

# Olena Reference Manual

---

A generic image processing library.

---

Copyright © 2001, 2002 Laboratoire de Recherche et Développement de l'Épita.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the Free Software Foundation.

# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Processings</b>	<b>3</b>
2.1	Morphological processings	3
	morpho::beucher_gradient	3
	morpho::black_top_hat	3
	morpho::closing	4
	morpho::dilation	4
	morpho::erosion	5
	morpho::external_gradient	6
	morpho::geodesic_dilation	6
	morpho::geodesic_erosion	7
	morpho::hit_or_miss	7
	morpho::hit_or_miss_closing	8
	morpho::hit_or_miss_closing_bg	9
	morpho::hit_or_miss_opening	10
	morpho::hit_or_miss_opening_bg	10
	morpho::hybrid_geodesic_reconstruction_dilation	11
	morpho::hybrid_geodesic_reconstruction_erosion	12
	morpho::hybrid_minima_imposition	12
	morpho::hybrid_regional_minima	12
	morpho::internal_gradient	13
	morpho::laplacian	13
	morpho::n_dilation	14
	morpho::n_erosion	14
	morpho::opening	15
	morpho::self_complementary_top_hat	15
	morpho::sequential_geodesic_reconstruction_dilation	16
	morpho::sequential_geodesic_reconstruction_erosion	16
	morpho::sequential_minima_imposition	17
	morpho::sequential_regional_minima	17
	morpho::simple_geodesic_dilation	18
	morpho::simple_geodesic_erosion	18
	morpho::sure_geodesic_reconstruction_dilation	19
	morpho::sure_geodesic_reconstruction_erosion	19
	morpho::sure_minima_imposition	20
	morpho::sure_regional_minima	20
	morpho::top_hat_contrast_op	20
	morpho::watershed_con	21
	morpho::watershed_seg	22
	morpho::watershed_seg_or	22
	morpho::white_top_hat	22
2.2	Level processings	23
	level::connected_component	23
	level::fast_maxima_killer	24
	level::fast_minima_killer	24
	level::frontp_connected_component	24
	level::sure_maxima_killer	25
	level::sure_minima_killer	25
	<b>Index</b>	<b>27</b>



# 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](mailto: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>& 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

<i>c</i>	IN	conversion object
<i>input</i>	IN	input image
<i>se</i>	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>& 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

<i>c</i>	IN	conversion object
<i>input</i>	IN	input image
<i>se</i>	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

<i>input</i>	IN	input image
<i>se</i>	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

<i>input</i>	IN	input image
<i>se</i>	IN	structural element

## DESCRIPTION

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

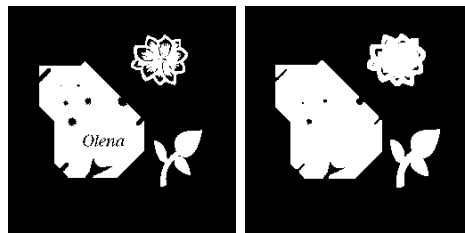
On grey-scale images, each point is replaced by the maximum value of its neighbors, as indicated by *se*. On binary images, a logical **or** is performed between neighbors.

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

<i>input</i>	IN	input image
<i>se</i>	IN	structural element

## DESCRIPTION

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

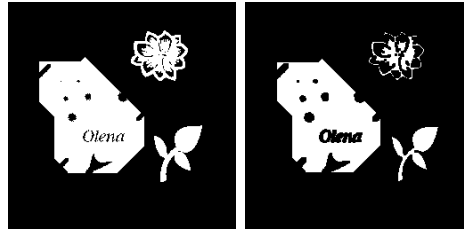
On grey-scale images, each point is replaced by the minimum value of its neighbors, as indicated by *se*. On binary images, a logical **and** is performed between neighbors.

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"

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

template<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**

<i>c</i>	IN	conversion object
<i>input</i>	IN	input image
<i>se</i>	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, const struct_elt<E>& se);
```

**PARAMETERS**

<i>marker</i>	IN	marker image
<i>mask</i>	IN	mask image
<i>se</i>	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

<i>marker</i>	IN	marker image
<i>mask</i>	IN	mask image
<i>se</i>	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

<i>c</i>	IN	conversion object
<i>input</i>	IN	input image
<i>se1</i>	IN	structural element
<i>se2</i>	IN	structural element

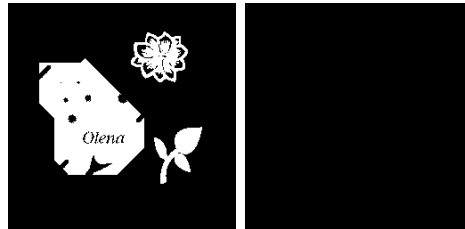
## DESCRIPTION

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*, const

Concrete(I) **morpho::fast::hit\_or\_miss\_closing** (const image<I>& *input*, const struct\_elt<E>& *se1*,

## PARAMETERS

<i>input</i>	IN	input image
<i>se1</i>	IN	structural element
<i>se2</i>	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

<i>input</i>	IN	input image
<i>se1</i>	IN	structural element
<i>se2</i>	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>& se1,
```

PARAMETERS

<i>input</i>	IN	input image
<i>se1</i>	IN	structural element
<i>se2</i>	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 (although 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

<i>input</i>	IN	input image
<i>se1</i>	IN	structural element
<i>se2</i>	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 (although 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

<i>marker</i>	IN	marker image
<i>mask</i>	IN	mask image
<i>se</i>	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

<i>marker</i>	IN	marker image
<i>mask</i>	IN	mask image
<i>se</i>	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_c8p()), "out.pgm")
```

## 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

<i>input</i>	IN	input image
<i>minima_map</i>	IN	bin image
<i>se</i>	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>& *se*)

#### PARAMETERS

<i>input</i>	IN	input image
<i>se</i>	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>& 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

<i>c</i>	IN	conversion object
<i>input</i>	IN	input image
<i>se</i>	IN	structural element

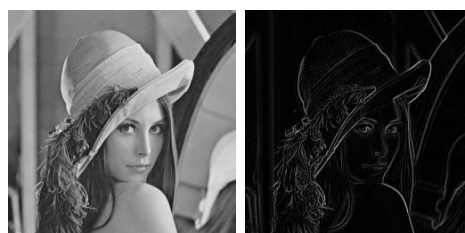
#### DESCRIPTION

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>& 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

<i>c</i>	IN	conversion object
<i>input</i>	IN	input image
<i>se</i>	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

<i>input</i>	IN	input image
<i>se</i>	IN	structural element
<i>n</i>	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

<i>input</i>	IN	input image
<i>se</i>	IN	structural element
<i>n</i>	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

<i>input</i>	IN	input image
<i>se</i>	IN	structural element

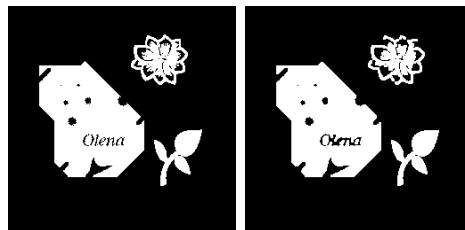
#### DESCRIPTION

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 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

<i>c</i>	IN	conversion object
<i>input</i>	IN	input image
<i>se</i>	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

<i>marker</i>	IN	marker image
<i>mask</i>	IN	mask image
<i>se</i>	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

<i>marker</i>	IN	marker image
<i>mask</i>	IN	mask image
<i>se</i>	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

<i>input</i>	IN	input image
<i>minima_map</i>	IN	bin image
<i>se</i>	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

<i>input</i>	IN	input image
<i>se</i>	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>& mask, const image<I3>& se)
```

## PARAMETERS

<i>marker</i>	IN	marker image
<i>mask</i>	IN	mask image
<i>se</i>	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>& mask, const image<I3>& se)
```

## PARAMETERS

<i>marker</i>	IN	marker image
<i>mask</i>	IN	mask image
<i>se</i>	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 im

**PARAMETERS**

<i>marker</i>	IN	marker image
<i>mask</i>	IN	mask image
<i>se</i>	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**

<i>marker</i>	IN	marker image
<i>mask</i>	IN	mask image
<i>se</i>	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>& minima)
```

PARAMETERS

<i>input</i>	IN	input image
<i>minima_map</i>	IN	bin image
<i>se</i>	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

<i>input</i>	IN	input image
<i>se</i>	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<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

<i>c</i>	IN	conversion object
<i>input</i>	IN	input image
<i>se</i>	IN	structural element

## DESCRIPTION

Enhance contrast *input* by adding the white top hat, then subtracting 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

<i>DestValue</i>		type of output labels
<i>im</i>	IN	image of levels
<i>ng</i>	IN	neighborhood to consider

## DESCRIPTION

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

**watershed\_con** creates an output 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**

<i>DestValue</i>			type of output labels
<i>im</i>	IN		image of levels
<i>ng</i>	IN		neighborhood to consider

**DESCRIPTION**

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

**watershed\_seg** creates an output 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**

<i>levels</i>	IN		image of levels
<i>markers</i>	IN	OUT	image of markers
<i>ng</i>	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 marking 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

<i>c</i>	IN	conversion object
<i>input</i>	IN	input image
<i>se</i>	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

<i>marker</i>	IN	marker image
<i>se</i>	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

<i>marker</i>	IN	marker image
<i>area</i>	IN	area
<i>Ng</i>	IN	neighborhood

DESCRIPTION

It removes the small (in area) connected components of the upper level sets of *input* using *Ng* as neighborhood. 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

<i>marker</i>	IN	marker image
<i>area</i>	IN	area
<i>Ng</i>	IN	neighborhood

DESCRIPTION

It removes the small (in area) connected components of the lower level sets of *input* using *Ng* as neighborhood. 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

<i>marker</i>	IN	marker image
<i>se</i>	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.pgm");
```

**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

<i>marker</i>	IN	marker image
<i>area</i>	IN	area
<i>se</i>	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 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

<i>marker</i>	IN	marker image
<i>area</i>	IN	area
<i>se</i>	IN	structural element

## DESCRIPTION

It removes the small (in area) connected components of the lower level sets of *input* using *se* as structural 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

## B

beucher_gradient .....	3
black_top_hat .....	3

## C

closing .....	4
connected_component .....	23

## D

dilation .....	5
----------------	---

## E

erosion .....	5
external_gradient .....	6

## F

fast_maxima_killer .....	24
fast_minima_killer .....	24
frontp_connected_component .....	24

## G

geodesic_dilation .....	6
geodesic_erosion .....	7

## H

hit_or_miss .....	7
hit_or_miss_closing .....	8
hit_or_miss_closing_bg .....	9
hit_or_miss_opening .....	10
hit_or_miss_opening_bg .....	10
hybrid_geodesic_reconstruction_dilation .....	11
hybrid_geodesic_reconstruction_erosion .....	12
hybrid_minima_imposition .....	12
hybrid_regional_minima .....	13

## I

internal_gradient .....	13
-------------------------	----

## L

laplacian .....	14
-----------------	----

## N

n_dilation .....	14
n_erosion .....	15

## O

opening .....	15
---------------	----

## S

self_complementary_top_hat .....	15
sequential_geodesic_reconstruction_dilation .....	16
sequential_geodesic_reconstruction_erosion .....	16
sequential_minima_imposition .....	17
sequential_regional_minima .....	17
simple_geodesic_dilation .....	18
simple_geodesic_erosion .....	18
sure_geodesic_reconstruction_dilation .....	19
sure_geodesic_reconstruction_erosion .....	19
sure_maxima_killer .....	25
sure_minima_imposition .....	20
sure_minima_killer .....	25
sure_regional_minima .....	20

## T

top_hat_contrast_op .....	21
---------------------------	----

## W

watershed_con .....	21
watershed_seg .....	22
watershed_seg_or .....	22
white_top_hat .....	23

