

Cellular Sensory and Wave Computing Laboratory of the Computer and Automation
Research Inst., Hungarian Academy of Sciences and the
Jedlik Laboratories of the Pázmány P. Catholic University

Software Library for Cellular Wave Computing Engines

in an era of kilo-processor chips

Budapest, December 16, 2014

Version 4.0 alpha 1

©2014 BY THE CELLULAR SENSORY WAVE COMPUTERS LABORATORY, HUNGARIAN ACADEMY OF SCIENCES (MTA SZTAKI) AND THE JEDLIK LABORATORIES OF THE PÁZMÁNY UNIVERSITY, BUDAPEST

EDITED BY K. KARACS, GY. CSEREY, Á. ZARÁNDY, P. SZOLGAY, CS. REKECZKY, L. KÉK, V. SZABÓ, G. PAZIENZA, M. MARCZELL, G. QUINTZ AND T. ROSKA
BUDAPEST, HUNGARY

Contents

1	Templates / Instructions	8
1.1	undefined type templates	8
1.1.1	AVERAGE	8
1.1.2	BipolarWave	9
1.1.3	bprop	10
1.1.4	CCDMASKL	11
1.1.5	CCDMASKR	12
1.1.6	CENTER	13
1.1.7	CENTER1	14
1.1.8	CENTER2	15
1.1.9	CENTER3	16
1.1.10	CENTER4	17
1.1.11	CENTER5	18
1.1.12	CENTER6	19
1.1.13	CENTER7	20
1.1.14	CENTER8	21
1.1.15	CLDILA	22
1.1.16	CLERO	23
1.1.17	CNTR2	24
1.1.18	CNTR3	25
1.1.19	CNTR4	26
1.1.20	CNTR5	27
1.1.21	CNTR6	28
1.1.22	CNTR7	29
1.1.23	CNTR8	30
1.1.24	ConcaveArcFiller	31
1.1.25	CONCCONT	32
1.1.26	CONCEROS	33
1.1.27	CONCHOLL	34
1.1.28	CONCTRES	35
1.1.29	CONNECTI	36
1.1.30	CORNER	37
1.1.31	DEADENDH	38
1.1.32	DEADENDV	39
1.1.33	DELDIAG1	40
1.1.34	DELVERT1	41

Contents

1.1.35	DIAG	42
1.1.36	DIAG1LIU	43
1.1.37	DiffM2	44
1.1.38	DIFFUSC	46
1.1.39	DILATION	47
1.1.40	EDGE	48
1.1.41	EDGEA	49
1.1.42	EDGEGRAY	50
1.1.43	ERASMASK	51
1.1.44	FIGDEL	52
1.1.45	FIGEXTR	53
1.1.46	FIGREC	54
1.1.47	FILL65	55
1.1.48	FINDAREA	56
1.1.49	GlobalConnectivityDetection1	57
1.1.50	Halfton	58
1.1.51	HOLE	59
1.1.52	HOLLOW	60
1.1.53	HORLINE	62
1.1.54	HORSKELL	63
1.1.55	HORSKELR	64
1.1.56	INCREASE	65
1.1.57	INTERP	66
1.1.58	JUNCTION	67
1.1.59	JunctionExtractor1	68
1.1.60	LCP	69
1.1.61	LINCUT7H	70
1.1.62	LINCUT7V	71
1.1.63	LINEXTR3	72
1.1.64	LOGAND	73
1.1.65	LOGDIF	74
1.1.66	LOGDIFNF	75
1.1.67	LOGNOT	76
1.1.68	LOGOR	77
1.1.69	LOGORN	78
1.1.70	LSE	79
1.1.71	MAJVOT1	80
1.1.72	MAJVOT3	81
1.1.73	MATCH	82
1.1.74	MOTDEPEN	83
1.1.75	MOTINDEP	84
1.1.76	MullerLyerIllusion	85
1.1.77	PATCHMAK	86
1.1.78	PathFinder	87

Contents

1.1.79	PEEL1PIX	88
1.1.80	PEELHOR	89
1.1.81	PixelSearch	90
1.1.82	POISSON	91
1.1.83	PROP1	92
1.1.84	PROP2	93
1.1.85	RECALL	94
1.1.86	RIGHTBC	95
1.1.87	RIGHTCON	96
1.1.88	RotationDetector	97
1.1.89	SHADMASK	98
1.1.90	SHADOW	99
1.1.91	shadow0	100
1.1.92	shadow45	101
1.1.93	SHADSIM	102
1.1.94	SHIFTE	103
1.1.95	SHIFTN	104
1.1.96	SHIFTNE	105
1.1.97	SHIFTNW	106
1.1.98	SHIFTS	107
1.1.99	SHIFTSE	108
1.1.100	SHIFTSW	109
1.1.101	SHIFTW	110
1.1.102	SKELBW1	111
1.1.103	SKELBW2	112
1.1.104	SKELBW3	113
1.1.105	SKELBW4	114
1.1.106	SKELBW5	115
1.1.107	SKELBW6	116
1.1.108	SKELBW7	117
1.1.109	SKELBW8	118
1.1.110	SpikeGeneration1	119
1.1.111	SpikeGeneration2	120
1.1.112	SpikeGeneration3	121
1.1.113	T1_RACC3	122
1.1.114	T2_RACC3	123
1.1.115	T3_RACC3	124
1.1.116	T4_RACC3	125
1.1.117	ThinLineRemover	126
1.1.118	TX_HCLC	127
1.1.119	TX_RACC3	128
1.1.120	TX_RACC5	129
1.1.121	VERSKELB	130
1.1.122	VERSKELT	131

Contents

1.1.123	WhitePropagation	132
1.2	Stretch type templates	133
1.2.1	STRETCH	133
1.3	Convolution type templates	134
1.3.1	CONVOL	134
1.3.2	optimedge	135
1.4	Threshold type templates	136
1.4.1	FILBLACK	136
1.4.2	FILWHITE	137
1.4.3	THRES	138
1.5	Erosion type templates	139
1.5.1	EROSION	139
1.5.2	TEXTUDIL	140
1.5.3	TEXTUERO	141
1.6	Diffusion type templates	142
1.6.1	DIFFUS	142
1.6.2	DIFFUS2	143
1.6.3	DIFFUS3	144
1.6.4	LAPLACE	145
1.7	Constrained diffusion type templates	146
1.7.1	DIFFUS4	146
1.8	Halftoning type templates	147
1.8.1	HLF3	147
1.8.2	HLF5	149
1.8.3	HLF5KC	151
1.9	Inverse halftoning type templates	153
1.9.1	HERRING	153
1.9.2	INVHLF3	155
1.9.3	INVHLF5	157
1.10	CCD type templates	159
1.10.1	CCD_DIAG	159
1.10.2	CCD_HOR	160
1.10.3	CCD_hor_l	161
1.10.4	CCD_hor_r	162
1.10.5	Ccd_NE	163
1.10.6	Ccd_NW	164
1.10.7	Ccd_SE	165
1.10.8	Ccd_SW	166
1.10.9	CCD_VERT	167
1.10.10	Ccd_vert_down	168
1.10.11	Ccd_vert_top	169
1.10.12	SMKILLER	170
1.10.13	WErosion	171

Introduction

We are witnessing a proliferation of cellular topographic processor arrays, such as the ones in the PlayStation 3 or the IBM Blue Gene and Roadrunner supercomputers, based on the multicore Cell technology developed by Sony, IBM and Toshiba; as well as kiloprocessor GPU and FPGA chips and the new 48-core Intel chip. We can also mention the Eye-RIS system by AnaFocus and the Xenon chip by Eutecus, developed in collaboration with MTA SZTAKI and the Pázmány University.

This new, completely revised edition of the Library abandons the error-prone process of manual compilation employed in previous editions, instead being generated automatically from a template database. This enables us to apply automatic verification, radically reducing the possibility of error. The new method also gave an opportunity to make the layout consistent and to give a polished look to the library. A further advantage is that the database can now be compiled into any other format, thus we could publish an HTML version as well available at <http://cnn-technology.itk.ppke.hu/CWCL>.

Presently only the first section of the old library is included, and only linear single-layer templates are listed, but we plan to extend it with the remaining content in upcoming editions. The style of the description still follows that of the first textbook¹.

As for the templates, they are for now classified and grouped based on their structural properties and whether the input and output images are grayscale or binary. Unless otherwise noted, the normalized first order CNN equation with linear delay-less templates is

$$\dot{x}_{ij} = -x_{ij} + z + \sum A(i, j; k, l) y_{kl} + \sum B(i, j; k, l) u_{kl} + \sum C(i, j; k, l) x_{kl} + \sum \hat{D}(i, j; k, l) (u_{kl}, y_{kl}, x_{kl})$$

Without the last two terms, we call it “standard” CNN dynamics.

Time is scaled in τ , the time constant of the first order CNN cell. As a default, $\tau = 1$. Observe that local template operators might have different forms (e.g. the **D** operator).

This library is the result of continuous development. It contains results published by dozens of researchers all over the world.

The library is not complete. New templates, operators and subroutines can be added. Moreover, the emergence of a new world of algorithms is foreseen. Completely new algorithms are evolving for a given task if it is implemented in a virtual cellular machine on kilo-processor chips. We encourage designers all over the world to send their templates, subroutines and programs to be included in this library, with proper reference to the original publication. You can e-mail your contributions to zarandy@sztaki.hu.

¹L. O. Chua and T. Roska, Cellular Neural Networks and visual computing: Foundations and applications, Cambridge University Press, 2002 (paperback: 2005)

1 Templates / Instructions

1.1 undefined type templates

1.1.1 AVERAGE

Smoothing with binary output.

Old names: `Smoothing`, `avetrsh`, `Average`, `Avertrsh`



Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 1.0 & 0.0 \\ 1.0 & 2.0 & 1.0 \\ 0.0 & 1.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0.0$$

Global task

Given: Static gray-scale image **P**
Input: Arbitrary(0)
Initial state: **P**
Boundary condition: Fixed(0)
Output: Binary image where black (white) pixels correspond to the locations in **P** where the average of pixel intensities over the $r=1$ feedback convolution window is positive (negative).

Examples

Input	Output
 madonna.png	

1.1.2 BipolarWave

Generates black and white waves [52]

Old names: **bipolar**



Available in: Template Library v3.1

$$\mathbf{A} = \begin{bmatrix} 0.3 & 0.3 & 0.3 \\ 0.3 & 0.8 & 0.3 \\ 0.3 & 0.3 & 0.3 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0.0$$

Global task

Given: image **P** containing three gray levels: +1, 0, -1 (black, gray, white)
 Input: **P**
 Initial state: **P**
 Boundary condition: Zero-flux
 Output: Black and white areas, the boundary of which is located at positions where the waves collided.

Examples

Input	Output
 <p>A_LETTER.png</p>	

1.1.3 bprop

Starts omni-directional black propagation from black pixels [54]

Old names: BlackPropagation

Available in: Template Library v3.1

$$\mathbf{A} = \begin{bmatrix} 0.25 & 0.25 & 0.25 \\ 0.25 & 3.0 & 0.25 \\ 0.25 & 0.25 & 0.25 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 3.75$$

Global task

Given: static binary image \mathbf{P}

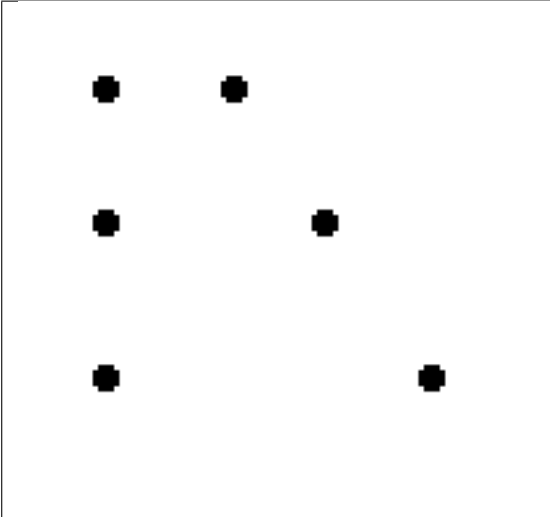
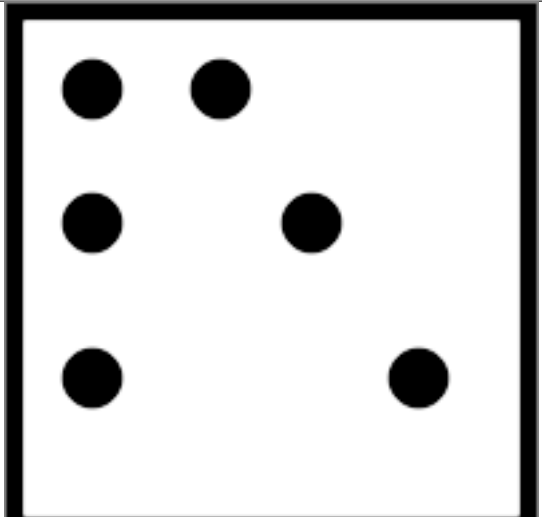
Input: Arbitrary(0)

Initial state: \mathbf{P}

Boundary condition: Fixed(0)

Output: Binary image showing black objects in \mathbf{P} with increasing black neighborhood (white objects decreasing in size).

Examples

Input	Output
 <p>points.png</p>	

1.1.4 CCDMASKL

Masked connected component detector [24]

Old names: MaskedCCD, CCDMASK

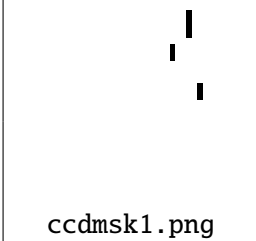
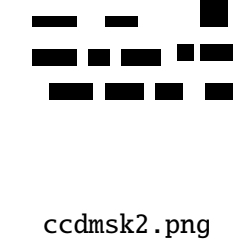
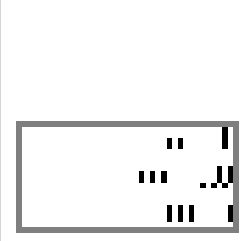
Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 1.0 & 2.0 & -1.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & -3.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = -3.0$$

Global task

Given: Static binary images P_1 (mask) and P_2
 Input: P_1
 Initial state: P_2
 Boundary condition: Fixed(0)
 Output: Binary image that is the result of CCD type shifting P_2 from right to left. Shifting is controlled by the mask P_1 .

Examples

P_1	P_2	Output
 <p>ccdmsk1.png</p>	 <p>ccdmsk2.png</p>	

1.1.5 CCDMASKR

Masked (left-to-right) connected component detection.

Old names: MaskedCCD

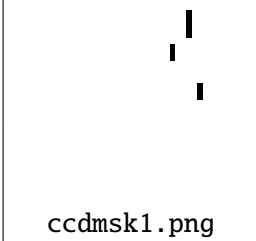
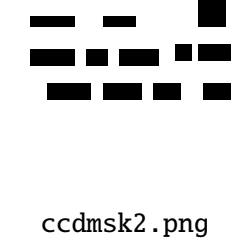
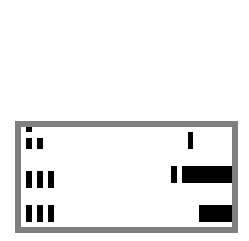
Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ -1.0 & 2.0 & 1.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & -3.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = -3.0$$

Global task

Given: Static binary images P_1 (mask) and P_2
 Input: P_1
 Initial state: P_2
 Boundary condition: Fixed(0)
 Output: Binary image that is the result of CCD type shifting P_2 from left to right. Shifting is controlled by the mask P_1 .

Examples

P_1	P_2	Output
 ccdmask1.png	 ccdmask2.png	

1.1.6 CENTER

Center point detection.

Old names: `center`, `CenterPointDetector`, `Center`

Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 1.0 & 4.0 & -1.0 \\ 1.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = [0] \quad z = -1.0$$

Global task

Given: static binary image \mathbf{P}



Input: Arbitrary(0)

Initial state: \mathbf{P}

Boundary condition: Fixed(0)

Output: Binary image where a black pixel indicates the center point of the object in \mathbf{P}

Examples

Input	Output
 <p>A_LETTER.png</p>	

1.1.7 CENTER1

Center point detection.

Old names: `CenterPointDetection(Algorithm!)`

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 1.0 & 4.0 & -1.0 \\ 1.0 & 0.0 & 0.0 \end{bmatrix} \quad z = -1.0$$

Global task

Given: Static binary image **P**

Input: -

Initial state: **P**

Boundary condition: Fixed(0)

Output: Binary image where a black pixel indicates the approximated center point of the object in **P**

1.1.8 CENTER2

Center point detection.

Old names: CenterPointDetection(Algorithm!)

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 1.0 & 6.0 & 0.0 \\ 1.0 & 0.0 & -1.0 \end{bmatrix} \quad z = -1.0$$

Global task

Given: Static binary image **P**

Input: -

Initial state: **P**

Boundary condition: Fixed(0)

Output: Binary image where a black pixel indicates the approximated center point of the object in **P**

1.1.9 CENTER3

Center point detection.

Old names: CenterPointDetection(Algorithm!)

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 0.0 & 4.0 & 0.0 \\ 0.0 & -1.0 & 0.0 \end{bmatrix} \quad z = -1.0$$

Global task

Given: Static binary image **P**

Input: -

Initial state: **P**

Boundary condition: Fixed(0)

Output: Binary image where a black pixel indicates the approximated center point of the object in **P**

1.1.10 CENTER4

Center point detection.

Old names: CenterPointDetection(Algorithm!)

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 0.0 & 6.0 & 1.0 \\ -1.0 & 0.0 & 1.0 \end{bmatrix} \quad z = -1.0$$

Global task

Given: Static binary image \mathbf{P}

Input: -

Initial state: \mathbf{P}

Boundary condition: Fixed(0)

Output: Binary image where a black pixel indicates the approximated center point of the object in \mathbf{P}

1.1.11 CENTER5

Center point detection.

Old names: CenterPointDetection(Algorithm!)

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 1.0 \\ -1.0 & 4.0 & 1.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \quad z = -1.0$$

Global task

Given:	Static binary image P
Input:	-
Initial state:	P
Boundary condition:	Fixed(0)
Output:	Binary image where a black pixel indicates the approximated center point of the object in P

1.1.12 CENTER6

Center point detection.

Old names: CenterPointDetection(Algorithm!)

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} -1.0 & 0.0 & 1.0 \\ 0.0 & 6.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix} \quad z = -1.0$$

Global task

Given: Static binary image \mathbf{P}

Input: -

Initial state: \mathbf{P}

Boundary condition: Fixed(0)

Output: Binary image where a black pixel indicates the approximated center point of the object in \mathbf{P}

1.1.13 CENTER7

Center point detection.

Old names: `CenterPointDetection(Algorithm!)`

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & -1.0 & 0.0 \\ 0.0 & 4.0 & 0.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix} \quad z = -1.0$$

Global task

Given: Static binary image **P**

Input: -

Initial state: **P**

Boundary condition: Fixed(0)

Output: Binary image where a black pixel indicates the approximated center point of the object in **P**

1.1.14 CENTER8

Center point detection.

Old names: `CenterPointDetection(Algorithm!)`

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1.0 & 0.0 & -1.0 \\ 1.0 & 6.0 & 0.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix} \quad z = -1.0$$

Global task

Given: Static binary image **P**

Input: -

Initial state: **P**

Boundary condition: Fixed(0)

Output: Binary image where a black pixel indicates the approximated center point of the object in **P**

1.1.15 CLDILA

Dilation (algo#).

Old names:

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.0 \\ 1.0 & 1.0 & 0.0 \end{bmatrix} \quad z = 3.5$$

Global task

Given:

Input:

Initial state:

Boundary condition:

Output:

1.1.16 CLERO

Erosion (algo#).

Old names:

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 1.0 & 1.0 \\ 0.0 & 1.0 & 1.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = -3.5$$

Global task

Given:

Input:

Initial state:

Boundary condition:

Output:

1.1.17 CNTR2

Center point detection.

Old names: CenterPointDetection(Algorithm!)

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 1.0 & 6.0 & 0.0 \\ 1.0 & 0.0 & -1.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 \end{bmatrix} \quad z = -1.0$$

Global task

Given:	Static binary image P
Input:	P
Initial state:	P
Boundary condition:	Fixed(0)
Output:	Binary image where a black pixel indicates the approximated center point of the object in P

1.1.18 CNTR3

Center point detection.

Old names: CenterPointDetection(Algorithm!)

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 0.0 & 4.0 & 0.0 \\ 0.0 & -1.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 \end{bmatrix} \quad z = -1.0$$

Global task

Given:	Static binary image P
Input:	P
Initial state:	P
Boundary condition:	Fixed(0)
Output:	Binary image where a black pixel indicates the approximated center point of the object in P

1.1.19 CNTR4

Center point detection.

Old names: CenterPointDetection(Algorithm!)

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 0.0 & 6.0 & 1.0 \\ -1.0 & 0.0 & 1.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 \end{bmatrix} \quad z = -1.0$$

Global task

Given: Static binary image **P**

Input: **P**

Initial state: **P**

Boundary condition: Fixed(0)

Output: Binary image where a black pixel indicates the approximated center point of the object in **P**

1.1.20 CNTR5

Center point detection.

Old names: CenterPointDetection(Algorithm!)

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 1.0 \\ -1.0 & 4.0 & 1.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 \end{bmatrix} \quad z = -1.0$$

Global task

Given: Static binary image **P**

Input: **P**

Initial state: **P**

Boundary condition: Fixed(0)

Output: Binary image where a black pixel indicates the approximated center point of the object in **P**

1.1.21 CNTR6

Center point detection.

Old names: CenterPointDetection(Algorithm!)

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} -1.0 & 0.0 & 1.0 \\ 0.0 & 6.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 \end{bmatrix} \quad z = -1.0$$

Global task

Given: Static binary image **P**

Input: **P**

Initial state: **P**

Boundary condition: Fixed(0)

Output: Binary image where a black pixel indicates the approximated center point of the object in **P**

1.1.22 CNTR7

Center point detection.

Old names: CenterPointDetection(Algorithm!)

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & -1.0 & 0.0 \\ 0.0 & 4.0 & 0.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 \end{bmatrix} \quad z = -1.0$$

Global task

Given:	Static binary image P
Input:	P
Initial state:	P
Boundary condition:	Fixed(0)
Output:	Binary image where a black pixel indicates the approximated center point of the object in P

1.1.23 CNTR8

Center point detection.

Old names: CenterPointDetection(Algorithm!)

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 1.0 & 0.0 & -1.0 \\ 1.0 & 6.0 & 0.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 \end{bmatrix} \quad z = -1.0$$

Global task

Given: Static binary image **P**

Input: **P**

Initial state: **P**

Boundary condition: Fixed(0)

Output: Binary image where a black pixel indicates the approximated center point of the object in **P**

1.1.24 ConcaveArcFiller

Fills the concave arcs of objects to prescribed direction

Old names: FILL35

Available in: Template Library v3.1

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 1 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z = 2$$

Global task

Given: static binary image P

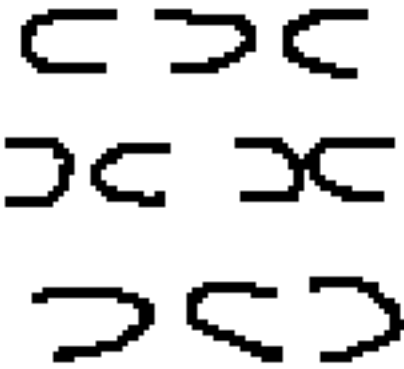
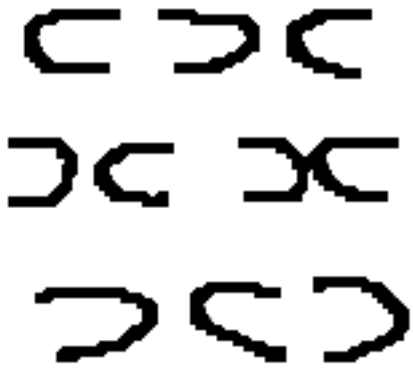
Input: P

Initial state: P

Boundary condition: Fixed(-1)

Output: Binary image in which those arcs of objects are filled which have a prescribed orientation.

Examples


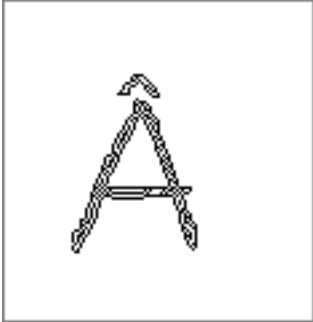
Input	Output
 <p>arcs.png</p>	

1.1.25 CONCCONT*Concentric contour detection.*Old names: `ConcentricContourDetector`, `Conccont`, `concont`

Available in: Template Library v3.1, Candy

$$A = \begin{bmatrix} 0.0 & -1.0 & 0.0 \\ -1.0 & 3.5 & -1.0 \\ 0.0 & -1.0 & 0.0 \end{bmatrix} \quad B = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 4.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = -4.0$$

Global taskGiven: Static binary image **P***Input:* **P***Initial state:* **P***Boundary condition:* Fixed(0)*Output:* Binary image representing the concentric black and white rings obtained from **P****Examples**

Input	Output
 A_LETTER.png	

1.1.26 CONCEROS

Erosion (algo#).

Old names:

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 1.0 & 2.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix} \quad z = -0.5$$

Global task

Given:

Input:

Initial state:

Boundary condition:

Output:

1.1.27 CONCHOLL

Hollow (algo#).

Old names:

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.5 & 2.0 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 3.5$$

Global task

Given:

Input:

Initial state:

Boundary condition:

Output:

1.1.28 CONCTRES

Thresholding (algo#).

Old names:

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 \end{bmatrix} \quad z = 0.0$$

Global task

Given:

Input:

Initial state:

Boundary condition:

Output:

1.1.29 CONNECTI*Deletes marked objects.*Old names: `Connectivity`, `GlobalConnectivityDetection`, `connecti`







Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.5 & 0.0 \\ 0.5 & 3.0 & 0.5 \\ 0.0 & 0.5 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & -0.5 & 0.0 \\ -0.5 & 3.0 & -0.5 \\ 0.0 & -0.5 & 0.0 \end{bmatrix} \quad z = -4.5$$

Global task

Given: Static binary images P_1 (mask) and P_2 (marker)
 Input: P_1
 Initial state: P_2
 Boundary condition: Fixed(0)
 Output: Binary image containing the unmarked objects only.

Examples

P_1	P_2	Output
 connect1.png	 connect2.png	   

1.1.30 CORNER*Convex corner detection.*Old names: `CornerDetection`, `CornerDetector`

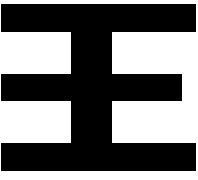
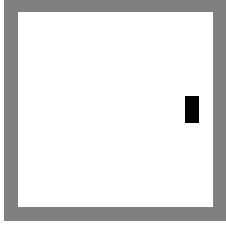
Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} -1.0 & -1.0 & -1.0 \\ -1.0 & 4.0 & -1.0 \\ -1.0 & -1.0 & -1.0 \end{bmatrix} \quad z = -5.0$$

Global task

Given: Static binary image **P**
 Input: **P**
 Initial state: Arbitrary(0)
 Boundary condition: Fixed(0)
 Output: Binary image where black pixels represent the convex corners of objects in **P**

Examples

Input	Output
 chinese.png	

1.1.31 DEADENDH

Finds the endings of horizontal (1-pixel wide) objects.

Old names: DeadEndH

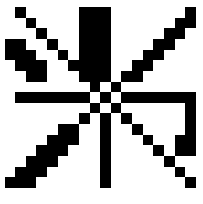
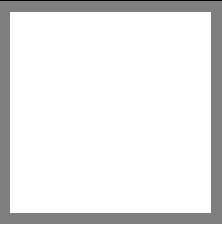
Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 3.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} -0.25 & -0.25 & -0.25 \\ -0.25 & 0.5 & -0.25 \\ -0.25 & -0.25 & -0.25 \end{bmatrix} \quad z = -5.8$$

Global task

Given: Static binary image P
 Input: P
 Initial state: Arbitrary(0)
 Boundary condition: Fixed(0)
 Output: Binary image of the endings of the horizontal (1-pixel wide) objects.

Examples

Input	Output
 <p>diag1liu.png</p>	

1.1.32 DEADENDV

Finds the endings of vertical (1-pixel wide) objects.

Old names: DeadEndV

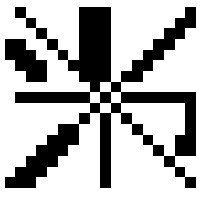
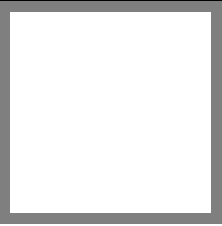
Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 3.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} -0.25 & -0.25 & -0.25 \\ -0.25 & 0.5 & -0.25 \\ -0.25 & -0.25 & -0.25 \end{bmatrix} \quad z = -5.8$$

Global task

Given: Static binary image P
 Input: P
 Initial state: Arbitrary(0)
 Boundary condition: Fixed(0)
 Output: Binary image of the endings of the vertical (1-pixel wide) objects.

Examples

Input	Output
 <p>diag1liu.png</p>	

1.1.33 DELDIAG1

Deletes one pixel wide diagonal lines.

Old names: DiagonalLineRemover, deldiag1

Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} -1.0 & 0.0 & -1.0 \\ 0.0 & 1.0 & 0.0 \\ -1.0 & 0.0 & -1.0 \end{bmatrix} \quad z = -4.0$$

Global task

Given: Static binary image P

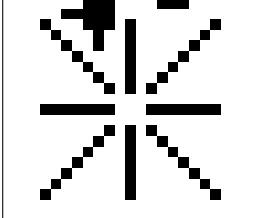
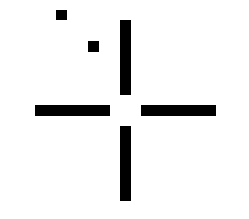
Input: P

Initial state: Arbitrary(0)

Boundary condition: Fixed(-1)

Output: Binary image where black pixels have no black neighbors in diagonal directions in P

Examples

Input	Output
 deldiag1.png	

1.1.34 DELVERT1

Deletes vertical lines.

Old names: VerticalLineRemover, delvert1

Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & -1.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & -1.0 & 0.0 \end{bmatrix} \quad z = -2.0$$

Global task

Given: Static binary image **P**

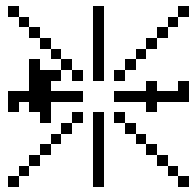
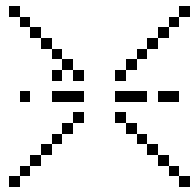
Input: **P**

Initial state: Arbitrary(0)

Boundary condition: Fixed(-1)

Output: Binary image representing **P** without vertical lines. Those parts of the objects that could be interpreted as vertical lines will also be deleted.

Examples

Input	Output
 <p>delvert1.png</p>	

1.1.35 DIAG*Detects approximately diagonal lines*Old names: `ApproxDiagonalLineDetector`, `diag`

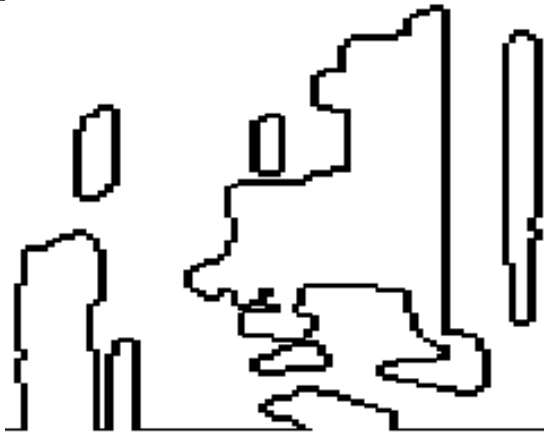

Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 2.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} -1.0 & -1.0 & -1.0 & 0.5 & 1.0 \\ -1.0 & -1.0 & 1.0 & 1.0 & 0.5 \\ -1.0 & 1.0 & 5.0 & 1.0 & -1.0 \\ 0.5 & 1.0 & 1.0 & -1.0 & -1.0 \\ 1.0 & 0.5 & -1.0 & -1.0 & -1.0 \end{bmatrix}$$

$$z = -13.0$$

Global taskGiven: Static binary image \mathbf{P} *Input:* \mathbf{P} *Initial state:* \mathbf{P} *Boundary condition:* Fixed(0)*Output:* Binary image representing the locations of approximately diagonal lines in \mathbf{P} **Examples**

Input	Output
 diag.png	

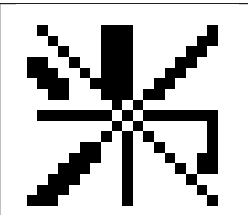
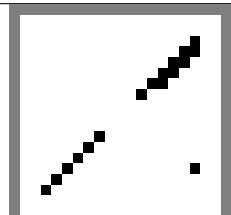
1.1.36 DIAG1LIU*Diagonal line-detector.*

Old names: DiagonalLineDetector, diag1liu

Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} -1.0 & 0.0 & 1.0 \\ 0.0 & 1.0 & 0.0 \\ 1.0 & 0.0 & -1.0 \end{bmatrix} \quad z = -4.0$$

Global taskGiven: Static binary image **P***Input:* **P***Initial state:* Arbitrary(0)*Boundary condition:* Fixed(0)*Output:* Binary image representing the locations of diagonal lines in **P****Examples**

Input	Output
 diag1liu.png	

1.1.37 DiffM2*Inverse of a linear template operation using dense support of input pixels [55]*Old names: `LinearTemplateInverse`

Available in: Template Library v3.1

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} -0.03 & -0.1 & -0.02 \\ 0.0 & 0.5 & -0.2 \\ -0.03 & -0.1 & -0.02 \end{bmatrix} \quad z = 0.0$$

Global task

Given:

a linear template as well as two static gray scale images \mathbf{P}_1 (result of the linear template operation (see the test template above and its output) and \mathbf{P}_2 . (masked version of the original image). \mathbf{P}_3 is a binary version of \mathbf{P}_2 providing the fixed state mask for CNN operation. \mathbf{P}_3 indicates the positions of supporting pixels where the interpolation is fixed.. The result of the inverse of a linear template operation is computed rapidly using masked diffusion even if the template cannot be inverted (linear template “convolution kernel” - have zero Eigen values).


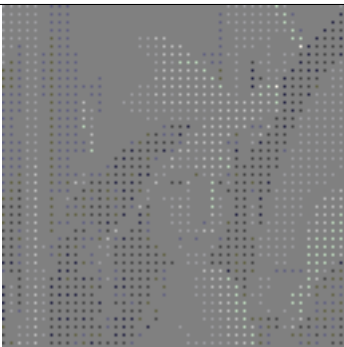
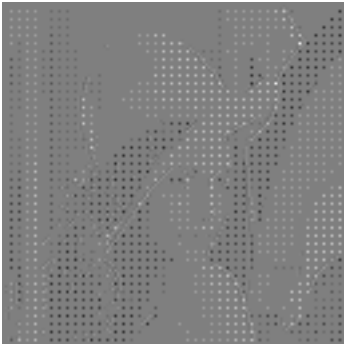
Input: \mathbf{P}_1 *Initial state:* \mathbf{P}_2 *Fixed state mask:* \mathbf{P}_3 *Boundary condition:*

Fixed(0)

Output:

Gray scale image containing the inverse of the B template operation ($\mathbf{B} = \mathbf{I} - \mathbf{A}$).

Examples

P ₁	P ₂	Output
<div></div> <div>LenaSCs.png</div>	<div></div> <div>LenaSMask.png</div>	<div></div>

1.1.38 DIFFUSC

Filtering-reconstruction with constrained linear diffusion.

Old names:

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.05 & 0.075 & 0.05 \\ 0.075 & 0.0 & 0.075 \\ 0.05 & 0.075 & 0.05 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.05 & 0.075 & 0.05 \\ 0.075 & 0.0 & 0.075 \\ 0.05 & 0.075 & 0.05 \end{bmatrix} \quad z = 0.0$$

Global task

Given: Static (noisy) gray-scale image \mathbf{P}

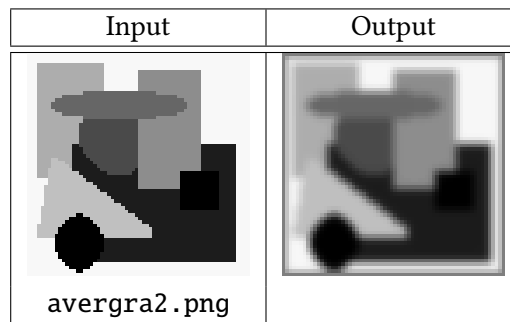
Input: \mathbf{P}

Initial state: \mathbf{P}

Boundary condition: Zero-flux

Output: Grayscale image.

Examples



1.1.39 DILATION

Binary dilation.

Old names:

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \end{bmatrix} \quad z = 2.0$$

Global task

Given: Static binary image P

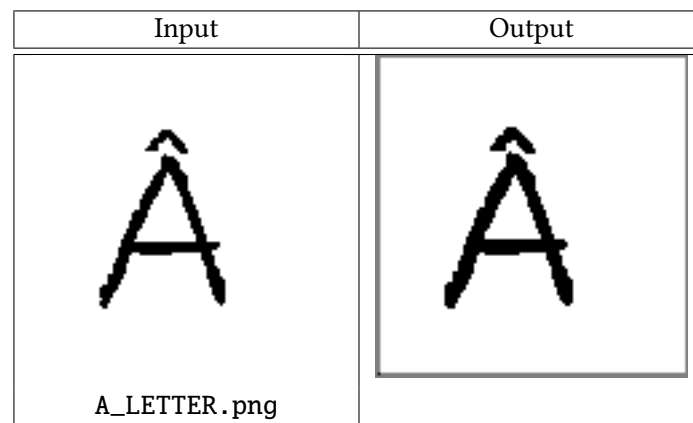
Input: P

Initial state: Arbitrary(0)

Boundary condition: Fixed(0)

Output: Binary image representing the result of the dilation operation.

Examples



1.1.40 EDGE

Binary edge detection.

Old names: EdgeDetector, EdgeDetection, edge

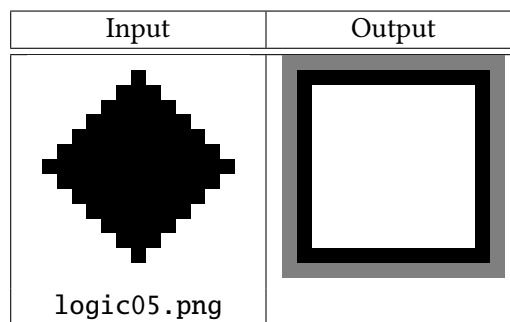
Available in: Template Library v3.1, Candy

$$A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad B = \begin{bmatrix} -1.0 & -1.0 & -1.0 \\ -1.0 & 8.0 & -1.0 \\ -1.0 & -1.0 & -1.0 \end{bmatrix} \quad z = -1.0$$

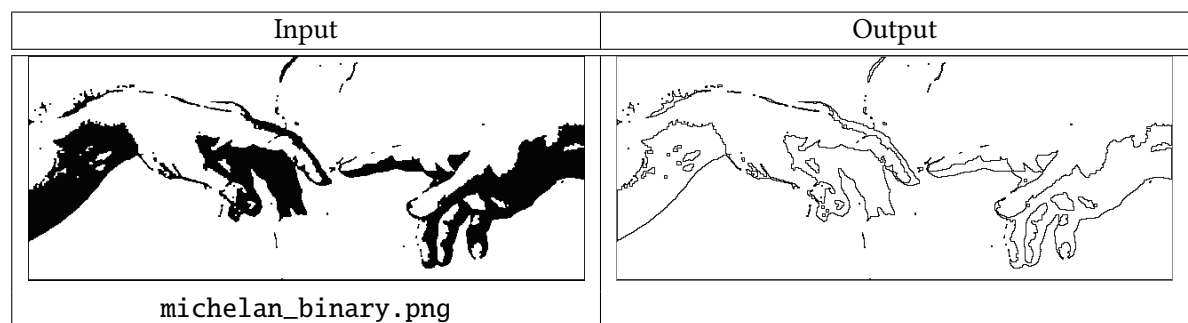
Global task

Given: Static binary image **P**
 Input: **P**
 Initial state: Arbitrary(0)
 Boundary condition: Zero-flux
 Output: Binary image showing all edges of **P** in black

Examples



Examples



1.1.41 EDGEA

Adaptive binary edge detection.

Old names:

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.5 & 0.0 & 0.0 \\ 0.0 & 0.5 & 2.0 & 0.5 & 0.0 \\ 0.0 & 0.0 & 0.5 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$



$$\mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.15 & 0.0 & 0.0 \\ 0.0 & 0.15 & 0.45 & 0.15 & 0.0 \\ 0.0 & 0.0 & 0.15 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$z = 0.0$$

Global task

Given: Static binary image **P**
 Input: **P**
 Initial state: **P**
 Boundary condition: Zero-flux
 Output: Binary image showing an edge map of **P** in black.

Examples

Input	Output
 <p>A_LETTER.png</p>	

1.1.42 EDGEGRAY

Gray-scale edge detection.

Old names:

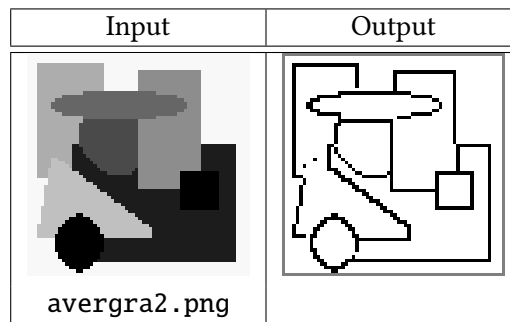
Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} -1.0 & -1.0 & -1.0 \\ -1.0 & 8.0 & -1.0 \\ -1.0 & -1.0 & -1.0 \end{bmatrix} \quad z = -0.5$$

Global task

Given: Static gray-scale image **P**
 Input: **P**
 Initial state: Arbitrary(0)
 Boundary condition: Zero-flux
 Output: Gray-scale image showing an edge map of **P** in black.

Examples



1.1.43 ERASMASK

Masked erase.

Old names: MaskedObjectExtractor, erasmask

Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 1.5 & 3.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.5 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = -1.5$$

Global task

Given: Static binary image P_1 (mask) and P_2

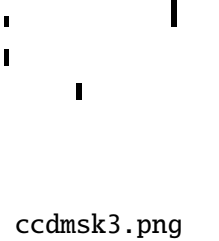
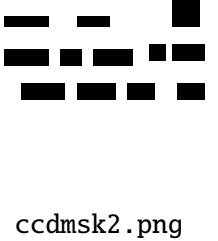

Input: P_1

Initial state: P_2

Boundary condition: Fixed(0)

Output: Binary image that is the result of erasing P_2 from left to right. Erasure is stopped by the black walls on the mask (P_1) image.

Examples

P_1	P_2	Output
 <p>ccdmsk3.png</p>	 <p>ccdmsk2.png</p>	

1.1.44 FIGDEL

Extracts isolated black pixels

Old names: FigureRemover, PointExtraction, figdel

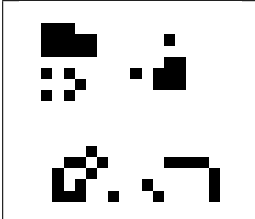
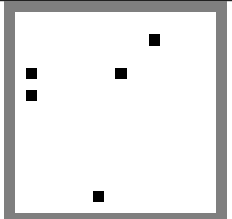
Available in: Template Library v3.1, Candy

$$A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad B = \begin{bmatrix} -1.0 & -1.0 & -1.0 \\ -1.0 & 1.0 & -1.0 \\ -1.0 & -1.0 & -1.0 \end{bmatrix} \quad z = -8.0$$

Global task

Given: Static binary image P
 Input: P
 Initial state: Arbitrary(0)
 Boundary condition: Fixed(0)
 Output: Binary image representing all isolated black pixels in P

Examples

Input	Output
 <p>figdel.png</p>	

1.1.45 FIGEXTR

Deletes isolated black pixels

Old names: `FigureExtractor`, `PointRemoval`, `figextr`

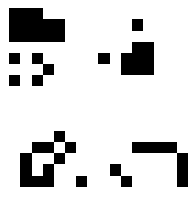
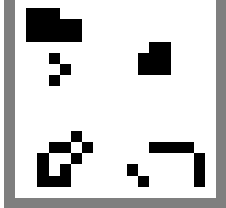
Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 1.0 & 8.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix} \quad z = -1.0$$

Global task

Given: Static binary image **P**
 Input: **P**
 Initial state: Arbitrary(0)
 Boundary condition: Fixed(0)
 Output: Binary image showing all connected components in **P**

Examples

Input	Output
 <p>figdel.png</p>	

1.1.46 FIGREC*Reconstructs marked figures.*

Old names: SelectedObjectsExtraction, FigureReconstructor

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.5 & 4.0 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 4.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 3.0$$

Global task

Given: Two static binary images P_1 (mask) and P_2 (marker). P_2 contains just a part of P_1 .

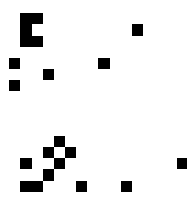
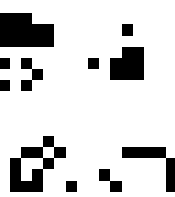
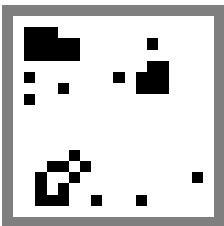
Input: P_1

Initial state: P_2

Boundary condition: Fixed(0)

Output: Binary image representing those objects of P_1 which are marked by P_2 .

Examples

P_1	P_2	Output
 figrec.png	 figdel.png	

1.1.47 FILL65

Fills the concave arcs of objects to prescribed direction

Old names: ConcaveArcFiller

Available in: Candy

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 2 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z = 3$$

Global task

Given: static binary image P



Input: P

Initial state: P

Boundary condition: Fixed(-1)

Output: Binary image in which those arcs of objects are filled which have a prescribed orientation.

Examples

Input	Output
 <p>A_LETTER.png</p>	

1.1.48 FINDAREA*Finds solid black framed areas*Old names: `FramedAreasFinder`, `FilledContourExtraction`, `findarea`

Available in: Template Library v3.1, Candy

$$A = \begin{bmatrix} 0.0 & 1.0 & 0.0 \\ 1.0 & 5.0 & 1.0 \\ 0.0 & 1.0 & 0.0 \end{bmatrix} \quad B = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = -5.25$$

Global task

Given: Two static binary images P_1 (mask) and P_2 (marker).

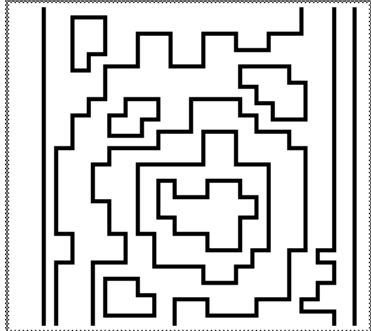
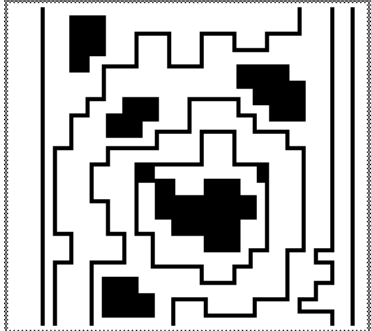
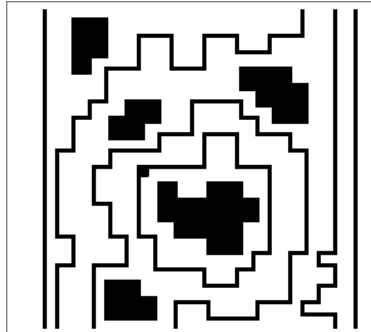
Input: P_1

Initial state: P_2

Boundary condition: Fixed(0)

Output: Binary image representing those objects of P_1 which are marked by P_2 .

Examples

P_1	P_2	Output
 <p>findare1.png</p>	 <p>findare2.png</p>	

1.1.49 GlobalConnectivityDetection1

Detects the one-pixel thick closed curves and deletes the open curves from a binary image [61]

Old names:

Available in: Template Library v3.1

$$\mathbf{A} = \begin{bmatrix} 6.0 & 6.0 & 6.0 \\ 6.0 & 9.0 & 6.0 \\ 6.0 & 6.0 & 6.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} -3.0 & -3.0 & -3.0 \\ -3.0 & 9.0 & -3.0 \\ -3.0 & -3.0 & -3.0 \end{bmatrix} \quad z = -4.5$$

Global task

Given: static binary image P

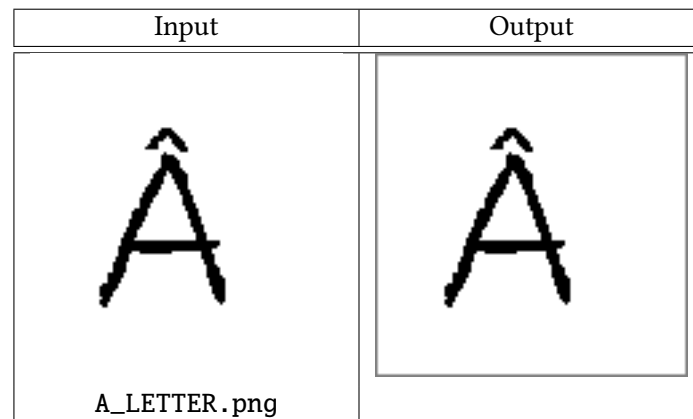
Input: P

Initial state: P

Boundary condition: Fixed(0)

Output: Binary image which contains all closed curves present in the initial image P

Examples



1.1.50 Halfton

no description

Old names:

Available in: AladdinPro

$$\mathbf{A} = \begin{bmatrix} -0.07 & -0.1 & -0.07 \\ -0.1 & 1.03 & -0.1 \\ -0.07 & -0.1 & -0.07 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.07 & 0.1 & 0.07 \\ 0.1 & 0.32 & 0.1 \\ 0.07 & 0.1 & 0.07 \end{bmatrix} \quad z = 0.0$$

Global task

Given:

Input:

Initial state:

Boundary condition:

Output:



1.1.51 HOLE*Performs hole filling.*

Old names: HoleFiller, Hole-Filling, hole

Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 1.0 & 0.0 \\ 1.0 & 3.0 & 1.0 \\ 0.0 & 1.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 4.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = -1.0$$

Global taskGiven: Static binary image **P***Input:* **P***Initial state:* 1*Boundary condition:* Fixed(0)*Output:* Binary image representing **P** with holes filled.**Examples**

Input	Output
 <p>A_LETTER.png</p>	

1.1.52 HOLLOW

Fills the concave locations of objects

Old names: `ConcaveLocationFiller`, `hollow`

Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.5 & 2.0 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 3.25$$

Global task

Given: Static binary image **P**

Input: **P**

Initial state: **P**

Boundary condition: Fixed(0)

Output: Binary image in which the concave locations of objects are black.

1.1.53 HORLINE

Horizontal line detector.

Old names:

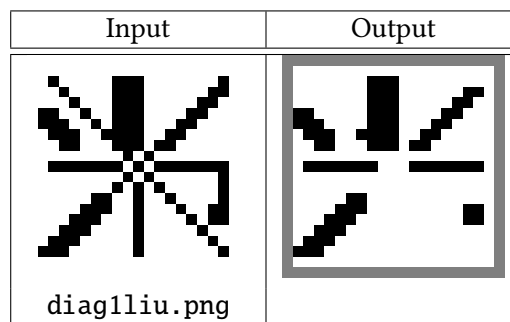
Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 1.0 & 2.0 & 1.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 1.0 & 2.0 & 1.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = -1.0$$

Global task

Given: Static binary image \mathbf{P}
 Input: \mathbf{P}
 Initial state: \mathbf{P}
 Boundary condition: Zero-flux
 Output: Binary image, representing the horizontal lines in \mathbf{P}

Examples



1.1.54 HORSKELL

Horizontal skeleton from the left.

Old names: HorSkell



Available in: Candy

$$A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 3.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad B = \begin{bmatrix} 0.5 & 0.0 & 0.125 \\ 0.5 & 0.5 & -0.5 \\ 0.5 & 0.0 & 0.125 \end{bmatrix} \quad z = -1.0$$

Global task

Given: Static binary image P
Input: P
Initial state: Arbitrary(0)
Boundary condition: Fixed(0)
Output: Binary image, peeling the black pixels from the left of the object.

Examples

Input	Output
 A_LETTER.png	



1.1.55 HORSKELR*Horizontal skeleton from the right.*

Old names: HorSkelR

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 3.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.125 & 0.0 & 0.5 \\ -0.5 & 0.5 & -0.5 \\ 0.125 & 0.0 & 0.5 \end{bmatrix} \quad z = -1.0$$

Global taskGiven: Static binary image **P***Input:* **P***Initial state:* Arbitrary(0)*Boundary condition:* Fixed(0)*Output:* Binary image, peeling the black pixels from the right of the object.**Examples**

Input	Output
 <p>A_LETTER.png</p>	

1.1.56 INCREASE

Increases the object by one pixel.

Old names: ObjectIncreasing, increase

Available in: Template Library v3.1, Candy

$$A = \begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix} \quad B = [0] \quad z = 4.0$$

Global task

Given: Static binary image **P**


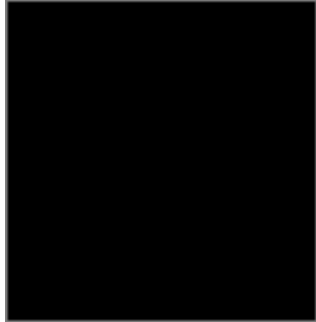
Input: Arbitrary(0)

Initial state: **P**

Boundary condition: Zero-flux

Output: Binary image representing the objects of **P** increased by 1 pixel in all direction.

Examples

Input	Output
 <p>A_LETTER.png</p>	

1.1.57 INTERP*Interpolates a smooth surface through given points*

Old names: SurfaceInterpolation, INTERPOL, interp

Available in: Template Library v3.1, Candy

$$A = \begin{bmatrix} 0.0 & 0.0 & -2.0 & 0.0 & 0.0 \\ 0.0 & -4.0 & 16.0 & -4.0 & 0.0 \\ -2.0 & 16.0 & -39.0 & 16.0 & -2.0 \\ 0.0 & -4.0 & 16.0 & -4.0 & 0.0 \\ 0.0 & 0.0 & -2.0 & 0.0 & 0.0 \end{bmatrix}$$

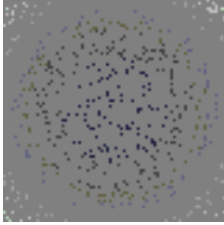

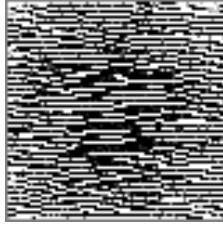
$$B = \begin{bmatrix} 0 \end{bmatrix}$$

$$z = 0$$

Global task

Given: A static grayscale image P_1 and a static binary image P_2
 Input: Arbitrary(0)
 Initial state: P_1
 Fixed state mask: P_2
 Bias map: -
 Boundary condition: Fixed(0)
 Output: Grayscale image representing an interpolated surface that fits the given points and is as smooth as possible.

Examples



P_1	P_2	Output
		
interp1.png	interp2.png	

1.1.58 JUNCTION*Extracts the junctions of a skeleton.*Old names: `JunctionExtractor`, `junction`

Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 1.0 & 6.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix} \quad z = -3.0$$

Global taskGiven: Static binary image **P***Input:* **P***Initial state:* Arbitrary(0)*Boundary condition:* Zero-flux*Output:* Binary image showing the junctions of a skeleton.**Examples**

Input	Output
 <p>junction.png</p>	

1.1.59 JunctionExtractor1

Finding the intersection points of thin (one-pixel thick) lines from two binary images

Old names:

Available in: Template Library v3.1

$$\mathbf{A} = \begin{bmatrix} -0.5 & -0.5 & -0.5 \\ -0.5 & 3.0 & -0.5 \\ -0.5 & -0.5 & -0.5 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} -0.5 & -0.5 & -0.5 \\ -0.5 & 3.0 & -0.5 \\ -0.5 & -0.5 & -0.5 \end{bmatrix} \quad z = -8.5$$

Global task

Given: two static binary images P_1 and P_2 containing thin (one-pixel thick) lines or curves, among other (compact) objects

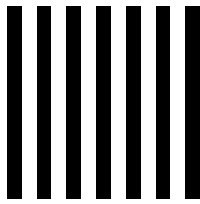
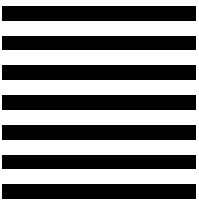
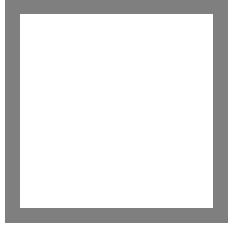
Input: P_1

Initial state: P_2

Boundary condition: Fixed(0)

Output: Binary image containing all the intersection points between the thin lines contained in the binary images P_1 and P_2

Examples

P_1	P_2	Output
 <p>logic01.png</p>	 <p>logic02.png</p>	

1.1.60 LCP

Local concave place detector.

Old names: LocalConcavePlaceDetector, lcp

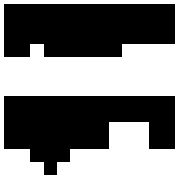
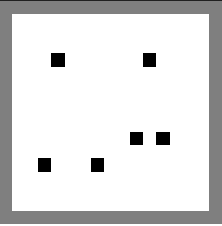
Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 2.0 & 2.0 & 2.0 \\ 1.0 & -2.0 & 1.0 \end{bmatrix} \quad z = -7.0$$

Global task

Given: Static binary image **P**
 Input: **P**
 Initial state: Arbitrary(0)
 Boundary condition: Fixed(0)
 Output: Binary image showing the local concave places of **P**

Examples

Input	Output
 <p>lcp_lse.png</p>	

1.1.61 LINCUT7H

Deletes horizontal lines not longer than 7 pixels.

Old names:

Available in: Candy

$$A = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 0.5 & 2.0 & 1.0 & 0.5 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$


$$B = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 1.0 & 1.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$z = -5.5$$

Global task

Given: Static binary image **P**
Input: **P**
Initial state: **P**
Boundary condition: Fixed(-1)
Output: Binary image where black pixels identify the horizontal lines with a length of 8 or more pixels in **P**

Examples

Input	Output
 lincut7v.png	

1.1.62 LINCUT7V

Deletes vertical lines not longer than 7 pixels.

Old names: LE7pixelVerticalLineRemover, lincut7v, CUT7V

Available in: Template Library v3.1, Candy

$$A = \begin{bmatrix} 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.5 & 0.0 & 0.0 \\ 0.0 & 0.0 & 2.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.5 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \end{bmatrix}$$

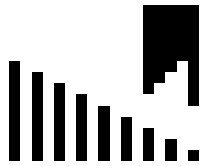
$$B = \begin{bmatrix} 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \end{bmatrix}$$

$$z = -5.5$$

Global task

Given: Static binary image **P**
 Input: **P**
 Initial state: **P**
 Boundary condition: Fixed(-1)
 Output: Binary image where black pixels identify the vertical lines with a length of 8 or more pixels in **P**

Examples

Input	Output
 <p>lincut7v.png</p>	

1.1.63 LINEXTR3

Lines-not-longer-than-3-pixels detector.

Old names: LE3pixelLineDetector, LGTHTUNE, linextr3

Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.3 & 0.3 & 0.3 & 0.0 \\ 0.0 & 0.3 & 3.0 & 0.3 & 0.0 \\ 0.0 & 0.3 & 0.3 & 0.3 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

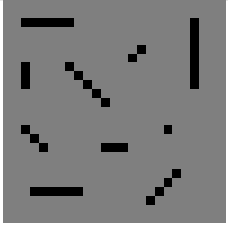
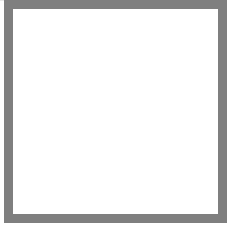
$$\mathbf{B} = \begin{bmatrix} -1.0 & 0.0 & -1.0 & 0.0 & -1.0 \\ 0.0 & -1.0 & -1.0 & -1.0 & 0.0 \\ -1.0 & -1.0 & 4.0 & -1.0 & -1.0 \\ 0.0 & -1.0 & -1.0 & -1.0 & 0.0 \\ -1.0 & 0.0 & -1.0 & 0.0 & -1.0 \end{bmatrix}$$

$$z = -2.0$$

Global task

Given: Static binary image \mathbf{P}
 Input: \mathbf{P}
 Initial state: \mathbf{P}
 Boundary condition: Fixed(0)
 Output: Binary image representing only lines not longer than 3 pixels in \mathbf{P}

Examples

Input	Output
 <p>linextr3.png</p>	

1.1.64 LOGAND*Logic AND (and Set Intersection).*

Old names: LogicAND, LogicANDOperation, AND, logand

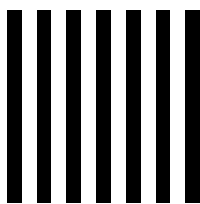
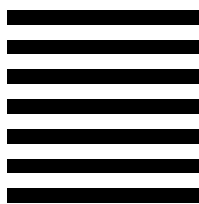
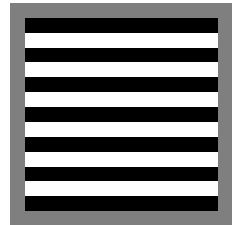
Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = -1.0$$

Global taskGiven: Two static binary images P_1 and P_2 Input: P_1 Initial state: P_2

Boundary condition: Fixed(0)

Output: Binary output of the logic operation AND between P_1 and P_2 (Set Intersection).**Examples**

P_1	P_2	Output
 logic01.png	 logic02.png	

1.1.65 LOGDIF

Logic Difference (alt: Relative Set Complement).

Old names: LogicDifference1, logdif, PA-PB

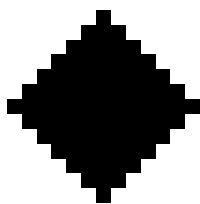
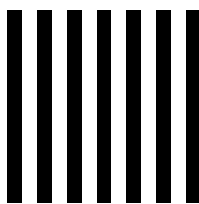
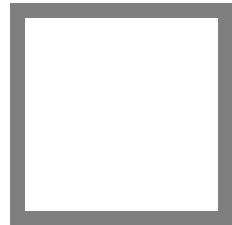
Available in: Template Library v3.1, Candy

$$A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad B = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & -1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = -1.0$$

Global task

Given: Two static binary images P_1 and P_2
 Input: P_1
 Initial state: P_2
 Boundary condition: Fixed(0)
 Output: Binary image representing the set-theoretic, or logic complement of P_2 relative to P_1 .

Examples

P_1	P_2	Output
 logic05.png	 logic01.png	

1.1.66 LOGDIFNF

Logic difference between the initial state and the input pictures with noise filtering.

Old names: LogicDifference2, ImageDifferenceComputation, PA-PB_F1, logdifnf

Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.25 & 0.25 & 0.25 \\ 0.25 & 2.0 & 0.25 \\ 0.25 & 0.25 & 0.25 \end{bmatrix} \quad z = -4.75$$

Global task

Given: Static binary image P


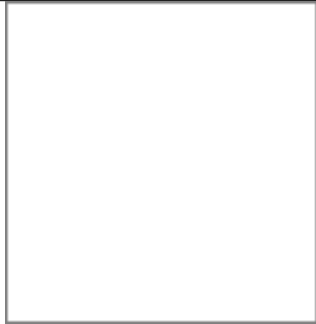
Input: P

Initial state: Arbitrary(0)

Boundary condition: Fixed(0)

Output: Binary image where black pixels identify the moving parts of P

Examples

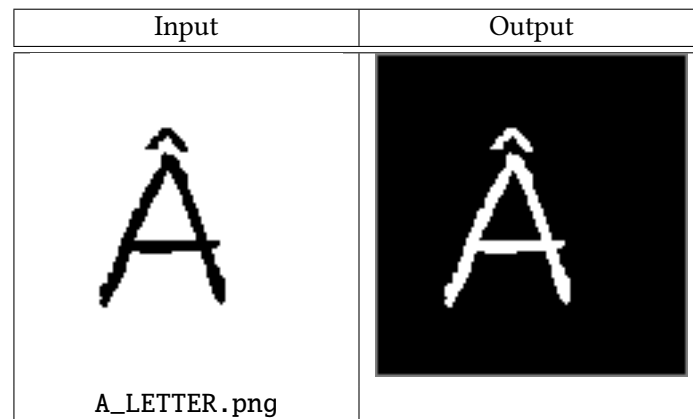
Input	Output
 <p>A_LETTER.png</p>	

1.1.67 LOGNOT*Logic NOT (alt: Set Complementation)*

Old names: LogicNOT, LogicNOToperation, INV, lognot

Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & -2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0.0$$

Global taskGiven: Static binary image **P***Input:* **P***Initial state:* Arbitrary(0)*Boundary condition:* Fixed(0)*Output:* Binary image where each black pixel in **P** becomes white, and vice versa.**Examples**

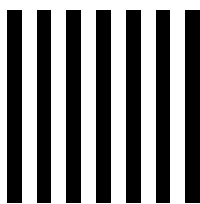
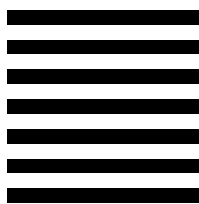
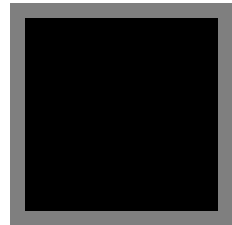
1.1.68 LOGOR*Logic OR (alt: Set Union).*

Old names: LogicOR, LogicOROperation, logor, OR

Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 1.0$$

Global taskGiven: Two static binary images P_1 and P_2 *Input:* P_1 *Initial state:* P_2 *Boundary condition:* Fixed(0)*Output:* Binary output of the logic operation OR between P_1 and P_2 (Set Union).**Examples**

P_1	P_2	Output
 logic01.png	 logic02.png	

1.1.69 LOGORN

Logic OR function of the initial state and logic NOT function of the input.

Old names: LogicORwithNOT, logorn, INV-OR

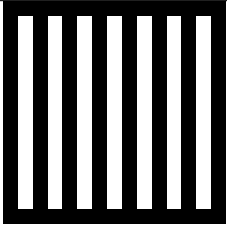
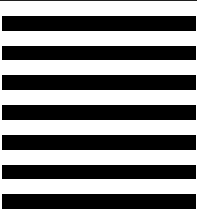
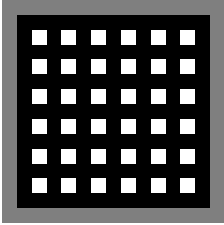
Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & -1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 1.0$$

Global task

Given: Two static binary images P_1 and P_2
 Input: P_1
 Initial state: P_2
 Boundary condition: Fixed(0)
 Output: Binary output of the logic operation OR between NOT P_1 and P_2 .

Examples

P_1	P_2	Output
 logic06.png	 logic02.png	

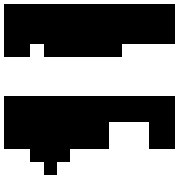
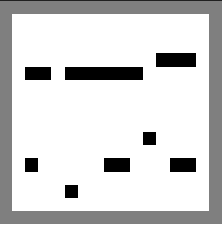
1.1.70 LSE*Local southern element detector.*

Old names: LocalSouthernElementDetector, lse

Available in: Template Library v3.1, Candy

$$A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad B = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ -1.0 & -1.0 & -1.0 \end{bmatrix} \quad z = -3.0$$

Global taskGiven: Static binary image **P***Input:* **P***Initial state:* Arbitrary(0)*Boundary condition:* Fixed(0)*Output:* Binary image representing local southern elements of objects in **P****Examples**

Input	Output
 lcp_lse.png	

1.1.71 MAJVOT1

Majority vote-taker.

Old names: MajorityVoteTaker(Algorithm!), MAJVOT, majvot1, MajorityVoteTaker, majvot2

Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.05 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0.0$$

Global task

Given: Static binary image **P**

Input: **P**

Initial state: **P**

Boundary condition: Fixed(0)

Output:

1.1.72 MAJVOT3

Majority vote-taker (compares the sum in a local neighborhood to the specified threshold).

Old names:

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix} \quad z = -6.5$$

Global task

Given: Static gray-scale image **P**

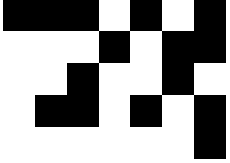

Input: **P**

Initial state: Arbitrary(0)

Boundary condition: Fixed(0)

Output: Binary image - black pixels mark those locations where the sum in local neighborhood ($r=1$) exceeds the specified threshold.

Examples

Input	Output
 <p>histogr.png</p>	

1.1.73 MATCH

Finds matching patterns

Old names: `PatternMatchingFinder`, `match`

Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1.0 & -1.0 & 1.0 \\ 0.0 & 1.0 & 0.0 \\ 1.0 & -1.0 & 1.0 \end{bmatrix} \quad z = -6.5$$

Global task

Given: Static binary image **P** possessing the 3x3 pattern prescribed by the template.

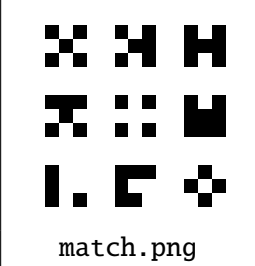
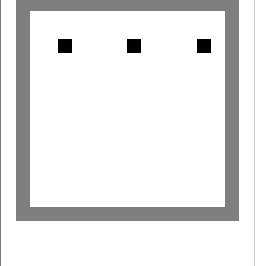
Input: **P**

Initial state: Arbitrary(0)

Boundary condition: Fixed(0)

Output: Binary image representing the locations of the 3x3 pattern prescribed by the template. The pattern having a black/white pixel where the template value is +1/-1, respectively, is detected.

Examples

Input	Output
 match.png	 match.png

1.1.74 MOTDEPEN

Direction and speed dependent motion detection.

Old names: MotionDetection1, MOVEHOR, MotionDetection, motdepen

Available in: Template Library v3.1, Candy

$$A = \begin{bmatrix} -0.1 & -0.1 & -0.1 \\ -0.1 & 0.0 & -0.1 \\ -0.1 & -0.1 & -0.1 \end{bmatrix} \quad B = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.5 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = -2.0$$

Global task

Given: Static binary image **P**



Input: **P**

Initial state: **P**

Boundary condition: Fixed(0)

Output: Binary image representing only objects of **P** moving horizontally to the right with a speed of 1 pixel/delay-time.

Examples

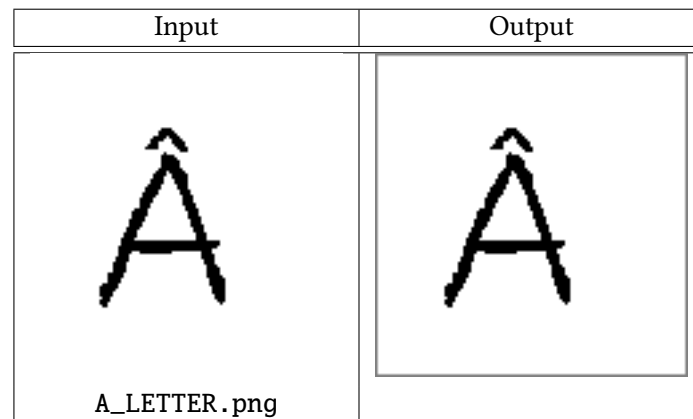
Input	Output
 <p>A_LETTER.png</p>	

1.1.75 MOTINDEP*Direction independent motion detection [7]*

Old names: MotionDetection2, MD_CONT, motindep, SpeedDetection

Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 6.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = -2.0$$

Global taskGiven: Static binary image **P***Input:* **P***Initial state:* **P***Boundary condition:* Fixed(0)*Output:* Binary image representing only objects of **P** moving slower than 1 pixel/delay-time.**Examples**

1.1.76 MullerLyerIllusion*Simulates the Müller-Lyer illusion [13]*

Old names: MULLER

Available in: Template Library v3.1

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.3 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} -0.1 & -0.1 & -0.1 & -0.1 & -0.1 \\ -0.1 & -0.1 & -0.1 & -0.1 & -0.1 \\ -0.1 & -0.1 & 1.3 & -0.1 & -0.1 \\ -0.1 & -0.1 & -0.1 & -0.1 & -0.1 \\ -0.1 & -0.1 & -0.1 & -0.1 & -0.1 \end{bmatrix}$$

$$z = -2.8$$

Global task

Given: static binary image **P** representing two horizontal lines between arrows. The arrows are dark-gray, the background is white


Input: **P**

Initial state: **P**

Boundary condition: Fixed(0)

Output: Binary image showing that the horizontal line on the top in **P** seems to be longer than the other one.

Examples

Input	Output
 <p>A_LETTER.png</p>	

1.1.77 PATCHMAK

Patch maker.

Old names: PatchMaker, patchmak

Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 1.0 & 0.0 \\ 1.0 & 2.0 & 1.0 \\ 0.0 & 1.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 4.5$$

Global task

Given: Static binary image P

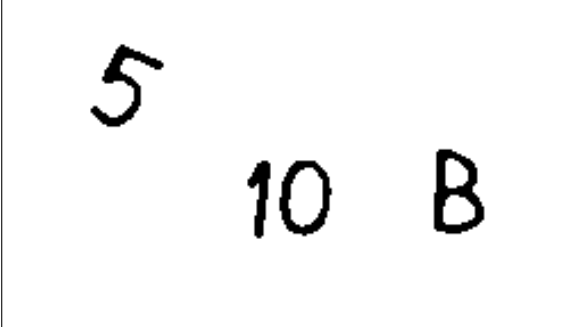

Input: P

Initial state: P

Boundary condition: Zero-flux

Output: Binary image with enlarged objects of P obtained after a certain time.

Examples

Input	Output
 <p>patchmak.png</p>	

1.1.78 PathFinder

Finding all paths between two selected points through a labyrinth [61]

Old names:

Available in: Template Library v3.1

$$\mathbf{A} = \begin{bmatrix} 0.5 & 4.0 & 0.5 \\ 4.0 & 12.0 & 4.0 \\ 0.5 & 4.0 & 0.5 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 8.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 8.0$$

Global task

Given: static binary image **P** representing a labyrinth made of one-pixel thick white curves on a black background

Input:

Initial state: **P**

Boundary condition: Fixed(0)

Output: Binary image containing all the paths connecting the marked points (made of white curves against a black background)

1.1.79 PEEL1PIX

Peel one pixel from all directions (#).

Old names:

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.4 & 0.0 \\ 0.4 & 1.4 & 0.4 \\ 0.0 & 0.4 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 4.6 & -2.8 & 4.6 \\ -2.8 & 1.0 & -2.8 \\ 4.6 & -2.8 & 4.6 \end{bmatrix} \quad z = -7.2$$

Global task

Given:

Input:

Initial state:

Boundary condition:

Output:

1.1.80 PEELHOR

Peels one pixel from the left.

Old names: LeftPeeler, peelhor

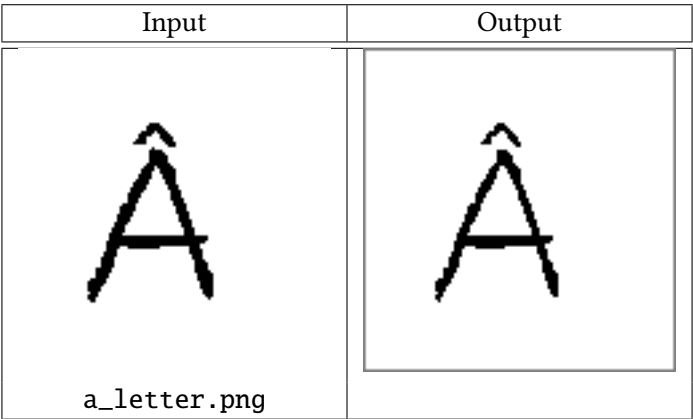
Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = -1.0$$

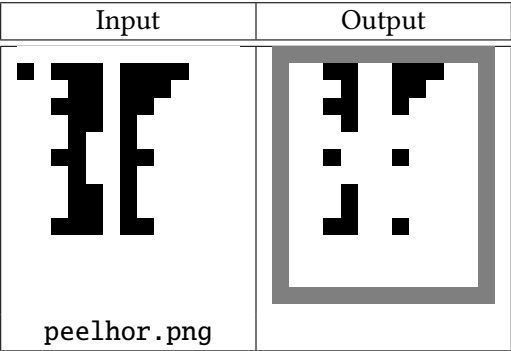
Global task

Given: Static binary image **P**
Input: **P**
Initial state: **P**
Boundary condition: Zero-flux
Output: Binary image representing the objects of **P** peeled with one pixel from the left.

Examples



Examples



1.1.81 PixelSearch

Pixel search in a given range [72]

Old names:

Available in: Template Library v3.1

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$z = -1$$

Global task

Given: static binary image P_1, P_2


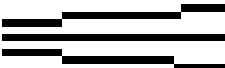
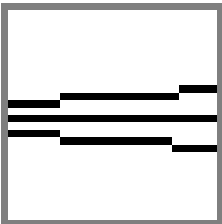
Input: \mathbf{P}_1

Initial state: \mathbf{P}_2

Boundary condition: Fixed(0)

Output: Binary image representing the pixels being at the specified distance from the reference.

Examples

P_1	P_2	Output
 <p>input_reference.png</p>	 <p>initial_available.png</p>	

1.1.82 POISSON

Solves the Poisson PDE ($Dx = -f(x)$).

Old names: PoissonPDESolver

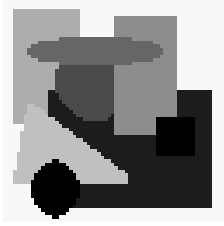
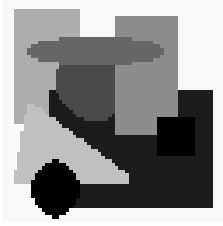
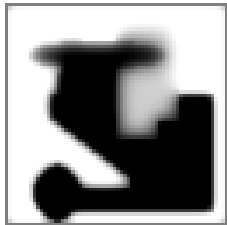
Available in: Candy

$$A = \begin{bmatrix} 0.0 & 1.0 & 0.0 \\ 1.0 & -3.0 & 1.0 \\ 0.0 & 1.0 & 0.0 \end{bmatrix} \quad B = [0] \quad z = 0$$

Global task

- Given: Static gray-scale images P_1 and $P_2 = -f(x)$
Input: Arbitrary(0)
Initial state: P_1
Bias map: P_2
Boundary condition: Zero-flux
Output: Gray-scale image - the solution of the Poisson equation.

Examples

P ₁	P ₂	Output
 avergra2.png	 avergra2.png	

1.1.83 PROP1

Trigger-wave generator (expands the black regions).

Old names:

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.25 & 0.25 & 0.25 \\ 0.25 & 3.0 & 0.25 \\ 0.25 & 0.25 & 0.25 \end{bmatrix} \quad \mathbf{B} = [0] \quad z = 3.75$$

Global task

Given: Static binary image \mathbf{P}



Input: -

Initial state: \mathbf{P}

Boundary condition: Zero-flux

Output: Binary image with enlarged objects of \mathbf{P} obtained after a certain time.

Examples

Input	Output
 <p>A_LETTER.png</p>	

1.1.84 PROP2

Trigger-wave generator (expands the white regions).

Old names:

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.25 & 0.25 & 0.25 \\ 0.25 & 3.0 & 0.25 \\ 0.25 & 0.25 & 0.25 \end{bmatrix} \quad \mathbf{B} = [0] \quad z = -2.75$$

Global task

Given: Static binary image \mathbf{P}


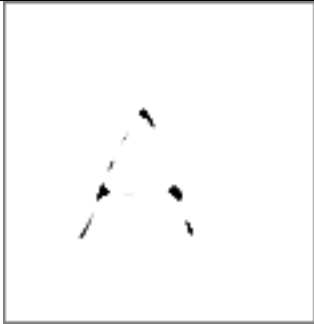
Input: -

Initial state: \mathbf{P}

Boundary condition: Zero-flux

Output: Binary image with reduced objects of \mathbf{P} obtained after a certain time.

Examples

Input	Output
 <p>A_LETTER.png</p>	

1.1.85 RECALL*Figure reconstruction from markers.*

Old names: FigureReconstructor

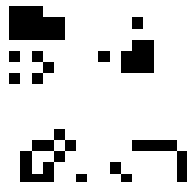
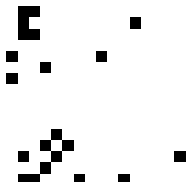
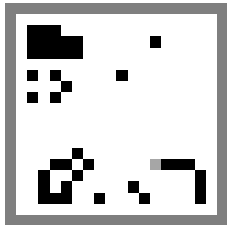
Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.5 & 4.0 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 4.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 2.5$$

Global taskGiven: Two static binary images P_1 (mask) and P_2 (marker)Input: P_1 Initial state: P_2

Boundary condition: Zero-flux

Output: Binary image representing those objects of P_1 which are marked by P_2 .**Examples**

P_1	P_2	Output
 figdel.png	 figrec.png	

1.1.86 RIGHTBC

Right (diagonal) contour detection (#).

Old names:

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & -1.0 \end{bmatrix} \quad z = -2.0$$

Global task

Given:

Input:

Initial state:

Boundary condition:

Output:

1.1.87 RIGHTCON

Right contour detector.

Old names: RightContourDetector, RightEdgeDetection, rightcon



Available in: Template Library v3.1, Candy

$$A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad B = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & -1.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = -2.0$$

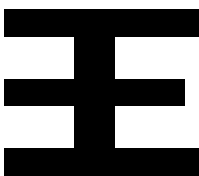
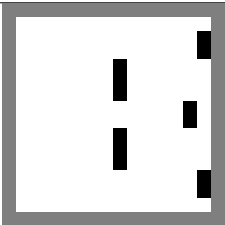
Global task

Given: Static binary image P
Input: P
Initial state: P
Boundary condition: Fixed(0)
Output: Binary image representing the right edges of objects in P

Examples

Input	Output
 a_letter.png	

Examples

Input	Output
 chinese.png	

1.1.88 RotationDetector

Detects the rotation of compact objects in a binary image, having only horizontal and vertical edges, removes all inclined objects or objects having at least one inclined edge [61]

Old names:

Available in: Template Library v3.1

$$\mathbf{A} = \begin{bmatrix} -0.8 & 5.0 & -0.8 \\ 5.0 & 5.0 & 5.0 \\ -0.8 & 5.0 & -0.8 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} -0.4 & -2.5 & -0.4 \\ -2.5 & 5.0 & -2.5 \\ -0.4 & -2.5 & -0.4 \end{bmatrix} \quad z = -11.2$$

Global task

Given: static binary image **P**

Input: **P**

Initial state: **P**

Boundary condition: Fixed(0)

Output: Binary image which retains from the initial state **P** only the compact objects with horizontal or vertical edges

1.1.89 SHADMASK

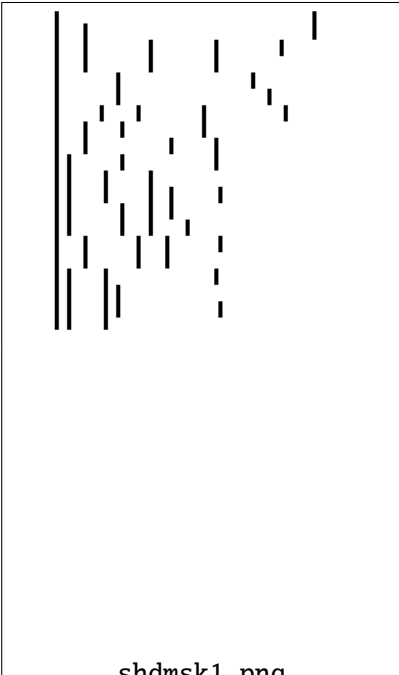
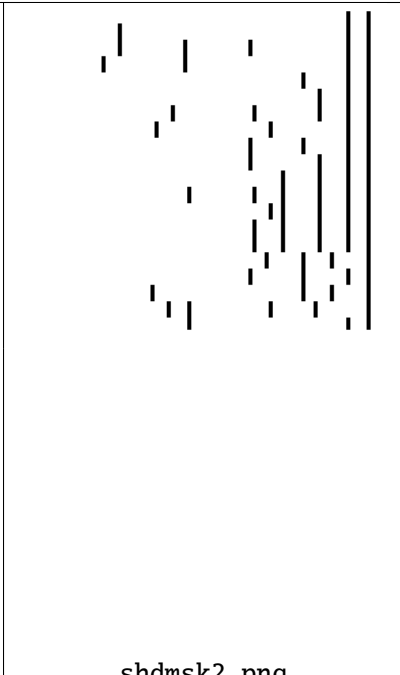
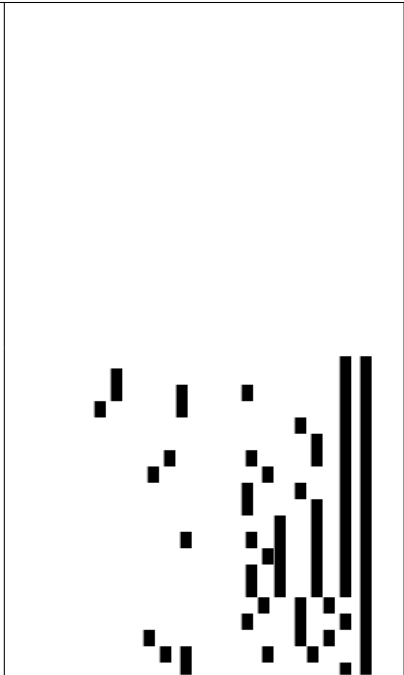
Masked shadow [24]
Old names: MaskedShadow, shadmask, MASKSHAD
Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.8 & 1.5 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & -1.2 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0.0$$

Global task

Given: Static binary images P_1 and P_2
Input: P_1
Initial state: P_2
Boundary condition: Fixed(-1)
Output: Binary image representing the result of pattern propagation of P_2 in a particular direction. The propagation goes from the direction of the non-zero off-center feedback template entry and is halted by the mask P_1 .

Examples

P_1	P_2	Output
 shdmsk1.png	 shdmsk2.png	

1.1.90 SHADOW

Creates the left shadow of the object.

Old names: LeftShadow, ShadowProjection, shadow

Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 2.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0.0$$

Global task

Given: Static binary image P



Input: P

Initial state: 1

Boundary condition: Fixed(0)

Output: Binary image representing the left shadow of the objects in P

Examples

Input	Output
 <p>a_letter.png</p>	

1.1.91 shadow0

Generate growing shadows starting from black points

Old names: DirectedGrowingShadow

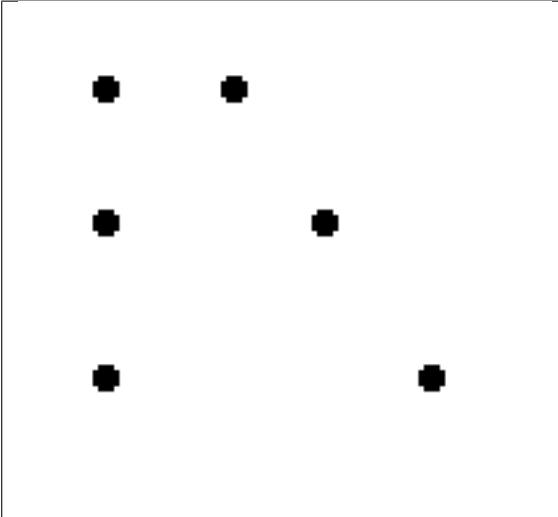
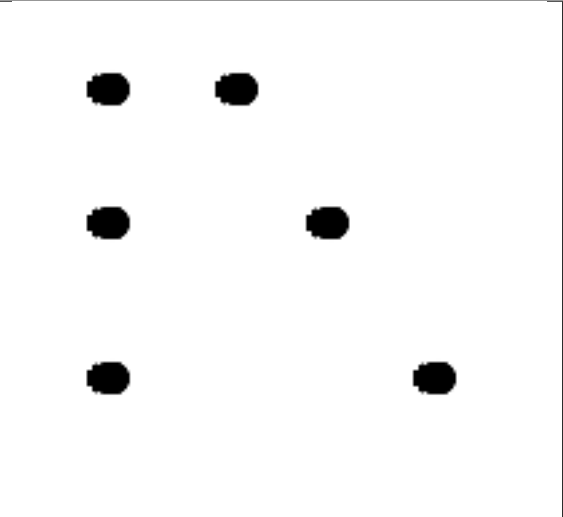
Available in: Template Library v3.1

$$A = \begin{bmatrix} 0.4 & 0.3 & 0.0 \\ 1.0 & 2.0 & -1.0 \\ 0.4 & 0.3 & 0.0 \end{bmatrix} \quad B = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.4 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 2.5$$

Global task

Given: static binary image P
Input: P
Initial state: P
Boundary condition: Fixed(-1)
Output: Binary image in which shadows are generated starting from black pixels. During the transient shadows become wider and wider.

Examples

Input	Output
 <p>points.png</p>	

1.1.92 shadow45

Generate growing shadows starting from black points

Old names: DirectedGrowingShadow

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & -1 \\ 1 & 2 & 0 \\ 1 & 1 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.4 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 2.5$$

Global task

Given: static binary image \mathbf{P}



Input: \mathbf{P}

Initial state: \mathbf{P}

Boundary condition: Fixed(-1)

Output: Binary image in which shadows are generated starting from black pixels. During the transient shadows become wider and wider.

Examples

Input	Output
 <p>A_LETTER.png</p>	



1.1.93 SHADSIM*Vertical shadow template*

Old names: VerticalShadow, shadsim, SUPSHAD

Available in: Template Library v3.1, Candy

$$A = \begin{bmatrix} 0.0 & 1.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \end{bmatrix} \quad B = [0] \quad z = 2.0$$

Global taskGiven: Static binary image **P***Input:* **P***Initial state:* **P***Boundary condition:* Zero-flux*Output:* Binary image representing the vertical shadow of the objects in **P** taken upward and downward simultaneously.**Examples**

Input	Output
 <p>A_LETTER.png</p>	

1.1.94 SHIFTE

Shifts the image toward eastern direction.

Old names:

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0$$

Global task

Given: Static binary image P



Input: P

Initial state: 0

Boundary condition: Zero-flux

Output: Binary image - P is shifted toward the eastern direction by one pixel.

Examples

Input	Output
 <p>A_LETTER.png</p>	

1.1.95 SHIFTN

Shifts the image toward northern direction.

Old names:



Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \end{bmatrix} \quad z = 0$$

Global task

- Given: Static binary image **P**
Input: **P**
Initial state: 0
Boundary condition: Zero-flux
Output: Binary image - **P** is shifted toward the northern direction by one pixel.

Examples

Input	Output
 A_LETTER.png	

1.1.96 SHIFTNE

Shifts the image toward north-estern direction.

Old names:



Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0$$

Global task

Given: Static binary image P
Input: P
Initial state: 0
Boundary condition: Zero-flux
Output: Binary image - P is shifted toward the north-estern direction by one pixel.

Examples

Input	Output
 A_LETTER.png	

1.1.97 SHIFTNW

Shifts the image toward north-western direction.

Old names:

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \quad z = 0$$

Global task

Given: Static binary image **P**



Input: **P**

Initial state: 0

Boundary condition: Zero-flux

Output: Binary image - **P** is shifted toward the north-western direction by one pixel.

Examples

Input	Output
 <p>A_LETTER.png</p>	

1.1.98 SHIFTS

Shifts the image toward southern direction.

Old names:



Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0$$

Global task

Given: Static binary image **P**
Input: **P**
Initial state: 0
Boundary condition: Zero-flux
Output: Binary image - **P** is shifted toward the southern direction by one pixel.

Examples

Input	Output
 A_LETTER.png	

1.1.99 SHIFTSE

Shifts the image toward south-estern direction.

Old names:



Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0$$

Global task

Given: Static binary image P
Input: P
Initial state: 0
Boundary condition: Zero-flux
Output: Binary image - P is shifted toward the south-estern direction by one pixel.

Examples

Input	Output
 A_LETTER.png	

1.1.100 SHIFTSW

Shifts the image toward south-western direction.

Old names:



Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0$$

Global task

Given: Static binary image P
Input: P
Initial state: 0
Boundary condition: Zero-flux
Output: Binary image - P is shifted toward the south-western direction by one pixel.

Examples

Input	Output
 A_LETTER.png	

1.1.101 SHIFTW

Shifts the image toward western direction.

Old names:

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0$$

Global task

Given: Static binary image \mathbf{P}



Input: \mathbf{P}

Initial state: 0

Boundary condition: Zero-flux

Output: Binary image - \mathbf{P} is shifted toward the western direction by one pixel.

Examples

Input	Output
 <p>A_LETTER.png</p>	

1.1.102 SKELBW1

The algorithm finds the skeleton of a black-and-white object.

Old names: BlackandWhiteSkeletonization(Algorithm!)

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1.0 & 1.0 & 0.0 \\ 1.0 & 7.0 & -1.0 \\ 0.0 & -1.0 & 0.0 \end{bmatrix} \quad z = -3.0$$

Global task

Given:	Static binary image P
Input:	P
Initial state:	Arbitrary(0)
Boundary condition:	Zero-flux
Output:	Binary image (Phase 1 of the skeletonization algorithm).

1.1.103 SKELBW2

The algorithm finds the skeleton of a black-and-white object.

Old names: BlackandWhiteSkeletonization(Algorithm!)

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 0.0 & 7.0 & 0.0 \\ -0.5 & -1.0 & -0.5 \end{bmatrix} \quad z = -3.4$$

Global task

Given: Static binary image **P**

Input: **P**

Initial state: Arbitrary(0)

Boundary condition: Zero-flux

Output: Binary image (Phase 2 of the skeletonization algorithm).

1.1.104 SKELBW3

The algorithm finds the skeleton of a black-and-white object.

Old names: BlackandWhiteSkeletonization(Algorithm!)

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 1.0 & 1.0 \\ -1.0 & 7.0 & 1.0 \\ 0.0 & -1.0 & 0.0 \end{bmatrix} \quad z = -3.0$$

Global task

Given: Static binary image **P**
 Input: **P**
 Initial state: Arbitrary(0)
 Boundary condition: Zero-flux
 Output: Binary image (Phase 3 of the skeletonization algorithm).

1.1.105 SKELBW4

The algorithm finds the skeleton of a black-and-white object.

Old names: BlackandWhiteSkeletonization(Algorithm!)

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} -0.5 & 0.0 & 1.0 \\ -1.0 & 7.0 & 1.0 \\ -0.5 & 0.0 & 1.0 \end{bmatrix} \quad z = -3.4$$

Global task

Given:	Static binary image P
Input:	P
Initial state:	Arbitrary(0)
Boundary condition:	Zero-flux
Output:	Binary image (Phase 4 of the skeletonization algorithm).

1.1.106 SKELBW5

The algorithm finds the skeleton of a black-and-white object.

Old names: BlackandWhiteSkeletonization(Algorithm!)

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & -1.0 & 0.0 \\ -1.0 & 7.0 & 1.0 \\ 0.0 & 1.0 & 1.0 \end{bmatrix} \quad z = -3.0$$

Global task

Given: Static binary image **P**

Input: **P**

Initial state: Arbitrary(0)

Boundary condition: Zero-flux

Output: Binary image (Phase 5 of the skeletonization algorithm).

1.1.107 SKELBW6

The algorithm finds the skeleton of a black-and-white object.

Old names: BlackandWhiteSkeletonization(Algorithm!)

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} -0.5 & -1.0 & -0.5 \\ 0.0 & 7.0 & 0.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix} \quad z = -3.4$$

Global task

Given: Static binary image **P**

Input: **P**

Initial state: Arbitrary(0)

Boundary condition: Zero-flux

Output: Binary image (Phase 6 of the skeletonization algorithm).

1.1.108 SKELBW7

The algorithm finds the skeleton of a black-and-white object.

Old names: BlackandWhiteSkeletonization(Algorithm!)

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & -1.0 & 0.0 \\ 1.0 & 7.0 & -1.0 \\ 1.0 & 1.0 & 0.0 \end{bmatrix} \quad z = -3.0$$

Global task

Given: Static binary image **P**

Input: **P**

Initial state: Arbitrary(0)

Boundary condition: Zero-flux

Output: Binary image (Phase 7 of the skeletonization algorithm).

1.1.109 SKELBW8

The algorithm finds the skeleton of a black-and-white object.

Old names: BlackandWhiteSkeletonization(Algorithm!)

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1.0 & 0.0 & -0.5 \\ 1.0 & 7.0 & -1.0 \\ 1.0 & 0.0 & -0.5 \end{bmatrix} \quad z = -3.4$$

Global task

Given: Static binary image **P**

Input: **P**

Initial state: Arbitrary(0)

Boundary condition: Zero-flux

Output: Binary image (Phase 8 of the skeletonization algorithm).

1.1.110 SpikeGeneration1

Rhythmic burst-like spike generation using 4 ion channels, 2 of them are delayed

Old names: SPIKE_BU

Available in: Template Library v3.1

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0.0$$

Global task

Given:

Input:

Initial state:

Boundary condition:

Output:

1.1.111 SpikeGeneration2

Action potential generation in a neuromorphic way without delay using 2 ion channels

Old names: SPIKE_N

Available in: Template Library v3.1

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0.0$$

Global task

Given:

Input:

Initial state:

Boundary condition:

Output:

1.1.112 SpikeGeneration3

Action potential generation in a neuromorphic way, using 2 ion channels where one is delayed.

Ion channels are modeled with voltage-controlled conductance (VCC) templates

Old names: SPIKE_ND

Available in: Template Library v3.1

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0.0$$

Global task

Given:

Input:

Initial state:

Boundary condition:

Output:

1.1.113 T1_RACC3*Textures detection.*

Old names: TextureDetector1

Available in: Template Library v3.1, Candy

$$A = \begin{bmatrix} 2.2656 & 1.7969 & 3.3594 \\ -0.7031 & -4.4531 & 1.4063 \\ 3.2031 & 3.9844 & -0.3125 \end{bmatrix} \quad B = \begin{bmatrix} -3.9063 & 1.25 & 3.0469 \\ 0.8594 & -3.0469 & 3.3594 \\ 1.7188 & -0.625 & -4.6094 \end{bmatrix} \quad z = -1.6406$$

Global task

Given: Static gray-scale image **P** representing textures having the same flat grayscale histograms

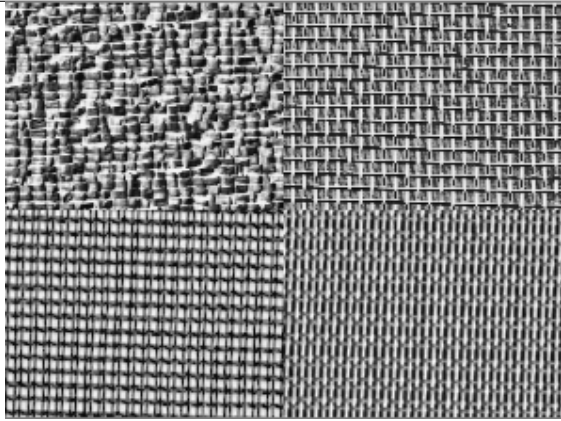
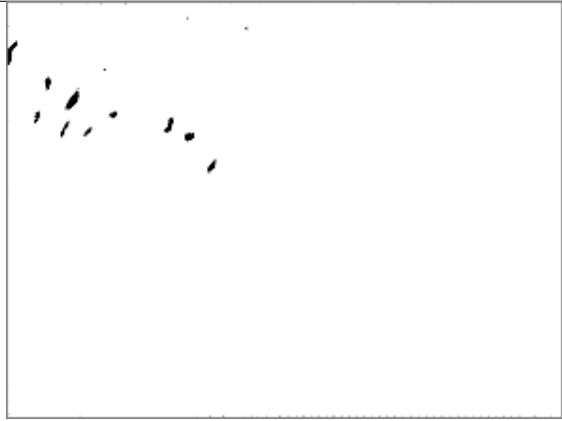
Input: **P**

Initial state: **P**

Boundary condition: Fixed(0)

Output: Nearly binary image where the detected texture becomes darker than the others.

Examples

Input	Output
 <p>tx_racc.png</p>	

1.1.114 T2_RACC3

Textures detection.

Old names: TextureDetector2

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 1.5625 & 4.375 & 2.4219 \\ 4.6875 & -3.125 & 1.4063 \\ 2.1875 & -5.0 & 0.8594 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} -2.8125 & 2.4219 & -3.75 \\ -5.0 & -0.3906 & -5.0 \\ 3.6719 & 4.2188 & 3.125 \end{bmatrix} \quad z = -3.2031$$

Global task

Given: Static gray-scale image **P** representing textures having the same flat grayscale histograms

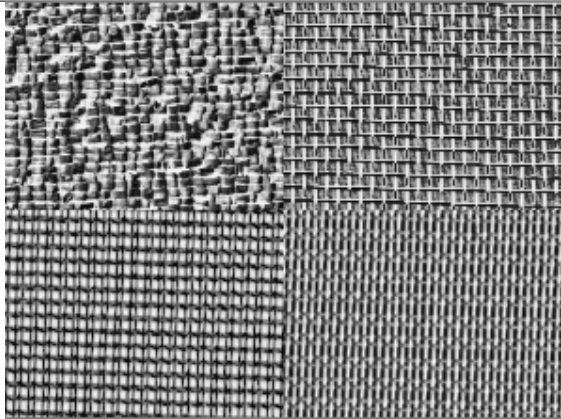
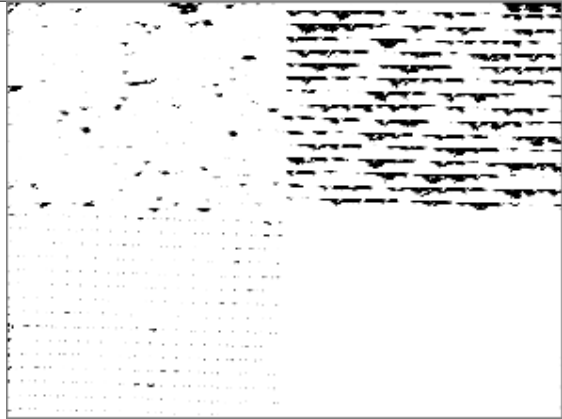
Input: **P**

Initial state: **P**

Boundary condition: Fixed(0)

Output: Nearly binary image where the detected texture becomes darker than the others.

Examples

Input	Output
 tx_racc.png	

1.1.115 T3_RACC3*Textures detection.*

Old names: TextureDetector3

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 1.6406 & -1.0156 & 1.3281 \\ 1.875 & -4.6094 & 2.8906 \\ 3.2813 & 2.0313 & 3.75 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} -3.9063 & -2.6563 & -3.125 \\ 0.9375 & 1.4844 & -3.125 \\ 1.3281 & 0.5469 & 2.3438 \end{bmatrix} \quad z = -2.4219$$

Global task

Given: Static gray-scale image **P** representing textures having the same flat grayscale histograms

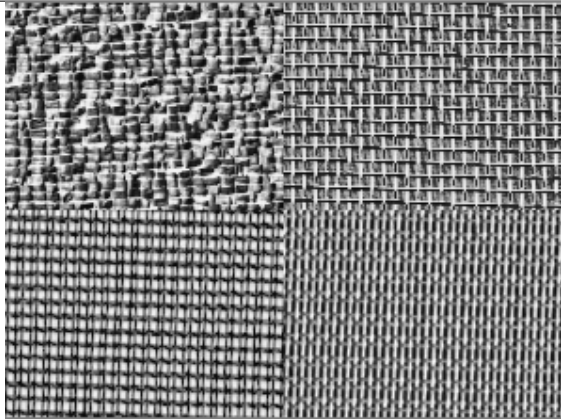

Input: **P**

Initial state: **P**

Boundary condition: Fixed(0)

Output: Nearly binary image where the detected texture becomes darker than the others.

Examples

Input	Output
 <p>tx_racc.png</p>	

1.1.116
T4_RACC3

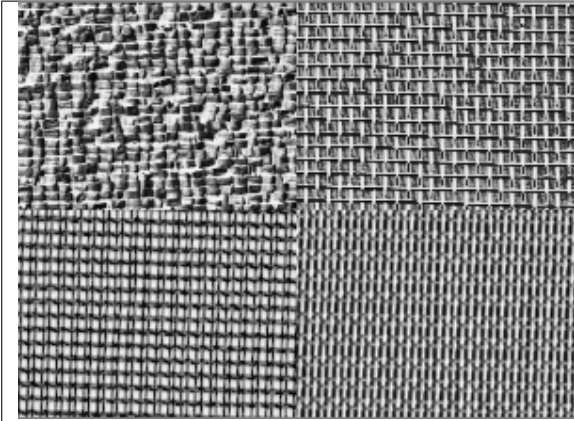
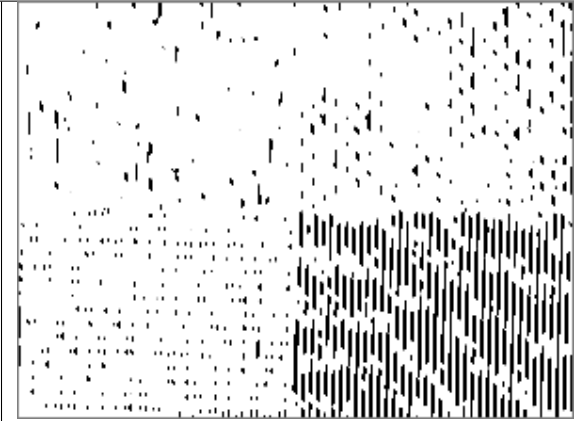
Textures detection.
Old names: TextureDetector4
Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 3.125 & 4.2969 & 2.1875 \\ -2.8125 & 3.125 & 0.1563 \\ 1.875 & 4.9219 & 4.5313 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} -3.5156 & 4.375 & -5.0 \\ -0.9375 & -3.0469 & -3.6719 \\ 1.4063 & -0.625 & -4.375 \end{bmatrix} \quad z = -2.4219$$

Global task

Given: Static gray-scale image **P** representing textures having the same flat grayscale histograms
Input: **P**
Initial state: **P**
Boundary condition: Fixed(0)
Output: Nearly binary image where the detected texture becomes darker than the others.

Examples

Input	Output
 <p>tx_racc.png</p>	

1.1.117 ThinLineRemover

Removes thin (one-pixel thick) lines from a binary image

Old names:

Available in: Template Library v3.1

$$\mathbf{A} = \begin{bmatrix} 2.0 & 2.0 & 2.0 \\ 2.0 & 8.0 & 2.0 \\ 2.0 & 2.0 & 2.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = -2.0$$

Global task

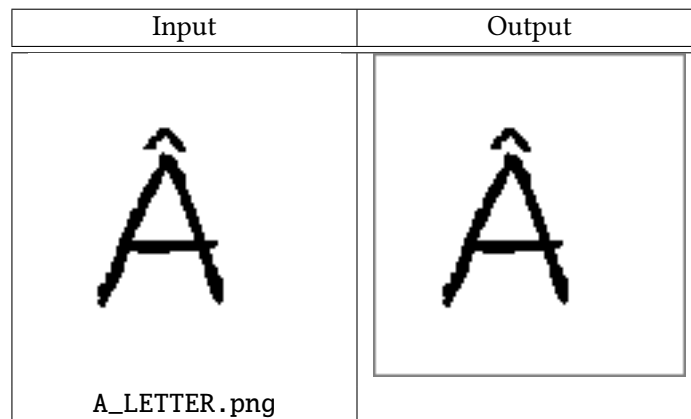
Given: static binary image **P**

Input: Arbitrary(0)

Initial state: **P**

Boundary condition: Fixed(0)

Output: Binary image containing compact black objects (without any thin lines) against a white background

Examples

1.1.118 TX_HCLC

Segmentation of four textures.

Old names: 5x5TextureSegmentation1, tx_hclc

Available in: Template Library v3.1, Candy

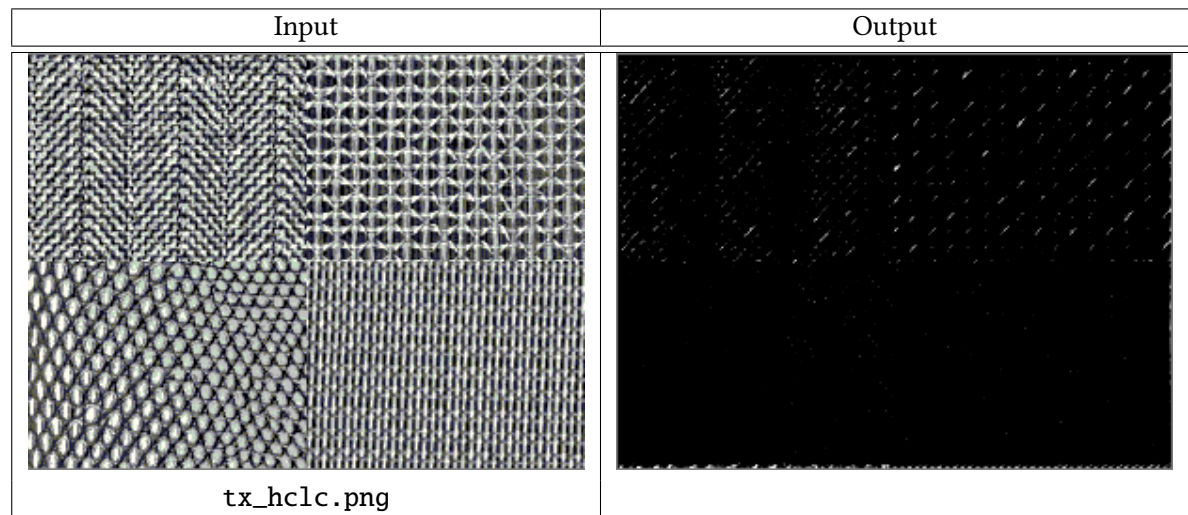
$$\mathbf{A} = \begin{bmatrix} -3.4375 & 0.8594 & -1.6406 & -0.1563 & -1.0156 \\ -1.0938 & 0.1563 & -2.1875 & -3.2031 & 3.5156 \\ 2.5 & 1.5625 & 3.9063 & 2.6563 & 2.4219 \\ 0.5469 & 2.8906 & -0.625 & 0.4688 & 3.6719 \\ -1.7969 & -0.5469 & 2.5 & -0.2344 & 2.3438 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} -2.1875 & -0.2344 & 0.1563 & -0.625 & -0.7813 \\ 1.6406 & 2.2656 & -3.2031 & 1.0938 & 2.0313 \\ 0.0781 & 0.5469 & 0.8594 & 3.5156 & 0.0781 \\ 0.3906 & -3.8281 & -3.125 & -2.3438 & -2.1094 \\ 0.7813 & -2.6563 & -1.1719 & -1.4063 & 1.0156 \end{bmatrix}$$

$$z = 3.2813$$

Global task

Given: Static gray-scale image **P** representing four textures
 Input: **P**
 Initial state: **P**
 Boundary condition: Fixed(0)
 Output: Nearly binary image representing four patterns that differ in average gray-levels.

Examples

1.1.119 TX_RACC3*Segmentation of four textures.*

Old names: 3x3TextureSegmentation

Available in: Candy

$$A = \begin{bmatrix} 0.8594 & 0.9375 & 3.75 \\ 2.1094 & -2.8125 & 3.75 \\ -1.3281 & -2.5781 & -1.0156 \end{bmatrix} \quad B = \begin{bmatrix} 0.1563 & -1.5625 & 1.25 \\ -2.8906 & 1.0938 & -3.2031 \\ 4.0625 & 4.6875 & 3.75 \end{bmatrix} \quad z = 1.7969$$

Global task

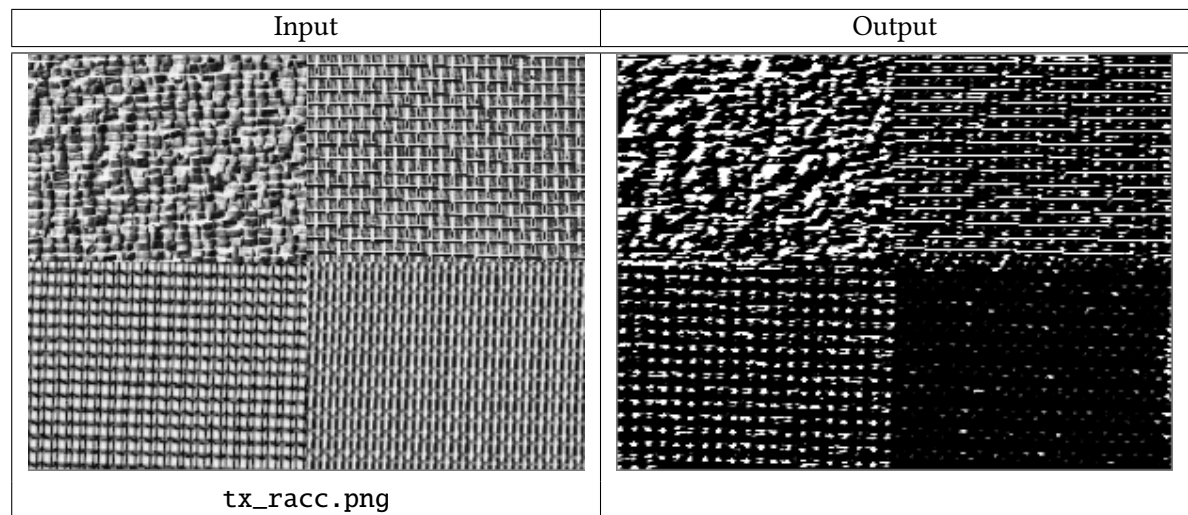
Given: Static gray-scale image **P** representing four textures

Input: **P**

Initial state: **P**

Boundary condition: Fixed(0)

Output: Nearly binary image representing four patterns that differ in average gray-levels.

Examples

1.1.120 TX_RACC5

Segmentation of four textures.

Old names: 5x5TextureSegmentation2, tx_racc5

Available in: Template Library v3.1, Candy

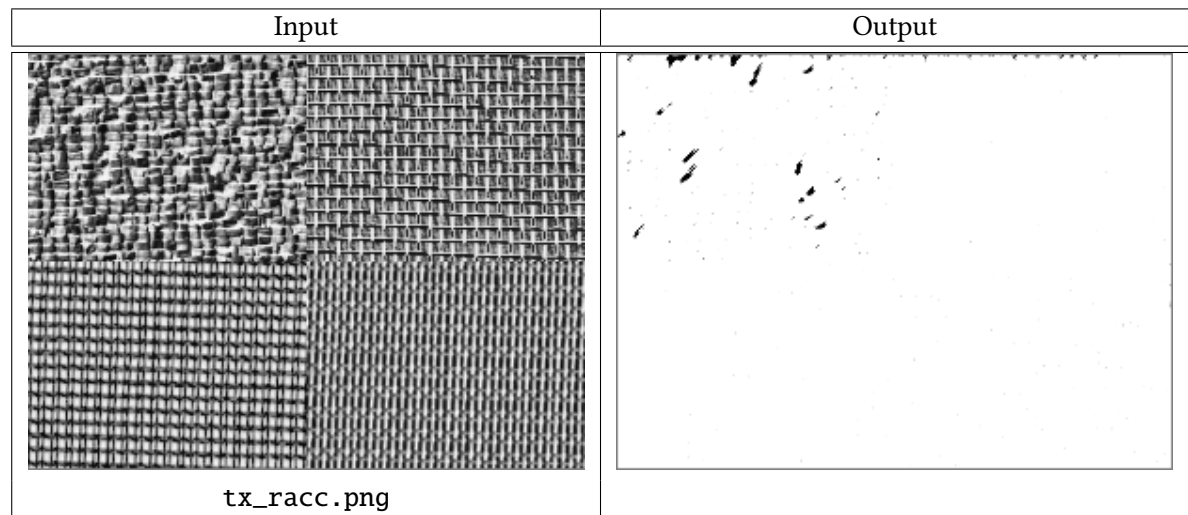
$$A = \begin{bmatrix} 4.2188 & -1.5625 & 1.5625 & 3.3594 & 0.625 \\ -2.8906 & 4.5313 & -0.2344 & 3.125 & -2.8906 \\ 2.6563 & 2.1875 & -4.6875 & -3.4375 & -2.8125 \\ 3.9844 & 1.5625 & -1.1719 & -3.125 & -3.2031 \\ -3.75 & -2.1875 & 3.2813 & 2.1875 & -0.625 \end{bmatrix}$$

$$B = \begin{bmatrix} 4.0625 & -5.0 & 0.3906 & 2.1094 & -1.875 \\ 3.9063 & 0.3125 & -1.9531 & 4.8438 & -0.3125 \\ 0.0 & -4.0625 & 0.9375 & -0.3125 & 0.4688 \\ -0.625 & -5.0 & 2.3438 & 0.625 & -1.875 \\ 3.5938 & -0.9375 & 0.1563 & 2.8125 & -1.875 \end{bmatrix}$$

$$z = -5.0$$

Global task

Given: Static gray-scale image **P** representing four textures
 Input: **P**
 Initial state: **P**
 Boundary condition: Fixed(0)
 Output: Nearly binary image representing four patterns that differ in average gray-levels.

Examples


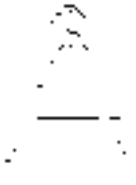
1.1.121 VERSKELB*Vertical skeleton from the bottom.*

Old names: VerSkelB

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 3.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.125 & -0.5 & 0.125 \\ 0.0 & 0.5 & 0.0 \\ 0.5 & -0.5 & 0.5 \end{bmatrix} \quad z = -1.0$$

Global taskGiven: Static binary image **P***Input:* **P***Initial state:* Arbitrary(0)*Boundary condition:* Fixed(0)*Output:* Binary image, peeling the black pixels from the bottom of the object.**Examples**

Input	Output
 <p>A_LETTER.png</p>	

1.1.122 VERSKELT*Vertical skeleton from the top.*

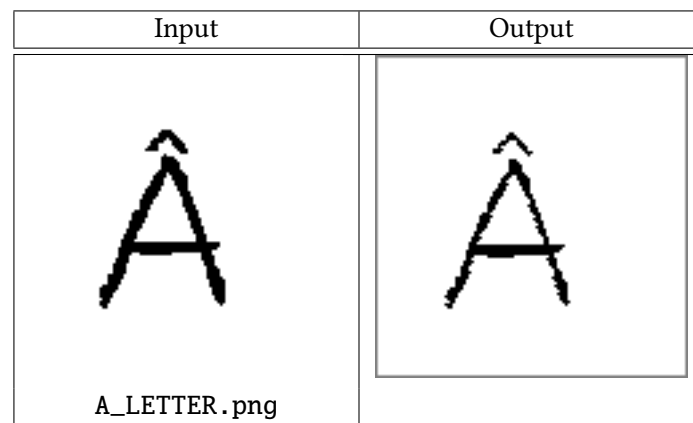
Old names: VerSkelT

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 3.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.0 & 0.5 & 0.0 \\ 0.125 & -0.5 & 0.125 \end{bmatrix} \quad z = -1.0$$

Global task

Given: Static binary image **P**
 Input: **P**
 Initial state: Arbitrary(0)
 Boundary condition: Fixed(0)
 Output: Binary image, peeling the black pixels from the top of the object.

Examples

1.1.123 WhitePropagation

Starts omni-directional white propagation from white pixels [54]

Old names: wprop

Available in: Template Library v3.1

$$\mathbf{A} = \begin{bmatrix} 0.25 & 0.25 & 0.25 \\ 0.25 & 3.0 & 0.25 \\ 0.25 & 0.25 & 0.25 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$z = -3.75$$

Global task

Given:

static binary image **P**

Input:

Arbitrary(0)

Initial state:

P

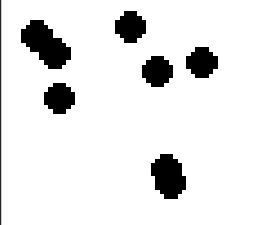
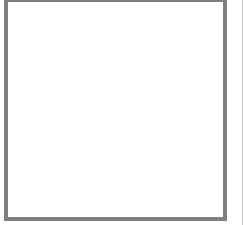
Boundary condition:

Fixed(0)

Output:

Binary image showing white objects in **P** with increasing white neighborhood (black objects decreasing in size).

Examples

Input	Output
 patches.png	

1.2 Stretch type templates

Definition

$$\mathbf{A} = 0, \quad \mathbf{B} \neq 0, \quad r(\mathbf{B}) = 0$$

Input: Grayscale

Output: Grayscale

1.2.1 STRETCH

"Contrast stretching".

Old names:

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0.0$$

Global task

Given: Static (noisy) gray-scale image \mathbf{P}

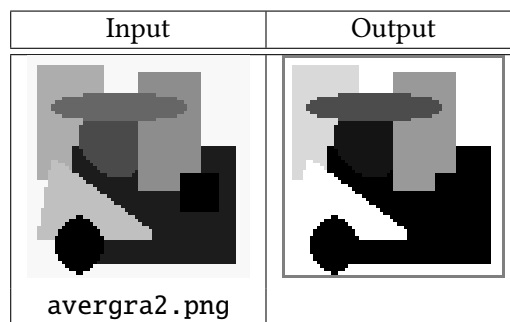
Input: \mathbf{P}

Initial state: -

Boundary condition: 0

Output: Grayscale image.

Examples



1.3 Convolution type templates

Definition

$$A = 0, \quad B \neq 0, \quad r(B) = 1$$

Input: Grayscale

Output: Grayscale

1.3.1 CONVOL

Convolution (linear averaging) in nearest neighborhood.

Old names:

Available in: Candy

$$A = \begin{bmatrix} 0 \end{bmatrix} \quad B = \begin{bmatrix} 0.11 & 0.11 & 0.11 \\ 0.11 & 0.11 & 0.11 \\ 0.11 & 0.11 & 0.11 \end{bmatrix} \quad z = 0.0$$

Global task

Given: Static (noisy) gray-scale image **P**

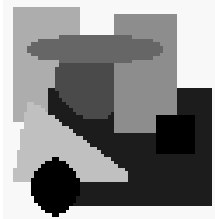

Input: **P**

Initial state: -

Boundary condition: zero-flux

Output: Grayscale image.

Examples

Input	Output
 <p>avergra2.png</p>	

1.3.2 optimedge

Optimal edge detector [43]

Old names: `OptimalEdgeDetector`

Available in: Template Library v3.1

$$A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad B = \begin{bmatrix} -0.11 & 0.0 & 0.11 \\ -0.28 & 0.0 & 0.28 \\ -0.11 & 0.0 & 0.11 \end{bmatrix} \quad z = 0.0$$

Global task

Given: static grayscale image P


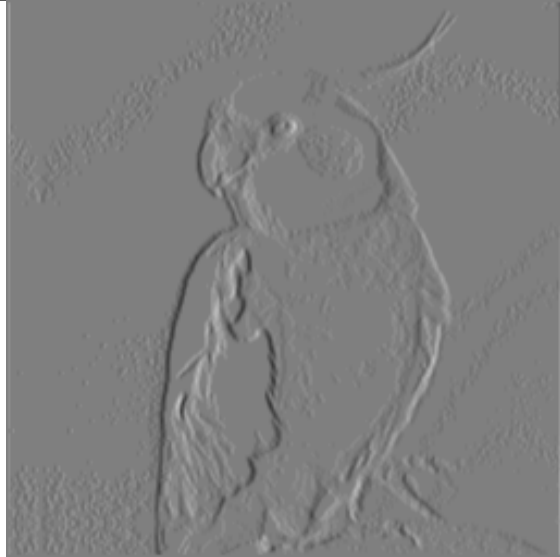
Input: P

Initial state: Arbitrary(0)

Boundary condition: Zero-flux

Output: Grayscale image representing edges calculated in horizontal direction.

Examples

Input	Output
 <p>bird.png</p>	

1.4 Threshold type templates

Definition

$$A \neq 0, \quad r(A) = 0, \quad B = 0, \quad z \neq 0$$

Input: Grayscale

Output: Binary

1.4.1 FILBLACK

Drives the whole network into black

Old names: BlackFiller, BLACK, filblack

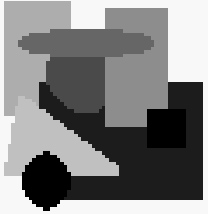
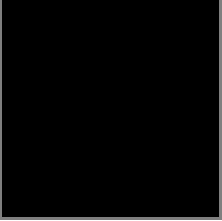
Available in: Template Library v3.1, Candy

$$A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad B = \begin{bmatrix} 0 \end{bmatrix} \quad z = 4.0$$

Global task

Given: Static gray-scale image **P**
Input: Arbitrary(0)
Initial state: **P**
Boundary condition: Fixed(0)
Output: Binary (black) image.

Examples

Input	Output
 avergra2.png	

1.4.2 FILWHITE

Drives the whole network into white

Old names: WhiteFiller, WHITE, filwhite

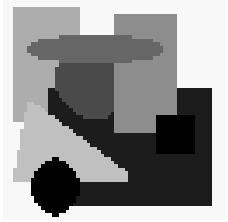
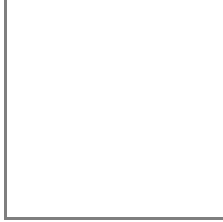
Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 \end{bmatrix} \quad z = -4.0$$

Global task

Given: Static gray-scale image \mathbf{P}
 Input: Arbitrary(0)
 Initial state: \mathbf{P}
 Boundary condition: Fixed(0)
 Output: Binary (white) image.

Examples

Input	Output
 avergra2.png	

1.4.3 THRES

Grayscale to binary threshold template

Old names: Threshold

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = -0.4$$

Global task

Given: Static gray-scale image **P**

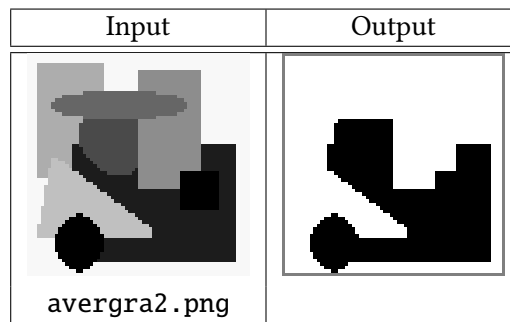
Input: Arbitrary(0)

Initial state: **P**

Boundary condition: Fixed(0)

Output: Binary image where black pixels correspond to pixels in **P** with grayscale intensity above the given threshold.

Examples



1.5 Erosion type templates

Definition

$$A = 0, \quad B \neq 0, \quad r(B) = 1, \quad z \neq 0$$

Input: Binary

Output: Binary

1.5.1 EROSION

Binary erosion.

Old names:

Available in: Candy

$$A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad B = \begin{bmatrix} 0.0 & 1.0 & 0.0 \\ 0.0 & 1.0 & 1.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = -2.0$$

Global task

Given: Static binary image **P**

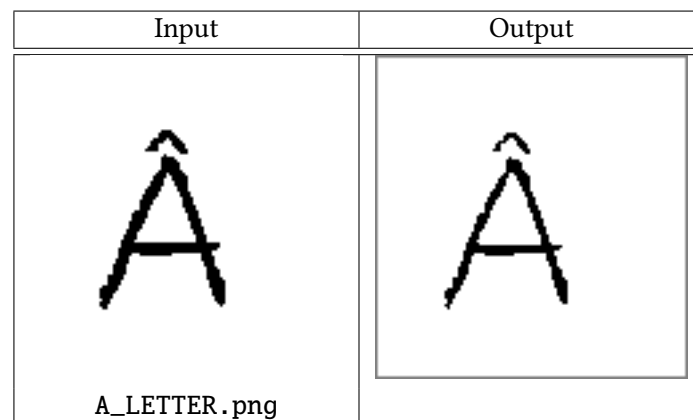
Input: **P**

Initial state: Arbitrary(0)

Boundary condition: Fixed(0)

Output: Binary image representing the result of the erosion operation.

Examples



1.5.2 TEXTUDIL

Dilation (algo#).

Old names:

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 1.0 & 0.0 \\ 1.0 & 1.0 & 1.0 \\ 0.0 & 1.0 & 0.0 \end{bmatrix} \quad z = 4.0$$

Global task

Given:

Input:

Initial state:

Boundary condition:

Output:

1.5.3 TEXTUERO

Erosion (algo#).

Old names:

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 1.0 & 0.0 \\ 1.0 & 1.0 & 1.0 \\ 0.0 & 1.0 & 0.0 \end{bmatrix} \quad z = -4.0$$

Global task

Given:

Input:

Initial state:

Boundary condition:

Output:

1.6 Diffusion type templates

Definition

$$\mathbf{A} \neq 0, \quad r(\mathbf{A}) = 1, \quad \mathbf{B} = 0, \quad z = 0$$

Input: Grayscale

Output: Grayscale

1.6.1 DIFFUS

Filtering-reconstruction with heat-diffusion.

Old names: HeatDiffusion.

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.1 & 0.15 & 0.1 \\ 0.15 & 0.0 & 0.15 \\ 0.1 & 0.15 & 0.1 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0.0$$

Global task

Given: Static (noisy) gray-scale image \mathbf{P}


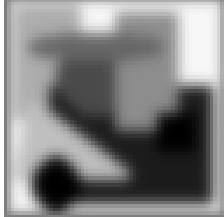
Input: Arbitrary(0)

Initial state: \mathbf{P}

Boundary condition: Fixed(0)

Output: Grayscale image representing the result of the heat diffusion operation.

Examples

Input	Output
 avergra2.png	

1.6.2 DIFFUS2

Filtering-reconstruction with heat-diffusion.

Old names: HeatDiffusion.

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.01 & 0.04 & 0.04 & 0.04 & 0.01 \\ 0.04 & 0.04 & 0.06 & 0.04 & 0.04 \\ 0.04 & 0.06 & 0.08 & 0.06 & 0.04 \\ 0.04 & 0.04 & 0.06 & 0.04 & 0.04 \\ 0.01 & 0.04 & 0.04 & 0.04 & 0.01 \end{bmatrix}$$


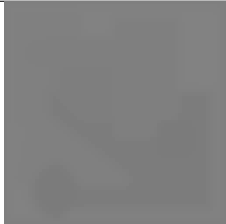
$$\mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$z = 0.0$$

Global task

Given: Static (noisy) gray-scale image **P**
 Input: -
 Initial state: **P**
 Boundary condition: Fixed(0)
 Output: Grayscale image representing the result of the heat diffusion operation.

Examples

Input	Output
 avergra2.png	

1.6.3 DIFFUS3

Filtering-reconstruction with heat-diffusion.

Old names: HeatDiffusion.

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.02 & 0.02 & 0.02 & 0.02 & 0.02 & 0.01 \\ 0.01 & 0.02 & 0.04 & 0.05 & 0.04 & 0.02 & 0.01 \\ 0.01 & 0.02 & 0.05 & 0.06 & 0.05 & 0.02 & 0.01 \\ 0.01 & 0.02 & 0.04 & 0.05 & 0.04 & 0.02 & 0.01 \\ 0.01 & 0.02 & 0.02 & 0.02 & 0.02 & 0.02 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \end{bmatrix}$$



$$\mathbf{B} = \begin{bmatrix} 0 \end{bmatrix}$$

$$z = 0.0$$

Global task

Given: Static (noisy) gray-scale image \mathbf{P}
 Input: -
 Initial state: \mathbf{P}
 Boundary condition: Fixed(0)
 Output: Grayscale image representing the result of the heat diffusion operation.

Examples

Input	Output
 avergra2.png	

1.6.4 LAPLACE

Solves the Laplace PDE ($Dx = 0$).

Old names: LaplacePDESolver

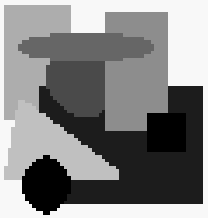
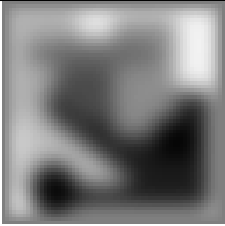
Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 1.0 & 0.0 \\ 1.0 & -3.0 & 1.0 \\ 0.0 & 1.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 \end{bmatrix} \quad z = 0$$

Global task

Given: Static gray-scale image \mathbf{P}
 Input: Arbitrary(0)
 Initial state: \mathbf{P}
 Boundary condition: Zero-flux
 Output: Gray-scale image - the solution of the Laplace equation.

Examples

Input	Output
 avergra2.png	

1.7 Constrained diffusion type templates

Definition

$$\mathbf{A} \neq 0, \quad r(\mathbf{A}) = 1, \quad \mathbf{B} \neq 0, \quad r(\mathbf{B}) = 1, \quad z = 0$$

Input: Grayscale

Output: Grayscale

1.7.1 DIFFUS4

Filtering-reconstruction with constrained heat-diffusion.

Old names: HeatDiffusion.

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 0.01 & 0.02 & 0.02 & 0.02 & 0.01 \\ 0.02 & 0.02 & 0.03 & 0.02 & 0.02 \\ 0.02 & 0.03 & 0.04 & 0.03 & 0.02 \\ 0.02 & 0.02 & 0.03 & 0.02 & 0.02 \\ 0.01 & 0.02 & 0.02 & 0.02 & 0.01 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} 0.01 & 0.02 & 0.02 & 0.02 & 0.01 \\ 0.02 & 0.02 & 0.03 & 0.02 & 0.02 \\ 0.02 & 0.03 & 0.04 & 0.03 & 0.02 \\ 0.02 & 0.02 & 0.03 & 0.02 & 0.02 \\ 0.01 & 0.02 & 0.02 & 0.02 & 0.01 \end{bmatrix}$$

$$z = 0.0$$

Global task

Given: Static (noisy) gray-scale image \mathbf{P}



Input: \mathbf{P}

Initial state: \mathbf{P}

Boundary condition: Fixed(0)

Output: Grayscale image representing the result of the heat diffusion operation.

Examples

Input	Output
 <p>avergra2.png</p>	

1.8 Halftoning type templates

Definition

$$\mathbf{A} \neq 0, \quad r(\mathbf{A}) = 1, \quad \mathbf{B} \neq 0, \quad r(\mathbf{B}) = 1, \quad z = 0$$

Input: Grayscale

Output: Binary

1.8.1 HLF3

3x3 image halftoning

Old names: 3x3Halftoning, hlf3, HLF33

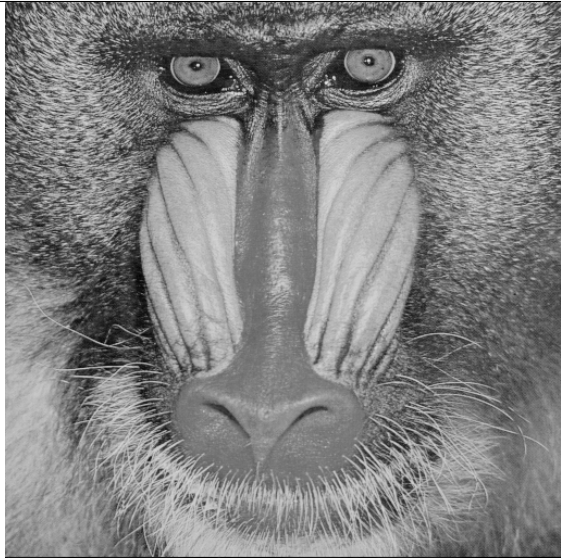
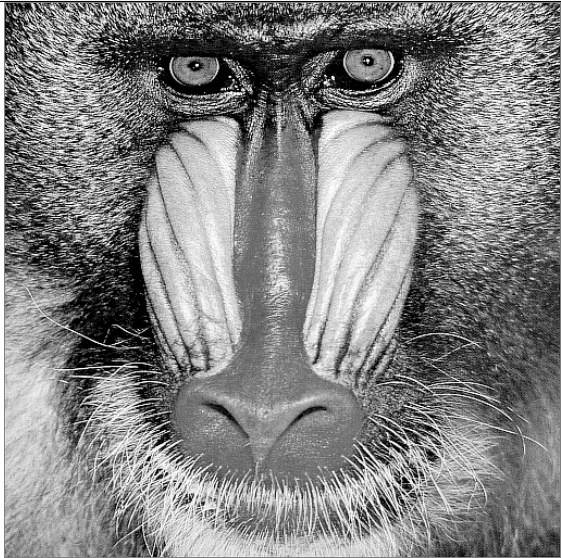
Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} -0.07 & -0.1 & -0.07 \\ -0.1 & 1.03 & -0.1 \\ -0.07 & -0.1 & -0.07 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.07 & 0.1 & 0.07 \\ 0.1 & 0.32 & 0.1 \\ 0.07 & 0.1 & 0.07 \end{bmatrix} \quad z = 0.0$$

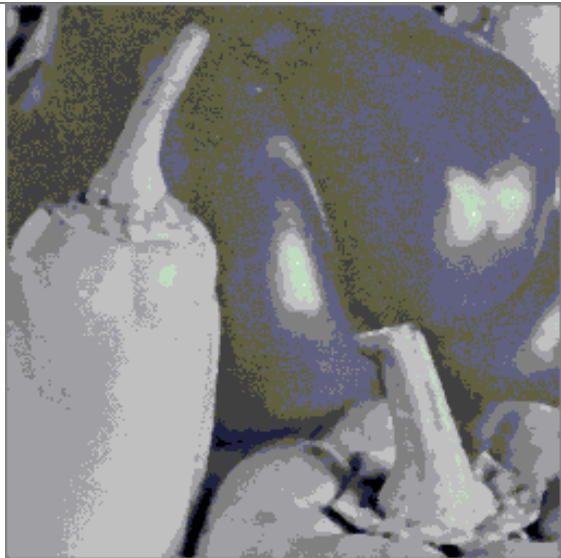

Global task

Given:	Static gray-scale image \mathbf{P}
Input:	\mathbf{P}
Initial state:	\mathbf{P}
Boundary condition:	Fixed(0)
Output:	Binary image preserving the main features of \mathbf{P}

Examples

Input	Output
 baboon.png	

Examples

Input	Output
 peppers.png	

1.8.2 HLF5

5x5 image halftoning

Old names: 5x5Halftoning2, HLF55, hlf5

Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} -0.0245 & -0.07 & -0.099 & -0.07 & -0.0245 \\ -0.07 & -0.324 & -0.46 & -0.324 & -0.07 \\ -0.099 & -0.46 & 1.05 & -0.46 & -0.099 \\ -0.07 & -0.324 & -0.46 & -0.324 & -0.07 \\ -0.0245 & -0.07 & -0.099 & -0.07 & -0.0245 \end{bmatrix}$$

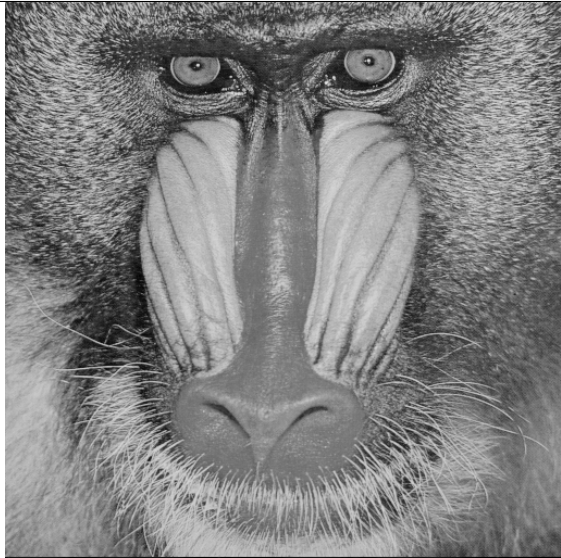
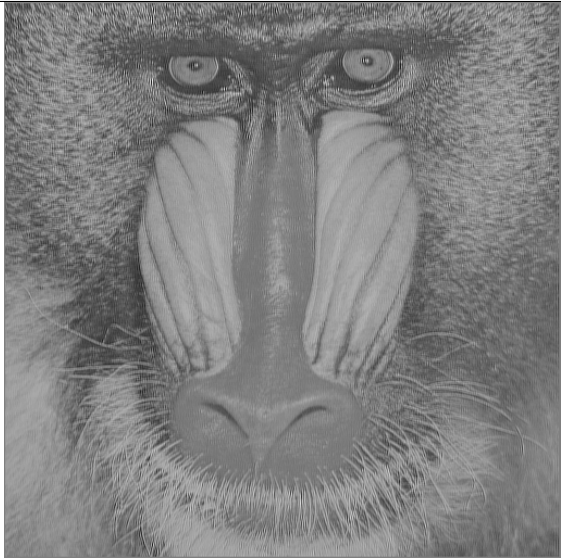
$$\mathbf{B} = \begin{bmatrix} 0.0245 & 0.07 & 0.099 & 0.07 & 0.0245 \\ 0.07 & 0.324 & 0.46 & 0.324 & 0.07 \\ 0.099 & 0.46 & 0.81 & 0.46 & 0.099 \\ 0.07 & 0.324 & 0.46 & 0.324 & 0.07 \\ 0.0245 & 0.07 & 0.099 & 0.07 & 0.0245 \end{bmatrix}$$

$$z = 0.0$$

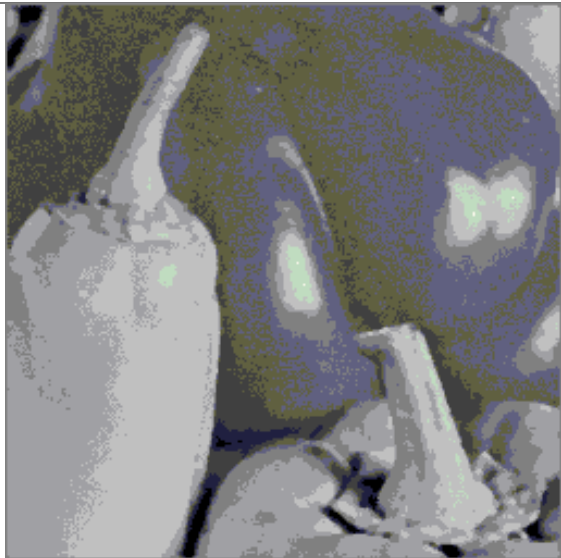

Global task

Given:	Static gray-scale image \mathbf{P}
Input:	\mathbf{P}
Initial state:	\mathbf{P}
Boundary condition:	Fixed(0)
Output:	Binary image preserving the main features of \mathbf{P}

Examples

Input	Output
 baboon.png	

Examples

Input	Output
 peppers.png	

1.8.3 HLF5KC

5x5 image halftoning.

Old names: 5x5Halftoning1, hlf5kc, HLF55_KC

Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} -0.03 & -0.086 & -0.13 & -0.086 & -0.03 \\ -0.086 & -0.359 & -0.604 & -0.359 & -0.086 \\ -0.13 & -0.604 & 1.05 & -0.604 & -0.13 \\ -0.086 & -0.359 & -0.604 & -0.359 & -0.086 \\ -0.03 & -0.086 & -0.13 & -0.086 & -0.03 \end{bmatrix}$$

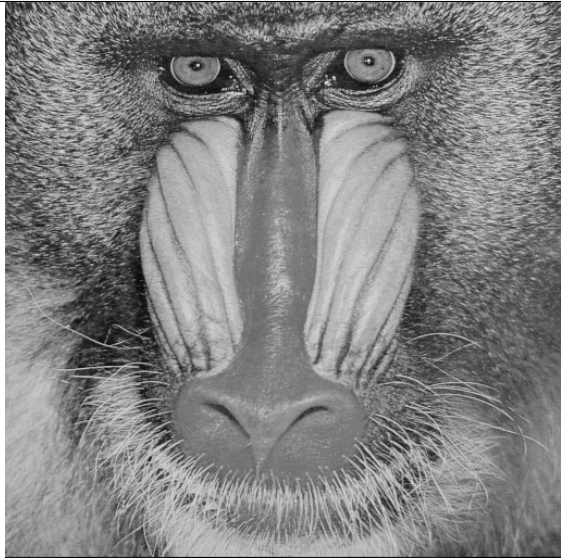
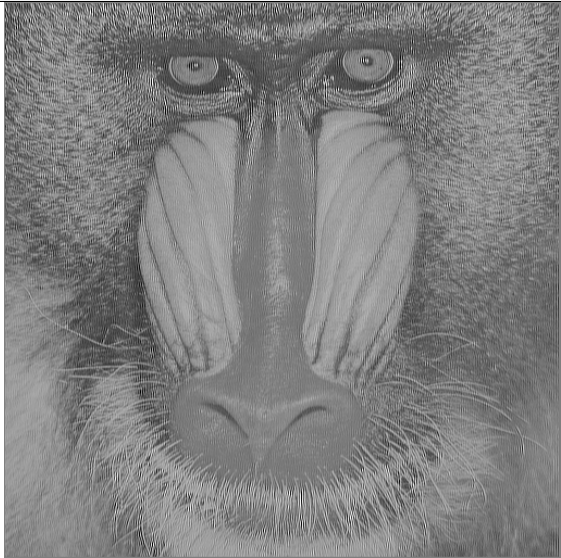
$$\mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.068 & 0.0 & 0.0 \\ 0.0 & 0.355 & 0.756 & 0.355 & 0.0 \\ 0.068 & 0.756 & 2.122 & 0.756 & 0.068 \\ 0.0 & 0.355 & 0.756 & 0.355 & 0.0 \\ 0.0 & 0.0 & 0.068 & 0.0 & 0.0 \end{bmatrix}$$

$$z = 0.0$$

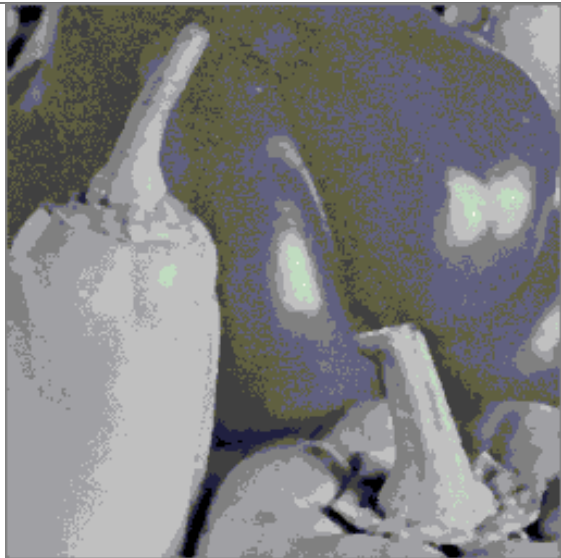

Global task

Given: Static gray-scale image \mathbf{P}
 Input: \mathbf{P}
 Initial state: \mathbf{P}
 Boundary condition: Fixed(0)
 Output: Binary image preserving the main features of \mathbf{P}

Examples

Input	Output
 baboon.png	

Examples

Input	Output
 peppers.png	

1.9 Inverse halftoning type templates

Definition

$$A = 0, \quad B \neq 0, \quad r(B) = 1, \quad z = 0$$

Input: Binary

Output: Grayscale

1.9.1 HERRING

Herring-grid illusion.

Old names: HerringGridIllusion

Available in: Candy

$$A = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

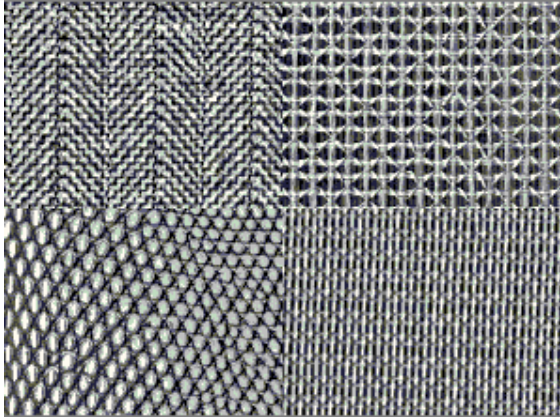
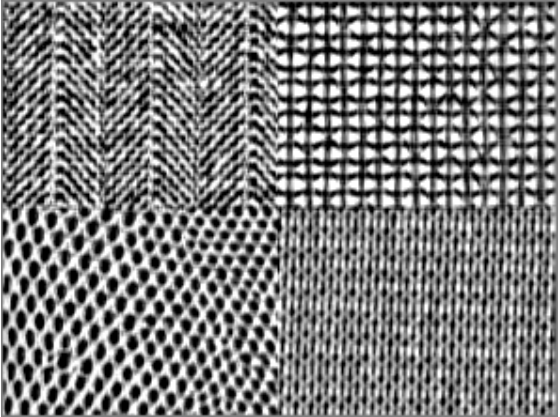
$$B = \begin{bmatrix} -0.16 & -0.16 & -0.16 & -0.16 & -0.16 \\ -0.16 & -0.4 & -0.4 & -0.4 & -0.16 \\ -0.16 & -0.4 & 4.0 & -0.4 & -0.16 \\ -0.16 & -0.4 & -0.4 & -0.4 & -0.16 \\ -0.16 & -0.16 & -0.16 & -0.16 & -0.16 \end{bmatrix}$$

$$z = 0.0$$

Global task

Given:	Static binary image P with a grid of black squares
Input:	P
Initial state:	Arbitrary(0)
Boundary condition:	Zero-flux
Output:	Gray-scale image representing P with gray patches at the intersections of the grid of black squares.

Examples

Input	Output
 <p>tx_hcllc.png</p>	

1.9.2 INVHLF3

Inverts the halftoned image.

Old names: 3x3InverseHalftoning, INVHLF33, invhlf3

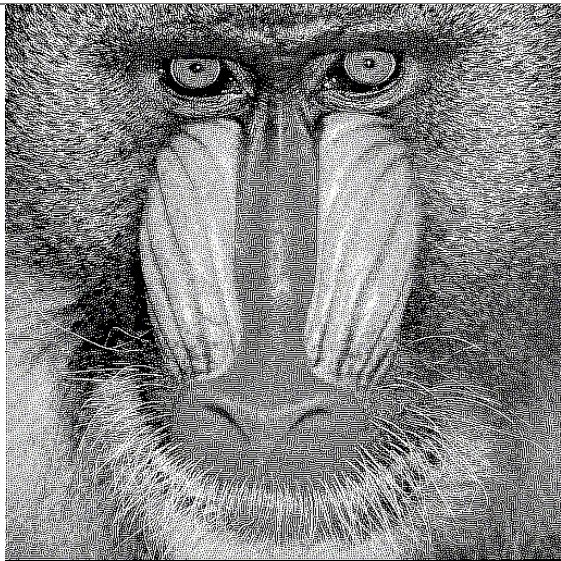
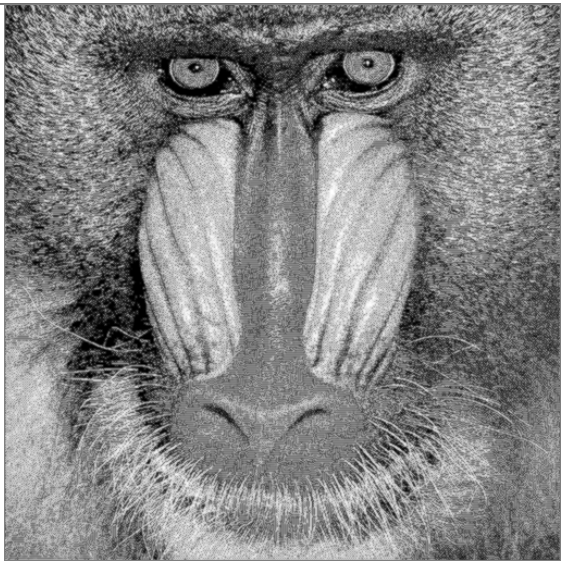
Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.07 & 0.1 & 0.07 \\ 0.1 & 0.32 & 0.1 \\ 0.07 & 0.1 & 0.07 \end{bmatrix} \quad z = 0.0$$


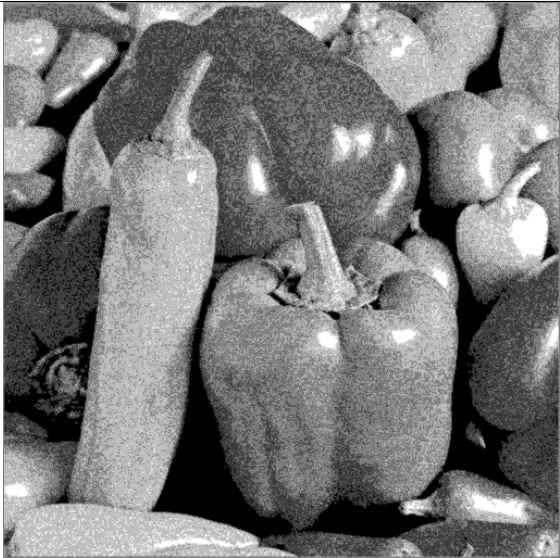
Global task

Given: Static binary image **P** obtained by using 3x3Halftoning
Input: **P**
Initial state: Arbitrary(0)
Boundary condition: Zero-flux
Output: Grayscale image representing **P**

Examples

Input	Output
 invhlf3_1.png	

Examples

Input	Output
<div><p>invhlf3_2.png</p></div>	<div></div>

1.9.3 INVHLF5

Inverts the halftoned image.

Old names: 5x5InverseHalftoning, INVHLF55, invhlf5

Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

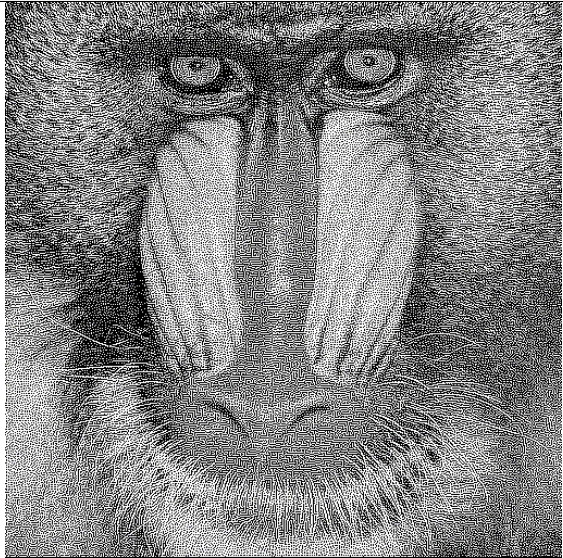
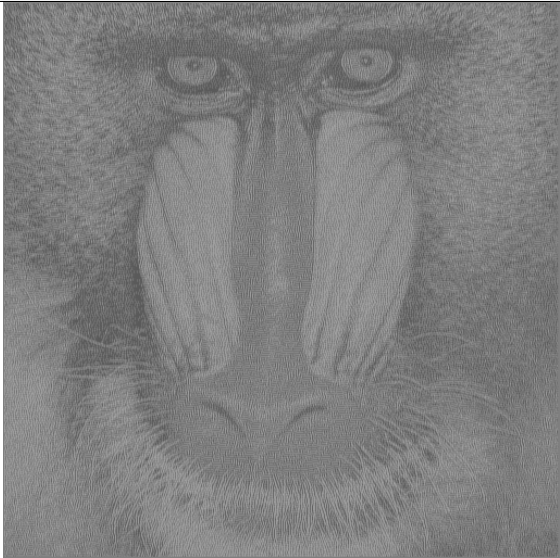
$$\mathbf{B} = \begin{bmatrix} 0.0049 & 0.014 & 0.0198 & 0.014 & 0.0049 \\ 0.014 & 0.0648 & 0.092 & 0.0648 & 0.014 \\ 0.0198 & 0.092 & 0.162 & 0.092 & 0.0198 \\ 0.014 & 0.0648 & 0.092 & 0.0648 & 0.014 \\ 0.0049 & 0.014 & 0.0198 & 0.014 & 0.0049 \end{bmatrix}$$

$$z = 0.0$$

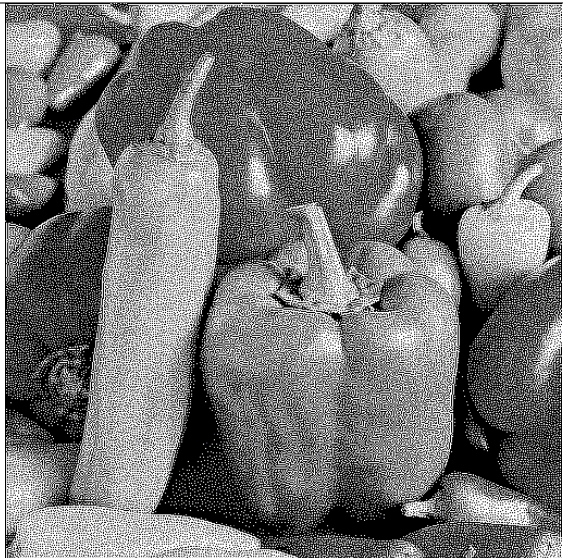
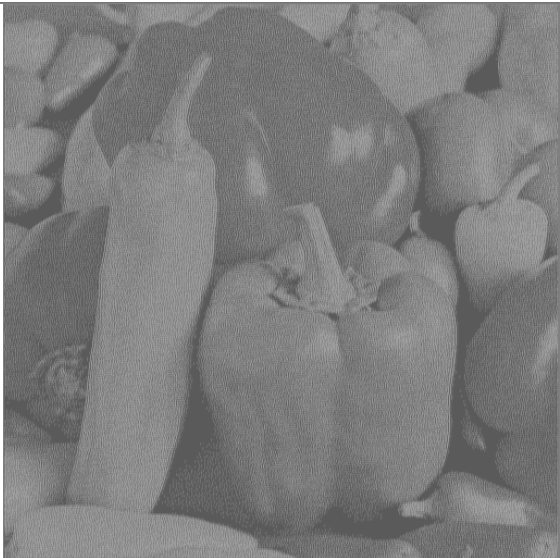
Global task

Given: Static binary image **P** obtained by using 5x5Halftoning
 Input: **P**
 Initial state: Arbitrary(0)
 Boundary condition: Zero-flux
 Output: Grayscale image representing **P**

Examples

Input	Output
 invhlf5_1.png	

Examples

Input	Output
 invhlf5_2.png	

1.10 CCD type templates

Definition

$$\mathbf{A} \neq 0, \quad r(\mathbf{A}) = 1, \quad \mathbf{B} = 0, \quad z = 0$$

Input: Binary

Output: Binary

1.10.1 CCD_DIAG

Diagonal connected component detection.

Old names: DiagonalHoleDetection, DiagonalCCD

Available in: Candy

$$\mathbf{A} = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & -1.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0.0$$

Global task

Given: Static binary image \mathbf{P}


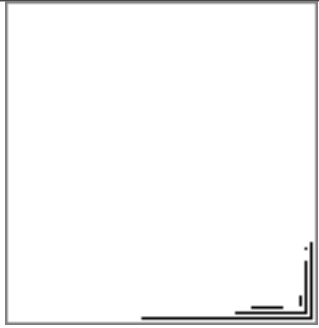
Input: Arbitrary(0)

Initial state: \mathbf{P}

Boundary condition: Fixed(0)

Output: Binary image that shows the number of diagonally connected components in \mathbf{P}

Examples

Input	Output
 <p>a_letter.png</p>	


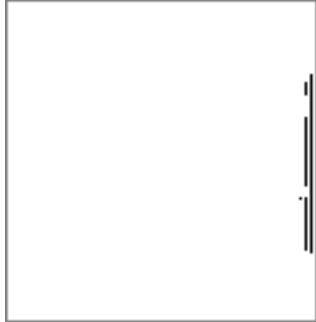
1.10.2 CCD_HOR*Horizontal connected component detection.*

Old names: HorizontalCCD, HorizontalHoleDetection, ccd_hor, HorizontalCCD, CCD_HOR

Available in: Template Library v3.1, Candy

$$A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 1.0 & 2.0 & -1.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad B = [0] \quad z = 0$$

Global taskGiven: Static binary image P *Input:* Arbitrary(0)*Initial state:* P *Boundary condition:* Fixed(0)*Output:* Binary image that shows the number of horizontally connected components in P **Examples**

Input	Output
 a_letter.png	

1.10.3 CCD_hor_I

no description

Old names:

Available in: AladdinPro

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ -1.0 & 2.0 & 1.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0.0$$

Global task

Given:

Input:

Initial state:

Boundary condition:

Output:

1.10.4 CCD_hor_r

no description

Old names:

Available in: AladdinPro

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 1.0 & 2.0 & -1.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0.0$$

Global task

Given:

Input:

Initial state:

Boundary condition:

Output:

1.10.5 Ccd_NE

no description

Old names:

Available in: AladdinPro

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & -1.0 \\ 0.0 & 2.0 & 0.0 \\ 1.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0.0$$

Global task

Given:

Input:

Initial state:

Boundary condition:

Output:

1.10.6 Ccd_NW

no description

Old names:

Available in: AladdinPro

$$\mathbf{A} = \begin{bmatrix} -1.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0.0$$

Global task

Given:

Input:

Initial state:

Boundary condition:

Output:

1.10.7 Ccd_SE

no description

Old names:

Available in: AladdinPro

$$\mathbf{A} = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & -1.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0.0$$

Global task

Given:

Input:

Initial state:

Boundary condition:

Output:

1.10.8 Ccd_SW

no description

Old names:

Available in: AladdinPro

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 1.0 \\ 0.0 & 2.0 & 0.0 \\ -1.0 & 0.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0.0$$

Global task

Given:

Input:

Initial state:

Boundary condition:

Output:


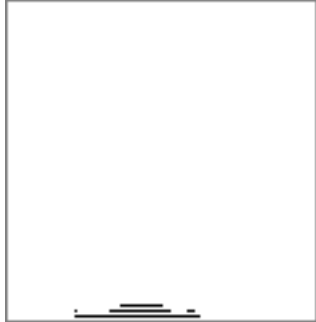
1.10.9 CCD_VERT*Vertical connected component detection.*

Old names: VerticalCCD, VerticalHoleDetection, ccd_vert

Available in: Template Library v3.1, Candy

$$A = \begin{bmatrix} 0.0 & 1.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & -1.0 & 0.0 \end{bmatrix} \quad B = [0] \quad z = 0$$

Global taskGiven: Static binary image **P***Input:* Arbitrary(0)*Initial state:* **P***Boundary condition:* Fixed(0)*Output:* Binary image that shows the number of vertically connected components in **P****Examples**

Input	Output
 a_letter.png	

1.10.10 Ccd_vert_down

no description

Old names:

Available in: AladdinPro

$$\mathbf{A} = \begin{bmatrix} 0.0 & 1.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & -1.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0.0$$

Global task

Given:

Input:

Initial state:

Boundary condition:

Output:

1.10.11 Ccd_vert_top

no description

Old names:

Available in: AladdinPro

$$\mathbf{A} = \begin{bmatrix} 0.0 & -1.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0.0$$

Global task

Given:

Input:

Initial state:

Boundary condition:

Output:

1.10.12 SMKILLER

Deletes small objects.

Old names: SmallObjectRemover, smkiller

Available in: Template Library v3.1, Candy

$$\mathbf{A} = \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 1.0 & 2.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix} \quad \mathbf{B} = [0] \quad z = 0$$

Global task

Given: Static binary image \mathbf{P}

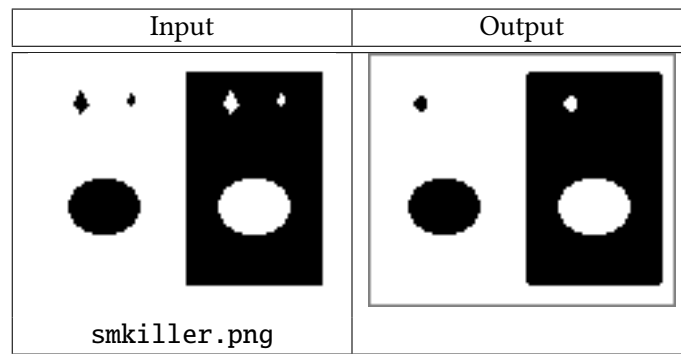
Input: \mathbf{P}

Initial state: \mathbf{P}

Boundary condition: Fixed(0)

Output: Binary image representing \mathbf{P} without small objects.

Examples



1.10.13 WErosion

no description

Old names:

Available in: AladdinPro

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.5 & 0.0 \\ 0.5 & -1.1 & 0.5 \\ 0.0 & 0.5 & 0.0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad z = 0.0$$

Global task

Given:

Input:

Initial state:

Boundary condition:

Output:

Index

- 3x3Halftoning, 147
- 3x3InverseHalftoning, 155
- 3x3TextureSegmentation, 128
- 5x5Halftoning1, 151
- 5x5Halftoning2, 149
- 5x5InverseHalftoning, 157
- 5x5TextureSegmentation1, 127
- 5x5TextureSegmentation2, 129

- AND, 73
- ApproxDiagonalLineDetector, 42
- AVERAGE, 8
- Average, 8
- Avertrsh, 8
- avertrsh, 8

- bipolar, 9
- BipolarWave, 9
- BLACK, 136
- BlackandWhiteSkeletonization(Algorithm), 111–118
- BlackFiller, 136
- BlackPropagation, 10
- bprop, 10

- CCD_DIAG, 159
- CCD_HOR, 160
- ccd_hor, 160
- CCD_hor_l, 161
- CCD_hor_r, 162
- Ccd_NE, 163
- Ccd_NW, 164
- Ccd_SE, 165
- Ccd_SW, 166
- CCD_VERT, 167
- ccd_vert, 167

- Ccd_vert_down, 168
- Ccd_vert_top, 169
- CCDMASK, 11
- CCDMASKL, 11
- CCDMASKR, 12
- CENTER, 13
- Center, 13
- center, 13
- CENTER1, 14
- CENTER2, 15
- CENTER3, 16
- CENTER4, 17
- CENTER5, 18
- CENTER6, 19
- CENTER7, 20
- CENTER8, 21
- CenterPointDetection(Algorithm), 14–21, 24–30
- CenterPointDetector, 13
- CLDILA, 22
- CLERO, 23
- CNTR2, 24
- CNTR3, 25
- CNTR4, 26
- CNTR5, 27
- CNTR6, 28
- CNTR7, 29
- CNTR8, 30
- ConcaveArcFiller, 31, 55
- ConcaveLocationFiller, 60
- CONCCONT, 32
- Conccont, 32
- ConcentricContourDetector, 32
- CONCEROS, 33
- CONCHOLL, 34

- concont, 32
- CONCTRES, 35
- CONNECTI, 36
- connecti, 36
- Connectivity, 36
- CONVOL, 134
- CORNER, 37
- CornerDetection, 37
- CornerDetector, 37
- CUT7V, 71

- DEADENDH, 38
- DeadEndH, 38
- DEADENDV, 39
- DeadEndV, 39
- DELDIAG1, 40
- deldiag1, 40
- DELVERT1, 41
- delvert1, 41
- DIAG, 42
- diag, 42
- DIAG1LIU, 43
- diag1liu, 43
- DiagonalCCD, 159
- DiagonalHoleDetection, 159
- DiagonalLineDetector, 43
- DiagonalLineRemover, 40
- DiffM2, 44
- DIFFUS, 142
- DIFFUS2, 143
- DIFFUS3, 144
- DIFFUS4, 146
- DIFFUSC, 46
- DILATION, 47
- DirectedGrowingShadow, 100, 101

- EDGE, 48
- edge, 48
- EDGEA, 49
- EdgeDetection, 48
- EdgeDetector, 48
- EDGEGRAY, 50
- ERASMASK, 51
- erasmask, 51

- EROSION, 139

- FIGDEL, 52
- figdel, 52
- FIGEXTR, 53
- figextr, 53
- FIGREC, 54
- FigureExtractor, 53
- FigureReconstructor, 54, 94
- FigureRemover, 52
- FILBLACK, 136
- filblack, 136
- FILL35, 31
- FILL65, 55
- FilledContourExtraction, 56
- FILWHITE, 137
- filwhite, 137
- FINDAREA, 56
- findarea, 56
- FramedAreasFinder, 56

- GlobalConnectivityDetection, 36
- GlobalConnectivityDetection1, 57

- Halfton, 58
- HeatDiffusion., 142–144, 146
- HERRING, 153
- HerringGridIllusion, 153
- HLF3, 147
- hlf3, 147
- HLF33, 147
- HLF5, 149
- hlf5, 149
- HLF55, 149
- HLF55_KC, 151
- HLF5KC, 151
- hlf5kc, 151
- HOLE, 59
- hole, 59
- Hole-Filling, 59
- HoleFiller, 59
- HOLLOW, 60
- hollow, 60
- HorizontalCCD, 160
- HorizontalCCD, CCD_HOR, 160

- HorizontalHoleDetection, 160
- HORLINE, 62
- HORSKELL, 63
- HorSkelL, 63
- HORSKELR, 64
- HorSkelR, 64
- ImageDifferenceComputation, 75
- INCREASE, 65
- increase, 65
- INTERP, 66
- interp, 66
- INTERPOL, 66
- INV, 76
- INV-OR, 78
- INVHLF3, 155
- invhlf3, 155
- INVHLF33, 155
- INVHLF5, 157
- invhlf5, 157
- INVHLF55, 157
- JUNCTION, 67
- junction, 67
- JunctionExtractor, 67
- JunctionExtractor1, 68
- LAPLACE, 145
- LaplacePDESolver, 145
- LCP, 69
- lcp, 69
- LE3pixelLineDetector, 72
- LE7pixelVerticalLineRemover, 71
- LeftPeeler, 89
- LeftShadow, 99
- LGHTTUNE, 72
- LINCUT7H, 70
- LINCUT7V, 71
- lincut7v, 71
- LinearTemplateInverse, 44
- LINEXTR3, 72
- linextr3, 72
- LocalConcavePlaceDetector, 69
- LocalSouthernElementDetector, 79
- LOGAND, 73
- logand, 73
- LOGDIF, 74
- logdif, 74
- LOGDIFNF, 75
- logdifnf, 75
- LogicAND, 73
- LogicANDOperation, 73
- LogicDifference1, 74
- LogicDifference2, 75
- LogicNOT, 76
- LogicNOTOperation, 76
- LogicOR, 77
- LogicOROperation, 77
- LogicORwithNOT, 78
- LOGNOT, 76
- lognot, 76
- LOGOR, 77
- logor, 77
- LOGORN, 78
- logorn, 78
- LSE, 79
- lse, 79
- MajorityVoteTaker, 80
- MajorityVoteTaker(Algorithm), 80
- MAJVOT, 80
- MAJVOT1, 80
- majvot1, 80
- majvot2, 80
- MAJVOT3, 81
- MaskedCCD, 11, 12
- MaskedObjectExtractor, 51
- MaskedShadow, 98
- MASKSHAD, 98
- MATCH, 82
- match, 82
- MD_CONT, 84
- MOTDEPEN, 83
- motdepen, 83
- MOTINDEP, 84
- motindep, 84
- MotionDetection, 83
- MotionDetection1, 83

MotionDetection2, 84
 MOVEHOR, 83
 MULLER, 85
 MullerLyerIllusion, 85

 ObjectIncreasing, 65
 OptimalEdgeDetector, 135
 optimedge, 135
 OR, 77

 PA-PB, 74
 PA-PB_F1, 75
 PATCHMAK, 86
 patchmak, 86
 PatchMaker, 86
 PathFinder, 87
 PatternMatchingFinder, 82
 PEEL1PIX, 88
 PEELHOR, 89
 peelhor, 89
 PixelSearch, 90
 PointExtraction, 52
 PointRemoval, 53
 POISSON, 91
 PoissonPDESolver, 91
 PROP1, 92
 PROP2, 93

 RECALL, 94
 RIGHTBC, 95
 RIGHTCON, 96
 rightcon, 96
 RightContourDetector, 96
 RightEdgeDetection, 96
 RotationDetector, 97

 SelectedObjectsExtraction, 54
 SHADMASK, 98
 shadmask, 98
 SHADOW, 99
 shadow, 99
 shadow0, 100
 shadow45, 101
 ShadowProjection, 99
 SHADSIM, 102
 shadsim, 102
 SHIFTE, 103
 SHIFTN, 104
 SHIFTNE, 105
 SHIFTNW, 106
 SHIFTS, 107
 SHIFTSE, 108
 SHIFTSW, 109
 SHIFTW, 110
 SKELBW1, 111
 SKELBW2, 112
 SKELBW3, 113
 SKELBW4, 114
 SKELBW5, 115
 SKELBW6, 116
 SKELBW7, 117
 SKELBW8, 118
 SmallObjectRemover, 170
 SMKILLER, 170
 smkiller, 170
 Smoothing, 8
 SpeedDetection, 84
 SPIKE_BU, 119
 SPIKE_N, 120
 SPIKE_ND, 121
 SpikeGeneration1, 119
 SpikeGeneration2, 120
 SpikeGeneration3, 121
 STRETCH, 133
 SUPSHAD, 102
 SurfaceInterpolation, 66

 T1_RACC3, 122
 T2_RACC3, 123
 T3_RACC3, 124
 T4_RACC3, 125
 TEXTUDIL, 140
 TEXTUERO, 141
 TextureDetector1, 122
 TextureDetector2, 123
 TextureDetector3, 124
 TextureDetector4, 125
 ThinLineRemover, 126
 THRES, 138

Index

Threshold, 138
TX_HCLC, 127
tx_hclc, 127
TX_RACC3, 128
TX_RACC5, 129
tx_racc5, 129

VERSKELB, 130
VerSkelB, 130
VERSKELT, 131
VerSkelT, 131
VerticalCCD, 167
VerticalHoleDetection, 167
VerticalLineRemover, 41
VerticalShadow, 102

WErosion, 171
WHITE, 137
WhiteFiller, 137
WhitePropagation, 132
wprop, 132