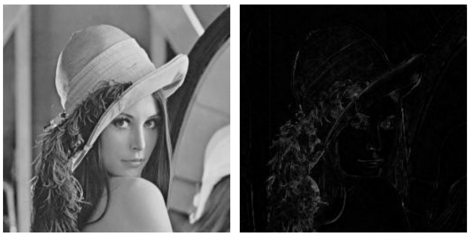
Description

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

See also [morpho.closing], page 4,
[morpho.opening], page 15.

Example

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



morpho::sequential_geodesic_reconstruction_dilation

Purpose Geodesic reconstruction by dilation.

Prototype

`#include "`*morpho/reconstruction.hh*`"`

Concrete(I1) **morpho::sequential_geodesic_reconstruction_dilation** (const image<I1>& *marker*, co

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], page 18.

Example

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

morpho::sequential_geodesic_reconstruction_erosion

Purpose Geodesic reconstruction by erosion.

Prototype

`#include "`*morpho/reconstruction.hh*`"`

Concrete(I1) **morpho::sequential_geodesic_reconstruction_erosion** (const image<I1>& *marker*, co

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], page 18.

Example

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

morpho::sequential_minima_imposition

Purpose Minima Imposition.

Prototype

`#include "`*morpho/extrema.hh*`"`

Concrete(_I) **morpho::sequential_minima_imposition** (const image<I1>& *input*, const image<I2>&

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], page 16.

Example

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

morpho::sequential_regional_minima

Purpose Regional minima.

Prototype

`#include "`*morpho/extrema.hh*`"`

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

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], page 16.

Example

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

morpho::simple_geodesic_dilation

Purpose Geodesic dilation.

Prototype

`#include "`*morpho/geodesic_dilation.hh*`"`

Concrete(I1) **morpho::simple_geodesic_dilation** (const image<I1>& *marker*, const image<I2>& *m*

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], page 18.

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");
```

morpho::simple_geodesic_erosion

Purpose Geodesic erosion.

Prototype

`#include "`*morpho/geodesic_erosion.hh*`"`

Concrete(I1) **morpho::simple_geodesic_erosion** (const image<I1>& *marker*, const image<I2>& *ma*

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], page 18.

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");
```

morpho::sure_geodesic_reconstruction_dilation

Purpose Geodesic reconstruction by dilation.

Prototype

`#include "`*morpho/reconstruction.hh*`"`

Concrete(I1) **morpho::sure_geodesic_reconstruction_dilation** (const image<I1>& *marker*, const in

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], page 18.

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()
```

morpho::sure_geodesic_reconstruction_erosion

Purpose Geodesic reconstruction by erosion.

Prototype

`#include "`*morpho/reconstruction.hh*`"`

Concrete(I1) **morpho::sure_geodesic_reconstruction_erosion** (const image<I1>& *marker*, const im

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], page 18.

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())
```

morpho::sure_minima_imposition

Purpose Minima Imposition.

Prototype

`#include "`*morpho/extrema.hh*`"`

Concrete(_I) **morpho::sure_minima_imposition** (const image<I1>& *input*, const image<I2>& *mini*

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], page 19.

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")
```

morpho::sure_regional_minima

Purpose Regional minima.

Prototype

`#include "`*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], page 19.

Example

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

morpho::top_hat_contrast_op

Purpose Top hat contrastor operator.

Prototype

`#include "`*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
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 struc

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], page 22,
[morpho.black_top_hat], page 3.

EXAMPLE

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

morpho::watershed_con

PURPOSE Connected Watershed.

PROTOTYPE

`#include "`*morpho/watershed.hh*`"`

typename mute<I, DestValue>::ret
morpho::watershed_con<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.

morpho::watershed_seg

Purpose Segmented Watershed.

Prototype

`#include "`*morpho/watershed.hh*`"`

typename mute<I, DestValue>::ret
morpho::watershed_seg<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).

morpho::watershed_seg_or

Purpose Segmented Watershed with user-supplied starting points.

Prototype

`#include "`*morpho/watershed.hh*`"`

Concrete(I2)& **morpho::watershed_seg_or** (const image<I1>& *levels*, image<I2>& *markers*, const

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.

morpho::white_top_hat

Purpose White top hat.

Prototype

`#include "`*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>&

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<
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

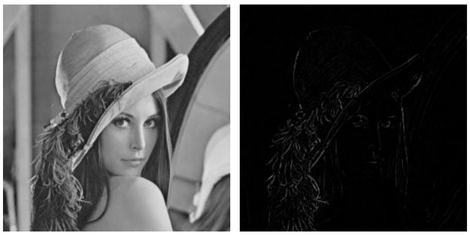
DESCRIPTION

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

SEE ALSO [morpho.opening], page 15.

EXAMPLE

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



2.2 Level processings

level::connected_component

PURPOSE Connected Component.

PROTOTYPE

`#include "`*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], page 24.

EXAMPLE

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

level::fast_maxima_killer

Purpose Maxima killer.

Prototype

`#include "`*level/extrema_killer.hh*`"`

Concrete(_I1) **level::fast_maxima_killer** (const image<I1>& *marker*, const unsigned int area *area*,

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], page 25.

Example

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

level::fast_minima_killer

Purpose Minima killer.

Prototype

`#include "`*level/extrema_killer.hh*`"`

Concrete(_I1) **level::fast_minima_killer** (const image<I1>& *marker*, const unsigned int area *area*,

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], page 25.

Example

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

level::frontp_connected_component

Purpose Connected Component.

Prototype

`#include "`*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], page 23.

EXAMPLE

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

level::sure_maxima_killer

PURPOSE Maxima killer.

PROTOTYPE
`#include "`*level/extrema_killer.hh*`"`

Concrete(I1) **level::sure_maxima_killer** (const image<I1>& *marker*, const unsigned int area *area*,

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], page 24.

EXAMPLE

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

level::sure_minima_killer

PURPOSE Minima killer.

PROTOTYPE
`#include "`*level/extrema_killer.hh*`"`

Concrete(I1) **level::sure_minima_killer** (const image<I1>& *marker*, const unsigned int area *area*,

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], page 24.

Example

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

Index