

The Clmg Library

4.0.6

Generated by Doxygen 1.9.8

1 The CImg Library	1
2 Topic Index	3
2.1 Topics	3
3 Namespace Index	5
3.1 Namespace List	5
4 Hierarchical Index	7
4.1 Class Hierarchy	7
5 Class Index	9
5.1 Class List	9
6 Topic Documentation	11
6.1 CImg Library Overview	11
6.1.1 Library structure	11
6.1.2 CImg version of "Hello world".	12
6.1.3 How to compile ?	12
6.1.4 What's next ?	13
6.2 FAQ : Frequently Asked Questions.	13
6.2.1 FAQ Summary	13
6.2.2 1. General information and availability	13
6.2.2.1 1.1. What is the CImg Library ?	13
6.2.2.2 1.2. What platforms are supported ?	14
6.2.2.3 1.3. How is CImg distributed ?	14
6.2.2.4 1.4. What kind of people are concerned by CImg ?	14
6.2.2.5 1.5. What are the specificities of the CeCILL license ?	14
6.2.2.6 1.6. Who is behind CImg ?	15
6.2.3 2. C++ related questions	15
6.2.3.1 2.1 What is the level of C++ knowledge needed to use CImg ?	15
6.2.3.2 2.2 How to use CImg in my own C++ program ?	15
6.2.3.3 2.3 Why is CImg entirely contained in a single header file ?	15
6.2.4 3. Other resources	16
6.2.4.1 3.1 Translations	16
6.3 Setting Environment Variables	16
6.4 How to use CImg library with Visual C++ 2005 Express Edition ?	17
6.4.1 How to use CImg library with Visual C++ 2005 Express Edition ?	17
6.5 Tutorial : Getting Started.	17
6.6 Using Image Loops.	19
6.6.1 Loops over the pixel buffer	19
6.6.2 Loops over image dimensions	20
6.6.3 Loops over interior regions and borders.	20
6.6.4 Loops using neighborhoods.	21

6.6.4.1 Neighborhood-based loops for 2D images	21
6.6.4.2 Neighborhood-based loops for 3D images	22
6.6.4.3 Defining neighborhoods	22
6.6.4.4 Using alternate variable names	22
6.6.4.5 Example codes	23
6.7 Using Display Windows.	23
6.8 How pixel data are stored with CImg.	24
6.9 Files IO in CImg.	25
6.10 Retrieving Command Line Arguments.	25
6.10.1 The cimg_usage() macro	25
6.10.2 The cimg_help() macro	25
6.10.3 The cimg_option() macro	26
6.10.4 Example of use	26
6.10.5 How to learn more about command line options ?	27
7 Namespace Documentation	29
7.1 cimg_library Namespace Reference	29
7.1.1 Detailed Description	29
7.2 cimg_library::cimg Namespace Reference	30
7.2.1 Detailed Description	39
7.2.2 Function Documentation	39
7.2.2.1 output()	39
7.2.2.2 info()	39
7.2.2.3 exception_mode() [1/2]	40
7.2.2.4 exception_mode() [2/2]	40
7.2.2.5 openmp_mode()	40
7.2.2.6 eval()	41
7.2.2.7 create_directory()	41
7.2.2.8 warn()	42
7.2.2.9 system()	42
7.2.2.10 endianness()	43
7.2.2.11 invert_endianness() [1/2]	43
7.2.2.12 invert_endianness() [2/2]	43
7.2.2.13 time()	43
7.2.2.14 tic()	44
7.2.2.15 toc()	44
7.2.2.16 sleep()	44
7.2.2.17 wait()	44
7.2.2.18 mod()	45
7.2.2.19 minmod()	45
7.2.2.20 round()	46
7.2.2.21 wave()	46

7.2.2.22	atof()	46
7.2.2.23	strncasecmp()	47
7.2.2.24	strcasecmp()	47
7.2.2.25	strellipsize() [1/2]	48
7.2.2.26	strellipsize() [2/2]	48
7.2.2.27	strpare()	48
7.2.2.28	strwindows_reserved()	49
7.2.2.29	strunescape()	49
7.2.2.30	fopen()	49
7.2.2.31	fclose()	50
7.2.2.32	is_directory()	50
7.2.2.33	is_file()	50
7.2.2.34	path_exists()	51
7.2.2.35	fsize() [1/2]	51
7.2.2.36	fsize() [2/2]	51
7.2.2.37	fdate() [1/2]	52
7.2.2.38	fdate() [2/2]	52
7.2.2.39	date() [1/2]	52
7.2.2.40	date() [2/2]	53
7.2.2.41	curl_path()	53
7.2.2.42	dcraw_path()	54
7.2.2.43	ffmpeg_path()	54
7.2.2.44	graphicsmagick_path()	54
7.2.2.45	gunzip_path()	55
7.2.2.46	gzip_path()	55
7.2.2.47	imagemagick_path()	55
7.2.2.48	medcon_path()	56
7.2.2.49	temporary_path()	56
7.2.2.50	wget_path()	57
7.2.2.51	split_filename()	57
7.2.2.52	fread()	57
7.2.2.53	fwrite()	58
7.2.2.54	fempty()	58
7.2.2.55	ftype()	58
7.2.2.56	load_network()	59
7.2.2.57	files()	59
7.2.2.58	dialog()	60

8 Class Documentation 61

8.1	Clmg< T > Struct Template Reference	61
8.1.1	Detailed Description	113
8.1.2	Member Typedef Documentation	115

8.1.2.1 iterator	115
8.1.2.2 const_iterator	115
8.1.2.3 value_type	116
8.1.3 Constructor & Destructor Documentation	116
8.1.3.1 ~CImg()	116
8.1.3.2 CImg() [1/13]	116
8.1.3.3 CImg() [2/13]	117
8.1.3.4 CImg() [3/13]	117
8.1.3.5 CImg() [4/13]	118
8.1.3.6 CImg() [5/13]	119
8.1.3.7 CImg() [6/13]	120
8.1.3.8 CImg() [7/13]	121
8.1.3.9 CImg() [8/13]	122
8.1.3.10 CImg() [9/13]	122
8.1.3.11 CImg() [10/13]	123
8.1.3.12 CImg() [11/13]	123
8.1.3.13 CImg() [12/13]	124
8.1.3.14 CImg() [13/13]	124
8.1.4 Member Function Documentation	125
8.1.4.1 assign() [1/11]	125
8.1.4.2 assign() [2/11]	125
8.1.4.3 assign() [3/11]	125
8.1.4.4 assign() [4/11]	126
8.1.4.5 assign() [5/11]	126
8.1.4.6 assign() [6/11]	126
8.1.4.7 assign() [7/11]	127
8.1.4.8 assign() [8/11]	127
8.1.4.9 assign() [9/11]	127
8.1.4.10 assign() [10/11]	127
8.1.4.11 assign() [11/11]	128
8.1.4.12 clear()	128
8.1.4.13 move_to() [1/2]	128
8.1.4.14 move_to() [2/2]	129
8.1.4.15 swap()	129
8.1.4.16 empty()	130
8.1.4.17 operator>() [1/2]	130
8.1.4.18 operator>() [2/2]	131
8.1.4.19 operator T*()	131
8.1.4.20 operator=() [1/4]	132
8.1.4.21 operator=() [2/4]	132
8.1.4.22 operator=() [3/4]	133
8.1.4.23 operator=() [4/4]	133

8.1.4.24 operator+=() [1/3]	133
8.1.4.25 operator+=() [2/3]	134
8.1.4.26 operator+=() [3/3]	134
8.1.4.27 operator++() [1/2]	135
8.1.4.28 operator++() [2/2]	135
8.1.4.29 operator+() [1/4]	135
8.1.4.30 operator+() [2/4]	136
8.1.4.31 operator+() [3/4]	136
8.1.4.32 operator+() [4/4]	136
8.1.4.33 operator-=() [1/3]	136
8.1.4.34 operator-=() [2/3]	136
8.1.4.35 operator-=() [3/3]	137
8.1.4.36 operator--() [1/2]	137
8.1.4.37 operator--() [2/2]	137
8.1.4.38 operator-() [1/4]	137
8.1.4.39 operator-() [2/4]	138
8.1.4.40 operator-() [3/4]	138
8.1.4.41 operator-() [4/4]	138
8.1.4.42 operator*() [1/3]	138
8.1.4.43 operator*() [2/3]	138
8.1.4.44 operator*() [3/3]	139
8.1.4.45 operator*() [1/3]	139
8.1.4.46 operator*() [2/3]	139
8.1.4.47 operator*() [3/3]	140
8.1.4.48 operator/() [1/3]	140
8.1.4.49 operator/() [2/3]	140
8.1.4.50 operator/() [3/3]	140
8.1.4.51 operator/() [1/3]	141
8.1.4.52 operator/() [2/3]	141
8.1.4.53 operator/() [3/3]	141
8.1.4.54 operator%() [1/3]	141
8.1.4.55 operator%() [2/3]	142
8.1.4.56 operator%() [3/3]	142
8.1.4.57 operator%() [1/3]	142
8.1.4.58 operator%() [2/3]	142
8.1.4.59 operator%() [3/3]	142
8.1.4.60 operator&() [1/3]	143
8.1.4.61 operator&() [2/3]	143
8.1.4.62 operator&() [3/3]	143
8.1.4.63 operator&() [1/3]	143
8.1.4.64 operator&() [2/3]	143
8.1.4.65 operator&() [3/3]	144

8.1.4.66 operator" =()	[1/3]	144
8.1.4.67 operator" =()	[2/3]	144
8.1.4.68 operator" =()	[3/3]	144
8.1.4.69 operator" ()	[1/3]	144
8.1.4.70 operator" ()	[2/3]	145
8.1.4.71 operator" ()	[3/3]	145
8.1.4.72 operator^=()	[1/3]	145
8.1.4.73 operator^=()	[2/3]	145
8.1.4.74 operator^=()	[3/3]	146
8.1.4.75 operator^()	[1/3]	146
8.1.4.76 operator^()	[2/3]	146
8.1.4.77 operator^()	[3/3]	146
8.1.4.78 operator<<=()	[1/3]	147
8.1.4.79 operator<<=()	[2/3]	147
8.1.4.80 operator<<=()	[3/3]	147
8.1.4.81 operator<<()	[1/3]	147
8.1.4.82 operator<<()	[2/3]	147
8.1.4.83 operator<<()	[3/3]	148
8.1.4.84 operator>>=()	[1/3]	148
8.1.4.85 operator>>=()	[2/3]	148
8.1.4.86 operator>>=()	[3/3]	148
8.1.4.87 operator>>()	[1/3]	148
8.1.4.88 operator>>()	[2/3]	149
8.1.4.89 operator>>()	[3/3]	149
8.1.4.90 operator~()		149
8.1.4.91 operator==()	[1/3]	149
8.1.4.92 operator==()	[2/3]	150
8.1.4.93 operator==()	[3/3]	150
8.1.4.94 operator"! =()	[1/3]	150
8.1.4.95 operator"! =()	[2/3]	151
8.1.4.96 operator"! =()	[3/3]	151
8.1.4.97 operator,()	[1/2]	151
8.1.4.98 operator,()	[2/2]	153
8.1.4.99 operator<()		154
8.1.4.100 pixel_type()		154
8.1.4.101 width()		154
8.1.4.102 height()		155
8.1.4.103 depth()		155
8.1.4.104 spectrum()		155
8.1.4.105 size()		156
8.1.4.106 data()	[1/2]	156
8.1.4.107 data()	[2/2]	156

8.1.4.108 offset()	157
8.1.4.109 begin()	157
8.1.4.110 end()	158
8.1.4.111 front()	158
8.1.4.112 back()	158
8.1.4.113 at() [1/2]	158
8.1.4.114 at() [2/2]	159
8.1.4.115 atX() [1/2]	159
8.1.4.116 atX() [2/2]	160
8.1.4.117 atXY() [1/2]	161
8.1.4.118 atXY() [2/2]	161
8.1.4.119 atXYZ() [1/2]	161
8.1.4.120 atXYZ() [2/2]	162
8.1.4.121 atXYZC() [1/2]	162
8.1.4.122 atXYZC() [2/2]	162
8.1.4.123 linear_atX() [1/2]	163
8.1.4.124 linear_atX() [2/2]	163
8.1.4.125 linear_atXY() [1/2]	164
8.1.4.126 linear_atXY() [2/2]	164
8.1.4.127 linear_atXYZ() [1/2]	165
8.1.4.128 linear_atXYZ() [2/2]	165
8.1.4.129 linear_atXYZC() [1/2]	165
8.1.4.130 linear_atXYZC() [2/2]	166
8.1.4.131 cubic_atX() [1/2]	166
8.1.4.132 cubic_atX_c() [1/2]	167
8.1.4.133 cubic_atX() [2/2]	167
8.1.4.134 cubic_atX_c() [2/2]	168
8.1.4.135 cubic_atXY() [1/2]	168
8.1.4.136 cubic_atXY_c() [1/2]	168
8.1.4.137 cubic_atXY() [2/2]	168
8.1.4.138 cubic_atXY_c() [2/2]	169
8.1.4.139 cubic_atXYZ() [1/2]	169
8.1.4.140 cubic_atXYZ_c() [1/2]	169
8.1.4.141 cubic_atXYZ() [2/2]	169
8.1.4.142 cubic_atXYZ_c() [2/2]	170
8.1.4.143 cubic_atXYZ_p()	170
8.1.4.144 set_linear_atX()	170
8.1.4.145 set_linear_atXY()	171
8.1.4.146 set_linear_atXYZ()	171
8.1.4.147 value_string()	171
8.1.4.148 is_shared()	172
8.1.4.149 is_empty()	172

8.1.4.150 <code>is_inf()</code>	172
8.1.4.151 <code>is_nan()</code>	173
8.1.4.152 <code>is_sameXY()</code> [1/3]	173
8.1.4.153 <code>is_sameXY()</code> [2/3]	173
8.1.4.154 <code>is_sameXY()</code> [3/3]	173
8.1.4.155 <code>is_sameXZ()</code> [1/2]	173
8.1.4.156 <code>is_sameXZ()</code> [2/2]	174
8.1.4.157 <code>is_sameXC()</code> [1/2]	174
8.1.4.158 <code>is_sameXC()</code> [2/2]	174
8.1.4.159 <code>is_sameYZ()</code> [1/2]	174
8.1.4.160 <code>is_sameYZ()</code> [2/2]	174
8.1.4.161 <code>is_sameYC()</code> [1/2]	175
8.1.4.162 <code>is_sameYC()</code> [2/2]	175
8.1.4.163 <code>is_sameZC()</code> [1/2]	175
8.1.4.164 <code>is_sameZC()</code> [2/2]	175
8.1.4.165 <code>is_sameXYZ()</code> [1/2]	175
8.1.4.166 <code>is_sameXYZ()</code> [2/2]	176
8.1.4.167 <code>is_sameXYC()</code> [1/2]	176
8.1.4.168 <code>is_sameXYC()</code> [2/2]	176
8.1.4.169 <code>is_sameXZC()</code> [1/2]	176
8.1.4.170 <code>is_sameXZC()</code> [2/2]	176
8.1.4.171 <code>is_sameYZC()</code> [1/2]	177
8.1.4.172 <code>is_sameYZC()</code> [2/2]	177
8.1.4.173 <code>is_sameXYZC()</code> [1/2]	177
8.1.4.174 <code>is_sameXYZC()</code> [2/2]	177
8.1.4.175 <code>containsXYZC()</code>	177
8.1.4.176 <code>contains()</code> [1/5]	178
8.1.4.177 <code>contains()</code> [2/5]	179
8.1.4.178 <code>contains()</code> [3/5]	179
8.1.4.179 <code>contains()</code> [4/5]	179
8.1.4.180 <code>contains()</code> [5/5]	179
8.1.4.181 <code>is_overlapped()</code>	179
8.1.4.182 <code>is_object3d()</code>	180
8.1.4.183 <code>is_CImg3d()</code>	181
8.1.4.184 <code>sqr()</code>	181
8.1.4.185 <code>sqrt()</code>	181
8.1.4.186 <code>exp()</code>	182
8.1.4.187 <code>log()</code>	182
8.1.4.188 <code>log2()</code>	182
8.1.4.189 <code>log10()</code>	183
8.1.4.190 <code>abs()</code>	183
8.1.4.191 <code>sign()</code>	183

8.1.4.192 cos()	184
8.1.4.193 sin()	184
8.1.4.194 sinc()	184
8.1.4.195 tan()	185
8.1.4.196 cosh()	185
8.1.4.197 sinh()	185
8.1.4.198 tanh()	186
8.1.4.199 acos()	186
8.1.4.200 asin()	186
8.1.4.201 atan()	186
8.1.4.202 atan2()	187
8.1.4.203 acosh()	187
8.1.4.204 asinh()	187
8.1.4.205 atanh()	188
8.1.4.206 mul()	188
8.1.4.207 div()	188
8.1.4.208 pow() [1/3]	189
8.1.4.209 pow() [2/3]	189
8.1.4.210 pow() [3/3]	189
8.1.4.211 rol() [1/3]	190
8.1.4.212 rol() [2/3]	190
8.1.4.213 rol() [3/3]	190
8.1.4.214 ror() [1/3]	190
8.1.4.215 ror() [2/3]	190
8.1.4.216 ror() [3/3]	191
8.1.4.217 min() [1/3]	191
8.1.4.218 min() [2/3]	191
8.1.4.219 min() [3/3]	191
8.1.4.220 max() [1/3]	193
8.1.4.221 max() [2/3]	193
8.1.4.222 max() [3/3]	193
8.1.4.223 minabs() [1/3]	194
8.1.4.224 minabs() [2/3]	194
8.1.4.225 minabs() [3/3]	194
8.1.4.226 maxabs() [1/3]	195
8.1.4.227 maxabs() [2/3]	195
8.1.4.228 maxabs() [3/3]	195
8.1.4.229 min_max()	196
8.1.4.230 max_min()	196
8.1.4.231 kth_smallest()	196
8.1.4.232 variance()	197
8.1.4.233 variance_mean()	197

8.1.4.234 variance_noise()	197
8.1.4.235 MSE()	198
8.1.4.236 PSNR()	198
8.1.4.237 eval() [1/3]	198
8.1.4.238 eval() [2/3]	199
8.1.4.239 eval() [3/3]	199
8.1.4.240 get_stats()	200
8.1.4.241 magnitude()	200
8.1.4.242 dot()	200
8.1.4.243 get_vector_at()	201
8.1.4.244 get_matrix_at()	201
8.1.4.245 get_tensor_at()	201
8.1.4.246 set_vector_at()	202
8.1.4.247 set_matrix_at()	202
8.1.4.248 set_tensor_at()	202
8.1.4.249 diagonal()	203
8.1.4.250 identity_matrix() [1/2]	203
8.1.4.251 sequence() [1/2]	203
8.1.4.252 transpose()	203
8.1.4.253 cross()	204
8.1.4.254 invert()	204
8.1.4.255 solve()	204
8.1.4.256 solve_tridiagonal()	205
8.1.4.257 eigen()	205
8.1.4.258 get_eigen()	205
8.1.4.259 symmetric_eigen()	206
8.1.4.260 get_symmetric_eigen()	207
8.1.4.261 sort() [1/2]	207
8.1.4.262 sort() [2/2]	207
8.1.4.263 SVD()	208
8.1.4.264 get_SVD()	208
8.1.4.265 QR()	209
8.1.4.266 project_matrix()	209
8.1.4.267 dijkstra() [1/2]	210
8.1.4.268 dijkstra() [2/2]	211
8.1.4.269 string()	211
8.1.4.270 row_vector() [1/4]	211
8.1.4.271 row_vector() [2/4]	212
8.1.4.272 row_vector() [3/4]	212
8.1.4.273 row_vector() [4/4]	212
8.1.4.274 vector() [1/4]	213
8.1.4.275 vector() [2/4]	213

8.1.4.276 vector() [3/4]	213
8.1.4.277 vector() [4/4]	213
8.1.4.278 matrix() [1/3]	214
8.1.4.279 matrix() [2/3]	214
8.1.4.280 matrix() [3/3]	215
8.1.4.281 tensor()	215
8.1.4.282 identity_matrix() [2/2]	215
8.1.4.283 sequence() [2/2]	216
8.1.4.284 rotation_matrix()	216
8.1.4.285 fill() [1/4]	216
8.1.4.286 fill() [2/4]	217
8.1.4.287 fill() [3/4]	217
8.1.4.288 fill_from_values()	217
8.1.4.289 fill() [4/4]	218
8.1.4.290 fillX()	218
8.1.4.291 fillY()	218
8.1.4.292 fillZ()	219
8.1.4.293 fillC()	219
8.1.4.294 discard()	219
8.1.4.295 rand() [1/2]	220
8.1.4.296 rand() [2/2]	220
8.1.4.297 round()	220
8.1.4.298 noise()	222
8.1.4.299 normalize() [1/2]	222
8.1.4.300 normalize() [2/2]	223
8.1.4.301 norm()	223
8.1.4.302 cut()	223
8.1.4.303 abscut()	224
8.1.4.304 quantize()	224
8.1.4.305 otsu()	224
8.1.4.306 threshold()	225
8.1.4.307 histogram()	225
8.1.4.308 equalize()	226
8.1.4.309 index()	226
8.1.4.310 map()	227
8.1.4.311 label() [1/2]	227
8.1.4.312 label() [2/2]	228
8.1.4.313 default_LUT256()	228
8.1.4.314 HSV_LUT256()	228
8.1.4.315 lines_LUT256()	229
8.1.4.316 hot_LUT256()	229
8.1.4.317 cool_LUT256()	229

8.1.4.318 jet_LUT256()	229
8.1.4.319 flag_LUT256()	230
8.1.4.320 cube_LUT256()	230
8.1.4.321 RGBtoXYZ()	230
8.1.4.322 XYZtoRGB()	230
8.1.4.323 resize() [1/3]	231
8.1.4.324 resize() [2/3]	231
8.1.4.325 resize() [3/3]	232
8.1.4.326 resize_doubleXY()	232
8.1.4.327 resize_tripleXY()	233
8.1.4.328 mirror() [1/2]	233
8.1.4.329 mirror() [2/2]	233
8.1.4.330 shift()	233
8.1.4.331 permute_axes()	234
8.1.4.332 unroll()	234
8.1.4.333 rotate() [1/4]	234
8.1.4.334 rotate() [2/4]	235
8.1.4.335 rotate() [3/4]	235
8.1.4.336 rotate() [4/4]	236
8.1.4.337 warp()	237
8.1.4.338 get_projections2d()	237
8.1.4.339 crop()	237
8.1.4.340 autocrop()	239
8.1.4.341 get_column()	239
8.1.4.342 columns()	239
8.1.4.343 row()	240
8.1.4.344 get_rows()	240
8.1.4.345 get_slice()	240
8.1.4.346 get_slices()	240
8.1.4.347 get_channel()	241
8.1.4.348 get_channels()	241
8.1.4.349 streamline()	241
8.1.4.350 get_shared_points()	242
8.1.4.351 get_shared_rows()	243
8.1.4.352 get_shared_row()	243
8.1.4.353 get_shared_slices()	243
8.1.4.354 get_shared_slice()	244
8.1.4.355 get_shared_channels()	244
8.1.4.356 get_shared_channel()	244
8.1.4.357 get_split() [1/2]	244
8.1.4.358 get_split() [2/2]	245
8.1.4.359 append()	245

8.1.4.360 correlate()	246
8.1.4.361 convolve()	247
8.1.4.362 cumulate() [1/2]	248
8.1.4.363 cumulate() [2/2]	248
8.1.4.364 erode() [1/3]	248
8.1.4.365 erode() [2/3]	249
8.1.4.366 erode() [3/3]	249
8.1.4.367 dilate() [1/3]	249
8.1.4.368 dilate() [2/3]	249
8.1.4.369 dilate() [3/3]	250
8.1.4.370 closing() [1/2]	250
8.1.4.371 closing() [2/2]	250
8.1.4.372 opening() [1/2]	252
8.1.4.373 opening() [2/2]	252
8.1.4.374 watershed()	252
8.1.4.375 deriche()	253
8.1.4.376 vanvliet()	253
8.1.4.377 blur() [1/2]	254
8.1.4.378 blur() [2/2]	254
8.1.4.379 blur_anisotropic() [1/2]	255
8.1.4.380 blur_anisotropic() [2/2]	255
8.1.4.381 blur_bilateral() [1/2]	256
8.1.4.382 blur_bilateral() [2/2]	257
8.1.4.383 boxfilter()	257
8.1.4.384 blur_box() [1/2]	257
8.1.4.385 blur_box() [2/2]	258
8.1.4.386 blur_guided()	258
8.1.4.387 blur_patch()	260
8.1.4.388 blur_median()	260
8.1.4.389 sharpen()	261
8.1.4.390 get_gradient()	261
8.1.4.391 get_hessian()	261
8.1.4.392 structure_tensors()	262
8.1.4.393 diffusion_tensors()	262
8.1.4.394 displacement()	262
8.1.4.395 matchpatch()	263
8.1.4.396 distance() [1/2]	264
8.1.4.397 distance() [2/2]	264
8.1.4.398 distance_dijkstra()	264
8.1.4.399 distance_eikonal() [1/2]	266
8.1.4.400 distance_eikonal() [2/2]	266
8.1.4.401 haar() [1/2]	266

8.1.4.402 haar() [2/2]	267
8.1.4.403 get_FFT()	267
8.1.4.404 FFT() [1/2]	267
8.1.4.405 FFT() [2/2]	268
8.1.4.406 rotate_object3d()	268
8.1.4.407 shift_object3d() [1/2]	268
8.1.4.408 shift_object3d() [2/2]	269
8.1.4.409 resize_object3d()	269
8.1.4.410 append_object3d()	269
8.1.4.411 texturize_object3d()	271
8.1.4.412 get_elevation3d()	271
8.1.4.413 get_projections3d()	272
8.1.4.414 get_isoline3d()	272
8.1.4.415 isoline3d() [1/2]	273
8.1.4.416 isoline3d() [2/2]	273
8.1.4.417 get_isosurface3d()	274
8.1.4.418 isosurface3d() [1/2]	275
8.1.4.419 isosurface3d() [2/2]	276
8.1.4.420 elevation3d()	276
8.1.4.421 box3d()	277
8.1.4.422 cone3d()	277
8.1.4.423 cylinder3d()	278
8.1.4.424 torus3d()	279
8.1.4.425 plane3d()	279
8.1.4.426 sphere3d()	280
8.1.4.427 ellipsoid3d()	281
8.1.4.428 object3dtoCImg3d()	281
8.1.4.429 CImg3dtoobject3d()	281
8.1.4.430 draw_point() [1/2]	282
8.1.4.431 draw_point() [2/2]	282
8.1.4.432 draw_line() [1/6]	283
8.1.4.433 draw_line() [2/6]	284
8.1.4.434 draw_line() [3/6]	284
8.1.4.435 draw_line() [4/6]	285
8.1.4.436 draw_line() [5/6]	286
8.1.4.437 draw_line() [6/6]	287
8.1.4.438 draw_arrow()	287
8.1.4.439 draw_spline() [1/4]	288
8.1.4.440 draw_spline() [2/4]	289
8.1.4.441 draw_spline() [3/4]	290
8.1.4.442 draw_spline() [4/4]	290
8.1.4.443 draw_triangle() [1/9]	291

8.1.4.444 draw_triangle() [2/9]	291
8.1.4.445 draw_triangle() [3/9]	292
8.1.4.446 draw_triangle() [4/9]	292
8.1.4.447 draw_triangle() [5/9]	293
8.1.4.448 draw_triangle() [6/9]	294
8.1.4.449 draw_triangle() [7/9]	294
8.1.4.450 draw_triangle() [8/9]	295
8.1.4.451 draw_triangle() [9/9]	296
8.1.4.452 draw_rectangle() [1/3]	297
8.1.4.453 draw_rectangle() [2/3]	298
8.1.4.454 draw_rectangle() [3/3]	298
8.1.4.455 draw_polygon()	299
8.1.4.456 draw_ellipse() [1/4]	299
8.1.4.457 draw_ellipse() [2/4]	300
8.1.4.458 draw_ellipse() [3/4]	300
8.1.4.459 draw_ellipse() [4/4]	301
8.1.4.460 draw_circle() [1/2]	301
8.1.4.461 draw_circle() [2/2]	302
8.1.4.462 draw_image() [1/2]	302
8.1.4.463 draw_image() [2/2]	303
8.1.4.464 draw_text() [1/4]	303
8.1.4.465 draw_text() [2/4]	304
8.1.4.466 draw_text() [3/4]	304
8.1.4.467 draw_text() [4/4]	305
8.1.4.468 draw_quiver() [1/2]	305
8.1.4.469 draw_quiver() [2/2]	306
8.1.4.470 draw_axis() [1/2]	306
8.1.4.471 draw_axis() [2/2]	307
8.1.4.472 draw_axes()	307
8.1.4.473 draw_grid()	308
8.1.4.474 draw_graph()	308
8.1.4.475 draw_fill()	309
8.1.4.476 draw_plasma()	310
8.1.4.477 draw_mandelbrot()	310
8.1.4.478 draw_gaussian() [1/2]	311
8.1.4.479 draw_gaussian() [2/2]	312
8.1.4.480 draw_object3d()	312
8.1.4.481 select()	314
8.1.4.482 load()	314
8.1.4.483 load_ascii()	315
8.1.4.484 load_dlm()	315
8.1.4.485 load_bmp()	315

8.1.4.486 load_jpeg()	316
8.1.4.487 load_jxl()	316
8.1.4.488 load_magick()	316
8.1.4.489 load_png()	316
8.1.4.490 load_pnm()	317
8.1.4.491 load_pfm()	317
8.1.4.492 load_rgb()	317
8.1.4.493 load_rgba()	318
8.1.4.494 load_tiff()	318
8.1.4.495 load_minc2()	319
8.1.4.496 load_analyze()	319
8.1.4.497 load_cimg() [1/2]	319
8.1.4.498 load_cimg() [2/2]	320
8.1.4.499 load_inr()	320
8.1.4.500 load_exr()	321
8.1.4.501 load_pandore()	321
8.1.4.502 load_parrec()	321
8.1.4.503 load_raw()	321
8.1.4.504 load_yuv()	322
8.1.4.505 load_off()	323
8.1.4.506 load_video()	323
8.1.4.507 load_ffmpeg_external()	323
8.1.4.508 load_gif_external()	324
8.1.4.509 load_heif()	324
8.1.4.510 load_webp()	324
8.1.4.511 load_graphicsmagick_external()	325
8.1.4.512 load_gzip_external()	325
8.1.4.513 load_imagemagick_external()	325
8.1.4.514 load_medcon_external()	325
8.1.4.515 load_pdf_external()	326
8.1.4.516 load_dcraw_external()	326
8.1.4.517 load_camera()	326
8.1.4.518 load_other()	327
8.1.4.519 print()	327
8.1.4.520 display() [1/3]	327
8.1.4.521 display() [2/3]	328
8.1.4.522 display() [3/3]	328
8.1.4.523 display_object3d()	328
8.1.4.524 display_graph()	329
8.1.4.525 save()	330
8.1.4.526 save_ascii()	330
8.1.4.527 save_cpp()	331

8.1.4.528	save_dlm()	331
8.1.4.529	save_bmp()	331
8.1.4.530	save_webp()	331
8.1.4.531	save_jpeg()	332
8.1.4.532	save_jxl()	332
8.1.4.533	save_magick()	332
8.1.4.534	save_png()	333
8.1.4.535	save_pnm()	333
8.1.4.536	save_pnk()	333
8.1.4.537	save_pfm()	334
8.1.4.538	save_rgb()	334
8.1.4.539	save_rgba()	334
8.1.4.540	save_tiff()	334
8.1.4.541	save_minc2()	335
8.1.4.542	save_analyze()	335
8.1.4.543	save_cimg() [1/2]	336
8.1.4.544	save_cimg() [2/2]	336
8.1.4.545	save_empty_cimg() [1/2]	336
8.1.4.546	save_empty_cimg() [2/2]	337
8.1.4.547	save_inr()	337
8.1.4.548	save_exr()	337
8.1.4.549	save_pandore() [1/2]	338
8.1.4.550	save_pandore() [2/2]	338
8.1.4.551	save_raw() [1/2]	338
8.1.4.552	save_raw() [2/2]	339
8.1.4.553	save_yuv() [1/2]	339
8.1.4.554	save_yuv() [2/2]	339
8.1.4.555	save_off() [1/2]	340
8.1.4.556	save_off() [2/2]	340
8.1.4.557	save_video()	340
8.1.4.558	save_ffmpeg_external()	341
8.1.4.559	save_gzip_external()	341
8.1.4.560	save_graphicmagick_external()	341
8.1.4.561	save_imagemagick_external()	342
8.1.4.562	save_medcon_external()	342
8.1.4.563	save_other()	343
8.1.4.564	get_serialize()	343
8.2	CImgDisplay Struct Reference	343
8.2.1	Detailed Description	348
8.2.2	Constructor & Destructor Documentation	348
8.2.2.1	~CImgDisplay()	348
8.2.2.2	CImgDisplay() [1/5]	349

8.2.2.3 CImgDisplay() [2/5]	349
8.2.2.4 CImgDisplay() [3/5]	349
8.2.2.5 CImgDisplay() [4/5]	351
8.2.2.6 CImgDisplay() [5/5]	351
8.2.3 Member Function Documentation	352
8.2.3.1 screenshot() [1/2]	352
8.2.3.2 assign()	352
8.2.3.3 empty()	352
8.2.3.4 operator=() [1/3]	352
8.2.3.5 operator=() [2/3]	353
8.2.3.6 operator=() [3/3]	353
8.2.3.7 operator bool()	353
8.2.3.8 is_closed()	353
8.2.3.9 is_key() [1/3]	353
8.2.3.10 is_key() [2/3]	353
8.2.3.11 is_key() [3/3]	354
8.2.3.12 is_key_sequence()	354
8.2.3.13 is_keyESC()	355
8.2.3.14 width()	355
8.2.3.15 height()	355
8.2.3.16 normalization()	356
8.2.3.17 title()	356
8.2.3.18 window_width()	356
8.2.3.19 window_height()	357
8.2.3.20 window_x()	357
8.2.3.21 window_y()	357
8.2.3.22 mouse_x()	357
8.2.3.23 mouse_y()	357
8.2.3.24 button()	358
8.2.3.25 wheel()	358
8.2.3.26 key()	359
8.2.3.27 released_key()	360
8.2.3.28 keycode()	360
8.2.3.29 frames_per_second()	361
8.2.3.30 display() [1/2]	361
8.2.3.31 display() [2/2]	361
8.2.3.32 show()	362
8.2.3.33 close()	362
8.2.3.34 move()	362
8.2.3.35 resize() [1/4]	363
8.2.3.36 resize() [2/4]	363
8.2.3.37 resize() [3/4]	363

8.2.3.38 resize() [4 / 4]	364
8.2.3.39 set_normalization()	364
8.2.3.40 set_title()	364
8.2.3.41 set_fullscreen()	365
8.2.3.42 toggle_fullscreen()	365
8.2.3.43 show_mouse()	365
8.2.3.44 hide_mouse()	366
8.2.3.45 set_mouse()	366
8.2.3.46 set_button() [1 / 2]	366
8.2.3.47 set_button() [2 / 2]	366
8.2.3.48 set_wheel() [1 / 2]	367
8.2.3.49 set_wheel() [2 / 2]	367
8.2.3.50 set_key() [1 / 2]	367
8.2.3.51 set_key() [2 / 2]	367
8.2.3.52 flush()	368
8.2.3.53 wait()	368
8.2.3.54 render()	368
8.2.3.55 paint()	369
8.2.3.56 screenshot() [2 / 2]	369
8.2.3.57 snapshot()	369
8.3 CImgException Struct Reference	370
8.3.1 Detailed Description	370
8.4 CImgList< T > Struct Template Reference	371
8.4.1 Detailed Description	382
8.4.2 Member Typedef Documentation	382
8.4.2.1 iterator	382
8.4.2.2 const_iterator	383
8.4.2.3 value_type	383
8.4.3 Constructor & Destructor Documentation	383
8.4.3.1 ~CImgList()	383
8.4.3.2 CImgList() [1 / 17]	384
8.4.3.3 CImgList() [2 / 17]	384
8.4.3.4 CImgList() [3 / 17]	384
8.4.3.5 CImgList() [4 / 17]	385
8.4.3.6 CImgList() [5 / 17]	385
8.4.3.7 CImgList() [6 / 17]	385
8.4.3.8 CImgList() [7 / 17]	386
8.4.3.9 CImgList() [8 / 17]	386
8.4.3.10 CImgList() [9 / 17]	387
8.4.3.11 CImgList() [10 / 17]	387
8.4.3.12 CImgList() [11 / 17]	387
8.4.3.13 CImgList() [12 / 17]	388

8.4.3.14 CImgList()	[13/17]	389
8.4.3.15 CImgList()	[14/17]	389
8.4.3.16 CImgList()	[15/17]	389
8.4.3.17 CImgList()	[16/17]	390
8.4.3.18 CImgList()	[17/17]	390
8.4.4 Member Function Documentation		390
8.4.4.1 get_shared()		390
8.4.4.2 assign()	[1/16]	391
8.4.4.3 clear()		391
8.4.4.4 assign()	[2/16]	391
8.4.4.5 assign()	[3/16]	391
8.4.4.6 assign()	[4/16]	392
8.4.4.7 assign()	[5/16]	392
8.4.4.8 assign()	[6/16]	392
8.4.4.9 assign()	[7/16]	393
8.4.4.10 assign()	[8/16]	393
8.4.4.11 assign()	[9/16]	393
8.4.4.12 assign()	[10/16]	394
8.4.4.13 assign()	[11/16]	394
8.4.4.14 assign()	[12/16]	394
8.4.4.15 assign()	[13/16]	395
8.4.4.16 assign()	[14/16]	395
8.4.4.17 assign()	[15/16]	395
8.4.4.18 assign()	[16/16]	396
8.4.4.19 move_to()	[1/2]	396
8.4.4.20 move_to()	[2/2]	396
8.4.4.21 swap()		397
8.4.4.22 empty()		397
8.4.4.23 operator>()	[1/3]	397
8.4.4.24 operator>()	[2/3]	397
8.4.4.25 operator>()	[3/3]	398
8.4.4.26 operator CImg< T > *()		398
8.4.4.27 operator=()	[1/4]	398
8.4.4.28 operator=()	[2/4]	399
8.4.4.29 operator=()	[3/4]	399
8.4.4.30 operator=()	[4/4]	399
8.4.4.31 operator+()		400
8.4.4.32 operator,()	[1/2]	400
8.4.4.33 operator,()	[2/2]	400
8.4.4.34 operator>()		400
8.4.4.35 operator<()		401
8.4.4.36 pixel_type()		401

8.4.4.37 width()	401
8.4.4.38 size()	402
8.4.4.39 data() [1/2]	402
8.4.4.40 data() [2/2]	402
8.4.4.41 at()	402
8.4.4.42 atNXYZC() [1/2]	403
8.4.4.43 atNXYZC() [2/2]	403
8.4.4.44 atNXYZ() [1/2]	404
8.4.4.45 atNXYZ() [2/2]	404
8.4.4.46 atNXY() [1/2]	405
8.4.4.47 atNXY() [2/2]	405
8.4.4.48 atNX() [1/2]	406
8.4.4.49 atNX() [2/2]	406
8.4.4.50 atN() [1/2]	407
8.4.4.51 atN() [2/2]	407
8.4.4.52 is_sameN() [1/2]	408
8.4.4.53 is_sameN() [2/2]	408
8.4.4.54 is_sameXYZC()	408
8.4.4.55 is_sameNXYZC()	409
8.4.4.56 containsNXYZC()	409
8.4.4.57 containsN()	409
8.4.4.58 contains() [1/8]	410
8.4.4.59 contains() [2/8]	410
8.4.4.60 contains() [3/8]	411
8.4.4.61 contains() [4/8]	411
8.4.4.62 contains() [5/8]	412
8.4.4.63 contains() [6/8]	412
8.4.4.64 contains() [7/8]	412
8.4.4.65 contains() [8/8]	413
8.4.4.66 min_max() [1/2]	413
8.4.4.67 min_max() [2/2]	413
8.4.4.68 max_min()	413
8.4.4.69 insert() [1/5]	414
8.4.4.70 insert() [2/5]	414
8.4.4.71 insert() [3/5]	414
8.4.4.72 insert() [4/5]	415
8.4.4.73 insert() [5/5]	415
8.4.4.74 remove() [1/2]	416
8.4.4.75 remove() [2/2]	416
8.4.4.76 images()	416
8.4.4.77 get_shared_images()	416
8.4.4.78 get_append()	417

8.4.4.79 split()	417
8.4.4.80 push_back() [1/2]	417
8.4.4.81 push_front() [1/2]	418
8.4.4.82 push_back() [2/2]	418
8.4.4.83 push_front() [2/2]	418
8.4.4.84 erase()	418
8.4.4.85 get_select() [1/2]	419
8.4.4.86 get_select() [2/2]	419
8.4.4.87 load()	420
8.4.4.88 load_cimg() [1/3]	420
8.4.4.89 load_cimg() [2/3]	420
8.4.4.90 load_cimg() [3/3]	420
8.4.4.91 load_parrec()	421
8.4.4.92 load_yuv()	421
8.4.4.93 load_video()	422
8.4.4.94 load_ffmpeg_external()	422
8.4.4.95 load_gif_external()	423
8.4.4.96 load_gzip_external()	423
8.4.4.97 load_tiff()	423
8.4.4.98 print()	424
8.4.4.99 display() [1/3]	424
8.4.4.100 display() [2/3]	424
8.4.4.101 display() [3/3]	425
8.4.4.102 save()	425
8.4.4.103 is_saveable()	426
8.4.4.104 save_gif_external()	426
8.4.4.105 save_yuv() [1/2]	426
8.4.4.106 save_yuv() [2/2]	427
8.4.4.107 save_cimg() [1/4]	427
8.4.4.108 save_cimg() [2/4]	427
8.4.4.109 save_cimg() [3/4]	428
8.4.4.110 save_cimg() [4/4]	428
8.4.4.111 save_empty_cimg() [1/2]	428
8.4.4.112 save_empty_cimg() [2/2]	429
8.4.4.113 save_tiff()	429
8.4.4.114 save_gzip_external()	430
8.4.4.115 save_video()	430
8.4.4.116 save_ffmpeg_external()	430
8.4.4.117 get_serialize()	431
8.4.4.118 font()	431
8.4.4.119 FFT() [1/2]	431
8.4.4.120 FFT() [2/2]	433

Chapter 1

The CImg Library

This is the reference documentation of [the CImg Library](#), the C++ template image processing library. This documentation have been generated using the tool [doxygen](#). It contains a detailed description of all classes and functions of the CImg Library.

Use the menu above to navigate through the documentation pages. As a first step, you may look at the list of [available modules](#).

You may be interested also in the [presentation slides](#) presenting an overview of the CImg Library capabilities.

Chapter 2

Topic Index

2.1 Topics

Here is a list of all topics with brief descriptions:

CImg Library Overview	11
FAQ : Frequently Asked Questions.	13
Setting Environment Variables	16
How to use CImg library with Visual C++ 2005 Express Edition ?.	17
Tutorial : Getting Started.	17
Using Image Loops.	19
Using Display Windows.	23
How pixel data are stored with CImg.	24
Files IO in CImg.	25
Retrieving Command Line Arguments.	25

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

cimg_library	Contains <i>all classes and functions</i> of the CImg library	29
cimg_library::cimg	Contains low-level functions and variables of the CImg Library	30

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

<code>Clmg< T ></code>	61
<code>Clmg< charT ></code>	61
<code>Clmg< doubleT ></code>	61
<code>Clmg< floatT ></code>	61
<code>Clmg< intT ></code>	61
<code>Clmg< uintT ></code>	61
<code>Clmg< ulongT ></code>	61
<code>ClmgDisplay</code>	343
<code>ClmgException</code>	370
<code>ClmgList< T ></code>	371
<code>ClmgList< charT ></code>	371
<code>ClmgList< doubleT ></code>	371
<code>ClmgList< ulongT ></code>	371

Chapter 5

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

CImg< T >	Class representing an image (up to 4 dimensions wide), where each pixel is of type <code>T</code>	61
CImgDisplay	Allow the creation of windows, display images on them and manage user events (keyboard, mouse and windows events)	343
CImgException	Instances of CImgException are thrown when errors are encountered in a <code>CImg</code> function call	370
CImgList< T >	Represents a list of images <code>CImg<T></code>	371

Chapter 6

Topic Documentation

6.1 CImg Library Overview

The **CImg Library** is an image processing library, designed for C++ programmers. It provides useful classes and functions to load/save, display and process various types of images.

6.1.1 Library structure

The CImg Library consists in a single header file `CImg.h` providing a set of C++ template classes that can be used in your own sources, to load/save, process and display images or list of images. Very portable (Unix/X11, Windows, MacOS X, FreeBSD,...), efficient, simple to use, it's a pleasant toolkit for coding image processing stuff in C++.

The header file `CImg.h` contains all the classes and functions that compose the library itself. This is one originality of the CImg Library. This particularly means that :

- No pre-compilation of the library is needed, since the compilation of the CImg functions is done at the same time as the compilation of your own C++ code.
- No complex dependencies have to be handled : Just include the `CImg.h` file, and you get a working C++ image processing toolkit.
- The compilation is done on the fly : only CImg functionalities really used by your program are compiled and appear in the compiled executable program. This leads to very compact code, without any unused stuff.
- Class members and functions are inlined, leading to better performance during the program execution.

The CImg Library is structured as follows :

- All library classes and functions are defined in the namespace `cimg_library`. This namespace encapsulates the library functionalities and avoid any class name collision that could happen with other includes. Generally, one uses this namespace as a default namespace :

```
#include "CImg.h"
using namespace cimg_library;
...
```
- The namespace `cimg_library::cimg` defines a set of *low-level* functions and variables used by the library. Documented functions in this namespace can be safely used in your own program. But, **never** use the `cimg_library::cimg` namespace as a default namespace, since it contains functions whose names are already defined in the standard C/C++ library.

- The class `cimg_library::CImg` represents images up to 4-dimensions wide, containing pixels of type `T` (template parameter). This is actually the main class of the library.
- The class `cimg_library::CImgList` represents lists of `cimg_library::CImg<T>` images. It can be used for instance to store different frames of an image sequence.
- The class `cimg_library::CImgDisplay` is able to display images or image lists into graphical display windows. As you may guess, the code of this class is highly system-dependent but this is transparent for the programmer, as environment variables are automatically set by the CImg library (see also [Setting Environment Variables](#)).
- The class `cimg_library::CImgException` (and its subclasses) are used by the library to throw exceptions when errors occur. Those exceptions can be caught with a `try { .. } catch (CImgException) { .. }` block. Subclasses define precisely the type of encountered errors.

Knowing these four classes is **enough** to get benefit of the CImg Library functionalities.

6.1.2 CImg version of "Hello world".

Below is some very simple code that creates a "Hello World" image. This shows you basically how a CImg program looks like.

```
#include "CImg.h"
using namespace cimg_library;

int main() {
    CImg<unsigned char> img(640,400,1,3); // Define a 640x400 color image with 8 bits per color component.
    img.fill(0); // Set pixel values to 0 (color : black)
    unsigned char purple[] = { 255,0,255 }; // Define a purple color
    img.draw_text(100,100,"Hello World",purple); // Draw a purple "Hello world" at coordinates (100,100).
    img.display("My first CImg code"); // Display the image in a display window.
    return 0;
}
```

Which can be also written in a more compact way as :

```
#include "CImg.h"
using namespace cimg_library;

int main() {
    const unsigned char purple[] = { 255,0,255 };
    CImg<unsigned char>(640,400,1,3,0).draw_text(100,100,"Hello World",purple).display("My first CImg code");
    return 0;
}
```

Generally, you can write very small code that performs complex image processing tasks. The CImg Library is very simple to use and provides a lot of interesting algorithms for image manipulation.

6.1.3 How to compile ?

The CImg library is a very light and user-friendly library : only standard system libraries are used. It avoids handling complex dependencies and problems with library compatibility. The only thing you need is a C++ compiler :

- **Microsoft Visual Studio.NET and Visual Express Edition** : Use the project files and solution files provided in the CImg Library package (directory 'compilation/') to see how it works.
- **Intel ICL compiler** : Use the following command to compile a CImg-based program with ICL :
`icl /Ox hello_world.cpp user32.lib gdi32.lib`
- **g++ (MingW windows version)** : Use the following command to compile a CImg-based program with g++, on Windows :
`g++ -o hello_world.exe hello_world.cpp -O2 -lgdi32`
- **g++ (Linux version)** : Use the following command to compile a CImg-based program with g++, on Linux :
`g++ -o hello_world.exe hello_world.cpp -O2 -L/usr/X11R6/lib -lm -lpthread -lX11`

- **g++ (Solaris version)** : Use the following command to compile a CImg-based program with g++, on Solaris :
`g++ -o hello_world.exe hello_world.cpp -O2 -lm -lpthread -R/usr/X11R6/lib -lrt -lnsl -lsocket`
- **g++ (Mac OS X version)** : Use the following command to compile a CImg-based program with g++, on Mac OS X :
`g++ -o hello_world.exe hello_world.cpp -O2 -lm -lpthread -I/usr/X11R6/include -L/usr/X11R6/lib -lm -lpthread -lX11`
- **Dev-Cpp** : Use the project file provided in the CImg library package to see how it works.

If you are using other compilers and encounter problems, please [write me](#) since maintaining compatibility is one of the priorities of the CImg Library. Nevertheless, old compilers that do not respect the C++ standard will not support the CImg Library.

6.1.4 What's next ?

If you are ready to get more, and to start writing more serious programs with CImg, you are invited to go to the [Tutorial : Getting Started](#). section.

6.2 FAQ : Frequently Asked Questions.

6.2.1 FAQ Summary

- [General information and availability](#)
 - [What is the CImg Library ?](#)
 - [What platforms are supported ?](#)
 - [How is CImg distributed ?](#)
 - [What kind of people are concerned by CImg ?](#)
 - [What are the specificities of the CeCILL license ?](#)
 - [Who is behind CImg ?](#)
- [C++ related questions](#)
 - [What is the level of C++ knowledge needed to use CImg ?](#)
 - [How to use CImg in my own C++ program ?](#)
 - [Why is CImg entirely contained in a single header file ?](#)
- [Other resources](#)
 - [Translations](#)

6.2.2 1. General information and availability

6.2.2.1 1.1. What is the CImg Library ?

The CImg Library is an *open-source C++ library for image processing*.

It mainly consists in a (big) single header file [CImg.h](#) providing a set of C++ classes and functions that can be used in your own sources, to load/save, manage/process and display generic images. It's actually a very simple and pleasant toolkit for coding image processing stuff in C++ : Just include the header file `CImg.h`, and you are ready to handle images in your C++ programs.

6.2.2.2 1.2. What platforms are supported ?

Clmg has been designed with *portability* in mind. It is regularly tested on different architectures and compilers, and should also work on any decent OS having a decent C++ compiler. Before each release, the Clmg Library is compiled under these different configurations :

- PC Linux 32/64 bits, with g++.
- PC Windows 32/64 bits, with Visual C++ Express Edition.

Clmg has a minimal number of dependencies. In its minimal version, it can be compiled only with standard C++ headers. Anyway, it has interesting extension capabilities and can use external libraries to perform specific tasks more efficiently (Fourier Transform computation using FFTW for instance).

6.2.2.3 1.3. How is Clmg distributed ?

The Clmg Library is freely distributed as a complete .zip compressed package, hosted at the [Clmg server](#). The package is distributed under the [CeCILL license](#).

This package contains :

- The main library file [Clmg.h](#) (C++ header file).
- Several C++ source code showing [examples of using Clmg](#).
- A complete library documentation, in [PDF](#) format.
- Additional [library plug-ins](#) that can be used to extend library capabilities for specific uses.

The Clmg Library is a quite lightweight library which is easy to maintain (due to its particular structure), and thus has a fast rythm of release. A new version of the Clmg package is released approximately every three months.

6.2.2.4 1.4. What kind of people are concerned by Clmg ?

The Clmg library is an *image processing* library, primarily intended for computer scientists or students working in the fields of image processing or computer vision, and knowing bases of C++. As the library is handy and really easy to use, it can be also used by any programmer needing occasional tools for dealing with images in C++, since there are no standard library yet for this purpose.

6.2.2.5 1.5. What are the specificities of the CeCILL license ?

The [CeCILL license](#) governs the use of the Clmg Library. This is an *open-source* license which gives you rights to access, use, modify and redistribute the source code, under certain conditions. There are two different variants of the CeCILL license used in Clmg (namely [CeCILL](#) and [CeCILL-C](#), all open-source), corresponding to different constraints on the source files :

- The [CeCILL-C](#) license is the most permissive one, close to the *GNU LGPL license*, and *applies **only** on the main library file [Clmg.h](#)*. Basically, this license allows to use [Clmg.h](#) in a closed-source product without forcing you to redistribute the entire software source code. Anyway, if one modifies the [Clmg.h](#) source file, one has to redistribute the modified version of the file that must be governed by the same [CeCILL-C](#) license.
- The [CeCILL](#) license applies to all other files (source examples, plug-ins and documentation) of the Clmg Library package, and is close (even *compatible*) with the *GNU GPL license*. It *does not allow* the use of these files in closed-source products.

You are invited to read the complete descriptions of the the [CeCILL-C](#) and [CeCILL](#) licenses before releasing a software based on the Clmg Library.

6.2.2.6 1.6. Who is behind CImg ?

CImg has been started by [David Tschumperlé](#) at the beginning of his PhD thesis, in October 1999. He is still the main coordinator of the project. Since the first release, a growing number of contributors has appeared. Due to the very simple and compact form of the library, submitting a contribution is quite easy and can be fastly integrated into the supported releases. List of contributors can be found on the front page.

6.2.3 2. C++ related questions

6.2.3.1 2.1 What is the level of C++ knowledge needed to use CImg ?

The CImg Library has been designed using C++ templates and object-oriented programming techniques, but in a very accessible level. There are only public classes without any derivation (just like C structures) and there is at most one template parameter for each CImg class (defining the pixel type of the images). The design is simple but clean, making the library accessible even for non professional C++ programmers, while proposing strong extension capabilities for C++ experts.

6.2.3.2 2.2 How to use CImg in my own C++ program ?

Basically, you need to add these two lines in your C++ source code, in order to be able to work with CImg images :

```
#include "CImg.h"
using namespace cimg_library;
```

6.2.3.3 2.3 Why is CImg entirely contained in a single header file ?

People are often surprised to see that the complete code of the library is contained in a single (big) C++ header file [CImg.h](#). There are good practical and technical reasons to do that. Some arguments are listed below to justify this approach, so (I hope) you won't think this is a awkwardly C++ design of the CImg library :

- First, the library is based on *template datatypes* (images with generic pixel type), meaning that the programmer is free to decide what type of image he instantiates in his code. Even if there are roughly a limited number of fully supported types (basically, the "atomic" types of C++ : *unsigned char*, *int*, *float*, ...), this is *not imaginable* to pre-compile the library classes and functions for *all possible atomic datatypes*, since many functions and methods can have two or three arguments having different template parameters. This really means *a huge number* of possible combinations. The size of the object binary file generated to cover all possible cases would be just *colossal*. Is the STL library a pre-compiled one ? No, CImg neither. CImg is not using a classical *.cpp* and *.h* mechanism, just like the STL. Architectures of C++ *template-based* libraries are somewhat special in this sense. This is a proven technical fact.
- Second, why CImg does not have several header files, just like the STL does (one for each class for instance) ? This would be possible of course. There are only 4 classes in CImg, the two most important being *CImg<↔T>* and *CImgList<T>* representing respectively an image and a collection of images. But contrary to the STL library, these two CImg classes are strongly *inter-dependent*. All CImg algorithms are actually not defined as separate functions acting on containers (as the STL does with his header *<algorithm>*), but are directly methods of the image and image collection classes. This inter-dependence practically means that you will undoubtedly need these two main classes at the same time if you are using CImg. If they were defined in separate header files, you would be forced to include both of them. What is the gain then ? No gain. Concerning the two other classes : You can disable the third most important class *CImgDisplay* of the CImg library, by setting the compilation macro *cimg_display* to 0, avoiding thus to compile this class if you don't use display capabilities of CImg in your code. But to be honest, this is a quite small class and doing this doesn't save much compilation time. The last and fourth class is *CImgException*, which is only few lines long and is obviously required in almost all methods of CImg. Including this one is *mandatory*. As a consequence, having a single header file instead of several ones is just a way for you to avoid including all of them, without any consequences on compilation time. This is both good technical and practical reasons to do like this.

- Third, having a single header file has plenty of advantages : Simplicity for the user, and for the developers (maintenance is in fact easier). Look at the `CImg.h` file, it looks like a mess at a first glance, but it is in fact very well organized and structured. Finding pieces of code in CImg functions or methods is particularly easy and fast. Also, how about the fact that library installation problems just disappear ? Just bring `CImg.h` with you, put it in your source directory, and the library is ready to go !

I admit the compilation time of CImg-based programs can be sometime long, but don't think that it is due to the fact that you are using a single header file. Using several header files wouldn't arrange anything since you would need all of them. Having a pre-compiled library object would be the only solution to speed up compilation time, but it is not possible at all, due to the too much generic nature of the library.

6.2.4 3. Other resources

6.2.4.1 3.1 Translations

This FAQ has been translated to [Serbo-Croatian](#) language by [Web Geeks](#) .

6.3 Setting Environment Variables

The CImg library is a multiplatform library, working on a wide variety of systems. This implies the existence of some *environment variables* that must be correctly defined depending on your current system. Most of the time, the CImg Library defines these variables automatically (for popular systems). Anyway, if your system is not recognized, you will have to set the environment variables by hand. Here is a quick explanations of environment variables.

Setting the environment variables is done with the `#define` keyword. This setting must be done *before including the file `CImg.h`* in your source code. For instance, defining the environment variable `cimg_display` would be done like this :

```
#define cimg_display 0
#include "CImg.h"
...
```

Here are the different environment variables used by the CImg Library :

- **cimg_OS** : This variable defines the type of your Operating System. It can be set to **1** (*Unix*), **2** (*Windows*), or **0** (*Other configuration*). It should be actually auto-detected by the CImg library. If this is not the case (`cimg_OS=0`), you will probably have to tune the environment variables described below.
- **cimg_display** : This variable defines the type of graphical library used to display images in windows. It can be set to 0 (no display library available), **1** (X11-based display), **2** (Windows-GDI display), or **3** (SDL3). If you are running on a system without X11 or Windows-GDI ability, please set this variable to 0. This will disable the display support, since the CImg Library doesn't contain the necessary code to display images on systems other than X11 or Windows GDI.
- **cimg_use_vt100** : This variable tells the library if the system terminal has VT100 color capabilities. It can be *defined* or *not defined*. Define this variable to get colored output on your terminal, when using the CImg Library.
- **cimg_verbosity** : This variable defines the level of run-time debug messages that will be displayed by the CImg Library. It can be set to 0 (no debug messages), 1 (normal debug messages displayed on standard error), 2 (normal debug messages displayed in modal windows, which is the default value), or 3 (high debug messages). Note that setting this value to 3 may slow down your program since more debug tests are made by the library (particularly to check if pixel access is made outside image boundaries). See also `CImgException` to better understand how debug messages are working.

- **cimg_plugin** : This variable tells the library to use a plugin file to add features to the CImg<T> class. Define it with the path of your plugin file, if you want to add member functions to the CImg<T> class, without having to modify directly the "`CImg.h`" file. An include of the plugin file is performed in the CImg<T> class. If `cimg_plugin` is not specified (default), no include is done.
- **cimglist_plugin** : Same as `cimg_plugin`, but to add features to the CImgList<T> class.
- **cimgdisplay_plugin** : Same as `cimg_plugin`, but to add features to the CImgDisplay<T> class.

All these compilation variables can be checked, using the function `cimg_library::cimg::info()`, which displays a list of the different configuration variables and their values on the standard error output.

6.4 How to use CImg library with Visual C++ 2005 Express Edition ?.

6.4.1 How to use CImg library with Visual C++ 2005 Express Edition ?

This section has been written by Vincent Garcia and Alexandre Fournier from I3S/Sophia_Antipolis.

- Download CImg library
- Download and install Visual C++ 2005 Express Edition
- Download and install Microsoft Windows SDK
- Configure Visual C++ to take into account Microsoft SDK
 - 1. Go to menu "Tools -> options"
 - 2. Select option "Projects and Solutions -> VC++ Directories"
 - 3. In the select list "Show directories for", choose "include files", and add `C:\Program Files\Microsoft Platform SDK\Include` (adapt if needed)
 - 4. In the select list "Show directories for", choose "library files", and add `C:\Program Files\Microsoft Platform SDK\Lib` (adapt if needed) Edit file `C:\Program Files\Microsoft Visual Studio 8\VC\VCProject Defaults\corewin_express.vsprops` (adapt if needed)
 - 6. 7. Replace the line `AdditionalDependencies="kernel32.lib" />` by `AdditionalDependencies="kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib" />`
- Restart Visual C++
- Import CImg library in your main file

6.5 Tutorial : Getting Started.

Let's start to write our first program to get the idea. This will demonstrate how to load and create images, as well as handle image display and mouse events. Assume we want to load a color image `lena.jpg`, smooth it, display it in a windows, and enter an event loop so that clicking a point in the image will draw the (R,G,B) intensity profiles of the corresponding image line (in another window). Yes, that sounds quite complex for a first code, but don't worry, it will be very simple using the CImg library ! Well, just look at the code below, it does the task :

```
#include "CImg.h"
using namespace cimg_library;

int main() {
    CImg<unsigned char> image("lena.jpg"), visu(500,400,1,3,0);
    const unsigned char red[] = { 255,0,0 }, green[] = { 0,255,0 }, blue[] = { 0,0,255 };
    image.blur(2.5);
```

```

CImgDisplay main_disp(image,"Click a point"), draw_disp(visu,"Intensity profile");
while (!main_disp.is_closed() && !draw_disp.is_closed()) {
    main_disp.wait();
    if (main_disp.button() && main_disp.mouse_y()>=0) {
        const int y = main_disp.mouse_y();
        visu.fill(0).draw_graph(image.get_crop(0,y,0,0,image.width()-1,y,0,0),red,1,1,0,255,0);
        visu.draw_graph(image.get_crop(0,y,0,1,image.width()-1,y,0,1),green,1,1,0,255,0);
        visu.draw_graph(image.get_crop(0,y,0,2,image.width()-1,y,0,2),blue,1,1,0,255,0).display(draw_disp);
    }
}
return 0;
}

```

Here is a screenshot of the resulting program :

And here is the detailed explanation of the source, line by line :

```
#include "Cimg.h"
```

Include the main and only header file of the CImg library.

```
using namespace cimg_library;
```

Use the library namespace to ease the declarations afterward.

```
int main() {
```

Definition of the main function.

```
CImg<unsigned char> image("lena.jpg"), visu(500,400,1,3,0);
```

Creation of two instances of images of unsigned char pixels. The first image `image` is initialized by reading an image file from the disk. Here, `lena.jpg` must be in the same directory as the current program. Note that you must also have installed the *ImageMagick* package in order to be able to read JPG images. The second image `visu` is initialized as a black color image with dimension `dx=500`, `dy=400`, `dz=1` (here, it is a 2D image, not a 3D one), and `dv=3` (each pixel has 3 'vector' channels R,G,B). The last argument in the constructor defines the default value of the pixel values (here 0, which means that `visu` will be initially black).

```
const unsigned char red[] = { 255,0,0 }, green[] = { 0,255,0 }, blue[] = { 0,0,255 };
```

Definition of three different colors as array of unsigned char. This will be used to draw plots with different colors.

```
image.blur(2.5);
```

Blur the image, with a gaussian blur and a standard variation of 2.5. Note that most of the CImg functions have two versions : one that acts in-place (which is the case of `blur`), and one that returns the result as a new image (the name of the function begins then with `get_`). In this case, one could have also written `image = image.get_<_blur(2.5)`; (more expensive, since it needs an additional copy operation).

```
CImgDisplay main_disp(image,"Click a point"), draw_disp(visu,"Intensity profile");
```

Creation of two display windows, one for the input image `image`, and one for the image `visu` which will be display intensity profiles. By default, CImg displays handles events (mouse,keyboard,..). On Windows, there is a way to create fullscreen displays.

```
while (!main_disp.is_closed() && !draw_disp.is_closed()) {
```

Enter the event loop, the code will exit when one of the two display windows is closed.

```
main_disp.wait();
```

Wait for an event (mouse, keyboard,..) in the display window `main_disp`.

```
if (main_disp.button() && main_disp.mouse_y()>=0) {
```

Test if the mouse button has been clicked on the image area. One may distinguish between the 3 different mouse buttons, but in this case it is not necessary

```
const int y = main_disp.mouse_y();
```

Get the image line y-coordinate that has been clicked.

```
visu.fill(0).draw_graph(image.get_crop(0,y,0,0,image.width()-1,y,0,0),red,1,0,256,0);
```

This line illustrates the pipeline property of most of the CImg class functions. The first function `fill(0)` simply sets all pixel values with 0 (i.e. clear the image `visu`). The interesting thing is that it returns a reference to `visu` and then, can be pipelined with the function `draw_graph()` which draws a plot in the image `visu`. The plot data are given by another image (the first argument of `draw_graph()`). In this case, the given image is the red-component of the line `y` of the original image, retrieved by the function `get_crop()` which returns a sub-image of

the image `image`. Remember that images coordinates are 4D (x,y,z,c) and for color images, the R,G,B channels are respectively given by `v=0`, `v=1` and `v=2`.

```
visu.draw_graph(image.get_crop(0,y,0,1,image.width()-1,y,0,1),green,1,0,256,0);
```

Plot the intensity profile for the green channel of the clicked line.

```
visu.draw_graph(image.get_crop(0,y,0,2,image.width()-1,y,0,2),blue,1,0,256,0).display(draw_disp);
```

Same thing for the blue channel. Note how the function (which return a reference to `visu`) is pipelined with the function `display()` that just paints the image `visu` in the corresponding display window.

```
...till the end
```

I don't think you need more explanations !

As you have noticed, the `CImg` library allows to write very small and intuitive code. Note also that this source will perfectly work on Unix and Windows systems. Take also a look to the examples provided in the `CImg` package (directory `examples/`). It will show you how `CImg`-based code can be surprisingly small. Moreover, there is surely one example close to what you want to do. A good start will be to look at the file `CImg_demo.cpp` which contains small and various examples of what you can do with the `CImg` Library. All `CImg` classes are used in this source, and the code can be easily modified to see what happens.

6.6 Using Image Loops.

The `CImg` Library provides different macros that define useful iterative loops over an image. Basically, it can be used to replace one or several `for( . . )` instructions, but it also proposes interesting extensions to classical loops. Below is a list of all existing loop macros, classified in four different categories :

- [Loops over the pixel buffer](#)
- [Loops over image dimensions](#)
- [Loops over interior regions and borders.](#)
- [Loops using neighborhoods.](#)

6.6.1 Loops over the pixel buffer

Loops over the pixel buffer are really basic loops that iterate a pointer on the pixel data buffer of a `cimg_library::CImg` image. Two macros are defined for this purpose :

- **`cimg_for(img,ptr,T)`** : This macro loops over the pixel data buffer of the image `img`, using a pointer `T* ptr`, starting from the beginning of the buffer (first pixel) till the end of the buffer (last pixel).
 - `img` must be a (non empty) `cimg_library::CImg` image of pixels `T`.
 - `ptr` is a pointer of type `T*`. This kind of loop should not appear a lot in your own source code, since this is a low-level loop and many functions of the `CImg` class may be used instead. Here is an example of use :


```
CImg<float> img(320,200);
cimg_for(img,ptr,float) { *ptr=0; }           // Equivalent to 'img.fill(0);'
```
- **`cimg_rof(img,ptr,T)`** : This macro does the same as `cimg_for()` but from the end to the beginning of the pixel buffer.
- **`cimg_foroff(img,off)`** : This macro loops over the pixel data buffer of the image `img`, using an offset , starting from the beginning of the buffer (first pixel, `off=0`) till the end of the buffer (last pixel value, `off = img.size()-1`).
 - `img` must be a (non empty) `cimg_library::CImg<T>` image of pixels `T`.
 - `off` is an inner-loop variable, only defined inside the scope of the loop.

Here is an example of use :

```
CImg<float> img(320,200);
cimg_foroff(img,off) { img[off]=0; }           // Equivalent to 'img.fill(0);'
```

6.6.2 Loops over image dimensions

The following loops are probably the most used loops in image processing programs. They allow to loop over the image along one or several dimensions, along a raster scan course. Here is the list of such loop macros for a single dimension :

- **cimg_forX(img,x)** : equivalent to : `for (int x = 0; x<img.width(); ++x).`
- **cimg_forY(img,y)** : equivalent to : `for (int y = 0; y<img.height(); ++y).`
- **cimg_forZ(img,z)** : equivalent to : `for (int z = 0; z<img.depth(); ++z).`
- **cimg_forC(img,c)** : equivalent to : `for (int c = 0; c<img.spectrum(); ++c).`

Combinations of these macros are also defined as other loop macros, allowing to loop directly over 2D, 3D or 4D images :

- **cimg_forXY(img,x,y)** : equivalent to : `cimg_forY(img, y) cimg_forX(img, x).`
- **cimg_forXZ(img,x,z)** : equivalent to : `cimg_forZ(img, z) cimg_forX(img, x).`
- **cimg_forYZ(img,y,z)** : equivalent to : `cimg_forZ(img, z) cimg_forY(img, y).`
- **cimg_forXC(img,x,c)** : equivalent to : `cimg_forC(img, c) cimg_forX(img, x).`
- **cimg_forYC(img,y,c)** : equivalent to : `cimg_forC(img, c) cimg_forY(img, y).`
- **cimg_forZC(img,z,c)** : equivalent to : `cimg_forC(img, c) cimg_forZ(img, z).`
- **cimg_forXYZ(img,x,y,z)** : equivalent to : `cimg_forZ(img, z) cimg_forXY(img, x, y).`
- **cimg_forXYC(img,x,y,c)** : equivalent to : `cimg_forC(img, c) cimg_forXY(img, x, y).`
- **cimg_forXZC(img,x,z,c)** : equivalent to : `cimg_forC(img, c) cimg_forXZ(img, x, z).`
- **cimg_forYZC(img,y,z,c)** : equivalent to : `cimg_forC(img, c) cimg_forYZ(img, y, z).`
- **cimg_forXYZC(img,x,y,z,c)** : equivalent to : `cimg_forC(img, c) cimg_forXYZ(img, x, y, z).`
- For all these loops, `x,y,z` and `v` are inner-defined variables only visible inside the scope of the loop. They don't have to be defined before the call of the macro.
- `img` must be a (non empty) [cimg_library::CImg](#) image.

Here is an example of use that creates an image with a smooth color gradient :

```
CImg<unsigned char> img(256,256,1,3); // Define a 256x256 color image
cimg_forXYC(img,x,y,c) { img(x,y,c) = (x+y)*(c+1)/6; }
img.display("Color gradient");
```

6.6.3 Loops over interior regions and borders.

Similar macros are also defined to loop only on the border of an image, or inside the image (excluding the border). The border may be several pixel wide :

- **cimg_for_insideX(img,x,n)** : Loop along the x-axis, except for pixels inside a border of `n` pixels wide.
- **cimg_for_insideY(img,y,n)** : Loop along the y-axis, except for pixels inside a border of `n` pixels wide.
- **cimg_for_insideZ(img,z,n)** : Loop along the z-axis, except for pixels inside a border of `n` pixels wide.

- **cimg_for_insideC(img,c,n)** : Loop along the c-axis, except for pixels inside a border of *n* pixels wide.
- **cimg_for_insideXY(img,x,y,n)** : Loop along the (x,y)-axes, excepted for pixels inside a border of *n* pixels wide.
- **cimg_for_insideXYZ(img,x,y,z,n)** : Loop along the (x,y,z)-axes, excepted for pixels inside a border of *n* pixels wide.

And also :

- **cimg_for_borderX(img,x,n)** : Loop along the x-axis, only for pixels inside a border of *n* pixels wide.
- **cimg_for_borderY(img,y,n)** : Loop along the y-axis, only for pixels inside a border of *n* pixels wide.
- **cimg_for_borderZ(img,z,n)** : Loop along the z-axis, only for pixels inside a border of *n* pixels wide.
- **cimg_for_borderC(img,c,n)** : Loop along the c-axis, only for pixels inside a border of *n* pixels wide.
- **cimg_for_borderXY(img,x,y,n)** : Loop along the (x,y)-axes, only for pixels inside a border of *n* pixels wide.
- **cimg_for_borderXYZ(img,x,y,z,n)** : Loop along the (x,y,z)-axes, only for pixels inside a border of *n* pixels wide.
- For all these loops, *x*, *y*, *z* and *c* are inner-defined variables only visible inside the scope of the loop. They don't have to be defined before the call of the macro.
- *img* must be a (non empty) [cimg_library::CImg](#) image.
- The constant *n* stands for the size of the border.

Here is an example of use, to create a 2d grayscale image with two different intensity gradients :

```
CImg<> img(256,256);
cimg_for_insideXY(img,x,y,50) img(x,y) = x+y;
cimg_for_borderXY(img,x,y,50) img(x,y) = x-y;
img.display();
```

6.6.4 Loops using neighborhoods.

Inside an image loop, it is often useful to get values of neighborhood pixels of the current pixel at the loop location. The CImg Library provides a very smart and fast mechanism for this purpose, with the definition of several loop macros that remember the neighborhood values of the pixels. The use of these macros can highly optimize your code, and also simplify your program.

6.6.4.1 Neighborhood-based loops for 2D images

For 2D images, the neighborhood-based loop macros are :

- **cimg_for2x2(img,x,y,z,c,l,T)** : Loop along the (x,y)-axes using a centered 2x2 neighborhood.
- **cimg_for3x3(img,x,y,z,c,l,T)** : Loop along the (x,y)-axes using a centered 3x3 neighborhood.
- **cimg_for4x4(img,x,y,z,c,l,T)** : Loop along the (x,y)-axes using a centered 4x4 neighborhood.
- **cimg_for5x5(img,x,y,z,c,l,T)** : Loop along the (x,y)-axes using a centered 5x5 neighborhood.

For all these loops, *x* and *y* are inner-defined variables only visible inside the scope of the loop. They don't have to be defined before the call of the macro. *img* is a non empty CImg<T> image. *z* and *c* are constants that define on which image slice and vector channel the loop must apply (usually both 0 for grayscale 2D images). Finally, *T* is the 2x2, 3x3, 4x4 or 5x5 neighborhood of type *T* that will be updated with the correct pixel values during the loop (see [Defining neighborhoods](#)).

6.6.4.2 Neighborhood-based loops for 3D images

For 3D images, the neighborhood-based loop macros are :

- **cimg_for2x2x2(img,x,y,z,c,I,T)** : Loop along the (x,y,z)-axes using a centered 2x2x2 neighborhood.
- **cimg_for3x3x3(img,x,y,z,c,I,T)** : Loop along the (x,y,z)-axes using a centered 3x3x3 neighborhood.

For all these loops, *x*, *y* and *z* are inner-defined variables only visible inside the scope of the loop. They don't have to be defined before the call of the macro. *img* is a non empty CImg<T> image. *c* is a constant that defines on which image channel the loop must apply (usually 0 for grayscale 3D images). Finally, *I* is the 2x2x2 or 3x3x3 neighborhood of type *T* that will be updated with the correct pixel values during the loop (see [Defining neighborhoods](#)).

6.6.4.3 Defining neighborhoods

A neighborhood is defined as an instance of a class having operator[] defined. This particularly includes classical C-array, as well as CImg<T> objects.

For instance, a 3x3 neighborhood can be defined either as a 'float[9]' or a 'CImg<float>(3,3)' variable.

6.6.4.4 Using alternate variable names

There are also some useful macros that can be used to define variables that reference the neighborhood elements. There are :

- **CImg_2x2(I,type)** : Define a 2x2 neighborhood named *I*, of type *type*.
- **CImg_3x3(I,type)** : Define a 3x3 neighborhood named *I*, of type *type*.
- **CImg_4x4(I,type)** : Define a 4x4 neighborhood named *I*, of type *type*.
- **CImg_5x5(I,type)** : Define a 5x5 neighborhood named *I*, of type *type*.
- **CImg_2x2x2(I,type)** : Define a 2x2x2 neighborhood named *I*, of type *type*.
- **CImg_3x3x3(I,type)** : Define a 3x3x3 neighborhood named *I*, of type *type*.

Actually, *I* is a *generic name* for the neighborhood. In fact, these macros declare a *set* of new variables. For instance, defining a 3x3 neighborhood CImg_3x3(*I*, *float*) declares 9 different float variables *Ipp*, *Icp*, *Inp*, *Ipc*, *Icc*, *Icn*, *Ipn*, *Icn*, *Inn* which correspond to each pixel value of a 3x3 neighborhood. Variable indices are *p*, *c* or *n*, and stand respectively for '*previous*', '*current*' and '*next*'. First indice denotes the *x-axis*, second indice denotes the *y-axis*. Then, the names of the variables are directly related to the position of the corresponding pixels in the neighborhood. For 3D neighborhoods, a third indice denotes the *z-axis*. Then, inside a neighborhood loop, you will have the following equivalence :

- *Ipp* = *img*(*x*-1, *y*-1)
- *Icn* = *img*(*x*, *y*+1)
- *Inp* = *img*(*x*+1, *y*-1)
- *Inpc* = *img*(*x*+1, *y*-1, *z*)
- *Ippn* = *img*(*x*-1, *y*-1, *z*+1)

- and so on...

For bigger neighborhoods, such as 4x4 or 5x5 neighborhoods, two additional indices are introduced : a (stands for 'after') and b (stands for 'before'), so that :

- `Ibb = img(x-2,y-2)`
- `Ina = img(x+1,y+2)`
- and so on...

The value of a neighborhood pixel outside the image range (image border problem) is automatically set to the same values as the nearest valid pixel in the image (this is also called the *Neumann border condition*).

6.6.4.5 Example codes

More than a long discussion, the above example will demonstrate how to compute the gradient norm of a 3D volume using the `cimg_for3x3x3()` loop macro :

```
CImg<float> volume("IRM.hdr"); // Load an IRM volume from an Analyze7.5 file
CImg_3x3x3(I,float); // Define a 3x3x3 neighborhood
CImg<float> gradnorm(volume); // Create an image with same size as 'volume'
cimg_for3x3x3(volume,x,y,z,0,I,float) { // Loop over the volume, using the neighborhood I
    const float ix = 0.5f*(Icc-Ipcc); // Compute the derivative along the x-axis.
    const float iy = 0.5f*(Icnc-Icpcc); // Compute the derivative along the y-axis.
    const float iz = 0.5f*(Iccn-Iccp); // Compute the derivative along the z-axis.
    gradnorm(x,y,z) = std::sqrt(ix*ix+iy*iy+iz*iz); // Set the gradient norm in the destination image
}
gradnorm.display("Gradient norm");
```

And the following example shows how to deal with neighborhood references to blur a color image by averaging pixel values on a 5x5 neighborhood.

```
CImg<unsigned char> src("image_color.jpg"), dest(src,false); // Image definitions.
typedef unsigned char uchar; // Avoid space in the second parameter of the macro CImg_5x5x1
// below.
CImg<> N(5,5); // Define a 5x5 neighborhood as a 5x5 image.
cimg_forC(src,k) // Standard loop on color channels
    cimg_for5x5(src,x,y,0,k,N,float) // 5x5 neighborhood loop.
        dest(x,y,k) = N.sum()/(5*5); // Averaging pixels to filter the color image.
CImgList<unsigned char> visu(src,dest);
visu.display("Original + Filtered"); // Display both original and filtered image.
```

As you can see, explaining the use of the `CImg` neighborhood macros is actually more difficult than using them !

6.7 Using Display Windows.

When opening a display window, you can choose the way the pixel values will be normalized before being displayed on the screen. Screen displays only support color values between [0,255], and some

When displaying an image into the display window using `CImgDisplay::display()`, values of the image pixels can be eventually linearly normalized between [0,255] for visualization purposes. This may be useful for instance when displaying `CImg<double>` images with pixel values between [0,1]. The normalization behavior depends on the value of `normalize` which can be either 0,1 or 2 :

- 0 : No pixel normalization is performed when displaying an image. This is the fastest process, but you must be sure your displayed image have pixel values inside the range [0,255].
- 1 : Pixel value normalization is done for each new image display. Image pixels are not modified themselves, only displayed pixels are normalized.
- 2 : Pixel value normalization is done for the first image display, then the normalization parameters are kept and used for all the next image displays.

6.8 How pixel data are stored with CImg.

First, `CImg<T>` are *very* basic structures, which means that there are no memory tricks, weird memory alignments or disk caches used to store pixel data of images. When an image is instanced, all its pixel values are stored in memory at the same time (yes, you should avoid working with huge images when dealing with CImg, if you have only 64kb of RAM).

A `CImg<T>` is basically a 4th-dimensional array (width,height,depth,dim), and its pixel data are stored linearly in a single memory buffer of general size (width*height*depth*dim). Nothing more, nothing less. The address of this memory buffer can be retrieved by the function `CImg<T>::data()`. As each image value is stored as a type `T` (`T` being known by the programmer of course), this pointer is a `T*`, or a `'const T*'` if your image is `'const'`. so, `'T *ptr = img.data()'` gives you the pointer to the first value of the image `'img'`. The overall size of the used memory for one instance image (in bytes) is then `'width*height*depth*dim*sizeof(T)'`.

Now, the ordering of the pixel values in this buffer follows these rules : The values are *not* interleaved, and are ordered first along the X,Y,Z and V axis respectively (corresponding to the width,height,depth,dim dimensions), starting from the upper-left pixel to the bottom-right pixel of the instane image, with a classical scanline run.

So, a color image with `dim=3` and `depth=1`, will be stored in memory as :

`R1R2R3R4R5R6.....G1G2G3G4G5G6.....B1B2B3B4B5B6....` (i.e following a 'planar' structure)

and *not* as `R1G1B1R2G2B2R3G3B3...` (interleaved channels), where `R1 = img(0,0,0,0)` is the first upper-left pixel of the red component of the image, `R2 = img(1,0,0,0)`, `G1 = img(0,0,0,1)`, `G2 = img(1,0,0,1)`, `B1 = img(0,0,0,2)`, and so on...

Another example, a (1x5x1x1) `CImg<T>` (column vector `A`) will be stored as : `A1A2A3A4A5` where `A1 = img(0,0)`, `A2 = img(0,1)`, ... , `A5 = img(0,4)`.

As you see, it is *very* simple and intuitive : no interleaving, no padding, just simple. This is cool not only because it is simple, but this has in fact a number of interesting properties. For instance, a 2D color image is stored in memory exactly as a 3D scalar image having a `depth=3`, meaning that when you are dealing with 2D color images, you can write `'img(x,y,k)'` instead of `'img(x,y,0,k)'` to access the `k`th channel of the (x,y) pixel. More generally, if you have one dimension that is 1 in your image, you can just skip it in the call to the operator(). Similarly, values of a column vector stored as an image with `width=depth=spectrum=1` can be accessed by `'img(y)'` instead of `'img(0,y)'`. This is very convenient.

Another cool thing is that it allows you to work easily with 'shared' images. A shared image is a `CImg<T>` instance that shares its memory with another one (the 'base' image). Destroying a shared image does nothing in fact. Shared images is a convenient way of modifying only *portions* (consecutive in memory) of an image. For instance, if `'img'` is a 2D color image, you can write :

```
img.get_shared_channel(0).blur(2); img.get_shared_channels(1,2).mirror('x');
```

which just blur the red channel of the image, and mirror the two others along the X-axis. This is possible since channels of an image are not interleaved but are stored as different consecutive planes in memory, so you see that constructing a shared image is possible (and trivial).

6.9 Files IO in CImg.

The CImg Library can NATIVELY handle the following file formats :

- RAW : consists in a very simple header (in ascii), then the image data.
- ASC (Ascii)
- HDR (Analyze 7.5)
- INR (Inrimage)
- PPM/PGM (Portable Pixmap)
- BMP (uncompressed)
- PAN (Pandore-5)
- DLM (Matlab ASCII)

If ImageMagick is installed, The CImg Library can save image in formats handled by ImageMagick : JPG, GIF, PNG, TIF,...

6.10 Retrieving Command Line Arguments.

The CImg library offers facilities to retrieve command line arguments in a console-based program, as it is a commonly needed operation. Three macros `cimg_usage()`, `cimg_help()` and `cimg_option()` are defined for this purpose. Using these macros allows to easily retrieve options values from the command line. Invoking the compiled executable with the option `-h` or `--help` will automatically display the program usage, followed by the list of requested options.

6.10.1 The `cimg_usage()` macro

The macro `cimg_usage(usage)` may be used to describe the program goal and usage. It is generally inserted one time after the `int main(int argc, char **argv)` definition.

Parameters

<code>usage</code>	: A string describing the program goal and usage.
--------------------	---

Precondition

The function where `cimg_usage()` is used must have correctly defined `argc` and `argv` variables.

6.10.2 The `cimg_help()` macro

The macro `cimg_help(str)` will display the string `str` only if the `-help` or `--help` option are invoked when running the program.

6.10.3 The `cimg_option()` macro

The macro `cimg_option(name, default, usage)` may be used to retrieve an option value from the command line.

Parameters

<i>name</i>	: The name of the option to be retrieved from the command line.
<i>default</i>	: The default value returned by the macro if no options <i>name</i> has been specified when running the program.
<i>usage</i>	: A brief explanation of the option. If <code>usage==0</code> , the option won't appear on the option list when invoking the executable with options <code>-h</code> or <code>--help</code> (hidden option).

Returns

`cimg_option()` returns an object that has the *same type* as the default value `default`. The return value is equal to the one specified on the command line. If no such option have been specified, the return value is equal to the default value `default`. Warning, this can be confusing in some situations (look at the end of the next section).

Precondition

The function where `cimg_option()` is used must have correctly defined `argc` and `argv` variables.

6.10.4 Example of use

The code below uses the macros `cimg_usage()` and `cimg_option()`. It loads an image, smoothes it and quantifies it with a specified number of values.

```
#include "CImg.h"
using namespace cimg_library;
int main(int argc, char **argv) {
    cimg_usage("Retrieve command line arguments");
    const char* filename = cimg_option("-i", "image.gif", "Input image file");
    const char* output = cimg_option("-o", (char*)0, "Output image file");
    const double sigma = cimg_option("-s", 1.0, "Standard variation of the gaussian smoothing");
    const int nblevels = cimg_option("-n", 16, "Number of quantification levels");
    const bool hidden = cimg_option("-hidden", false, 0); // This is a hidden option

    CImg<unsigned char> img(filename);
    img.blur(sigma).quantize(nblevels);
    if (output) img.save(output); else img.display("Output image");
    if (hidden) std::fprintf(stderr, "You found me !\n");
    return 0;
}
```

Invoking the corresponding executable with `test -h -hidden -n 20 -i foo.jpg` will display :

```
./test -h -hidden -n 20 -i foo.jpg

test : Retrieve command line arguments (Oct 16 2004, 12:34:26)

-i      = foo.jpg      : Input image file
-o      = 0            : Output image file
-s      = 1            : Standard variation of the gaussian smoothing
-n      = 20           : Number of quantification levels

You found me !
```

Warning

As the type of object returned by the macro `cimg_option(option, default, usage)` is defined by the type of `default`, undesired casts may appear when writing code such as :

```
const double sigma = cimg_option("-val", 0, "A floating point value");
```

In this case, `sigma` will always be equal to an integer (since the default value `0` is an integer). When passing a float value on the command line, a *float to integer* cast is then done, truncating the given parameter to an integer value (this is surely not a desired behavior). You must specify `0.0` as the default value in this case.

6.10.5 How to learn more about command line options ?

You should take a look at the examples `examples/gmic.cpp` provided in the CImg Library package. This is a command line based image converter which intensively uses the `cimg_option()` and `cimg_usage()` macros to retrieve command line parameters.

Chapter 7

Namespace Documentation

7.1 cimg_library Namespace Reference

Contains *all classes and functions* of the CImg library.

Namespaces

- namespace [cimg](#)

Contains low-level functions and variables of the CImg Library.

Classes

- struct [CImg](#)

Class representing an image (up to 4 dimensions wide), where each pixel is of type T.

- struct [CImgDisplay](#)

Allow the creation of windows, display images on them and manage user events (keyboard, mouse and windows events).

- struct [CImgException](#)

Instances of [CImgException](#) are thrown when errors are encountered in a CImg function call.

- struct [CImgList](#)

Represents a list of images CImg<T>.

7.1.1 Detailed Description

Contains *all classes and functions* of the CImg library.

This namespace is defined to avoid function and class names collisions that should occur with the inclusion of other C++ header files. Regardless, collisions are rare, and you should typically start most of your CImg-based programs with

```
#include "CImg.h"
using namespace cimg_library;
```

to simplify the declaration of CImg Library objects afterwards.

7.2 cimg_library::cimg Namespace Reference

Contains low-level functions and variables of the CImg Library.

Functions

- `std::FILE * output (std::FILE *file)`
Get/set default output stream for the CImg library messages.
- `void info ()`
Print information about CImg environment variables.
- `template<typename T >`
`void unused (const T &)`
Avoid warning messages due to unused parameters. This function intentionally does nothing.
- `unsigned int & exception_mode (const unsigned int mode)`
Set current CImg exception mode.
- `unsigned int & exception_mode ()`
Return current CImg exception mode.
- `unsigned int & openmp_mode (const unsigned int mode)`
Set current CImg openmp mode.
- `unsigned int & openmp_mode ()`
Return current CImg openmp mode.
- `const char * user_agent (const char *const str)`
Set/Get a global user_agent value.
- `int dialog (const char *const title, const char *const msg, const char *const button1_label, const char *const button2_label, const char *const button3_label, const char *const button4_label, const char *const button5_label, const char *const button6_label, const bool is_centered)`
Display a simple dialog box, and wait for the user's response [specialization].
- `double eval (const char *const expression, const double x, const double y, const double z, const double c)`
Evaluate math expression.
- `void create_directory (const char *const dirname, const bool force_overwrite)`
Create a directory.
- `void warn (const char *const format,...)`
Display a warning message on the default output stream.
- `int system (const char *const command, const char *const module_name=0, const bool is_verbose=false)`
- `template<typename T >`
`T & temporary (const T &)`
Return a reference to a temporary variable of type T.
- `template<typename T >`
`void swap (T &a, T &b)`
Exchange values of variables a and b.
- `template<typename T1 , typename T2 >`
`void swap (T1 &a1, T1 &b1, T2 &a2, T2 &b2)`
Exchange values of variables (a1,a2) and (b1,b2).
- `template<typename T1 , typename T2 , typename T3 >`
`void swap (T1 &a1, T1 &b1, T2 &a2, T2 &b2, T3 &a3, T3 &b3)`
Exchange values of variables (a1,a2,a3) and (b1,b2,b3).
- `template<typename T1 , typename T2 , typename T3 , typename T4 >`
`void swap (T1 &a1, T1 &b1, T2 &a2, T2 &b2, T3 &a3, T3 &b3, T4 &a4, T4 &b4)`
Exchange values of variables (a1,a2,...,a4) and (b1,b2,...,b4).
- `template<typename T1 , typename T2 , typename T3 , typename T4 , typename T5 >`
`void swap (T1 &a1, T1 &b1, T2 &a2, T2 &b2, T3 &a3, T3 &b3, T4 &a4, T4 &b4, T5 &a5, T5 &b5)`

Exchange values of variables (a1,a2,...,a5) and (b1,b2,...,b5).

- `template<typename T1 , typename T2 , typename T3 , typename T4 , typename T5 , typename T6 >`
`void swap (T1 &a1, T1 &b1, T2 &a2, T2 &b2, T3 &a3, T3 &b3, T4 &a4, T4 &b4, T5 &a5, T5 &b5, T6 &a6, T6 &b6)`

Exchange values of variables (a1,a2,...,a6) and (b1,b2,...,b6).

- `template<typename T1 , typename T2 , typename T3 , typename T4 , typename T5 , typename T6 , typename T7 >`
`void swap (T1 &a1, T1 &b1, T2 &a2, T2 &b2, T3 &a3, T3 &b3, T4 &a4, T4 &b4, T5 &a5, T5 &b5, T6 &a6, T6 &b6, T7 &a7, T7 &b7)`

Exchange values of variables (a1,a2,...,a7) and (b1,b2,...,b7).

- `template<typename T1 , typename T2 , typename T3 , typename T4 , typename T5 , typename T6 , typename T7 , typename T8 >`
`void swap (T1 &a1, T1 &b1, T2 &a2, T2 &b2, T3 &a3, T3 &b3, T4 &a4, T4 &b4, T5 &a5, T5 &b5, T6 &a6, T6 &b6, T7 &a7, T7 &b7, T8 &a8, T8 &b8)`

Exchange values of variables (a1,a2,...,a8) and (b1,b2,...,b8).

- `bool endianness ()`

Return the endianness of the current architecture.

- `template<typename T >`
`void invert_endianness (T *const buffer, const unsigned long size)`

Reverse endianness of all elements in a memory buffer.

- `template<typename T >`
`T & invert_endianness (T &a)`

Reverse endianness of a single variable.

- `unsigned long long time ()`

Return the value of a system timer, with a millisecond precision.

- `unsigned long long tic ()`

Start tic/toc timer for time measurement between code instructions.

- `unsigned long long toc ()`

End tic/toc timer and displays elapsed time from last call to tic().

- `void sleep (const unsigned int milliseconds)`

Sleep for a given numbers of milliseconds.

- `unsigned int wait (const unsigned int milliseconds)`

Wait for a given number of milliseconds since the last call to wait().

- `template<typename T >`
`T cut (const T &val, const T &val_min, const T &val_max)`

Cut (i.e. clamp) value in specified interval.

- `template<typename T >`
`T bsl (const T &a, const int n=1)`

Bitwise shift value to the left (safeguarded against overflow).

- `template<typename T >`
`T bsr (const T &a, const int n=1)`

Bitwise shift value to the right (safeguarded against overflow).

- `template<typename T >`
`T rol (const T &a, const unsigned int n=1)`

Bitwise-rotate value on the left.

- `template<typename T >`
`T ror (const T &a, const unsigned int n=1)`

Bitwise-rotate value on the right.

- `template<typename T >`
`T abs (const T &a)`

Return absolute value of a value.

- `template<typename T >`
`T frac (const T x)`

Return fractional part of a value.

- `double acosh (const double x)`

- Return hyperbolic arcosine of a value.*

 - **double asinh** (const double x)

Return hyperbolic arcsine of a value.

 - **double atanh** (const double x)

Return hyperbolic arctangent of a value.

 - **double sinc** (const double x)

Return the sinc of a given value.

 - **double log2** (const double x)

Return base-2 logarithm of a value.

 - template<typename T >
T sqr (const T &val)

Return square of a value.

 - template<typename T >
double cbrt (const T &x)

Return cubic root of a value.

 - template<typename t >
t min (const t &a, const t &b, const t &c)

Return the minimum between three values.

 - template<typename t >
t min (const t &a, const t &b, const t &c, const t &d)

Return the minimum between four values.

 - template<typename t >
t minabs (const t &a, const t &b)

Return the minabs between two values.

 - template<typename t >
t max (const t &a, const t &b, const t &c)

Return the maximum between three values.

 - template<typename t >
t max (const t &a, const t &b, const t &c, const t &d)

Return the maximum between four values.

 - template<typename t >
t maxabs (const t &a, const t &b)

Return the maxabs between two values.

 - template<typename T >
T sigmoid (const T &x)

Return the sigmoid of a value.

 - template<typename T >
T logit (const T &x)

Return the logit of a value.

 - template<typename T >
T sign (const T &x)

Return the sign of a value.

 - template<typename T >
T abscut (const T &val, const T &val_min, const T &val_max, const T &offset)

Cut (i.e. clamp) absolute value in specified interval.

 - template<typename T >
unsigned long long nearest_pow2 (const T &x)

Return the nearest power of 2 higher than given value.

 - template<typename T >
T mod (const T &x, const T &m)

Return the modulo of a value.

 - template<typename T >
T minmod (const T &a, const T &b)

- Return the min-mod of two values.*

 - `template<typename T >`
`T round (const T &x, const double y, const int rounding_type=0)`

Return rounded value.
- `template<typename T >`
`T hypot (const T x, const T y)`

Return $\sqrt{x^2 + y^2}$.
- `template<typename T >`
`T hypot (const T x, const T y, const T z)`

Return $\sqrt{x^2 + y^2 + z^2}$.
- `double factorial (const int n)`

Return the factorial of n .
- `double permutations (const int k, const int n, const bool with_order)`

Return the number of permutations of k objects in a set of n objects.
- `double fibonacci (const int n)`

Calculate fibonacci number.
- `double wave (const double x, const unsigned int type=4)`

Wave function.
- `template<typename T >`
`T gcd (T a, T b)`

Calculate greatest common divisor of two integers.
- `template<typename T >`
`T lcm (T a, T b)`

Calculate least common multiple of two integers.
- `char lowercase (const char x)`

Convert character to lower case.
- `void lowercase (char *const str)`

Convert C-string to lower case.
- `char uppercase (const char x)`

Convert character to upper case.
- `void uppercase (char *const str)`

Convert C-string to upper case.
- `bool is_blank (const char c)`

Return `true` if input character is blank (space, tab, or non-printable character).
- `bool is_varname (const char *const str, const unsigned int length=~0U)`

Return `true` if argument `str` can be considered as a regular variable name.
- `double atof (const char *const str)`

Read value in a C-string.
- `int strncasecmp (const char *const str1, const char *const str2, const int length)`

Compare the first `length` characters of two C-strings, ignoring the case.
- `int strcasecmp (const char *const str1, const char *const str2)`

Compare two C-strings, ignoring the case.
- `char * strellipse (char *const str, const unsigned int l=64, const bool is_ending=true)`

Ellipsize a string.
- `char * strellipse (const char *const str, char *const res, const unsigned int l=64, const bool is_ending=true)`

Ellipsize a string.
- `bool strpare (char *const str, const char delimiter, const bool is_symmetric, const bool is_iterative)`

Remove delimiters on the start and/or end of a C-string.
- `bool strpare (char *const str, const bool is_symmetric, const bool is_iterative)`

Remove white spaces on the start and/or end of a C-string.
- `void strwindows_reserved (char *const str, const char c='_')`

- Replace reserved characters (for Windows filename) by another character.*

 - `void strunescape (char *const str)`
- Replace escape sequences in C-strings by character values.*

 - `const char * basename (const char *const s, const char separator='/')`

Return the basename of a filename.
- `std::FILE * fopen (const char *const path, const char *const mode)`

Open a file.
- `int fclose (std::FILE *file)`

Close a file.
- `int fseek (FILE *stream, long offset, int origin)`

Version of 'fseek()' that supports >=64bits offsets everywhere (for Windows).
- `long ftell (FILE *stream)`

Version of 'ftell()' that supports >=64bits offsets everywhere (for Windows).
- `bool is_directory (const char *const path)`

Check if a path is a directory.
- `bool is_file (const char *const path)`

Check if a path is a file.
- `bool path_exists (const char *const path)`

Check if a path exists.
- `long long fsize (std::FILE *const file)`

Get file size.
- `long long fsize (const char *const filename)`

Get file size from filename.
- `template<typename T > int fddate (const char *const path, T *attr, const unsigned int nb_attr)`

Get last write time of a given file or directory (multiple-attributes version).
- `int fddate (const char *const path, unsigned int attr)`

Get last write time of a given file or directory (single-attribute version).
- `template<typename T > int date (T *attr, const unsigned int nb_attr)`

Get current local time (multiple-attributes version).
- `int date (unsigned int attr)`

Get current local time (single-attribute version).
- `long long epoch (const int year, const int month=1, const int day=1, const int hour=0, const int minute=0, const int second=0)`

Convert date to epoch (UTC time).
- `const char * curl_path (const char *const user_path, const bool reinit_path)`

Search path of an executable.
- `const char * dcraw_path (const char *const user_path, const bool reinit_path)`

Get/set path to the dcraw binary.
- `const char * ffmpeg_path (const char *const user_path, const bool reinit_path)`

Get/set path to the FFmpeg's ffmpeg binary.
- `const char * graphicsmagick_path (const char *const user_path, const bool reinit_path)`

Get/set path to the GraphicsMagick's gm binary.
- `const char * gunzip_path (const char *const user_path, const bool reinit_path)`

Get/set path to the gunzip binary.
- `const char * gzip_path (const char *const user_path, const bool reinit_path)`

Get/set path to the gzip binary.
- `const char * imagemagick_path (const char *const user_path, const bool reinit_path)`

Get/set path to the ImageMagick's convert binary.
- `const char * medcon_path (const char *const user_path, const bool reinit_path)`

- Get/set path to the Medcon's medcon binary.*

 - `const char * temporary_path (const char *const user_path, const bool reinit_path)`

Get/set path to store temporary files.

 - `const char * wget_path (const char *const user_path, const bool reinit_path)`

Get/set path to the wget binary.

 - `const char * split_filename (const char *const filename, char *const body=0)`

Split filename into two C-strings body and extension.

 - `char * number_filename (const char *const filename, const int number, const unsigned int digits, char *const str)`

Generate a numbered version of a filename.

 - `template<typename T >`
`size_t fread (T *const ptr, const size_t nmemb, std::FILE *stream)`

Read data from file.

 - `template<typename T >`
`size_t fwrite (const T *ptr, const size_t nmemb, std::FILE *stream)`

Write data to file.

 - `void fempty (std::FILE *const file, const char *const filename)`

Create an empty file.

 - `const char * ftype (std::FILE *const file, const char *const filename)`

Try to guess format from an image file.

 - `char * load_network (const char *const url, char *const filename_local, const unsigned int timeout, const bool try_fallback, const char *const referer, const char *const user_agent)`

Load file from network as a local temporary file.

 - `const char * option (const char *const name, const int argc, const char *const *const argv, const char *const _default, const char *const usage, const bool reset_static)`

Return options specified on the command line.

 - `CImgList< char > files (const char *const path, const bool is_pattern=false, const unsigned int mode=2, const bool include_path=false)`

Return the list of files/directories in specified directory.

 - `template<typename t >`
`int dialog (const char *const title, const char *const msg, const char *const button1_label, const char *const button2_label, const char *const button3_label, const char *const button4_label, const char *const button5_label, const char *const button6_label, const CImg< t > &logo, const bool is_centered=false)`

Display a simple dialog box, and wait for the user's response.

Variables

- `const unsigned int keyESC = 1U`
Keycode for the ESC key (architecture-dependent)
- `const unsigned int keyF1 = 2U`
Keycode for the F1 key (architecture-dependent)
- `const unsigned int keyF2 = 3U`
Keycode for the F2 key (architecture-dependent)
- `const unsigned int keyF3 = 4U`
Keycode for the F3 key (architecture-dependent)
- `const unsigned int keyF4 = 5U`
Keycode for the F4 key (architecture-dependent)
- `const unsigned int keyF5 = 6U`
Keycode for the F5 key (architecture-dependent)
- `const unsigned int keyF6 = 7U`
Keycode for the F6 key (architecture-dependent)

- `const unsigned int keyF7 = 8U`
Keycode for the F7 key (architecture-dependent)
- `const unsigned int keyF8 = 9U`
Keycode for the F8 key (architecture-dependent)
- `const unsigned int keyF9 = 10U`
Keycode for the F9 key (architecture-dependent)
- `const unsigned int keyF10 = 11U`
Keycode for the F10 key (architecture-dependent)
- `const unsigned int keyF11 = 12U`
Keycode for the F11 key (architecture-dependent)
- `const unsigned int keyF12 = 13U`
Keycode for the F12 key (architecture-dependent)
- `const unsigned int keyPAUSE = 14U`
Keycode for the PAUSE key (architecture-dependent)
- `const unsigned int key1 = 15U`
Keycode for the 1 key (architecture-dependent)
- `const unsigned int key2 = 16U`
Keycode for the 2 key (architecture-dependent)
- `const unsigned int key3 = 17U`
Keycode for the 3 key (architecture-dependent)
- `const unsigned int key4 = 18U`
Keycode for the 4 key (architecture-dependent)
- `const unsigned int key5 = 19U`
Keycode for the 5 key (architecture-dependent)
- `const unsigned int key6 = 20U`
Keycode for the 6 key (architecture-dependent)
- `const unsigned int key7 = 21U`
Keycode for the 7 key (architecture-dependent)
- `const unsigned int key8 = 22U`
Keycode for the 8 key (architecture-dependent)
- `const unsigned int key9 = 23U`
Keycode for the 9 key (architecture-dependent)
- `const unsigned int key0 = 24U`
Keycode for the 0 key (architecture-dependent)
- `const unsigned int keyBACKSPACE = 25U`
Keycode for the BACKSPACE key (architecture-dependent)
- `const unsigned int keyINSERT = 26U`
Keycode for the INSERT key (architecture-dependent)
- `const unsigned int keyHOME = 27U`
Keycode for the HOME key (architecture-dependent)
- `const unsigned int keyPAGEUP = 28U`
Keycode for the PAGEUP key (architecture-dependent)
- `const unsigned int keyTAB = 29U`
Keycode for the TAB key (architecture-dependent)
- `const unsigned int keyQ = 30U`
Keycode for the Q key (architecture-dependent)
- `const unsigned int keyW = 31U`
Keycode for the W key (architecture-dependent)
- `const unsigned int keyE = 32U`
Keycode for the E key (architecture-dependent)
- `const unsigned int keyR = 33U`

- Keycode for the *R* key (architecture-dependent)

 - `const unsigned int keyT = 34U`
- Keycode for the *T* key (architecture-dependent)

 - `const unsigned int keyY = 35U`
- Keycode for the *Y* key (architecture-dependent)

 - `const unsigned int keyU = 36U`
- Keycode for the *U* key (architecture-dependent)

 - `const unsigned int keyI = 37U`
- Keycode for the *I* key (architecture-dependent)

 - `const unsigned int keyO = 38U`
- Keycode for the *O* key (architecture-dependent)

 - `const unsigned int keyP = 39U`
- Keycode for the *P* key (architecture-dependent)

 - `const unsigned int keyDELETE = 40U`
- Keycode for the *DELETE* key (architecture-dependent)

 - `const unsigned int keyEND = 41U`
- Keycode for the *END* key (architecture-dependent)

 - `const unsigned int keyPAGEDOWN = 42U`
- Keycode for the *PAGEDOWN* key (architecture-dependent)

 - `const unsigned int keyCAPSLOCK = 43U`
- Keycode for the *CAPSLOCK* key (architecture-dependent)

 - `const unsigned int keyA = 44U`
- Keycode for the *A* key (architecture-dependent)

 - `const unsigned int keyS = 45U`
- Keycode for the *S* key (architecture-dependent)

 - `const unsigned int keyD = 46U`
- Keycode for the *D* key (architecture-dependent)

 - `const unsigned int keyF = 47U`
- Keycode for the *F* key (architecture-dependent)

 - `const unsigned int keyG = 48U`
- Keycode for the *G* key (architecture-dependent)

 - `const unsigned int keyH = 49U`
- Keycode for the *H* key (architecture-dependent)

 - `const unsigned int keyJ = 50U`
- Keycode for the *J* key (architecture-dependent)

 - `const unsigned int keyK = 51U`
- Keycode for the *K* key (architecture-dependent)

 - `const unsigned int keyL = 52U`
- Keycode for the *L* key (architecture-dependent)

 - `const unsigned int keyRETURN = 53U`
- Keycode for the *RETURN* key (architecture-dependent)

 - `const unsigned int keySHIFTLEFT = 54U`
- Keycode for the *SHIFTLEFT* key (architecture-dependent)

 - `const unsigned int keyZ = 55U`
- Keycode for the *Z* key (architecture-dependent)

 - `const unsigned int keyX = 56U`
- Keycode for the *X* key (architecture-dependent)

 - `const unsigned int keyC = 57U`
- Keycode for the *C* key (architecture-dependent)

 - `const unsigned int keyV = 58U`
- Keycode for the *V* key (architecture-dependent)

- `const unsigned int keyB = 59U`
Keycode for the `B` key (architecture-dependent)
- `const unsigned int keyN = 60U`
Keycode for the `N` key (architecture-dependent)
- `const unsigned int keyM = 61U`
Keycode for the `M` key (architecture-dependent)
- `const unsigned int keySHIFTRIGHT = 62U`
Keycode for the `SHIFTRIGHT` key (architecture-dependent)
- `const unsigned int keyARROWUP = 63U`
Keycode for the `ARROWUP` key (architecture-dependent)
- `const unsigned int keyCTRLLEFT = 64U`
Keycode for the `CTRLLEFT` key (architecture-dependent)
- `const unsigned int keyAPPLEFT = 65U`
Keycode for the `APPLEFT` key (architecture-dependent)
- `const unsigned int keyALT = 66U`
Keycode for the `ALT` key (architecture-dependent)
- `const unsigned int keySPACE = 67U`
Keycode for the `SPACE` key (architecture-dependent)
- `const unsigned int keyALTGR = 68U`
Keycode for the `ALTGR` key (architecture-dependent)
- `const unsigned int keyAPPRIGHT = 69U`
Keycode for the `APPRIGHT` key (architecture-dependent)
- `const unsigned int keyMENU = 70U`
Keycode for the `MENU` key (architecture-dependent)
- `const unsigned int keyCTRLRIGHT = 71U`
Keycode for the `CTRLRIGHT` key (architecture-dependent)
- `const unsigned int keyARROWLEFT = 72U`
Keycode for the `ARROWLEFT` key (architecture-dependent)
- `const unsigned int keyARROWDOWN = 73U`
Keycode for the `ARROWDOWN` key (architecture-dependent)
- `const unsigned int keyARROWRIGHT = 74U`
Keycode for the `ARROWRIGHT` key (architecture-dependent)
- `const unsigned int keyPAD0 = 75U`
Keycode for the `PAD0` key (architecture-dependent)
- `const unsigned int keyPAD1 = 76U`
Keycode for the `PAD1` key (architecture-dependent)
- `const unsigned int keyPAD2 = 77U`
Keycode for the `PAD2` key (architecture-dependent)
- `const unsigned int keyPAD3 = 78U`
Keycode for the `PAD3` key (architecture-dependent)
- `const unsigned int keyPAD4 = 79U`
Keycode for the `PAD4` key (architecture-dependent)
- `const unsigned int keyPAD5 = 80U`
Keycode for the `PAD5` key (architecture-dependent)
- `const unsigned int keyPAD6 = 81U`
Keycode for the `PAD6` key (architecture-dependent)
- `const unsigned int keyPAD7 = 82U`
Keycode for the `PAD7` key (architecture-dependent)
- `const unsigned int keyPAD8 = 83U`
Keycode for the `PAD8` key (architecture-dependent)
- `const unsigned int keyPAD9 = 84U`

- Keycode for the `PAD9` key (architecture-dependent)*
 - `const unsigned int keyPADADD = 85U`
- Keycode for the `PADADD` key (architecture-dependent)*
 - `const unsigned int keyPADSUB = 86U`
- Keycode for the `PADSUB` key (architecture-dependent)*
 - `const unsigned int keyPADMUL = 87U`
- Keycode for the `PADMUL` key (architecture-dependent)*
 - `const unsigned int keyPADDIV = 88U`
- Keycode for the `PADDDIV` key (architecture-dependent)*
 - `const unsigned int keyENTER = 89U`
- Keycode for the `ENTER` key (architecture-dependent)*
 - `const double PI = 3.14159265358979323846`
- Value of the mathematical constant `PI`.*

7.2.1 Detailed Description

Contains low-level functions and variables of the `CImg` Library.

Most of the functions and variables within this namespace are used by the `CImg` library for low-level operations. You may use them to access specific const values or environment variables internally used by `CImg`.

Warning

Never write using namespace `cimg_library::cimg;` in your source code. A lot of functions in the `cimg:: namespace` have the same names as standard C functions that may be defined in the global namespace `::`.

7.2.2 Function Documentation

7.2.2.1 output()

```
std::FILE * output (
    std::FILE * file )
```

Get/set default output stream for the `CImg` library messages.

Parameters

<i>file</i>	Desired output stream. Set to 0 to get the currently used output stream only.
-------------	---

Returns

Currently used output stream.

7.2.2.2 info()

```
void info ( )
```

Print information about `CImg` environment variables.

Note

Output is done on the default output stream.

7.2.2.3 exception_mode() [1/2]

```
unsigned int & exception_mode (
    const unsigned int mode )
```

Set current CImg exception mode.

The way error messages are handled by CImg can be changed dynamically, using this function.

Parameters

<i>mode</i>	Desired exception mode. Possible values are: • 0: Hide library messages (quiet mode). • 1: Print library messages on the console. • 2: Display library messages on a dialog window. • 3: Do as 1 + add extra debug warnings (slow down the code!). • 4: Do as 2 + add extra debug warnings (slow down the code!).
-------------	---

7.2.2.4 exception_mode() [2/2]

```
unsigned int & exception_mode ( )
```

Return current CImg exception mode.

Note

By default, return the value of configuration macro `cimg_verbosity`

7.2.2.5 openmp_mode()

```
unsigned int openmp_mode (
    const unsigned int mode )
```

Set current CImg openmp mode.

The way openmp-based functions are handled by CImg can be changed dynamically, using this function.

Parameters

<i>mode</i>	Desired openmp mode. Possible values are: • 0: Never parallelize. • 1: Always parallelize. • 2: Adaptive parallelization mode (default behavior).
-------------	---

7.2.2.6 eval()

```
double eval (
    const char *const expression,
    const double x,
    const double y,
    const double z,
    const double c )
```

Evaluate math expression.

Parameters

<i>expression</i>	C-string describing the formula to evaluate.
<i>x</i>	Value of the predefined variable <i>x</i> .
<i>y</i>	Value of the predefined variable <i>y</i> .
<i>z</i>	Value of the predefined variable <i>z</i> .
<i>c</i>	Value of the predefined variable <i>c</i> .

Returns

Result of the formula evaluation.

Note

Set *expression* to 0 to keep evaluating the last specified expression.

Example

```
const double
res1 = cimg::eval("cos(x)^2 + sin(y)^2",2,2), // will return '1'
res2 = cimg::eval(0,1,1); // will return '1' too
```

7.2.2.7 create_directory()

```
void create_directory (
    const char *const dirname,
    const bool force_overwrite )
```

Create a directory.

Parameters

<i>dirname</i>	The path of the directory to create.
<i>force_overwrite</i>	Force overwrite of the directory when necessary.

7.2.2.8 warn()

```
void warn (
    const char *const format,
    ... )
```

Display a warning message on the default output stream.

Parameters

<i>format</i>	C-string containing the format of the message, as with <code>std::printf()</code> .
---------------	---

Note

If configuration macro `cimg_strict_warnings` is set, this function throws a `CImgWarning` Exception instead.

Warning

As the first argument is a format string, it is highly recommended to write

```
cimg::warn("%s", warning_message);
```

instead of

```
cimg::warn(warning_message);
```

if `warning_message` can be arbitrary, to prevent invalid memory access.

7.2.2.9 system()

```
int system (
    const char *const command,
    const char *const module_name = 0,
    const bool is_verbose = false )
```

Parameters

<i>command</i>	C-string containing the command line to execute.
<i>module_name</i>	Module name.
<i>is_verbose</i>	Indicates whether the command output should be silent or verbose.

Returns

Status value of the executed command, whose meaning is OS-dependent.

Note

This function is similar to `std::system()` but it does not open an extra console windows on Windows-based systems.

7.2.2.10 endianness()

```
bool endianness ( )
```

Return the endianness of the current architecture.

Returns

`false` for *Little Endian* or `true` for *Big Endian*.

7.2.2.11 invert_endianness() [1/2]

```
template<typename T >
void invert_endianness (
    T *const buffer,
    const unsigned long size )
```

Reverse endianness of all elements in a memory buffer.

Parameters

in, out	<i>buffer</i>	Memory buffer whose endianness must be reversed.
	<i>size</i>	Number of buffer elements to reverse.

7.2.2.12 invert_endianness() [2/2]

```
template<typename T >
T & invert_endianness (
    T & a )
```

Reverse endianness of a single variable.

Parameters

in, out	<i>a</i>	Variable to reverse.
---------	----------	----------------------

Returns

Reference to reversed variable.

7.2.2.13 time()

```
unsigned long long time ( )
```

Return the value of a system timer, with a millisecond precision.

Note: The timer does not necessarily start from 0.

7.2.2.14 tic()

```
unsigned long long tic ( )
```

Start tic/toc timer for time measurement between code instructions.

Returns

Current value of the timer (same value as `time()`).

7.2.2.15 toc()

```
unsigned long long toc ( )
```

End tic/toc timer and displays elapsed time from last call to `tic()`.

Returns

Time elapsed (in ms) since last call to `tic()`.

7.2.2.16 sleep()

```
void sleep (
    const unsigned int milliseconds )
```

Sleep for a given numbers of milliseconds.

Parameters

<i>milliseconds</i>	Number of milliseconds to wait for.
---------------------	-------------------------------------

Note

This function frees the CPU resources during the sleeping time. It can be used to pace your program properly, without wasting CPU time.

7.2.2.17 wait()

```
unsigned int wait (
    const unsigned int milliseconds )
```

Wait for a given number of milliseconds since the last call to `wait()`.

Parameters

<i>milliseconds</i>	Number of milliseconds to wait for.
---------------------	-------------------------------------

Returns

Number of milliseconds elapsed since the last call to wait().

Note

Same as [sleep\(\)](#) with a waiting time computed with regard to the last call of wait(). It may be used to temporize your program properly, without wasting CPU time.

7.2.2.18 mod()

```
template<typename T >
T mod (
    const T & x,
    const T & m )
```

Return the modulo of a value.

Parameters

<i>x</i>	Input value.
<i>m</i>	Modulo value.

Note

This modulo function accepts negative and floating-points modulo numbers, as well as variables of any type.

7.2.2.19 minmod()

```
template<typename T >
T minmod (
    const T & a,
    const T & b )
```

Return the min-mod of two values.

Note

minmod(a,b) is defined to be:

- *minmod(a,b)* = *min(a,b)*, if a and b have the same sign.
- *minmod(a,b)* = 0, if a and b have different signs.

7.2.2.20 round()

```
template<typename T >
T round (
    const T & x,
    const double y,
    const int rounding_type = 0 )
```

Return rounded value.

Parameters

<i>x</i>	Value to be rounded.
<i>y</i>	Rounding precision.
<i>rounding_type</i>	Type of rounding operation (0 = nearest, -1 = backward, 1 = forward).

Returns

Rounded value, having the same type as input value *x*.

7.2.2.21 wave()

```
double wave (
    const double x,
    const unsigned int type = 4 )
```

Wave function.

Parameters

<i>x</i>	Value to evaluate.
<i>type</i>	Wave type. Can be { 0:Square 1:Triangular 2:Ascending sawtooth 3: Descending sawtooth 4:Sinusoidal }.

Note

A wave function has a period of 1, and has value in [-1,1].

Returns

Function value.

7.2.2.22 atof()

```
double atof (
    const char *const str )
```

Read value in a C-string.

Parameters

<i>str</i>	C-string containing the float value to read.
------------	--

Returns

Read value.

Note

Same as `std::atof()` extended to manage the retrieval of fractions from C-strings, as in "1/2".

7.2.2.23 strncasecmp()

```
int strncasecmp (  
    const char *const str1,  
    const char *const str2,  
    const int length )
```

Compare the first `length` characters of two C-strings, ignoring the case.

Parameters

<i>str1</i>	C-string.
<i>str2</i>	C-string.
<i>length</i>	Number of characters to compare.

Returns

0 if the two strings are equal, something else otherwise.

Note

This function has to be defined since it is not provided by all C++-compilers (not ANSI).

7.2.2.24 strcasecmp()

```
int strcasecmp (  
    const char *const str1,  
    const char *const str2 )
```

Compare two C-strings, ignoring the case.

Parameters

<i>str1</i>	C-string.
<i>str2</i>	C-string.

Returns

0 if the two strings are equal, something else otherwise.

Note

This function has to be defined since it is not provided by all C++-compilers (not ANSI).

7.2.2.25 strellipsize() [1/2]

```
char * strellipsize (
    char *const str,
    const unsigned int l = 64,
    const bool is_ending = true )
```

Ellipsize a string.

Parameters

<i>str</i>	C-string.
<i>l</i>	Max number of printed characters.
<i>is_ending</i>	Indicates whether the dots are placed at the end or at the center of the ellipsized string.

7.2.2.26 strellipsize() [2/2]

```
char * strellipsize (
    const char *const str,
    char *const res,
    const unsigned int l = 64,
    const bool is_ending = true )
```

Ellipsize a string.

Parameters

<i>str</i>	C-string.
<i>res</i>	output C-string.
<i>l</i>	Max number of printed characters. String 'res' must be a size of at least 'l+1'.
<i>is_ending</i>	Indicates whether the dots are placed at the end or at the center of the ellipsized string.

7.2.2.27 strpare()

```
bool strpare (
    char *const str,
    const char delimiter,
    const bool is_symmetric,
    const bool is_iterative )
```

Remove delimiters on the start and/or end of a C-string.

Parameters

<i>in, out</i>	<i>str</i>	C-string to work with (modified at output).
	<i>delimiter</i>	Delimiter character code to remove.
	<i>is_symmetric</i>	Indicates whether the removal is done only if delimiters are symmetric (both at the beginning and the end of <i>s</i>).
	<i>is_iterative</i>	Indicates whether the removal is done if several iterations are possible.

Returns

`true` if delimiters have been removed, `false` otherwise.

7.2.2.28 strwindows_reserved()

```
void strwindows_reserved (
    char *const str,
    const char c = '_' )
```

Replace reserved characters (for Windows filename) by another character.

Parameters

<i>in, out</i>	<i>str</i>	C-string to work with (modified at output).
<i>in</i>	<i>c</i>	Replacement character.

7.2.2.29 strunescape()

```
void strunescape (
    char *const str )
```

Replace escape sequences in C-strings by character values.

Parameters

<i>in, out</i>	<i>str</i>	C-string to work with (modified at output).
----------------	------------	---

7.2.2.30 fopen()

```
std::FILE * fopen (
    const char *const path,
    const char *const mode )
```

Open a file.

Parameters

<i>path</i>	Path of the filename to open.
<i>mode</i>	C-string describing the opening mode.

Returns

Opened file.

Note

Same as `std::fopen()` but throw a `CImgIOException` when the specified file cannot be opened, instead of returning 0.

7.2.2.31 fclose()

```
int fclose (
    std::FILE * file )
```

Close a file.

Parameters

<i>file</i>	File to close.
-------------	----------------

Returns

0 if file has been closed properly, something else otherwise.

Note

Same as `std::fclose()` but display a warning message if the file has not been closed properly.

7.2.2.32 is_directory()

```
bool is_directory (
    const char *const path )
```

Check if a path is a directory.

Parameters

<i>path</i>	Specified path to test.
-------------	-------------------------

7.2.2.33 is_file()

```
bool is_file (
    const char *const path )
```

Check if a path is a file.

Parameters

<i>path</i>	Specified path to test.
-------------	-------------------------

7.2.2.34 path_exists()

```
bool path_exists (
    const char *const path )
```

Check if a path exists.

Parameters

<i>path</i>	Specified path to test.
-------------	-------------------------

7.2.2.35 fsize() [1/2]

```
long long fsize (
    std::FILE *const file )
```

Get file size.

Parameters

<i>file</i>	Specified file to get size from.
-------------	----------------------------------

Returns

File size or '-1' if file does not exist.

7.2.2.36 fsize() [2/2]

```
long long fsize (
    const char *const filename )
```

Get file size from filename.

Parameters

<i>filename</i>	Specified filename to get size from.
-----------------	--------------------------------------

Returns

File size or '-1' if file does not exist.

7.2.2.37 fdate() [1/2]

```
template<typename T >
int fdate (
    const char *const path,
    T * attr,
    const unsigned int nb_attr )
```

Get last write time of a given file or directory (multiple-attributes version).

Parameters

	<i>path</i>	Specified path to get attributes from.
<i>in, out</i>	<i>attr</i>	Type of requested time attributes. Can be { 0=year 1=month 2=day 3=day of week 4=hour 5=minute 6=second } Replaced by read attributes after return (or -1 if an error occurred).
	<i>nb_attr</i>	Number of attributes to read/write.

Returns

Latest read attribute.

7.2.2.38 fdate() [2/2]

```
int fdate (
    const char *const path,
    unsigned int attr )
```

Get last write time of a given file or directory (single-attribute version).

Parameters

<i>path</i>	Specified path to get attributes from.
<i>attr</i>	Type of requested time attributes. Can be { 0=year 1=month 2=day 3=day of week 4=hour 5=minute 6=second }

Returns

Specified attribute or -1 if an error occurred.

7.2.2.39 date() [1/2]

```
template<typename T >
int date (
    T * attr,
    const unsigned int nb_attr )
```

Get current local time (multiple-attributes version).

Parameters

<code>in, out</code>	<code>attr</code>	Type of requested time attributes. Can be { 0=year 1=month 2=day 3=day of week 4=hour 5=minute 6=second 7=millisecond } Replaced by read attributes after return (or -1 if an error occurred).
	<code>nb_attr</code>	Number of attributes to read/write.

Returns

Latest read attribute.

7.2.2.40 date() [2/2]

```
int date (
    unsigned int attr )
```

Get current local time (single-attribute version).

Parameters

<code>attr</code>	Type of requested time attribute. Can be { 0=year 1=month 2=day 3=day of week 4=hour 5=minute 6=second 7=millisecond }
-------------------	--

Returns

Specified attribute or -1 if an error occurred.

7.2.2.41 curl_path()

```
const char * curl_path (
    const char *const user_path,
    const bool reinit_path )
```

Search path of an executable.

Get the file or directory attributes with support for UTF-8 paths (Windows only). Get/set path to the *Program Files/* directory (Windows only).

Parameters

<code>user_path</code>	Specified path, or 0 to get the path currently used.
<code>reinit_path</code>	Force path to be recalculated (may take some time).

Returns

Path containing the program files. Get/set path to the `curl` binary.

Parameters

<i>user_path</i>	Specified path, or 0 to get the path currently used.
<i>reinit_path</i>	Force path to be recalculated (may take some time).

Returns

Path containing the `curl` binary.

7.2.2.42 ddraw_path()

```
const char * ddraw_path (
    const char *const user_path,
    const bool reinit_path )
```

Get/set path to the `ddraw` binary.

Parameters

<i>user_path</i>	Specified path, or 0 to get the path currently used.
<i>reinit_path</i>	Force path to be recalculated (may take some time).

Returns

Path containing the `ddraw` binary.

7.2.2.43 ffmpeg_path()

```
const char * ffmpeg_path (
    const char *const user_path,
    const bool reinit_path )
```

Get/set path to the FFMPEG's `ffmpeg` binary.

Parameters

<i>user_path</i>	Specified path, or 0 to get the path currently used.
<i>reinit_path</i>	Force path to be recalculated (may take some time).

Returns

Path containing the `ffmpeg` binary.

7.2.2.44 graphicsmagick_path()

```
const char * graphicsmagick_path (
    const char *const user_path,
    const bool reinit_path )
```


Get/set path to the GraphicsMagick's `gm` binary.

Parameters

<i>user_path</i>	Specified path, or 0 to get the path currently used.
<i>reinit_path</i>	Force path to be recalculated (may take some time).

Returns

Path containing the `gm` binary.

7.2.2.45 gunzip_path()

```
const char * gunzip_path (  
    const char *const user_path,  
    const bool reinit_path )
```

Get/set path to the `gunzip` binary.

Parameters

<i>user_path</i>	Specified path, or 0 to get the path currently used.
<i>reinit_path</i>	Force path to be recalculated (may take some time).

Returns

Path containing the `gunzip` binary.

7.2.2.46 gzip_path()

```
const char * gzip_path (  
    const char *const user_path,  
    const bool reinit_path )
```

Get/set path to the `gzip` binary.

Parameters

<i>user_path</i>	Specified path, or 0 to get the path currently used.
<i>reinit_path</i>	Force path to be recalculated (may take some time).

Returns

Path containing the `gzip` binary.

7.2.2.47 imagemagick_path()

```
const char * imagemagick_path (
```

```
const char *const user_path,  
const bool reinit_path )
```

Get/set path to the ImageMagick's `convert` binary.

Parameters

<i>user_path</i>	Specified path, or 0 to get the path currently used.
<i>reinit_path</i>	Force path to be recalculated (may take some time).

Returns

Path containing the `convert` binary.

7.2.2.48 `medcon_path()`

```
const char * medcon_path (  
    const char *const user_path,  
    const bool reinit_path )
```

Get/set path to the Medcon's `medcon` binary.

Parameters

<i>user_path</i>	Specified path, or 0 to get the path currently used.
<i>reinit_path</i>	Force path to be recalculated (may take some time).

Returns

Path containing the `medcon` binary.

7.2.2.49 `temporary_path()`

```
const char * temporary_path (  
    const char *const user_path,  
    const bool reinit_path )
```

Get/set path to store temporary files.

Parameters

<i>user_path</i>	Specified path, or 0 to get the path currently used.
<i>reinit_path</i>	Force path to be recalculated (may take some time).

Returns

Path where temporary files can be saved.

7.2.2.50 wget_path()

```
const char * wget_path (
    const char *const user_path,
    const bool reinit_path )
```

Get/set path to the wget binary.

Parameters

<i>user_path</i>	Specified path, or 0 to get the path currently used.
<i>reinit_path</i>	Force path to be recalculated (may take some time).

Returns

Path containing the wget binary.

7.2.2.51 split_filename()

```
const char * split_filename (
    const char *const filename,
    char *const body = 0 )
```

Split filename into two C-strings *body* and *extension*.

filename and body must not overlap!

7.2.2.52 fread()

```
template<typename T >
size_t fread (
    T *const ptr,
    const size_t nmemb,
    std::FILE * stream )
```

Read data from file.

Parameters

out	<i>ptr</i>	Pointer to memory buffer that will contain the binary data read from file.
	<i>nmemb</i>	Number of elements to read.
	<i>stream</i>	File to read data from.

Returns

Number of read elements.

Note

Same as `std::fread()` but may display warning message if all elements could not be read.

7.2.2.53 fwrite()

```
template<typename T >
size_t fwrite (
    const T * ptr,
    const size_t nmemb,
    std::FILE * stream )
```

Write data to file.

Parameters

	<i>ptr</i>	Pointer to memory buffer containing the binary data to write on file.
	<i>nmemb</i>	Number of elements to write.
out	<i>stream</i>	File to write data on.

Returns

Number of written elements.

Note

Similar to `std::fwrite` but may display warning messages if all elements could not be written.

7.2.2.54 fempty()

```
void fempty (
    std::FILE *const file,
    const char *const filename )
```

Create an empty file.

Parameters

<i>file</i>	Input file (can be 0 if filename is set).
<i>filename</i>	Filename, as a C-string (can be 0 if file is set).

7.2.2.55 ftype()

```
const char * ftype (
    std::FILE *const file,
    const char *const filename )
```

Try to guess format from an image file.

Parameters

<i>file</i>	Input file (can be 0 if filename is set).
<i>filename</i>	Filename, as a C-string (can be 0 if file is set).

Returns

C-string containing the guessed file format, or 0 if nothing has been guessed.

7.2.2.56 load_network()

```
char * load_network (
    const char *const url,
    char *const filename_local,
    const unsigned int timeout,
    const bool try_fallback,
    const char *const referer,
    const char *const user_agent )
```

Load file from network as a local temporary file.

Parameters

	<i>url</i>	URL of the filename, as a C-string.
out	<i>filename_local</i>	C-string containing the path to a local copy of <i>filename</i> .
	<i>timeout</i>	Maximum time (in seconds) authorized for downloading the file from the URL.
	<i>try_fallback</i>	When using libcurl, it asks for using system calls as fallbacks in case of libcurl failure.
	<i>referer</i>	Referer used, as a C-string.
	<i>user_agent</i>	User agent used, as a C-string.

Returns

Value of *filename_local*.

Note

Use the `libcurl` library, or the external binaries `wget` or `curl` to perform the download.

7.2.2.57 files()

```
CImgList< char > files (
    const char *const path,
    const bool is_pattern = false,
    const unsigned int mode = 2,
    const bool include_path = false )
```

Return the list of files/directories in specified directory.

Parameters

<i>path</i>	Path to the directory. Set to 0 for current directory.
<i>is_pattern</i>	Indicates that a specified path has a matching pattern in it.
<i>mode</i>	Output type, can be primary { 0=files only 1=folders only 2=files + folders }.
<i>include_path</i>	Indicates that <i>path</i> must be included in resulting filenames.

Returns

A list of filenames.

7.2.2.58 dialog()

```
template<typename t >
int dialog (
    const char *const title,
    const char *const msg,
    const char *const button1_label,
    const char *const button2_label,
    const char *const button3_label,
    const char *const button4_label,
    const char *const button5_label,
    const char *const button6_label,
    const CImg< t > & logo,
    const bool is_centered = false )
```

Display a simple dialog box, and wait for the user's response.

Parameters

<i>title</i>	Title of the dialog window.
<i>msg</i>	Main message displayed inside the dialog window.
<i>button1_label</i>	Label of the 1st button.
<i>button2_label</i>	Label of the 2nd button (0 to hide button).
<i>button3_label</i>	Label of the 3rd button (0 to hide button).
<i>button4_label</i>	Label of the 4th button (0 to hide button).
<i>button5_label</i>	Label of the 5th button (0 to hide button).
<i>button6_label</i>	Label of the 6th button (0 to hide button).
<i>logo</i>	Image logo displayed at the left of the main message.
<i>is_centered</i>	Indicates whether the dialog window must be centered on the screen.

Returns

Index of clicked button (from 0 to 5), or -1 if the dialog window has been closed by the user.

Note

- Up to 6 buttons can be defined in the dialog window.
- The function returns when a user clicked one of the button or closed the dialog window.
- If a button text is set to 0, the corresponding button (and the following) will not appear in the dialog box. At least one button must be specified.

Chapter 8

Class Documentation

8.1 CImg< T > Struct Template Reference

Class representing an image (up to 4 dimensions wide), where each pixel is of type T.

```
#include <CImg.h>
```

Public Types

- `typedef T * iterator`
Simple iterator type to loop through each pixel value of an image instance.
- `typedef const T * const_iterator`
Simple const iterator type, to loop through each pixel value of a `const` image instance.
- `typedef T value_type`
Pixel value type.

Constructors / Destructor / Instance Management

- `static CImg< T > & empty ()`
Return a reference to an empty image.
- `static const CImg< T > & const_empty ()`
*Return a reference to an empty image [**const version**].*
- `~CImg ()`
Destroy image.
- `CImg ()`
Construct empty image.
- `CImg (const unsigned int size_x, const unsigned int size_y=1, const unsigned int size_z=1, const unsigned int size_c=1)`
Construct image with specified size.
- `CImg (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const T &value)`
Construct image with specified size and initialize pixel values.
- `CImg (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const int value0, const int value1,...)`
Construct image with specified size and initialize pixel values from a sequence of integers.

- `Clmg (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const double value0, const double value1,...)`
Construct image with specified size and initialize pixel values from a sequence of doubles.
- `Clmg (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const char *const values, const bool repeat_values)`
Construct image with specified size and initialize pixel values from a value string.
- `template<typename t >`
`Clmg (const t *const values, const unsigned int size_x, const unsigned int size_y=1, const unsigned int size_z=1, const unsigned int size_c=1, const bool is_shared=false)`
Construct image with specified size and initialize pixel values from a memory buffer.
- `Clmg (const T *const values, const unsigned int size_x, const unsigned int size_y=1, const unsigned int size_z=1, const unsigned int size_c=1, const bool is_shared=false)`
*Construct image with specified size and initialize pixel values from a memory buffer **[specialization]**.*
- `template<typename t >`
`Clmg (const t *const values, const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const char *const axes_order)`
Construct image from memory buffer with specified size and pixel ordering scheme.
- `Clmg (const char *const filename)`
Construct image from reading an image file.
- `template<typename t >`
`Clmg (const Clmg< t > &img)`
Construct image copy.
- `Clmg (const Clmg< T > &img)`
*Construct image copy **[specialization]**.*
- `template<typename t >`
`Clmg (const Clmg< t > &img, const bool is_shared)`
Advanced copy constructor.
- `Clmg (const Clmg< T > &img, const bool is_shared)`
*Advanced copy constructor **[specialization]**.*
- `template<typename t >`
`Clmg (const Clmg< t > &img, const char *const dimensions)`
Construct image with dimensions borrowed from another image.
- `template<typename t >`
`Clmg (const Clmg< t > &img, const char *const dimensions, const T &value)`
Construct image with dimensions borrowed from another image and initialize pixel values.
- `Clmg (const ClmgDisplay &disp)`
Construct image from a display window.
- `Clmg< T > & assign ()`
*Construct empty image **[in-place version]**.*
- `Clmg< T > & assign (const unsigned int size_x, const unsigned int size_y=1, const unsigned int size_z=1, const unsigned int size_c=1)`
*Construct image with specified size **[in-place version]**.*
- `Clmg< T > & assign (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const T &value)`
*Construct image with specified size and initialize pixel values **[in-place version]**.*
- `Clmg< T > & assign (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const int value0, const int value1,...)`
*Construct image with specified size and initialize pixel values from a sequence of integers **[in-place version]**.*
- `Clmg< T > & assign (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const double value0, const double value1,...)`
*Construct image with specified size and initialize pixel values from a sequence of doubles **[in-place version]**.*
- `Clmg< T > & assign (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const char *const values, const bool repeat_values)`

- Construct image with specified size and initialize pixel values from a value string [in-place version].*

 - `template<typename t >`
`Clmg< T > & assign (const t *const values, const unsigned int size_x, const unsigned int size_y=1, const unsigned int size_z=1, const unsigned int size_c=1)`

Construct image with specified size and initialize pixel values from a memory buffer [in-place version].

 - `Clmg< T > & assign (const T *const values, const unsigned int size_x, const unsigned int size_y=1, const unsigned int size_z=1, const unsigned int size_c=1)`

Construct image with specified size and initialize pixel values from a memory buffer [specialization].

 - `template<typename t >`
`Clmg< T > & assign (const t *const values, const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const char *const axes_order)`

Construct image from memory buffer with specified size and pixel ordering scheme.

 - `Clmg< T > & assign (const char *const filename)`

Construct image from reading an image file [in-place version].

 - `template<typename t >`
`Clmg< T > & assign (const Clmg< t > &img)`

Construct image copy [in-place version].

 - `template<typename t >`
`Clmg< T > & assign (const Clmg< t > &img, const bool is_shared)`

In-place version of the advanced copy constructor.

 - `Clmg< T > & assign (const ClmgDisplay &disp)`

Construct image from a display window [in-place version].

 - `Clmg< T > & clear ()`

Construct empty image [in-place version].

 - `template<typename t >`
`Clmg< t > & move_to (Clmg< t > &img)`

Transfer content of an image instance into another one.

 - `Clmg< T > & move_to (Clmg< T > &img)`

Transfer content of an image instance into another one [specialization].

 - `template<typename t >`
`ClmgList< t > & move_to (ClmgList< t > &list, const unsigned int pos=~0U)`

Transfer content of an image instance into a new image in an image list.

 - `Clmg< T > & swap (Clmg< T > &img)`

Swap fields of two image instances.

Instance Characteristics

- `static const char * pixel_type ()`
Return the type of image pixel values as a C string.
- `int width () const`
Return the number of image columns.
- `int height () const`
Return the number of image rows.
- `int depth () const`
Return the number of image slices.
- `int spectrum () const`
Return the number of image channels.
- `ulongT size () const`
Return the total number of pixel values.
- `T * data ()`
Return a pointer to the first pixel value.

- `const T * data () const`
Return a pointer to the first pixel value [const version].
- `T * data (const unsigned int x, const unsigned int y=0, const unsigned int z=0, const unsigned int c=0)`
Return a pointer to a located pixel value.
- `const T * data (const unsigned int x, const unsigned int y=0, const unsigned int z=0, const unsigned int c=0) const`
Return a pointer to a located pixel value [const version].
- `longT offset (const int x, const int y=0, const int z=0, const int c=0) const`
Return the offset to a located pixel value, with respect to the beginning of the pixel buffer.
- `iterator begin ()`
Return a `CImg<T>::iterator` pointing to the first pixel value.
- `const_iterator begin () const`
Return a `CImg<T>::iterator` pointing to the first value of the pixel buffer [const version].
- `iterator end ()`
Return a `CImg<T>::iterator` pointing next to the last pixel value.
- `const_iterator end () const`
Return a `CImg<T>::iterator` pointing next to the last pixel value [const version].
- `T & front ()`
Return a reference to the first pixel value.
- `const T & front () const`
Return a reference to the first pixel value [const version].
- `T & back ()`
Return a reference to the last pixel value.
- `const T & back () const`
Return a reference to the last pixel value [const version].
- `T & at (const int offset, const T &out_value)`
Access to a pixel value at a specified offset, using Dirichlet boundary conditions.
- `T at (const int offset, const T &out_value) const`
Access to a pixel value at a specified offset, using Dirichlet boundary conditions [const version].
- `T & at (const int offset)`
Access to a pixel value at a specified offset, using Neumann boundary conditions.
- `const T & at (const int offset) const`
Access to a pixel value at a specified offset, using Neumann boundary conditions [const version].
- `T & atX (const int x, const int y, const int z, const int c, const T &out_value)`
Access to a pixel value, using Dirichlet boundary conditions for the X-coordinate.
- `T atX (const int x, const int y, const int z, const int c, const T &out_value) const`
Access to a pixel value, using Dirichlet boundary conditions for the X-coordinate [const version].
- `T & atX (const int x, const int y=0, const int z=0, const int c=0)`
Access to a pixel value, using Neumann boundary conditions for the X-coordinate.
- `const T & atX (const int x, const int y=0, const int z=0, const int c=0) const`
Access to a pixel value, using Neumann boundary conditions for the X-coordinate [const version].
- `T & atXY (const int x, const int y, const int z, const int c, const T &out_value)`
Access to a pixel value, using Dirichlet boundary conditions for the X and Y-coordinates.
- `T atXY (const int x, const int y, const int z, const int c, const T &out_value) const`
Access to a pixel value, using Dirichlet boundary conditions for the X and Y coordinates [const version].
- `T & atXY (const int x, const int y, const int z=0, const int c=0)`
Access to a pixel value, using Neumann boundary conditions for the X and Y-coordinates.
- `const T & atXY (const int x, const int y, const int z=0, const int c=0) const`
Access to a pixel value, using Neumann boundary conditions for the X and Y-coordinates [const version].
- `T & atXYZ (const int x, const int y, const int z, const int c, const T &out_value)`
Access to a pixel value, using Dirichlet boundary conditions for the X,Y and Z-coordinates.

- **T atXYZ** (const int x, const int y, const int z, const int c, const T &out_value) const
Access to a pixel value, using Dirichlet boundary conditions for the X,Y and Z-coordinates **[const version]**.
- **T & atXYZ** (const int x, const int y, const int z, const int c=0)
Access to a pixel value, using Neumann boundary conditions for the X,Y and Z-coordinates.
- **const T & atXYZ** (const int x, const int y, const int z, const int c=0) const
Access to a pixel value, using Neumann boundary conditions for the X,Y and Z-coordinates **[const version]**.
- **T & atXYZC** (const int x, const int y, const int z, const int c, const T &out_value)
Access to a pixel value, using Dirichlet boundary conditions.
- **T atXYZC** (const int x, const int y, const int z, const int c, const T &out_value) const
Access to a pixel value, using Dirichlet boundary conditions **[const version]**.
- **T & atXYZC** (const int x, const int y, const int z, const int c)
Access to a pixel value, using Neumann boundary conditions.
- **const T & atXYZC** (const int x, const int y, const int z, const int c) const
Access to a pixel value, using Neumann boundary conditions **[const version]**.
- **Tfloat linear_atX** (const float fx, const int y, const int z, const int c, const T &out_value) const
Return pixel value, using linear interpolation and Dirichlet boundary conditions for the X-coordinate.
- **Tfloat linear_atX** (const float fx, const int y=0, const int z=0, const int c=0) const
Return pixel value, using linear interpolation and Neumann boundary conditions for the X-coordinate.
- **Tfloat linear_atX_p** (const float fx, const int y=0, const int z=0, const int c=0) const
Return pixel value, using linear interpolation and periodic boundary conditions for the X-coordinate.
- **Tfloat linear_atXY** (const float fx, const float fy, const int z, const int c, const T &out_value) const
Return pixel value, using linear interpolation and Dirichlet boundary conditions for the X and Y-coordinates.
- **Tfloat linear_atXY** (const float fx, const float fy, const int z=0, const int c=0) const
Return pixel value, using linear interpolation and Neumann boundary conditions for the X and Y-coordinates.
- **Tfloat linear_atXY_p** (const float fx, const float fy, const int z=0, const int c=0) const
Return pixel value, using linear interpolation and periodic boundary conditions for the X and Y-coordinates.
- **Tfloat linear_atXYZ** (const float fx, const float fy, const float fz, const int c, const T &out_value) const
Return pixel value, using linear interpolation and Dirichlet boundary conditions for the X,Y and Z-coordinates.
- **Tfloat linear_atXYZ** (const float fx, const float fy=0, const float fz=0, const int c=0) const
Return pixel value, using linear interpolation and Neumann boundary conditions for the X,Y and Z-coordinates.
- **Tfloat linear_atXYZ_p** (const float fx, const float fy=0, const float fz=0, const int c=0) const
Return pixel value, using linear interpolation and periodic boundary conditions for the X,Y and Z-coordinates.
- **Tfloat linear_atXYZC** (const float fx, const float fy, const float fz, const float fc, const T &out_value) const
Return pixel value, using linear interpolation and Dirichlet boundary conditions for all X,Y,Z,C-coordinates.
- **Tfloat linear_atXYZC** (const float fx, const float fy=0, const float fz=0, const float fc=0) const
Return pixel value, using linear interpolation and Neumann boundary conditions for all X,Y,Z and C-coordinates.
- **Tfloat linear_atXYZC_p** (const float fx, const float fy=0, const float fz=0, const float fc=0) const
Return pixel value, using linear interpolation and periodic boundary conditions for all X,Y,Z and C-coordinates.
- **Tfloat cubic_atX** (const float fx, const int y, const int z, const int c, const T &out_value) const
Return pixel value, using cubic interpolation and Dirichlet boundary conditions for the X-coordinate.
- **T cubic_atX_c** (const float fx, const int y, const int z, const int c, const T &out_value) const
Return clamped pixel value, using cubic interpolation and Dirichlet boundary conditions for the X-coordinate.
- **Tfloat cubic_atX** (const float fx, const int y=0, const int z=0, const int c=0) const
Return pixel value, using cubic interpolation and Neumann boundary conditions for the X-coordinate.
- **T cubic_atX_c** (const float fx, const int y, const int z, const int c) const
Return clamped pixel value, using cubic interpolation and Neumann boundary conditions for the X-coordinate.
- **Tfloat cubic_atX_p** (const float fx, const int y=0, const int z=0, const int c=0) const
Return pixel value, using cubic interpolation and periodic boundary conditions for the X-coordinate.
- **Tfloat cubic_atXY** (const float fx, const float fy, const int z, const int c, const T &out_value) const
Return pixel value, using cubic interpolation and Dirichlet boundary conditions for the X and Y-coordinates.
- **T cubic_atXY_c** (const float fx, const float fy, const int z, const int c, const T &out_value) const

- Return clamped pixel value, using cubic interpolation and Dirichlet boundary conditions for the X,Y-coordinates.*
- `Tfloat cubic_atXY (const float fx, const float fy, const int z=0, const int c=0) const`
- Return pixel value, using cubic interpolation and Neumann boundary conditions for the X and Y-coordinates.*
- `T cubic_atXY_c (const float fx, const float fy, const int z, const int c) const`
- Return clamped pixel value, using cubic interpolation and Neumann boundary conditions for the X,Y-coordinates.*
- `Tfloat cubic_atXY_p (const float fx, const float fy, const int z=0, const int c=0) const`
- Return pixel value, using cubic interpolation and periodic boundary conditions for the X and Y-coordinates.*
- `Tfloat cubic_atXYZ (const float fx, const float fy, const float fz, const int c, const T &out_value) const`
- Return pixel value, using cubic interpolation and Dirichlet boundary conditions for the X,Y and Z-coordinates.*
- `T cubic_atXYZ_c (const float fx, const float fy, const float fz, const int c, const T &out_value) const`
- Return clamped pixel value, using cubic interpolation and Dirichlet boundary conditions for the XYZ-coordinates.*
- `Tfloat cubic_atXYZ (const float fx, const float fy, const float fz, const int c=0) const`
- Return pixel value, using cubic interpolation and Neumann boundary conditions for the X,Y and Z-coordinates.*
- `T cubic_atXYZ_c (const float fx, const float fy, const float fz, const int c) const`
- Return clamped pixel value, using cubic interpolation and Neumann boundary conditions for the XYZ-coordinates.*
- `Tfloat cubic_atXYZ_p (const float fx, const float fy, const float fz, const int c=0) const`
- Return pixel value, using cubic interpolation and Neumann boundary conditions for the X,Y and Z-coordinates.*
- `Clmg< T > & set_linear_atX (const T &value, const float fx, const int y=0, const int z=0, const int c=0, const bool is_added=false)`
- Set pixel value, using linear interpolation for the X-coordinates.*
- `Clmg< T > & set_linear_atXY (const T &value, const float fx, const float fy=0, const int z=0, const int c=0, const bool is_added=false)`
- Set pixel value, using linear interpolation for the X and Y-coordinates.*
- `Clmg< T > & set_linear_atXYZ (const T &value, const float fx, const float fy=0, const float fz=0, const int c=0, const bool is_added=false)`
- Set pixel value, using linear interpolation for the X,Y and Z-coordinates.*
- `Clmg< charT > value_string (const char separator=',', const unsigned int max_size=0, const char *const format=0) const`
- Return a C-string containing a list of all values of the image instance.*

Vector / Matrix Operations

- `template<typename tf, typename t >`
`static Clmg< Tfloat > dijkstra (const tf &distance, const unsigned int nb_nodes, const unsigned int starting_node, const unsigned int ending_node, Clmg< t > &previous_node)`
Compute minimal path in a graph, using the Dijkstra algorithm.
- `template<typename tf, typename t >`
`static Clmg< T > dijkstra (const tf &distance, const unsigned int nb_nodes, const unsigned int starting_node, const unsigned int ending_node=~0U)`
Return minimal path in a graph, using the Dijkstra algorithm.
- `static Clmg< T > string (const char *const str, const bool is_last_zero=true, const bool is_shared=false)`
Return an image containing the character codes of specified string.
- `static Clmg< T > row_vector (const T &a0)`
Return a 1x1 image containing specified value.
- `static Clmg< T > row_vector (const T &a0, const T &a1)`
Return a 2x1 image containing specified values.
- `static Clmg< T > row_vector (const T &a0, const T &a1, const T &a2)`
Return a 3x1 image containing specified values.
- `static Clmg< T > row_vector (const T &a0, const T &a1, const T &a2, const T &a3)`
Return a 4x1 image containing specified values.
- `static Clmg< T > row_vector (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4)`

Return a 2x2 matrix containing specified coefficients.

- `static Clmg< T > matrix (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7, const T &a8)`

Return a 3x3 matrix containing specified coefficients.

- `static Clmg< T > matrix (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7, const T &a8, const T &a9, const T &a10, const T &a11, const T &a12, const T &a13, const T &a14, const T &a15)`

Return a 4x4 matrix containing specified coefficients.

- `static Clmg< T > matrix (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7, const T &a8, const T &a9, const T &a10, const T &a11, const T &a12, const T &a13, const T &a14, const T &a15, const T &a16, const T &a17, const T &a18, const T &a19, const T &a20, const T &a21, const T &a22, const T &a23, const T &a24)`

Return a 5x5 matrix containing specified coefficients.

- `static Clmg< T > tensor (const T &a0)`

Return a 1x1 symmetric matrix containing specified coefficients.

- `static Clmg< T > tensor (const T &a0, const T &a1, const T &a2)`

Return a 2x2 symmetric matrix tensor containing specified coefficients.

- `static Clmg< T > tensor (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5)`

Return a 3x3 symmetric matrix containing specified coefficients.

- `static Clmg< T > diagonal (const T &a0)`

Return a 1x1 diagonal matrix containing specified coefficients.

- `static Clmg< T > diagonal (const T &a0, const T &a1)`

Return a 2x2 diagonal matrix containing specified coefficients.

- `static Clmg< T > diagonal (const T &a0, const T &a1, const T &a2)`

Return a 3x3 diagonal matrix containing specified coefficients.

- `static Clmg< T > diagonal (const T &a0, const T &a1, const T &a2, const T &a3)`

Return a 4x4 diagonal matrix containing specified coefficients.

- `static Clmg< T > diagonal (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4)`

Return a 5x5 diagonal matrix containing specified coefficients.

- `static Clmg< T > identity_matrix (const unsigned int N)`

Return a NxN identity matrix.

- `static Clmg< T > sequence (const unsigned int N, const T &a0, const T &a1)`

Return a N-numbered sequence vector from a0 to a1.

- `static Clmg< T > rotation_matrix (const float x, const float y, const float z, const float w, const bool is_quaternion=false)`

Return a 3x3 rotation matrix from an { axis + angle } or a quaternion.

- `double magnitude (const float magnitude_type=2) const`

Compute norm of the image, viewed as a matrix.

- `double trace () const`

Compute the trace of the image, viewed as a matrix.

- `double det () const`

Compute the determinant of the image, viewed as a matrix.

- `template<typename t >
double dot (const Clmg< t > &img) const`

Compute the dot product between instance and argument, viewed as matrices.

- `Clmg< T > get_vector_at (const unsigned int x, const unsigned int y=0, const unsigned int z=0) const`

Get vector-valued pixel located at specified position.

- `Clmg< T > get_matrix_at (const unsigned int x=0, const unsigned int y=0, const unsigned int z=0) const`

Get (square) matrix-valued pixel located at specified position.

- `Clmg< T > get_tensor_at (const unsigned int x, const unsigned int y=0, const unsigned int z=0) const`

Get tensor-valued pixel located at specified position.

- `template<typename t >`
`Clmg< T > & set_vector_at (const Clmg< t > &vec, const unsigned int x, const unsigned int y=0, const unsigned int z=0)`
Set vector-valued pixel at specified position.
- `template<typename t >`
`Clmg< T > & set_matrix_at (const Clmg< t > &mat, const unsigned int x=0, const unsigned int y=0, const unsigned int z=0)`
Set (square) matrix-valued pixel at specified position.
- `template<typename t >`
`Clmg< T > & set_tensor_at (const Clmg< t > &ten, const unsigned int x=0, const unsigned int y=0, const unsigned int z=0)`
Set tensor-valued pixel at specified position.
- `Clmg< T > & diagonal ()`
Resize image to become a diagonal matrix.
- `Clmg< T > get_diagonal () const`
Resize image to become a diagonal matrix [new-instance version].
- `Clmg< T > & identity_matrix ()`
Replace the image by an identity matrix.
- `Clmg< T > get_identity_matrix () const`
Replace the image by an identity matrix [new-instance version].
- `Clmg< T > & sequence (const T &a0, const T &a1)`
Fill image with a linear sequence of values.
- `Clmg< T > get_sequence (const T &a0, const T &a1) const`
Fill image with a linear sequence of values [new-instance version].
- `Clmg< T > & transpose ()`
Transpose the image, viewed as a matrix.
- `Clmg< T > get_transpose () const`
Transpose the image, viewed as a matrix [new-instance version].
- `template<typename t >`
`Clmg< T > & cross (const Clmg< t > &img)`
Compute the cross product between two 1×3 images, viewed as 3D vectors.
- `template<typename t >`
`Clmg< typename cimg::superset< T, t >::type > get_cross (const Clmg< t > &img) const`
Compute the cross product between two 1×3 images, viewed as 3D vectors [new-instance version].
- `Clmg< T > & invert ()`
Invert the instance matrix.
- `Clmg< Tfloat > get_invert () const`
Invert the instance matrix.
- `template<typename t >`
`Clmg< T > & solve (const Clmg< t > &A)`
Solve a (possibly over- or under-determined) linear system using QR decomposition.
- `template<typename t >`
`Clmg< typename cimg::superset2< T, t, float >::type > get_solve (const Clmg< t > &A) const`
Solve a (possibly over- or under-determined) linear system using QR decomposition [new-instance version].
- `template<typename t >`
`Clmg< T > & solve_tridiagonal (const Clmg< t > &A)`
Solve a tridiagonal system of linear equations.
- `template<typename t >`
`Clmg< typename cimg::superset2< T, t, float >::type > get_solve_tridiagonal (const Clmg< t > &A) const`
Solve a tridiagonal system of linear equations [new-instance version].
- `template<typename t >`
`const Clmg< T > & eigen (Clmg< t > &val, Clmg< t > &vec) const`

- Compute eigenvalues and eigenvectors of the instance image, viewed as a matrix.*

 - `ClmgList< Tfloat > get_eigen () const`

Compute eigenvalues and eigenvectors of the instance image, viewed as a matrix.
- template<typename t >

 - `const Clmg< T > & symmetric_eigen (Clmg< t > &val, Clmg< t > &vec) const`

Compute eigenvalues and eigenvectors of the instance image, viewed as a symmetric matrix.
- `ClmgList< Tfloat > get_symmetric_eigen () const`

Compute eigenvalues and eigenvectors of the instance image, viewed as a symmetric matrix.
- template<typename t >

 - `Clmg< T > & sort (Clmg< t > &permutations, const bool is_increasing=true)`

Sort pixel values and get sorting permutations.
- template<typename t >

 - `Clmg< T > get_sort (Clmg< t > &permutations, const bool is_increasing=true) const`

Sort pixel values and get sorting permutations [new-instance version].
- `Clmg< T > & sort (const bool is_increasing=true, const char axis=0)`

Sort pixel values.
- `Clmg< T > get_sort (const bool is_increasing=true, const char axis=0) const`

Sort pixel values [new-instance version].
- template<typename t >

 - `const Clmg< T > & SVD (Clmg< t > &U, Clmg< t > &S, Clmg< t > &V, const bool sorting=true, const unsigned int max_iteration=40, const float lambda=0) const`

Compute the SVD of the instance image, viewed as a general matrix.
- `ClmgList< Tfloat > get_SVD (const bool sorting=true, const unsigned int max_iteration=40, const float lambda=0) const`

Compute the SVD of the instance image, viewed as a general matrix.
- template<typename t >

 - `const Clmg< T > & QR (Clmg< t > &Q, Clmg< t > &R, const bool is_reduced_form=true, const bool is_pivoting=false, Clmg< uintT > *const perm=0) const`

Compute the QR decomposition of the instance matrix.
- template<typename t >

 - `Clmg< T > & project_matrix (const Clmg< t > &dictionary, const unsigned int method=0, const unsigned int max_iter=0, const double max_residual=1e-6)`

Compute the projection of the instance matrix onto the specified dictionary.
- template<typename t >

 - `Clmg< T > & dijkstra (const unsigned int starting_node, const unsigned int ending_node, Clmg< t > &previous_node)`

Return minimal path in a graph, using the Dijkstra algorithm.
- template<typename t >

 - `Clmg< T > get_dijkstra (const unsigned int starting_node, const unsigned int ending_node, Clmg< t > &previous_node) const`

Return minimal path in a graph, using the Dijkstra algorithm [new-instance version].
- `Clmg< T > & dijkstra (const unsigned int starting_node, const unsigned int ending_node=~0U)`

Return minimal path in a graph, using the Dijkstra algorithm.
- `Clmg< Tfloat > get_dijkstra (const unsigned int starting_node, const unsigned int ending_node=~0U) const`

Return minimal path in a graph, using the Dijkstra algorithm [new-instance version].

Color Base Management

- `static const Clmg< Tuchar > & default_LUT256 ()`
- Return palette "default", containing 256 colors entries in RGB.*
- `static const Clmg< Tuchar > & HSV_LUT256 ()`
- Return palette "HSV", containing 256 colors entries in RGB.*

- `static const CImg< Tuchar > & lines_LUT256 ()`
Return palette "lines", containing 256 colors entries in RGB.
- `static const CImg< Tuchar > & hot_LUT256 ()`
Return palette "hot", containing 256 colors entries in RGB.
- `static const CImg< Tuchar > & cool_LUT256 ()`
Return palette "cool", containing 256 colors entries in RGB.
- `static const CImg< Tuchar > & jet_LUT256 ()`
Return palette "jet", containing 256 colors entries in RGB.
- `static const CImg< Tuchar > & flag_LUT256 ()`
Return palette "flag", containing 256 colors entries in RGB.
- `static const CImg< Tuchar > & cube_LUT256 ()`
Return palette "cube", containing 256 colors entries in RGB.
- `CImg< T > & sRGBtoRGB ()`
Convert pixel values from sRGB to RGB color spaces.
- `CImg< Tfloat > get_sRGBtoRGB () const`
Convert pixel values from sRGB to RGB color spaces [new-instance version].
- `CImg< T > & RGBtosRGB ()`
Convert pixel values from RGB to sRGB color spaces.
- `CImg< Tfloat > get_RGBtosRGB () const`
Convert pixel values from RGB to sRGB color spaces [new-instance version].
- `CImg< T > & RGBtoHSI ()`
Convert pixel values from RGB to HSI color spaces.
- `CImg< Tfloat > get_RGBtoHSI () const`
Convert pixel values from RGB to HSI color spaces [new-instance version].
- `CImg< T > & HSItoRGB ()`
Convert pixel values from HSI to RGB color spaces.
- `CImg< Tfloat > get_HSItoRGB () const`
Convert pixel values from HSI to RGB color spaces [new-instance version].
- `CImg< T > & RGBtoHSL ()`
Convert pixel values from RGB to HSL color spaces.
- `CImg< Tfloat > get_RGBtoHSL () const`
Convert pixel values from RGB to HSL color spaces [new-instance version].
- `CImg< T > & HSLtoRGB ()`
Convert pixel values from HSL to RGB color spaces.
- `CImg< Tuchar > get_HSLtoRGB () const`
Convert pixel values from HSL to RGB color spaces [new-instance version].
- `CImg< T > & RGBtoHSV ()`
Convert pixel values from RGB to HSV color spaces.
- `CImg< Tfloat > get_RGBtoHSV () const`
Convert pixel values from RGB to HSV color spaces [new-instance version].
- `CImg< T > & HSVtoRGB ()`
Convert pixel values from HSV to RGB color spaces.
- `CImg< Tuchar > get_HSVtoRGB () const`
Convert pixel values from HSV to RGB color spaces [new-instance version].
- `CImg< T > & RGBtoYCbCr ()`
Convert pixel values from RGB to YCbCr color spaces.
- `CImg< Tuchar > get_RGBtoYCbCr () const`
Convert pixel values from RGB to YCbCr color spaces [new-instance version].
- `CImg< T > & YCbCrtoRGB ()`
Convert pixel values from YCbCr to RGB color spaces.
- `CImg< Tuchar > get_YCbCrtoRGB () const`

- Convert pixel values from RGB to YCbCr color spaces *[new-instance version]*.

 - `Clmg< T > & RGBtoYUV ()`

Convert pixel values from RGB to YUV color spaces.
- `Clmg< Tfloat > get_RGBtoYUV () const`

Convert pixel values from RGB to YUV color spaces *[new-instance version]*.
- `Clmg< T > & YUVtoRGB ()`

Convert pixel values from YUV to RGB color spaces.
- `Clmg< Tuchar > get_YUVtoRGB () const`

Convert pixel values from YUV to RGB color spaces *[new-instance version]*.
- `Clmg< T > & RGBtoCMY ()`

Convert pixel values from RGB to CMY color spaces.
- `Clmg< Tuchar > get_RGBtoCMY () const`

Convert pixel values from RGB to CMY color spaces *[new-instance version]*.
- `Clmg< T > & CMYtoRGB ()`

Convert pixel values from CMY to RGB color spaces.
- `Clmg< Tuchar > get_CMYtoRGB () const`

Convert pixel values from CMY to RGB color spaces *[new-instance version]*.
- `Clmg< T > & CMYtoCMYK ()`

Convert pixel values from CMY to CMYK color spaces.
- `Clmg< Tuchar > get_CMYtoCMYK () const`

Convert pixel values from CMY to CMYK color spaces *[new-instance version]*.
- `Clmg< T > & CMYKtoCMY ()`

Convert pixel values from CMYK to CMY color spaces.
- `Clmg< Tfloat > get_CMYKtoCMY () const`

Convert pixel values from CMYK to CMY color spaces *[new-instance version]*.
- `Clmg< T > & RGBtoXYZ (const bool use_D65=true)`

Convert pixel values from RGB to XYZ color spaces.
- `Clmg< Tfloat > get_RGBtoXYZ (const bool use_D65=true) const`

Convert pixel values from RGB to XYZ color spaces *[new-instance version]*.
- `Clmg< T > & XYZtoRGB (const bool use_D65=true)`

Convert pixel values from XYZ to RGB color spaces.
- `Clmg< Tuchar > get_XYZtoRGB (const bool use_D65=true) const`

Convert pixel values from XYZ to RGB color spaces *[new-instance version]*.
- `Clmg< T > & XYZtoLab (const bool use_D65=true)`

Convert pixel values from XYZ to Lab color spaces.
- `Clmg< Tfloat > get_XYZtoLab (const bool use_D65=true) const`

Convert pixel values from XYZ to Lab color spaces *[new-instance version]*.
- `Clmg< T > & LabtoXYZ (const bool use_D65=true)`

Convert pixel values from Lab to XYZ color spaces.
- `Clmg< Tfloat > get_LabtoXYZ (const bool use_D65=true) const`

Convert pixel values from Lab to XYZ color spaces *[new-instance version]*.
- `Clmg< T > & XYZtoxyY ()`

Convert pixel values from XYZ to xyY color spaces.
- `Clmg< Tfloat > get_XYZtoxyY () const`

Convert pixel values from XYZ to xyY color spaces *[new-instance version]*.
- `Clmg< T > & xyYtoXYZ ()`

Convert pixel values from xyY to XYZ color spaces.
- `Clmg< Tfloat > get_xyYtoXYZ () const`

Convert pixel values from xyY to XYZ color spaces *[new-instance version]*.
- `Clmg< T > & RGBtoLab (const bool use_D65=true)`

Convert pixel values from RGB to Lab color spaces.

- `Clmg< Tfloat > get_RGBtoLab (const bool use_D65=true) const`
Convert pixel values from RGB to Lab color spaces *[new-instance version]*.
- `Clmg< T > & LabtoRGB (const bool use_D65=true)`
Convert pixel values from Lab to RGB color spaces.
- `Clmg< Tuchar > get_LabtoRGB (const bool use_D65=true) const`
Convert pixel values from Lab to RGB color spaces *[new-instance version]*.
- `Clmg< T > & RGBtoxyY (const bool use_D65=true)`
Convert pixel values from RGB to xyY color spaces.
- `Clmg< Tfloat > get_RGBtoxyY (const bool use_D65=true) const`
Convert pixel values from RGB to xyY color spaces *[new-instance version]*.
- `Clmg< T > & xyYtoRGB (const bool use_D65=true)`
Convert pixel values from xyY to RGB color spaces.
- `Clmg< Tuchar > get_xyYtoRGB (const bool use_D65=true) const`
Convert pixel values from xyY to RGB color spaces *[new-instance version]*.
- `Clmg< T > & RGBtoCMYK ()`
Convert pixel values from RGB to CMYK color spaces.
- `Clmg< Tfloat > get_RGBtoCMYK () const`
Convert pixel values from RGB to CMYK color spaces *[new-instance version]*.
- `Clmg< T > & CMYKtoRGB ()`
Convert pixel values from CMYK to RGB color spaces.
- `Clmg< Tuchar > get_CMYKtoRGB () const`
Convert pixel values from CMYK to RGB color spaces *[new-instance version]*.

Geometric / Spatial Manipulation

- `template<typename tfunc >`
`static Clmg< floatT > streamline (const tfunc &func, const float x, const float y, const float z, const float L=256, const float dl=0.1f, const unsigned int interpolation_type=2, const bool is_backward_tracking=false, const bool is_oriented_only=false, const float x0=0, const float y0=0, const float z0=0, const float x1=0, const float y1=0, const float z1=0)`
Return stream line of a 3D vector field.
- `Clmg< T > & resize (const int size_x, const int size_y=-100, const int size_z=-100, const int size_c=-100, const int interpolation_type=1, const unsigned int boundary_conditions=0, const float centering_x=0, const float centering_y=0, const float centering_z=0, const float centering_c=0)`
Resize image to new dimensions.
- `Clmg< T > get_resize (const int size_x, const int size_y=-100, const int size_z=-100, const int size_c=-100, const int interpolation_type=1, const unsigned int boundary_conditions=0, const float centering_x=0, const float centering_y=0, const float centering_z=0, const float centering_c=0) const`
Resize image to new dimensions *[new-instance version]*.
- `template<typename t >`
`Clmg< T > & resize (const Clmg< t > &src, const int interpolation_type=1, const unsigned int boundary_conditions=0, const float centering_x=0, const float centering_y=0, const float centering_z=0, const float centering_c=0)`
Resize image to dimensions of another image.
- `template<typename t >`
`Clmg< T > get_resize (const Clmg< t > &src, const int interpolation_type=1, const unsigned int boundary_conditions=0, const float centering_x=0, const float centering_y=0, const float centering_z=0, const float centering_c=0) const`
Resize image to dimensions of another image *[new-instance version]*.
- `Clmg< T > & resize (const ClmgDisplay &disp, const int interpolation_type=1, const unsigned int boundary_conditions=0, const float centering_x=0, const float centering_y=0, const float centering_z=0, const float centering_c=0)`

- Resize image to dimensions of a display window.*
- `CImg< T > get_resize (const CImgDisplay &disp, const int interpolation_type=1, const unsigned int boundary_conditions=0, const float centering_x=0, const float centering_y=0, const float centering_z=0, const float centering_c=0) const`
- Resize image to dimensions of a display window [new-instance version].*
- `CImg< T > & resize_doubleXY ()`
- Resize image to double-size, using the Scale2X algorithm.*
- `CImg< T > get_resize_doubleXY () const`
- Resize image to double-size, using the Scale2X algorithm [new-instance version].*
- `CImg< T > & resize_tripleXY ()`
- Resize image to triple-size, using the Scale3X algorithm.*
- `CImg< T > get_resize_tripleXY () const`
- Resize image to triple-size, using the Scale3X algorithm [new-instance version].*
- `CImg< T > & mirror (const char axis)`
- Mirror image content along specified axis.*
- `CImg< T > get_mirror (const char axis) const`
- Mirror image content along specified axis [new-instance version].*
- `CImg< T > & mirror (const char *const axes)`
- Mirror image content along specified axes.*
- `CImg< T > get_mirror (const char *const axes) const`
- Mirror image content along specified axes [new-instance version].*
- `CImg< T > & shift (const int delta_x, const int delta_y=0, const int delta_z=0, const int delta_c=0, const unsigned int boundary_conditions=0)`
- Shift the image content.*
- `CImg< T > get_shift (const int delta_x, const int delta_y=0, const int delta_z=0, const int delta_c=0, const unsigned int boundary_conditions=0) const`
- Shift image content [new-instance version].*
- `CImg< T > & permute_axes (const char *const axes_order)`
- Permute axes order.*
- `CImg< T > get_permute_axes (const char *const axes_order) const`
- Permute axes order [new-instance version].*
- `CImg< T > & unroll (const char axis)`
- Unroll pixel values along the specified axis.*
- `CImg< T > get_unroll (const char axis) const`
- Unroll pixel values along specified axis [new-instance version].*
- `CImg< T > & rotate (const float angle, const unsigned int interpolation=1, const unsigned int boundary_conditions=0)`
- Rotate the image by an arbitrary angle.*
- `CImg< T > get_rotate (const float angle, const unsigned int interpolation=1, const unsigned int boundary_conditions=0) const`
- Rotate image with arbitrary angle [new-instance version].*
- `CImg< T > & rotate (const float angle, const float cx, const float cy, const unsigned int interpolation, const unsigned int boundary_conditions=0)`
- Rotate image with arbitrary angle, around a center point.*
- `CImg< T > get_rotate (const float angle, const float cx, const float cy, const unsigned int interpolation, const unsigned int boundary_conditions=0) const`
- Rotate image with arbitrary angle, around a center point [new-instance version].*
- `CImg< T > rotate (const float u, const float v, const float w, const float angle, const unsigned int interpolation, const unsigned int boundary_conditions)`
- Rotate a volumetric image by an arbitrary angle and axis.*
- `CImg< T > get_rotate (const float u, const float v, const float w, const float angle, const unsigned int interpolation, const unsigned int boundary_conditions) const`

Rotate volumetric image with arbitrary angle and axis [**new-instance version**].

- `Clmg< T > rotate (const float u, const float v, const float w, const float angle, const float cx, const float cy, const float cz, const unsigned int interpolation=1, const unsigned int boundary_conditions=0)`

Rotate volumetric image with arbitrary angle and axis, around a center point.

- `Clmg< T > get_rotate (const float u, const float v, const float w, const float angle, const float cx, const float cy, const float cz, const unsigned int interpolation=1, const unsigned int boundary_conditions=0) const`

Rotate volumetric image with arbitrary angle and axis, around a center point [**new-instance version**].

- `template<typename t >`
`Clmg< T > & warp (const Clmg< t > &p_warp, const unsigned int mode=0, const unsigned int interpolation=1, const unsigned int boundary_conditions=0)`

Warp the image content using a warping field.

- `template<typename t >`
`Clmg< T > get_warp (const Clmg< t > &p_warp, const unsigned int mode=0, const unsigned int interpolation=1, const unsigned int boundary_conditions=0) const`

Warp image content by a warping field [**new-instance version**].

- `Clmg< T > get_projections2d (const unsigned int x0, const unsigned int y0, const unsigned int z0) const`

Generate a 2D representation of a 3D image, with XY, XZ and YZ views.

- `Clmg< T > & projections2d (const unsigned int x0, const unsigned int y0, const unsigned int z0)`

Construct a 2D representation of a 3D image, with XY,XZ and YZ views [**in-place version**].

- `Clmg< T > & crop (const int x0, const int y0, const int z0, const int c0, const int x1, const int y1, const int z1, const int c1, const unsigned int boundary_conditions=0)`

Crop image region.

- `Clmg< T > get_crop (const int x0, const int y0, const int z0, const int c0, const int x1, const int y1, const int z1, const int c1, const unsigned int boundary_conditions=0) const`

Crop image region [**new-instance version**].

- `Clmg< T > get_crop (const int x0, const int y0, const int z0, const int x1, const int y1, const int z1, const unsigned int boundary_conditions=0) const`

Crop image region [**new-instance version**].

- `Clmg< T > get_crop (const int x0, const int y0, const int x1, const int y1, const unsigned int boundary_conditions=0) const`

Crop image region [**new-instance version**].

- `Clmg< T > get_crop (const int x0, const int x1, const unsigned int boundary_conditions=0) const`

Crop image region [**new-instance version**].

- `Clmg< T > & autocrop (const T &value, const char *const axes="czyx")`

Autocrop image region, regarding the specified background value.

- `Clmg< T > get_autocrop (const T &value, const char *const axes="czyx") const`

Autocrop image region, regarding the specified background value [**new-instance version**].

- `Clmg< T > & autocrop (const T *const color=0, const char *const axes="zyx")`

Autocrop image region, regarding the specified background color.

- `Clmg< T > get_autocrop (const T *const color=0, const char *const axes="zyx") const`

Autocrop image region, regarding the specified background color [**new-instance version**].

- `Clmg< T > get_column (const int x0) const`

Return specified image column.

- `Clmg< T > & column (const int x0)`

Return specified image column [**in-place version**].

- `Clmg< T > & columns (const int x0, const int x1)`

Return specified range of image columns.

- `Clmg< T > get_columns (const int x0, const int x1) const`

Return specified range of image columns [**in-place version**].

- `Clmg< T > get_row (const int y0) const`

Return specified image row.

- `Clmg< T > & row (const int y0)`

- Return specified image row [in-place version].*

 - `Clmg< T > get_rows (const int y0, const int y1) const`

Return specified range of image rows.

 - `Clmg< T > & rows (const int y0, const int y1)`

Return specified range of image rows [in-place version].

 - `Clmg< T > get_slice (const int z0) const`

Return specified image slice.

 - `Clmg< T > & slice (const int z0)`

Return specified image slice [in-place version].

 - `Clmg< T > get_slices (const int z0, const int z1) const`

Return specified range of image slices.

 - `Clmg< T > & slices (const int z0, const int z1)`

Return specified range of image slices [in-place version].

 - `Clmg< T > get_channel (const int c0) const`

Return specified image channel.

 - `Clmg< T > & channel (const int c0)`

Return specified image channel [in-place version].

 - `Clmg< T > get_channels (const int c0, const int c1) const`

Return specified range of image channels.

 - `Clmg< T > & channels (const int c0, const int c1)`

Return specified range of image channels [in-place version].

 - `Clmg< floatT > get_streamline (const float x, const float y, const float z, const float L=256, const float dl=0.1f, const unsigned int interpolation_type=2, const bool is_backward_tracking=false, const bool is_oriented_only=false) const`

Return stream line of a 2D or 3D vector field.

 - `Clmg< T > get_shared_points (const unsigned int x0, const unsigned int x1, const unsigned int y0=0, const unsigned int z0=0, const unsigned int c0=0)`

Return a shared-memory image referencing a range of pixels of the image instance.

 - `const Clmg< T > get_shared_points (const unsigned int x0, const unsigned int x1, const unsigned int y0=0, const unsigned int z0=0, const unsigned int c0=0) const`

Return a shared-memory image referencing a range of pixels of the image instance [const version].

 - `Clmg< T > get_shared_rows (const unsigned int y0, const unsigned int y1, const unsigned int z0=0, const unsigned int c0=0)`

Return a shared-memory image referencing a range of rows of the image instance.

 - `const Clmg< T > get_shared_rows (const unsigned int y0, const unsigned int y1, const unsigned int z0=0, const unsigned int c0=0) const`

Return a shared-memory image referencing a range of rows of the image instance [const version].

 - `Clmg< T > get_shared_row (const unsigned int y0, const unsigned int z0=0, const unsigned int c0=0)`

Return a shared-memory image referencing one row of the image instance.

 - `const Clmg< T > get_shared_row (const unsigned int y0, const unsigned int z0=0, const unsigned int c0=0) const`

Return a shared-memory image referencing one row of the image instance [const version].

 - `Clmg< T > get_shared_slices (const unsigned int z0, const unsigned int z1, const unsigned int c0=0)`

Return a shared memory image referencing a range of slices of the image instance.

 - `const Clmg< T > get_shared_slices (const unsigned int z0, const unsigned int z1, const unsigned int c0=0) const`

Return a shared memory image referencing a range of slices of the image instance [const version].

 - `Clmg< T > get_shared_slice (const unsigned int z0, const unsigned int c0=0)`

Return a shared-memory image referencing one slice of the image instance.

 - `const Clmg< T > get_shared_slice (const unsigned int z0, const unsigned int c0=0) const`

Return a shared-memory image referencing one slice of the image instance [const version].

 - `Clmg< T > get_shared_channels (const unsigned int c0, const unsigned int c1)`

- Return a shared-memory image referencing a range of channels of the image instance.*

 - `const CImg< T > get_shared_channels (const unsigned int c0, const unsigned int c1) const`

Return a shared-memory image referencing a range of channels of the image instance [const version].
- `CImg< T > get_shared_channel (const unsigned int c0)`

Return a shared-memory image referencing one channel of the image instance.

 - `const CImg< T > get_shared_channel (const unsigned int c0) const`

Return a shared-memory image referencing one channel of the image instance [const version].
- `CImg< T > get_shared ()`

Return a shared-memory version of the image instance.

 - `const CImg< T > get_shared () const`

Return a shared-memory version of the image instance [const version].
- `CImgList< T > get_split (const char axis, const int nb=-1, const unsigned int max_parts=~0U) const`

Split the image into a list along specified axis.
- `template<typename t >`
`CImgList< T > get_split (const CImg< t > &values, const char axis=0, const bool keep_values=true) const`
- `template<typename t >`
`CImg< T > & append (const CImg< t > &img, const char axis='x', const float align=0)`

Append two images along specified axis.
- `template<typename t >`
`CImg< typename cimg::superset< T, t >::type > get_append (const CImg< t > &img, const char axis='x', const float align=0) const`

Append two images along specified axis [const version].

Filtering / Transforms

- `static void FFT (CImg< T > &real, CImg< T > &imag, const char axis, const bool is_inverse=false, const unsigned int nb_threads=0)`

Compute 1D Fast Fourier Transform, along a specified axis.
- `static void FFT (CImg< T > &real, CImg< T > &imag, const bool is_inverse=false, const unsigned int nb_threads=0)`

Compute n-D Fast Fourier Transform.
- `template<typename t >`
`CImg< T > & correlate (const CImg< t > &kernel, const unsigned int boundary_conditions=1, const bool is_normalized=false, const unsigned int channel_mode=1, const int xcenter=(int)(~0U > >1), const int ycenter=(int)(~0U > >1), const int zcenter=(int)(~0U > >1), const unsigned int xstride=1, const unsigned int ystride=1, const unsigned int zstride=1, const int xdilation=1, const int ydilation=1, const int zdilation=1, const int xoffset=0, const int yoffset=0, const int zoffset=0, const unsigned int xsize=~0U, const unsigned int ysize=~0U, const unsigned int zsize=~0U)`

Correlate the image with a kernel.
- `template<typename t >`
`CImg< typename cimg::superset2< T, t, float >::type > get_correlate (const CImg< t > &kernel, const unsigned int boundary_conditions=1, const bool is_normalized=false, const unsigned int channel_mode=1, const int xcenter=(int)(~0U > >1), const int ycenter=(int)(~0U > >1), const int zcenter=(int)(~0U > >1), const unsigned int xstride=1, const unsigned int ystride=1, const unsigned int zstride=1, const int xdilation=1, const int ydilation=1, const int zdilation=1, const int xoffset=0, const int yoffset=0, const int zoffset=0, const unsigned int xsize=~0U, const unsigned int ysize=~0U, const unsigned int zsize=~0U) const`

Correlate the image with a kernel [new-instance version].
- `template<typename t >`
`CImg< T > & convolve (const CImg< t > &kernel, const unsigned int boundary_conditions=1, const bool is_normalized=false, const unsigned int channel_mode=1, const int xcenter=(int)(~0U > >1), const int ycenter=(int)(~0U > >1), const int zcenter=(int)(~0U > >1), const unsigned int xstride=1, const unsigned int ystride=1, const unsigned int zstride=1, const int xdilation=1, const int ydilation=1, const int zdilation=1, const int xoffset=0, const int yoffset=0, const int zoffset=0, const unsigned int xsize=~0U, const unsigned int ysize=~0U, const unsigned int zsize=~0U)`

Convolve the image with a kernel.

- `template<typename t >`
`CImg< typename cimg::superset2< T, t, float >::type > get_convolve (const CImg< t > &kernel, const unsigned int boundary_conditions=1, const bool is_normalized=false, const unsigned int channel_mode=1, const int xcenter=(int)(~0U > >1), const int ycenter=(int)(~0U > >1), const int zcenter=(int)(~0U > >1), const unsigned int xstride=1, const unsigned int ystride=1, const unsigned int zstride=1, const int xdilation=1, const int ydilation=1, const int zdilation=1, const int xoffset=0, const int yoffset=0, const int zoffset=0, const unsigned int xsize=~0U, const unsigned int ysize=~0U, const unsigned int zsize=~0U) const`

Convolve the image with a kernel [new-instance version].

- `CImg< T > & cumulate (const char axis=0)`
Cumulate image values, optionally along specified axis.
- `CImg< Tlong > get_cumulate (const char axis=0) const`
Cumulate image values, optionally along specified axis [new-instance version].
- `CImg< T > & cumulate (const char *const axes)`
Cumulate image values, along specified axes.
- `CImg< Tlong > get_cumulate (const char *const axes) const`
Cumulate image values, along specified axes [new-instance version].
- `template<typename t >`
`CImg< T > & erode (const CImg< t > &kernel, const unsigned int boundary_conditions=1, const bool is_real=false)`

Erode image by a structuring element.

- `template<typename t >`
`CImg< typename cimg::superset< T, t >::type > get_erode (const CImg< t > &kernel, const unsigned int boundary_conditions=1, const bool is_real=false) const`
Erode image by a structuring element [new-instance version].
- `CImg< T > & erode (const unsigned int sx, const unsigned int sy, const unsigned int sz=1)`
Erode image by a rectangular structuring element of specified size.
- `CImg< T > get_erode (const unsigned int sx, const unsigned int sy, const unsigned int sz=1) const`
Erode image by a rectangular structuring element of specified size [new-instance version].
- `CImg< T > & erode (const unsigned int s)`
Erode the image by a square structuring element of specified size.
- `CImg< T > get_erode (const unsigned int s) const`
Erode the image by a square structuring element of specified size [new-instance version].

- `template<typename t >`
`CImg< T > & dilate (const CImg< t > &kernel, const unsigned int boundary_conditions=1, const bool is_real=false)`

Dilate image by a structuring element.

- `template<typename t >`
`CImg< typename cimg::superset< T, t >::type > get_dilate (const CImg< t > &kernel, const unsigned int boundary_conditions=1, const bool is_real=false) const`
Dilate image by a structuring element [new-instance version].
- `CImg< T > & dilate (const unsigned int sx, const unsigned int sy, const unsigned int sz=1)`
Dilate image by a rectangular structuring element of specified size.
- `CImg< T > get_dilate (const unsigned int sx, const unsigned int sy, const unsigned int sz=1) const`
Dilate image by a rectangular structuring element of specified size [new-instance version].
- `CImg< T > & dilate (const unsigned int s)`
Dilate image by a square structuring element of specified size.
- `CImg< T > get_dilate (const unsigned int s) const`
Dilate image by a square structuring element of specified size [new-instance version].

- `template<typename t >`
`CImg< T > & closing (const CImg< t > &kernel, const unsigned int boundary_conditions=1, const bool is_real=false)`

Apply morphological closing by a structuring element.

- `template<typename t >`
`Clmg< T > get_closing (const Clmg< t > &kernel, const unsigned int boundary_conditions=1, const bool is_real=false) const`
Apply morphological closing by a structuring element [new-instance version].
- `Clmg< T > & closing (const unsigned int sx, const unsigned int sy, const unsigned int sz=1)`
Apply morphological closing by a rectangular structuring element of specified size.
- `Clmg< T > get_closing (const unsigned int sx, const unsigned int sy, const unsigned int sz=1) const`
Apply morphological closing by a rectangular structuring element of specified size [new-instance version].
- `Clmg< T > & closing (const unsigned int s)`
Apply morphological closing by a square structuring element of specified size.
- `Clmg< T > get_closing (const unsigned int s) const`
Apply morphological closing by a square structuring element of specified size [new-instance version].
- `template<typename t >`
`Clmg< T > & opening (const Clmg< t > &kernel, const unsigned int boundary_conditions=1, const bool is_real=false)`
Apply morphological opening by a structuring element.
- `template<typename t >`
`Clmg< T > get_opening (const Clmg< t > &kernel, const unsigned int boundary_conditions=1, const bool is_real=false) const`
Apply morphological opening by a structuring element [new-instance version].
- `Clmg< T > & opening (const unsigned int sx, const unsigned int sy, const unsigned int sz=1)`
Apply morphological opening by a rectangular structuring element of specified size.
- `Clmg< T > get_opening (const unsigned int sx, const unsigned int sy, const unsigned int sz=1) const`
Apply morphological opening by a rectangular structuring element of specified size [new-instance version].
- `Clmg< T > & opening (const unsigned int s)`
Apply morphological opening by a square structuring element of specified size.
- `Clmg< T > get_opening (const unsigned int s) const`
Apply morphological opening by a square structuring element of specified size [new-instance version].
- `template<typename t >`
`Clmg< T > & watershed (const Clmg< t > &priority, const bool is_high_connectivity=false)`
Compute watershed transform.
- `template<typename t >`
`Clmg< T > get_watershed (const Clmg< t > &priority, const bool is_high_connectivity=false) const`
Compute watershed transform [new-instance version].
- `Clmg< T > & deriche (const float sigma, const unsigned int order=0, const char axis='x', const unsigned int boundary_conditions=1)`
Apply recursive Deriche filter.
- `Clmg< Tfloat > get_deriche (const float sigma, const unsigned int order=0, const char axis='x', const unsigned int boundary_conditions=1) const`
Apply recursive Deriche filter [new-instance version].
- `Clmg< T > & vanvliet (const float sigma, const unsigned int order, const char axis='x', const unsigned int boundary_conditions=1)`
Van Vliet recursive Gaussian filter.
- `Clmg< Tfloat > get_vanvliet (const float sigma, const unsigned int order, const char axis='x', const unsigned int boundary_conditions=1) const`
Blur image using Van Vliet recursive Gaussian filter. [new-instance version].
- `Clmg< T > & blur (const float sigma_x, const float sigma_y, const float sigma_z, const unsigned int boundary_conditions=1, const bool is_gaussian=true)`
Blur image.
- `Clmg< Tfloat > get_blur (const float sigma_x, const float sigma_y, const float sigma_z, const unsigned int boundary_conditions=1, const bool is_gaussian=true) const`
Blur image [new-instance version].

- `Clmg< T > & blur (const float sigma, const unsigned int boundary_conditions=1, const bool is_gaussian=true)`
Blur image isotropically.
- `Clmg< Tfloat > get_blur (const float sigma, const unsigned int boundary_conditions=1, const bool is_gaussian=true) const`
Blur image isotropically [new-instance version].
- `template<typename t >`
`Clmg< T > & blur_anisotropic (const Clmg< t > &G, const float amplitude=60, const float dl=0.8f, const float da=30, const float gauss_prec=2, const unsigned int interpolation_type=0, const bool is_fast_approx=1)`
Blur image anisotropically, directed by a field of diffusion tensors.
- `template<typename t >`
`Clmg< Tfloat > get_blur_anisotropic (const Clmg< t > &G, const float amplitude=60, const float dl=0.8f, const float da=30, const float gauss_prec=2, const unsigned int interpolation_type=0, const bool is_fast_approx=true) const`
Blur image anisotropically, directed by a field of diffusion tensors [new-instance version].
- `Clmg< T > & blur_anisotropic (const float amplitude, const float sharpness=0.7f, const float anisotropy=0.6f, const float alpha=0.6f, const float sigma=1.1f, const float dl=0.8f, const float da=30, const float gauss_prec=2, const unsigned int interpolation_type=0, const bool is_fast_approx=true)`
Blur image anisotropically, in an edge-preserving way.
- `Clmg< Tfloat > get_blur_anisotropic (const float amplitude, const float sharpness=0.7f, const float anisotropy=0.6f, const float alpha=0.6f, const float sigma=1.1f, const float dl=0.8f, const float da=30, const float gauss_prec=2, const unsigned int interpolation_type=0, const bool is_fast_approx=true) const`
Blur image anisotropically, in an edge-preserving way [new-instance version].
- `template<typename t >`
`Clmg< T > & blur_bilateral (const Clmg< t > &guide, const float sigma_x, const float sigma_y, const float sigma_z, const float sigma_r, const float sampling_x, const float sampling_y, const float sampling_z, const float sampling_r)`
Blur image, with the joint bilateral filter.
- `template<typename t >`
`Clmg< Tfloat > get_blur_bilateral (const Clmg< t > &guide, const float sigma_x, const float sigma_y, const float sigma_z, const float sigma_r, const float sampling_x, const float sampling_y, const float sampling_z, const float sampling_r) const`
Blur image, with the joint bilateral filter [new-instance version].
- `template<typename t >`
`Clmg< T > & blur_bilateral (const Clmg< t > &guide, const float sigma_s, const float sigma_r, const float sampling_s=0, const float sampling_r=0)`
Blur image using the joint bilateral filter.
- `template<typename t >`
`Clmg< Tfloat > get_blur_bilateral (const Clmg< t > &guide, const float sigma_s, const float sigma_r, const float sampling_s=0, const float sampling_r=0) const`
Blur image using the bilateral filter [new-instance version].
- `Clmg< T > & boxfilter (const float boxsize, const int order, const char axis='x', const unsigned int boundary_conditions=1, const unsigned int nb_iter=1)`
- `Clmg< T > & blur_box (const float boxsize_x, const float boxsize_y, const float boxsize_z, const unsigned int boundary_conditions=1, const unsigned int nb_iter=1)`
Blur image with a box filter.
- `Clmg< Tfloat > get_blur_box (const float boxsize_x, const float boxsize_y, const float boxsize_z, const unsigned int boundary_conditions=1) const`
Blur image with a box filter [new-instance version].
- `Clmg< T > & blur_box (const float boxsize, const unsigned int boundary_conditions=1)`
Blur image with a box filter.
- `Clmg< Tfloat > get_blur_box (const float boxsize, const unsigned int boundary_conditions=1) const`
Blur image with a box filter [new-instance version].

- `template<typename t >`
`Clmg< T > & blur_guided (const Clmg< t > &guide, const float radius, const float regularization)`
Blur image, with the image guided filter.
- `template<typename t >`
`Clmg< Tfloat > get_blur_guided (const Clmg< t > &guide, const float radius, const float regularization)`
`const`
Blur image, with the image guided filter [new-instance version].
- `template<typename t >`
`Clmg< T > & blur_patch (const Clmg< t > &guide, const float sigma_s, const float sigma_r, const unsigned int patch_size=3, const unsigned int lookup_size=4, const float smoothness=0, const bool is_fast_approx=true)`
Blur image using patch-based space.
- `template<typename t >`
`Clmg< Tfloat > get_blur_patch (const Clmg< t > &guide, const float sigma_s, const float sigma_r, const unsigned int patch_size=3, const unsigned int lookup_size=4, const float smoothness=0, const bool is_fast_approx=true) const`
Blur image using patch-based space [new-instance version].
- `Clmg< T > & blur_median (const unsigned int n, const float threshold=0)`
Blur image with the median filter.
- `Clmg< T > get_blur_median (const unsigned int n, const float threshold=0) const`
Blur image with the median filter [new-instance version].
- `Clmg< T > & sharpen (const float amplitude, const bool sharpen_type=false, const float edge=1, const float alpha=0, const float sigma=0)`
Sharpen image.
- `Clmg< T > get_sharpen (const float amplitude, const bool sharpen_type=false, const float edge=1, const float alpha=0, const float sigma=0) const`
Sharpen image [new-instance version].
- `ClmgList< Tfloat > get_gradient (const char *const axes=0, const int scheme=0) const`
Return the image gradient.
- `ClmgList< Tfloat > get_hessian (const char *const axes=0) const`
Return the image hessian.
- `Clmg< T > & laplacian ()`
Compute image Laplacian.
- `Clmg< Tfloat > get_laplacian () const`
Compute image Laplacian [new-instance version].
- `Clmg< T > & structure_tensors (const bool is_fwbw_scheme=false)`
Compute the structure tensor field of an image.
- `Clmg< Tfloat > get_structure_tensors (const bool is_fwbw_scheme=false) const`
Compute the structure tensor field of an image [new-instance version].
- `Clmg< T > & diffusion_tensors (const float sharpness=0.7f, const float anisotropy=0.6f, const float alpha=0.6f, const float sigma=1.1f, const bool is_sqrt=false)`
Compute field of diffusion tensors for edge-preserving smoothing.
- `Clmg< Tfloat > get_diffusion_tensors (const float sharpness=0.7f, const float anisotropy=0.6f, const float alpha=0.6f, const float sigma=1.1f, const bool is_sqrt=false) const`
Compute field of diffusion tensors for edge-preserving smoothing [new-instance version].
- `Clmg< T > & displacement (const Clmg< T > &reference, const float smoothness=0.1f, const float precision=7.f, const unsigned int nb_scales=0, const unsigned int iteration_max=1000, const bool is_forward=false, const Clmg< floatT > &guide=Clmg< floatT >::const_empty())`
Estimate the displacement field from a given reference image to the current image instance.
- `Clmg< Tfloat > get_displacement (const Clmg< T > &reference, const float smoothness=0.1f, const float precision=7.f, const unsigned int nb_scales=0, const unsigned int iteration_max=1000, const bool is_forward=false, const Clmg< Tfloat > &guide=Clmg< Tfloat >::const_empty()) const`
Estimate displacement field between two images [new-instance version].

- `template<typename t1 , typename t2 >`
`Clmg< T > & matchpatch (const Clmg< T > &patch_image, const unsigned int patch_width, const unsigned int patch_height, const unsigned int patch_depth, const unsigned int nb_iterations, const unsigned int nb_randoms, const float patch_penalization, const Clmg< t1 > &guide, Clmg< t2 > &matching_score)`
Compute correspondence map between two images, using a patch-matching algorithm.
- `template<typename t1 , typename t2 >`
`Clmg< intT > get_matchpatch (const Clmg< T > &patch_image, const unsigned int patch_width, const unsigned int patch_height, const unsigned int patch_depth, const unsigned int nb_iterations, const unsigned int nb_randoms, const float patch_penalization, const Clmg< t1 > &guide, Clmg< t2 > &matching_score) const`
*Compute correspondence map between two images, using the patch-match algorithm **[new-instance version]**.*
- `Clmg< T > & distance (const T &value, const unsigned int metric=2)`
Compute Euclidean distance function to a specified value.
- `Clmg< Tfloat > get_distance (const T &value, const unsigned int metric=2) const`
*Compute distance to a specified value **[new-instance version]**.*
- `template<typename t >`
`Clmg< T > & distance (const T &value, const Clmg< t > &metric_mask)`
Compute chamfer distance to a specified value, with a custom metric.
- `template<typename t >`
`Clmg< Tfloat > get_distance (const T &value, const Clmg< t > &metric_mask) const`
*Compute chamfer distance to a specified value, with a custom metric **[new-instance version]**.*
- `template<typename t , typename to >`
`Clmg< T > & distance_dijkstra (const T &value, const Clmg< t > &metric, const bool is_high_connectivity, Clmg< to > &return_path)`
Compute distance to a specified value, according to a custom metric (use dijkstra algorithm).
- `template<typename t , typename to >`
`Clmg< typename cimng::superset< t, long >::type > get_distance_dijkstra (const T &value, const Clmg< t > &metric, const bool is_high_connectivity, Clmg< to > &return_path) const`
*Compute distance map to a specified value, according to a custom metric (use dijkstra algorithm) **[new-instance version]**.*
- `template<typename t >`
`Clmg< Tfloat > get_distance_dijkstra (const T &value, const Clmg< t > &metric, const bool is_high_connectivity=false) const`
*Compute distance map to a specified value, according to a custom metric (use dijkstra algorithm). **[new-instance version]**.*
- `template<typename t >`
`Clmg< T > & distance_eikonal (const T &value, const Clmg< t > &metric)`
Compute distance map to one source point, according to a custom metric (use fast marching algorithm).
- `template<typename t >`
`Clmg< Tfloat > get_distance_eikonal (const T &value, const Clmg< t > &metric) const`
Compute distance map to one source point, according to a custom metric (use fast marching algorithm).
- `Clmg< T > & distance_eikonal (const unsigned int nb_iterations, const float band_size=0, const float time_step=0.5f)`
Compute distance function to 0-valued isophotes, using the Eikonal PDE.
- `Clmg< Tfloat > get_distance_eikonal (const unsigned int nb_iterations, const float band_size=0, const float time_step=0.5f) const`
*Compute distance function to 0-valued isophotes, using the Eikonal PDE **[new-instance version]**.*
- `Clmg< T > & haar (const char axis, const bool invert=false, const unsigned int nb_scales=1)`
Compute Haar multiscale wavelet transform.
- `Clmg< Tfloat > get_haar (const char axis, const bool invert=false, const unsigned int nb_scales=1) const`
*Compute Haar multiscale wavelet transform **[new-instance version]**.*
- `Clmg< T > & haar (const bool invert=false, const unsigned int nb_scales=1)`
- `Clmg< Tfloat > get_haar (const bool invert=false, const unsigned int nb_scales=1) const`
*Compute Haar multiscale wavelet transform **[new-instance version]**.*

- `ClmgList< Tfloat > get_FFT (const char axis, const bool is_inverse=false) const`
Compute 1D Fast Fourier Transform, along a specified axis.
- `ClmgList< Tfloat > get_FFT (const bool is_inverse=false) const`
Compute n-D Fast Fourier Transform.

3D Objects Management

- `template<typename tf, typename tfunc >`
`static Clmg< floatT > isoline3d (ClmgList< tf > &primitives, const tfunc &func, const float isovalue, const float x0, const float y0, const float x1, const float y1, const int size_x=256, const int size_y=256)`
Compute isolines of a function, as a 3D object.
- `template<typename tv, typename tf, typename tfunc >`
`static void isoline3d (tv &add_vertex, tf &add_segment, const tfunc &func, const float isovalue, const float x0, const float y0, const float x1, const float y1, const int size_x, const int size_y)`
Compute isolines of a function, as a 3D object.
- `template<typename tf, typename tfunc >`
`static Clmg< floatT > isosurface3d (ClmgList< tf > &primitives, const tfunc &func, const float isovalue, const float x0, const float y0, const float z0, const float x1, const float y1, const float z1, const int size_x=32, const int size_y=32, const int size_z=32)`
Compute isosurface of a function, as a 3D object.
- `template<typename tv, typename tf, typename tfunc >`
`static void isosurface3d (tv &add_vertex, tf &add_triangle, const tfunc &func, const float isovalue, const float x0, const float y0, const float z0, const float x1, const float y1, const float z1, const int size_x, const int size_y, const int size_z)`
Compute isosurface of a function, as a 3D object.
- `template<typename tf, typename tfunc >`
`static Clmg< floatT > elevation3d (ClmgList< tf > &primitives, const tfunc &func, const float x0, const float y0, const float x1, const float y1, const int size_x=256, const int size_y=256)`
Compute 3D elevation of a function as a 3D object.
- `template<typename tf >`
`static Clmg< floatT > box3d (ClmgList< tf > &primitives, const float size_x=200, const float size_y=100, const float size_z=100)`
Generate a 3D box object.
- `template<typename tf >`
`static Clmg< floatT > cone3d (ClmgList< tf > &primitives, const float radius=50, const float size_z=100, const unsigned int subdivisions=24)`
Generate a 3D cone.
- `template<typename tf >`
`static Clmg< floatT > cylinder3d (ClmgList< tf > &primitives, const float radius=50, const float size_z=100, const unsigned int subdivisions=24)`
Generate a 3D cylinder.
- `template<typename tf >`
`static Clmg< floatT > torus3d (ClmgList< tf > &primitives, const float radius1=100, const float radius2=30, const unsigned int subdivisions1=24, const unsigned int subdivisions2=12)`
Generate a 3D torus.
- `template<typename tf >`
`static Clmg< floatT > plane3d (ClmgList< tf > &primitives, const float size_x=100, const float size_y=100, const unsigned int subdivisions_x=10, const unsigned int subdivisions_y=10)`
Generate a 3D XY-plane.
- `template<typename tf >`
`static Clmg< floatT > sphere3d (ClmgList< tf > &primitives, const float radius=50, const unsigned int subdivisions=3)`
Generate a 3D sphere.

- `template<typename tf, typename t >`
`static Clmg< floatT > ellipsoid3d (ClmgList< tf > &primitives, const Clmg< t > &tensor, const unsigned int subdivisions=3)`
Generate a 3D ellipsoid.
- `Clmg< T > & rotate_object3d (const float x, const float y, const float z, const float w, const bool is_quaternion=false)`
Rotate 3D object's vertices.
- `Clmg< T > & shift_object3d (const float tx, const float ty=0, const float tz=0)`
Shift 3D object's vertices.
- `Clmg< Tfloat > get_shift_object3d (const float tx, const float ty=0, const float tz=0) const`
Shift 3D object's vertices [new-instance version].
- `Clmg< T > & shift_object3d ()`
Shift 3D object's vertices, so that it becomes centered.
- `Clmg< Tfloat > get_shift_object3d () const`
Shift 3D object's vertices, so that it becomes centered [new-instance version].
- `Clmg< T > & resize_object3d (const float sx, const float sy=-100, const float sz=-100)`
Resize 3D object.
- `Clmg< Tfloat > get_resize_object3d (const float sx, const float sy=-100, const float sz=-100) const`
Resize 3D object [new-instance version].
- `Clmg< T > resize_object3d ()`
Resize 3D object to unit size.
- `Clmg< Tfloat > get_resize_object3d () const`
Resize 3D object to unit size [new-instance version].
- `template<typename tf, typename tp, typename tff >`
`Clmg< T > & append_object3d (ClmgList< tf > &primitives, const Clmg< tp > &obj_vertices, const ClmgList< tff > &obj_primitives)`
Merge two 3D objects.
- `template<typename tp, typename tc, typename tt, typename tx >`
`const Clmg< T > & texturize_object3d (ClmgList< tp > &primitives, ClmgList< tc > &colors, const Clmg< tt > &texture, const Clmg< tx > &coords=Clmg< tx >::const_empty()) const`
Texturize the primitives of a 3D object.
- `template<typename tf, typename tc, typename te >`
`Clmg< floatT > get_elevation3d (ClmgList< tf > &primitives, ClmgList< tc > &colors, const Clmg< te > &elevation) const`
Generate a 3D elevation map from the instance image, as a 3D object.
- `template<typename tf, typename tc >`
`Clmg< floatT > get_projections3d (ClmgList< tf > &primitives, ClmgList< tc > &colors, const unsigned int x0, const unsigned int y0, const unsigned int z0, const bool normalize_colors=false) const`
Generate the 3D projection planes of the image instance.
- `template<typename tf >`
`Clmg< floatT > get_isoline3d (ClmgList< tf > &primitives, const float isovalue, const int size_x=-100, const int size_y=-100) const`
Generate an isoline from the current image instance, as a 3D object.
- `template<typename tf >`
`Clmg< floatT > get_isosurface3d (ClmgList< tf > &primitives, const float isovalue, const int size_x=-100, const int size_y=-100, const int size_z=-100) const`
Generate an isosurface from the instance image, as a 3D object.
- `template<typename tp, typename tc, typename to >`
`Clmg< T > & object3dtoClmg3d (const ClmgList< tp > &primitives, const ClmgList< tc > &colors, const to &opacities, const bool full_check=true)`
Convert 3D object into a Clmg3d representation.
- `template<typename tp, typename tc, typename to >`
`Clmg< floatT > get_object3dtoClmg3d (const ClmgList< tp > &primitives, const ClmgList< tc > &colors, const to &opacities, const bool full_check=true) const`

Convert 3D object into a `Clmg3d` representation *[new-instance version]*.

- `template<typename tp, typename tc, typename to >`
`Clmg< T > & Clmg3dtoobject3d (ClmgList< tp > &primitives, ClmgList< tc > &colors, ClmgList< to >`
`&opacities, const bool full_check=true)`

Convert `Clmg3d` representation into a 3D object.

- `template<typename tp, typename tc, typename to >`
`Clmg< T > get_Clmg3dtoobject3d (ClmgList< tp > &primitives, ClmgList< tc > &colors, ClmgList< to >`
`&opacities, const bool full_check=true) const`

Convert `Clmg3d` representation into a 3D object *[new-instance version]*.

Data Input

- `static Clmg< T > get_load (const char *const filename)`
Load image from a file *[new-instance version]*.
- `static Clmg< T > get_load_ascii (const char *const filename)`
Load image from an ASCII file *[new-instance version]*.
- `static Clmg< T > get_load_ascii (std::FILE *const file)`
Load image from an ASCII file *[new-instance version]*.
- `static Clmg< T > get_load_dlm (const char *const filename)`
Load image from a DLM file *[new-instance version]*.
- `static Clmg< T > get_load_dlm (std::FILE *const file)`
Load image from a DLM file *[new-instance version]*.
- `static Clmg< T > get_load_bmp (const char *const filename)`
Load image from a BMP file *[new-instance version]*.
- `static Clmg< T > get_load_bmp (std::FILE *const file)`
Load image from a BMP file *[new-instance version]*.
- `static Clmg< T > get_load_jpeg (const char *const filename)`
Load image from a JPEG file *[new-instance version]*.
- `static Clmg< T > get_load_jpeg (std::FILE *const file)`
Load image from a JPEG file *[new-instance version]*.
- `static Clmg< T > get_load_jxl (const char *const filename)`
Load image from a JPEG XL file *[new-instance version]*.
- `static Clmg< T > get_load_jxl (std::FILE *const file)`
Load image from a JPEG XL file *[new-instance version]*.
- `static Clmg< T > get_load_magick (const char *const filename)`
Load image from a file, using Magick++ library *[new-instance version]*.
- `static Clmg< T > get_load_png (const char *const filename, unsigned int *const bits_per_value=0)`
Load image from a PNG file *[new-instance version]*.
- `static Clmg< T > get_load_png (std::FILE *const file, unsigned int *const bits_per_value=0)`
Load image from a PNG file *[new-instance version]*.
- `static Clmg< T > get_load_pnm (const char *const filename)`
Load image from a PNM file *[new-instance version]*.
- `static Clmg< T > get_load_pnm (std::FILE *const file)`
Load image from a PNM file *[new-instance version]*.
- `static Clmg< T > get_load_pfm (const char *const filename)`
Load image from a PFM file *[new-instance version]*.
- `static Clmg< T > get_load_pfm (std::FILE *const file)`
Load image from a PFM file *[new-instance version]*.
- `static Clmg< T > get_load_rgb (const char *const filename, const unsigned int dimw, const unsigned int dimh=1)`
Load image from a RGB file *[new-instance version]*.

- `static Clmg< T > get_load_rgb` (std::FILE *const file, const unsigned int dimw, const unsigned int dimh=1)
Load image from a RGB file *[new-instance version]*.
- `static Clmg< T > get_load_rgba` (const char *const filename, const unsigned int dimw, const unsigned int dimh=1)
Load image from a RGBA file *[new-instance version]*.
- `static Clmg< T > get_load_rgba` (std::FILE *const file, const unsigned int dimw, const unsigned int dimh=1)
Load image from a RGBA file *[new-instance version]*.
- `static Clmg< T > get_load_tiff` (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, unsigned int *const bits_per_value=0, float *const voxel_size=0, Clmg< charT > *const description=0)
Load image from a TIFF file *[new-instance version]*.
- `static Clmg< T > get_load_minc2` (const char *const filename)
Load image from a MINC2 file *[new-instance version]*.
- `static Clmg< T > get_load_analyze` (const char *const filename, float *const voxel_size=0)
Load image from an ANALYZE7.5/NIFTI file *[new-instance version]*.
- `static Clmg< T > get_load_analyze` (std::FILE *const file, float *const voxel_size=0)
Load image from an ANALYZE7.5/NIFTI file *[new-instance version]*.
- `static Clmg< T > get_load_cimg` (const char *const filename, const char axis='z', const float align=0)
Load image from a .cimg[z] file *[new-instance version]*.
- `static Clmg< T > get_load_cimg` (std::FILE *const file, const char axis='z', const float align=0)
Load image from a .cimg[z] file *[new-instance version]*.
- `static Clmg< T > get_load_cimg` (const char *const filename, const unsigned int n0, const unsigned int n1, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0, const unsigned int x1, const unsigned int y1, const unsigned int z1, const unsigned int c1, const char axis='z', const float align=0)
Load sub-images of a .cimg file *[new-instance version]*.
- `static Clmg< T > get_load_cimg` (std::FILE *const file, const unsigned int n0, const unsigned int n1, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0, const unsigned int x1, const unsigned int y1, const unsigned int z1, const unsigned int c1, const char axis='z', const float align=0)
Load sub-images of a .cimg file *[new-instance version]*.
- `static Clmg< T > get_load_inr` (const char *const filename, float *const voxel_size=0)
Load image from an INRIMAGE-4 file *[new-instance version]*.
- `static Clmg< T > get_load_inr` (std::FILE *const file, float *const voxel_size=0)
Load image from an INRIMAGE-4 file *[new-instance version]*.
- `static Clmg< T > get_load_exr` (const char *const filename)
Load image from a EXR file *[new-instance version]*.
- `static Clmg< T > get_load_pandore` (const char *const filename)
Load image from a PANDORE-5 file *[new-instance version]*.
- `static Clmg< T > get_load_pandore` (std::FILE *const file)
Load image from a PANDORE-5 file *[new-instance version]*.
- `static Clmg< T > get_load_parrec` (const char *const filename, const char axis='c', const float align=0)
Load image from a PAR-REC (Philips) file *[new-instance version]*.
- `static Clmg< T > get_load_raw` (const char *const filename, const unsigned int size_x=0, const unsigned int size_y=1, const unsigned int size_z=1, const unsigned int size_c=1, const bool is_multiplexed=false, const bool invert_endianness=false, const unsigned longT offset=0)
Load image from a raw binary file *[new-instance version]*.
- `static Clmg< T > get_load_raw` (std::FILE *const file, const unsigned int size_x=0, const unsigned int size_y=1, const unsigned int size_z=1, const unsigned int size_c=1, const bool is_multiplexed=false, const bool invert_endianness=false, const unsigned longT offset=0)
Load image from a raw binary file *[new-instance version]*.
- `static Clmg< T > get_load_yuv` (const char *const filename, const unsigned int size_x, const unsigned int size_y=1, const unsigned int chroma_subsampling=444, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool yuv2rgb=true, const char axis='z')

Load image sequence from a YUV file *[new-instance version]*.

- `static Clmg< T > get_load_yuv (std::FILE *const file, const unsigned int size_x, const unsigned int size_y=1, const unsigned int chroma_subsampling=444, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool yuv2rgb=true, const char axis='z')`

Load image sequence from a YUV file *[new-instance version]*.

- `template<typename tf, typename tc >
static Clmg< T > get_load_off (ClmgList< tf > &primitives, ClmgList< tc > &colors, const char *const filename)`

Load 3D object from a .OFF file *[new-instance version]*.

- `template<typename tf, typename tc >
static Clmg< T > get_load_off (ClmgList< tf > &primitives, ClmgList< tc > &colors, std::FILE *const file)`

Load 3D object from a .OFF file *[new-instance version]*.

- `static Clmg< T > get_load_video (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const char axis='z', const float align=0)`

Load image sequence from a video file, using OpenCV library *[new-instance version]*.

- `static Clmg< T > get_load_ffmpeg_external (const char *const filename, const char axis='z', const float align=0)`

Load image sequence using FFmpeg's external tool 'ffmpeg' *[new-instance version]*.

- `static Clmg< T > get_load_gif_external (const char *const filename, const char axis='z', const float align=0)`

Load gif file, using ImageMagick or GraphicsMagick's external tool 'convert' *[new-instance version]*.

- `static Clmg< T > get_load_heif (const char *const filename)`

Load image from a HEIC file *[new-instance version]*.

- `static Clmg< T > get_load_webp (const char *const filename)`

Load image from a WebP file *[new-instance version]*.

- `static Clmg< T > get_load_graphicsmagick_external (const char *const filename)`

Load image using GraphicsMagick's external tool 'gm' *[new-instance version]*.

- `static Clmg< T > get_load_gzip_external (const char *const filename)`

Load gzipped image file, using external tool 'gunzip' *[new-instance version]*.

- `static Clmg< T > get_load_imagemagick_external (const char *const filename)`

Load image using ImageMagick's external tool 'convert' *[new-instance version]*.

- `static Clmg< T > get_load_medcon_external (const char *const filename)`

Load image from a DICOM file, using Medcon's external tool 'medcon' *[new-instance version]*.

- `static Clmg< T > get_load_pdf_external (const char *const filename, const unsigned int resolution=400)`

Load image from a .pdf file *[new-instance version]*.

- `static Clmg< T > get_load_dcraw_external (const char *const filename)`

Load image from a RAW Color Camera file, using external tool 'dcraw' *[new-instance version]*.

- `static Clmg< T > get_load_camera (const unsigned int camera_index=0, const unsigned int capture_width=0, const unsigned int capture_height=0, const unsigned int skip_frames=0, const bool release_camera=true)`

Load image from a camera stream, using OpenCV *[new-instance version]*.

- `static Clmg< T > get_load_other (const char *const filename)`

Load image using various non-native ways *[new-instance version]*.

- `Clmg< T > & select (ClmgDisplay &disp, const unsigned int feature_type=2, unsigned int *const XYZ=0, const bool exit_on_anykey=false, const bool is_deep_selection_default=false)`

Launches a simple interface to select a shape from an image.

- `Clmg< intT > get_select (ClmgDisplay &disp, const unsigned int feature_type=2, unsigned int *const XYZ=0, const bool exit_on_anykey=false, const bool is_deep_selection_default=false) const`

Simple interface to select a shape from an image *[new-instance version]*.

- `Clmg< intT > get_select (const char *const title, const unsigned int feature_type=2, unsigned int *const XYZ=0, const bool exit_on_anykey=false, const bool is_deep_selection_default=false) const`

Simple interface to select a shape from an image *[new-instance version]*.

- `Clmg< intT > get_select_graph (ClmgDisplay &disp, const unsigned int plot_type=1, const unsigned int vertex_type=1, const char *const labelx=0, const double xmin=0, const double xmax=0, const char *const labely=0, const double ymin=0, const double ymax=0, const bool exit_on_anykey=false) const`
Select a sub-graph within a graph.
- `Clmg< T > & load (const char *const filename)`
Load image from a file.
- `Clmg< T > & load_ascii (const char *const filename)`
Load image from an ASCII file.
- `Clmg< T > & load_ascii (std::FILE *const file)`
Load image from an ASCII file *[in-place version]*.
- `Clmg< T > & load_dlm (const char *const filename)`
Load image from a DLM file.
- `Clmg< T > & load_dlm (std::FILE *const file)`
Load image from a DLM file *[in-place version]*.
- `Clmg< T > & load_bmp (const char *const filename)`
Load image from a BMP file.
- `Clmg< T > & load_bmp (std::FILE *const file)`
Load image from a BMP file *[in-place version]*.
- `Clmg< T > & load_jpeg (const char *const filename)`
Load image from a JPEG file.
- `Clmg< T > & load_jpeg (std::FILE *const file)`
Load image from a JPEG file *[in-place version]*.
- `Clmg< T > & load_jxl (const char *const filename)`
Load image from a JPEG XL file.
- `Clmg< T > & load_jxl (std::FILE *const file)`
Load image from a JPEG XL file *[in-place version]*.
- `Clmg< T > & load_magick (const char *const filename)`
Load image from a file, using Magick++ library.
- `Clmg< T > & load_png (const char *const filename, unsigned int *const bits_per_value=0)`
Load an image from a PNG file.
- `Clmg< T > & load_png (std::FILE *const file, unsigned int *const bits_per_value=0)`
Load image from a PNG file *[in-place version]*.
- `Clmg< T > & load_pnm (const char *const filename)`
Load image from a PNM file.
- `Clmg< T > & load_pnm (std::FILE *const file)`
Load image from a PNM file *[in-place version]*.
- `Clmg< T > & load_pfm (const char *const filename)`
Load image from a PFM file.
- `Clmg< T > & load_pfm (std::FILE *const file)`
Load image from a PFM file *[in-place version]*.
- `Clmg< T > & load_rgb (const char *const filename, const unsigned int dimw, const unsigned int dimh=1)`
Load image from a RGB file.
- `Clmg< T > & load_rgb (std::FILE *const file, const unsigned int dimw, const unsigned int dimh=1)`
Load image from a RGB file *[in-place version]*.
- `Clmg< T > & load_rgba (const char *const filename, const unsigned int dimw, const unsigned int dimh=1)`
Load image from a RGBA file.
- `Clmg< T > & load_rgba (std::FILE *const file, const unsigned int dimw, const unsigned int dimh=1)`
Load image from a RGBA file *[in-place version]*.
- `Clmg< T > & load_tiff (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, unsigned int *const bits_per_value=0, float *const voxel_size=0, Clmg< charT > *const description=0)`

- Load image from a TIFF file.*

 - `Clmg< T > & load_minc2 (const char *const filename)`
- Load image from a MINC2 file.*

 - `Clmg< T > & load_analyze (const char *const filename, float *const voxel_size=0)`
- Load image from an ANALYZE7.5/NIFTI file.*

 - `Clmg< T > & load_analyze (std::FILE *const file, float *const voxel_size=0)`
- Load image from an ANALYZE7.5/NIFTI file [in-place version].*

 - `Clmg< T > & load_cimg (const char *const filename, const char axis='z', const float align=0)`
- Load image from a .cimg[z] file.*

 - `Clmg< T > & load_cimg (std::FILE *const file, const char axis='z', const float align=0)`
- Load image from a .cimg[z] file [in-place version].*

 - `Clmg< T > & load_cimg (const char *const filename, const unsigned int n0, const unsigned int n1, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0, const unsigned int x1, const unsigned int y1, const unsigned int z1, const unsigned int c1, const char axis='z', const float align=0)`
- Load sub-images of a .cimg file.*

 - `Clmg< T > & load_cimg (std::FILE *const file, const unsigned int n0, const unsigned int n1, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0, const unsigned int x1, const unsigned int y1, const unsigned int z1, const unsigned int c1, const char axis='z', const float align=0)`
- Load sub-images of a .cimg file [in-place version].*

 - `Clmg< T > & load_inr (const char *const filename, float *const voxel_size=0)`
- Load image from an INRIMAGE-4 file.*

 - `Clmg< T > & load_inr (std::FILE *const file, float *const voxel_size=0)`
- Load image from an INRIMAGE-4 file [in-place version].*

 - `Clmg< T > & load_exr (const char *const filename)`
- Load image from a EXR file.*

 - `Clmg< T > & load_pandore (const char *const filename)`
- Load image from a PANDORE-5 file.*

 - `Clmg< T > & load_pandore (std::FILE *const file)`
- Load image from a PANDORE-5 file [in-place version].*

 - `Clmg< T > & load_parrec (const char *const filename, const char axis='c', const float align=0)`
- Load image from a PAR-REC (Philips) file.*

 - `Clmg< T > & load_raw (const char *const filename, const unsigned int size_x=0, const unsigned int size_y=1, const unsigned int size_z=1, const unsigned int size_c=1, const bool is_multiplexed=false, const bool invert_endianness=false, const ulongT offset=0)`
- Load image from a raw binary file.*

 - `Clmg< T > & load_raw (std::FILE *const file, const unsigned int size_x=0, const unsigned int size_y=1, const unsigned int size_z=1, const unsigned int size_c=1, const bool is_multiplexed=false, const bool invert_endianness=false, const ulongT offset=0)`
- Load image from a raw binary file [in-place version].*

 - `Clmg< T > & load_yuv (const char *const filename, const unsigned int size_x, const unsigned int size_y=1, const unsigned int chroma_subsampling=444, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool yuv2rgb=true, const char axis='z')`
- Load image sequence from a YUV file.*

 - `Clmg< T > & load_yuv (std::FILE *const file, const unsigned int size_x, const unsigned int size_y=1, const unsigned int chroma_subsampling=444, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool yuv2rgb=true, const char axis='z')`
- Load image sequence from a YUV file [in-place version].*

 - `template<typename tf, typename tc>`
`Clmg< T > & load_off (ClmgList< tf > &primitives, ClmgList< tc > &colors, const char *const filename)`
- Load 3D object from a .OFF file.*

 - `template<typename tf, typename tc>`
`Clmg< T > & load_off (ClmgList< tf > &primitives, ClmgList< tc > &colors, std::FILE *const file)`

- Load 3D object from a .OFF file [in-place version].*

 - `CImg< T > & load_video (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const char axis='z', const float align=0)`

Load image sequence from a video file, using OpenCV library.
- `CImg< T > & load_ffmpeg_external (const char *const filename, const char axis='z', const float align=0)`

Load image sequence using FFmpeg's external tool 'ffmpeg'.
- `CImg< T > & load_gif_external (const char *const filename, const char axis='z', const float align=0)`

Load gif file, using ImageMagick or GraphicsMagick's external tools.
- `CImg< T > & load_heif (const char *const filename)`

Load image from a HEIC file.
- `CImg< T > & load_webp (const char *const filename)`

Load image from a WebP file.
- `CImg< T > & load_graphicsmagick_external (const char *const filename)`

Load image using GraphicsMagick's external tool 'gm'.
- `CImg< T > & load_gzip_external (const char *const filename)`

Load gzipped image file, using external tool 'gunzip'.
- `CImg< T > & load_imagemagick_external (const char *const filename)`

Load image using ImageMagick's external tool 'convert'.
- `CImg< T > & load_medcon_external (const char *const filename)`

Load image from a DICOM file, using Medcon's external tool 'medcon'.
- `CImg< T > & load_pdf_external (const char *const filename, const unsigned int resolution=400)`

Load image from a .pdf file.
- `CImg< T > & load_dcraw_external (const char *const filename)`

Load image from a RAW Color Camera file, using external tool 'dcraw'.
- `CImg< T > & load_camera (const unsigned int camera_index=0, const unsigned int capture_width=0, const unsigned int capture_height=0, const unsigned int skip_frames=0, const bool release_camera=true)`

Captures an image from a connected camera device (requires OpenCV).
- `CImg< T > & load_other (const char *const filename)`

Load image using various non-native ways.

Data Output

- `static void save_empty_cimg (const char *const filename, const unsigned int dx, const unsigned int dy=1, const unsigned int dz=1, const unsigned int dc=1)`

Save a blank image as a .cimg file.
- `static void save_empty_cimg (std::FILE *const file, const unsigned int dx, const unsigned int dy=1, const unsigned int dz=1, const unsigned int dc=1)`
- `const CImg< T > & print (const char *const title=0, const bool display_stats=true) const`

Display information about the image data.
- `const CImg< T > & display (CImgDisplay &disp) const`

Display image into a CImgDisplay window.
- `const CImg< T > & display (CImgDisplay &disp, const bool display_info, unsigned int *const XYZ=0, const bool exit_on_anykey=false) const`

Display image into a CImgDisplay window, in an interactive way.
- `const CImg< T > & display (const char *const title=0, const bool display_info=true, unsigned int *const XYZ=0, const bool exit_on_anykey=false) const`

Display image into an interactive window.

- `template<typename tp, typename tf, typename tc, typename to >`
`const Clmg< T > & display_object3d (ClmgDisplay &disp, const Clmg< tp > &vertices, const ClmgList<`
`tf > &primitives, const ClmgList< tc > &colors, const to &opacities, const bool centering=true, const`
`int render_static=4, const int render_motion=1, const bool is_double_sided=false, const float focal=700,`
`const float light_x=0, const float light_y=0, const float light_z=-5e8f, const float specular_lightness=0.25f,`
`const float specular_shininess=0.1f, const bool display_axes=true, float *const pose_matrix=0, const bool`
`exit_on_anykey=false) const`
Display a 3D object in an interactive window.
- `const Clmg< T > & display_graph (ClmgDisplay &disp, const unsigned int plot_type=1, const unsigned int`
`vertex_type=1, const char *const labelx=0, const double xmin=0, const double xmax=0, const char *const`
`labely=0, const double ymin=0, const double ymax=0, const bool exit_on_anykey=false) const`
Display 1D graph in an interactive window.
- `const Clmg< T > & save (const char *const filename, const int number=-1, const unsigned int digits=6) const`
Save the image as a file.
- `const Clmg< T > & save_ascii (const char *const filename) const`
Save the image as an ASCII file.
- `const Clmg< T > & save_cpp (const char *const filename) const`
Save the image as a .cpp source file.
- `const Clmg< T > & save_dlm (const char *const filename) const`
Save the image as a DLM file.
- `const Clmg< T > & save_bmp (const char *const filename) const`
Save the image as a BMP file.
- `const Clmg< T > & save_webp (const char *const filename, const int quality=100) const`
Save the image as a WebP file.
- `const Clmg< T > & save_jpeg (const char *const filename, const unsigned int quality=100) const`
Save the image as a JPEG file.
- `const Clmg< T > & save_jxl (const char *const filename, const float distance=1.0f, const unsigned int`
`bytes_per_pixel=0) const`
Save the image as a JPEG XL file.
- `const Clmg< T > & save_magick (const char *const filename, const unsigned int bytes_per_pixel=0) const`
Save the image, using built-in ImageMagick++ library.
- `const Clmg< T > & save_png (const char *const filename, const unsigned int bytes_per_pixel=0) const`
Save the image as a PNG file.
- `const Clmg< T > & save_pnm (const char *const filename, const unsigned int bytes_per_pixel=0) const`
Save the image as a PNM file.
- `const Clmg< T > & save_pnk (const char *const filename) const`
Save the image as a PNK file.
- `const Clmg< T > & save_pfm (const char *const filename) const`
Save the image as a PFM file.
- `const Clmg< T > & save_rgb (const char *const filename) const`
Save the image as a RGB file.
- `const Clmg< T > & save_rgba (const char *const filename) const`
Save the image as a RGBA file.
- `const Clmg< T > & save_tiff (const char *const filename, const unsigned int compression_type=0, const`
`float *const voxel_size=0, const char *const description=0, const bool use_bigtiff=true) const`
Save the image as a TIFF file.
- `const Clmg< T > & save_minc2 (const char *const filename, const char *const imitate_file=0) const`
Save the image as a MINC2 file.
- `const Clmg< T > & save_analyze (const char *const filename, const float *const voxel_size=0) const`
Save the image as an ANALYZE7.5 or NIFTI file.
- `const Clmg< T > & save_cimg (const char *const filename, const bool is_compressed=false) const`
Save the image as a .cimg file.

- `const Clmg< T > & save_cimg (const char *const filename, const unsigned int n0, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0) const`
Save the image as a sub-image into an existing .cimg file.
- `const Clmg< T > & save_inr (const char *const filename, const float *const voxel_size=0) const`
Save the image as an INRIMAGE-4 file.
- `const Clmg< T > & save_exr (const char *const filename) const`
Save the image as an OpenEXR file.
- `const Clmg< T > & save_pandore (const char *const filename, const unsigned int colorspace=0) const`
Save the image as a Pandore-5 file.
- `const Clmg< T > & save_pandore (std::FILE *const file, const unsigned int colorspace=0) const`
- `const Clmg< T > & save_raw (const char *const filename, const bool is_multiplexed=false) const`
Save the image as a raw data file.
- `const Clmg< T > & save_raw (std::FILE *const file, const bool is_multiplexed=false) const`
- `const Clmg< T > & save_yuv (const char *const filename, const unsigned int chroma_subsampling=444, const bool is_rgb=true) const`
Save the image as a .yuv video file.
- `const Clmg< T > & save_yuv (std::FILE *const file, const unsigned int chroma_subsampling=444, const bool is_rgb=true) const`
- `template<typename tf, typename tc>
const Clmg< T > & save_off (const ClmgList< tf > &primitives, const ClmgList< tc > &colors, const char *const filename) const`
Save 3D object as an Object File Format (.off) file.
- `template<typename tf, typename tc>
const Clmg< T > & save_off (const ClmgList< tf > &primitives, const ClmgList< tc > &colors, std::FILE *const file) const`
- `const Clmg< T > & save_video (const char *const filename, const unsigned int fps=25, const char *const codec=0, const bool keep_open=false) const`
Save volumetric image as a video (using the OpenCV library when available).
- `const Clmg< T > & save_ffmpeg_external (const char *const filename, const unsigned int fps=25, const char *const codec=0, const unsigned int bitrate=2048) const`
Save volumetric image as a video, using ffmpeg external binary.
- `const Clmg< T > & save_gzip_external (const char *const filename) const`
Save the image using gzip external binary.
- `const Clmg< T > & save_graphicmagick_external (const char *const filename, const unsigned int quality=100) const`
Save the image using GraphicsMagick's external binary.
- `const Clmg< T > & save_imagemagick_external (const char *const filename, const unsigned int quality=100) const`
Save the image using ImageMagick's external binary.
- `const Clmg< T > & save_medcon_external (const char *const filename) const`
Save the image as a Dicom file.
- `const Clmg< T > & save_other (const char *const filename, const unsigned int quality=100) const`
- `Clmg< ucharT > get_serialize (const bool is_compressed=false, const unsigned int header_size=0) const`
Serialize a Clmg<T> instance into a raw Clmg<unsigned char> buffer.

Overloaded Operators

- `T & operator() (const unsigned int x, const unsigned int y=0, const unsigned int z=0, const unsigned int c=0)`
Access to a pixel value.
- `const T & operator() (const unsigned int x, const unsigned int y=0, const unsigned int z=0, const unsigned int c=0) const`
Access to a pixel value [const version].

- `T & operator() (const unsigned int x, const unsigned int y, const unsigned int z, const unsigned int c, const unsigned longT wh, const unsigned longT whd=0)`
Access to a pixel value.
- `const T & operator() (const unsigned int x, const unsigned int y, const unsigned int z, const unsigned int c, const unsigned longT wh, const unsigned longT whd=0) const`
*Access to a pixel value **[const version]**.*
- `operator T* ()`
Implicitly cast an image into a T.*
- `operator const T * () const`
Implicitly cast an image into a T **[const version]**.*
- `CImg< T > & operator= (const T &value)`
Assign a value to all image pixels.
- `CImg< T > & operator= (const char *const expression)`
Assign pixels values from a specified expression.
- `template<typename t > CImg< T > & operator= (const CImg< t > &img)`
Copy an image into the current image instance.
- `CImg< T > & operator= (const CImg< T > &img)`
*Copy an image into the current image instance **[specialization]**.*
- `CImg< T > & operator= (const CImgDisplay &disp)`
Copy the content of a display window to the current image instance.
- `template<typename t > CImg< T > & operator+= (const t value)`
In-place addition operator.
- `CImg< T > & operator+= (const char *const expression)`
In-place addition operator.
- `template<typename t > CImg< T > & operator+= (const CImg< t > &img)`
In-place addition operator.
- `CImg< T > & operator++ ()`
In-place increment operator (prefix).
- `CImg< T > operator++ (int)`
In-place increment operator (postfix).
- `CImg< T > operator+ () const`
Return a non-shared copy of the image instance.
- `template<typename t > CImg< typename cimg::superset< T, t>::type > operator+ (const t value) const`
Addition operator.
- `CImg< Tfloat > operator+ (const char *const expression) const`
Addition operator.
- `template<typename t > CImg< typename cimg::superset< T, t>::type > operator+ (const CImg< t > &img) const`
Addition operator.
- `template<typename t > CImg< T > & operator-= (const t value)`
In-place subtraction operator.
- `CImg< T > & operator-= (const char *const expression)`
In-place subtraction operator.
- `template<typename t > CImg< T > & operator-= (const CImg< t > &img)`
In-place subtraction operator.
- `CImg< T > & operator-- ()`

- In-place decrement operator (prefix).*
- `CImg< T > operator-- (int)`
In-place decrement operator (postfix).
- `CImg< T > operator- () const`
Replace each pixel by its opposite value.
- `template<typename t >`
`CImg< typename cimg::superset< T, t >::type > operator- (const t value) const`
Subtraction operator.
- `CImg< Tfloat > operator- (const char *const expression) const`
Subtraction operator.
- `template<typename t >`
`CImg< typename cimg::superset< T, t >::type > operator- (const CImg< t > &img) const`
Subtraction operator.
- `template<typename t >`
`CImg< T > & operator*= (const t value)`
In-place multiplication operator.
- `CImg< T > & operator*= (const char *const expression)`
In-place multiplication operator.
- `template<typename t >`
`CImg< T > & operator*= (const CImg< t > &img)`
In-place multiplication operator.
- `template<typename t >`
`CImg< typename cimg::superset< T, t >::type > operator* (const t value) const`
Multiplication operator.
- `CImg< Tfloat > operator* (const char *const expression) const`
Multiplication operator.
- `template<typename t >`
`CImg< typename cimg::superset< T, t >::type > operator* (const CImg< t > &img) const`
Multiplication operator.
- `template<typename t >`
`CImg< T > & operator/= (const t value)`
In-place division operator.
- `CImg< T > & operator/= (const char *const expression)`
In-place division operator.
- `template<typename t >`
`CImg< T > & operator/= (const CImg< t > &img)`
In-place division operator.
- `template<typename t >`
`CImg< typename cimg::superset< T, t >::type > operator/ (const t value) const`
Division operator.
- `CImg< Tfloat > operator/ (const char *const expression) const`
Division operator.
- `template<typename t >`
`CImg< typename cimg::superset< T, t >::type > operator/ (const CImg< t > &img) const`
Division operator.
- `template<typename t >`
`CImg< T > & operator%= (const t value)`
In-place modulo operator.
- `CImg< T > & operator%= (const char *const expression)`
In-place modulo operator.
- `template<typename t >`
`CImg< T > & operator%= (const CImg< t > &img)`

- In-place modulo operator.*

 - `template<typename t >`
`CImg< typename cimg::superset< T, t >::type > operator% (const t value) const`

Modulo operator.
- `CImg< Tfloat > operator% (const char *const expression) const`

Modulo operator.
- `template<typename t >`
`CImg< typename cimg::superset< T, t >::type > operator% (const CImg< t > &img) const`

Modulo operator.
- `template<typename t >`
`CImg< T > & operator&= (const t value)`

In-place bitwise AND operator.
- `CImg< T > & operator&= (const char *const expression)`

In-place bitwise AND operator.
- `template<typename t >`
`CImg< T > & operator&= (const CImg< t > &img)`

In-place bitwise AND operator.
- `template<typename t >`
`CImg< T > operator& (const t value) const`

Bitwise AND operator.
- `CImg< T > operator& (const char *const expression) const`

Bitwise AND operator.
- `template<typename t >`
`CImg< T > operator& (const CImg< t > &img) const`

Bitwise AND operator.
- `template<typename t >`
`CImg< T > & operator|= (const t value)`

In-place bitwise OR operator.
- `CImg< T > & operator|= (const char *const expression)`

In-place bitwise OR operator.
- `template<typename t >`
`CImg< T > & operator|= (const CImg< t > &img)`

In-place bitwise OR operator.
- `template<typename t >`
`CImg< T > operator| (const t value) const`

Bitwise OR operator.
- `CImg< T > operator| (const char *const expression) const`

Bitwise OR operator.
- `template<typename t >`
`CImg< T > operator| (const CImg< t > &img) const`

Bitwise OR operator.
- `template<typename t >`
`CImg< T > & operator^= (const t value)`

In-place bitwise XOR operator.
- `CImg< T > & operator^= (const char *const expression)`

In-place bitwise XOR operator.
- `template<typename t >`
`CImg< T > & operator^= (const CImg< t > &img)`

In-place bitwise XOR operator.
- `template<typename t >`
`CImg< T > operator^ (const t value) const`

Bitwise XOR operator.

- `CImg< T > operator^ (const char *const expression) const`
Bitwise XOR operator.
- `template<typename t >`
`CImg< T > operator^ (const CImg< t > &img) const`
Bitwise XOR operator.
- `template<typename t >`
`CImg< T > & operator<<= (const t value)`
In-place bitwise left shift operator.
- `CImg< T > & operator<<= (const char *const expression)`
In-place bitwise left shift operator.
- `template<typename t >`
`CImg< T > & operator<<= (const CImg< t > &img)`
In-place bitwise left shift operator.
- `template<typename t >`
`CImg< T > operator<< (const t value) const`
Bitwise left shift operator.
- `CImg< T > operator<< (const char *const expression) const`
Bitwise left shift operator.
- `template<typename t >`
`CImg< T > operator<< (const CImg< t > &img) const`
Bitwise left shift operator.
- `template<typename t >`
`CImg< T > & operator>>= (const t value)`
In-place bitwise right shift operator.
- `CImg< T > & operator>>= (const char *const expression)`
In-place bitwise right shift operator.
- `template<typename t >`
`CImg< T > & operator>>= (const CImg< t > &img)`
In-place bitwise right shift operator.
- `template<typename t >`
`CImg< T > operator>> (const t value) const`
Bitwise right shift operator.
- `CImg< T > operator>> (const char *const expression) const`
Bitwise right shift operator.
- `template<typename t >`
`CImg< T > operator>> (const CImg< t > &img) const`
Bitwise right shift operator.
- `CImg< T > operator~ () const`
Bitwise inversion operator.
- `template<typename t >`
`bool operator== (const t value) const`
Test if all pixels of an image have the same value.
- `bool operator== (const char *const expression) const`
Test if all pixel values of an image follow a specified expression.
- `template<typename t >`
`bool operator== (const CImg< t > &img) const`
Test if two images have the same size and values.
- `template<typename t >`
`bool operator!= (const t value) const`
Test if pixels of an image are all different from a value.
- `bool operator!= (const char *const expression) const`
Test if all pixel values of an image are different from a specified expression.

- `template<typename t >`
`bool operator!= (const CImg< t > &img) const`
Test if two images have different sizes or values.
- `template<typename t >`
`CImgList< typename cimg::superset< T, t >::type > operator, (const CImg< t > &img) const`
Construct an image list from two images.
- `template<typename t >`
`CImgList< typename cimg::superset< T, t >::type > operator, (const CImgList< t > &list) const`
Construct an image list from image instance and an input image list.
- `CImgList< T > operator< (const char axis) const`
Split the image along a specified axis.

Instance Checking

- `bool is_shared () const`
Test shared state of the pixel buffer.
- `bool is_empty () const`
Test if image instance is empty.
- `bool is_inf () const`
Test if image instance contains a 'inf' value.
- `bool is_nan () const`
Test if image instance contains a NaN value.
- `bool is_sameX (const unsigned int size_x) const`
Test if image width is equal to specified value.
- `template<typename t >`
`bool is_sameX (const CImg< t > &img) const`
Test if image width is equal to specified value.
- `bool is_sameX (const CImgDisplay &disp) const`
Test if image width is equal to specified value.
- `bool is_sameY (const unsigned int size_y) const`
Test if image height is equal to specified value.
- `template<typename t >`
`bool is_sameY (const CImg< t > &img) const`
Test if image height is equal to specified value.
- `bool is_sameY (const CImgDisplay &disp) const`
Test if image height is equal to specified value.
- `bool is_sameZ (const unsigned int size_z) const`
Test if image depth is equal to specified value.
- `template<typename t >`
`bool is_sameZ (const CImg< t > &img) const`
Test if image depth is equal to specified value.
- `bool is_sameC (const unsigned int size_c) const`
Test if image spectrum is equal to specified value.
- `template<typename t >`
`bool is_sameC (const CImg< t > &img) const`
Test if image spectrum is equal to specified value.
- `bool is_sameXY (const unsigned int size_x, const unsigned int size_y) const`
Test if image width and height are equal to specified values.
- `template<typename t >`
`bool is_sameXY (const CImg< t > &img) const`
Test if image width and height are the same as those of another image.

- `bool is_sameXY (const ClmgDisplay &disp) const`
Test if image width and height are the same as that of an existing display window.
- `bool is_sameXZ (const unsigned int size_x, const unsigned int size_z) const`
Test if image width and depth are equal to specified values.
- `template<typename t >`
`bool is_sameXZ (const Clmg< t > &img) const`
Test if image width and depth are the same as those of another image.
- `bool is_sameXC (const unsigned int size_x, const unsigned int size_c) const`
Test if image width and spectrum are equal to specified values.
- `template<typename t >`
`bool is_sameXC (const Clmg< t > &img) const`
Test if image width and spectrum are the same as those of another image.
- `bool is_sameYZ (const unsigned int size_y, const unsigned int size_z) const`
Test if image height and depth are equal to specified values.
- `template<typename t >`
`bool is_sameYZ (const Clmg< t > &img) const`
Test if image height and depth are the same as those of another image.
- `bool is_sameYC (const unsigned int size_y, const unsigned int size_c) const`
Test if image height and spectrum are equal to specified values.
- `template<typename t >`
`bool is_sameYC (const Clmg< t > &img) const`
Test if image height and spectrum are the same as those of another image.
- `bool is_sameZC (const unsigned int size_z, const unsigned int size_c) const`
Test if image depth and spectrum are equal to specified values.
- `template<typename t >`
`bool is_sameZC (const Clmg< t > &img) const`
Test if image depth and spectrum are the same as those of another image.
- `bool is_sameXYZ (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z) const`
Test if image width, height and depth are equal to specified values.
- `template<typename t >`
`bool is_sameXYZ (const Clmg< t > &img) const`
Test if image width, height and depth are the same as those of another image.
- `bool is_sameXYC (const unsigned int size_x, const unsigned int size_y, const unsigned int size_c) const`
Test if image width, height and spectrum are equal to specified values.
- `template<typename t >`
`bool is_sameXYC (const Clmg< t > &img) const`
Test if image width, height and spectrum are the same as those of another image.
- `bool is_sameXZC (const unsigned int size_x, const unsigned int size_z, const unsigned int size_c) const`
Test if image width, depth and spectrum are equal to specified values.
- `template<typename t >`
`bool is_sameXZC (const Clmg< t > &img) const`
Test if image width, depth and spectrum are the same as those of another image.
- `bool is_sameYZC (const unsigned int size_y, const unsigned int size_z, const unsigned int size_c) const`
Test if image height, depth and spectrum are equal to specified values.
- `template<typename t >`
`bool is_sameYZC (const Clmg< t > &img) const`
Test if image height, depth and spectrum are the same as those of another image.
- `bool is_sameXYZC (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c) const`
Test if image width, height, depth and spectrum are equal to specified values.
- `template<typename t >`
`bool is_sameXYZC (const Clmg< t > &img) const`

- Test if image width, height, depth and spectrum are the same as those of another image.*

 - `bool containsXYZC (const int x, const int y=0, const int z=0, const int c=0) const`
Test if specified coordinates are inside image bounds.
 - `template<typename t >`
`bool contains (const T &pixel, t &x, t &y, t &z, t &c) const`
Test if pixel value is inside image bounds and get its X,Y,Z and C-coordinates.
 - `template<typename t >`
`bool contains (const T &pixel, t &x, t &y, t &z) const`
Test if pixel value is inside image bounds and get its X,Y and Z-coordinates.
 - `template<typename t >`
`bool contains (const T &pixel, t &x, t &y) const`
Test if pixel value is inside image bounds and get its X and Y-coordinates.
 - `template<typename t >`
`bool contains (const T &pixel, t &x) const`
Test if pixel value is inside image bounds and get its X-coordinate.
 - `bool contains (const T &pixel) const`
Test if pixel value is inside image bounds.
 - `template<typename t >`
`bool is_overlapped (const Clmg< t > &img) const`
Test if pixel buffers of instance and input images overlap.
 - `template<typename tp , typename tc , typename to >`
`bool is_object3d (const ClmgList< tp > &primitives, const ClmgList< tc > &colors, const to &opacities, const bool full_check=true, char *const error_message=0) const`
*Test if the set {*this,primitives,colors,opacities} defines a valid 3D object.*
 - `bool is_Clmg3d (const bool full_check=true, char *const error_message=0) const`
Test if image instance represents a valid serialization of a 3D object.

Mathematical Functions

- `Clmg< T > & sqr ()`
Compute the square value of each pixel value.
- `Clmg< T > & sqrt ()`
Compute the square root of each pixel value.
- `Clmg< T > & exp ()`
Compute the exponential of each pixel value.
- `Clmg< T > & log ()`
Compute the error function of each pixel value.
- `Clmg< T > & log2 ()`
Compute the base-2 logarithm of each pixel value.
- `Clmg< T > & log10 ()`
Compute the base-10 logarithm of each pixel value.
- `Clmg< T > & abs ()`
Compute the absolute value of each pixel value.
- `Clmg< T > & sign ()`
Compute the sign of each pixel value.
- `Clmg< T > & cos ()`
Compute the cosine of each pixel value.
- `Clmg< T > & sin ()`
Compute the sine of each pixel value.
- `Clmg< T > & sinc ()`
Compute the sinc of each pixel value.

- `CImg< T > & tan ()`
Compute the tangent of each pixel value.
- `CImg< T > & cosh ()`
Compute the hyperbolic cosine of each pixel value.
- `CImg< T > & sinh ()`
Compute the hyperbolic sine of each pixel value.
- `CImg< T > & tanh ()`
Compute the hyperbolic tangent of each pixel value.
- `CImg< T > & acos ()`
Compute the arccosine of each pixel value.
- `CImg< T > & asin ()`
Compute the arcsine of each pixel value.
- `CImg< T > & atan ()`
Compute the arctangent of each pixel value.
- `template<typename t >`
`CImg< T > & atan2 (const CImg< t > &img)`
Compute the arctangent2 of each pixel value.
- `template<typename t >`
`CImg< Tfloat > get_atan2 (const CImg< t > &img) const`
Compute the arctangent2 of each pixel value [new-instance version].
- `CImg< T > & acosh ()`
Compute the hyperbolic arccosine of each pixel value.
- `CImg< T > & asinh ()`
Compute the hyperbolic arcsine of each pixel value.
- `CImg< T > & atanh ()`
Compute the hyperbolic arctangent of each pixel value.
- `template<typename t >`
`CImg< T > & mul (const CImg< t > &img)`
In-place pointwise multiplication.
- `template<typename t >`
`CImg< typename cimg::superset< T, t >::type > get_mul (const CImg< t > &img) const`
In-place pointwise multiplication [new-instance version].
- `template<typename t >`
`CImg< T > & div (const CImg< t > &img)`
In-place pointwise division.
- `template<typename t >`
`CImg< typename cimg::superset< T, t >::type > get_div (const CImg< t > &img) const`
In-place pointwise division [new-instance version].
- `CImg< T > & pow (const double p)`
Raise each pixel value to a specified power.
- `CImg< Tfloat > get_pow (const double p) const`
Raise each pixel value to a specified power [new-instance version].
- `CImg< T > & pow (const char *const expression)`
Raise each pixel value to a power, specified from an expression.
- `CImg< Tfloat > get_pow (const char *const expression) const`
Raise each pixel value to a power, specified from an expression [new-instance version].
- `template<typename t >`
`CImg< T > & pow (const CImg< t > &img)`
Raise each pixel value to a power, pointwisely specified from another image.
- `template<typename t >`
`CImg< Tfloat > get_pow (const CImg< t > &img) const`
Raise each pixel value to a power, pointwisely specified from another image [new-instance version].

- `Clmg< T > & rol (const unsigned int n=1)`
Compute the bitwise left rotation of each pixel value.
- `Clmg< T > get_rol (const unsigned int n=1) const`
Compute the bitwise left rotation of each pixel value [new-instance version].
- `Clmg< T > & rol (const char *const expression)`
Compute the bitwise left rotation of each pixel value.
- `Clmg< T > get_rol (const char *const expression) const`
Compute the bitwise left rotation of each pixel value [new-instance version].
- `template<typename t >`
`Clmg< T > & rol (const Clmg< t > &img)`
Compute the bitwise left rotation of each pixel value.
- `template<typename t >`
`Clmg< T > get_rol (const Clmg< t > &img) const`
Compute the bitwise left rotation of each pixel value [new-instance version].
- `Clmg< T > & ror (const unsigned int n=1)`
Compute the bitwise right rotation of each pixel value.
- `Clmg< T > get_ror (const unsigned int n=1) const`
Compute the bitwise right rotation of each pixel value [new-instance version].
- `Clmg< T > & ror (const char *const expression)`
Compute the bitwise right rotation of each pixel value.
- `Clmg< T > get_ror (const char *const expression) const`
Compute the bitwise right rotation of each pixel value [new-instance version].
- `template<typename t >`
`Clmg< T > & ror (const Clmg< t > &img)`
Compute the bitwise right rotation of each pixel value.
- `template<typename t >`
`Clmg< T > get_ror (const Clmg< t > &img) const`
Compute the bitwise right rotation of each pixel value [new-instance version].
- `Clmg< T > & softmax (const float temperature=1)`
Softmax operator.
- `Clmg< Tfloat > get_softmax (const float temperature=1) const`
Softmax operator [new-instance version].
- `Clmg< T > & softmin (const float temperature=1)`
Softmin operator.
- `Clmg< Tfloat > get_softmin (const float temperature=1) const`
Softmin operator [new-instance version].
- `Clmg< T > & min (const T &value)`
Pointwise min operator between instance image and a value.
- `Clmg< T > get_min (const T &value) const`
Pointwise min operator between instance image and a value [new-instance version].
- `template<typename t >`
`Clmg< T > & min (const Clmg< t > &img)`
Pointwise min operator between two images.
- `template<typename t >`
`Clmg< typename cimng::superset< T, t >::type > get_min (const Clmg< t > &img) const`
Pointwise min operator between two images [new-instance version].
- `Clmg< T > & min (const char *const expression)`
Pointwise min operator between an image and an expression.
- `Clmg< Tfloat > get_min (const char *const expression) const`
Pointwise min operator between an image and an expression [new-instance version].
- `Clmg< T > & max (const T &value)`

- Pointwise max operator between instance image and a value.*

 - `Clmg< T > get_max (const T &value) const`

Pointwise max operator between instance image and a value [new-instance version].
- `template<typename t >`
`Clmg< T > & max (const Clmg< t > &img)`

Pointwise max operator between two images.
- `template<typename t >`
`Clmg< typename cimg::superset< T, t >::type > get_max (const Clmg< t > &img) const`

Pointwise max operator between two images [new-instance version].
- `Clmg< T > & max (const char *const expression)`

Pointwise max operator between an image and an expression.
- `Clmg< Tfloat > get_max (const char *const expression) const`

Pointwise max operator between an image and an expression [new-instance version].
- `Clmg< T > & minabs (const T &value)`

Pointwise minabs operator between instance image and a value.
- `Clmg< T > get_minabs (const T &value) const`

Pointwise minabs operator between instance image and a value [new-instance version].
- `template<typename t >`
`Clmg< T > & minabs (const Clmg< t > &img)`

Pointwise minabs operator between two images.
- `template<typename t >`
`Clmg< typename cimg::superset< T, t >::type > get_minabs (const Clmg< t > &img) const`

Pointwise minabs operator between two images [new-instance version].
- `Clmg< T > & minabs (const char *const expression)`

Pointwise minabs operator between an image and an expression.
- `Clmg< Tfloat > get_minabs (const char *const expression) const`

Pointwise minabs operator between an image and an expression [new-instance version].
- `Clmg< T > & maxabs (const T &value)`

Pointwise maxabs operator between instance image and a value.
- `Clmg< T > get_maxabs (const T &value) const`

Pointwise maxabs operator between instance image and a value [new-instance version].
- `template<typename t >`
`Clmg< T > & maxabs (const Clmg< t > &img)`

Pointwise maxabs operator between two images.
- `template<typename t >`
`Clmg< typename cimg::superset< T, t >::type > get_maxabs (const Clmg< t > &img) const`

Pointwise maxabs operator between two images [new-instance version].
- `Clmg< T > & maxabs (const char *const expression)`

Pointwise maxabs operator between an image and an expression.
- `Clmg< Tfloat > get_maxabs (const char *const expression) const`

Pointwise maxabs operator between an image and an expression [new-instance version].
- `T & min ()`

Return a reference to the minimum pixel value.
- `const T & min () const`

Return a reference to the minimum pixel value [const version].
- `T & minabs ()`

Return a reference to the minimum pixel value in absolute value.
- `const T & minabs () const`

Return a reference to the minimum pixel value in absolute value [const version].
- `T & max ()`

Return a reference to the maximum pixel value.

- `const T & max () const`
Return a reference to the maximum pixel value [const version].
- `T & maxabs ()`
Return a reference to the maximum pixel value in absolute value.
- `const T & maxabs () const`
Return a reference to the maximum pixel value in absolute value [const version].
- `template<typename t >`
`T & min_max (t &max_val)`
Return a reference to the minimum pixel value as well as the maximum pixel value.
- `template<typename t >`
`const T & min_max (t &max_val) const`
Return a reference to the minimum pixel value as well as the maximum pixel value [const version].
- `template<typename t >`
`T & max_min (t &min_val)`
Return a reference to the maximum pixel value as well as the minimum pixel value.
- `template<typename t >`
`const T & max_min (t &min_val) const`
Return a reference to the maximum pixel value as well as the minimum pixel value [const version].
- `T kth_smallest (const ulongT k) const`
Return the kth smallest pixel value.
- `T median () const`
Return the median pixel value.
- `T gcd () const`
Return greatest common divider of all image values.
- `T lcm () const`
Return least common multiplier of all image values.
- `double product () const`
Return the product of all the pixel values.
- `double sum () const`
Return the sum of all the pixel values.
- `double mean () const`
Return the average pixel value.
- `double variance (const unsigned int variance_method=0) const`
Return the variance of the pixel values.
- `template<typename t >`
`double variance_mean (const unsigned int variance_method, t &mean) const`
Return the variance as well as the average of the pixel values.
- `double variance_noise (const unsigned int variance_method=2) const`
Return estimated variance of the noise.
- `template<typename t >`
`double MSE (const CImg< t > &img) const`
Compute the MSE (Mean-Squared Error) between two images.
- `template<typename t >`
`double PSNR (const CImg< t > &img, const double max_value=255) const`
Compute the PSNR (Peak Signal-to-Noise Ratio) between two images.
- `double eval (const char *const expression, const double x=0, const double y=0, const double z=0, const double c=0, CImgList< T > *const list_images=0)`
Evaluate math formula.
- `double eval (const char *const expression, const double x=0, const double y=0, const double z=0, const double c=0, CImgList< T > *const list_images=0) const`
Evaluate math formula [const version].

- `template<typename t >`
`void eval (Clmg< t > &output, const char *const expression, const double x=0, const double y=0, const double z=0, const double c=0, ClmgList< T > *const list_images=0)`
Evaluate math formula.
- `template<typename t >`
`void eval (Clmg< t > &output, const char *const expression, const double x=0, const double y=0, const double z=0, const double c=0, ClmgList< T > *const list_images=0) const`
Evaluate math formula [const version].
- `template<typename t >`
`Clmg< doubleT > eval (const char *const expression, const Clmg< t > &xyzc, ClmgList< T > *const list_images=0)`
Evaluate math formula on a set of variables.
- `template<typename t >`
`Clmg< doubleT > eval (const char *const expression, const Clmg< t > &xyzc, ClmgList< T > *const list_images=0) const`
Evaluate math formula on a set of variables [const version].
- `Clmg< Tdouble > get_stats (const unsigned int variance_method=0) const`
Compute statistics vector from the pixel values.
- `Clmg< T > & stats (const unsigned int variance_method=0)`
Compute statistics vector from the pixel values [in-place version].

Value Manipulation

- `Clmg< T > & fill (const T &val)`
Fill all pixel values with specified value.
- `Clmg< T > get_fill (const T &val) const`
Fill all pixel values with specified value [new-instance version].
- `Clmg< T > & fill (const T &val0, const T &val1)`
Fill sequentially all pixel values with specified values.
- `Clmg< T > get_fill (const T &val0, const T &val1) const`
Fill sequentially all pixel values with specified values [new-instance version].
- `Clmg< T > get_fill (const T &val0, const T &val1, const T &val2) const`
Fill sequentially all pixel values with specified values [new-instance version].
- `Clmg< T > get_fill (const T &val0, const T &val1, const T &val2, const T &val3) const`
Fill sequentially all pixel values with specified values [new-instance version].
- `Clmg< T > get_fill (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4) const`
Fill sequentially all pixel values with specified values [new-instance version].
- `Clmg< T > get_fill (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5) const`
Fill sequentially all pixel values with specified values [new-instance version].
- `Clmg< T > get_fill (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6) const`
Fill sequentially all pixel values with specified values [new-instance version].
- `Clmg< T > get_fill (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6, const T &val7) const`
Fill sequentially all pixel values with specified values [new-instance version].
- `Clmg< T > get_fill (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6, const T &val7, const T &val8) const`
Fill sequentially all pixel values with specified values [new-instance version].
- `Clmg< T > get_fill (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6, const T &val7, const T &val8, const T &val9) const`
Fill sequentially all pixel values with specified values [new-instance version].

- `CImg< T > get_fill (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6, const T &val7, const T &val8, const T &val9, const T &val10) const`
Fill sequentially all pixel values with specified values [new-instance version].
- `CImg< T > get_fill (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6, const T &val7, const T &val8, const T &val9, const T &val10, const T &val11) const`
Fill sequentially all pixel values with specified values [new-instance version].
- `CImg< T > get_fill (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6, const T &val7, const T &val8, const T &val9, const T &val10, const T &val11, const T &val12) const`
Fill sequentially all pixel values with specified values [new-instance version].
- `CImg< T > get_fill (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6, const T &val7, const T &val8, const T &val9, const T &val10, const T &val11, const T &val12, const T &val13) const`
Fill sequentially all pixel values with specified values [new-instance version].
- `CImg< T > get_fill (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6, const T &val7, const T &val8, const T &val9, const T &val10, const T &val11, const T &val12, const T &val13, const T &val14) const`
Fill sequentially all pixel values with specified values [new-instance version].
- `CImg< T > get_fill (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6, const T &val7, const T &val8, const T &val9, const T &val10, const T &val11, const T &val12, const T &val13, const T &val14, const T &val15) const`
Fill sequentially all pixel values with specified values [new-instance version].
- `CImg< T > & fill (const char *const expression, const bool repeat_values, const bool allow_formula=true, CImgList< T > *const list_images=0)`
Fill sequentially pixel values according to a given expression.
- `CImg< T > get_fill (const char *const expression, const bool repeat_values, const bool allow_formula=true, CImgList< T > *const list_images=0) const`
Fill sequentially pixel values according to a given expression [new-instance version].
- `CImg< T > & fill_from_values (const char *const values, const bool repeat_values)`
Fill sequentially pixel values according to a value sequence, given as a string.
- `CImg< T > get_fill_from_values (const char *const values, const bool repeat_values) const`
Fill sequentially pixel values according to a value sequence, given as a string [new-instance version].
- `template<typename t > CImg< T > & fill (const CImg< t > &values, const bool repeat_values=true)`
Fill sequentially pixel values according to the values found in another image.
- `template<typename t > CImg< T > get_fill (const CImg< t > &values, const bool repeat_values=true) const`
Fill sequentially pixel values according to the values found in another image [new-instance version].
- `CImg< T > & fillX (const unsigned int y, const unsigned int z, const unsigned int c, const int a0,...)`
Fill pixel values along the X-axis at a specified pixel position.
- `CImg< T > & fillY (const unsigned int x, const unsigned int z, const unsigned int c, const int a0,...)`
Fill pixel values along the Y-axis at a specified pixel position.
- `CImg< T > & fillZ (const unsigned int x, const unsigned int y, const unsigned int c, const int a0,...)`
Fill pixel values along the Z-axis at a specified pixel position.
- `CImg< T > & fillC (const unsigned int x, const unsigned int y, const unsigned int z, const int a0,...)`
Fill pixel values along the C-axis at a specified pixel position.
- `template<typename t > CImg< T > & discard (const CImg< t > &values, const char axis=0)`
Discard specified sequence of values in the image buffer, along a specific axis.
- `CImg< T > & discard (const char axis=0)`
Discard neighboring duplicates in the image buffer, along the specified axis.
- `CImg< T > get_discard (const char axis=0) const`
Discard neighboring duplicates in the image buffer, along the specified axis [new-instance version].

- `Clmg< T > & invert_endianness ()`
Invert endianness of all pixel values.
- `Clmg< T > get_invert_endianness () const`
Invert endianness of all pixel values [new-instance version].
- `Clmg< T > & rand (const T &val_min, const T &val_max)`
Fill image with random values in specified range.
- `Clmg< T > get_rand (const T &val_min, const T &val_max) const`
Fill image with random values in specified range [new-instance version].
- `template<typename t >`
`Clmg< T > & rand (const T &val_min, const T &val_max, const Clmg< t > &pdf, const int precision=65536)`
Fill image with random values following specified distribution and range.
- `template<typename t >`
`Clmg< T > get_rand (const T &val_min, const T &val_max, const Clmg< t > &pdf, const int precision=65536) const`
Fill image with random values following specified distribution and range [new-instance version].
- `Clmg< T > & round (const double y=1, const int rounding_type=0)`
Round pixel values.
- `Clmg< T > get_round (const double y=1, const unsigned int rounding_type=0) const`
Round pixel values [new-instance version].
- `Clmg< T > & noise (const double amplitude, const unsigned int noise_type=0)`
Add random noise to pixel values.
- `Clmg< T > get_noise (const double amplitude, const unsigned int noise_type=0) const`
Add random noise to pixel values [new-instance version].
- `Clmg< T > & normalize (const T &min_value, const T &max_value, const float constant_case_ratio=0)`
Linearly normalize pixel values.
- `Clmg< Tfloat > get_normalize (const T &min_value, const T &max_value, const float ratio_if_constant_image=0) const`
Linearly normalize pixel values [new-instance version].
- `Clmg< T > & normalize ()`
Normalize multi-valued pixels of the image instance, with respect to their L2-norm.
- `Clmg< Tfloat > get_normalize () const`
Normalize multi-valued pixels of the image instance, with respect to their L2-norm [new-instance version].
- `Clmg< T > & norm (const int norm_type=2)`
Compute Lp-norm of each multi-valued pixel of the image instance.
- `Clmg< Tfloat > get_norm (const int norm_type=2) const`
Compute L2-norm of each multi-valued pixel of the image instance [new-instance version].
- `Clmg< T > & cut (const T &min_value, const T &max_value)`
Cut pixel values in specified range.
- `Clmg< T > get_cut (const T &min_value, const T &max_value) const`
Cut pixel values in specified range [new-instance version].
- `Clmg< T > & abscut (const T &min_value, const T &max_value, const T &offset)`
Cut pixel absolute values in specified range.
- `Clmg< T > get_abscut (const T &min_value, const T &max_value, const T &offset) const`
Cut pixel values in specified range [new-instance version].
- `Clmg< T > & quantize (const unsigned int nb_levels, const bool keep_range=true)`
Uniformly quantize pixel values.
- `Clmg< T > get_quantize (const unsigned int n, const bool keep_range=true) const`
Uniformly quantize pixel values [new-instance version].
- `T otsu (const unsigned int nb_levels=256) const`
Return the Otsu threshold.
- `Clmg< T > & threshold (const T &value, const bool soft_threshold=false, const bool strict_threshold=false)`

Threshold pixel values.

- `Clmg< T > get_threshold (const T &value, const bool soft_threshold=false, const bool strict_threshold=false) const`

Threshold pixel values [new-instance version].

- `Clmg< T > & histogram (const unsigned int nb_levels, const T &min_value, const T &max_value)`

Compute the histogram of pixel values.

- `Clmg< ulongT > get_histogram (const unsigned int nb_levels, const T &min_value, const T &max_value) const`

Compute the histogram of pixel values [new-instance version].

- `Clmg< ulongT > get_histogram (const unsigned int nb_levels) const`

Compute the histogram of pixel values [new-instance version].

- `Clmg< T > & equalize (const unsigned int nb_levels, const T &min_value, const T &max_value)`

Equalize histogram of pixel values.

- `Clmg< T > get_equalize (const unsigned int nblevels, const T &val_min, const T &val_max) const`

Equalize histogram of pixel values [new-instance version].

- `Clmg< T > get_equalize (const unsigned int nblevels) const`

Equalize histogram of pixel values [new-instance version].

- `template<typename t > Clmg< T > & index (const Clmg< t > &colormap, const float dithering=1, const bool map_colors=false)`

Index multi-valued pixels regarding to a specified palette.

- `template<typename t > Clmg< typename Clmg< t >::Tuint > get_index (const Clmg< t > &colormap, const float dithering=1, const bool map_colors=true) const`

Index multi-valued pixels regarding to a specified colormap [new-instance version].

- `template<typename t > Clmg< T > & map (const Clmg< t > &palette, const unsigned int boundary_conditions=0)`

Map predefined palette on the scalar (indexed) image instance.

- `template<typename t > Clmg< t > get_map (const Clmg< t > &palette, const unsigned int boundary_conditions=0) const`

Map predefined palette on the scalar (indexed) image instance [new-instance version].

- `Clmg< T > & label (const bool is_high_connectivity=false, const Tfloat tolerance=0, const bool is_L2_norm=true)`

Label connected components.

- `Clmg< ulongT > get_label (const bool is_high_connectivity=false, const Tfloat tolerance=0, const bool is_L2_norm=true) const`

Label connected components [new-instance version].

- `template<typename t > Clmg< T > & label (const Clmg< t > &connectivity_mask, const Tfloat tolerance=0, const bool is_L2_norm=true)`

Label connected components [in-place version].

- `template<typename t > Clmg< ulongT > get_label (const Clmg< t > &connectivity_mask, const Tfloat tolerance=0, const bool is_L2_norm=true) const`

Label connected components [new-instance version].

Drawing Functions

- `template<typename tc > Clmg< T > & draw_point (const int x0, const int y0, const int z0, const tc *const color, const float opacity=1)`

Draw a 3D point.

- `template<typename t, typename tc > Clmg< T > & draw_point (const Clmg< t > &points, const tc *const color, const float opacity=1)`

- `template<typename tc >`
`Clmg< T > & draw_line (int x0, int y0, int x1, int y1, const tc *const color, const float opacity=1, const unsigned int pattern=~0U, const bool init_hatch=true, const bool draw_last_pixel=true)`
Draw a 2D line.
- `template<typename tz , typename tc >`
`Clmg< T > & draw_line (Clmg< tz > &zbuffer, int x0, int y0, const float z0, int x1, int y1, const float z1, const tc *const color, const float opacity=1, const unsigned int pattern=~0U, const bool init_hatch=true)`
Draw a 2D line, with z-buffering.
- `template<typename tc >`
`Clmg< T > & draw_line (int x0, int y0, int x1, int y1, const Clmg< tc > &texture, int tx0, int ty0, int tx1, int ty1, const float opacity=1, const unsigned int pattern=~0U, const bool init_hatch=true)`
Draw a textured 2D line.
- `template<typename tc >`
`Clmg< T > & draw_line (int x0, int y0, const float z0, int x1, int y1, const float z1, const Clmg< tc > &texture, const int tx0, const int ty0, const int tx1, const int ty1, const float opacity=1, const unsigned int pattern=~0U, const bool init_hatch=true)`
Draw a textured 2D line, with perspective correction.
- `template<typename tz , typename tc >`
`Clmg< T > & draw_line (Clmg< tz > &zbuffer, int x0, int y0, const float z0, int x1, int y1, const float z1, const Clmg< tc > &texture, const int tx0, const int ty0, const int tx1, const int ty1, const float opacity=1, const unsigned int pattern=~0U, const bool init_hatch=true)`
Draw a textured 2D line, with perspective correction and z-buffering.
- `template<typename tp , typename tc >`
`Clmg< T > & draw_line (const Clmg< tp > &points, const tc *const color, const float opacity=1, const unsigned int pattern=~0U, const bool init_hatch=true)`
Draw a polyline.
- `template<typename tc >`
`Clmg< T > & draw_arrow (const int x0, const int y0, const int x1, const int y1, const tc *const color, const float opacity=1, const float angle=30, const float length=-10, const unsigned int pattern=~0U)`
Draw a 2D arrow.
- `template<typename tc >`
`Clmg< T > & draw_spline (const int x0, const int y0, const float u0, const float v0, const int x1, const int y1, const float u1, const float v1, const tc *const color, const float opacity=1, const float precision=0.25f, const unsigned int pattern=~0U, const bool init_hatch=true)`
Draw a 2D spline.
- `template<typename t >`
`Clmg< T > & draw_spline (const int x0, const int y0, const float u0, const float v0, const int x1, const int y1, const float u1, const float v1, const Clmg< t > &texture, const int tx0, const int ty0, const int tx1, const int ty1, const float opacity=1, const float precision=4, const unsigned int pattern=~0U, const bool init_hatch=true)`
Draw a textured 2D spline.
- `template<typename tp , typename tt , typename tc >`
`Clmg< T > & draw_spline (const Clmg< tp > &points, const Clmg< tt > &tangents, const tc *const color, const float opacity=1, const bool is_closed_set=false, const float precision=4, const unsigned int pattern=~0U, const bool init_hatch=true)`
Draw a set of consecutive splines.
- `template<typename tp , typename tc >`
`Clmg< T > & draw_spline (const Clmg< tp > &points, const tc *const color, const float opacity=1, const bool is_closed_set=false, const float precision=4, const unsigned int pattern=~0U, const bool init_hatch=true)`
- `template<typename tc >`
`Clmg< T > & draw_triangle (const int x0, const int y0, const int x1, const int y1, const int x2, const int y2, const tc *const color, const float opacity=1)`
Draw a filled 2D triangle.
- `template<typename tc >`
`Clmg< T > & draw_triangle (const int x0, const int y0, const int x1, const int y1, const int x2, const int y2, const tc *const color, const float opacity, const unsigned int pattern)`

Draw a outlined 2D triangle.

- `template<typename tz , typename tc >`
`Clmg< T > & draw_triangle (Clmg< tz > &zbuffer, int x0, int y0, const float z0, int x1, int y1, const float z1,`
`int x2, int y2, const float z2, const tc *const color, const float opacity=1, const float brightness=1)`

Draw a filled 2D triangle, with z-buffering.

- `template<typename tc >`
`Clmg< T > & draw_triangle (int x0, int y0, int x1, int y1, int x2, int y2, const tc *const color, float bs0, float`
`bs1, float bs2, const float opacity=1)`

Draw a Gouraud-shaded 2D triangle.

- `template<typename tc >`
`Clmg< T > & draw_triangle (int x0, int y0, int x1, int y1, int x2, int y2, const tc *color0, const tc *color1, const`
`tc *color2, const float opacity=1)`

Draw a color-interpolated 2D triangle.

- `template<typename tc >`
`Clmg< T > & draw_triangle (int x0, int y0, int x1, int y1, int x2, int y2, const Clmg< tc > &texture, int tx0, int`
`ty0, int tx1, int ty1, int tx2, int ty2, const float opacity=1, const float brightness=1)`

Draw a textured 2D triangle.

- `template<typename tc >`
`Clmg< T > & draw_triangle (int x0, int y0, const float z0, int x1, int y1, const float z1, int x2, int y2, const`
`float z2, const Clmg< tc > &texture, int tx0, int ty0, int tx1, int ty1, int tx2, int ty2, const float opacity=1, const`
`float brightness=1)`

Draw a 2D textured triangle, with perspective correction.

- `template<typename tz , typename tc >`
`Clmg< T > & draw_triangle (Clmg< tz > &zbuffer, int x0, int y0, const float z0, int x1, int y1, const float z1,`
`int x2, int y2, const float z2, const Clmg< tc > &texture, int tx0, int ty0, int tx1, int ty1, int tx2, int ty2, const`
`float opacity=1, const float brightness=1)`

Draw a textured 2D triangle, with perspective correction and z-buffering.

- `template<typename tc , typename tl >`
`Clmg< T > & draw_triangle (int x0, int y0, int x1, int y1, int x2, int y2, const tc *const color, const Clmg< tl`
`> &light, int lx0, int ly0, int lx1, int ly1, int lx2, int ly2, const float opacity=1)`

Draw a Phong-shaded 2D triangle.

- `template<typename tz , typename tc , typename tl >`
`Clmg< T > & draw_triangle (Clmg< tz > &zbuffer, int x0, int y0, const float z0, int x1, int y1, const float z1,`
`int x2, int y2, const float z2, const tc *const color, const Clmg< tl > &light, int lx0, int ly0, int lx1, int ly1, int`
`lx2, int ly2, const float opacity=1)`

Draw a Phong-shaded 2D triangle, with z-buffering.

- `template<typename tc >`
`Clmg< T > & draw_triangle (int x0, int y0, int x1, int y1, int x2, int y2, const Clmg< tc > &texture, int tx0, int`
`ty0, int tx1, int ty1, int tx2, int ty2, float bs0, float bs1, float bs2, const float opacity=1)`

Draw a textured Gouraud-shaded 2D triangle.

- `template<typename tc , typename tl >`
`Clmg< T > & draw_triangle (int x0, int y0, int x1, int y1, int x2, int y2, const Clmg< tc > &texture, int tx0, int`
`ty0, int tx1, int ty1, int tx2, int ty2, const Clmg< tl > &light, int lx0, int ly0, int lx1, int ly1, int lx2, int ly2, const`
`float opacity=1)`

Draw a textured Phong-shaded 2D triangle.

- `template<typename tc , typename tl >`
`Clmg< T > & draw_triangle (int x0, int y0, const float z0, int x1, int y1, const float z1, int x2, int y2, const`
`float z2, const Clmg< tc > &texture, int tx0, int ty0, int tx1, int ty1, int tx2, int ty2, const Clmg< tl > &light, int`
`lx0, int ly0, int lx1, int ly1, int lx2, int ly2, const float opacity=1)`

Draw a textured Phong-shaded 2D triangle, with perspective correction.

- `template<typename tz , typename tc , typename tl >`
`Clmg< T > & draw_triangle (Clmg< tz > &zbuffer, int x0, int y0, const float z0, int x1, int y1, const float z1,`
`int x2, int y2, const float z2, const Clmg< tc > &texture, int tx0, int ty0, int tx1, int ty1, int tx2, int ty2, const`
`Clmg< tl > &light, int lx0, int ly0, int lx1, int ly1, int lx2, int ly2, const float opacity=1)`

Draw a textured Phong-shaded 2D triangle, with perspective correction and z-buffering.

- `Clmg< T > & draw_rectangle (const int x0, const int y0, const int z0, const int c0, const int x1, const int y1, const int z1, const int c1, const T val, const float opacity=1)`
Draw a filled 4D rectangle.
- `template<typename tc >`
`Clmg< T > & draw_rectangle (const int x0, const int y0, const int z0, const int x1, const int y1, const int z1, const tc *const color, const float opacity=1)`
Draw a filled 3D rectangle.
- `template<typename tc >`
`Clmg< T > & draw_rectangle (const int x0, const int y0, const int x1, const int y1, const tc *const color, const float opacity=1)`
Draw a filled 2D rectangle.
- `template<typename tp , typename tc >`
`Clmg< T > & draw_polygon (const Clmg< tp > &points, const tc *const color, const float opacity=1)`
Draw a filled 2D polygon.
- `template<typename tc >`
`Clmg< T > & draw_ellipse (const int x0, const int y0, const float r1, const float r2, const float angle, const tc *const color, const float opacity=1)`
Draw a filled 2D ellipse.
- `template<typename t , typename tc >`
`Clmg< T > & draw_ellipse (const int x0, const int y0, const Clmg< t > &tensor, const tc *const color, const float opacity=1)`
- `template<typename tc >`
`Clmg< T > & draw_ellipse (const int x0, const int y0, const float r1, const float r2, const float angle, const tc *const color, const float opacity, const unsigned int pattern)`
Draw an outlined 2D ellipse.
- `template<typename t , typename tc >`
`Clmg< T > & draw_ellipse (const int x0, const int y0, const Clmg< t > &tensor, const tc *const color, const float opacity, const unsigned int pattern)`
- `template<typename tc >`
`Clmg< T > & draw_circle (const int x0, const int y0, int radius, const tc *const color, const float opacity=1)`
Draw a filled 2D circle.
- `template<typename tc >`
`Clmg< T > & draw_circle (const int x0, const int y0, int radius, const tc *const color, const float opacity, const unsigned int pattern)`
Draw an outlined 2D circle.
- `template<typename t >`
`Clmg< T > & draw_image (const int x0, const int y0, const int z0, const int c0, const Clmg< t > &sprite, const float opacity=1)`
Draw an image.
- `Clmg< T > & draw_image (const int x0, const int y0, const int z0, const int c0, const Clmg< T > &sprite, const float opacity=1)`
Draw an image [specialization].
- `template<typename ti , typename tm >`
`Clmg< T > & draw_image (const int x0, const int y0, const int z0, const int c0, const Clmg< ti > &sprite, const Clmg< tm > &mask, const float opacity=1, const float mask_max_value=1)`
Draw a masked image.
- `template<typename ti , typename tm >`
`Clmg< T > & draw_image (const Clmg< ti > &sprite, const Clmg< tm > &mask, const float opacity=1, const float mask_max_value=1)`
Draw an image.
- `template<typename tc1 , typename tc2 , typename t >`
`Clmg< T > & draw_text (const int x0, const int y0, const char *const text, const tc1 *const foreground_color, const tc2 *const background_color, const float opacity, const ClmgList< t > *const font,...)`
Draw a text string.

- `template<typename tc, typename t >`
`Clmg< T > & draw_text (const int x0, const int y0, const char *const text, const tc *const foreground_color,`
`const int, const float opacity, const ClmgList< t > *const font,...)`
- `template<typename tc, typename t >`
`Clmg< T > & draw_text (const int x0, const int y0, const char *const text, const int, const tc *const`
`background_color, const float opacity, const ClmgList< t > *const font,...)`
- `template<typename tc1, typename tc2 >`
`Clmg< T > & draw_text (const int x0, const int y0, const char *const text, const tc1 *const foreground_color,`
`const tc2 *const background_color, const float opacity=1, const unsigned int font_height=13,...)`
- `template<typename t1, typename t2 >`
`Clmg< T > & draw_quiver (const Clmg< t1 > &flow, const t2 *const color, const float opacity=1, const`
`unsigned int sampling=25, const float factor=-20, const bool is_arrow=true, const unsigned int pattern=~0U)`
Draw a 2D vector field.
- `template<typename t1, typename t2 >`
`Clmg< T > & draw_quiver (const Clmg< t1 > &flow, const Clmg< t2 > &color, const float opacity=1, const`
`unsigned int sampling=25, const float factor=-20, const bool is_arrow=true, const unsigned int pattern=~0U)`
Draw a 2D vector field using a field of colors.
- `template<typename t, typename tc >`
`Clmg< T > & draw_axis (const Clmg< t > &values_x, const int y, const tc *const color, const float opacity=1,`
`const unsigned int pattern=~0U, const unsigned int font_height=13, const bool allow_zero=true, const float`
`round_x=0)`
Draw a labeled horizontal axis.
- `template<typename t, typename tc >`
`Clmg< T > & draw_axis (const int x, const Clmg< t > &values_y, const tc *const color, const float opacity=1,`
`const unsigned int pattern=~0U, const unsigned int font_height=13, const bool allow_zero=true, const float`
`round_y=0)`
Draw a labeled vertical axis.
- `template<typename tx, typename ty, typename tc >`
`Clmg< T > & draw_axes (const Clmg< tx > &values_x, const Clmg< ty > &values_y, const tc *const color,`
`const float opacity=1, const unsigned int pattern_x=~0U, const unsigned int pattern_y=~0U, const unsigned`
`int font_height=13, const bool allow_zero=true, const float round_x=0, const float round_y=0)`
Draw labeled horizontal and vertical axes.
- `template<typename tx, typename ty, typename tc >`
`Clmg< T > & draw_grid (const Clmg< tx > &values_x, const Clmg< ty > &values_y, const tc *const color,`
`const float opacity=1, const unsigned int pattern_x=~0U, const unsigned int pattern_y=~0U)`
Draw a 2D grid.
- `template<typename t, typename tc >`
`Clmg< T > & draw_graph (const Clmg< t > &data, const tc *const color, const float opacity=1, const`
`unsigned int plot_type=1, const int vertex_type=1, const double ymin=0, const double ymax=0, const`
`unsigned int pattern=~0U)`
Draw a 1D graph.
- `template<typename tc, typename t >`
`Clmg< T > & draw_fill (const int x0, const int y0, const int z0, const tc *const color, const float opacity,`
`Clmg< t > &region, const float tolerance=0, const bool is_high_connectivity=false)`
Fill a 3D region using the flood-fill algorithm.
- `Clmg< T > & draw_plasma (const float alpha=1, const float beta=0, const unsigned int scale=8)`
Draw a random plasma texture.
- `template<typename tc >`
`Clmg< T > & draw_mandelbrot (const int x0, const int y0, const int x1, const int y1, const Clmg< tc`
`> &colormap, const float opacity=1, const double z0r=-2, const double z0i=-2, const double z1r=2, const`
`double z1i=2, const unsigned int iteration_max=255, const bool is_normalized_iteration=false, const bool`
`is_julia_set=false, const double param_r=0, const double param_i=0)`
Draw a quadratic Mandelbrot or Julia 2D fractal.
- `template<typename tc >`
`Clmg< T > & draw_gaussian (const float xc, const float sigma, const tc *const color, const float opacity=1)`

Draw a 1D gaussian function.

- `template<typename t , typename tc >`
`CImg< T > & draw_gaussian (const float xc, const float yc, const CImg< t > &tensor, const tc *const color,`
`const float opacity=1)`

Draw a 2D gaussian function.

- `template<typename tp , typename tf , typename tc , typename to >`
`CImg< T > & draw_object3d (const float x0, const float y0, const float z0, const CImg< tp > &vertices, const`
`CImgList< tf > &primitives, const CImgList< tc > &colors, const CImg< to > &opacities, const unsigned`
`int render_type=4, const bool is_double_sided=false, const float focal=700, const float lightx=0, const float`
`lighty=0, const float lightz=-5e8, const float specular_lightness=0.25f, const float specular_shininess=0.1f,`
`const float g_opacity=1, const bool is_multithreaded_rendering=false)`

Draw a 3D object.

8.1.1 Detailed Description

```
template<typename T>
struct cimg_library::CImg< T >
```

Class representing an image (up to 4 dimensions wide), where each pixel is of type T.

This is the main class of the CImg Library. It declares and constructs an image, allows access to its pixel values, and is able to perform various image operations.

Image representation

A CImg image is defined as an instance of the container CImg<T>, which contains a regular grid of pixels, each pixel value being of type T. The image grid can have up to 4 dimensions: width, height, depth and number of channels. Usually, the first three dimensions are used to describe spatial coordinates (x, y, z), while the number of channels is rather used as a vector-valued dimension (it may describe the R,G,B color channels for instance). If you need a fifth dimension, you can use image lists CImgList<T> rather than simple images CImg<T>.

Thus, the CImg<T> class is able to represent volumetric images of vector-valued pixels, as well as images with less dimensions (1D scalar signal, 2D color images, ...). Most member functions of the class CImg<T> are designed to handle this maximum case of (3+1) dimensions.

Concerning the pixel value type T: fully supported template types are the basic C++ types: unsigned char, char, short, unsigned int, int, unsigned long, long, float, double, ... Typically, fast image display can be done using CImg<unsigned char> images, while complex image processing algorithms may be rather coded using CImg<float> or CImg<double> images that have floating-point pixel values. The default value for the template T is float. Using your own template types may be possible. However, you will certainly have to define the complete set of arithmetic and logical operators for your class.

Image structure

The CImg<T> structure contains *six* fields:

- `_width` defines the number of *columns* of the image (size along the X-axis).

- `_height` defines the number of *rows* of the image (size along the Y-axis).
- `_depth` defines the number of *slices* of the image (size along the Z-axis).
- `_spectrum` defines the number of *channels* of the image (size along the C-axis).
- `_data` defines a *pointer* to the *pixel data* (of type `T`).
- `_is_shared` is a boolean indicating that the memory buffer `data` is shared with another image.

You can access these fields publicly although it is recommended to use the dedicated functions `width()`, `height()`, `depth()`, `spectrum()` and `ptr()` to do so. Image dimensions are not limited to a specific range (as long as you have enough available memory). A value of `1` usually means that the corresponding dimension is *flat*. If one of the dimensions is `0`, or if the data pointer is null, the image is considered as *empty*. Empty images should not contain any pixel data and thus, will not be processed by `CImg` member functions (a `CImgInstanceException` will be thrown instead). Pixel data is stored in memory, in a non-interleaved mode (See [How pixel data are stored with CImg.](#)).

Image declaration and construction

Declaring an image can be done by using one of the several available constructors. Here is a list of the most used:

- Construct images from arbitrary dimensions:
 - `CImg<char> img;` declares an empty image.
 - `CImg<unsigned char> img(128,128);` declares a 128x128 greyscale image with unsigned char pixel values.
 - `CImg<double> img(3,3);` declares a 3x3 matrix with double coefficients.
 - `CImg<unsigned char> img(256,256,1,3);` declares a 256x256x1x3 (color) image (colors are stored as an image with three channels).
 - `CImg<double> img(128,128,128);` declares a 128x128x128 volumetric and greyscale image (with double pixel values).
 - `CImg<> img(128,128,128,3);` declares a 128x128x128 volumetric color image (with float pixels, which is the default value of the template parameter `T`).
 - **Note:** images pixels are **not automatically initialized to 0**. You may use the function `fill()` to do it, or use the specific constructor taking 5 parameters like this: `CImg<> img(128,128,128,3,0);` declares a 128x128x128 volumetric color image with all pixel values to 0.
- Construct images from filenames:
 - `CImg<unsigned char> img("image.jpg");` reads a JPEG color image from the file "image.jpg".
 - `CImg<float> img("analyze.hdr");` reads a volumetric image (ANALYZE7.5 format) from the file "analyze.hdr".
 - **Note:** You need to install `ImageMagick` to be able to read common compressed image formats (JPG,PNG, ...) (See [Files IO in CImg.](#)).
- Construct images from C-style arrays:
 - `CImg<int> img(data_buffer,256,256);` constructs a 256x256 greyscale image from a `int*` buffer `data_buffer` (of size 256x256=65536).
 - `CImg<unsigned char> img(data_buffer,256,256,1,3);` constructs a 256x256 color image from a `unsigned char*` buffer `data_buffer` (where R,G,B channels follow each others).

The complete list of constructors can be found [here](#).

Most useful functions

The CImg<T> class contains a lot of functions that operate on images. Some of the most useful are:

- `operator()()`: Read or write pixel values.
- `display()`: Displays the image in a new window.

8.1.2 Member Typedef Documentation

8.1.2.1 iterator

```
template<typename T >
typedef T* iterator
```

Simple iterator type to loop through each pixel value of an image instance.

Note

- The `CImg<T>::iterator` type is defined to be a `T*`.
- You will seldom have to use iterators in CImg, most classical operations being achieved (often in a faster way) using functions of CImg<T>.

Example

```
CImg<float> img("reference.jpg"); // Load image from file
// Set all pixels to '0', with a CImg iterator.
for (CImg<float>::iterator it = img.begin(); it<img.end(); ++it) *it = 0;
img.fill(0); // Do the same with a built-in function
```

8.1.2.2 const_iterator

```
template<typename T >
typedef const T* const_iterator
```

Simple const iterator type, to loop through each pixel value of a `const` image instance.

Note

- The `CImg<T>::const_iterator` type is defined to be a `const T*`.
- You will seldom have to use iterators in CImg, most classical operations being achieved (often in a faster way) using functions of CImg<T>.

Example

```
const CImg<float> img("reference.jpg"); // Load image from file
float sum = 0;
// Compute sum of all pixel values, with a CImg iterator.
for (CImg<float>::const_iterator it = img.begin(); it<img.end(); ++it) sum+=*it;
const float sum2 = img.sum(); // Do the same with a built-in function
```

8.1.2.3 value_type

```
template<typename T >
typedef T value_type
```

Pixel value type.

Refer to the type of the pixel values of an image instance.

Note

- The `CImg<T>::value_type` type of a `CImg<T>` is defined to be a `T`.
- `CImg<T>::value_type` is actually not used in `CImg` functions. It has been mainly defined for compatibility with STL naming conventions.

8.1.3 Constructor & Destructor Documentation

8.1.3.1 ~CImg()

```
template<typename T >
~CImg ( )
```

Destroy image.

Note

- The pixel buffer `data()` is deallocated if necessary, e.g. for non-empty and non-shared image instances.
- Destroying an empty or shared image does nothing actually.

Warning

- When destroying a non-shared image, make sure that you will *not* operate on a remaining shared image that shares its buffer with the destroyed instance, in order to avoid further invalid memory access (to a deallocated buffer).

8.1.3.2 CImg() [1/13]

```
template<typename T >
CImg ( )
```

Construct empty image.

Note

- An empty image has no pixel data and all of its dimensions `width()`, `height()`, `depth()`, `spectrum()` are set to 0, as well as its pixel buffer pointer `data()`.
- An empty image may be re-assigned afterwards, e.g. with the family of `assign(unsigned int,unsigned int,unsigned int,unsigned int)` functions, or by `operator=(const CImg<t>&)`. In all cases, the type of pixels stays `T`.
- An empty image is never shared.

Example

```
CImg<float> img1, img2; // Construct two empty images
img1.assign(256,256,1,3); // Re-assign 'img1' to be a 256x256x1x3 (color) image
img2 = img1.get_rand(0,255); // Re-assign 'img2' to be a random-valued version of 'img1'
img2.assign(); // Re-assign 'img2' to be an empty image again
```

8.1.3.3 CImg() [2/13]

```
template<typename T >
CImg (
    const unsigned int size_x,
    const unsigned int size_y = 1,
    const unsigned int size_z = 1,
    const unsigned int size_c = 1 ) [explicit]
```

Construct image with specified size.

Parameters

<i>size</i> ↔ _x	Image width() .
<i>size</i> ↔ _y	Image height() .
<i>size</i> ↔ _z	Image depth() .
<i>size</i> ↔ _c	Image spectrum() (number of channels).

Note

- It can only create *non-shared* images, and allocates thus a pixel buffer [data\(\)](#) for each constructed image instance.
- Setting one dimension `size_x`, `size_y`, `size_z` or `size_c` to 0 leads to the construction of an *empty* image.
- A `CImgInstanceException` is thrown when the pixel buffer cannot be allocated (e.g. when requested size is too big for available memory).

Warning

- The allocated pixel buffer is *not* filled with a default value, and is likely to contain garbage values. In order to initialize pixel values during construction (e.g. with 0), use constructor `CImg(unsigned int,unsigned int,unsigned int,unsigned int,T)` instead.

Example

```
CImg<float> img1(256,256,1,3); // Construct a 256x256x1x3 (color) image, filled with garbage values
CImg<float> img2(256,256,1,3,0); // Construct a 256x256x1x3 (color) image, filled with value '0'
```

8.1.3.4 CImg() [3/13]

```
template<typename T >
CImg (
    const unsigned int size_x,
    const unsigned int size_y,
    const unsigned int size_z,
    const unsigned int size_c,
    const T & value )
```

Construct image with specified size and initialize pixel values.

Parameters

<i>size</i> ↔ _x	Image width() .
<i>size</i> ↔ _y	Image height() .
<i>size</i> ↔ _z	Image depth() .
<i>size</i> ↔ _c	Image spectrum() (number of channels).
<i>value</i>	Initialization value.

Note

- Similar to [CImg\(unsigned int,unsigned int,unsigned int,unsigned int\)](#), but it also fills the pixel buffer with the specified *value*.

Warning

- It cannot be used to construct a vector-valued image and initialize it with *vector-valued* pixels (e.g. RGB vector, for color images). For this task, you may use [fillC\(\)](#) after construction.

8.1.3.5 CImg() [4/13]

```
template<typename T >
CImg (
    const unsigned int size_x,
    const unsigned int size_y,
    const unsigned int size_z,
    const unsigned int size_c,
    const int value0,
    const int value1,
    ... )
```

Construct image with specified size and initialize pixel values from a sequence of integers.

Construct a new image instance of size *size_x* x *size_y* x *size_z* x *size_c*, with pixels of type *T*, and initialize pixel values from the specified sequence of integers *value0,value1,...*

Parameters

<i>size</i> ↔ _x	Image width() .
<i>size</i> ↔ _y	Image height() .
<i>size</i> ↔ _z	Image depth() .
<i>size</i> ↔ _c	Image spectrum() (number of channels).
<i>value0</i>	First value of the initialization sequence (must be an <i>integer</i>).
<i>value1</i>	Second value of the initialization sequence (must be an <i>integer</i>).
...	

Note

- Similar to `CImg(unsigned int,unsigned int,unsigned int,unsigned int)`, but it also fills the pixel buffer with a sequence of specified integer values.

Warning

- You must specify *exactly* `size_x*size_y*size_z*size_c` integers in the initialization sequence. Otherwise, the constructor may crash or fill your image pixels with garbage.

Example

```
const CImg<float> img(2,2,1,3, // Construct a 2x2 color (RGB) image
                    0,255,0,255, // Set the 4 values for the red component
                    0,0,255,255, // Set the 4 values for the green component
                    64,64,64,64); // Set the 4 values for the blue component
img.resize(150,150).display();
```

8.1.3.6 CImg() [5/13]

```
template<typename T >
CImg (
    const unsigned int size_x,
    const unsigned int size_y,
    const unsigned int size_z,
    const unsigned int size_c,
    const double value0,
    const double value1,
    ... )
```

Construct image with specified size and initialize pixel values from a sequence of doubles.

Construct a new image instance of size `size_x x size_y x size_z x size_c`, with pixels of type `T`, and initialize pixel values from the specified sequence of doubles `value0,value1,...`

Parameters

<code>size_x</code>	Image <code>width()</code> .
<code>size_y</code>	Image <code>height()</code> .
<code>size_z</code>	Image <code>depth()</code> .
<code>size_c</code>	Image <code>spectrum()</code> (number of channels).
<code>value0</code>	First value of the initialization sequence (must be a <i>double</i>).
<code>value1</code>	Second value of the initialization sequence (must be a <i>double</i>).
...	

Note

- Similar to `CImg(unsigned int,unsigned int,unsigned int,unsigned int,int,int,...)`, but takes a sequence of double values instead of integers.

Warning

- You must specify *exactly* `dx*dy*dz*dc` doubles in the initialization sequence. Otherwise, the constructor may crash or fill your image with garbage. For instance, the code below will probably crash on most platforms:

```
const CImg<float> img(2,2,1,1, 0.5,0.5,255,255); // FAIL: The two last arguments are 'int', not
'double'!
```

8.1.3.7 CImg() [6/13]

```
template<typename T >
CImg (
    const unsigned int size_x,
    const unsigned int size_y,
    const unsigned int size_z,
    const unsigned int size_c,
    const char *const values,
    const bool repeat_values )
```

Construct image with specified size and initialize pixel values from a value string.

Construct a new image instance of size `size_x` x `size_y` x `size_z` x `size_c`, with pixels of type `T`, and initializes pixel values from the specified string `values`.

Parameters

<code>size_x</code>	Image <code>width()</code> .
<code>size_y</code>	Image <code>height()</code> .
<code>size_z</code>	Image <code>depth()</code> .
<code>size_c</code>	Image <code>spectrum()</code> (number of channels).
<code>values</code>	Value string describing the way pixel values are set.
<code>repeat_values</code>	Indicates whether the value filling process is repeated over the image.

Note

- Similar to `CImg(unsigned int,unsigned int,unsigned int,unsigned int)`, but it also fills the pixel buffer with values described in the value string `values`.
- Value string `values` may describe two different filling processes:
 - Either `values` is a sequences of values assigned to the image pixels, as in `"1,2,3,7,8,2"`. In this case, set `repeat_values` to `true` to periodically fill the image with the value sequence.
 - Either, `values` is a formula, as in `"cos(x/10)*sin(y/20)"`. In this case, parameter `repeat_values` is pointless.
- For both cases, specifying `repeat_values` is mandatory. It disambiguates the possible overloading of constructor `CImg(unsigned int,unsigned int,unsigned int,unsigned int,T)` with `T` being a `const char*`.
- A `CImgArgumentException` is thrown when an invalid value string `values` is specified.

Example

```
const CImg<float> img1(129,129,1,3,"0,64,128,192,255",true), // Construct image from a value sequence
img2(129,129,1,3,"if(c==0,255*abs(cos(x/10)),1.8*y)",false); // Construct image from
a formula
(img1,img2).display();
```

8.1.3.8 CImg() [7/13]

```
template<typename T >
template<typename t >
CImg (
    const t *const values,
    const unsigned int size_x,
    const unsigned int size_y = 1,
    const unsigned int size_z = 1,
    const unsigned int size_c = 1,
    const bool is_shared = false )
```

Construct image with specified size and initialize pixel values from a memory buffer.

Construct a new image instance of size `size_x` x `size_y` x `size_z` x `size_c`, with pixels of type `T`, and initializes pixel values from the specified `t*` memory buffer.

Parameters

<i>values</i>	Pointer to the input memory buffer.
<i>size_x</i>	Image width() .
<i>size_y</i>	Image height() .
<i>size_z</i>	Image depth() .
<i>size_c</i>	Image spectrum() (number of channels).
<i>is_shared</i>	Indicates whether input memory buffer must be shared by the current instance.

Note

- If `is_shared` is `false`, the image instance allocates its own pixel buffer, and values from the specified input buffer are copied to the instance buffer. If buffer types `T` and `t` are different, a regular static cast is performed during buffer copy.
- Otherwise, the image instance does *not* allocate a new buffer, and uses the input memory buffer as its own pixel buffer. This case requires that types `T` and `t` are the same. Later, destroying such a shared image will not deallocate the pixel buffer, this task being obviously charged to the initial buffer allocator.
- A `CImgInstanceException` is thrown when the pixel buffer cannot be allocated (e.g. when requested size is too big for available memory).

Warning

- You must take care when operating on a shared image, since it may have an invalid pixel buffer pointer [data\(\)](#) (e.g. already deallocated).

Example

```
unsigned char tab[256*256] = {};
CImg<unsigned char> img1(tab,256,256,1,1,false), // Construct new non-shared image from buffer 'tab'
                  img2(tab,256,256,1,1,true); // Construct new shared-image from buffer 'tab'
tab[1024] = 255; // Here, 'img2' is indirectly modified, but not 'img1'
```

8.1.3.9 CImg() [8/13]

```
template<typename T >
CImg (
    const char *const filename ) [explicit]
```

Construct image from reading an image file.

Construct a new image instance with pixels of type `T`, and initialize pixel values with the data read from an image file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

Note

- Similar to `CImg(unsigned int,unsigned int,unsigned int,unsigned int)`, but it reads the image dimensions and pixel values from the specified image file.
- The recognition of the image file format by `CImg` highly depends on the tools installed on your system and on the external libraries you used to link your code against.
- Considered pixel type `T` should better fit the file format specification, or data loss may occur during file load (e.g. constructing a `CImg<unsigned char>` from a float-valued image file).
- A `CImgIOException` is thrown when the specified `filename` cannot be read, or if the file format is not recognized.

Example

```
const CImg<float> img("reference.jpg");
img.display();
```

8.1.3.10 CImg() [9/13]

```
template<typename T >
template<typename t >
CImg (
    const CImg< t > & img )
```

Construct image copy.

Construct a new image instance with pixels of type `T`, as a copy of an existing `CImg<t>` instance.

Parameters

<i>img</i>	Input image to copy.
------------	----------------------

Note

- Constructed copy has the same size `width()` x `height()` x `depth()` x `spectrum()` and pixel values as the input image `img`.

- If input image `img` is *shared* and if types `T` and `t` are the same, the constructed copy is also *shared*, and shares its pixel buffer with `img`. Modifying a pixel value in the constructed copy will thus also modifies it in the input image `img`. This behavior is needful to allow functions to return shared images.
- Otherwise, the constructed copy allocates its own pixel buffer, and copies pixel values from the input image `img` into its buffer. The copied pixel values may be eventually statically casted if types `T` and `t` are different.
- Constructing a copy from an image `img` when types `t` and `T` are the same is significantly faster than with different types.
- A `CImgInstanceException` is thrown when the pixel buffer cannot be allocated (e.g. not enough available memory).

8.1.3.11 CImg() [10/13]

```
template<typename T >
template<typename t >
CImg (
    const CImg< t > & img,
    const bool is_shared )
```

Advanced copy constructor.

Construct a new image instance with pixels of type `T`, as a copy of an existing `CImg<t>` instance, while forcing the shared state of the constructed copy.

Parameters

<code>img</code>	Input image to copy.
<code>is_shared</code>	Indicates the shared state of the constructed copy.

Note

- Similar to `CImg(const CImg<t>&)`, except that it allows users to decide the shared state of the constructed image, which does not depend anymore on the shared state of the input image `img`:
 - If `is_shared` is `true`, the constructed copy will share its pixel buffer with the input image `img`. For that case, the pixel types `T` and `t` *must* be the same.
 - If `is_shared` is `false`, the constructed copy will allocate its own pixel buffer, whether the input image `img` is shared or not.
- A `CImgArgumentException` is thrown when a shared copy is requested with different pixel types `T` and `t`.

8.1.3.12 CImg() [11/13]

```
template<typename T >
template<typename t >
CImg (
    const CImg< t > & img,
    const char *const dimensions )
```

Construct image with dimensions borrowed from another image.

Construct a new image instance with pixels of type `T`, and size get from some dimensions of an existing `CImg<t>` instance.

Parameters

<i>img</i>	Input image from which dimensions are borrowed.
<i>dimensions</i>	C-string describing the image size along the X,Y,Z and C-dimensions.

Note

- Similar to [CImg\(unsigned int,unsigned int,unsigned int,unsigned int\)](#), but it takes the image dimensions (*not* its pixel values) from an existing `CImg<t>` instance.
- The allocated pixel buffer is *not* filled with a default value, and is likely to contain garbage values. In order to initialize pixel values (e.g. with 0), use constructor `CImg(const CImg<t>&,const char*,T)` instead.

Example

```
const CImg<float> img1(256,128,1,3), // 'img1' is a 256x128x1x3 image
img2(img1,"xyzc"), // 'img2' is a 256x128x1x3 image
img3(img1,"y,x,z,c"), // 'img3' is a 128x256x1x3 image
img4(img1,"c,x,y,3",0), // 'img4' is a 3x128x256x3 image (with pixels initialized to
'0')
```

8.1.3.13 CImg() [12/13]

```
template<typename T >
template<typename t >
CImg (
    const CImg< t > & img,
    const char *const dimensions,
    const T & value )
```

Construct image with dimensions borrowed from another image and initialize pixel values.

Construct a new image instance with pixels of type `T`, and size get from the dimensions of an existing `CImg<t>` instance, and set all pixel values to specified `value`.

Parameters

<i>img</i>	Input image from which dimensions are borrowed.
<i>dimensions</i>	String describing the image size along the X,Y,Z and V-dimensions.
<i>value</i>	Value used for initialization.

Note

- Similar to [CImg\(const CImg<t>&,const char*\)](#), but it also fills the pixel buffer with the specified `value`.

8.1.3.14 CImg() [13/13]

```
template<typename T >
CImg (
    const CImgDisplay & disp ) [explicit]
```

Construct image from a display window.

Construct a new image instance with pixels of type `T`, as a snapshot of an existing `CImgDisplay` instance.

Parameters

<i>disp</i>	Input display window.
-------------	-----------------------

Note

- The [width\(\)](#) and [height\(\)](#) of the constructed image instance are the same as the specified [CImgDisplay](#).
- The [depth\(\)](#) and [spectrum\(\)](#) of the constructed image instance are respectively set to 1 and 3 (i.e. a 2D color image).
- The image pixels are read as 8-bits RGB values.

8.1.4 Member Function Documentation

8.1.4.1 [assign\(\)](#) [1/11]

```
template<typename T >
CImg< T > & assign ( )
```

Construct empty image **[in-place version]**.

In-place version of the default constructor [CImg\(\)](#). It simply resets the instance to an empty image.

8.1.4.2 [assign\(\)](#) [2/11]

```
template<typename T >
CImg< T > & assign (
    const unsigned int size_x,
    const unsigned int size_y = 1,
    const unsigned int size_z = 1,
    const unsigned int size_c = 1 )
```

Construct image with specified size **[in-place version]**.

In-place version of the constructor [CImg\(unsigned int,unsigned int,unsigned int,unsigned int\)](#).

8.1.4.3 [assign\(\)](#) [3/11]

```
template<typename T >
CImg< T > & assign (
    const unsigned int size_x,
    const unsigned int size_y,
    const unsigned int size_z,
    const unsigned int size_c,
    const T & value )
```

Construct image with specified size and initialize pixel values **[in-place version]**.

In-place version of the constructor [CImg\(unsigned int,unsigned int,unsigned int,unsigned int,T\)](#).

8.1.4.4 assign() [4/11]

```
template<typename T >
CImg< T > & assign (
    const unsigned int size_x,
    const unsigned int size_y,
    const unsigned int size_z,
    const unsigned int size_c,
    const int value0,
    const int value1,
    ... )
```

Construct image with specified size and initialize pixel values from a sequence of integers **[in-place version]**.

In-place version of the constructor [CImg\(unsigned int,unsigned int,unsigned int,unsigned int,int,int,...\)](#).

8.1.4.5 assign() [5/11]

```
template<typename T >
CImg< T > & assign (
    const unsigned int size_x,
    const unsigned int size_y,
    const unsigned int size_z,
    const unsigned int size_c,
    const double value0,
    const double value1,
    ... )
```

Construct image with specified size and initialize pixel values from a sequence of doubles **[in-place version]**.

In-place version of the constructor [CImg\(unsigned int,unsigned int,unsigned int,unsigned int,double,double,...\)](#).

8.1.4.6 assign() [6/11]

```
template<typename T >
CImg< T > & assign (
    const unsigned int size_x,
    const unsigned int size_y,
    const unsigned int size_z,
    const unsigned int size_c,
    const char *const values,
    const bool repeat_values )
```

Construct image with specified size and initialize pixel values from a value string **[in-place version]**.

In-place version of the constructor [CImg\(unsigned int,unsigned int,unsigned int,unsigned int,const char*,bool\)](#).

8.1.4.7 assign() [7/11]

```
template<typename T >
template<typename t >
CImg< T > & assign (
    const t *const values,
    const unsigned int size_x,
    const unsigned int size_y = 1,
    const unsigned int size_z = 1,
    const unsigned int size_c = 1 )
```

Construct image with specified size and initialize pixel values from a memory buffer **[in-place version]**.

In-place version of the constructor CImg(const t*,unsigned int,unsigned int,unsigned int,unsigned int).

8.1.4.8 assign() [8/11]

```
template<typename T >
CImg< T > & assign (
    const char *const filename )
```

Construct image from reading an image file **[in-place version]**.

In-place version of the constructor CImg(const char*).

8.1.4.9 assign() [9/11]

```
template<typename T >
template<typename t >
CImg< T > & assign (
    const CImg< t > & img )
```

Construct image copy **[in-place version]**.

In-place version of the constructor CImg(const CImg<t>&).

8.1.4.10 assign() [10/11]

```
template<typename T >
template<typename t >
CImg< T > & assign (
    const CImg< t > & img,
    const bool is_shared )
```

In-place version of the advanced copy constructor.

In-place version of the constructor CImg(const CImg<t>&,bool).

8.1.4.11 assign() [11/11]

```
template<typename T >
CImg< T > & assign (
    const CImgDisplay & disp )
```

Construct image from a display window **[in-place version]**.

In-place version of the constructor `CImg(const CImgDisplay&)`.

8.1.4.12 clear()

```
template<typename T >
CImg< T > & clear ( )
```

Construct empty image **[in-place version]**.

Equivalent to `assign()`.

Note

- It has been defined for compatibility with STL naming conventions.

8.1.4.13 move_to() [1/2]

```
template<typename T >
template<typename t >
CImg< t > & move_to (
    CImg< t > & img )
```

Transfer content of an image instance into another one.

Transfers the dimensions and the pixel buffer content of an image instance into another one, and replace instance by an empty image. It avoids the copy of the pixel buffer when possible.

Parameters

<i>img</i>	Destination image.
------------	--------------------

Note

- Pixel types `T` and `t` of source and destination images can be different, though the process is designed to be instantaneous when `T` and `t` are the same.

Example

```
CImg<float> src(256,256,1,3,0), // Construct a 256x256x1x3 (color) image filled with value '0'
            dest(16,16); // Construct a 16x16x1x1 (scalar) image
src.move_to(dest); // Now, 'src' is empty and 'dest' is the 256x256x1x3 image
```

8.1.4.14 move_to() [2/2]

```
template<typename T >
template<typename t >
CImgList< t > & move_to (
    CImgList< t > & list,
    const unsigned int pos = ~0U )
```

Transfer content of an image instance into a new image in an image list.

Transfers the dimensions and the pixel buffer content of an image instance into a newly inserted image at position *pos* in specified CImgList<t> instance.

Parameters

<i>list</i>	Destination list.
<i>pos</i>	Position of the newly inserted image in the list.

Note

- When optional parameter *pos* is omitted, the image instance is transferred as a new image at the end of the specified *list*.
- It is convenient to sequentially insert new images into image lists, with no additional copies of memory buffer.

Example

```
CImgList<float> list; // Construct an empty image list
CImg<float> img("reference.jpg"); // Read image from filename
img.move_to(list); // Transfer image content as a new item in the list (no buffer copy)
```

8.1.4.15 swap()

```
template<typename T >
CImg< T > & swap (
    CImg< T > & img )
```

Swap fields of two image instances.

Parameters

<i>img</i>	Image to swap fields with.
------------	----------------------------

Note

- It can be used to interchange the content of two images in a very fast way. Can be convenient when dealing with algorithms requiring two swapping buffers.

Example

```
CImg<float> img1("lena.jpg"),
img2("milla.jpg");
img1.swap(img2); // Now, 'img1' is 'milla' and 'img2' is 'lena'
```

8.1.4.16 empty()

```
template<typename T >
static CImg< T > & empty ( ) [static]
```

Return a reference to an empty image.

Note

This function is useful mainly to declare optional parameters having type `CImg<T>` in functions prototypes, e.g.

```
void f(const int x=0, const int y=0, const CImg<float>& img=CImg<float>::empty());
```

8.1.4.17 operator()() [1/2]

```
template<typename T >
T & operator() (
    const unsigned int x,
    const unsigned int y = 0,
    const unsigned int z = 0,
    const unsigned int c = 0 )
```

Access to a pixel value.

Return a reference to a located pixel value of the image instance, being possibly *const*, whether the image instance is *const* or not. This is the standard function to get/set pixel values in `CImg<T>` images.

Parameters

<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.

Note

- Range of pixel coordinates start from `(0,0,0,0)` to `(width() - 1, height() - 1, depth() - 1, spectrum() - 1)`.
- Due to the particular arrangement of the pixel buffers defined in `CImg`, you can omit one coordinate if the corresponding dimension is equal to 1. For instance, pixels of a 2D image (`depth()` equal to 1) can be accessed by `img(x,y,c)` instead of `img(x,y,0,c)`.

Warning

- There is *no* boundary checking done in this operator, to make it as fast as possible. You *must* take care of out-of-bounds access by yourself, if necessary. For debugging purposes, you may want to define macro `'cimg_verbosity'>=3` to enable additional boundary checking operations in this operator. In that case, warning messages will be printed on the error output when accessing out-of-bounds pixels.

Example

```

CImg<float> img(100,100,1,3,0); // Construct a 100x100x1x3 (color) image with pixels set to '0'
const float
    valR = img(10,10,0,0), // Read red value at coordinates (10,10)
    valG = img(10,10,0,1), // Read green value at coordinates (10,10)
    valB = img(10,10,2), // Read blue value at coordinates (10,10) (Z-coordinate can be omitted)
    avg = (valR + valG + valB)/3; // Compute average pixel value
img(10,10,0) = img(10,10,1) = img(10,10,2) = avg; // Replace the color pixel (10,10) by the average
grey value

```

8.1.4.18 operator>() [2/2]

```

template<typename T >
T & operator() (
    const unsigned int x,
    const unsigned int y,
    const unsigned int z,
    const unsigned int c,
    const unsignedT wh,
    const unsignedT whd = 0 )

```

Access to a pixel value.

Parameters

<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.
<i>wh</i>	Precomputed offset, must be equal to <code>width()*height()</code> .
<i>whd</i>	Precomputed offset, must be equal to <code>width()*height()*depth()</code> .

Note

- Similar to (but faster than) `operator()()`. It uses precomputed offsets to optimize memory access. You may use it to optimize the reading/writing of several pixel values in the same image (e.g. in a loop).

8.1.4.19 operator T*()

```

template<typename T >
operator T* ( )

```

Implicitly cast an image into a T*.

Implicitly cast a CImg<T> instance into a T* or const T* pointer, whether the image instance is *const* or not. The returned pointer points on the first value of the image pixel buffer.

Note

- It simply returns the pointer `data()` to the pixel buffer.
- This implicit conversion is convenient to test the empty state of images (`data()` being 0 in this case), e.g.


```
CImg<float> img1(100,100), img2; // 'img1' is a 100x100 image, 'img2' is an empty image
if (img1) { // Test succeeds, 'img1' is not an empty image
    if (!img2) { // Test succeeds, 'img2' is an empty image
        std::printf("'img1' is not empty, 'img2' is empty.");
    }
}
```
- It also allows users to use brackets to access pixel values, without need for a `CImg<T>::operator[ ]()`, e.g.


```
CImg<float> img(100,100);
const float value = img[99]; // Access to value of the last pixel on the first row
img[510] = 255; // Set pixel value at (10,5)
```

8.1.4.20 `operator=()` [1/4]

```
template<typename T >
CImg< T > & operator= (
    const T & value )
```

Assign a value to all image pixels.

Assign specified `value` to each pixel value of the image instance.

Parameters

<i>value</i>	Value that will be assigned to image pixels.
--------------	--

Note

- The image size is never modified.
- The `value` may be casted to pixel type `T` if necessary.

Example

```
CImg<char> img(100,100); // Declare image (with garbage values)
img = 0; // Set all pixel values to '0'
img = 1.2; // Set all pixel values to '1' (cast of '1.2' as a 'char')
```

8.1.4.21 `operator=()` [2/4]

```
template<typename T >
CImg< T > & operator= (
    const char *const expression )
```

Assign pixels values from a specified expression.

Initialize all pixel values from the specified string `expression`.

Parameters

<i>expression</i>	Value string describing the way pixel values are set.
-------------------	---

Note

- String parameter `expression` may describe different things:
 - If `expression` is a list of values (as in "1, 2, 3, 8, 3, 2"), or a formula (as in "(x*y) %255"), the pixel values are set from specified `expression` and the image size is not modified.
 - If `expression` is a filename (as in "reference.jpg"), the corresponding image file is loaded and replace the image instance. The image size is modified if necessary.

Example

```
CImg<float> img1(100,100), img2(img1), img3(img1); // Declare 3 scalar images 100x100 with
uninitialized values
img1 = "0,50,100,150,200,250,200,150,100,50"; // Set pixel values of 'img1' from a value sequence
img2 = "10*((x*y)%25)"; // Set pixel values of 'img2' from a formula
img3 = "reference.jpg"; // Set pixel values of 'img3' from a file (image size is modified)
(img1,img2,img3).display();
```

8.1.4.22 `operator=()` [3/4]

```
template<typename T >
template<typename t >
CImg< T > & operator= (
    const CImg< t > & img )
```

Copy an image into the current image instance.

Similar to the in-place copy constructor [assign\(const CImg<t>&\)](#).

8.1.4.23 `operator=()` [4/4]

```
template<typename T >
CImg< T > & operator= (
    const CImgDisplay & disp )
```

Copy the content of a display window to the current image instance.

Similar to [assign\(const CImgDisplay&\)](#).

8.1.4.24 `operator+=()` [1/3]

```
template<typename T >
template<typename t >
CImg< T > & operator+= (
    const t value )
```

In-place addition operator.

Add specified `value` to all pixels of an image instance.

Parameters

<i>value</i>	Value to add.
--------------	---------------

Note

- Resulting pixel values are casted to fit the pixel type `T`. For instance, adding `0.2` to a `CImg<char>` is possible but does nothing indeed.
- Overflow values are treated as with standard C++ numeric types. For instance,


```
CImg<unsigned char> img(100,100,1,1,255); // Construct a 100x100 image with pixel values '255'
img+=1; // Add '1' to each pixels -> Overflow
// here all pixels of image 'img' are equal to '0'.
```
- To prevent value overflow, you may want to consider pixel type `T` as `float` or `double`, and use `cut()` after addition.

Example

```
CImg<unsigned char> img1("reference.jpg"); // Load a 8-bits RGB image (values in [0,255])
CImg<float> img2(img1); // Construct a float-valued copy of 'img1'
img2+=100; // Add '100' to pixel values -> goes out of [0,255] but no problems with floats
img2.cut(0,255); // Cut values in [0,255] to fit the 'unsigned char' constraint
img1 = img2; // Rewrite safe result in 'unsigned char' version 'img1'
const CImg<unsigned char> img3 = (img1 + 100).cut(0,255); // Do the same in a more simple and elegant way
(img1, img2, img3).display();
```

8.1.4.25 `operator+=()` [2/3]

```
template<typename T >
CImg< T > & operator+= (
    const char *const expression )
```

In-place addition operator.

Add values to image pixels, according to the specified string `expression`.

Parameters

<i>expression</i>	Value string describing the way pixel values are added.
-------------------	---

Note

- Similar to `operator=(const char*)`, except that it adds values to the pixels of the current image instance, instead of assigning them.

8.1.4.26 `operator+=()` [3/3]

```
template<typename T >
template<typename t >
CImg< T > & operator+= (
    const CImg< t > & img )
```

In-place addition operator.

Add values to image pixels, according to the values of the input image `img`.

Parameters

<i>img</i>	Input image to add.
------------	---------------------

Note

- The size of the image instance is never modified.
- It is not mandatory that input image `img` has the same size as the image instance. If fewer values are available in `img`, then the values are added periodically. For instance, adding one WxH scalar image (`spectrum()` equal to 1) to one WxH color image (`spectrum()` equal to 3) means each color channel will be incremented with the same values at the same locations.

Example

```
CImg<float> img1("reference.jpg"); // Load a RGB color image (img1.spectrum()==3)
// Construct a scalar shading (img2.spectrum()==1).
const CImg<float> img2(img1.width(),img1.height(),1,1,"255*(x/w)^2");
img1+=img2; // Add shading to each channel of 'img1'
img1.cut(0,255); // Prevent [0,255] overflow
(img2,img1).display();
```

8.1.4.27 operator++() [1/2]

```
template<typename T >
CImg< T > & operator++ ( )
```

In-place increment operator (prefix).

Add 1 to all image pixels, and return a reference to the current incremented image instance.

Note

- Writing `++img` is equivalent to `img+=1`.

8.1.4.28 operator++() [2/2]

```
template<typename T >
CImg< T > operator++ (
    int )
```

In-place increment operator (postfix).

Add 1 to all image pixels, and return a new copy of the initial (pre-incremented) image instance.

Note

- Use the prefixed version `operator++()` if you don't need a copy of the initial (pre-incremented) image instance, since a useless image copy may be expensive in terms of memory usage.

8.1.4.29 operator+() [1/4]

```
template<typename T >
CImg< T > operator+ ( ) const
```

Return a non-shared copy of the image instance.

Note

- Use this operator to ensure you get a non-shared copy of an image instance with same pixel type `T`. Indeed, the usual copy constructor `CImg<T>(const CImg<T>&)` returns a shared copy of a shared input image, and it may be not desirable to work on a regular copy (e.g. for a resize operation) if you have no information about the shared state of the input image.
- Writing `(+img)` is equivalent to `CImg<T>(img, false)`.

8.1.4.30 operator+() [2/4]

```
template<typename T >
template<typename t >
CImg< typename cimg::superset< T, t >::type > operator+ (
    const t value ) const
```

Addition operator.

Similar to `operator+=(const t)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type `T`, if necessary.

8.1.4.31 operator+() [3/4]

```
template<typename T >
CImg< Tfloat > operator+ (
    const char *const expression ) const
```

Addition operator.

Similar to `operator+=(const char*)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type `T`, if necessary.

8.1.4.32 operator+() [4/4]

```
template<typename T >
template<typename t >
CImg< typename cimg::superset< T, t >::type > operator+ (
    const CImg< t > & img ) const
```

Addition operator.

Similar to `operator+=(const CImg<t>&)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type `T`, if necessary.

8.1.4.33 operator-=() [1/3]

```
template<typename T >
template<typename t >
CImg< T > & operator-= (
    const t value )
```

In-place subtraction operator.

Similar to `operator+=(const t)`, except that it performs a subtraction instead of an addition.

8.1.4.34 operator-=() [2/3]

```
template<typename T >
CImg< T > & operator-= (
    const char *const expression )
```

In-place subtraction operator.

Similar to `operator+=(const char*)`, except that it performs a subtraction instead of an addition.

8.1.4.35 operator-=() [3/3]

```
template<typename T >
template<typename t >
CImg< T > & operator-= (
    const CImg< t > & img )
```

In-place subtraction operator.

Similar to `operator+=(const CImg<t>&)`, except that it performs a subtraction instead of an addition.

8.1.4.36 operator--() [1/2]

```
template<typename T >
CImg< T > & operator-- ( )
```

In-place decrement operator (prefix).

Similar to `operator++()`, except that it performs a decrement instead of an increment.

8.1.4.37 operator--() [2/2]

```
template<typename T >
CImg< T > operator-- (
    int )
```

In-place decrement operator (postfix).

Similar to `operator++(int)`, except that it performs a decrement instead of an increment.

8.1.4.38 operator-() [1/4]

```
template<typename T >
CImg< T > operator- ( ) const
```

Replace each pixel by its opposite value.

Note

- If the computed opposite values are out-of-range, they are treated as with standard C++ numeric types. For instance, the `unsigned char` opposite of 1 is 255.

Example

```
const CImg<unsigned char>
img1("reference.jpg"), // Load a RGB color image
img2 = -img1; // Compute its opposite (in 'unsigned char')
(img1,img2).display();
```

8.1.4.39 operator-() [2/4]

```
template<typename T >
template<typename t >
CImg< typename cimg::superset< T, t >::type > operator- (
    const t value ) const
```

Subtraction operator.

Similar to [operator-=\(const t\)](#), except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type `T`, if necessary.

8.1.4.40 operator-() [3/4]

```
template<typename T >
CImg< Tfloat > operator- (
    const char *const expression ) const
```

Subtraction operator.

Similar to [operator-=\(const char*\)](#), except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type `T`, if necessary.

8.1.4.41 operator-() [4/4]

```
template<typename T >
template<typename t >
CImg< typename cimg::superset< T, t >::type > operator- (
    const CImg< t > & img ) const
```

Subtraction operator.

Similar to [operator-=\(const CImg<t>&\)](#), except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type `T`, if necessary.

8.1.4.42 operator*=() [1/3]

```
template<typename T >
template<typename t >
CImg< T > & operator*= (
    const t value )
```

In-place multiplication operator.

Similar to [operator+=\(const t\)](#), except that it performs a multiplication instead of an addition.

8.1.4.43 operator*=() [2/3]

```
template<typename T >
CImg< T > & operator*= (
    const char *const expression )
```

In-place multiplication operator.

Similar to [operator+=\(const char*\)](#), except that it performs a multiplication instead of an addition.

8.1.4.44 operator*=() [3/3]

```
template<typename T >
template<typename t >
CImg< T > & operator*= (
    const CImg< t > & img )
```

In-place multiplication operator.

Replace the image instance by the matrix multiplication between the image instance and the specified matrix `img`.

Parameters

<code>img</code>	Second operand of the matrix multiplication.
------------------	--

Note

- It does *not* compute a pointwise multiplication between two images. For this purpose, use `mul(const CImg<t>&)` instead.
- The size of the image instance can be modified by this operator.

Example

```
CImg<float> A(2,2,1,1, 1,2,3,4); // Construct 2x2 matrix A = [1,2;3,4]
const CImg<float> X(1,2,1,1, 1,2); // Construct 1x2 vector X = [1;2]
A*=X; // Assign matrix multiplication A*X to 'A'
// 'A' is now a 1x2 vector whose values are [5;11].
```

8.1.4.45 operator*() [1/3]

```
template<typename T >
template<typename t >
CImg< typename cimg::superset< T, t >::type > operator* (
    const t value ) const
```

Multiplication operator.

Similar to `operator*=(const t)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type `T`, if necessary.

8.1.4.46 operator*() [2/3]

```
template<typename T >
CImg< Tfloat > operator* (
    const char *const expression ) const
```

Multiplication operator.

Similar to `operator*=(const char*)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type `T`, if necessary.

8.1.4.47 operator*() [3/3]

```
template<typename T >
template<typename t >
CImg< typename cimg::superset< T, t >::type > operator* (
    const CImg< t > & img ) const
```

Multiplication operator.

Similar to `operator*=(const CImg<t>&)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type `T`, if necessary.

8.1.4.48 operator/=(1/3)

```
template<typename T >
template<typename t >
CImg< T > & operator/= (
    const t value )
```

In-place division operator.

Similar to `operator+=(const t)`, except that it performs a division instead of an addition.

8.1.4.49 operator/=(2/3)

```
template<typename T >
CImg< T > & operator/= (
    const char *const expression )
```

In-place division operator.

Similar to `operator+=(const char*)`, except that it performs a division instead of an addition.

8.1.4.50 operator/=(3/3)

```
template<typename T >
template<typename t >
CImg< T > & operator/= (
    const CImg< t > & img )
```

In-place division operator.

Replace the image instance by the (right) matrix division between the image instance and the specified matrix `img`.

Parameters

<i>img</i>	Second operand of the matrix division.
------------	--

Note

- It does *not* compute a pointwise division between two images. For this purpose, use `div(const CImg<t>&)` instead.
- It returns the matrix operation `A*inverse(img)`.
- The size of the image instance can be modified by this operator.

8.1.4.51 operator/() [1/3]

```
template<typename T >
template<typename t >
CImg< typename cimg::superset< T, t >::type > operator/ (
    const t value ) const
```

Division operator.

Similar to `operator/=(const t)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type `T`, if necessary.

8.1.4.52 operator/() [2/3]

```
template<typename T >
CImg< Tfloat > operator/ (
    const char *const expression ) const
```

Division operator.

Similar to `operator/=(const char*)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type `T`, if necessary.

8.1.4.53 operator/() [3/3]

```
template<typename T >
template<typename t >
CImg< typename cimg::superset< T, t >::type > operator/ (
    const CImg< t > & img ) const
```

Division operator.

Similar to `operator/=(const CImg<t>&)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type `T`, if necessary.

8.1.4.54 operator%=() [1/3]

```
template<typename T >
template<typename t >
CImg< T > & operator%=(
    const t value )
```

In-place modulo operator.

Similar to `operator+=(const t)`, except that it performs a modulo operation instead of an addition.

8.1.4.55 operator%=() [2/3]

```
template<typename T >
CImg< T > & operator%=(
    const char *const expression )
```

In-place modulo operator.

Similar to `operator+=(const char*)`, except that it performs a modulo operation instead of an addition.

8.1.4.56 operator%=() [3/3]

```
template<typename T >
template<typename t >
CImg< T > & operator%=(
    const CImg< t > & img )
```

In-place modulo operator.

Similar to `operator+=(const CImg<t>&)`, except that it performs a modulo operation instead of an addition.

8.1.4.57 operator%() [1/3]

```
template<typename T >
template<typename t >
CImg< typename cimg::superset< T, t >::type > operator% (
    const t value ) const
```

Modulo operator.

Similar to `operator%=(const t)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type `T`, if necessary.

8.1.4.58 operator%() [2/3]

```
template<typename T >
CImg< Tfloat > operator% (
    const char *const expression ) const
```

Modulo operator.

Similar to `operator%=(const char*)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type `T`, if necessary.

8.1.4.59 operator%() [3/3]

```
template<typename T >
template<typename t >
CImg< typename cimg::superset< T, t >::type > operator% (
    const CImg< t > & img ) const
```

Modulo operator.

Similar to `operator%=(const CImg<t>&)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type `T`, if necessary.

8.1.4.60 operator&=() [1/3]

```
template<typename T >
template<typename t >
CImg< T > & operator&= (
    const t value )
```

In-place bitwise AND operator.

Similar to `operator+=(const t)`, except that it performs a bitwise AND operation instead of an addition.

8.1.4.61 operator&=() [2/3]

```
template<typename T >
CImg< T > & operator&= (
    const char *const expression )
```

In-place bitwise AND operator.

Similar to `operator+=(const char*)`, except that it performs a bitwise AND operation instead of an addition.

8.1.4.62 operator&=() [3/3]

```
template<typename T >
template<typename t >
CImg< T > & operator&= (
    const CImg< t > & img )
```

In-place bitwise AND operator.

Similar to `operator+=(const CImg<t>&)`, except that it performs a bitwise AND operation instead of an addition.

8.1.4.63 operator&() [1/3]

```
template<typename T >
template<typename t >
CImg< T > operator& (
    const t value ) const
```

Bitwise AND operator.

Similar to `operator&=(const t)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is T.

8.1.4.64 operator&() [2/3]

```
template<typename T >
CImg< T > operator& (
    const char *const expression ) const
```

Bitwise AND operator.

Similar to `operator&=(const char*)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is T.

8.1.4.65 operator&() [3/3]

```
template<typename T >
template<typename t >
CImg< T > operator& (
    const CImg< t > & img ) const
```

Bitwise AND operator.

Similar to `operator&=(const CImg<t>&)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is T.

8.1.4.66 operator" |=() [1/3]

```
template<typename T >
template<typename t >
CImg< T > & operator|= (
    const t value )
```

In-place bitwise OR operator.

Similar to `operator+=(const t)`, except that it performs a bitwise OR operation instead of an addition.

8.1.4.67 operator" |=() [2/3]

```
template<typename T >
CImg< T > & operator|= (
    const char *const expression )
```

In-place bitwise OR operator.

Similar to `operator+=(const char*)`, except that it performs a bitwise OR operation instead of an addition.

8.1.4.68 operator" |=() [3/3]

```
template<typename T >
template<typename t >
CImg< T > & operator|= (
    const CImg< t > & img )
```

In-place bitwise OR operator.

Similar to `operator+=(const CImg<t>&)`, except that it performs a bitwise OR operation instead of an addition.

8.1.4.69 operator" |() [1/3]

```
template<typename T >
template<typename t >
CImg< T > operator| (
    const t value ) const
```

Bitwise OR operator.

Similar to `operator|=(const t)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is T.

8.1.4.70 operator" | () [2/3]

```
template<typename T >
CImg< T > operator| (
    const char *const expression ) const
```

Bitwise OR operator.

Similar to `operator|=(const char*)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is T.

8.1.4.71 operator" | () [3/3]

```
template<typename T >
template<typename t >
CImg< T > operator| (
    const CImg< t > & img ) const
```

Bitwise OR operator.

Similar to `operator|=(const CImg<t>&)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is T.

8.1.4.72 operator^=() [1/3]

```
template<typename T >
template<typename t >
CImg< T > & operator^= (
    const t value )
```

In-place bitwise XOR operator.

Similar to `operator+=(const t)`, except that it performs a bitwise XOR operation instead of an addition.

Warning

- It does *not* compute the *power* of pixel values. For this purpose, use `pow(const t)` instead.

8.1.4.73 operator^=() [2/3]

```
template<typename T >
CImg< T > & operator^= (
    const char *const expression )
```

In-place bitwise XOR operator.

Similar to `operator+=(const char*)`, except that it performs a bitwise XOR operation instead of an addition.

Warning

- It does *not* compute the *power* of pixel values. For this purpose, use `pow(const char*)` instead.

8.1.4.74 operator^=() [3/3]

```
template<typename T >
template<typename t >
CImg< T > & operator^= (
    const CImg< t > & img )
```

In-place bitwise XOR operator.

Similar to `operator+=(const CImg<t>&)`, except that it performs a bitwise XOR operation instead of an addition.

Warning

- It does *not* compute the *power* of pixel values. For this purpose, use `pow(const CImg<t>&)` instead.

8.1.4.75 operator^() [1/3]

```
template<typename T >
template<typename t >
CImg< T > operator^ (
    const t value ) const
```

Bitwise XOR operator.

Similar to `operator^=(const t)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is T.

8.1.4.76 operator^() [2/3]

```
template<typename T >
CImg< T > operator^ (
    const char *const expression ) const
```

Bitwise XOR operator.

Similar to `operator^=(const char*)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is T.

8.1.4.77 operator^() [3/3]

```
template<typename T >
template<typename t >
CImg< T > operator^ (
    const CImg< t > & img ) const
```

Bitwise XOR operator.

Similar to `operator^=(const CImg<t>&)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is T.

8.1.4.78 operator<<=() [1/3]

```
template<typename T >
template<typename t >
CImg< T > & operator<<= (
    const t value )
```

In-place bitwise left shift operator.

Similar to `operator+=(const t)`, except that it performs a bitwise left shift instead of an addition.

8.1.4.79 operator<<=() [2/3]

```
template<typename T >
CImg< T > & operator<<= (
    const char *const expression )
```

In-place bitwise left shift operator.

Similar to `operator+=(const char*)`, except that it performs a bitwise left shift instead of an addition.

8.1.4.80 operator<<=() [3/3]

```
template<typename T >
template<typename t >
CImg< T > & operator<<= (
    const CImg< t > & img )
```

In-place bitwise left shift operator.

Similar to `operator+=(const CImg<t>&)`, except that it performs a bitwise left shift instead of an addition.

8.1.4.81 operator<<() [1/3]

```
template<typename T >
template<typename t >
CImg< T > operator<< (
    const t value ) const
```

Bitwise left shift operator.

Similar to `operator<<=(const t)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is T.

8.1.4.82 operator<<() [2/3]

```
template<typename T >
CImg< T > operator<< (
    const char *const expression ) const
```

Bitwise left shift operator.

Similar to `operator<<=(const char*)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is T.

8.1.4.83 operator<<() [3/3]

```
template<typename T >
template<typename t >
CImg< T > operator<< (
    const CImg< t > & img ) const
```

Bitwise left shift operator.

Similar to `operator<<=(const CImg<t>&)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is T.

8.1.4.84 operator>>=() [1/3]

```
template<typename T >
template<typename t >
CImg< T > & operator>>= (
    const t value )
```

In-place bitwise right shift operator.

Similar to `operator+=(const t)`, except that it performs a bitwise right shift instead of an addition.

8.1.4.85 operator>>=() [2/3]

```
template<typename T >
CImg< T > & operator>>= (
    const char *const expression )
```

In-place bitwise right shift operator.

Similar to `operator+=(const char*)`, except that it performs a bitwise right shift instead of an addition.

8.1.4.86 operator>>=() [3/3]

```
template<typename T >
template<typename t >
CImg< T > & operator>>= (
    const CImg< t > & img )
```

In-place bitwise right shift operator.

Similar to `operator+=(const CImg<t>&)`, except that it performs a bitwise right shift instead of an addition.

8.1.4.87 operator>>() [1/3]

```
template<typename T >
template<typename t >
CImg< T > operator>> (
    const t value ) const
```

Bitwise right shift operator.

Similar to `operator>>=(const t)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is T.

8.1.4.88 operator>>() [2/3]

```
template<typename T >
CImg< T > operator>> (
    const char *const expression ) const
```

Bitwise right shift operator.

Similar to `operator>>=(const char*)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is T.

8.1.4.89 operator>>() [3/3]

```
template<typename T >
template<typename t >
CImg< T > operator>> (
    const CImg< t > & img ) const
```

Bitwise right shift operator.

Similar to `operator>>=(const CImg<t>&)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is T.

8.1.4.90 operator~()

```
template<typename T >
CImg< T > operator~ ( ) const
```

Bitwise inversion operator.

Similar to `operator-()`, except that it compute the bitwise inverse instead of the opposite value.

8.1.4.91 operator==() [1/3]

```
template<typename T >
template<typename t >
bool operator==(
    const t value ) const
```

Test if all pixels of an image have the same value.

Return true is all pixels of the image instance are equal to the specified value.

Parameters

<i>value</i>	Reference value to compare with.
--------------	----------------------------------

8.1.4.92 operator==() [2/3]

```
template<typename T >
bool operator== (
    const char *const expression ) const
```

Test if all pixel values of an image follow a specified expression.

Return `true` is all pixels of the image instance are equal to the specified `expression`.

Parameters

<i>expression</i>	Value string describing the way pixel values are compared.
-------------------	--

8.1.4.93 operator==() [3/3]

```
template<typename T >
template<typename t >
bool operator== (
    const CImg< t > & img ) const
```

Test if two images have the same size and values.

Return `true` if the image instance and the input image `img` have the same pixel values, even if the dimensions of the two images do not match. It returns `false` otherwise.

Parameters

<i>img</i>	Input image to compare with.
------------	------------------------------

Note

- The pixel buffer pointers `data()` of the two compared images do not have to be the same for `operator==( )` to return `true`. Only the dimensions and the pixel values matter. Thus, the comparison can be `true` even for different pixel types `T` and `t`.

Example

```
const CImg<float> img1(1,3,1,1, 0,1,2); // Construct a 1x3 vector [0;1;2] (with 'float' pixel values)
const CImg<char> img2(1,3,1,1, 0,1,2); // Construct a 1x3 vector [0;1;2] (with 'char' pixel values)
if (img1==img2) { // Test succeeds, image dimensions and values are the same
    std::printf("'img1' and 'img2' have same dimensions and values.");
}
```

8.1.4.94 operator!=() [1/3]

```
template<typename T >
template<typename t >
bool operator!= (
    const t value ) const
```

Test if pixels of an image are all different from a value.

Return `true` is all pixels of the image instance are different than the specified `value`.

Parameters

<i>value</i>	Reference value to compare with.
--------------	----------------------------------

8.1.4.95 operator"!=()" [2/3]

```
template<typename T >
bool operator!= (
    const char *const expression ) const
```

Test if all pixel values of an image are different from a specified expression.

Return `true` if all pixels of the image instance are different to the specified `expression`.

Parameters

<i>expression</i>	Value string describing the way pixel values are compared.
-------------------	--

8.1.4.96 operator"!=()" [3/3]

```
template<typename T >
template<typename t >
bool operator!= (
    const CImg< t > & img ) const
```

Test if two images have different sizes or values.

Return `true` if the image instance and the input image `img` have different dimensions or pixel values, and `false` otherwise.

Parameters

<i>img</i>	Input image to compare with.
------------	------------------------------

Note

- Writing `img1!=img2` is equivalent to `!(img1==img2)`.

8.1.4.97 operator,() [1/2]

```
template<typename T >
template<typename t >
CImgList< typename cimg::superset< T, t >::type > operator, (
    const CImg< t > & img ) const
```

Construct an image list from two images.

Return a new list of image (`CImgList` instance) containing exactly two elements:

- A copy of the image instance, at position [0].
- A copy of the specified image `img`, at position [1].

Parameters

<i>img</i>	Input image that will be the second image of the resulting list.
------------	--

Note

- The family of [operator,\(\)](#) is convenient to easily create list of images, but it is also *quite slow* in practice (see warning below).
- Constructed lists contain no shared images. If image instance or input image `img` are shared, they are inserted as new non-shared copies in the resulting list.
- The pixel type of the returned list may be a superset of the initial pixel type `T`, if necessary.

Warning

- Pipelining [operator,\(\)](#) `N` times will perform `N` copies of the entire content of a (growing) image list. This may become very expensive in terms of speed and used memory. You should avoid using this technique to build a new [CImgList](#) instance from several images, if you are seeking for performance. Fast insertions of images in an image list are possible with [CImgList<T>::insert\(const CImg<t>&,unsigned int,bool\)](#) or [move_to\(CImgList<t>&,unsigned int\)](#).

Example

```
const CImg<float>
img1("reference.jpg"),
img2 = img1.get_mirror('x'),
img3 = img2.get_blur(5);
const CImgList<float> list = (img1,img2); // Create list of two elements from 'img1' and 'img2'
(list,img3).display(); // Display image list containing copies of 'img1','img2' and 'img3'
```

8.1.4.98 operator,() [2/2]

```
template<typename T >
template<typename t >
CImgList< typename cimg::superset< T, t >::type > operator, (
    const CImgList< t > & list ) const
```

Construct an image list from image instance and an input image list.

Return a new list of images ([CImgList](#) instance) containing exactly `list.size() + 1` elements:

- A copy of the image instance, at position `[0]`.
- A copy of the specified image list `list`, from positions `[1]` to `[list.size()]`.

Parameters

<i>list</i>	Input image list that will be appended to the image instance.
-------------	---

Note

- Similar to [operator,\(const CImg<t>&\) const](#), except that it takes an image list as an argument.

8.1.4.99 operator<()

```
template<typename T >
CImgList< T > operator< (
    const char axis ) const
```

Split the image along a specified axis.

Return a new list of images (`CImgList` instance) containing the split components of the instance image along the specified axis.

Parameters

<code>axis</code>	Splitting axis (can be 'x','y','z' or 'c')
-------------------	--

Note

- Similar to `get_split(char,int) const`, with default second argument.

Example

```
const CImg<unsigned char> img("reference.jpg"); // Load a RGB color image
const CImgList<unsigned char> list = (img<'c'>); // Get a list of its three R,G,B channels
(img,list).display();
```

8.1.4.100 pixel_type()

```
template<typename T >
static const char * pixel_type ( ) [static]
```

Return the type of image pixel values as a C string.

Return a `char*` string containing the usual type name of the image pixel values (i.e. a stringified version of the template parameter `T`).

Note

- The returned string does not contain any spaces.
- If the pixel type `T` does not correspond to a registered type, the string "unknown" is returned.

8.1.4.101 width()

```
template<typename T >
int width ( ) const
```

Return the number of image columns.

Return the image width, i.e. the image dimension along the X-axis.

Note

- The `width()` of an empty image is equal to 0.
- `width()` is typically equal to 1 when considering images as *vectors* for matrix calculations.
- `width()` returns an `int`, although the image width is internally stored as an `unsigned int`. Using an `int` is safer and prevents arithmetic traps possibly encountered when doing calculations involving `unsigned int` variables. Access to the initial `unsigned int` variable is possible (though not recommended) by `(*this)._width`.

8.1.4.102 height()

```
template<typename T >
int height ( ) const
```

Return the number of image rows.

Return the image height, i.e. the image dimension along the Y-axis.

Note

- The `height()` of an empty image is equal to 0.
- `height()` returns an `int`, although the image height is internally stored as an `unsigned int`. Using an `int` is safer and prevents arithmetic traps possibly encountered when doing calculations involving `unsigned int` variables. Access to the initial `unsigned int` variable is possible (though not recommended) by `(*this)._height`.

8.1.4.103 depth()

```
template<typename T >
int depth ( ) const
```

Return the number of image slices.

Return the image depth, i.e. the image dimension along the Z-axis.

Note

- The `depth()` of an empty image is equal to 0.
- `depth()` is typically equal to 1 when considering usual 2D images. When `depth() > 1`, the image is said to be *volumetric*.
- `depth()` returns an `int`, although the image depth is internally stored as an `unsigned int`. Using an `int` is safer and prevents arithmetic traps possibly encountered when doing calculations involving `unsigned int` variables. Access to the initial `unsigned int` variable is possible (though not recommended) by `(*this)._depth`.

8.1.4.104 spectrum()

```
template<typename T >
int spectrum ( ) const
```

Return the number of image channels.

Return the number of image channels, i.e. the image dimension along the C-axis.

Note

- The `spectrum()` of an empty image is equal to 0.
- `spectrum()` is typically equal to 1 when considering scalar-valued images, to 3 for RGB-coded color images, and to 4 for RGBA-coded color images (with alpha-channel). The number of channels of an image instance is not limited. The meaning of the pixel values is not linked up to the number of channels (e.g. a 4-channel image may represent either a RGBA or a CMYK color image).
- `spectrum()` returns an `int`, although the image spectrum is internally stored as an `unsigned int`. Using an `int` is safer and prevents arithmetic traps possibly encountered when doing calculations involving `unsigned int` variables. Access to the initial `unsigned int` variable is possible (though not recommended) by `(*this)._spectrum`.

8.1.4.105 size()

```
template<typename T >
ulongT size ( ) const
```

Return the total number of pixel values.

Return `width()*height()*depth()*spectrum()`, i.e. the total number of values of type `T` in the pixel buffer of the image instance.

Note

- The `size()` of an empty image is equal to 0.
- The allocated memory size for a pixel buffer of a non-shared `CImg<T>` instance is equal to `size()*sizeof(T)`.

Example

```
const CImg<float> img(100,100,1,3); // Construct new 100x100 color image
if (img.size()==30000) // Test succeeds
    std::printf("Pixel buffer uses %lu bytes",
                img.size()*sizeof(float));
```

8.1.4.106 data() [1/2]

```
template<typename T >
T * data ( )
```

Return a pointer to the first pixel value.

Return a `T*`, or a `const T*` pointer to the first value in the pixel buffer of the image instance, whether the instance is `const` or not.

Note

- The `data()` of an empty image is equal to 0 (null pointer).
- The allocated pixel buffer for the image instance starts from `data()` and goes to `data()+size() - 1` (included).
- To get the pointer to one particular location of the pixel buffer, use `data(unsigned int,unsigned int,unsigned int,unsigned int)` instead.

8.1.4.107 data() [2/2]

```
template<typename T >
T * data (
    const unsigned int x,
    const unsigned int y = 0,
    const unsigned int z = 0,
    const unsigned int c = 0 )
```

Return a pointer to a located pixel value.

Return a `T*`, or a `const T*` pointer to the value located at `(x,y,z,c)` in the pixel buffer of the image instance, whether the instance is `const` or not.

Parameters

<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.

Note

- Writing `img.data(x,y,z,c)` is equivalent to `&(img(x,y,z,c))`. Thus, this function has the same properties as `operator()(unsigned int,unsigned int,unsigned int,unsigned int)`.

8.1.4.108 `offset()`

```
template<typename T >
longT offset (
    const int x,
    const int y = 0,
    const int z = 0,
    const int c = 0 ) const
```

Return the offset to a located pixel value, with respect to the beginning of the pixel buffer.

Parameters

<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.

Note

- Writing `img.data(x,y,z,c)` is equivalent to `&(img(x,y,z,c)) - img.data()`. Thus, this function has the same properties as `operator()(unsigned int,unsigned int,unsigned int,unsigned int)`.

Example

```
const CImg<float> img(100,100,1,3); // Define a 100x100 RGB-color image
const long off = img.offset(10,10,0,2); // Get the offset of the blue value of the pixel located at
(10,10)
const float val = img[off]; // Get the blue value of this pixel
```

8.1.4.109 `begin()`

```
template<typename T >
iterator begin ( )
```

Return a `CImg<T>::iterator` pointing to the first pixel value.

Note

- Equivalent to `data()`.
- It has been mainly defined for compatibility with STL naming conventions.

8.1.4.110 end()

```
template<typename T >
iterator end ( )
```

Return a `CImg<T>::iterator` pointing next to the last pixel value.

Note

- Writing `img.end()` is equivalent to `img.data() + img.size()`.
- It has been mainly defined for compatibility with STL naming conventions.

Warning

- The returned iterator actually points to a value located *outside* the acceptable bounds of the pixel buffer. Trying to read or write the content of the returned iterator will probably result in a crash. Use it mainly as a strict upper bound for a `CImg<T>::iterator`.

Example

```
CImg<float> img(100,100,1,3); // Define a 100x100 RGB color image
// 'img.end()' used below as an upper bound for the iterator.
for (CImg<float>::iterator it = img.begin(); it<img.end(); ++it)
    *it = 0;
```

8.1.4.111 front()

```
template<typename T >
T & front ( )
```

Return a reference to the first pixel value.

Note

- Writing `img.front()` is equivalent to `img[0]`, or `img(0,0,0,0)`.
- It has been mainly defined for compatibility with STL naming conventions.

8.1.4.112 back()

```
template<typename T >
T & back ( )
```

Return a reference to the last pixel value.

Note

- Writing `img.back()` is equivalent to `img[img.size() - 1]`, or `img(img.width() - 1, img.height() - 1, img.depth() - 1, img.spectrum() - 1)`.
- It has been mainly defined for compatibility with STL naming conventions.

8.1.4.113 at() [1/2]

```
template<typename T >
T & at (
    const int offset,
    const T & out_value )
```

Access to a pixel value at a specified offset, using Dirichlet boundary conditions.

Return a reference to the pixel value of the image instance located at a specified `offset`, or to a specified default value in case of out-of-bounds access.

Parameters

<i>offset</i>	Offset to the desired pixel value.
<i>out_value</i>	Default value returned if <i>offset</i> is outside image bounds.

Note

- Writing `img.at(offset, out_value)` is similar to `img[offset]`, except that if `offset` is outside bounds (e.g. `offset < 0` or `offset >= img.size()`), a reference to a value `out_value` is safely returned instead.
- Due to the additional boundary checking operation, this function is slower than `operator()()`. Use it when you are *not* sure about the validity of the specified pixel offset.

8.1.4.114 `at()` [2/2]

```
template<typename T >
T & at (
    const int offset )
```

Access to a pixel value at a specified offset, using Neumann boundary conditions.

Return a reference to the pixel value of the image instance located at a specified `offset`, or to the nearest pixel location in the image instance in case of out-of-bounds access.

Parameters

<i>offset</i>	Offset to the desired pixel value.
---------------	------------------------------------

Note

- Similar to `at(int, const T)`, except that an out-of-bounds access returns the value of the nearest pixel in the image instance, regarding the specified offset, i.e.
 - If `offset < 0`, then `img[0]` is returned.
 - If `offset >= img.size()`, then `img[img.size() - 1]` is returned.
- Due to the additional boundary checking operation, this function is slower than `operator()()`. Use it when you are *not* sure about the validity of the specified pixel offset.
- If you know your image instance is *not* empty, you may rather use the slightly faster function `_at(int)`.

8.1.4.115 `atX()` [1/2]

```
template<typename T >
T & atX (
    const int x,
    const int y,
    const int z,
    const int c,
    const T & out_value )
```

Access to a pixel value, using Dirichlet boundary conditions for the X-coordinate.

Return a reference to the pixel value of the image instance located at `(x,y,z,c)`, or to a specified default value in case of out-of-bounds access along the X-axis.

Parameters

<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.
<i>out_value</i>	Default value returned if (<i>x</i> , <i>y</i> , <i>z</i> , <i>c</i>) is outside image bounds.

Note

- Similar to `operator()()`, except that an out-of-bounds access along the X-axis returns the specified value `out_value`.
- Due to the additional boundary checking operation, this function is slower than `operator()()`. Use it when you are *not* sure about the validity of the specified pixel coordinates.

Warning

- There is *no* boundary checking performed for the Y,Z and C-coordinates, so they must be inside image bounds.

8.1.4.116 `atX()` [2/2]

```
template<typename T >
T & atX (
    const int x,
    const int y = 0,
    const int z = 0,
    const int c = 0 )
```

Access to a pixel value, using Neumann boundary conditions for the X-coordinate.

Return a reference to the pixel value of the image instance located at (*x*,*y*,*z*,*c*), or to the nearest pixel location in the image instance in case of out-of-bounds access along the X-axis.

Parameters

<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.

Note

- Similar to `at(int,int,int,int,const T)`, except that an out-of-bounds access returns the value of the nearest pixel in the image instance, regarding the specified X-coordinate.
- Due to the additional boundary checking operation, this function is slower than `operator()()`. Use it when you are *not* sure about the validity of the specified pixel coordinates.
- If you know your image instance is *not* empty, you may rather use the slightly faster function `__at__` `at(int,int,int,int)`.

Warning

- There is *no* boundary checking performed for the Y,Z and C-coordinates, so they must be inside image bounds.

8.1.4.117 atXY() [1/2]

```
template<typename T >
T & atXY (
    const int x,
    const int y,
    const int z,
    const int c,
    const T & out_value )
```

Access to a pixel value, using Dirichlet boundary conditions for the X and Y-coordinates.

Similar to `atX(int,int,int,int,const T)`, except that boundary checking is performed both on X and Y-coordinates.

8.1.4.118 atXY() [2/2]

```
template<typename T >
T & atXY (
    const int x,
    const int y,
    const int z = 0,
    const int c = 0 )
```

Access to a pixel value, using Neumann boundary conditions for the X and Y-coordinates.

Similar to `atX(int,int,int,int)`, except that boundary checking is performed both on X and Y-coordinates.

Note

- If you know your image instance is *not* empty, you may rather use the slightly faster function `_atXY(int,int,int,int)`.

8.1.4.119 atXYZ() [1/2]

```
template<typename T >
T & atXYZ (
    const int x,
    const int y,
    const int z,
    const int c,
    const T & out_value )
```

Access to a pixel value, using Dirichlet boundary conditions for the X,Y and Z-coordinates.

Similar to `atX(int,int,int,int,const T)`, except that boundary checking is performed both on X,Y and Z-coordinates.

8.1.4.120 atXYZ() [2/2]

```
template<typename T >
T & atXYZ (
    const int x,
    const int y,
    const int z,
    const int c = 0 )
```

Access to a pixel value, using Neumann boundary conditions for the X,Y and Z-coordinates.

Similar to [atX\(int,int,int,int\)](#), except that boundary checking is performed both on X,Y and Z-coordinates.

Note

- If you know your image instance is *not* empty, you may rather use the slightly faster function `_atXYZ(int,int,int,int)`.

8.1.4.121 atXYZC() [1/2]

```
template<typename T >
T & atXYZC (
    const int x,
    const int y,
    const int z,
    const int c,
    const T & out_value )
```

Access to a pixel value, using Dirichlet boundary conditions.

Similar to `atX(int,int,int,int,const T)`, except that boundary checking is performed on all X,Y,Z and C-coordinates.

8.1.4.122 atXYZC() [2/2]

```
template<typename T >
T & atXYZC (
    const int x,
    const int y,
    const int z,
    const int c )
```

Access to a pixel value, using Neumann boundary conditions.

Similar to [atX\(int,int,int,int\)](#), except that boundary checking is performed on all X,Y,Z and C-coordinates.

Note

- If you know your image instance is *not* empty, you may rather use the slightly faster function `_atXYZC(int,int,int,int)`.

8.1.4.123 linear_atX() [1/2]

```
template<typename T >
Tfloat linear_atX (
    const float fx,
    const int y,
    const int z,
    const int c,
    const T & out_value ) const
```

Return pixel value, using linear interpolation and Dirichlet boundary conditions for the X-coordinate.

Return a linearly-interpolated pixel value of the image instance located at (f_x, y, z, c) , or a specified default value in case of out-of-bounds access along the X-axis.

Parameters

<i>fx</i>	X-coordinate of the pixel value (float-valued).
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.
<i>out_value</i>	Default value returned if (f_x, y, z, c) is outside image bounds.

Note

- Similar to `atX(int,int,int,int,const T)`, except that the returned pixel value is approximated by a linear interpolation along the X-axis, if corresponding coordinates are not integers.
- The type of the returned pixel value is extended to `float`, if the pixel type `T` is not float-valued.

Warning

- There is *no* boundary checking performed for the Y,Z and C-coordinates, so they must be inside image bounds.

8.1.4.124 linear_atX() [2/2]

```
template<typename T >
Tfloat linear_atX (
    const float fx,
    const int y = 0,
    const int z = 0,
    const int c = 0 ) const
```

Return pixel value, using linear interpolation and Neumann boundary conditions for the X-coordinate.

Return a linearly-interpolated pixel value of the image instance located at (f_x, y, z, c) , or the value of the nearest pixel location in the image instance in case of out-of-bounds access along the X-axis.

Parameters

<i>fx</i>	X-coordinate of the pixel value (float-valued).
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.

Note

- Similar to `linear_atX(float,int,int,int,const T) const`, except that an out-of-bounds access returns the value of the nearest pixel in the image instance, regarding the specified X-coordinate.
- If you know your image instance is *not* empty, you may rather use the slightly faster function `_linear_atX(float,int,int,int)`.

Warning

- There is *no* boundary checking performed for the Y,Z and C-coordinates, so they must be inside image bounds.

8.1.4.125 linear_atXY() [1/2]

```
template<typename T >
Tfloat linear_atXY (
    const float fx,
    const float fy,
    const int z,
    const int c,
    const T & out_value ) const
```

Return pixel value, using linear interpolation and Dirichlet boundary conditions for the X and Y-coordinates.

Similar to `linear_atX(float,int,int,int,const T) const`, except that the linear interpolation and the boundary checking are achieved both for X and Y-coordinates.

8.1.4.126 linear_atXY() [2/2]

```
template<typename T >
Tfloat linear_atXY (
    const float fx,
    const float fy,
    const int z = 0,
    const int c = 0 ) const
```

Return pixel value, using linear interpolation and Neumann boundary conditions for the X and Y-coordinates.

Similar to `linear_atX(float,int,int,int) const`, except that the linear interpolation and the boundary checking are achieved both for X and Y-coordinates.

Note

- If you know your image instance is *not* empty, you may rather use the slightly faster function `_linear_atXY(float,float,int,int)`.

8.1.4.127 linear_atXYZ() [1/2]

```
template<typename T >
Tfloat linear_atXYZ (
    const float fx,
    const float fy,
    const float fz,
    const int c,
    const T & out_value ) const
```

Return pixel value, using linear interpolation and Dirichlet boundary conditions for the X,Y and Z-coordinates.

Similar to `linear_atX(float,int,int,int,const T) const`, except that the linear interpolation and the boundary checking are achieved both for X,Y and Z-coordinates.

8.1.4.128 linear_atXYZ() [2/2]

```
template<typename T >
Tfloat linear_atXYZ (
    const float fx,
    const float fy = 0,
    const float fz = 0,
    const int c = 0 ) const
```

Return pixel value, using linear interpolation and Neumann boundary conditions for the X,Y and Z-coordinates.

Similar to `linear_atX(float,int,int,int) const`, except that the linear interpolation and the boundary checking are achieved both for X,Y and Z-coordinates.

Note

- If you know your image instance is *not* empty, you may rather use the slightly faster function `_linear_atXYZ(float,float,float,int)`.

8.1.4.129 linear_atXYZC() [1/2]

```
template<typename T >
Tfloat linear_atXYZC (
    const float fx,
    const float fy,
    const float fz,
    const float fc,
    const T & out_value ) const
```

Return pixel value, using linear interpolation and Dirichlet boundary conditions for all X,Y,Z,C-coordinates.

Similar to `linear_atX(float,int,int,int,const T) const`, except that the linear interpolation and the boundary checking are achieved for all X,Y,Z and C-coordinates.

8.1.4.130 linear_atXYZC() [2/2]

```
template<typename T >
Tfloat linear_atXYZC (
    const float fx,
    const float fy = 0,
    const float fz = 0,
    const float fc = 0 ) const
```

Return pixel value, using linear interpolation and Neumann boundary conditions for all X,Y,Z and C-coordinates.

Similar to [linear_atX\(float,int,int,int\) const](#), except that the linear interpolation and the boundary checking are achieved for all X,Y,Z and C-coordinates.

Note

- If you know your image instance is *not* empty, you may rather use the slightly faster function `_linear_atXYZC(float, float, float, float)`.

8.1.4.131 cubic_atX() [1/2]

```
template<typename T >
Tfloat cubic_atX (
    const float fx,
    const int y,
    const int z,
    const int c,
    const T & out_value ) const
```

Return pixel value, using cubic interpolation and Dirichlet boundary conditions for the X-coordinate.

Return a cubically-interpolated pixel value of the image instance located at (fx, y, z, c) , or a specified default value in case of out-of-bounds access along the X-axis. The cubic interpolation uses Hermite splines.

Parameters

<i>fx</i>	d X-coordinate of the pixel value (float-valued).
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.
<i>out_value</i>	Default value returned if (fx, y, z, c) is outside image bounds.

Note

- Similar to `linear_atX(float,int,int,int,const T) const`, except that the returned pixel value is approximated by a *cubic* interpolation along the X-axis.
- The type of the returned pixel value is extended to `float`, if the pixel type `T` is not float-valued.

Warning

- There is *no* boundary checking performed for the Y,Z and C-coordinates, so they must be inside image bounds.

8.1.4.132 cubic_atX_c() [1/2]

```
template<typename T >
T cubic_atX_c (
    const float fx,
    const int y,
    const int z,
    const int c,
    const T & out_value ) const
```

Return clamped pixel value, using cubic interpolation and Dirichlet boundary conditions for the X-coordinate.

Similar to `cubic_atX(float,int,int,int,const T) const`, except that the return value is clamped to stay in the min/max range of the datatype T.

8.1.4.133 cubic_atX() [2/2]

```
template<typename T >
Tfloat cubic_atX (
    const float fx,
    const int y = 0,
    const int z = 0,
    const int c = 0 ) const
```

Return pixel value, using cubic interpolation and Neumann boundary conditions for the X-coordinate.

Return a cubically-interpolated pixel value of the image instance located at (f_x, y, z, c) , or the value of the nearest pixel location in the image instance in case of out-of-bounds access along the X-axis. The cubic interpolation uses Hermite splines.

Parameters

<i>fx</i>	X-coordinate of the pixel value (float-valued).
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.

Note

- Similar to `cubic_atX(float,int,int,int,const T) const`, except that the returned pixel value is approximated by a cubic interpolation along the X-axis.
- If you know your image instance is *not* empty, you may rather use the slightly faster function `_cubic_atX(float, int, int, int)`.

Warning

- There is *no* boundary checking performed for the Y,Z and C-coordinates, so they must be inside image bounds.

8.1.4.134 cubic_atX_c() [2/2]

```
template<typename T >
T cubic_atX_c (
    const float fx,
    const int y,
    const int z,
    const int c ) const
```

Return clamped pixel value, using cubic interpolation and Neumann boundary conditions for the X-coordinate.

Similar to [cubic_atX\(float,int,int,int\) const](#), except that the return value is clamped to stay in the min/max range of the datatype T.

8.1.4.135 cubic_atXY() [1/2]

```
template<typename T >
Tfloat cubic_atXY (
    const float fx,
    const float fy,
    const int z,
    const int c,
    const T & out_value ) const
```

Return pixel value, using cubic interpolation and Dirichlet boundary conditions for the X and Y-coordinates.

Similar to [cubic_atX\(float,int,int,int,const T\) const](#), except that the cubic interpolation and boundary checking are achieved both for X and Y-coordinates.

8.1.4.136 cubic_atXY_c() [1/2]

```
template<typename T >
T cubic_atXY_c (
    const float fx,
    const float fy,
    const int z,
    const int c,
    const T & out_value ) const
```

Return clamped pixel value, using cubic interpolation and Dirichlet boundary conditions for the X,Y-coordinates.

Similar to [cubic_atXY\(float,float,int,int,const T\) const](#), except that the return value is clamped to stay in the min/max range of the datatype T.

8.1.4.137 cubic_atXY() [2/2]

```
template<typename T >
Tfloat cubic_atXY (
    const float fx,
    const float fy,
    const int z = 0,
    const int c = 0 ) const
```

Return pixel value, using cubic interpolation and Neumann boundary conditions for the X and Y-coordinates.

Similar to [cubic_atX\(float,int,int,int\) const](#), except that the cubic interpolation and boundary checking are achieved for both X and Y-coordinates.

Note

- If you know your image instance is *not* empty, you may rather use the slightly faster function `_cubic_atXY(float, float, int, int)`.

8.1.4.138 cubic_atXY_c() [2/2]

```
template<typename T >
T cubic_atXY_c (
    const float fx,
    const float fy,
    const int z,
    const int c ) const
```

Return clamped pixel value, using cubic interpolation and Neumann boundary conditions for the X,Y-coordinates.

Similar to `cubic_atXY(float,float,int,int) const`, except that the return value is clamped to stay in the min/max range of the datatype T.

8.1.4.139 cubic_atXYZ() [1/2]

```
template<typename T >
Tfloat cubic_atXYZ (
    const float fx,
    const float fy,
    const float fz,
    const int c,
    const T & out_value ) const
```

Return pixel value, using cubic interpolation and Dirichlet boundary conditions for the X,Y and Z-coordinates.

Similar to `cubic_atX(float,int,int,int,const T) const`, except that the cubic interpolation and boundary checking are achieved both for X,Y and Z-coordinates.

8.1.4.140 cubic_atXYZ_c() [1/2]

```
template<typename T >
T cubic_atXYZ_c (
    const float fx,
    const float fy,
    const float fz,
    const int c,
    const T & out_value ) const
```

Return clamped pixel value, using cubic interpolation and Dirichlet boundary conditions for the XYZ-coordinates.

Similar to `cubic_atXYZ(float,float,float,int,const T) const`, except that the return value is clamped to stay in the min/max range of the datatype T.

8.1.4.141 cubic_atXYZ() [2/2]

```
template<typename T >
Tfloat cubic_atXYZ (
    const float fx,
    const float fy,
    const float fz,
    const int c = 0 ) const
```

Return pixel value, using cubic interpolation and Neumann boundary conditions for the X,Y and Z-coordinates.

Similar to `cubic_atX(float,int,int,int) const`, except that the cubic interpolation and boundary checking are achieved both for X,Y and Z-coordinates.

Note

- If you know your image instance is *not* empty, you may rather use the slightly faster function `_cubic_atXYZ(float, float, float, int)`.

8.1.4.142 cubic_atXYZ_c() [2/2]

```
template<typename T >
T cubic_atXYZ_c (
    const float fx,
    const float fy,
    const float fz,
    const int c ) const
```

Return clamped pixel value, using cubic interpolation and Neumann boundary conditions for the XYZ-coordinates.

Similar to `cubic_atXYZ(float,float,float,int) const`, except that the return value is clamped to stay in the min/max range of the datatype T.

8.1.4.143 cubic_atXYZ_p()

```
template<typename T >
Tfloat cubic_atXYZ_p (
    const float fx,
    const float fy,
    const float fz,
    const int c = 0 ) const
```

Return pixel value, using cubic interpolation and Neumann boundary conditions for the X,Y and Z-coordinates.

Similar to `cubic_atX(float,int,int,int) const`, except that the cubic interpolation and boundary checking are achieved both for X,Y and Z-coordinates.

Note

- If you know your image instance is *not* empty, you may rather use the slightly faster function `_cubic_atXYZ(float,float,float,int)`.

8.1.4.144 set_linear_atX()

```
template<typename T >
CImg< T > & set_linear_atX (
    const T & value,
    const float fx,
    const int y = 0,
    const int z = 0,
    const int c = 0,
    const bool is_added = false )
```

Set pixel value, using linear interpolation for the X-coordinates.

Set pixel value at specified coordinates (fx,y,z,c) in the image instance, in a way that the value is spread amongst several neighbors if the pixel coordinates are float-valued.

Parameters

<i>value</i>	Pixel value to set.
<i>fx</i>	X-coordinate of the pixel value (float-valued).
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value. Generated by Doxygen
<i>c</i>	C-coordinate of the pixel value.
<i>is_added</i>	Indicates whether the pixel value is added to (<code>true</code>), or simply replace (<code>false</code>) the current image pixel(s).

Returns

A reference to the current image instance.

Note

- Calling this function with out-of-bounds coordinates does nothing.

8.1.4.145 set_linear_atXY()

```
template<typename T >
CImg< T > & set_linear_atXY (
    const T & value,
    const float fx,
    const float fy = 0,
    const int z = 0,
    const int c = 0,
    const bool is_added = false )
```

Set pixel value, using linear interpolation for the X and Y-coordinates.

Similar to [set_linear_atX\(const T&,float,int,int,int,bool\)](#), except that the linear interpolation is achieved both for X and Y-coordinates.

8.1.4.146 set_linear_atXYZ()

```
template<typename T >
CImg< T > & set_linear_atXYZ (
    const T & value,
    const float fx,
    const float fy = 0,
    const float fz = 0,
    const int c = 0,
    const bool is_added = false )
```

Set pixel value, using linear interpolation for the X,Y and Z-coordinates.

Similar to [set_linear_atXY\(const T&,float,float,int,int,bool\)](#), except that the linear interpolation is achieved both for X,Y and Z-coordinates.

8.1.4.147 value_string()

```
template<typename T >
CImg< charT > value_string (
    const char separator = ',',
    const unsigned int max_size = 0,
    const char *const format = 0 ) const
```

Return a C-string containing a list of all values of the image instance.

Return a new CImg<char> image whose buffer [data\(\)](#) is a char* string describing the list of all pixel values of the image instance (written in base 10), separated by specified separator character.

Parameters

<i>separator</i>	A <code>char</code> character which specifies the separator between values in the returned C-string.
<i>max_size</i>	Maximum size of the returned image (or 0 if no limits are set).
<i>format</i>	For float/double-values, tell the printf format used to generate the text representation of the numbers (or 0 for default representation).

Note

- The returned image is never empty.
- For an empty image instance, the returned string is " ".
- If `max_size` is equal to 0, there are no limits on the size of the returned string.
- Otherwise, if the maximum number of string characters is exceeded, the value string is cut off and terminated by character `'\0'`. In that case, the returned image size is `max_size + 1`.

8.1.4.148 is_shared()

```
template<typename T >
bool is_shared ( ) const
```

Test shared state of the pixel buffer.

Return `true` if image instance has a shared memory buffer, and `false` otherwise.

Note

- A shared image does not own its pixel buffer `data()` and will not deallocate it on destruction.
- Most of the time, a `CImg<T>` image instance will *not* be shared.
- A shared image can only be obtained by a limited set of constructors and functions (see list below).

8.1.4.149 is_empty()

```
template<typename T >
bool is_empty ( ) const
```

Test if image instance is empty.

Return `true` if image instance is empty, i.e. does *not* contain any pixel values, has dimensions `0 x 0 x 0 x 0` and a pixel buffer pointer set to 0 (null pointer), and `false` otherwise.

8.1.4.150 is_inf()

```
template<typename T >
bool is_inf ( ) const
```

Test if image instance contains a 'inf' value.

Return `true` if image instance contains a 'inf' value, and `false` otherwise.

8.1.4.151 is_nan()

```
template<typename T >
bool is_nan ( ) const
```

Test if image instance contains a NaN value.

Return `true` if image instance contains a NaN value, and `false` otherwise.

8.1.4.152 is_sameXY() [1/3]

```
template<typename T >
bool is_sameXY (
    const unsigned int size_x,
    const unsigned int size_y ) const
```

Test if image width and height are equal to specified values.

Test if `is_sameX(unsigned int) const` and `is_sameY(unsigned int) const` are both verified.

8.1.4.153 is_sameXY() [2/3]

```
template<typename T >
template<typename t >
bool is_sameXY (
    const CImg< t > & img ) const
```

Test if image width and height are the same as those of another image.

Test if `is_sameX(const CImg<t>&) const` and `is_sameY(const CImg<t>&) const` are both verified.

8.1.4.154 is_sameXY() [3/3]

```
template<typename T >
bool is_sameXY (
    const CImgDisplay & disp ) const
```

Test if image width and height are the same as that of an existing display window.

Test if `is_sameX(const CImgDisplay&) const` and `is_sameY(const CImgDisplay&) const` are both verified.

8.1.4.155 is_sameXZ() [1/2]

```
template<typename T >
bool is_sameXZ (
    const unsigned int size_x,
    const unsigned int size_z ) const
```

Test if image width and depth are equal to specified values.

Test if `is_sameX(unsigned int) const` and `is_sameZ(unsigned int) const` are both verified.

8.1.4.156 is_sameXZ() [2/2]

```
template<typename T >
template<typename t >
bool is_sameXZ (
    const CImg< t > & img ) const
```

Test if image width and depth are the same as those of another image.

Test if [is_sameX\(const CImg<t>&\) const](#) and [is_sameZ\(const CImg<t>&\) const](#) are both verified.

8.1.4.157 is_sameXC() [1/2]

```
template<typename T >
bool is_sameXC (
    const unsigned int size_x,
    const unsigned int size_c ) const
```

Test if image width and spectrum are equal to specified values.

Test if [is_sameX\(unsigned int\) const](#) and [is_sameC\(unsigned int\) const](#) are both verified.

8.1.4.158 is_sameXC() [2/2]

```
template<typename T >
template<typename t >
bool is_sameXC (
    const CImg< t > & img ) const
```

Test if image width and spectrum are the same as those of another image.

Test if [is_sameX\(const CImg<t>&\) const](#) and [is_sameC\(const CImg<t>&\) const](#) are both verified.

8.1.4.159 is_sameYZ() [1/2]

```
template<typename T >
bool is_sameYZ (
    const unsigned int size_y,
    const unsigned int size_z ) const
```

Test if image height and depth are equal to specified values.

Test if [is_sameY\(unsigned int\) const](#) and [is_sameZ\(unsigned int\) const](#) are both verified.

8.1.4.160 is_sameYZ() [2/2]

```
template<typename T >
template<typename t >
bool is_sameYZ (
    const CImg< t > & img ) const
```

Test if image height and depth are the same as those of another image.

Test if [is_sameY\(const CImg<t>&\) const](#) and [is_sameZ\(const CImg<t>&\) const](#) are both verified.

8.1.4.161 is_sameYC() [1/2]

```
template<typename T >
bool is_sameYC (
    const unsigned int size_y,
    const unsigned int size_c ) const
```

Test if image height and spectrum are equal to specified values.

Test if `is_sameY(unsigned int) const` and `is_sameC(unsigned int) const` are both verified.

8.1.4.162 is_sameYC() [2/2]

```
template<typename T >
template<typename t >
bool is_sameYC (
    const CImg< t > & img ) const
```

Test if image height and spectrum are the same as those of another image.

Test if `is_sameY(const CImg<t>&) const` and `is_sameC(const CImg<t>&) const` are both verified.

8.1.4.163 is_sameZC() [1/2]

```
template<typename T >
bool is_sameZC (
    const unsigned int size_z,
    const unsigned int size_c ) const
```

Test if image depth and spectrum are equal to specified values.

Test if `is_sameZ(unsigned int) const` and `is_sameC(unsigned int) const` are both verified.

8.1.4.164 is_sameZC() [2/2]

```
template<typename T >
template<typename t >
bool is_sameZC (
    const CImg< t > & img ) const
```

Test if image depth and spectrum are the same as those of another image.

Test if `is_sameZ(const CImg<t>&) const` and `is_sameC(const CImg<t>&) const` are both verified.

8.1.4.165 is_sameXYZ() [1/2]

```
template<typename T >
bool is_sameXYZ (
    const unsigned int size_x,
    const unsigned int size_y,
    const unsigned int size_z ) const
```

Test if image width, height and depth are equal to specified values.

Test if `is_sameXY(unsigned int,unsigned int) const` and `is_sameZ(unsigned int) const` are both verified.

8.1.4.166 is_sameXYZ() [2/2]

```
template<typename T >
template<typename t >
bool is_sameXYZ (
    const CImg< t > & img ) const
```

Test if image width, height and depth are the same as those of another image.

Test if [is_sameXY\(const CImg<t>&\) const](#) and [is_sameZ\(const CImg<t>&\) const](#) are both verified.

8.1.4.167 is_sameXYC() [1/2]

```
template<typename T >
bool is_sameXYC (
    const unsigned int size_x,
    const unsigned int size_y,
    const unsigned int size_c ) const
```

Test if image width, height and spectrum are equal to specified values.

Test if [is_sameXY\(unsigned int,unsigned int\) const](#) and [is_sameC\(unsigned int\) const](#) are both verified.

8.1.4.168 is_sameXYC() [2/2]

```
template<typename T >
template<typename t >
bool is_sameXYC (
    const CImg< t > & img ) const
```

Test if image width, height and spectrum are the same as those of another image.

Test if [is_sameXY\(const CImg<t>&\) const](#) and [is_sameC\(const CImg<t>&\) const](#) are both verified.

8.1.4.169 is_sameXZC() [1/2]

```
template<typename T >
bool is_sameXZC (
    const unsigned int size_x,
    const unsigned int size_z,
    const unsigned int size_c ) const
```

Test if image width, depth and spectrum are equal to specified values.

Test if [is_sameXZ\(unsigned int,unsigned int\) const](#) and [is_sameC\(unsigned int\) const](#) are both verified.

8.1.4.170 is_sameXZC() [2/2]

```
template<typename T >
template<typename t >
bool is_sameXZC (
    const CImg< t > & img ) const
```

Test if image width, depth and spectrum are the same as those of another image.

Test if [is_sameXZ\(const CImg<t>&\) const](#) and [is_sameC\(const CImg<t>&\) const](#) are both verified.

8.1.4.171 is_sameYZC() [1/2]

```
template<typename T >
bool is_sameYZC (
    const unsigned int size_y,
    const unsigned int size_z,
    const unsigned int size_c ) const
```

Test if image height, depth and spectrum are equal to specified values.

Test if `is_sameYZ(unsigned int,unsigned int) const` and `is_sameC(unsigned int) const` are both verified.

8.1.4.172 is_sameYZC() [2/2]

```
template<typename T >
template<typename t >
bool is_sameYZC (
    const CImg< t > & img ) const
```

Test if image height, depth and spectrum are the same as those of another image.

Test if `is_sameYZ(const CImg<t>&) const` and `is_sameC(const CImg<t>&) const` are both verified.

8.1.4.173 is_sameXYZC() [1/2]

```
template<typename T >
bool is_sameXYZC (
    const unsigned int size_x,
    const unsigned int size_y,
    const unsigned int size_z,
    const unsigned int size_c ) const
```

Test if image width, height, depth and spectrum are equal to specified values.

Test if `is_sameXYZ(unsigned int,unsigned int,unsigned int) const` and `is_sameC(unsigned int) const` are both verified.

8.1.4.174 is_sameXYZC() [2/2]

```
template<typename T >
template<typename t >
bool is_sameXYZC (
    const CImg< t > & img ) const
```

Test if image width, height, depth and spectrum are the same as those of another image.

Test if `is_sameXYZ(const CImg<t>&) const` and `is_sameC(const CImg<t>&) const` are both verified.

8.1.4.175 containsXYZC()

```
template<typename T >
bool containsXYZC (
    const int x,
    const int y = 0,
    const int z = 0,
    const int c = 0 ) const
```

Test if specified coordinates are inside image bounds.

Return `true` if pixel located at (x,y,z,c) is inside bounds of the image instance, and `false` otherwise.

Parameters

<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.

Note

- Return `true` only if all these conditions are verified:
 - The image instance is *not* empty.
 - `0<=x<=width()` - 1.
 - `0<=y<=height()` - 1.
 - `0<=z<=depth()` - 1.
 - `0<=c<=spectrum()` - 1.

8.1.4.176 `contains()` [1/5]

```
template<typename T >
template<typename t >
bool contains (
    const T & pixel,
    t & x,
    t & y,
    t & z,
    t & c ) const
```

Test if pixel value is inside image bounds and get its X,Y,Z and C-coordinates.

Return `true` if specified reference refers to a pixel value inside bounds of the image instance, and `false` otherwise.

Parameters

	<i>pixel</i>	Reference to pixel value to test.
out	<i>x</i>	X-coordinate of the pixel value, if test succeeds.
out	<i>y</i>	Y-coordinate of the pixel value, if test succeeds.
out	<i>z</i>	Z-coordinate of the pixel value, if test succeeds.
out	<i>c</i>	C-coordinate of the pixel value, if test succeeds.

Note

- Useful to convert an offset to a buffer value into pixel value coordinates:


```
const CImg<float> img(100,100,1,3); // Construct a 100x100 RGB color image
const unsigned long offset = 1249; // Offset to the pixel (49,12,0,0)
unsigned int x,y,z,c;
if (img.contains(img[offset],x,y,z,c)) { // Convert offset to (x,y,z,c) coordinates
    std::printf("Offset %u refers to pixel located at (%u,%u,%u,%u).\n",
                offset,x,y,z,c);
}
```

8.1.4.177 contains() [2/5]

```
template<typename T >
template<typename t >
bool contains (
    const T & pixel,
    t & x,
    t & y,
    t & z ) const
```

Test if pixel value is inside image bounds and get its X,Y and Z-coordinates.

Similar to [contains\(const T&,t&,t&,t&\) const](#), except that only the X,Y and Z-coordinates are set.

8.1.4.178 contains() [3/5]

```
template<typename T >
template<typename t >
bool contains (
    const T & pixel,
    t & x,
    t & y ) const
```

Test if pixel value is inside image bounds and get its X and Y-coordinates.

Similar to [contains\(const T&,t&,t&,t&\) const](#), except that only the X and Y-coordinates are set.

8.1.4.179 contains() [4/5]

```
template<typename T >
template<typename t >
bool contains (
    const T & pixel,
    t & x ) const
```

Test if pixel value is inside image bounds and get its X-coordinate.

Similar to [contains\(const T&,t&,t&,t&\) const](#), except that only the X-coordinate is set.

8.1.4.180 contains() [5/5]

```
template<typename T >
bool contains (
    const T & pixel ) const
```

Test if pixel value is inside image bounds.

Similar to [contains\(const T&,t&,t&,t&\) const](#), except that no pixel coordinates are set.

8.1.4.181 is_overlapped()

```
template<typename T >
template<typename t >
bool is_overlapped (
    const CImg< t > & img ) const
```

Test if pixel buffers of instance and input images overlap.

Return `true` if pixel buffers attached to image instance and input image `img` overlap, and `false` otherwise.

Parameters

<i>img</i>	Input image to compare with.
------------	------------------------------

Note

- Buffer overlapping may happen when manipulating *shared* images.
- If two image buffers overlap, operating on one of the image will probably modify the other one.
- Most of the time, `CImg<T>` instances are *non-shared* and do not overlap between each others.

Example

```
const CImg<float>
img1("reference.jpg"), // Load RGB-color image
img2 = img1.get_shared_channel(1); // Get shared version of the green channel
if (img1.is_overlapped(img2)) { // Test succeeds, 'img1' and 'img2' overlaps
    std::printf("Buffers overlap!\n");
}
```

8.1.4.182 `is_object3d()`

```
template<typename T >
template<typename tp , typename tc , typename to >
bool is_object3d (
    const CImgList< tp > & primitives,
    const CImgList< tc > & colors,
    const to & opacities,
    const bool full_check = true,
    char *const error_message = 0 ) const
```

Test if the set `{*this,primitives,colors,opacities}` defines a valid 3D object.

Return `true` if the 3D object represented by the set `{*this,primitives,colors,opacities}` is valid, and `false` otherwise. The vertex coordinates are defined by the instance image.

Parameters

	<i>primitives</i>	List of primitives of the 3D object.
	<i>colors</i>	List of colors of the 3D object.
	<i>opacities</i>	List (or image) of opacities of the 3D object.
	<i>full_check</i>	Indicates whether full checking of the 3D object must be performed.
out	<i>error_message</i>	C-string to contain the error message, if the test does not succeed (at least 256 bytes).

Note

- Set `full_checking` to `false` to speed-up the 3D object checking. In this case, only the size of each 3D object component is checked.
- Size of the string `error_message` should be at least 128-bytes long, to be able to contain the error message.

8.1.4.183 is_CImg3d()

```
template<typename T >
bool is_CImg3d (
    const bool full_check = true,
    char *const error_message = 0 ) const
```

Test if image instance represents a valid serialization of a 3D object.

Return `true` if the image instance represents a valid serialization of a 3D object, and `false` otherwise.

Parameters

	<code>full_check</code>	Indicates whether full checking of the instance must be performed.
<code>out</code>	<code>error_message</code>	C-string to contain the error message, if the test does not succeed.

Note

- Set `full_check` to `false` to speed-up the 3D object checking. In this case, only the size of each 3D object component is checked.
- Size of the string `error_message` should be at least 256-bytes long, to be able to contain the error message.

8.1.4.184 sqr()

```
template<typename T >
CImg< T > & sqr ( )
```

Compute the square value of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its square value $I_{(x,y,z,c)}^2$.

Note

- The **[in-place version]** of this function statically casts the computed values to the pixel type `T`.
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type `T` is *not* float-valued.

Example

```
const CImg<float> img("reference.jpg");
(img, img.get_sqr().normalize(0,255)).display();
```

8.1.4.185 sqrt()

```
template<typename T >
CImg< T > & sqrt ( )
```

Compute the square root of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its square root $\sqrt{I_{(x,y,z,c)}}$.

Note

- The **[in-place version]** of this function statically casts the computed values to the pixel type `T`.
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type `T` is *not* float-valued.

Example

```
const CImg<float> img("reference.jpg");
(img, img.get_sqrt().normalize(0,255)).display();
```

8.1.4.186 exp()

```
template<typename T >
CImg< T > & exp ( )
```

Compute the exponential of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its exponential $e^{I_{(x,y,z,c)}}$.

Note

- The **[in-place version]** of this function statically casts the computed values to the pixel type T.
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type T is *not* float-valued.

8.1.4.187 log()

```
template<typename T >
CImg< T > & log ( )
```

Compute the error function of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its error function.

Note

- The **[in-place version]** of this function statically casts the computed values to the pixel type T.
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type T is *not* float-valued. Compute the logarithm of each pixel value. Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its logarithm $\log_e(I_{(x,y,z,c)})$.
- The **[in-place version]** of this function statically casts the computed values to the pixel type T.
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type T is *not* float-valued.

8.1.4.188 log2()

```
template<typename T >
CImg< T > & log2 ( )
```

Compute the base-2 logarithm of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its base-2 logarithm $\log_2(I_{(x,y,z,c)})$.

Note

- The **[in-place version]** of this function statically casts the computed values to the pixel type T.
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type T is *not* float-valued.

8.1.4.189 log10()

```
template<typename T >
CImg< T > & log10 ( )
```

Compute the base-10 logarithm of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its base-10 logarithm $\log_{10}(I_{(x,y,z,c)})$.

Note

- The **[in-place version]** of this function statically casts the computed values to the pixel type T.
- The **[new-instance version]** returns a CImg<float> image, if the pixel type T is *not* float-valued.

8.1.4.190 abs()

```
template<typename T >
CImg< T > & abs ( )
```

Compute the absolute value of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its absolute value $|I_{(x,y,z,c)}|$.

Note

- The **[in-place version]** of this function statically casts the computed values to the pixel type T.
- The **[new-instance version]** returns a CImg<float> image, if the pixel type T is *not* float-valued.

8.1.4.191 sign()

```
template<typename T >
CImg< T > & sign ( )
```

Compute the sign of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its sign $\text{sign}(I_{(x,y,z,c)})$.

Note

- The sign is set to:
 - 1 if pixel value is strictly positive.
 - -1 if pixel value is strictly negative.
 - 0 if pixel value is equal to 0.
- The **[in-place version]** of this function statically casts the computed values to the pixel type T.
- The **[new-instance version]** returns a CImg<float> image, if the pixel type T is *not* float-valued.

8.1.4.192 cos()

```
template<typename T >  
CImg< T > & cos ( )
```

Compute the cosine of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its cosine $\cos(I_{(x,y,z,c)})$.

Note

- Pixel values are regarded as being in *radians*.
- The **[in-place version]** of this function statically casts the computed values to the pixel type T.
- The **[new-instance version]** returns a CImg<float> image, if the pixel type T is *not* float-valued.

8.1.4.193 sin()

```
template<typename T >  
CImg< T > & sin ( )
```

Compute the sine of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its sine $\sin(I_{(x,y,z,c)})$.

Note

- Pixel values are regarded as being in *radians*.
- The **[in-place version]** of this function statically casts the computed values to the pixel type T.
- The **[new-instance version]** returns a CImg<float> image, if the pixel type T is *not* float-valued.

8.1.4.194 sinc()

```
template<typename T >  
CImg< T > & sinc ( )
```

Compute the sinc of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its sinc $\text{sinc}(I_{(x,y,z,c)})$.

Note

- Pixel values are regarded as being in *radians*.
- The **[in-place version]** of this function statically casts the computed values to the pixel type T.
- The **[new-instance version]** returns a CImg<float> image, if the pixel type T is *not* float-valued.

8.1.4.195 tan()

```
template<typename T >  
CImg< T > & tan ( )
```

Compute the tangent of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its tangent $\tan(I_{(x,y,z,c)})$.

Note

- Pixel values are regarded as being in *radians*.
- The **[in-place version]** of this function statically casts the computed values to the pixel type T.
- The **[new-instance version]** returns a CImg<float> image, if the pixel type T is *not* float-valued.

8.1.4.196 cosh()

```
template<typename T >  
CImg< T > & cosh ( )
```

Compute the hyperbolic cosine of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its hyperbolic cosine $\cosh(I_{(x,y,z,c)})$.

Note

- The **[in-place version]** of this function statically casts the computed values to the pixel type T.
- The **[new-instance version]** returns a CImg<float> image, if the pixel type T is *not* float-valued.

8.1.4.197 sinh()

```
template<typename T >  
CImg< T > & sinh ( )
```

Compute the hyperbolic sine of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its hyperbolic sine $\sinh(I_{(x,y,z,c)})$.

Note

- The **[in-place version]** of this function statically casts the computed values to the pixel type T.
- The **[new-instance version]** returns a CImg<float> image, if the pixel type T is *not* float-valued.

8.1.4.198 tanh()

```
template<typename T >
CImg< T > & tanh ( )
```

Compute the hyperbolic tangent of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its hyperbolic tangent $\tanh(I_{(x,y,z,c)})$.

Note

- The **[in-place version]** of this function statically casts the computed values to the pixel type T .
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type T is *not* float-valued.

8.1.4.199 acos()

```
template<typename T >
CImg< T > & acos ( )
```

Compute the arccosine of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its arccosine $\arccos(I_{(x,y,z,c)})$.

Note

- The **[in-place version]** of this function statically casts the computed values to the pixel type T .
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type T is *not* float-valued.

8.1.4.200 asin()

```
template<typename T >
CImg< T > & asin ( )
```

Compute the arcsine of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its arcsine $\arcsin(I_{(x,y,z,c)})$.

Note

- The **[in-place version]** of this function statically casts the computed values to the pixel type T .
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type T is *not* float-valued.

8.1.4.201 atan()

```
template<typename T >
CImg< T > & atan ( )
```

Compute the arctangent of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its arctangent $\arctan(I_{(x,y,z,c)})$.

Note

- The **[in-place version]** of this function statically casts the computed values to the pixel type T .
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type T is *not* float-valued.

8.1.4.202 atan2()

```
template<typename T >
template<typename t >
CImg< T > & atan2 (
    const CImg< t > & img )
```

Compute the arctangent2 of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its arctangent2 $\text{atan2}(I_{(x,y,z,c)})$.

Parameters

<i>img</i>	Image whose pixel values specify the second argument of the <code>atan2()</code> function.
------------	--

Note

- The **[in-place version]** of this function statically casts the computed values to the pixel type `T`.
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type `T` is *not* float-valued.

Example

```
const CImg<float>
img_x(100,100,1,1,"x-w/2",false), // Define an horizontal centered gradient, from '-width/2' to
'width/2'
img_y(100,100,1,1,"y-h/2",false), // Define a vertical centered gradient, from '-height/2' to
'height/2'
img_atan2 = img_y.get_atan2(img_x); // Compute atan2(y,x) for each pixel value
(img_x,img_y,img_atan2).display();
```

8.1.4.203 acosh()

```
template<typename T >
CImg< T > & acosh ( )
```

Compute the hyperbolic arccosine of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its arccosineh $\text{acosh}(I_{(x,y,z,c)})$.

Note

- The **[in-place version]** of this function statically casts the computed values to the pixel type `T`.
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type `T` is *not* float-valued.

8.1.4.204 asinh()

```
template<typename T >
CImg< T > & asinh ( )
```

Compute the hyperbolic arcsine of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its hyperbolic arcsine $\text{asinh}(I_{(x,y,z,c)})$.

Note

- The **[in-place version]** of this function statically casts the computed values to the pixel type `T`.
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type `T` is *not* float-valued.

8.1.4.205 atanh()

```
template<typename T >
CImg< T > & atanh ( )
```

Compute the hyperbolic arctangent of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its hyperbolic arctangent $\operatorname{atanh}(I_{(x,y,z,c)})$.

Note

- The **[in-place version]** of this function statically casts the computed values to the pixel type `T`.
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type `T` is *not* float-valued.

8.1.4.206 mul()

```
template<typename T >
template<typename t >
CImg< T > & mul (
    const CImg< t > & img )
```

In-place pointwise multiplication.

Compute the pointwise multiplication between the image instance and the specified input image `img`.

Parameters

<code>img</code>	Input image, as the second operand of the multiplication.
------------------	---

Note

- Similar to `operator+=(const CImg<t>&)`, except that it performs a pointwise multiplication instead of an addition.
- It does *not* perform a *matrix* multiplication. For this purpose, use `operator*=(const CImg<t>&)` instead.

Example

```
CImg<float>
img("reference.jpg"),
shade(img.width,img.height(),1,1,"-(x-w/2)^2-(y-h/2)^2",false);
shade.normalize(0,1);
(img,shade,img.get_mul(shade)).display();
```

8.1.4.207 div()

```
template<typename T >
template<typename t >
CImg< T > & div (
    const CImg< t > & img )
```

In-place pointwise division.

Similar to `mul(const CImg<t>&)`, except that it performs a pointwise division instead of a multiplication.

8.1.4.208 pow() [1/3]

```
template<typename T >
CImg< T > & pow (
    const double p )
```

Raise each pixel value to a specified power.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its power $I_{(x,y,z,c)}^p$.

Parameters

p	Exponent value.
-----	-----------------

Note

- The **[in-place version]** of this function statically casts the computed values to the pixel type T.
- The **[new-instance version]** returns a CImg<float> image, if the pixel type T is *not* float-valued.

Example

```
const CImg<float>
img0("reference.jpg"), // Load reference color image
img1 = (img0/255).pow(1.8)*=255, // Compute gamma correction, with gamma = 1.8
img2 = (img0/255).pow(0.5)*=255; // Compute gamma correction, with gamma = 0.5
(img0,img1,img2).display();
```

8.1.4.209 pow() [2/3]

```
template<typename T >
CImg< T > & pow (
    const char *const expression )
```

Raise each pixel value to a power, specified from an expression.

Similar to `operator+=(const char*)`, except it performs a pointwise exponentiation instead of an addition.

8.1.4.210 pow() [3/3]

```
template<typename T >
template<typename t >
CImg< T > & pow (
    const CImg< t > & img )
```

Raise each pixel value to a power, pointwisely specified from another image.

Similar to `operator+=(const CImg<t>& img)`, except that it performs an exponentiation instead of an addition.

8.1.4.211 rol() [1/3]

```
template<typename T >
CImg< T > & rol (
    const unsigned int n = 1 )
```

Compute the bitwise left rotation of each pixel value.

Similar to `operator<<=(unsigned int)`, except that it performs a left rotation instead of a left shift.

8.1.4.212 rol() [2/3]

```
template<typename T >
CImg< T > & rol (
    const char *const expression )
```

Compute the bitwise left rotation of each pixel value.

Similar to `operator<<=(const char*)`, except that it performs a left rotation instead of a left shift.

8.1.4.213 rol() [3/3]

```
template<typename T >
template<typename t >
CImg< T > & rol (
    const CImg< t > & img )
```

Compute the bitwise left rotation of each pixel value.

Similar to `operator<<=(const CImg<t>&)`, except that it performs a left rotation instead of a left shift.

8.1.4.214 ror() [1/3]

```
template<typename T >
CImg< T > & ror (
    const unsigned int n = 1 )
```

Compute the bitwise right rotation of each pixel value.

Similar to `operator>>=(unsigned int)`, except that it performs a right rotation instead of a right shift.

8.1.4.215 ror() [2/3]

```
template<typename T >
CImg< T > & ror (
    const char *const expression )
```

Compute the bitwise right rotation of each pixel value.

Similar to `operator>>=(const char*)`, except that it performs a right rotation instead of a right shift.

8.1.4.216 ror() [3/3]

```
template<typename T >
template<typename t >
CImg< T > & ror (
    const CImg< t > & img )
```

Compute the bitwise right rotation of each pixel value.

Similar to `operator>>=(const CImg<t>&)`, except that it performs a right rotation instead of a right shift.

8.1.4.217 min() [1/3]

```
template<typename T >
CImg< T > & min (
    const T & value )
```

Pointwise min operator between instance image and a value.

Parameters

<i>value</i>	Value used as the reference argument of the min operator.
--------------	---

Note

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by $\min(I_{(x,y,z,c)}, \text{val})$.

8.1.4.218 min() [2/3]

```
template<typename T >
template<typename t >
CImg< T > & min (
    const CImg< t > & img )
```

Pointwise min operator between two images.

Parameters

<i>img</i>	Image used as the reference argument of the min operator.
------------	---

Note

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by $\min(I_{(x,y,z,c)}, \text{img}_{(x,y,z,c)})$.

8.1.4.219 min() [3/3]

```
template<typename T >
CImg< T > & min (
    const char *const expression )
```

Pointwise min operator between an image and an expression.

Parameters

<i>expression</i>	Math formula as a C-string.
-------------------	-----------------------------

Note

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by $\min(I_{(x,y,z,c)}, \text{expr}_{(x,y,z,c)})$.

8.1.4.220 max() [1/3]

```
template<typename T >
CImg< T > & max (
    const T & value )
```

Pointwise max operator between instance image and a value.

Parameters

<i>value</i>	Value used as the reference argument of the max operator.
--------------	---

Note

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by $\max(I_{(x,y,z,c)}, \text{val})$.

8.1.4.221 max() [2/3]

```
template<typename T >
template<typename t >
CImg< T > & max (
    const CImg< t > & img )
```

Pointwise max operator between two images.

Parameters

<i>img</i>	Image used as the reference argument of the max operator.
------------	---

Note

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by $\max(I_{(x,y,z,c)}, \text{img}_{(x,y,z,c)})$.

8.1.4.222 max() [3/3]

```
template<typename T >
CImg< T > & max (
    const char *const expression )
```

Pointwise max operator between an image and an expression.

Parameters

<i>expression</i>	Math formula as a C-string.
-------------------	-----------------------------

Note

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by $\max(I_{(x,y,z,c)}, \text{expr}_{(x,y,z,c)})$.

8.1.4.223 minabs() [1/3]

```
template<typename T >
CImg< T > & minabs (
    const T & value )
```

Pointwise minabs operator between instance image and a value.

Parameters

<i>value</i>	Value used as the reference argument of the minabs operator.
--------------	--

Note

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by $\minabs(I_{(x,y,z,c)}, \text{val})$.

8.1.4.224 minabs() [2/3]

```
template<typename T >
template<typename t >
CImg< T > & minabs (
    const CImg< t > & img )
```

Pointwise minabs operator between two images.

Parameters

<i>img</i>	Image used as the reference argument of the minabs operator.
------------	--

Note

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by $\minabs(I_{(x,y,z,c)}, \text{img}_{(x,y,z,c)})$.

8.1.4.225 minabs() [3/3]

```
template<typename T >
CImg< T > & minabs (
    const char *const expression )
```

Pointwise minabs operator between an image and an expression.

Parameters

<i>expression</i>	Math formula as a C-string.
-------------------	-----------------------------

Note

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by $\minabs(I_{(x,y,z,c)}, \text{expr}_{(x,y,z,c)})$.

8.1.4.226 maxabs() [1/3]

```
template<typename T >
CImg< T > & maxabs (
    const T & value )
```

Pointwise maxabs operator between instance image and a value.

Parameters

<i>value</i>	Value used as the reference argument of the maxabs operator.
--------------	--

Note

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by $\maxabs(I_{(x,y,z,c)}, \text{val})$.

8.1.4.227 maxabs() [2/3]

```
template<typename T >
template<typename t >
CImg< T > & maxabs (
    const CImg< t > & img )
```

Pointwise maxabs operator between two images.

Parameters

<i>img</i>	Image used as the reference argument of the maxabs operator.
------------	--

Note

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by $\maxabs(I_{(x,y,z,c)}, \text{img}_{(x,y,z,c)})$.

8.1.4.228 maxabs() [3/3]

```
template<typename T >
CImg< T > & maxabs (
    const char *const expression )
```

Pointwise maxabs operator between an image and an expression.

Parameters

<i>expression</i>	Math formula as a C-string.
-------------------	-----------------------------

Note

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by $\maxabs(I_{(x,y,z,c)}, \text{expr}_{(x,y,z,c)})$.

8.1.4.229 min_max()

```
template<typename T >
template<typename t >
T & min_max (
    t & max_val )
```

Return a reference to the minimum pixel value as well as the maximum pixel value.

Parameters

out	<i>max_val</i>	Maximum pixel value.
-----	----------------	----------------------

8.1.4.230 max_min()

```
template<typename T >
template<typename t >
T & max_min (
    t & min_val )
```

Return a reference to the maximum pixel value as well as the minimum pixel value.

Parameters

out	<i>min_val</i>	Minimum pixel value.
-----	----------------	----------------------

8.1.4.231 kth_smallest()

```
template<typename T >
T kth_smallest (
    const_ulongT k ) const
```

Return the kth smallest pixel value.

Parameters

<i>k</i>	Rank of the smallest element searched.
----------	--

8.1.4.232 variance()

```
template<typename T >
double variance (
    const unsigned int variance_method = 0 ) const
```

Return the variance of the pixel values.

Parameters

<i>variance_method</i>	Method used to estimate the variance. Can be: • 0: Second moment, computed as $1/N \sum_{k=1}^N (x_k - \bar{x})^2 = 1/N \left(\sum_{k=1}^N x_k^2 - \left(\sum_{k=1}^N x_k \right)^2 / N \right)$ with $\bar{x} = 1/N \sum_{k=1}^N x_k$. • 1: Best unbiased estimator, computed as $\frac{1}{N-1} \sum_{k=1}^N (x_k - \bar{x})^2$. • 2: Least median of squares. • 3: Least trimmed of squares.
------------------------	---

8.1.4.233 variance_mean()

```
template<typename T >
template<typename t >
double variance_mean (
    const unsigned int variance_method,
    t & mean ) const
```

Return the variance as well as the average of the pixel values.

Parameters

	<i>variance_method</i>	Method used to estimate the variance (see variance(const unsigned int) const).
out	<i>mean</i>	Average pixel value.

8.1.4.234 variance_noise()

```
template<typename T >
double variance_noise (
    const unsigned int variance_method = 2 ) const
```

Return estimated variance of the noise.

Parameters

<i>variance_method</i>	Method used to compute the variance (see variance(const unsigned int) const).
------------------------	--

Note

Because of structures such as edges in images it is recommended to use an unbiased variance estimation. The variance of the noise is estimated by computing the variance of the Laplacian $(\Delta I)^2$ scaled by a factor c insuring $cE[(\Delta I)^2] = \sigma^2$ where σ is the noise variance.

8.1.4.235 MSE()

```
template<typename T >
template<typename t >
double MSE (
    const CImg< t > & img ) const
```

Compute the MSE (Mean-Squared Error) between two images.

Parameters

<i>img</i>	Image used as the second argument of the MSE operator.
------------	--

8.1.4.236 PSNR()

```
template<typename T >
template<typename t >
double PSNR (
    const CImg< t > & img,
    const double max_value = 255 ) const
```

Compute the PSNR (Peak Signal-to-Noise Ratio) between two images.

Parameters

<i>img</i>	Image used as the second argument of the PSNR operator.
<i>max_value</i>	Maximum theoretical value of the signal.

8.1.4.237 eval() [1/3]

```
template<typename T >
double eval (
    const char *const expression,
    const double x = 0,
    const double y = 0,
    const double z = 0,
    const double c = 0,
    CImgList< T > *const list_images = 0 )
```

Evaluate math formula.

Parameters

<i>expression</i>	Math formula, as a C-string.
-------------------	------------------------------

Parameters

<i>x</i>	Value of the predefined variable <i>x</i> .
<i>y</i>	Value of the predefined variable <i>y</i> .
<i>z</i>	Value of the predefined variable <i>z</i> .
<i>c</i>	Value of the predefined variable <i>c</i> .
<i>list_images</i>	A list of images attached to the specified math formula.

8.1.4.238 eval() [2/3]

```
template<typename T >
template<typename t >
void eval (
    CImg< t > & output,
    const char *const expression,
    const double x = 0,
    const double y = 0,
    const double z = 0,
    const double c = 0,
    CImgList< T > *const list_images = 0 )
```

Evaluate math formula.

Parameters

out	<i>output</i>	Contains values of output vector returned by the evaluated expression (or is empty if the returned type is scalar).
	<i>expression</i>	Math formula, as a C-string.
	<i>x</i>	Value of the predefined variable <i>x</i> .
	<i>y</i>	Value of the predefined variable <i>y</i> .
	<i>z</i>	Value of the predefined variable <i>z</i> .
	<i>c</i>	Value of the predefined variable <i>c</i> .
	<i>list_images</i>	A list of input images attached to the specified math formula.

8.1.4.239 eval() [3/3]

```
template<typename T >
template<typename t >
CImg< doubleT > eval (
    const char *const expression,
    const CImg< t > & xyzc,
    CImgList< T > *const list_images = 0 )
```

Evaluate math formula on a set of variables.

Parameters

<i>expression</i>	Math formula, as a C-string.
<i>xyzc</i>	Set of values (x,y,z,c) used for the evaluation.
<i>list_images</i>	A list of input images attached to the specified math formula.

8.1.4.240 get_stats()

```
template<typename T >
CImg< Tdouble > get_stats (
    const unsigned int variance_method = 0 ) const
```

Compute statistics vector from the pixel values.

Parameters

<i>variance_method</i>	Method used to compute the variance (see variance(const unsigned int) const).
------------------------	--

Returns

Statistics vector as [min, max, mean, variance, xmin, ymin, zmin, cmin, xmax, ymax, zmax, cmax, sum, product, L2-norm].

8.1.4.241 magnitude()

```
template<typename T >
double magnitude (
    const float magnitude_type = 2 ) const
```

Compute norm of the image, viewed as a matrix.

Parameters

<i>magnitude_type</i>	Can be: • 0: L0-norm • 1: L1-norm • 2: L2-norm • $p > 2$: Lp-norm • $\sim 0U$: Linf-norm
-----------------------	--

8.1.4.242 dot()

```
template<typename T >
template<typename t >
double dot (
    const CImg< t > & img ) const
```

Compute the dot product between instance and argument, viewed as matrices.

Parameters

<i>img</i>	Image used as a second argument of the dot product.
------------	---

8.1.4.243 get_vector_at()

```
template<typename T >
CImg< T > get_vector_at (
    const unsigned int x,
    const unsigned int y = 0,
    const unsigned int z = 0 ) const
```

Get vector-valued pixel located at specified position.

Parameters

<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.

8.1.4.244 get_matrix_at()

```
template<typename T >
CImg< T > get_matrix_at (
    const unsigned int x = 0,
    const unsigned int y = 0,
    const unsigned int z = 0 ) const
```

Get (square) matrix-valued pixel located at specified position.

Parameters

<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.

Note

- The [spectrum\(\)](#) of the image must be a square.

8.1.4.245 get_tensor_at()

```
template<typename T >
CImg< T > get_tensor_at (
    const unsigned int x,
    const unsigned int y = 0,
    const unsigned int z = 0 ) const
```

Get tensor-valued pixel located at specified position.

Parameters

<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.

8.1.4.246 set_vector_at()

```

template<typename T >
template<typename t >
CImg< T > & set_vector_at (
    const CImg< t > & vec,
    const unsigned int x,
    const unsigned int y = 0,
    const unsigned int z = 0 )

```

Set vector-valued pixel at specified position.

Parameters

<i>vec</i>	Vector to put on the instance image.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.

8.1.4.247 set_matrix_at()

```

template<typename T >
template<typename t >
CImg< T > & set_matrix_at (
    const CImg< t > & mat,
    const unsigned int x = 0,
    const unsigned int y = 0,
    const unsigned int z = 0 )

```

Set (square) matrix-valued pixel at specified position.

Parameters

<i>mat</i>	Matrix to put on the instance image.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.

8.1.4.248 set_tensor_at()

```

template<typename T >
template<typename t >
CImg< T > & set_tensor_at (
    const CImg< t > & ten,
    const unsigned int x = 0,
    const unsigned int y = 0,
    const unsigned int z = 0 )

```

Set tensor-valued pixel at specified position.

Parameters

<i>ten</i>	Tensor to put on the instance image.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.

8.1.4.249 diagonal()

```
template<typename T >
CImg< T > & diagonal ( )
```

Resize image to become a diagonal matrix.

Note

Transform the image as a diagonal matrix so that each of its initial value becomes a diagonal coefficient.

8.1.4.250 identity_matrix() [1/2]

```
template<typename T >
CImg< T > & identity_matrix ( )
```

Replace the image by an identity matrix.

Note

If the instance image is not square, it is resized to a square matrix using its maximum dimension as a reference.

8.1.4.251 sequence() [1/2]

```
template<typename T >
CImg< T > & sequence (
    const T & a0,
    const T & a1 )
```

Fill image with a linear sequence of values.

Parameters

<i>a0</i>	Starting value of the sequence.
<i>a1</i>	Ending value of the sequence.

8.1.4.252 transpose()

```
template<typename T >
```

```
CImg< T > & transpose ( )
```

Transpose the image, viewed as a matrix.

Note

Equivalent to

```
permute_axes("yxzc");
```

.

8.1.4.253 cross()

```
template<typename T >
template<typename t >
CImg< T > & cross (
    const CImg< t > & img )
```

Compute the cross product between two 1×3 images, viewed as 3D vectors.

Parameters

<i>img</i>	Image used as the second argument of the cross product.
------------	---

Note

The first argument of the cross product is `*this`.

8.1.4.254 invert()

```
template<typename T >
CImg< T > & invert ( )
```

Invert the instance matrix.

If the instance matrix is not square, the Moore-Penrose pseudo-inverse is computed instead.

8.1.4.255 solve()

```
template<typename T >
template<typename t >
CImg< T > & solve (
    const CImg< t > & A )
```

Solve a (possibly over- or under-determined) linear system using QR decomposition.

Solve the matrix equation $AX = B$, where the current instance `*this` represents B , and the argument `A` is the system matrix. This function supports both over-determined and under-determined systems by internally performing a QR decomposition with column pivoting, which improves numerical stability and correctly handles rank-deficient matrices.

- If A has more rows than columns ($m \geq n$), the system is square or over-determined, and the least-squares solution minimizing $\|AX - B\|_2$ is computed.
- If A has more columns than rows ($m < n$), the system is under-determined. The minimum-norm solution is computed using QR decomposition with column pivoting of the transposed system.

The computation is performed in double precision for numerical stability.

8.1.4.256 solve_tridiagonal()

```
template<typename T >
template<typename t >
CImg< T > & solve_tridiagonal (
    const CImg< t > & A )
```

Solve a tridiagonal system of linear equations.

Parameters

A	Coefficients of the tridiagonal system. A is a tridiagonal matrix $A = [b_0, c_0, 0, \dots; a_1, b_1, c_1, 0, \dots; \dots; \dots, 0, a_N, b_N]$, stored as a 3 columns matrix
----------	---

Note

Solve $AX=B$ where $B=*this$, using the Thomas algorithm.

8.1.4.257 eigen()

```
template<typename T >
template<typename t >
const CImg< T > & eigen (
    CImg< t > & val,
    CImg< t > & vec ) const
```

Compute eigenvalues and eigenvectors of the instance image, viewed as a matrix.

Parameters

out	val	Vector of the estimated eigenvalues, in decreasing order.
out	vec	Matrix of the estimated eigenvectors, sorted by columns.

8.1.4.258 get_eigen()

```
template<typename T >
CImgList< Tfloat > get_eigen ( ) const
```

Compute eigenvalues and eigenvectors of the instance image, viewed as a matrix.

Returns

A list of two images $[val; vec]$, whose meaning is similar as in `eigen(CImg<t>&,CImg<t>&) const`.

8.1.4.259 symmetric_eigen()

```
template<typename T >
template<typename t >
const CImg< T > & symmetric_eigen (
    CImg< t > & val,
    CImg< t > & vec ) const
```

Compute eigenvalues and eigenvectors of the instance image, viewed as a symmetric matrix.

Parameters

out	val	Vector of the estimated eigenvalues, in decreasing order.
out	vec	Matrix of the estimated eigenvectors, sorted by columns.

8.1.4.260 get_symmetric_eigen()

```
template<typename T >
CImgList< Tfloat > get_symmetric_eigen ( ) const
```

Compute eigenvalues and eigenvectors of the instance image, viewed as a symmetric matrix.

Returns

A list of two images [val; vec], whose meaning are similar as in [symmetric_eigen\(CImg<t>&,CImg<t>&\) const](#).

8.1.4.261 sort() [1/2]

```
template<typename T >
template<typename t >
CImg< T > & sort (
    CImg< t > & permutations,
    const bool is_increasing = true )
```

Sort pixel values and get sorting permutations.

Parameters

out	permutations	Permutation map used for the sorting.
	is_increasing	Indicates whether pixel values are sorted in an increasing (true) or decreasing (false) way.

8.1.4.262 sort() [2/2]

```
template<typename T >
CImg< T > & sort (
    const bool is_increasing = true,
    const char axis = 0 )
```

Sort pixel values.

Parameters

is_increasing	Indicates whether pixel values are sorted in an increasing (true) or decreasing (false) way.
---------------	--

Parameters

<i>axis</i>	Indicates whether the value sorting must be done along a specific axis. Can be: • 0: All pixel values are sorted, independently on their initial position. • 'x': Image columns are sorted, according to the first value in each column. • 'y': Image rows are sorted, according to the first value in each row. • 'z': Image slices are sorted, according to the first value in each slice. • 'c': Image channels are sorted, according to the first value in each channel.
-------------	---

8.1.4.263 SVD()

```
template<typename T >
template<typename t >
const CImg< T > & SVD (
    CImg< t > & U,
    CImg< t > & S,
    CImg< t > & V,
    const bool sorting = true,
    const unsigned int max_iteration = 40,
    const float lambda = 0 ) const
```

Compute the SVD of the instance image, viewed as a general matrix.

Compute the SVD decomposition $*this=U*S*V'$ where U and V are orthogonal matrices and S is a diagonal matrix. V' denotes the matrix transpose of V .

Parameters

out	U	First matrix of the SVD product.
out	S	Coefficients of the second (diagonal) matrix of the SVD product. These coefficients are stored as a vector.
out	V	Third matrix of the SVD product.
	<i>sorting</i>	Indicates whether the diagonal coefficients are sorted (in decreasing order).
	<i>max_iteration</i>	Maximum number of iterations considered for the algorithm convergence.
	<i>lambda</i>	Epsilon used for the algorithm convergence.

Note

The instance matrix can be computed from U, S and V by

```
const CImg<> A; // Input matrix (assumed to contain some values)
CImg<> U, S, V;
A.SVD(U, S, V)
```

8.1.4.264 get_SVD()

```
template<typename T >
CImgList< Tfloat > get_SVD (
```

```
const bool sorting = true,
const unsigned int max_iteration = 40,
const float lambda = 0 ) const
```

Compute the SVD of the instance image, viewed as a general matrix.

Returns

A list of three images [U; S; V], whose meaning is similar as in `SVD(CImg<t>&,CImg<t>&,CImg<t>&,bool,unsigned int,fl`

8.1.4.265 QR()

```
template<typename T >
template<typename t >
const CImg< T > & QR (
    CImg< t > & Q,
    CImg< t > & R,
    const bool is_reduced_form = true,
    const bool is_pivoting = false,
    CImg< uintT > *const perm = 0 ) const
```

Compute the QR decomposition of the instance matrix.

Given an instance matrix (*this) of size m×n (m rows, n columns), fill the matrices Q and R, so that *this = Q*R (without pivoting) or *this*P = Q*R (with pivoting).

- Q is an orthogonal matrix, of size 'm×m' if 'is_reduced_form==false', or 'm×min(m,n)' otherwise.
- R is an upper-triangular matrix of size 'm×n' if 'is_reduced_form==false' or 'min(m,n)×n' otherwise.
- $Q^T Q = Id$.
- If $n > m$, only the first m×m part of R is upper triangular.
- If 'is_pivoting==true', column pivoting is applied and the permutation is stored in 'perm', so that the matrix 'A_perm' formed by reordering the columns of *this according to 'perm' satisfies $A_perm = Q * R$, where $A_perm(col,row) = (*this)(perm[col],row)$. If 'is_pivoting==false', 'perm' is left unchanged.

8.1.4.266 project_matrix()

```
template<typename T >
template<typename t >
CImg< T > & project_matrix (
    const CImg< t > & dictionary,
    const unsigned int method = 0,
    const unsigned int max_iter = 0,
    const double max_residual = 1e-6 )
```

Compute the projection of the instance matrix onto the specified dictionary.

Find the best-matching projection of the selected matrix onto the span of an over-complete dictionary D, using the orthogonal projection or (optionally Orthogonal) Matching Pursuit algorithm. The instance image must be a 2D matrix in which each column represents a signal to project.

Parameters

<i>dictionary</i>	A matrix in which each column is an element of the dictionary D.
<i>method</i>	Specifies which projection method is applied. It can be: • 0 = orthogonal projection (default). • 1 = matching pursuit. • 2 = matching pursuit, with a single orthogonal projection step at the end. • ≥ 3 = orthogonal matching pursuit where an orthogonal projection step is performed every 'method-2' iterations.
<i>max_iter</i>	Sets the max number of iterations processed for each signal. If set to '0' (default), 'max_iter' is set to the number of dictionary columns. (only meaningful for matching pursuit and its variants).
<i>max_residual</i>	Provides a stopping criterion based on the signal reconstruction accuracy (only meaningful for matching pursuit and its variants).

Returns

A matrix W whose columns correspond to the sparse weights associated with each input matrix column. Thus, the matrix product $D \cdot W$ is an approximation of the input matrix.

8.1.4.267 **dijkstra()** [1/2]

```
template<typename T >
template<typename tf , typename t >
static CImg< Tfloat > dijkstra (
    const tf & distance,
    const unsigned int nb_nodes,
    const unsigned int starting_node,
    const unsigned int ending_node,
    CImg< t > & previous_node ) [static]
```

Compute minimal path in a graph, using the Dijkstra algorithm.

Parameters

<i>distance</i>	An object having operator()(unsigned int i, unsigned int j) which returns distance between two nodes (i,j).
<i>nb_nodes</i>	Number of graph nodes.
<i>starting_node</i>	Index of the starting node.
<i>ending_node</i>	Index of the ending node (set to ~0U to ignore ending node).
<i>previous_node</i>	Array that stores the index of the previous node in the shortest path to the starting node. (optional parameter).

Returns

Array of distances of each node to the starting node, typed as Tfloat to prevent overflows.

8.1.4.268 dijkstra() [2/2]

```
template<typename T >
template<typename t >
CImg< T > & dijkstra (
    const unsigned int starting_node,
    const unsigned int ending_node,
    CImg< t > & previous_node )
```

Return minimal path in a graph, using the Dijkstra algorithm.

Parameters

<i>starting_node</i>	Index of the starting node.
<i>ending_node</i>	Index of the ending node.
<i>previous_node</i>	Array that gives the previous node index in the path to the starting node (optional parameter).

Returns

Array of distances of each node to the starting node.

Note

image instance corresponds to the adjacency matrix of the graph.

8.1.4.269 string()

```
template<typename T >
static CImg< T > string (
    const char *const str,
    const bool is_last_zero = true,
    const bool is_shared = false ) [static]
```

Return an image containing the character codes of specified string.

Parameters

<i>str</i>	input C-string to encode as an image.
<i>is_last_zero</i>	Indicates whether the terminating null character ('\0') appears in the resulting image.
<i>is_shared</i>	Specifies whether the returned image shares its buffer with <i>str</i> .

8.1.4.270 row_vector() [1/4]

```
template<typename T >
static CImg< T > row_vector (
    const T & a0 ) [static]
```

Return a 1x1 image containing specified value.

Parameters

<i>a0</i>	First vector value.
-----------	---------------------

8.1.4.271 row_vector() [2/4]

```
template<typename T >
static CImg< T > row_vector (
    const T & a0,
    const T & a1 ) [static]
```

Return a 2x1 image containing specified values.

Parameters

<i>a0</i>	First vector value.
<i>a1</i>	Second vector value.

8.1.4.272 row_vector() [3/4]

```
template<typename T >
static CImg< T > row_vector (
    const T & a0,
    const T & a1,
    const T & a2 ) [static]
```

Return a 3x1 image containing specified values.

Parameters

<i>a0</i>	First vector value.
<i>a1</i>	Second vector value.
<i>a2</i>	Third vector value.

8.1.4.273 row_vector() [4/4]

```
template<typename T >
static CImg< T > row_vector (
    const T & a0,
    const T & a1,
    const T & a2,
    const T & a3 ) [static]
```

Return a 4x1 image containing specified values.

Parameters

<i>a0</i>	First vector value.
<i>a1</i>	Second vector value.
<i>a2</i>	Third vector value.
<i>a3</i>	Fourth vector value.

8.1.4.274 vector() [1/4]

```
template<typename T >
static CImg< T > vector (
    const T & a0 ) [static]
```

Return a 1x1 image containing specified value.

Parameters

<i>a0</i>	First vector value.
-----------	---------------------

8.1.4.275 vector() [2/4]

```
template<typename T >
static CImg< T > vector (
    const T & a0,
    const T & a1 ) [static]
```

Return a 1x2 image containing specified values.

Parameters

<i>a0</i>	First vector value.
<i>a1</i>	Second vector value.

8.1.4.276 vector() [3/4]

```
template<typename T >
static CImg< T > vector (
    const T & a0,
    const T & a1,
    const T & a2 ) [static]
```

Return a 1x3 image containing specified values.

Parameters

<i>a0</i>	First vector value.
<i>a1</i>	Second vector value.
<i>a2</i>	Third vector value.

8.1.4.277 vector() [4/4]

```
template<typename T >
static CImg< T > vector (
    const T & a0,
    const T & a1,
```

```
const T & a2,
const T & a3 ) [static]
```

Return a 1x4 image containing specified values.

Parameters

<i>a0</i>	First vector value.
<i>a1</i>	Second vector value.
<i>a2</i>	Third vector value.
<i>a3</i>	Fourth vector value.

8.1.4.278 matrix() [1/3]

```
template<typename T >
static CImg< T > matrix (
    const T & a0 ) [static]
```

Return a 1x1 matrix containing specified coefficients.

Parameters

<i>a0</i>	First matrix value.
-----------	---------------------

Note

Equivalent to `vector(const T&)`.

8.1.4.279 matrix() [2/3]

```
template<typename T >
static CImg< T > matrix (
    const T & a0,
    const T & a1,
    const T & a2,
    const T & a3 ) [static]
```

Return a 2x2 matrix containing specified coefficients.

Parameters

<i>a0</i>	First matrix value.
<i>a1</i>	Second matrix value.
<i>a2</i>	Third matrix value.
<i>a3</i>	Fourth matrix value.

8.1.4.280 matrix() [3/3]

```
template<typename T >
static CImg< T > matrix (
    const T & a0,
    const T & a1,
    const T & a2,
    const T & a3,
    const T & a4,
    const T & a5,
    const T & a6,
    const T & a7,
    const T & a8 ) [static]
```

Return a 3x3 matrix containing specified coefficients.

Parameters

<i>a0</i>	First matrix value.
<i>a1</i>	Second matrix value.
<i>a2</i>	Third matrix value.
<i>a3</i>	Fourth matrix value.
<i>a4</i>	Fifth matrix value.
<i>a5</i>	Sixth matrix value.
<i>a6</i>	Seventh matrix value.
<i>a7</i>	Eighth matrix value.
<i>a8</i>	Ninth matrix value.

8.1.4.281 tensor()

```
template<typename T >
static CImg< T > tensor (
    const T & a0 ) [static]
```

Return a 1x1 symmetric matrix containing specified coefficients.

Parameters

<i>a0</i>	First matrix value.
-----------	---------------------

Note

Equivalent to `vector(const T&)`.

8.1.4.282 identity_matrix() [2/2]

```
template<typename T >
static CImg< T > identity_matrix (
    const unsigned int N ) [static]
```

Return a NxN identity matrix.

Parameters

<i>N</i>	Dimension of the matrix.
----------	--------------------------

8.1.4.283 sequence() [2/2]

```
template<typename T >
static CImg< T > sequence (
    const unsigned int N,
    const T & a0,
    const T & a1 ) [static]
```

Return a N-numbered sequence vector from a0 to a1.

Parameters

<i>N</i>	Size of the resulting vector.
<i>a0</i>	Starting value of the sequence.
<i>a1</i>	Ending value of the sequence.

8.1.4.284 rotation_matrix()

```
template<typename T >
static CImg< T > rotation_matrix (
    const float x,
    const float y,
    const float z,
    const float w,
    const bool is_quaternion = false ) [static]
```

Return a 3x3 rotation matrix from an { axis + angle } or a quaternion.

Parameters

<i>x</i>	X-coordinate of the rotation axis, or first quaternion coordinate.
<i>y</i>	Y-coordinate of the rotation axis, or second quaternion coordinate.
<i>z</i>	Z-coordinate of the rotation axis, or third quaternion coordinate.
<i>w</i>	Angle of the rotation axis (in degree), or fourth quaternion coordinate.
<i>is_quaternion</i>	Tell is the four arguments denotes a set { axis + angle } or a quaternion (x,y,z,w).

8.1.4.285 fill() [1/4]

```
template<typename T >
CImg< T > & fill (
    const T & val )
```

Fill all pixel values with specified value.

Parameters

<i>val</i>	Fill value.
------------	-------------

8.1.4.286 fill() [2/4]

```
template<typename T >
CImg< T > & fill (
    const T & val0,
    const T & val1 )
```

Fill sequentially all pixel values with specified values.

Parameters

<i>val0</i>	First fill value.
<i>val1</i>	Second fill value.

8.1.4.287 fill() [3/4]

```
template<typename T >
CImg< T > & fill (
    const char *const expression,
    const bool repeat_values,
    const bool allow_formula = true,
    CImgList< T > *const list_images = 0 )
```

Fill sequentially pixel values according to a given expression.

Parameters

<i>expression</i>	C-string describing a math formula, or a sequence of values.
<i>repeat_values</i>	In case a list of values is provided, indicates that this list must be repeated for the filling.
<i>allow_formula</i>	Indicates that mathematical formulas are authorized for the filling.
<i>list_images</i>	In case of a mathematical expression, attach a list of images to the specified expression.

8.1.4.288 fill_from_values()

```
template<typename T >
CImg< T > & fill_from_values (
    const char *const values,
    const bool repeat_values )
```

Fill sequentially pixel values according to a value sequence, given as a string.

Parameters

<i>values</i>	C-string describing a sequence of values.
<i>repeat_values</i>	Indicates whether this sequence must be repeated when filling.

8.1.4.289 fill() [4/4]

```
template<typename T >
template<typename t >
CImg< T > & fill (
    const CImg< t > & values,
    const bool repeat_values = true )
```

Fill sequentially pixel values according to the values found in another image.

Parameters

<i>values</i>	Image containing the values used for the filling.
<i>repeat_values</i>	In case there are less values than necessary in <i>values</i> , indicates that these values must be repeated for the filling.

8.1.4.290 fillX()

```
template<typename T >
CImg< T > & fillX (
    const unsigned int y,
    const unsigned int z,
    const unsigned int c,
    const int a0,
    ... )
```

Fill pixel values along the X-axis at a specified pixel position.

Parameters

<i>y</i>	Y-coordinate of the filled column.
<i>z</i>	Z-coordinate of the filled column.
<i>c</i>	C-coordinate of the filled column.
<i>a0</i>	First fill value.

8.1.4.291 fillY()

```
template<typename T >
CImg< T > & fillY (
    const unsigned int x,
    const unsigned int z,
    const unsigned int c,
    const int a0,
    ... )
```

Fill pixel values along the Y-axis at a specified pixel position.

Parameters

<i>x</i>	X-coordinate of the filled row.
<i>z</i>	Z-coordinate of the filled row.
<i>c</i>	C-coordinate of the filled row.
<i>a0</i>	First fill value.

8.1.4.292 fillZ()

```
template<typename T >
CImg< T > & fillZ (
    const unsigned int x,
    const unsigned int y,
    const unsigned int c,
    const int a0,
    ... )
```

Fill pixel values along the Z-axis at a specified pixel position.

Parameters

<i>x</i>	X-coordinate of the filled slice.
<i>y</i>	Y-coordinate of the filled slice.
<i>c</i>	C-coordinate of the filled slice.
<i>a0</i>	First fill value.

8.1.4.293 fillC()

```
template<typename T >
CImg< T > & fillC (
    const unsigned int x,
    const unsigned int y,
    const unsigned int z,
    const int a0,
    ... )
```

Fill pixel values along the C-axis at a specified pixel position.

Parameters

<i>x</i>	X-coordinate of the filled channel.
<i>y</i>	Y-coordinate of the filled channel.
<i>z</i>	Z-coordinate of the filled channel.
<i>a0</i>	First filling value.

8.1.4.294 discard()

```
template<typename T >
template<typename t >
CImg< T > & discard (
    const CImg< t > & values,
    const char axis = 0 )
```

Discard specified sequence of values in the image buffer, along a specific axis.

Parameters

<i>values</i>	Sequence of values to discard.
<i>axis</i>	Axis along which the values are discarded. If set to 0 (default value) the function does it for all the
Generated by Doxygen	
buffer values and returns a one-column vector.	

Note

Discarded values will change the image geometry, so the resulting image is returned as a one-column vector.

8.1.4.295 rand() [1/2]

```
template<typename T >
CImg< T > & rand (
    const T & val_min,
    const T & val_max )
```

Fill image with random values in specified range.

Parameters

<i>val_min</i>	Minimal authorized random value.
<i>val_max</i>	Maximal authorized random value.

Note

Random variables are uniformly distributed in [val_min,val_max].

8.1.4.296 rand() [2/2]

```
template<typename T >
template<typename t >
CImg< T > & rand (
    const T & val_min,
    const T & val_max,
    const CImg< t > & pdf,
    const int precision = 65536 )
```

Fill image with random values following specified distribution and range.

Parameters

<i>val_min</i>	Minimal authorized random value.
<i>val_max</i>	Maximal authorized random value.
<i>pdf</i>	Probability density function.
<i>precision</i>	Precision of generated values. Set to '0' for automatic precision. A negative value means 'percentage of the pdf size'.

8.1.4.297 round()

```
template<typename T >
CImg< T > & round (
    const double y = 1,
    const int rounding_type = 0 )
```

Round pixel values.

Parameters

<i>y</i>	Rounding precision.
<i>rounding_type</i>	Rounding type. Can be: • -1: Backward. • 0: Nearest. • 1: Forward.

8.1.4.298 noise()

```
template<typename T >
CImg< T > & noise (
    const double amplitude,
    const unsigned int noise_type = 0 )
```

Add random noise to pixel values.

Parameters

<i>amplitude</i>	Amplitude of the random additive noise. If <code>sigma<0</code> , it stands for a percentage of the global value range.
<i>noise_type</i>	Type of additive noise (can be 0=gaussian, 1=uniform, 2=Salt and Pepper, 3=Poisson or 4=Rician).

Returns

A reference to the modified image instance.

Note

- For Poisson noise (`noise_type=3`), parameter `sigma` is ignored, as Poisson noise only depends on the image value itself.
- Function `CImg<T>::get_noise()` is also defined. It returns a non-shared modified copy of the image instance.

Example

```
const CImg<float> img("reference.jpg"), res = img.get_noise(40);
(img, res.normalize(0,255)).display();
```

8.1.4.299 normalize() [1/2]

```
template<typename T >
CImg< T > & normalize (
    const T & min_value,
    const T & max_value,
    const float constant_case_ratio = 0 )
```

Linearly normalize pixel values.

Parameters

<i>min_value</i>	Minimum desired value of the resulting image.
<i>max_value</i>	Maximum desired value of the resulting image.
<i>constant_case_ratio</i>	In case of instance image having a constant value, tell what ratio of [min_value,max_value] is used to fill the normalized image (=0 for min_value, =1 for max_value, =0.5 for (min_value + max_value)/2).

Example

```
const CImg<float> img("reference.jpg"), res = img.get_normalize(160,220);


```

8.1.4.300 normalize() [2/2]

```
template<typename T >
CImg< T > & normalize ( )
```

Normalize multi-valued pixels of the image instance, with respect to their L2-norm.

Example

```
const CImg<float> img("reference.jpg"), res = img.get_normalize();


```

8.1.4.301 norm()

```
template<typename T >
CImg< T > & norm (
    const int norm_type = 2 )
```

Compute Lp-norm of each multi-valued pixel of the image instance.

Parameters

<i>norm_type</i>	Type of computed vector norm (can be -1=Linf, or greater or equal than 0).
------------------	--

Example

```
const CImg<float> img("reference.jpg"), res = img.get_norm();


```

8.1.4.302 cut()

```
template<typename T >
CImg< T > & cut (
    const T & min_value,
    const T & max_value )
```

Cut pixel values in specified range.

Parameters

<i>min_value</i>	Minimum desired value of the resulting image.
<i>max_value</i>	Maximum desired value of the resulting image.

Example

```
const CImg<float> img("reference.jpg"), res = img.get_cut(160,220);
(img,res).display();
```

8.1.4.303 abscut()

```
template<typename T >
CImg< T > & abscut (
    const T & min_value,
    const T & max_value,
    const T & offset )
```

Cut pixel absolute values in specified range.

Parameters

<i>min_value</i>	Minimum desired absolute value of the resulting image.
<i>max_value</i>	Maximum desired absolute value of the resulting image.
<i>offset</i>	Offset applied to absolute value of the resulting image.

8.1.4.304 quantize()

```
template<typename T >
CImg< T > & quantize (
    const unsigned int nb_levels,
    const bool keep_range = true )
```

Uniformly quantize pixel values.

Parameters

<i>nb_levels</i>	Number of quantization levels.
<i>keep_range</i>	Indicates whether resulting values keep the same range as the original ones.

Example

```
const CImg<float> img("reference.jpg"), res = img.get_quantize(4);
(img,res).display();
```

8.1.4.305 otsu()

```
template<typename T >
T otsu (
    const unsigned int nb_levels = 256 ) const
```

Return the Otsu threshold.

Parameters

<i>nb_levels</i>	Number of histogram levels used for the estimation.
------------------	---

8.1.4.306 threshold()

```
template<typename T >
CImg< T > & threshold (
    const T & value,
    const bool soft_threshold = false,
    const bool strict_threshold = false )
```

Threshold pixel values.

Parameters

<i>value</i>	Threshold value
<i>soft_threshold</i>	Indicates whether soft thresholding must be applied (instead of hard one).
<i>strict_threshold</i>	Indicates whether threshold value is strict.

Example

```
const CImg<float> img("reference.jpg"), res = img.get_threshold(128);
(img,res.normalize(0,255)).display();
```

8.1.4.307 histogram()

```
template<typename T >
CImg< T > & histogram (
    const unsigned int nb_levels,
    const T & min_value,
    const T & max_value )
```

Compute the histogram of pixel values.

Parameters

<i>nb_levels</i>	Number of desired histogram levels.
<i>min_value</i>	Minimum pixel value considered for the histogram computation. All pixel values lower than <i>min_value</i> will not be counted.
<i>max_value</i>	Maximum pixel value considered for the histogram computation. All pixel values higher than <i>max_value</i> will not be counted.

Note

- The histogram *H* of an image *I* is the 1D function where *H*(*x*) counts the number of occurrences of the value *x* in the image *I*.

- The resulting histogram is always defined in 1D. Histograms of multi-valued images are not multi-dimensional.

Example

```
const CImg<float> img = CImg<float>("reference.jpg").histogram(256);
img.display_graph(0,3);
```

8.1.4.308 equalize()

```
template<typename T >
CImg< T > & equalize (
    const unsigned int nb_levels,
    const T & min_value,
    const T & max_value )
```

Equalize histogram of pixel values.

Parameters

<i>nb_levels</i>	Number of histogram levels used for the equalization.
<i>min_value</i>	Minimum pixel value considered for the histogram computation. All pixel values lower than <i>min_value</i> will not be counted.
<i>max_value</i>	Maximum pixel value considered for the histogram computation. All pixel values higher than <i>max_value</i> will not be counted.

Example

```
const CImg<float> img("reference.jpg"), res = img.get_equalize(256,0,255);
(img,res).display();
```

8.1.4.309 index()

```
template<typename T >
template<typename t >
CImg< T > & index (
    const CImg< t > & colormap,
    const float dithering = 1,
    const bool map_colors = false )
```

Index multi-valued pixels regarding to a specified palette.

Parameters

<i>colormap</i>	Multi-valued colormap used as the basis for multi-valued pixel indexing.
<i>dithering</i>	Level of dithering (0=disable, 1=standard level).
<i>map_colors</i>	Indicates whether the values of the resulting image are the colormap indices or the colormap vectors.

Note

- `img.index(colormap,dithering,1)` is equivalent to `img.index(colormap,dithering,0).map(colormap)`

Example

```
const CImg<float> img("reference.jpg"), colormap(3,1,1,3, 0,128,255, 0,128,255, 0,128,255);
const CImg<float> res = img.get_index(colormap,1,true);
(img,res).display();
```

8.1.4.310 map()

```
template<typename T >
template<typename t >
CImg< T > & map (
    const CImg< t > & palette,
    const unsigned int boundary_conditions = 0 )
```

Map predefined palette on the scalar (indexed) image instance.

Parameters

<i>palette</i>	Multi-valued palette used for mapping the indexes.
<i>boundary_conditions</i>	Boundary conditions. Can be { 0=dirichlet 1=neumann 2=periodic 3=mirror }.

Example

```
const CImg<float> img("reference.jpg"),
    palette1(3,1,1,3, 0,128,255, 0,128,255, 0,128,255),
    palette2(3,1,1,3, 255,0,0, 0,255,0, 0,0,255),
    res = img.get_index(palette1,0).map(palette2);
(img,res).display();
```

8.1.4.311 label() [1/2]

```
template<typename T >
CImg< T > & label (
    const bool is_high_connectivity = false,
    const Tfloat tolerance = 0,
    const bool is_L2_norm = true )
```

Label connected components.

Parameters

<i>is_high_connectivity</i>	Indicates whether the algorithm uses low or high connectivity.
<i>tolerance</i>	Tolerance used to determine if two neighboring pixels belong to the same region.
<i>is_L2_norm</i>	If true, tolerance is compared against L2 difference, otherwise L1 is used.

Note

The algorithm of connected components computation has been primarily done by A. Meijster, according to the publication: 'W.H. Hesselink, A. Meijster, C. Bron, "Concurrent Determination of Connected Components.", In: Science of Computer Programming 41 (2001), pp. 173–194'. The submitted code has then been modified to fit [CImg](#) coding style and constraints.

8.1.4.312 label() [2/2]

```
template<typename T >
template<typename t >
CImg< T > & label (
    const CImg< t > & connectivity_mask,
    const Tfloat tolerance = 0,
    const bool is_L2_norm = true )
```

Label connected components **[in-place version]**.

Parameters

<i>connectivity_mask</i>	Mask of the neighboring pixels.
<i>tolerance</i>	Tolerance used to determine if two neighboring pixels belong to the same region.
<i>is_L2_norm</i>	If true, tolerance is compared against L2 difference, otherwise L1 is used.

8.1.4.313 default_LUT256()

```
template<typename T >
static const CImg< Tuchar > & default_LUT256 ( ) [static]
```

Return palette "*default*", containing 256 colors entries in RGB.

Returns

The following 256×1×1×3 palette is returned:

8.1.4.314 HSV_LUT256()

```
template<typename T >
static const CImg< Tuchar > & HSV_LUT256 ( ) [static]
```

Return palette "*HSV*", containing 256 colors entries in RGB.

Returns

The following 256×1×1×3 palette is returned:

8.1.4.315 lines_LUT256()

```
template<typename T >
static const CImg< Tuchar > & lines_LUT256 ( ) [static]
```

Return palette "*lines*", containing 256 colors entries in RGB.

Returns

The following 256x1x1x3 palette is returned:

8.1.4.316 hot_LUT256()

```
template<typename T >
static const CImg< Tuchar > & hot_LUT256 ( ) [static]
```

Return palette "*hot*", containing 256 colors entries in RGB.

Returns

The following 256x1x1x3 palette is returned:

8.1.4.317 cool_LUT256()

```
template<typename T >
static const CImg< Tuchar > & cool_LUT256 ( ) [static]
```

Return palette "*cool*", containing 256 colors entries in RGB.

Returns

The following 256x1x1x3 palette is returned:

8.1.4.318 jet_LUT256()

```
template<typename T >
static const CImg< Tuchar > & jet_LUT256 ( ) [static]
```

Return palette "*jet*", containing 256 colors entries in RGB.

Returns

The following 256x1x1x3 palette is returned:

8.1.4.319 flag_LUT256()

```
template<typename T >
static const CImg< Tuchar > & flag_LUT256 ( ) [static]
```

Return palette "*flag*", containing 256 colors entries in RGB.

Returns

The following 256x1x1x3 colormap is returned:

8.1.4.320 cube_LUT256()

```
template<typename T >
static const CImg< Tuchar > & cube_LUT256 ( ) [static]
```

Return palette "*cube*", containing 256 colors entries in RGB.

Returns

The following 256x1x1x3 palette is returned:

8.1.4.321 RGBtoXYZ()

```
template<typename T >
CImg< T > & RGBtoXYZ (
    const bool use_D65 = true )
```

Convert pixel values from RGB to XYZ color spaces.

Parameters

<i>use_D65</i>	Tell to use the D65 illuminant (D50 otherwise).
----------------	---

8.1.4.322 XYZtoRGB()

```
template<typename T >
CImg< T > & XYZtoRGB (
    const bool use_D65 = true )
```

Convert pixel values from XYZ to RGB color spaces.

Parameters

<i>use_D65</i>	Tell to use the D65 illuminant (D50 otherwise).
----------------	---

8.1.4.323 `resize()` [1/3]

```
template<typename T >
CImg< T > & resize (
    const int size_x,
    const int size_y = -100,
    const int size_z = -100,
    const int size_c = -100,
    const int interpolation_type = 1,
    const unsigned int boundary_conditions = 0,
    const float centering_x = 0,
    const float centering_y = 0,
    const float centering_z = 0,
    const float centering_c = 0 )
```

Resize image to new dimensions.

Parameters

<i>size_x</i>	Number of columns (new size along the X-axis).
<i>size_y</i>	Number of rows (new size along the Y-axis).
<i>size_z</i>	Number of slices (new size along the Z-axis).
<i>size_c</i>	Number of vector-channels (new size along the C-axis).
<i>interpolation_type</i>	Method of interpolation: • -1 = no interpolation: raw memory resizing. • 0 = no interpolation: additional space is filled according to <code>boundary_conditions</code>. • 1 = nearest-neighbor interpolation. • 2 = moving average interpolation. • 3 = linear interpolation. • 4 = grid interpolation. • 5 = cubic interpolation. • 6 = lanczos interpolation.
<i>boundary_conditions</i>	Type of boundary conditions used if necessary.
<i>centering_x</i>	Set centering type (only if <code>interpolation_type=0</code>).
<i>centering_y</i>	Set centering type (only if <code>interpolation_type=0</code>).
<i>centering_z</i>	Set centering type (only if <code>interpolation_type=0</code>).
<i>centering_c</i>	Set centering type (only if <code>interpolation_type=0</code>).

Note

If `pd[x,y,z,v]<0`, it corresponds to a percentage of the original size (the default value is -100).

8.1.4.324 `resize()` [2/3]

```
template<typename T >
template<typename t >
```

```

CImg< T > & resize (
    const CImg< t > & src,
    const int interpolation_type = 1,
    const unsigned int boundary_conditions = 0,
    const float centering_x = 0,
    const float centering_y = 0,
    const float centering_z = 0,
    const float centering_c = 0 )

```

Resize image to dimensions of another image.

Parameters

<i>src</i>	Reference image used for dimensions.
<i>interpolation_type</i>	Interpolation method.
<i>boundary_conditions</i>	Boundary conditions. Can be { 0=dirichlet 1=neumann 2=periodic 3=mirror }.
<i>centering_x</i>	Set centering type (only if interpolation_type=0).
<i>centering_y</i>	Set centering type (only if interpolation_type=0).
<i>centering_z</i>	Set centering type (only if interpolation_type=0).
<i>centering_c</i>	Set centering type (only if interpolation_type=0).

8.1.4.325 `resize()` [3/3]

```

template<typename T >
CImg< T > & resize (
    const CImgDisplay & disp,
    const int interpolation_type = 1,
    const unsigned int boundary_conditions = 0,
    const float centering_x = 0,
    const float centering_y = 0,
    const float centering_z = 0,
    const float centering_c = 0 )

```

Resize image to dimensions of a display window.

Parameters

<i>disp</i>	Reference display window used for dimensions.
<i>interpolation_type</i>	Interpolation method.
<i>boundary_conditions</i>	Boundary conditions. Can be { 0=dirichlet 1=neumann 2=periodic 3=mirror }.
<i>centering_x</i>	Set centering type (only if interpolation_type=0).
<i>centering_y</i>	Set centering type (only if interpolation_type=0).
<i>centering_z</i>	Set centering type (only if interpolation_type=0).
<i>centering_c</i>	Set centering type (only if interpolation_type=0).

8.1.4.326 `resize_doubleXY()`

```

template<typename T >
CImg< T > & resize_doubleXY ( )

```

Resize image to double-size, using the Scale2X algorithm.

Note

Use anisotropic upscaling algorithm [described here](#).

8.1.4.327 resize_tripleXY()

```
template<typename T >
CImg< T > & resize_tripleXY ( )
```

Resize image to triple-size, using the Scale3X algorithm.

Note

Use anisotropic upscaling algorithm [described here](#).

8.1.4.328 mirror() [1/2]

```
template<typename T >
CImg< T > & mirror (
    const char axis )
```

Mirror image content along specified axis.

Parameters

<i>axis</i>	Mirror axis
-------------	-------------

8.1.4.329 mirror() [2/2]

```
template<typename T >
CImg< T > & mirror (
    const char *const axes )
```

Mirror image content along specified axes.

Parameters

<i>axes</i>	Mirror axes, as a C-string.
-------------	-----------------------------

Note

axes may contains multiple characters, e.g. "xyz"

8.1.4.330 shift()

```
template<typename T >
CImg< T > & shift (
```

```

const int delta_x,
const int delta_y = 0,
const int delta_z = 0,
const int delta_c = 0,
const unsigned int boundary_conditions = 0 )

```

Shift the image content.

Parameters

<i>delta_x</i>	Amount of displacement along the X-axis.
<i>delta_y</i>	Amount of displacement along the Y-axis.
<i>delta_z</i>	Amount of displacement along the Z-axis.
<i>delta_c</i>	Amount of displacement along the C-axis.
<i>boundary_conditions</i>	Boundary conditions. Can be { 0=dirichlet 1=neumann 2=periodic 3=mirror }.

8.1.4.331 permute_axes()

```

template<typename T >
CImg< T > & permute_axes (
    const char *const axes_order )

```

Permute axes order.

Parameters

<i>axes_order</i>	Axes permutations, as a C-string of 4 characters. This function permutes the image content according to the specified axes order.
-------------------	---

8.1.4.332 unroll()

```

template<typename T >
CImg< T > & unroll (
    const char axis )

```

Unroll pixel values along the specified axis.

Parameters

<i>axis</i>	Unroll axis (can be 'x', 'y', 'z' or 'c').
-------------	--

8.1.4.333 rotate() [1/4]

```

template<typename T >
CImg< T > & rotate (
    const float angle,
    const unsigned int interpolation = 1,
    const unsigned int boundary_conditions = 0 )

```

Rotate the image by an arbitrary angle.

Parameters

<i>angle</i>	Rotation angle, in degrees.
<i>interpolation</i>	Type of interpolation. Can be { 0=nearest 1=linear 2=cubic }.
<i>boundary_conditions</i>	Boundary conditions. Can be { 0=dirichlet 1=neumann 2=periodic 3=mirror }.

Note

The size of the image is modified.

8.1.4.334 rotate() [2/4]

```
template<typename T >
CImg< T > & rotate (
    const float angle,
    const float cx,
    const float cy,
    const unsigned int interpolation,
    const unsigned int boundary_conditions = 0 )
```

Rotate image with arbitrary angle, around a center point.

Parameters

<i>angle</i>	Rotation angle, in degrees.
<i>cx</i>	X-coordinate of the rotation center.
<i>cy</i>	Y-coordinate of the rotation center.
<i>interpolation</i>	Type of interpolation, { 0=nearest 1=linear 2=cubic 3=mirror }.
<i>boundary_conditions</i>	Boundary conditions, { 0=dirichlet 1=neumann 2=periodic 3=mirror }.

8.1.4.335 rotate() [3/4]

```
template<typename T >
CImg< T > rotate (
    const float u,
    const float v,
    const float w,
    const float angle,
    const unsigned int interpolation,
    const unsigned int boundary_conditions )
```

Rotate a volumetric image by an arbitrary angle and axis.

Parameters

<i>u</i>	X-coordinate of the 3D rotation axis.
<i>v</i>	Y-coordinate of the 3D rotation axis.
<i>w</i>	Z-coordinate of the 3D rotation axis.
<i>angle</i>	Rotation angle, in degrees.
<i>interpolation</i>	Type of interpolation. Can be { 0=nearest 1=linear 2=cubic }.
<i>boundary_conditions</i>	Boundary conditions. Can be { 0=dirichlet 1=neumann 2=periodic 3=mirror }.

Note

Most of the time, size of the image is modified.

8.1.4.336 rotate() [4/4]

```
template<typename T >
CImg< T > rotate (
    const float u,
    const float v,
    const float w,
    const float angle,
    const float cx,
    const float cy,
    const float cz,
    const unsigned int interpolation = 1,
    const unsigned int boundary_conditions = 0 )
```

Rotate volumetric image with arbitrary angle and axis, around a center point.

Parameters

<i>u</i>	X-coordinate of the 3D rotation axis.
<i>v</i>	Y-coordinate of the 3D rotation axis.
<i>w</i>	Z-coordinate of the 3D rotation axis.
<i>angle</i>	Rotation angle, in degrees.
<i>cx</i>	X-coordinate of the rotation center.
<i>cy</i>	Y-coordinate of the rotation center.
<i>cz</i>	Z-coordinate of the rotation center.
<i>interpolation</i>	Type of interpolation. Can be { 0=nearest 1=linear 2=cubic 3=mirror }.
<i>boundary_conditions</i>	Boundary conditions. Can be { 0=dirichlet 1=neumann 2=periodic }.

Note

Most of the time, size of the image is modified.

8.1.4.337 warp()

```
template<typename T >
template<typename t >
CImg< T > & warp (
    const CImg< t > & p_warp,
    const unsigned int mode = 0,
    const unsigned int interpolation = 1,
    const unsigned int boundary_conditions = 0 )
```

Warp the image content using a warping field.

Parameters

<i>p_warp</i>	Warping field.
<i>mode</i>	Can be { 0=backward-absolute 1=backward-relative 2=forward-absolute 3=forward-relative }
<i>interpolation</i>	Can be { 0=nearest 1=linear 2=cubic }.
<i>boundary_conditions</i>	Boundary conditions { 0=dirichlet 1=neumann 2=periodic 3=mirror }.

8.1.4.338 get_projections2d()

```
template<typename T >
CImg< T > get_projections2d (
    const unsigned int x0,
    const unsigned int y0,
    const unsigned int z0 ) const
```

Generate a 2D representation of a 3D image, with XY, XZ and YZ views.

Parameters

<i>x0</i>	X-coordinate of the projection point.
<i>y0</i>	Y-coordinate of the projection point.
<i>z0</i>	Z-coordinate of the projection point.

8.1.4.339 crop()

```
template<typename T >
CImg< T > & crop (
    const int x0,
    const int y0,
    const int z0,
    const int c0,
    const int x1,
    const int y1,
    const int z1,
    const int c1,
    const unsigned int boundary_conditions = 0 )
```

Crop image region.

Parameters

<i>x0</i>	= X-coordinate of the upper-left crop rectangle corner.
<i>y0</i>	= Y-coordinate of the upper-left crop rectangle corner.
<i>z0</i>	= Z-coordinate of the upper-left crop rectangle corner.
<i>c0</i>	= C-coordinate of the upper-left crop rectangle corner.
<i>x1</i>	= X-coordinate of the lower-right crop rectangle corner.
<i>y1</i>	= Y-coordinate of the lower-right crop rectangle corner.
<i>z1</i>	= Z-coordinate of the lower-right crop rectangle corner.
<i>c1</i>	= C-coordinate of the lower-right crop rectangle corner.
<i>boundary_conditions</i>	= Can be { 0=dirichlet 1=neumann 2=periodic 3=mirror }.

8.1.4.340 autocrop()

```
template<typename T >
CImg< T > & autocrop (
    const T *const color = 0,
    const char *const axes = "zyx" )
```

Autocrop image region, regarding the specified background color.

Parameters

<i>color</i>	Color used for the crop. If 0, color is guessed.
<i>axes</i>	Axes used for the crop.

8.1.4.341 get_column()

```
template<typename T >
CImg< T > get_column (
    const int x0 ) const
```

Return specified image column.

Parameters

<i>x0</i>	Image column.
-----------	---------------

8.1.4.342 columns()

```
template<typename T >
CImg< T > & columns (
    const int x0,
    const int x1 )
```

Return specified range of image columns.

Parameters

<i>x0</i>	Starting image column.
<i>x1</i>	Ending image column.

8.1.4.343 row()

```
template<typename T >
CImg< T > & row (
    const int y0 )
```

Return specified image row **[in-place version]**.

Parameters

<i>y0</i>	Image row.
-----------	------------

8.1.4.344 get_rows()

```
template<typename T >
CImg< T > get_rows (
    const int y0,
    const int y1 ) const
```

Return specified range of image rows.

Parameters

<i>y0</i>	Starting image row.
<i>y1</i>	Ending image row.

8.1.4.345 get_slice()

```
template<typename T >
CImg< T > get_slice (
    const int z0 ) const
```

Return specified image slice.

Parameters

<i>z0</i>	Image slice.
-----------	--------------

8.1.4.346 get_slices()

```
template<typename T >
```

```
CImg< T > get_slices (
    const int z0,
    const int z1 ) const
```

Return specified range of image slices.

Parameters

<i>z0</i>	Starting image slice.
<i>z1</i>	Ending image slice.

8.1.4.347 get_channel()

```
template<typename T >
CImg< T > get_channel (
    const int c0 ) const
```

Return specified image channel.

Parameters

<i>c0</i>	Image channel.
-----------	----------------

8.1.4.348 get_channels()

```
template<typename T >
CImg< T > get_channels (
    const int c0,
    const int c1 ) const
```

Return specified range of image channels.

Parameters

<i>c0</i>	Starting image channel.
<i>c1</i>	Ending image channel.

8.1.4.349 streamline()

```
template<typename T >
template<typename tfunc >
static CImg< floatT > streamline (
    const tfunc & func,
    const float x,
    const float y,
    const float z,
    const float L = 256,
    const float dl = 0.1f,
```

```

const unsigned int interpolation_type = 2,
const bool is_backward_tracking = false,
const bool is_oriented_only = false,
const float x0 = 0,
const float y0 = 0,
const float z0 = 0,
const float x1 = 0,
const float y1 = 0,
const float z1 = 0 ) [static]

```

Return stream line of a 3D vector field.

Parameters

<i>func</i>	Vector field function.
<i>x</i>	X-coordinate of the starting point of the streamline.
<i>y</i>	Y-coordinate of the starting point of the streamline.
<i>z</i>	Z-coordinate of the starting point of the streamline.
<i>L</i>	Streamline length.
<i>dl</i>	Streamline length increment.
<i>interpolation_type</i>	Type of interpolation. Can be { 0=nearest integer 1=linear 2=2nd-order RK 3=4th-order RK }.
<i>is_backward_tracking</i>	Indicates whether the streamline is estimated forward or backward.
<i>is_oriented_only</i>	Indicates whether the direction of the vectors must be ignored.
<i>x0</i>	X-coordinate of the first bounding-box vertex.
<i>y0</i>	Y-coordinate of the first bounding-box vertex.
<i>z0</i>	Z-coordinate of the first bounding-box vertex.
<i>x1</i>	X-coordinate of the second bounding-box vertex.
<i>y1</i>	Y-coordinate of the second bounding-box vertex.
<i>z1</i>	Z-coordinate of the second bounding-box vertex.

8.1.4.350 get_shared_points()

```

template<typename T >
CImg< T > get_shared_points (
    const unsigned int x0,
    const unsigned int x1,
    const unsigned int y0 = 0,
    const unsigned int z0 = 0,
    const unsigned int c0 = 0 )

```

Return a shared-memory image referencing a range of pixels of the image instance.

Parameters

<i>x0</i>	X-coordinate of the starting pixel.
<i>x1</i>	X-coordinate of the ending pixel.
<i>y0</i>	Y-coordinate.
<i>z0</i>	Z-coordinate.
<i>c0</i>	C-coordinate.

8.1.4.351 get_shared_rows()

```
template<typename T >
CImg< T > get_shared_rows (
    const unsigned int y0,
    const unsigned int y1,
    const unsigned int z0 = 0,
    const unsigned int c0 = 0 )
```

Return a shared-memory image referencing a range of rows of the image instance.

Parameters

<i>y0</i>	Y-coordinate of the starting row.
<i>y1</i>	Y-coordinate of the ending row.
<i>z0</i>	Z-coordinate.
<i>c0</i>	C-coordinate.

8.1.4.352 get_shared_row()

```
template<typename T >
CImg< T > get_shared_row (
    const unsigned int y0,
    const unsigned int z0 = 0,
    const unsigned int c0 = 0 )
```

Return a shared-memory image referencing one row of the image instance.

Parameters

<i>y0</i>	Y-coordinate.
<i>z0</i>	Z-coordinate.
<i>c0</i>	C-coordinate.

8.1.4.353 get_shared_slices()

```
template<typename T >
CImg< T > get_shared_slices (
    const unsigned int z0,
    const unsigned int z1,
    const unsigned int c0 = 0 )
```

Return a shared memory image referencing a range of slices of the image instance.

Parameters

<i>z0</i>	Z-coordinate of the starting slice.
<i>z1</i>	Z-coordinate of the ending slice.
<i>c0</i>	C-coordinate.

8.1.4.354 get_shared_slice()

```
template<typename T >
CImg< T > get_shared_slice (
    const unsigned int z0,
    const unsigned int c0 = 0 )
```

Return a shared-memory image referencing one slice of the image instance.

Parameters

<i>z0</i>	Z-coordinate.
<i>c0</i>	C-coordinate.

8.1.4.355 get_shared_channels()

```
template<typename T >
CImg< T > get_shared_channels (
    const unsigned int c0,
    const unsigned int c1 )
```

Return a shared-memory image referencing a range of channels of the image instance.

Parameters

<i>c0</i>	C-coordinate of the starting channel.
<i>c1</i>	C-coordinate of the ending channel.

8.1.4.356 get_shared_channel()

```
template<typename T >
CImg< T > get_shared_channel (
    const unsigned int c0 )
```

Return a shared-memory image referencing one channel of the image instance.

Parameters

<i>c0</i>	C-coordinate.
-----------	---------------

8.1.4.357 get_split() [1/2]

```
template<typename T >
CImgList< T > get_split (
    const char axis,
    const int nb = -1,
    const unsigned int max_parts = ~0U ) const
```

Split the image into a list along specified axis.

Parameters

<i>axis</i>	Splitting axis. Can be { 'x' 'y' 'z' 'c' }.
<i>nb</i>	Number of split parts.
<i>max_parts</i>	Number of max parts allowed for the split.

Note

- If `nb==0`, the instance image is split into blocks of equal values along the specified axis.
- If `nb>0`, the instance image is split into `nb` blocks.
- If `nb<0`, the instance image is split into blocks of `-nb` pixels wide.

8.1.4.358 `get_split()` [2/2]

```
template<typename T >
template<typename t >
CImgList< T > get_split (
    const CImg< t > & values,
    const char axis = 0,
    const bool keep_values = true ) const
```

Split the image into a list of sub-images according to a specified sequence of splitting values and optionally an axis.

Parameters

<i>values</i>	Splitting value sequence.
<i>axis</i>	Axis along which the splitting is performed. Can be '0' to ignore axis.
<i>keep_values</i>	Indicates whether the splitting sequence must be kept in the split blocks.

8.1.4.359 `append()`

```
template<typename T >
template<typename t >
CImg< T > & append (
    const CImg< t > & img,
    const char axis = 'x',
    const float align = 0 )
```

Append two images along specified axis.

Parameters

<i>img</i>	Image to append with instance image.
<i>axis</i>	Concatenation axis. Can be { 'x' 'y' 'z' 'c' }.
<i>align</i>	Concatenation alignment in [0,1].

8.1.4.360 correlate()

```
template<typename T >
template<typename t >
CImg< T > & correlate (
    const CImg< t > & kernel,
    const unsigned int boundary_conditions = 1,
    const bool is_normalized = false,
    const unsigned int channel_mode = 1,
    const int xcenter = (int)(~0U>>1),
    const int ycenter = (int)(~0U>>1),
    const int zcenter = (int)(~0U>>1),
    const unsigned int xstride = 1,
    const unsigned int ystride = 1,
    const unsigned int zstride = 1,
    const int xdilation = 1,
    const int ydilation = 1,
    const int zdilation = 1,
    const int xoffset = 0,
    const int yoffset = 0,
    const int zoffset = 0,
    const unsigned int xsize = ~0U,
    const unsigned int ysize = ~0U,
    const unsigned int zsize = ~0U )
```

Correlate the image with a kernel.

Parameters

<i>kernel</i>	= the correlation kernel.
<i>boundary_conditions</i>	Boundary condition. Can be { 0=dirichlet 1=neumann 2=periodic 3=mirror }.
<i>is_normalized</i>	= enable local normalization.
<i>channel_mode</i>	Channel processing mode. Can be { 0=all 1=one for one (default) 2=partial sum 3=full sum }.
<i>xcenter</i>	X-coordinate of the kernel center (~0U>>1 means 'centered').
<i>ycenter</i>	Y-coordinate of the kernel center (~0U>>1 means 'centered').
<i>zcenter</i>	Z-coordinate of the kernel center (~0U>>1 means 'centered').
<i>xstride</i>	Stride along the X-axis.
<i>ystride</i>	Stride along the Y-axis.
<i>zstride</i>	Stride along the Z-axis.
<i>xdilation</i>	Dilation along the X-axis.
<i>ydilation</i>	Dilation along the Y-axis.
<i>zdilation</i>	Dilation along the Z-axis.
<i>xoffset</i>	X-offset.
<i>yoffset</i>	Y-offset.
<i>zoffset</i>	Z-offset.
<i>xsize</i>	Width of the resulting image (~0U means 'ceil(instance_width/xstride)').
<i>ysize</i>	Height of the resulting image (~0U means 'ceil(instance_height/ystride)').
<i>zsize</i>	Depth of the resulting image (~0U means 'ceil(instance_depth/zstride)').

Note

- The correlation of the image instance **this* by the kernel *kernel* is defined to be: $res(x, y, z) = \sum_{i,j,k} (*this)(\alpha_x x + \beta_x (i - c_x), \alpha_y y + \beta_y (j - c_y), \alpha_z z + \beta_z (k - c_z)) * kernel(i, j, k)$

8.1.4.361 `convolve()`

```
template<typename T >
template<typename t >
CImg< T > & convolve (
    const CImg< t > & kernel,
    const unsigned int boundary_conditions = 1,
    const bool is_normalized = false,
    const unsigned int channel_mode = 1,
    const int xcenter = (int)(~0U>>1),
    const int ycenter = (int)(~0U>>1),
    const int zcenter = (int)(~0U>>1),
    const unsigned int xstride = 1,
    const unsigned int ystride = 1,
    const unsigned int zstride = 1,
    const int xdilation = 1,
    const int ydilation = 1,
    const int zdilation = 1,
    const int xoffset = 0,
    const int yoffset = 0,
    const int zoffset = 0,
    const unsigned int xsize = ~0U,
    const unsigned int ysize = ~0U,
    const unsigned int zsize = ~0U )
```

Convolve the image with a kernel.

Parameters

<i>kernel</i>	= the correlation kernel.
<i>boundary_conditions</i>	Boundary condition. Can be { 0=dirichlet 1=neumann 2=periodic 3=mirror }.
<i>is_normalized</i>	= enable local normalization.
<i>channel_mode</i>	Channel processing mode. Can be { 0=all 1=one for one (default) 2=partial sum 3=full sum }.
<i>xcenter</i>	X-coordinate of the kernel center (~0U>>1 means 'centered').
<i>ycenter</i>	Y-coordinate of the kernel center (~0U>>1 means 'centered').
<i>zcenter</i>	Z-coordinate of the kernel center (~0U>>1 means 'centered').
<i>xstride</i>	Stride along the X-axis.
<i>ystride</i>	Stride along the Y-axis.
<i>zstride</i>	Stride along the Z-axis.
<i>xdilation</i>	Dilation along the X-axis.
<i>ydilation</i>	Dilation along the Y-axis.
<i>zdilation</i>	Dilation along the Z-axis.
<i>xoffset</i>	X-offset.
<i>yoffset</i>	Y-offset.
<i>zoffset</i>	Z-offset.
<i>xsize</i>	Width of the resulting image (~0U means 'ceil(instance_width/xstride)').
<i>ysize</i>	Height of the resulting image (~0U means 'ceil(instance_height/ystride)').
<i>zsize</i>	Depth of the resulting image (~0U means 'ceil(instance_depth/zstride)').

Note

- The convolution of the image instance `*this` by the kernel `kernel` is defined to be: $res(x, y, z) = \sum_{i,j,k} (*this)(\alpha_x x - \beta_x (i - c_x), \alpha_y y - \beta_y (j - c_y), \alpha_z z - \beta_z (k - c_z)) * kernel(i, j, k)$.

8.1.4.362 `cumulate()` [1/2]

```
template<typename T >
CImg< T > & cumulate (
    const char axis = 0 )
```

Cumulate image values, optionally along specified axis.

Parameters

<i>axis</i>	Cumulation axis. Set it to 0 to cumulate all values globally without taking axes into account.
-------------	--

8.1.4.363 `cumulate()` [2/2]

```
template<typename T >
CImg< T > & cumulate (
    const char *const axes )
```

Cumulate image values, along specified axes.

Parameters

<i>axes</i>	Cumulation axes, as a C-string.
-------------	---------------------------------

Note

`axes` may contains multiple characters, e.g. "xyz"

8.1.4.364 `erode()` [1/3]

```
template<typename T >
template<typename t >
CImg< T > & erode (
    const CImg< t > & kernel,
    const unsigned int boundary_conditions = 1,
    const bool is_real = false )
```

Erode image by a structuring element.

Parameters

<i>kernel</i>	Structuring element.
<i>boundary_conditions</i>	Boundary conditions. Can be { 0=dirichlet 1=neumann 2=periodic 3=mirror }.
<i>is_real</i>	Do the erosion in real (a.k.a 'non-flat') mode (<code>true</code>) rather than binary mode (<code>false</code>).

8.1.4.365 erode() [2/3]

```
template<typename T >
CImg< T > & erode (
    const unsigned int sx,
    const unsigned int sy,
    const unsigned int sz = 1 )
```

Erode image by a rectangular structuring element of specified size.

Parameters

<i>sx</i>	Width of the structuring element.
<i>sy</i>	Height of the structuring element.
<i>sz</i>	Depth of the structuring element.

8.1.4.366 erode() [3/3]

```
template<typename T >
CImg< T > & erode (
    const unsigned int s )
```

Erode the image by a square structuring element of specified size.

Parameters

<i>s</i>	Size of the structuring element.
----------	----------------------------------

8.1.4.367 dilate() [1/3]

```
template<typename T >
template<typename t >
CImg< T > & dilate (
    const CImg< t > & kernel,
    const unsigned int boundary_conditions = 1,
    const bool is_real = false )
```

Dilate image by a structuring element.

Parameters

<i>kernel</i>	Structuring element.
<i>boundary_conditions</i>	Boundary conditions. Can be { 0=dirichlet 1=neumann 2=periodic 3=mirror }.
<i>is_real</i>	Do the dilation in real (a.k.a 'non-flat') mode (<code>true</code>) rather than binary mode (<code>false</code>).

8.1.4.368 dilate() [2/3]

```
template<typename T >
```

```
CImg< T > & dilate (
    const unsigned int  sx,
    const unsigned int  sy,
    const unsigned int  sz = 1 )
```

Dilate image by a rectangular structuring element of specified size.

Parameters

<i>sx</i>	Width of the structuring element.
<i>sy</i>	Height of the structuring element.
<i>sz</i>	Depth of the structuring element.

8.1.4.369 dilate() [3/3]

```
template<typename T >
CImg< T > & dilate (
    const unsigned int  s )
```

Dilate image by a square structuring element of specified size.

Parameters

<i>s</i>	Size of the structuring element.
----------	----------------------------------

8.1.4.370 closing() [1/2]

```
template<typename T >
template<typename t >
CImg< T > & closing (
    const CImg< t > & kernel,
    const unsigned int  boundary_conditions = 1,
    const bool  is_real = false )
```

Apply morphological closing by a structuring element.

Parameters

<i>kernel</i>	Structuring element.
<i>boundary_conditions</i>	Boundary conditions. Can be { 0=dirichlet 1=neumann 2=periodic 3=mirror }.
<i>is_real</i>	Do the closing in real (a.k.a 'non-flat') mode (<code>true</code>) rather than binary mode (<code>false</code>).

8.1.4.371 closing() [2/2]

```
template<typename T >
CImg< T > & closing (
    const unsigned int  s )
```

Apply morphological closing by a square structuring element of specified size.

Parameters

<i>s</i>	Size of the structuring element.
----------	----------------------------------

8.1.4.372 opening() [1/2]

```
template<typename T >
template<typename t >
CImg< T > & opening (
    const CImg< t > & kernel,
    const unsigned int boundary_conditions = 1,
    const bool is_real = false )
```

Apply morphological opening by a structuring element.

Parameters

<i>kernel</i>	Structuring element.
<i>boundary_conditions</i>	Boundary conditions. Can be { 0=dirichlet 1=neumann 2=periodic 3=mirror }.
<i>is_real</i>	Do the opening in real (a.k.a 'non-flat') mode (<code>true</code>) rather than binary mode (<code>false</code>).

8.1.4.373 opening() [2/2]

```
template<typename T >
CImg< T > & opening (
    const unsigned int s )
```

Apply morphological opening by a square structuring element of specified size.

Parameters

<i>s</i>	Size of the structuring element.
----------	----------------------------------

8.1.4.374 watershed()

```
template<typename T >
template<typename t >
CImg< T > & watershed (
    const CImg< t > & priority,
    const bool is_high_connectivity = false )
```

Compute watershed transform.

Parameters

<i>priority</i>	Priority map.
<i>is_high_connectivity</i>	Indicates whether the algorithm uses low or high connectivity.

Note

Non-zero values of the instance instance are propagated to zero-valued ones according to specified the priority map.

8.1.4.375 deriche()

```
template<typename T >
CImg< T > & deriche (
    const float sigma,
    const unsigned int order = 0,
    const char axis = 'x',
    const unsigned int boundary_conditions = 1 )
```

Apply recursive Deriche filter.

Parameters

<i>sigma</i>	Standard deviation of the filter.
<i>order</i>	Order of the filter. Can be { 0=smooth-filter 1=1st-derivative 2=2nd-derivative }.
<i>axis</i>	Axis along which the filter is computed. Can be { 'x' 'y' 'z' 'c' }.
<i>boundary_conditions</i>	Boundary conditions. Can be { 0=dirichlet 1=neumann 2=periodic 3=mirror }.

8.1.4.376 vanvliet()

```
template<typename T >
CImg< T > & vanvliet (
    const float sigma,
    const unsigned int order,
    const char axis = 'x',
    const unsigned int boundary_conditions = 1 )
```

Van Vliet recursive Gaussian filter.

Parameters

<i>sigma</i>	standard deviation of the Gaussian filter
<i>order</i>	the order of the filter 0,1,2,3
<i>axis</i>	Axis along which the filter is computed. Can be { 'x' 'y' 'z' 'c' }.
<i>boundary_conditions</i>	Boundary conditions. Can be { 0=dirichlet 1=neumann 2=periodic 3=mirror }.

Note

dirichlet boundary condition has a strange behavior

I.T. Young, L.J. van Vliet, M. van Ginkel, Recursive Gabor filtering. IEEE Trans. Sig. Proc., vol. 50, pp. 2799-2805, 2002.

(this is an improvement over Young-Van Vliet, Sig. Proc. 44, 1995)

Boundary conditions (only for order 0) using Triggs matrix, from B. Triggs and M. Sdika. Boundary conditions for Young-van Vliet recursive filtering. IEEE Trans. Signal Processing, vol. 54, pp. 2365-2367, 2006.

8.1.4.377 blur() [1/2]

```
template<typename T >
CImg< T > & blur (
    const float sigma_x,
    const float sigma_y,
    const float sigma_z,
    const unsigned int boundary_conditions = 1,
    const bool is_gaussian = true )
```

Blur image.

Parameters

<i>sigma_x</i>	Standard deviation of the blur, along the X-axis.
<i>sigma_y</i>	Standard deviation of the blur, along the Y-axis.
<i>sigma_z</i>	Standard deviation of the blur, along the Z-axis.
<i>boundary_conditions</i>	Boundary conditions. Can be { 0=dirichlet 1=neumann 2=periodic 3=mirror }.
<i>is_gaussian</i>	Indicates whether the blur uses a gaussian (<code>true</code>) or quasi-gaussian (<code>false</code>) kernel.

Note

- The blur is computed as a 0-order Vanvliet (gaussian) or Deriche filter (quasi-gaussian).
- This is a recursive algorithm, not depending on the values of the standard deviations.

See also

[deriche\(\)](#), [vanvliet\(\)](#).

8.1.4.378 blur() [2/2]

```
template<typename T >
CImg< T > & blur (
    const float sigma,
    const unsigned int boundary_conditions = 1,
    const bool is_gaussian = true )
```

Blur image isotropically.

Parameters

<i>sigma</i>	Standard deviation of the blur.
<i>boundary_conditions</i>	Boundary conditions. Can be { 0=dirichlet 1=neumann 2=periodic 3=mirror }.
<i>is_gaussian</i>	Use a gaussian kernel (VanVliet) is set, a quasi-gaussian (Deriche) otherwise.

See also

[deriche\(\)](#), [vanvliet\(\)](#).

8.1.4.379 blur_anisotropic() [1/2]

```
template<typename T >
template<typename t >
CImg< T > & blur_anisotropic (
    const CImg< t > & G,
    const float amplitude = 60,
    const float dl = 0.8f,
    const float da = 30,
    const float gauss_prec = 2,
    const unsigned int interpolation_type = 0,
    const bool is_fast_approx = 1 )
```

Blur image anisotropically, directed by a field of diffusion tensors.

Parameters

<i>G</i>	Field of square roots of diffusion tensors/vectors used to drive the smoothing.
<i>amplitude</i>	Amplitude of the smoothing.
<i>dl</i>	Spatial discretization.
<i>da</i>	Angular discretization.
<i>gauss_prec</i>	Precision of the diffusion process.
<i>interpolation_type</i>	Interpolation scheme. Can be { 0=nearest-neighbor 1=linear 2=Runge-Kutta }.
<i>is_fast_approx</i>	Indicates whether a fast approximation of the gaussian function is used.

8.1.4.380 blur_anisotropic() [2/2]

```
template<typename T >
CImg< T > & blur_anisotropic (
    const float amplitude,
    const float sharpness = 0.7f,
    const float anisotropy = 0.6f,
    const float alpha = 0.6f,
    const float sigma = 1.1f,
    const float dl = 0.8f,
    const float da = 30,
    const float gauss_prec = 2,
    const unsigned int interpolation_type = 0,
    const bool is_fast_approx = true )
```

Blur image anisotropically, in an edge-preserving way.

Parameters

<i>amplitude</i>	Amplitude of the smoothing.
<i>sharpness</i>	Sharpness.
<i>anisotropy</i>	Anisotropy.

Parameters

<i>alpha</i>	Standard deviation of the gradient blur.
<i>sigma</i>	Standard deviation of the structure tensor blur.
<i>dl</i>	Spatial discretization.
<i>da</i>	Angular discretization.
<i>gauss_prec</i>	Precision of the diffusion process.
<i>interpolation_type</i>	Interpolation scheme. Can be { 0=nearest-neighbor 1=linear 2=Runge-Kutta }.
<i>is_fast_approx</i>	Indicates whether a fast approximation of the gaussian function is used.

8.1.4.381 blur_bilateral() [1/2]

```

template<typename T >
template<typename t >
CImg< T > & blur_bilateral (
    const CImg< t > & guide,
    const float sigma_x,
    const float sigma_y,
    const float sigma_z,
    const float sigma_r,
    const float sampling_x,
    const float sampling_y,
    const float sampling_z,
    const float sampling_r )

```

Blur image, with the joint bilateral filter.

Parameters

<i>guide</i>	Image used to model the smoothing weights.
<i>sigma_x</i>	Amount of blur along the X-axis.
<i>sigma_y</i>	Amount of blur along the Y-axis.
<i>sigma_z</i>	Amount of blur along the Z-axis.
<i>sigma_r</i>	Amount of blur along the value axis.
<i>sampling_x</i>	Amount of downsampling along the X-axis used for the approximation. Defaults (0) to sigma_x.
<i>sampling_y</i>	Amount of downsampling along the Y-axis used for the approximation. Defaults (0) to sigma_y.
<i>sampling_z</i>	Amount of downsampling along the Z-axis used for the approximation. Defaults (0) to sigma_z.
<i>sampling_r</i>	Amount of downsampling along the value axis used for the approximation. Defaults (0) to sigma_r.

Note

This algorithm uses the optimisation technique proposed by S. Paris and F. Durand, in ECCV'2006 (extended for 3D volumetric images). It is based on the reference implementation <http://people.csail.mit.edu/jiawen/software/bilateralFilter.m>

8.1.4.382 blur_bilateral() [2/2]

```
template<typename T >
template<typename t >
CImg< T > & blur_bilateral (
    const CImg< t > & guide,
    const float sigma_s,
    const float sigma_r,
    const float sampling_s = 0,
    const float sampling_r = 0 )
```

Blur image using the joint bilateral filter.

Parameters

<i>guide</i>	Image used to model the smoothing weights.
<i>sigma_s</i>	Amount of blur along the XYZ-axes.
<i>sigma_r</i>	Amount of blur along the value axis.
<i>sampling_s</i>	Amount of downsampling along the XYZ-axes used for the approximation. Defaults to <i>sigma_s</i> .
<i>sampling_r</i>	Amount of downsampling along the value axis used for the approximation. Defaults to <i>sigma_r</i> .

8.1.4.383 boxfilter()

```
template<typename T >
CImg< T > & boxfilter (
    const float boxsize,
    const int order,
    const char axis = 'x',
    const unsigned int boundary_conditions = 1,
    const unsigned int nb_iter = 1 )
```

Parameters

<i>boxsize</i>	Size of the box window (can be subpixel)
<i>order</i>	the order of the filter 0,1 or 2.
<i>axis</i>	Axis along which the filter is computed. Can be { 'x' 'y' 'z' 'c' }.
<i>boundary_conditions</i>	Boundary conditions. Can be { 0=dirichlet 1=neumann 2=periodic 3=mirror }.
<i>nb_iter</i>	Number of filter iterations.

8.1.4.384 blur_box() [1/2]

```
template<typename T >
CImg< T > & blur_box (
    const float boxsize_x,
    const float boxsize_y,
    const float boxsize_z,
```

```
const unsigned int boundary_conditions = 1,
const unsigned int nb_iter = 1 )
```

Blur image with a box filter.

Parameters

<i>boxsize_x</i>	Size of the box window, along the X-axis (can be subpixel).
<i>boxsize_y</i>	Size of the box window, along the Y-axis (can be subpixel).
<i>boxsize_z</i>	Size of the box window, along the Z-axis (can be subpixel).
<i>boundary_conditions</i>	Boundary conditions. Can be { false=dirichlet true=neumann 2=periodic 3=mirror }.
<i>nb_iter</i>	Number of filter iterations.

Note

- This is a recursive algorithm, not depending on the values of the box kernel size.

See also

[blur\(\)](#).

8.1.4.385 blur_box() [2/2]

```
template<typename T >
CImg< T > & blur_box (
    const float boxsize,
    const unsigned int boundary_conditions = 1 )
```

Blur image with a box filter.

Parameters

<i>boxsize</i>	Size of the box window (can be subpixel).
<i>boundary_conditions</i>	Boundary conditions. Can be { 0=dirichlet 1=neumann 2=periodic 3=mirror }.

See also

[deriche\(\)](#), [vanvliet\(\)](#).

8.1.4.386 blur_guided()

```
template<typename T >
template<typename t >
CImg< T > & blur_guided (
    const CImg< t > & guide,
```

```
const float radius,  
const float regularization )
```

Blur image, with the image guided filter.

Parameters

<i>guide</i>	Image used to guide the smoothing process.
<i>radius</i>	Spatial radius. If negative, it is expressed as a percentage of the largest image size.
<i>regularization</i>	Regularization parameter. If negative, it is expressed as a percentage of the guide value range.

Note

This function implements the filtering algorithm described in: He, Kaiming; Sun, Jian; Tang, Xiaoou, "Guided Image Filtering," Pattern Analysis and Machine Intelligence, IEEE Transactions on , vol.35, no.6, pp.1397,1409, June 2013

8.1.4.387 blur_patch()

```
template<typename T >
template<typename t >
CImg< T > & blur_patch (
    const CImg< t > & guide,
    const float sigma_s,
    const float sigma_r,
    const unsigned int patch_size = 3,
    const unsigned int lookup_size = 4,
    const float smoothness = 0,
    const bool is_fast_approx = true )
```

Blur image using patch-based space.

Parameters

<i>guide</i>	Image used to model the smoothing weights.
<i>sigma_s</i>	Amount of blur along the XYZ-axes.
<i>sigma_r</i>	Amount of blur along the value axis.
<i>patch_size</i>	Size of the patches.
<i>lookup_size</i>	Size of the window to search similar patches.
<i>smoothness</i>	Smoothness for the patch comparison.
<i>is_fast_approx</i>	Indicates whether a fast approximation of the gaussian function is used.

8.1.4.388 blur_median()

```
template<typename T >
CImg< T > & blur_median (
    const unsigned int n,
    const float threshold = 0 )
```

Blur image with the median filter.

Parameters

<i>n</i>	Size of the median filter.
<i>threshold</i>	Threshold used to discard pixels too far from the current pixel value in the median computation.

8.1.4.389 sharpen()

```
template<typename T >
CImg< T > & sharpen (
    const float amplitude,
    const bool sharpen_type = false,
    const float edge = 1,
    const float alpha = 0,
    const float sigma = 0 )
```

Sharpen image.

Parameters

<i>amplitude</i>	Sharpening amplitude
<i>sharpen_type</i>	Select sharpening method. Can be { false=inverse diffusion true=shock filters }.
<i>edge</i>	Edge threshold (shock filters only).
<i>alpha</i>	Gradient smoothness (shock filters only).
<i>sigma</i>	Tensor smoothness (shock filters only).

8.1.4.390 get_gradient()

```
template<typename T >
CImgList< Tfloat > get_gradient (
    const char *const axes = 0,
    const int scheme = 0 ) const
```

Return the image gradient.

Parameters

<i>axes</i>	Axes considered for the gradient computation, as a C-string (e.g "xy").
<i>scheme</i>	= Numerical scheme used for the gradient computation: • -1 = Backward finite differences • 0 = Centered finite differences (default) • 1 = Forward finite differences • 2 = Using Sobel kernels • 3 = Using rotation invariant kernels • 4 = Using Deriche recursive filter. • 5 = Using Van Vliet recursive filter.

8.1.4.391 get_hessian()

```
template<typename T >
```

```
CImgList< Tfloat > get_hessian (
    const char *const axes = 0 ) const
```

Return the image hessian.

Parameters

<i>axes</i>	Axes considered for the hessian computation, as a C-string (e.g "xy").
-------------	--

8.1.4.392 structure_tensors()

```
template<typename T >
CImg< T > & structure_tensors (
    const bool is_fwbw_scheme = false )
```

Compute the structure tensor field of an image.

Parameters

<i>is_fwbw_scheme</i>	scheme. Can be { false=centered true=forward-backward }
-----------------------	---

8.1.4.393 diffusion_tensors()

```
template<typename T >
CImg< T > & diffusion_tensors (
    const float sharpness = 0.7f,
    const float anisotropy = 0.6f,
    const float alpha = 0.6f,
    const float sigma = 1.1f,
    const bool is_sqrt = false )
```

Compute field of diffusion tensors for edge-preserving smoothing.

Parameters

<i>sharpness</i>	Sharpness
<i>anisotropy</i>	Anisotropy
<i>alpha</i>	Standard deviation of the gradient blur.
<i>sigma</i>	Standard deviation of the structure tensor blur.
<i>is_sqrt</i>	Indicates whether the square root of the tensor field is computed instead.

8.1.4.394 displacement()

```
template<typename T >
CImg< T > & displacement (
    const CImg< T > & reference,
    const float smoothness = 0.1f,
```



```

const float precision = 7.f,
const unsigned int nb_scales = 0,
const unsigned int iteration_max = 1000,
const bool is_forward = false,
const CImg< floatT > & guide = CImg<floatT>::const_empty() )

```

Estimate the displacement field from a given reference image to the current image instance.

Parameters

<i>reference</i>	Reference image R.
<i>smoothness</i>	Smoothness of estimated displacement field. If smoothness is positive, Tikhonov regularization is applied, otherwise TV regularization is applied, with specified strength (absolute value of the smoothnes).
<i>precision</i>	Precision required for algorithm convergence.
<i>nb_scales</i>	Number of scales used to estimate the displacement field.
<i>iteration_max</i>	Maximum number of iterations allowed for one scale.
<i>is_forward</i>	If true, optimize $(I(X + U(X)) - R(X))$. If false, optimize $(I(X) - R(X - U(X)))$.
<i>guide</i>	Image used as the initial correspondence estimate for the algorithm. 'guide' may have a last channel with boolean values (0=false other=true) that indicates for each pixel if its correspondence vector is constrained to its initial value (constraint mask).

8.1.4.395 matchpatch()

```

template<typename T >
template<typename t1 , typename t2 >
CImg< T > & matchpatch (
    const CImg< T > & patch_image,
    const unsigned int patch_width,
    const unsigned int patch_height,
    const unsigned int patch_depth,
    const unsigned int nb_iterations,
    const unsigned int nb_randoms,
    const float patch_penalization,
    const CImg< t1 > & guide,
    CImg< t2 > & matching_score )

```

Compute correspondence map between two images, using a patch-matching algorithm.

Parameters

	<i>patch_image</i>	The image containing the reference patches to match with the instance image.
	<i>patch_width</i>	Width of the patch used for matching.
	<i>patch_height</i>	Height of the patch used for matching.
	<i>patch_depth</i>	Depth of the patch used for matching.
	<i>nb_iterations</i>	Number of patch-match iterations.
	<i>nb_randoms</i>	Number of randomization attempts (per pixel).
	<i>patch_penalization</i>	Penalization factor in score related patch occurrences. if negative, also indicates that identity result is not avoided.
	<i>guide</i>	Image used as the initial correspondence estimate for the algorithm. 'guide' may have a last channel with boolean values (0=false other=true) that indicates for each pixel if its correspondence vector is constrained to its initial value (constraint mask).
<i>out</i>	<i>matching_score</i>	Returned as the image of matching scores.

8.1.4.396 distance() [1/2]

```
template<typename T >
CImg< T > & distance (
    const T & value,
    const unsigned int metric = 2 )
```

Compute Euclidean distance function to a specified value.

Parameters

<i>value</i>	Reference value.
<i>metric</i>	Type of metric. Can be { 0=Chebyshev 1=Manhattan 2=Euclidean 3=Squared-euclidean }.

Note

The distance transform implementation was submitted by A. Meijster, and implements the article 'W.H. Hesselink, A. Meijster, J.B.T.M. Roerdink, "A general algorithm for computing distance transforms in linear time.", In: Mathematical Morphology and its Applications to Image and Signal Processing, J. Goutsias, L. Vincent, and D.S. Bloomberg (eds.), Kluwer, 2000, pp. 331-340.' The submitted code was then modified to fit [CImg](#) coding style and constraints.

8.1.4.397 distance() [2/2]

```
template<typename T >
template<typename t >
CImg< T > & distance (
    const T & value,
    const CImg< t > & metric_mask )
```

Compute chamfer distance to a specified value, with a custom metric.

Parameters

<i>value</i>	Reference value.
<i>metric_mask</i>	Metric mask.

Note

The algorithm code has been initially proposed by A. Meijster, and modified by D. Tschumperlé.

8.1.4.398 distance_dijkstra()

```
template<typename T >
template<typename t , typename to >
CImg< T > & distance_dijkstra (
    const T & value,
    const CImg< t > & metric,
```

```
const bool is_high_connectivity,  
CImg< to > & return_path )
```

Compute distance to a specified value, according to a custom metric (use dijkstra algorithm).

Parameters

	<i>value</i>	Reference value.
	<i>metric</i>	Field of distance potentials.
	<i>is_high_connectivity</i>	Indicates whether the algorithm uses low or high connectivity.
out	<i>return_path</i>	An image containing the nodes of the minimal path.

8.1.4.399 distance_eikonal() [1/2]

```
template<typename T >
template<typename t >
CImg< T > & distance_eikonal (
    const T & value,
    const CImg< t > & metric )
```

Compute distance map to one source point, according to a custom metric (use fast marching algorithm).

Parameters

<i>value</i>	Reference value.
<i>metric</i>	Field of distance potentials.

8.1.4.400 distance_eikonal() [2/2]

```
template<typename T >
CImg< T > & distance_eikonal (
    const unsigned int nb_iterations,
    const float band_size = 0,
    const float time_step = 0.5f )
```

Compute distance function to 0-valued isophotes, using the Eikonal PDE.

Parameters

<i>nb_iterations</i>	Number of PDE iterations.
<i>band_size</i>	Size of the narrow band.
<i>time_step</i>	Time step of the PDE iterations.

8.1.4.401 haar() [1/2]

```
template<typename T >
CImg< T > & haar (
    const char axis,
    const bool invert = false,
    const unsigned int nb_scales = 1 )
```

Compute Haar multiscale wavelet transform.

Parameters

<i>axis</i>	Axis considered for the transform.
<i>invert</i>	Set inverse of direct transform.
<i>nb_scales</i>	Number of scales used for the transform.

8.1.4.402 haar() [2/2]

```
template<typename T >
CImg< T > & haar (
    const bool invert = false,
    const unsigned int nb_scales = 1 )
```

Parameters

<i>invert</i>	Set inverse of direct transform.
<i>nb_scales</i>	Number of scales used for the transform.

8.1.4.403 get_FFT()

```
template<typename T >
CImgList< Tfloat > get_FFT (
    const char axis,
    const bool is_inverse = false ) const
```

Compute 1D Fast Fourier Transform, along a specified axis.

Parameters

<i>axis</i>	Axis along which the FFT is computed.
<i>is_inverse</i>	Indicates whether the forward (<i>false</i>) or inverse (<i>true</i>) FFT is computed.

8.1.4.404 FFT() [1/2]

```
template<typename T >
static void FFT (
    CImg< T > & real,
    CImg< T > & imag,
    const char axis,
    const bool is_inverse = false,
    const unsigned int nb_threads = 0 ) [static]
```

Compute 1D Fast Fourier Transform, along a specified axis.

Parameters

in, out	<i>real</i>	Real part of the pixel values.
in, out	<i>imag</i>	Imaginary part of the pixel values.

Parameters

	<i>axis</i>	Axis along which the FFT is computed.
	<i>is_inverse</i>	Indicates whether the forward (<i>false</i>) or inverse (<i>true</i>) FFT is computed.
	<i>nb_threads</i>	Set the maximum number of threads used for FFT computation.

8.1.4.405 FFT() [2/2]

```
template<typename T >
static void FFT (
    CImg< T > & real,
    CImg< T > & imag,
    const bool is_inverse = false,
    const unsigned int nb_threads = 0 ) [static]
```

Compute n-D Fast Fourier Transform.

Parameters

in, out	<i>real</i>	Real part of the pixel values.
in, out	<i>imag</i>	Imaginary part of the pixel values.
	<i>is_inverse</i>	Indicates whether the forward (<i>false</i>) or inverse (<i>true</i>) FFT is computed.
	<i>nb_threads</i>	Number of parallel threads used for the computation. Use 0 to set this to the number of available cpus.

8.1.4.406 rotate_object3d()

```
template<typename T >
CImg< T > & rotate_object3d (
    const float x,
    const float y,
    const float z,
    const float w,
    const bool is_quaternion = false )
```

Rotate 3D object's vertices.

Parameters

<i>x</i>	X-coordinate of the rotation axis, or first quaternion coordinate.
<i>y</i>	Y-coordinate of the rotation axis, or second quaternion coordinate.
<i>z</i>	Z-coordinate of the rotation axis, or second quaternion coordinate.
<i>w</i>	Angle of the rotation axis (in degree), or fourth quaternion coordinate.
<i>is_quaternion</i>	Specifies whether the four arguments denote a set of { axis + angle } or a quaternion (x,y,z,w).

8.1.4.407 shift_object3d() [1/2]

```
template<typename T >
```

```
CImg< T > & shift_object3d (
    const float tx,
    const float ty = 0,
    const float tz = 0 )
```

Shift 3D object's vertices.

Parameters

<i>tx</i>	X-coordinate of the 3D displacement vector.
<i>ty</i>	Y-coordinate of the 3D displacement vector.
<i>tz</i>	Z-coordinate of the 3D displacement vector.

8.1.4.408 shift_object3d() [2/2]

```
template<typename T >
CImg< T > & shift_object3d ( )
```

Shift 3D object's vertices, so that it becomes centered.

Note

The object center is computed as its barycenter.

8.1.4.409 resize_object3d()

```
template<typename T >
CImg< T > & resize_object3d (
    const float sx,
    const float sy = -100,
    const float sz = -100 )
```

Resize 3D object.

Parameters

<i>sx</i>	Width of the 3D object's bounding box.
<i>sy</i>	Height of the 3D object's bounding box.
<i>sz</i>	Depth of the 3D object's bounding box.

8.1.4.410 append_object3d()

```
template<typename T >
template<typename tf , typename tp , typename tff >
CImg< T > & append_object3d (
    CImgList< tf > & primitives,
    const CImg< tp > & obj_vertices,
    const CImgList< tff > & obj_primitives )
```

Merge two 3D objects.

Parameters

in, out	<i>primitives</i>	Primitives data of the current 3D object.
	<i>obj_vertices</i>	Vertices data of the additional 3D object.
	<i>obj_primitives</i>	Primitives data of the additional 3D object.

8.1.4.411 texturize_object3d()

```
template<typename T >
template<typename tp , typename tc , typename tt , typename tx >
const CImg< T > & texturize_object3d (
    CImgList< tp > & primitives,
    CImgList< tc > & colors,
    const CImg< tt > & texture,
    const CImg< tx > & coords = CImg<tx>::const_empty() ) const
```

Texturize the primitives of a 3D object.

Parameters

in, out	<i>primitives</i>	Primitives data of the 3D object.
in, out	<i>colors</i>	Colors data of the 3D object.
	<i>texture</i>	Texture image to map to 3D object.
	<i>coords</i>	Texture-mapping coordinates.

8.1.4.412 get_elevation3d()

```
template<typename T >
template<typename tf , typename tc , typename te >
CImg< floatT > get_elevation3d (
    CImgList< tf > & primitives,
    CImgList< tc > & colors,
    const CImg< te > & elevation ) const
```

Generate a 3D elevation map from the instance image, as a 3D object.

Parameters

out	<i>primitives</i>	The returned list of the 3D object primitives (template type <i>tf</i> should be at least <i>unsigned int</i>).
out	<i>colors</i>	The returned list of the 3D object colors.
	<i>elevation</i>	The input elevation map.

Returns

The N vertices (xi,yi,zi) of the 3D object as a Nx3 CImg<float> image (0<=i<=N - 1).

Example

```

const CImg<float> img("reference.jpg");
CImgList<unsigned int> faces3d;
CImgList<unsigned char> colors3d;
const CImg<float> points3d = img.get_elevation3d(faces3d, colors3d, img.get_norm()*0.2);
CImg<unsigned char>().display_object3d("Elevation3d", points3d, faces3d, colors3d);

```

8.1.4.413 get_projections3d()

```

template<typename T >
template<typename tf , typename tc >
CImg< floatT > get_projections3d (
    CImgList< tf > & primitives,
    CImgList< tc > & colors,
    const unsigned int x0,
    const unsigned int y0,
    const unsigned int z0,
    const bool normalize_colors = false ) const

```

Generate the 3D projection planes of the image instance.

Parameters

out	<i>primitives</i>	Primitives data of the returned 3D object.
out	<i>colors</i>	Colors data of the returned 3D object.
	<i>x0</i>	X-coordinate of the projection point.
	<i>y0</i>	Y-coordinate of the projection point.
	<i>z0</i>	Z-coordinate of the projection point.
	<i>normalize_colors</i>	Indicates whether the created textures have normalized colors.

8.1.4.414 get_isoline3d()

```

template<typename T >
template<typename tf >
CImg< floatT > get_isoline3d (
    CImgList< tf > & primitives,
    const float isovalue,
    const int size_x = -100,
    const int size_y = -100 ) const

```

Generate an isoline from the current image instance, as a 3D object.

Parameters

out	<i>primitives</i>	The returned list of the 3D object primitives (template type <i>tf</i> should be at least <i>unsigned int</i>).
	<i>isovalue</i>	The scalar value used as the threshold for extraction.
	<i>size_x</i>	The number of subdivisions along the X-axis.
	<i>size_y</i>	The number of subdivisions along the Y-axis.

Returns

The N vertices (xi,yi,zi) of the 3D object as a Nx3 CImg<float> image ($0 \leq i \leq N - 1$).

Example

```
const CImg<float> img("reference.jpg");
CImgList<unsigned int> faces3d;
const CImg<float> points3d = img.get_isoline3d(faces3d,100);
CImg<unsigned char>().display_object3d("Isoline3d",points3d,faces3d,colors3d);
```

8.1.4.415 isoline3d() [1/2]

```
template<typename T >
template<typename tf , typename tfunc >
static CImg< floatT > isoline3d (
    CImgList< tf > & primitives,
    const tfunc & func,
    const float isovalue,
    const float x0,
    const float y0,
    const float x1,
    const float y1,
    const int size_x = 256,
    const int size_y = 256 ) [static]
```

Compute isolines of a function, as a 3D object.

Parameters

out	<i>primitives</i>	Primitives data of the resulting 3D object.
	<i>func</i>	Elevation functor. Must have <code>operator() (x,y)</code> defined.
	<i>isovalue</i>	The scalar value used as the threshold for extraction.
	<i>x0</i>	X-coordinate of the starting point.
	<i>y0</i>	Y-coordinate of the starting point.
	<i>x1</i>	X-coordinate of the ending point.
	<i>y1</i>	Y-coordinate of the ending point.
	<i>size_x</i>	Resolution of the function along the X-axis.
	<i>size_y</i>	Resolution of the function along the Y-axis.

Note

Uses the marching squares algorithm for extracting the isolines.

8.1.4.416 isoline3d() [2/2]

```
template<typename T >
template<typename tv , typename tf , typename tfunc >
static void isoline3d (
    tv & add_vertex,
    tf & add_segment,
```

```

const tfunc & func,
const float isovalue,
const float x0,
const float y0,
const float x1,
const float y1,
const int size_x,
const int size_y ) [static]

```

Compute isolines of a function, as a 3D object.

Parameters

out	<i>add_vertex</i>	: Functor with operator()(x,y,z) defined for adding a new vertex.
out	<i>add_segment</i>	: Functor with operator()(i,j) defined for adding a new segment.
	<i>func</i>	Elevation function. Is of type float (*func) (const float x, const float y).
	<i>isovalue</i>	The scalar value used as the threshold for extraction.
	<i>x0</i>	X-coordinate of the starting point.
	<i>y0</i>	Y-coordinate of the starting point.
	<i>x1</i>	X-coordinate of the ending point.
	<i>y1</i>	Y-coordinate of the ending point.
	<i>size_x</i>	Resolution of the function along the X-axis.
	<i>size_y</i>	Resolution of the function along the Y-axis.

Note

Uses the marching squares algorithm for extracting the isolines.

8.1.4.417 get_isosurface3d()

```

template<typename T >
template<typename tf >
CImg< floatT > get_isosurface3d (
    CImgList< tf > & primitives,
    const float isovalue,
    const int size_x = -100,
    const int size_y = -100,
    const int size_z = -100 ) const

```

Generate an isosurface from the instance image, as a 3D object.

Parameters

out	<i>primitives</i>	The returned list of the 3D object primitives (template type <i>tf</i> should be at least <i>unsigned int</i>).
	<i>isovalue</i>	The scalar value used as the threshold for extraction.
	<i>size_x</i>	Number of subdivisions along the X-axis.
	<i>size_y</i>	Number of subdivisions along the Y-axis.
	<i>size_z</i>	Number of subdivisions along the Z-axis.

Returns

The N vertices (xi,yi,zi) of the 3D object as a Nx3 CImg<float> image ($0 \leq i \leq N - 1$).

Example

```
const CImg<float> img = CImg<unsigned char>("reference.jpg").resize(-100,-100,20);
CImgList<unsigned int> faces3d;
const CImg<float> points3d = img.get_isosurface3d(faces3d,100);
CImg<unsigned char>().display_object3d("Isosurface3d",points3d,faces3d,colors3d);
```

8.1.4.418 isosurface3d() [1/2]

```
template<typename T >
template<typename tf , typename tfunc >
static CImg< floatT > isosurface3d (
    CImgList< tf > & primitives,
    const tfunc & func,
    const float isovalue,
    const float x0,
    const float y0,
    const float z0,
    const float x1,
    const float y1,
    const float z1,
    const int size_x = 32,
    const int size_y = 32,
    const int size_z = 32 ) [static]
```

Compute isosurface of a function, as a 3D object.

Parameters

out	<i>primitives</i>	Primitives data of the resulting 3D object.
	<i>func</i>	Implicit function. Is of type <code>float (*func)(const float x, const float y, const float z)</code> .
	<i>isovalue</i>	The scalar value used as the threshold for extraction.
	<i>x0</i>	X-coordinate of the starting point.
	<i>y0</i>	Y-coordinate of the starting point.
	<i>z0</i>	Z-coordinate of the starting point.
	<i>x1</i>	X-coordinate of the ending point.
	<i>y1</i>	Y-coordinate of the ending point.
	<i>z1</i>	Z-coordinate of the ending point.
	<i>size_x</i>	Resolution of the elevation function along the X-axis.
	<i>size_y</i>	Resolution of the elevation function along the Y-axis.
	<i>size_z</i>	Resolution of the elevation function along the Z-axis.

Note

Uses the marching cubes algorithm for extracting the isosurface.

8.1.4.419 isosurface3d() [2/2]

```

template<typename T >
template<typename tv , typename tf , typename tfunc >
static void isosurface3d (
    tv & add_vertex,
    tf & add_triangle,
    const tfunc & func,
    const float isovalue,
    const float x0,
    const float y0,
    const float z0,
    const float x1,
    const float y1,
    const float z1,
    const int size_x,
    const int size_y,
    const int size_z ) [static]

```

Compute isosurface of a function, as a 3D object.

Parameters

out	<i>add_vertex</i>	: Functor with operator()(x,y,z) defined for adding a new vertex.
out	<i>add_triangle</i>	: Functor with operator()(i,j) defined for adding a new segment.
	<i>func</i>	Implicit function. Is of type <code>float (*func)(const float x, const float y, const float z)</code> .
	<i>isovalue</i>	The scalar value used as the threshold for extraction.
	<i>x0</i>	X-coordinate of the starting point.
	<i>y0</i>	Y-coordinate of the starting point.
	<i>z0</i>	Z-coordinate of the starting point.
	<i>x1</i>	X-coordinate of the ending point.
	<i>y1</i>	Y-coordinate of the ending point.
	<i>z1</i>	Z-coordinate of the ending point.
	<i>size_x</i>	Resolution of the elevation function along the X-axis.
	<i>size_y</i>	Resolution of the elevation function along the Y-axis.
	<i>size_z</i>	Resolution of the elevation function along the Z-axis.

Note

Uses the marching cubes algorithm for extracting the isosurface.

8.1.4.420 elevation3d()

```

template<typename T >
template<typename tf , typename tfunc >
static CImg< floatT > elevation3d (
    CImgList< tf > & primitives,
    const tfunc & func,
    const float x0,
    const float y0,

```

```

const float x1,
const float y1,
const int size_x = 256,
const int size_y = 256 ) [static]

```

Compute 3D elevation of a function as a 3D object.

Parameters

out	<i>primitives</i>	Primitives data of the resulting 3D object.
	<i>func</i>	Elevation function. Is of type float (*func) (const float x, const float y).
	<i>x0</i>	X-coordinate of the starting point.
	<i>y0</i>	Y-coordinate of the starting point.
	<i>x1</i>	X-coordinate of the ending point.
	<i>y1</i>	Y-coordinate of the ending point.
	<i>size_x</i>	Resolution of the function along the X-axis.
	<i>size_y</i>	Resolution of the function along the Y-axis.

8.1.4.421 box3d()

```

template<typename T >
template<typename tf >
static CImg< floatT > box3d (
    CImgList< tf > & primitives,
    const float size_x = 200,
    const float size_y = 100,
    const float size_z = 100 ) [static]

```

Generate a 3D box object.

Parameters

out	<i>primitives</i>	The returned list of the 3D object primitives (template type <i>tf</i> should be at least <i>unsigned int</i>).
	<i>size_x</i>	The width of the box (dimension along the X-axis).
	<i>size_y</i>	The height of the box (dimension along the Y-axis).
	<i>size_z</i>	The depth of the box (dimension along the Z-axis).

Returns

The N vertices (xi,yi,zi) of the 3D object as a Nx3 CImg<float> image (0<=i<=N - 1).

Example

```

CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::box3d(faces3d,10,20,30);
CImg<unsigned char>().display_object3d("Box3d",points3d,faces3d);

```

8.1.4.422 cone3d()

```

template<typename T >
template<typename tf >

```

```
static CImg< floatT > cone3d (
    CImgList< tf > & primitives,
    const float radius = 50,
    const float size_z = 100,
    const unsigned int subdivisions = 24 ) [static]
```

Generate a 3D cone.

Parameters

out	<i>primitives</i>	The returned list of the 3D object primitives (template type <i>tf</i> should be at least <i>unsigned int</i>).
	<i>radius</i>	The radius of the cone base.
	<i>size_z</i>	The cone's height.
	<i>subdivisions</i>	The number angular subdivisions for the base.

Returns

The N vertices (xi,yi,zi) of the 3D object as a Nx3 CImg<float> image (0<=i<=N - 1).

Example

```
CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::cone3d(faces3d,50);
CImg<unsigned char>().display_object3d("Cone3d",points3d,faces3d);
```

8.1.4.423 cylinder3d()

```
template<typename T >
template<typename tf >
static CImg< floatT > cylinder3d (
    CImgList< tf > & primitives,
    const float radius = 50,
    const float size_z = 100,
    const unsigned int subdivisions = 24 ) [static]
```

Generate a 3D cylinder.

Parameters

out	<i>primitives</i>	The returned list of the 3D object primitives (template type <i>tf</i> should be at least <i>unsigned int</i>).
	<i>radius</i>	The radius of the cylinder base.
	<i>size_z</i>	The cylinder's height.
	<i>subdivisions</i>	The number of angular subdivisions for the base.

Returns

The N vertices (xi,yi,zi) of the 3D object as a Nx3 CImg<float> image (0<=i<=N - 1).

Example

```
CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::cylinder3d(faces3d,50);
CImg<unsigned char>().display_object3d("Cylinder3d",points3d,faces3d);
```

8.1.4.424 torus3d()

```
template<typename T >
template<typename tf >
static CImg< floatT > torus3d (
    CImgList< tf > & primitives,
    const float radius1 = 100,
    const float radius2 = 30,
    const unsigned int subdivisions1 = 24,
    const unsigned int subdivisions2 = 12 ) [static]
```

Generate a 3D torus.

Parameters

out	<i>primitives</i>	The returned list of the 3D object primitives (template type <i>tf</i> should be at least <i>unsigned int</i>).
	<i>radius1</i>	The large radius.
	<i>radius2</i>	The small radius.
	<i>subdivisions1</i>	The number of angular subdivisions for the large radius.
	<i>subdivisions2</i>	The number of angular subdivisions for the small radius.

Returns

The N vertices (xi,yi,zi) of the 3D object as a Nx3 CImg<float> image ($0 \leq i \leq N - 1$).

Example

```
CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::torus3d(faces3d,20,4);
CImg<unsigned char>().display_object3d("Torus3d",points3d,faces3d);
```

8.1.4.425 plane3d()

```
template<typename T >
template<typename tf >
static CImg< floatT > plane3d (
    CImgList< tf > & primitives,
    const float size_x = 100,
    const float size_y = 100,
    const unsigned int subdivisions_x = 10,
    const unsigned int subdivisions_y = 10 ) [static]
```

Generate a 3D XY-plane.

Parameters

out	<i>primitives</i>	The returned list of the 3D object primitives (template type <i>tf</i> should be at least <i>unsigned int</i>).
	<i>size_x</i>	The width of the plane (dimension along the X-axis).
	<i>size_y</i>	The height of the plane (dimensions along the Y-axis).
	<i>subdivisions</i> <i>_x</i>	The number of planar subdivisions along the X-axis.
	<i>subdivisions</i> <i>_y</i>	The number of planar subdivisions along the Y-axis.

Returns

The N vertices (xi,yi,zi) of the 3D object as a Nx3 CImg<float> image (0<=i<=N - 1).

Example

```
CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::plane3d(faces3d,100,50);
CImg<unsigned char>().display_object3d("Plane3d",points3d,faces3d);
```

8.1.4.426 sphere3d()

```
template<typename T >
template<typename tf >
static CImg< floatT > sphere3d (
    CImgList< tf > & primitives,
    const float radius = 50,
    const unsigned int subdivisions = 3 ) [static]
```

Generate a 3D sphere.

Parameters

out	<i>primitives</i>	The returned list of the 3D object primitives (template type <i>tf</i> should be at least <i>unsigned int</i>).
	<i>radius</i>	The radius of the sphere (dimension along the X-axis).
	<i>subdivisions</i>	The number of recursive subdivisions from an initial icosahedron.

Returns

The N vertices (xi,yi,zi) of the 3D object as a Nx3 CImg<float> image (0<=i<=N - 1).

Example

```
CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::sphere3d(faces3d,100,4);
CImg<unsigned char>().display_object3d("Sphere3d",points3d,faces3d);
```

8.1.4.427 ellipsoid3d()

```
template<typename T >
template<typename tf , typename t >
static CImg< floatT > ellipsoid3d (
    CImgList< tf > & primitives,
    const CImg< t > & tensor,
    const unsigned int subdivisions = 3 ) [static]
```

Generate a 3D ellipsoid.

Parameters

out	<i>primitives</i>	The returned list of the 3D object primitives (template type <i>tf</i> should be at least <i>unsigned int</i>).
	<i>tensor</i>	The tensor which gives the shape and size of the ellipsoid.
	<i>subdivisions</i>	The number of recursive subdivisions from an initial stretched icosahedron.

Returns

The N vertices (xi,yi,zi) of the 3D object as a Nx3 CImg<float> image (0<=i<=N - 1).

Example

```
CImgList<unsigned int> faces3d;
const CImg<float> tensor = CImg<float>::diagonal(10,7,3),
    points3d = CImg<float>::ellipsoid3d(faces3d,tensor,4);
CImg<unsigned char>().display_object3d("Ellipsoid3d",points3d,faces3d);
```

8.1.4.428 object3dtoCImg3d()

```
template<typename T >
template<typename tp , typename tc , typename to >
CImg< T > & object3dtoCImg3d (
    const CImgList< tp > & primitives,
    const CImgList< tc > & colors,
    const to & opacities,
    const bool full_check = true )
```

Convert 3D object into a CImg3d representation.

Parameters

<i>primitives</i>	Primitives data of the 3D object.
<i>colors</i>	Colors data of the 3D object.
<i>opacities</i>	Opacities data of the 3D object.
<i>full_check</i>	Indicates whether full checking of the 3D object must be performed.

8.1.4.429 CImg3dtoobject3d()

```
template<typename T >
```

```
template<typename tp , typename tc , typename to >
CImg< T > & CImg3dtoobject3d (
    CImgList< tp > & primitives,
    CImgList< tc > & colors,
    CImgList< to > & opacities,
    const bool full_check = true )
```

Convert CImg3d representation into a 3D object.

Parameters

out	<i>primitives</i>	Primitives data of the 3D object.
out	<i>colors</i>	Colors data of the 3D object.
out	<i>opacities</i>	Opacities data of the 3D object.
	<i>full_check</i>	Indicates whether full checking of the 3D object must be performed.

8.1.4.430 draw_point() [1/2]

```
template<typename T >
template<typename tc >
CImg< T > & draw_point (
    const int x0,
    const int y0,
    const int z0,
    const tc *const color,
    const float opacity = 1 )
```

Draw a 3D point.

Parameters

<i>x0</i>	X-coordinate of the point.
<i>y0</i>	Y-coordinate of the point.
<i>z0</i>	Z-coordinate of the point.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.

Note

- To set pixel values without clipping needs, you should use the faster [CImg::operator>\(\)](#) function.

Example:

```
CImg<unsigned char> img(100,100,1,3,0);
const unsigned char color[] = { 255,128,64 };
img.draw_point(50,50,color);
```

8.1.4.431 draw_point() [2/2]

```
template<typename T >
template<typename t , typename tc >
```

```
CImg< T > & draw_point (
    const CImg< t > & points,
    const tc *const color,
    const float opacity = 1 )
```

Parameters

<i>points</i>	Image of vertices coordinates.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.

8.1.4.432 draw_line() [1/6]

```
template<typename T >
template<typename tc >
CImg< T > & draw_line (
    int x0,
    int y0,
    int x1,
    int y1,
    const tc *const color,
    const float opacity = 1,
    const unsigned int pattern = ~0U,
    const bool init_hatch = true,
    const bool draw_last_pixel = true )
```

Draw a 2D line.

Parameters

<i>x0</i>	X-coordinate of the starting line point.
<i>y0</i>	Y-coordinate of the starting line point.
<i>x1</i>	X-coordinate of the ending line point.
<i>y1</i>	Y-coordinate of the ending line point.
<i>color</i>	Pointer to spectrum() consecutive values of type T, defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the line stipple pattern.
<i>init_hatch</i>	Indicates whether the hatch variable must be reinitialized.
<i>draw_last_pixel</i>	Indicates whether last pixel of the line must be drawn (e.g. can be disabled when drawing multi-line curves with transparency).

Note

- Set `init_hatch = false` to draw consecutive hatched segments without breaking the line stipple pattern.

Example:

```
CImg<unsigned char> img(100,100,1,3,0);
const unsigned char color[] = { 255,128,64 };
img.draw_line(40,40,80,70,color);
```

8.1.4.433 draw_line() [2/6]

```

template<typename T >
template<typename tz , typename tc >
CImg< T > & draw_line (
    CImg< tz > & zbuffer,
    int x0,
    int y0,
    const float z0,
    int x1,
    int y1,
    const float z1,
    const tc *const color,
    const float opacity = 1,
    const unsigned int pattern = ~0U,
    const bool init_hatch = true )

```

Draw a 2D line, with z-buffering.

Parameters

<i>zbuffer</i>	Zbuffer image.
<i>x0</i>	X-coordinate of the starting point.
<i>y0</i>	Y-coordinate of the starting point.
<i>z0</i>	Z-coordinate of the starting point
<i>x1</i>	X-coordinate of the ending point.
<i>y1</i>	Y-coordinate of the ending point.
<i>z1</i>	Z-coordinate of the ending point.
<i>color</i>	Pointer to spectrum() consecutive values of type <i>T</i> , defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the line stipple pattern.
<i>init_hatch</i>	Indicates whether the hatch variable must be reinitialized.

8.1.4.434 draw_line() [3/6]

```

template<typename T >
template<typename tc >
CImg< T > & draw_line (
    int x0,
    int y0,
    int x1,
    int y1,
    const CImg< tc > & texture,
    int tx0,
    int ty0,
    int tx1,
    int ty1,
    const float opacity = 1,
    const unsigned int pattern = ~0U,
    const bool init_hatch = true )

```

Draw a textured 2D line.

Parameters

<i>x0</i>	X-coordinate of the starting line point.
<i>y0</i>	Y-coordinate of the starting line point.
<i>x1</i>	X-coordinate of the ending line point.
<i>y1</i>	Y-coordinate of the ending line point.
<i>texture</i>	Texture image defining the pixel colors.
<i>tx0</i>	X-coordinate of the starting texture point.
<i>ty0</i>	Y-coordinate of the starting texture point.
<i>tx1</i>	X-coordinate of the ending texture point.
<i>ty1</i>	Y-coordinate of the ending texture point.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the line stipple pattern.
<i>init_hatch</i>	Indicates whether the hatch variable must be reinitialized.

Note

- Line routine uses the well known Bresenham's algorithm.

Example:

```
CImg<unsigned char> img(100,100,1,3,0), texture("texture256x256.ppm");
const unsigned char color[] = { 255,128,64 };
img.draw_line(40,40,80,70,texture,0,0,255,255);
```

8.1.4.435 draw_line() [4/6]

```
template<typename T >
template<typename tc >
CImg< T > & draw_line (
    int x0,
    int y0,
    const float z0,
    int x1,
    int y1,
    const float z1,
    const CImg< tc > & texture,
    const int tx0,
    const int ty0,
    const int tx1,
    const int ty1,
    const float opacity = 1,
    const unsigned int pattern = ~0U,
    const bool init_hatch = true )
```

Draw a textured 2D line, with perspective correction.

Parameters

<i>x0</i>	X-coordinate of the starting point.
<i>y0</i>	Y-coordinate of the starting point.
<i>z0</i>	Z-coordinate of the starting point

Parameters

<i>x1</i>	X-coordinate of the ending point.
<i>y1</i>	Y-coordinate of the ending point.
<i>z1</i>	Z-coordinate of the ending point.
<i>texture</i>	Texture image defining the pixel colors.
<i>tx0</i>	X-coordinate of the starting texture point.
<i>ty0</i>	Y-coordinate of the starting texture point.
<i>tx1</i>	X-coordinate of the ending texture point.
<i>ty1</i>	Y-coordinate of the ending texture point.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the line stipple pattern.
<i>init_hatch</i>	Indicates whether the hatch variable must be reinitialized.

8.1.4.436 **draw_line()** [5/6]

```

template<typename T >
template<typename tz , typename tc >
CImg< T > & draw_line (
    CImg< tz > & zbuffer,
    int x0,
    int y0,
    const float z0,
    int x1,
    int y1,
    const float z1,
    const CImg< tc > & texture,
    const int tx0,
    const int ty0,
    const int tx1,
    const int ty1,
    const float opacity = 1,
    const unsigned int pattern = ~0U,
    const bool init_hatch = true )

```

Draw a textured 2D line, with perspective correction and z-buffering.

Parameters

<i>zbuffer</i>	Z-buffer image.
<i>x0</i>	X-coordinate of the starting point.
<i>y0</i>	Y-coordinate of the starting point.
<i>z0</i>	Z-coordinate of the starting point
<i>x1</i>	X-coordinate of the ending point.
<i>y1</i>	Y-coordinate of the ending point.
<i>z1</i>	Z-coordinate of the ending point.
<i>texture</i>	Texture image defining the pixel colors.
<i>tx0</i>	X-coordinate of the starting texture point.
<i>ty0</i>	Y-coordinate of the starting texture point.
<i>tx1</i>	X-coordinate of the ending texture point.
<i>ty1</i>	Y-coordinate of the ending texture point.

Parameters

<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the line stipple pattern.
<i>init_hatch</i>	Indicates whether the hatch variable must be reinitialized.

8.1.4.437 draw_line() [6/6]

```
template<typename T >
template<typename tp , typename tc >
CImg< T > & draw_line (
    const CImg< tp > & points,
    const tc *const color,
    const float opacity = 1,
    const unsigned int pattern = ~0U,
    const bool init_hatch = true )
```

Draw a polyline.

Parameters

<i>points</i>	Coordinates of vertices, stored as a list of vectors.
<i>color</i>	Pointer to spectrum() consecutive values of type T, defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the line stipple pattern.
<i>init_hatch</i>	Indicates whether the hatch variable must be reinitialized.

Note

- This function uses several call to the single [CImg::draw_line\(\)](#) procedure, depending on the vectors size in points.

8.1.4.438 draw_arrow()

```
template<typename T >
template<typename tc >
CImg< T > & draw_arrow (
    const int x0,
    const int y0,
    const int x1,
    const int y1,
    const tc *const color,
    const float opacity = 1,
    const float angle = 30,
    const float length = -10,
    const unsigned int pattern = ~0U )
```

Draw a 2D arrow.

Parameters

<i>x0</i>	X-coordinate of the starting arrow point (tail).
<i>y0</i>	Y-coordinate of the starting arrow point (tail).
<i>x1</i>	X-coordinate of the ending arrow point (head).
<i>y1</i>	Y-coordinate of the ending arrow point (head).
<i>color</i>	Pointer to spectrum() consecutive values of type <code>T</code> , defining the drawing color.
<i>angle</i>	Aperture angle of the arrow head.
<i>length</i>	Length of the arrow head. If negative, describes a percentage of the arrow length.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the line stipple pattern.

8.1.4.439 draw_spline() [1/4]

```

template<typename T >
template<typename tc >
CImg< T > & draw_spline (
    const int x0,
    const int y0,
    const float u0,
    const float v0,
    const int x1,
    const int y1,
    const float u1,
    const float v1,
    const tc *const color,
    const float opacity = 1,
    const float precision = 0.25f,
    const unsigned int pattern = ~0U,
    const bool init_hatch = true )

```

Draw a 2D spline.

Parameters

<i>x0</i>	X-coordinate of the starting curve point
<i>y0</i>	Y-coordinate of the starting curve point
<i>u0</i>	X-coordinate of the starting velocity
<i>v0</i>	Y-coordinate of the starting velocity
<i>x1</i>	X-coordinate of the ending curve point
<i>y1</i>	Y-coordinate of the ending curve point
<i>u1</i>	X-coordinate of the ending velocity
<i>v1</i>	Y-coordinate of the ending velocity
<i>color</i>	Pointer to spectrum() consecutive values of type <code>T</code> , defining the drawing color.
<i>precision</i>	Curve drawing precision.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the line stipple pattern.
<i>init_hatch</i>	Indicates whether the hatch variable must be reinitialized.

Note

- The curve is a 2D cubic Bezier spline, from the set of specified starting/ending points and corresponding velocity vectors.
- The spline is drawn as a sequence of connected segments. The `precision` parameter sets the average number of pixels in each drawn segment.
- A cubic Bezier curve is sometimes defined by a set of 4 points { (x0,y0), (xa,ya), (xb,yb), (x1,y1) } where (x0,y0) is the starting point, (x1,y1) is the ending point and (xa,ya), (xb,yb) are two *control* points. The starting and ending velocities (u0,v0) and (u1,v1) can be deduced easily from the control points as $u0 = (xa - x0)$, $v0 = (ya - y0)$, $u1 = (x1 - xb)$ and $v1 = (y1 - yb)$.

Example:

```
CImg<unsigned char> img(100,100,1,3,0);
const unsigned char color[] = { 255,255,255 };
img.draw_spline(30,30,0,100,90,40,0,-100,color);
```

8.1.4.440 draw_spline() [2/4]

```
template<typename T >
template<typename t >
CImg< T > & draw_spline (
    const int x0,
    const int y0,
    const float u0,
    const float v0,
    const int x1,
    const int y1,
    const float u1,
    const float v1,
    const CImg< t > & texture,
    const int tx0,
    const int ty0,
    const int tx1,
    const int ty1,
    const float opacity = 1,
    const float precision = 4,
    const unsigned int pattern = ~0U,
    const bool init_hatch = true )
```

Draw a textured 2D spline.

Parameters

<i>x0</i>	X-coordinate of the starting curve point
<i>y0</i>	Y-coordinate of the starting curve point
<i>u0</i>	X-coordinate of the starting velocity
<i>v0</i>	Y-coordinate of the starting velocity
<i>x1</i>	X-coordinate of the ending curve point
<i>y1</i>	Y-coordinate of the ending curve point
<i>u1</i>	X-coordinate of the ending velocity
<i>v1</i>	Y-coordinate of the ending velocity
<i>texture</i>	Texture image defining line pixel colors.
<i>tx0</i>	X-coordinate of the starting texture point.

Parameters

<i>ty0</i>	Y-coordinate of the starting texture point.
<i>tx1</i>	X-coordinate of the ending texture point.
<i>ty1</i>	Y-coordinate of the ending texture point.
<i>precision</i>	Curve drawing precision.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the line stipple pattern.
<i>init_hatch</i>	Indicates whether the hatch variable must be reinitialized.

8.1.4.441 draw_spline() [3/4]

```

template<typename T >
template<typename tp , typename tt , typename tc >
CImg< T > & draw_spline (
    const CImg< tp > & points,
    const CImg< tt > & tangents,
    const tc *const color,
    const float opacity = 1,
    const bool is_closed_set = false,
    const float precision = 4,
    const unsigned int pattern = ~0U,
    const bool init_hatch = true )

```

Draw a set of consecutive splines.

Parameters

<i>points</i>	Vertices data.
<i>tangents</i>	Tangents data.
<i>color</i>	Pointer to <code>spectrum()</code> consecutive values of type T, defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>is_closed_set</i>	Indicates whether the drawn spline set is closed.
<i>precision</i>	Precision of the drawing.
<i>pattern</i>	An integer whose bits describe the line stipple pattern.
<i>init_hatch</i>	Indicates whether the hatch variable must be reinitialized.

8.1.4.442 draw_spline() [4/4]

```

template<typename T >
template<typename tp , typename tc >
CImg< T > & draw_spline (
    const CImg< tp > & points,
    const tc *const color,
    const float opacity = 1,
    const bool is_closed_set = false,
    const float precision = 4,
    const unsigned int pattern = ~0U,
    const bool init_hatch = true )

```

Similar to previous function, with the point tangents automatically estimated from the given points set.

8.1.4.443 draw_triangle() [1/9]

```
template<typename T >
template<typename tc >
CImg< T > & draw_triangle (
    const int x0,
    const int y0,
    const int x1,
    const int y1,
    const int x2,
    const int y2,
    const tc *const color,
    const float opacity = 1 )
```

Draw a filled 2D triangle.

Parameters

<i>x0</i>	X-coordinate of the first vertex.
<i>y0</i>	Y-coordinate of the first vertex.
<i>x1</i>	X-coordinate of the second vertex.
<i>y1</i>	Y-coordinate of the second vertex.
<i>x2</i>	X-coordinate of the third vertex.
<i>y2</i>	Y-coordinate of the third vertex.
<i>color</i>	Pointer to spectrum() consecutive values of type <i>T</i> , defining the drawing color.
<i>opacity</i>	Drawing opacity.

8.1.4.444 draw_triangle() [2/9]

```
template<typename T >
template<typename tc >
CImg< T > & draw_triangle (
    const int x0,
    const int y0,
    const int x1,
    const int y1,
    const int x2,
    const int y2,
    const tc *const color,
    const float opacity,
    const unsigned int pattern )
```

Draw a outlined 2D triangle.

Parameters

<i>x0</i>	X-coordinate of the first vertex.
<i>y0</i>	Y-coordinate of the first vertex.
<i>x1</i>	X-coordinate of the second vertex.
<i>y1</i>	Y-coordinate of the second vertex.
<i>x2</i>	X-coordinate of the third vertex.
<i>y2</i>	Y-coordinate of the third vertex.
<i>color</i>	Pointer to spectrum() consecutive values of type <i>T</i> , defining the drawing color.

Parameters

<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the outline stipple pattern.

8.1.4.445 draw_triangle() [3/9]

```

template<typename T >
template<typename tz , typename tc >
CImg< T > & draw_triangle (
    CImg< tz > & zbuffer,
    int x0,
    int y0,
    const float z0,
    int x1,
    int y1,
    const float z1,
    int x2,
    int y2,
    const float z2,
    const tc *const color,
    const float opacity = 1,
    const float brightness = 1 )

```

Draw a filled 2D triangle, with z-buffering.

Parameters

<i>zbuffer</i>	Z-buffer image.
<i>x0</i>	X-coordinate of the first vertex.
<i>y0</i>	Y-coordinate of the first vertex.
<i>z0</i>	Z-coordinate of the first vertex.
<i>x1</i>	X-coordinate of the second vertex.
<i>y1</i>	Y-coordinate of the second vertex.
<i>z1</i>	Z-coordinate of the second vertex.
<i>x2</i>	X-coordinate of the third vertex.
<i>y2</i>	Y-coordinate of the third vertex.
<i>z2</i>	Z-coordinate of the third vertex.
<i>color</i>	Pointer to spectrum() consecutive values of type <code>T</code> , defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>brightness</i>	Brightness factor.

8.1.4.446 draw_triangle() [4/9]

```

template<typename T >
template<typename tc >
CImg< T > & draw_triangle (
    int x0,
    int y0,
    int x1,

```

```

int y1,
int x2,
int y2,
const tc *const color,
float bs0,
float bs1,
float bs2,
const float opacity = 1 )

```

Draw a Gouraud-shaded 2D triangle.

Parameters

<i>x0</i>	X-coordinate of the first vertex in the image instance.
<i>y0</i>	Y-coordinate of the first vertex in the image instance.
<i>x1</i>	X-coordinate of the second vertex in the image instance.
<i>y1</i>	Y-coordinate of the second vertex in the image instance.
<i>x2</i>	X-coordinate of the third vertex in the image instance.
<i>y2</i>	Y-coordinate of the third vertex in the image instance.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>bs0</i>	Brightness factor of the first vertex (in [0,2]).
<i>bs1</i>	brightness factor of the second vertex (in [0,2]).
<i>bs2</i>	brightness factor of the third vertex (in [0,2]).
<i>opacity</i>	Drawing opacity.

8.1.4.447 draw_triangle() [5/9]

```

template<typename T >
template<typename tc >
CImg< T > & draw_triangle (
    int x0,
    int y0,
    int x1,
    int y1,
    int x2,
    int y2,
    const tc * color0,
    const tc * color1,
    const tc * color2,
    const float opacity = 1 )

```

Draw a color-interpolated 2D triangle.

Parameters

<i>x0</i>	X-coordinate of the first vertex in the image instance.
<i>y0</i>	Y-coordinate of the first vertex in the image instance.
<i>x1</i>	X-coordinate of the second vertex in the image instance.
<i>y1</i>	Y-coordinate of the second vertex in the image instance.
<i>x2</i>	X-coordinate of the third vertex in the image instance.
<i>y2</i>	Y-coordinate of the third vertex in the image instance.
<i>color0</i>	Pointer to spectrum() consecutive values of type <i>T</i> , defining the color of the first vertex.

Parameters

<i>color1</i>	Pointer to spectrum() consecutive values of type <code>T</code> , defining the color of the second vertex.
<i>color2</i>	Pointer to spectrum() consecutive values of type <code>T</code> , defining the color of the third vertex.
<i>opacity</i>	Drawing opacity.

8.1.4.448 draw_triangle() [6/9]

```
template<typename T >
template<typename tc >
CImg< T > & draw_triangle (
    int x0,
    int y0,
    int x1,
    int y1,
    int x2,
    int y2,
    const CImg< tc > & texture,
    int tx0,
    int ty0,
    int tx1,
    int ty1,
    int tx2,
    int ty2,
    const float opacity = 1,
    const float brightness = 1 )
```

Draw a textured 2D triangle.

Parameters

<i>x0</i>	X-coordinate of the first vertex in the image instance.
<i>y0</i>	Y-coordinate of the first vertex in the image instance.
<i>x1</i>	X-coordinate of the second vertex in the image instance.
<i>y1</i>	Y-coordinate of the second vertex in the image instance.
<i>x2</i>	X-coordinate of the third vertex in the image instance.
<i>y2</i>	Y-coordinate of the third vertex in the image instance.
<i>texture</i>	Texture image used to fill the triangle.
<i>tx0</i>	X-coordinate of the first vertex in the texture image.
<i>ty0</i>	Y-coordinate of the first vertex in the texture image.
<i>tx1</i>	X-coordinate of the second vertex in the texture image.
<i>ty1</i>	Y-coordinate of the second vertex in the texture image.
<i>tx2</i>	X-coordinate of the third vertex in the texture image.
<i>ty2</i>	Y-coordinate of the third vertex in the texture image.
<i>opacity</i>	Drawing opacity.
<i>brightness</i>	Brightness factor of the drawing (in [0,2]).

8.1.4.449 draw_triangle() [7/9]

```
template<typename T >
```



```

template<typename tc , typename tl >
CImg< T > & draw_triangle (
    int x0,
    int y0,
    int x1,
    int y1,
    int x2,
    int y2,
    const tc *const color,
    const CImg< tl > & light,
    int lx0,
    int ly0,
    int lx1,
    int ly1,
    int lx2,
    int ly2,
    const float opacity = 1 )

```

Draw a Phong-shaded 2D triangle.

Parameters

<i>x0</i>	X-coordinate of the first vertex in the image instance.
<i>y0</i>	Y-coordinate of the first vertex in the image instance.
<i>x1</i>	X-coordinate of the second vertex in the image instance.
<i>y1</i>	Y-coordinate of the second vertex in the image instance.
<i>x2</i>	X-coordinate of the third vertex in the image instance.
<i>y2</i>	Y-coordinate of the third vertex in the image instance.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>light</i>	Light image.
<i>lx0</i>	X-coordinate of the first vertex in the light image.
<i>ly0</i>	Y-coordinate of the first vertex in the light image.
<i>lx1</i>	X-coordinate of the second vertex in the light image.
<i>ly1</i>	Y-coordinate of the second vertex in the light image.
<i>lx2</i>	X-coordinate of the third vertex in the light image.
<i>ly2</i>	Y-coordinate of the third vertex in the light image.
<i>opacity</i>	Drawing opacity.

8.1.4.450 draw_triangle() [8/9]

```

template<typename T >
template<typename tc >
CImg< T > & draw_triangle (
    int x0,
    int y0,
    int x1,
    int y1,
    int x2,
    int y2,
    const CImg< tc > & texture,
    int tx0,
    int ty0,

```

```

int tx1,
int ty1,
int tx2,
int ty2,
float bs0,
float bs1,
float bs2,
const float opacity = 1 )

```

Draw a textured Gouraud-shaded 2D triangle.

Parameters

<i>x0</i>	X-coordinate of the first vertex in the image instance.
<i>y0</i>	Y-coordinate of the first vertex in the image instance.
<i>x1</i>	X-coordinate of the second vertex in the image instance.
<i>y1</i>	Y-coordinate of the second vertex in the image instance.
<i>x2</i>	X-coordinate of the third vertex in the image instance.
<i>y2</i>	Y-coordinate of the third vertex in the image instance.
<i>texture</i>	Texture image used to fill the triangle.
<i>tx0</i>	X-coordinate of the first vertex in the texture image.
<i>ty0</i>	Y-coordinate of the first vertex in the texture image.
<i>tx1</i>	X-coordinate of the second vertex in the texture image.
<i>ty1</i>	Y-coordinate of the second vertex in the texture image.
<i>tx2</i>	X-coordinate of the third vertex in the texture image.
<i>ty2</i>	Y-coordinate of the third vertex in the texture image.
<i>bs0</i>	Brightness factor of the first vertex.
<i>bs1</i>	Brightness factor of the second vertex.
<i>bs2</i>	Brightness factor of the third vertex.
<i>opacity</i>	Drawing opacity.

8.1.4.451 draw_triangle() [9/9]

```

template<typename T >
template<typename tc , typename tl >
CImg< T > & draw_triangle (
    int x0,
    int y0,
    int x1,
    int y1,
    int x2,
    int y2,
    const CImg< tc > & texture,
    int tx0,
    int ty0,
    int tx1,
    int ty1,
    int tx2,
    int ty2,
    const CImg< tl > & light,
    int lx0,
    int ly0,

```

```

int lx1,
int ly1,
int lx2,
int ly2,
const float opacity = 1 )

```

Draw a textured Phong-shaded 2D triangle.

Parameters

<i>x0</i>	X-coordinate of the first vertex in the image instance.
<i>y0</i>	Y-coordinate of the first vertex in the image instance.
<i>x1</i>	X-coordinate of the second vertex in the image instance.
<i>y1</i>	Y-coordinate of the second vertex in the image instance.
<i>x2</i>	X-coordinate of the third vertex in the image instance.
<i>y2</i>	Y-coordinate of the third vertex in the image instance.
<i>texture</i>	Texture image used to fill the triangle.
<i>tx0</i>	X-coordinate of the first vertex in the texture image.
<i>ty0</i>	Y-coordinate of the first vertex in the texture image.
<i>tx1</i>	X-coordinate of the second vertex in the texture image.
<i>ty1</i>	Y-coordinate of the second vertex in the texture image.
<i>tx2</i>	X-coordinate of the third vertex in the texture image.
<i>ty2</i>	Y-coordinate of the third vertex in the texture image.
<i>light</i>	Light image.
<i>lx0</i>	X-coordinate of the first vertex in the light image.
<i>ly0</i>	Y-coordinate of the first vertex in the light image.
<i>lx1</i>	X-coordinate of the second vertex in the light image.
<i>ly1</i>	Y-coordinate of the second vertex in the light image.
<i>lx2</i>	X-coordinate of the third vertex in the light image.
<i>ly2</i>	Y-coordinate of the third vertex in the light image.
<i>opacity</i>	Drawing opacity.

8.1.4.452 draw_rectangle() [1/3]

```

template<typename T >
CImg< T > & draw_rectangle (
    const int x0,
    const int y0,
    const int z0,
    const int c0,
    const int x1,
    const int y1,
    const int z1,
    const int c1,
    const T val,
    const float opacity = 1 )

```

Draw a filled 4D rectangle.

Parameters

<i>x0</i>	X-coordinate of the upper-left rectangle corner.
<i>y0</i>	Y-coordinate of the upper-left rectangle corner.
<i>z0</i>	Z-coordinate of the upper-left rectangle corner.
<i>c0</i>	C-coordinate of the upper-left rectangle corner.
<i>x1</i>	X-coordinate of the lower-right rectangle corner.
<i>y1</i>	Y-coordinate of the lower-right rectangle corner.
<i>z1</i>	Z-coordinate of the lower-right rectangle corner.
<i>c1</i>	C-coordinate of the lower-right rectangle corner.
<i>val</i>	Scalar value used to fill the rectangle area.
<i>opacity</i>	Drawing opacity.

8.1.4.453 draw_rectangle() [2/3]

```

template<typename T >
template<typename tc >
CImg< T > & draw_rectangle (
    const int x0,
    const int y0,
    const int z0,
    const int x1,
    const int y1,
    const int z1,
    const tc *const color,
    const float opacity = 1 )

```

Draw a filled 3D rectangle.

Parameters

<i>x0</i>	X-coordinate of the upper-left rectangle corner.
<i>y0</i>	Y-coordinate of the upper-left rectangle corner.
<i>z0</i>	Z-coordinate of the upper-left rectangle corner.
<i>x1</i>	X-coordinate of the lower-right rectangle corner.
<i>y1</i>	Y-coordinate of the lower-right rectangle corner.
<i>z1</i>	Z-coordinate of the lower-right rectangle corner.
<i>color</i>	Pointer to spectrum() consecutive values of type <code>T</code> , defining the drawing color.
<i>opacity</i>	Drawing opacity.

8.1.4.454 draw_rectangle() [3/3]

```

template<typename T >
template<typename tc >
CImg< T > & draw_rectangle (
    const int x0,
    const int y0,
    const int x1,

```

```

const int y1,
const tc *const color,
const float opacity = 1 )

```

Draw a filled 2D rectangle.

Parameters

<i>x0</i>	X-coordinate of the upper-left rectangle corner.
<i>y0</i>	Y-coordinate of the upper-left rectangle corner.
<i>x1</i>	X-coordinate of the lower-right rectangle corner.
<i>y1</i>	Y-coordinate of the lower-right rectangle corner.
<i>color</i>	Pointer to spectrum() consecutive values of type T, defining the drawing color.
<i>opacity</i>	Drawing opacity.

8.1.4.455 draw_polygon()

```

template<typename T >
template<typename tp , typename tc >
CImg< T > & draw_polygon (
    const CImg< tp > & points,
    const tc *const color,
    const float opacity = 1 )

```

Draw a filled 2D polygon.

Parameters

<i>points</i>	Set of polygon vertices.
<i>color</i>	Pointer to spectrum() consecutive values of type T, defining the drawing color.
<i>opacity</i>	Drawing opacity.

8.1.4.456 draw_ellipse() [1/4]

```

template<typename T >
template<typename tc >
CImg< T > & draw_ellipse (
    const int x0,
    const int y0,
    const float r1,
    const float r2,
    const float angle,
    const tc *const color,
    const float opacity = 1 )

```

Draw a filled 2D ellipse.

Parameters

<i>x0</i>	X-coordinate of the ellipse center.
-----------	-------------------------------------

Parameters

<i>y0</i>	Y-coordinate of the ellipse center.
<i>r1</i>	First radius of the ellipse.
<i>r2</i>	Second radius of the ellipse.
<i>angle</i>	Angle of the first radius.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.

8.1.4.457 draw_ellipse() [2/4]

```

template<typename T >
template<typename t , typename tc >
CImg< T > & draw_ellipse (
    const int x0,
    const int y0,
    const CImg< t > & tensor,
    const tc *const color,
    const float opacity = 1 )

```

Parameters

<i>x0</i>	X-coordinate of the ellipse center.
<i>y0</i>	Y-coordinate of the ellipse center.
<i>tensor</i>	Diffusion tensor describing the ellipse.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.

8.1.4.458 draw_ellipse() [3/4]

```

template<typename T >
template<typename tc >
CImg< T > & draw_ellipse (
    const int x0,
    const int y0,
    const float r1,
    const float r2,
    const float angle,
    const tc *const color,
    const float opacity,
    const unsigned int pattern )

```

Draw an outlined 2D ellipse.

Parameters

<i>x0</i>	X-coordinate of the ellipse center.
<i>y0</i>	Y-coordinate of the ellipse center.
<i>r1</i>	First radius of the ellipse.
<i>r2</i>	Second radius of the ellipse.

Parameters

<i>angle</i>	Angle of the first radius.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the outline stipple pattern.

8.1.4.459 draw_ellipse() [4/4]

```
template<typename T >
template<typename t , typename tc >
CImg< T > & draw_ellipse (
    const int x0,
    const int y0,
    const CImg< t > & tensor,
    const tc *const color,
    const float opacity,
    const unsigned int pattern )
```

Parameters

<i>x0</i>	X-coordinate of the ellipse center.
<i>y0</i>	Y-coordinate of the ellipse center.
<i>tensor</i>	Diffusion tensor describing the ellipse.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the outline stipple pattern.

8.1.4.460 draw_circle() [1/2]

```
template<typename T >
template<typename tc >
CImg< T > & draw_circle (
    const int x0,
    const int y0,
    int radius,
    const tc *const color,
    const float opacity = 1 )
```

Draw a filled 2D circle.

Parameters

<i>x0</i>	X-coordinate of the circle center.
<i>y0</i>	Y-coordinate of the circle center.
<i>radius</i>	Circle radius.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.

Note

- Circle version of the Bresenham's algorithm is used.

8.1.4.461 draw_circle() [2/2]

```
template<typename T >
template<typename tc >
CImg< T > & draw_circle (
    const int x0,
    const int y0,
    int radius,
    const tc *const color,
    const float opacity,
    const unsigned int pattern )
```

Draw an outlined 2D circle.

Parameters

<i>x0</i>	X-coordinate of the circle center.
<i>y0</i>	Y-coordinate of the circle center.
<i>radius</i>	Circle radius.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the outline stipple pattern.

8.1.4.462 draw_image() [1/2]

```
template<typename T >
template<typename t >
CImg< T > & draw_image (
    const int x0,
    const int y0,
    const int z0,
    const int c0,
    const CImg< t > & sprite,
    const float opacity = 1 )
```

Draw an image.

Parameters

<i>sprite</i>	Sprite image.
<i>x0</i>	X-coordinate of the sprite position.
<i>y0</i>	Y-coordinate of the sprite position.
<i>z0</i>	Z-coordinate of the sprite position.
<i>c0</i>	C-coordinate of the sprite position.
<i>opacity</i>	Drawing opacity.

8.1.4.463 draw_image() [2/2]

```

template<typename T >
template<typename ti , typename tm >
CImg< T > & draw_image (
    const int x0,
    const int y0,
    const int z0,
    const int c0,
    const CImg< ti > & sprite,
    const CImg< tm > & mask,
    const float opacity = 1,
    const float mask_max_value = 1 )

```

Draw a masked image.

Parameters

<i>sprite</i>	Sprite image.
<i>mask</i>	Mask image.
<i>x0</i>	X-coordinate of the sprite position in the image instance.
<i>y0</i>	Y-coordinate of the sprite position in the image instance.
<i>z0</i>	Z-coordinate of the sprite position in the image instance.
<i>c0</i>	C-coordinate of the sprite position in the image instance.
<i>mask_max_value</i>	Maximum pixel value of the mask image <i>mask</i> .
<i>opacity</i>	Drawing opacity.

Note

- Pixel values of *mask* set the opacity of the corresponding pixels in *sprite*.
- Dimensions along x,y and z of *sprite* and *mask* must be the same.

8.1.4.464 draw_text() [1/4]

```

template<typename T >
template<typename tc1 , typename tc2 , typename t >
CImg< T > & draw_text (
    const int x0,
    const int y0,
    const char *const text,
    const tc1 *const foreground_color,
    const tc2 *const background_color,
    const float opacity,
    const CImgList< t > *const font,
    ... )

```

Draw a text string.

Parameters

<i>x0</i>	X-coordinate of the text in the image instance.
<i>y0</i>	Y-coordinate of the text in the image instance.

Parameters

<i>text</i>	Format of the text ('printf'-style format string).
<i>foreground_color</i>	Pointer to spectrum() consecutive values, defining the foreground drawing color.
<i>background_color</i>	Pointer to spectrum() consecutive values, defining the background drawing color.
<i>opacity</i>	Drawing opacity.
<i>font</i>	Font used for drawing text.

8.1.4.465 draw_text() [2/4]

```

template<typename T >
template<typename tc , typename t >
CImg< T > & draw_text (
    const int x0,
    const int y0,
    const char *const text,
    const tc *const foreground_color,
    const int ,
    const float opacity,
    const CImgList< t > *const font,
    ... )

```

Note

A transparent background is used for the text.

8.1.4.466 draw_text() [3/4]

```

template<typename T >
template<typename tc , typename t >
CImg< T > & draw_text (
    const int x0,
    const int y0,
    const char *const text,
    const int ,
    const tc *const background_color,
    const float opacity,
    const CImgList< t > *const font,
    ... )

```

Note

A transparent foreground is used for the text.

8.1.4.467 draw_text() [4/4]

```

template<typename T >
template<typename tc1 , typename tc2 >
CImg< T > & draw_text (
    const int x0,
    const int y0,
    const char *const text,
    const tc1 *const foreground_color,
    const tc2 *const background_color,
    const float opacity = 1,
    const unsigned int font_height = 13,
    ... )

```

Parameters

<i>x0</i>	X-coordinate of the text in the image instance.
<i>y0</i>	Y-coordinate of the text in the image instance.
<i>text</i>	Format of the text ('printf'-style format string).
<i>foreground_color</i>	Array of spectrum() values of type T, defining the foreground color (0 means 'transparent').
<i>background_color</i>	Array of spectrum() values of type T, defining the background color (0 means 'transparent').
<i>opacity</i>	Drawing opacity.
<i>font_height</i>	Height of the text font (exact match for 13,32,64,128, interpolated otherwise).

Note

To ensure thread-safety, this function uses a mutex lock. For real multi-threaded drawing of text, use another version of `CImg<T>::draw_text()` with argument `font`, that must be a copy of what is returned by `CImgList<T>::font()`.

8.1.4.468 draw_quiver() [1/2]

```

template<typename T >
template<typename t1 , typename t2 >
CImg< T > & draw_quiver (
    const CImg< t1 > & flow,
    const t2 *const color,
    const float opacity = 1,
    const unsigned int sampling = 25,
    const float factor = -20,
    const bool is_arrow = true,
    const unsigned int pattern = ~0U )

```

Draw a 2D vector field.

Parameters

<i>flow</i>	Image of 2D vectors used as input data.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>sampling</i>	Length (in pixels) between each arrow.
<i>factor</i>	Length factor of each arrow (if <0, computed as a percentage of the maximum length).
<i>is_arrow</i>	Indicates whether arrows must be drawn, instead of oriented segments.
<i>pattern</i>	Used pattern to draw lines.

Note

Clipping is supported.

8.1.4.469 draw_quiver() [2/2]

```
template<typename T >
template<typename t1 , typename t2 >
CImg< T > & draw_quiver (
    const CImg< t1 > & flow,
    const CImg< t2 > & color,
    const float opacity = 1,
    const unsigned int sampling = 25,
    const float factor = -20,
    const bool is_arrow = true,
    const unsigned int pattern = ~0U )
```

Draw a 2D vector field using a field of colors.

Parameters

<i>flow</i>	Image of 2D vectors used as input data.
<i>color</i>	Image of spectrum() -D vectors corresponding to the color of each arrow.
<i>opacity</i>	Opacity of the drawing.
<i>sampling</i>	Length (in pixels) between each arrow.
<i>factor</i>	Length factor of each arrow (if <0, computed as a percentage of the maximum length).
<i>is_arrow</i>	Indicates whether arrows must be drawn, instead of oriented segments.
<i>pattern</i>	Used pattern to draw lines.

Note

Clipping is supported.

8.1.4.470 draw_axis() [1/2]

```
template<typename T >
template<typename t , typename tc >
CImg< T > & draw_axis (
    const CImg< t > & values_x,
    const int y,
    const tc *const color,
    const float opacity = 1,
    const unsigned int pattern = ~0U,
    const unsigned int font_height = 13,
    const bool allow_zero = true,
    const float round_x = 0 )
```

Draw a labeled horizontal axis.

Parameters

<i>values_x</i>	Values along the horizontal axis.
<i>y</i>	Y-coordinate of the horizontal axis in the image instance.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	Drawing pattern.
<i>font_height</i>	Height of the labels (exact match for 13,23,53,103, interpolated otherwise).
<i>allow_zero</i>	Enable/disable the drawing of label '0' if found.
<i>round_x</i>	

8.1.4.471 draw_axis() [2/2]

```
template<typename T >
template<typename t , typename tc >
CImg< T > & draw_axis (
    const int x,
    const CImg< t > & values_y,
    const tc *const color,
    const float opacity = 1,
    const unsigned int pattern = ~0U,
    const unsigned int font_height = 13,
    const bool allow_zero = true,
    const float round_y = 0 )
```

Draw a labeled vertical axis.

Parameters

<i>x</i>	X-coordinate of the vertical axis in the image instance.
<i>values_y</i>	Values along the Y-axis.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	Drawing pattern.
<i>font_height</i>	Height of the labels (exact match for 13,23,53,103, interpolated otherwise).
<i>allow_zero</i>	Enable/disable the drawing of label '0' if found.
<i>round_y</i>	

8.1.4.472 draw_axes()

```
template<typename T >
template<typename tx , typename ty , typename tc >
CImg< T > & draw_axes (
    const CImg< tx > & values_x,
    const CImg< ty > & values_y,
    const tc *const color,
    const float opacity = 1,
    const unsigned int pattern_x = ~0U,
    const unsigned int pattern_y = ~0U,
```

```

const unsigned int font_height = 13,
const bool allow_zero = true,
const float round_x = 0,
const float round_y = 0 )

```

Draw labeled horizontal and vertical axes.

Parameters

<i>values_x</i>	Values along the X-axis.
<i>values_y</i>	Values along the Y-axis.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>pattern_x</i>	Drawing pattern for the X-axis.
<i>pattern_y</i>	Drawing pattern for the Y-axis.
<i>font_height</i>	Height of the labels (exact match for 13,23,53,103, interpolated otherwise).
<i>allow_zero</i>	Enable/disable the drawing of label '0' if found.
<i>round_x</i>	
<i>round_y</i>	

8.1.4.473 draw_grid()

```

template<typename T >
template<typename tx , typename ty , typename tc >
CImg< T > & draw_grid (
    const CImg< tx > & values_x,
    const CImg< ty > & values_y,
    const tc *const color,
    const float opacity = 1,
    const unsigned int pattern_x = ~0U,
    const unsigned int pattern_y = ~0U )

```

Draw a 2D grid.

Parameters

<i>values↔ _x</i>	X-coordinates of the vertical lines.
<i>values↔ _y</i>	Y-coordinates of the horizontal lines.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>pattern↔ _x</i>	Drawing pattern for vertical lines.
<i>pattern↔ _y</i>	Drawing pattern for horizontal lines.

8.1.4.474 draw_graph()

```

template<typename T >

```

```
template<typename t , typename tc >
CImg< T > & draw_graph (
    const CImg< t > & data,
    const tc *const color,
    const float opacity = 1,
    const unsigned int plot_type = 1,
    const int vertex_type = 1,
    const double ymin = 0,
    const double ymax = 0,
    const unsigned int pattern = ~0U )
```

Draw a 1D graph.

Parameters

<i>data</i>	Image containing the graph values $I = f(x)$.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>plot_type</i>	Define the plot type: • 0 = No plot. • 1 = Plot using segments. • 2 = Plot using cubic splines. • 3 = Plot with bars.
<i>vertex_type</i>	Define the type of points: • 0 = No points. • 1 = Point. • 2 = Straight cross. • 3 = Diagonal cross. • 4 = Filled circle. • 5 = Outlined circle. • 6 = Square. • 7 = Diamond.
<i>ymin</i>	Lower bound of the y-range.
<i>ymax</i>	Upper bound of the y-range.
<i>pattern</i>	Drawing pattern.

Note

- if `ymin==ymax==0`, the y-range is computed automatically from the input samples.

8.1.4.475 draw_fill()

```
template<typename T >
template<typename tc , typename t >
```

```

CImg< T > & draw_fill (
    const int x0,
    const int y0,
    const int z0,
    const tc *const color,
    const float opacity,
    CImg< t > & region,
    const float tolerance = 0,
    const bool is_high_connectivity = false )

```

Fill a 3D region using the flood-fill algorithm.

Parameters

	<i>x0</i>	X-coordinate of the starting point of the region to fill.
	<i>y0</i>	Y-coordinate of the starting point of the region to fill.
	<i>z0</i>	Z-coordinate of the starting point of the region to fill.
	<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
out	<i>region</i>	Image that will contain the mask of the filled region [output].
	<i>tolerance</i>	Tolerance concerning neighborhood values.
	<i>opacity</i>	Opacity of the drawing.
	<i>is_high_connectivity</i>	Indicates whether the algorithm uses low or high connectivity.

Returns

region is initialized with the binary mask of the filled region.

8.1.4.476 draw_plasma()

```

template<typename T >
CImg< T > & draw_plasma (
    const float alpha = 1,
    const float beta = 0,
    const unsigned int scale = 8 )

```

Draw a random plasma texture.

Parameters

<i>alpha</i>	Alpha-parameter.
<i>beta</i>	Beta-parameter.
<i>scale</i>	Scale-parameter.

Note

Use the mid-point algorithm to render.

8.1.4.477 draw_mandelbrot()

```

template<typename T >
template<typename tc >

```



```

CImg< T > & draw_mandelbrot (
    const int x0,
    const int y0,
    const int x1,
    const int y1,
    const CImg< tc > & colormap,
    const float opacity = 1,
    const double z0r = -2,
    const double z0i = -2,
    const double z1r = 2,
    const double z1i = 2,
    const unsigned int iteration_max = 255,
    const bool is_normalized_iteration = false,
    const bool is_julia_set = false,
    const double param_r = 0,
    const double param_i = 0 )

```

Draw a quadratic Mandelbrot or Julia 2D fractal.

Parameters

<i>x0</i>	X-coordinate of the upper-left pixel.
<i>y0</i>	Y-coordinate of the upper-left pixel.
<i>x1</i>	X-coordinate of the lower-right pixel.
<i>y1</i>	Y-coordinate of the lower-right pixel.
<i>colormap</i>	Colormap.
<i>opacity</i>	Drawing opacity.
<i>z0r</i>	Real part of the upper-left fractal vertex.
<i>z0i</i>	Imaginary part of the upper-left fractal vertex.
<i>z1r</i>	Real part of the lower-right fractal vertex.
<i>z1i</i>	Imaginary part of the lower-right fractal vertex.
<i>iteration_max</i>	Maximum number of iterations for each estimated point.
<i>is_normalized_iteration</i>	Indicates whether iterations are normalized.
<i>is_julia_set</i>	Indicates whether the Mandelbrot or Julia set is rendered.
<i>param_r</i>	Real part of the Julia set parameter.
<i>param_i</i>	Imaginary part of the Julia set parameter.

Note

Fractal rendering is done by the Escape Time Algorithm.

8.1.4.478 draw_gaussian() [1/2]

```

template<typename T >
template<typename tc >
CImg< T > & draw_gaussian (
    const float xc,
    const float sigma,
    const tc *const color,
    const float opacity = 1 )

```

Draw a 1D gaussian function.

Parameters

<i>xc</i>	X-coordinate of the gaussian center.
<i>sigma</i>	Standard variation of the gaussian distribution.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.

8.1.4.479 draw_gaussian() [2/2]

```
template<typename T >
template<typename t , typename tc >
CImg< T > & draw_gaussian (
    const float xc,
    const float yc,
    const CImg< t > & tensor,
    const tc *const color,
    const float opacity = 1 )
```

Draw a 2D gaussian function.

Parameters

<i>xc</i>	X-coordinate of the gaussian center.
<i>yc</i>	Y-coordinate of the gaussian center.
<i>tensor</i>	Covariance matrix (must be 2x2).
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.

8.1.4.480 draw_object3d()

```
template<typename T >
template<typename tp , typename tf , typename tc , typename to >
CImg< T > & draw_object3d (
    const float x0,
    const float y0,
    const float z0,
    const CImg< tp > & vertices,
    const CImgList< tf > & primitives,
    const CImgList< tc > & colors,
    const CImg< to > & opacities,
    const unsigned int render_type = 4,
    const bool is_double_sided = false,
    const float focal = 700,
    const float lightx = 0,
    const float lighty = 0,
    const float lightz = -5e8,
    const float specular_lightness = 0.25f,
    const float specular_shininess = 0.1f,
    const float g_opacity = 1,
    const bool is_multithreaded_rendering = false )
```

Draw a 3D object.

Parameters

<i>x0</i>	X-coordinate of the 3D object position
<i>y0</i>	Y-coordinate of the 3D object position
<i>z0</i>	Z-coordinate of the 3D object position
<i>vertices</i>	Image Nx3 describing 3D point coordinates
<i>primitives</i>	List of P primitives
<i>colors</i>	List of P color (or textures)
<i>opacities</i>	Image or list of P opacities
<i>render_type</i>	Render type (0=Points, 1=Lines, 2=Faces (no light), 3=Faces (flat), 4=Faces(Gouraud)
<i>is_double_sided</i>	Indicates whether object faces have two sides or are oriented.
<i>focal</i>	Focal length (0 for parallel projection).
<i>lightx</i>	X-coordinate of the light
<i>lighty</i>	Y-coordinate of the light
<i>lightz</i>	Z-coordinate of the light
<i>specular_lightness</i>	Amount of specular light.
<i>specular_shininess</i>	Shininess of the object
<i>g_opacity</i>	Global opacity of the object.
<i>is_multithreaded_rendering</i>	Indicates whether mesh rendering is done with multiple threads

8.1.4.481 select()

```
template<typename T >
CImg< T > & select (
    CImgDisplay & disp,
    const unsigned int feature_type = 2,
    unsigned int *const XYZ = 0,
    const bool exit_on_anykey = false,
    const bool is_deep_selection_default = false )
```

Launches a simple interface to select a shape from an image.

Parameters

<i>disp</i>	Display window to use.
<i>feature_type</i>	Type of feature to select. Can be { 0=point 1=line 2=rectangle 3=ellipse }.
<i>XYZ</i>	Pointer to 3 values X,Y,Z which indicate the projection point coordinates, for volumetric images.
<i>exit_on_anykey</i>	Exit function when any key is pressed.
<i>is_deep_selection_default</i>	

8.1.4.482 load()

```
template<typename T >
CImg< T > & load (
    const char *const filename )
```

Load image from a file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

Note

The extension of *filename* defines the file format. If no filename extension is provided, [CImg<T>::get_load\(\)](#) will try to load the file as a .cimg or .cimgz file.

8.1.4.483 load_ascii()

```
template<typename T >
CImg< T > & load_ascii (
    const char *const filename )
```

Load image from an ASCII file.

Parameters

<i>filename</i>	Filename, as a C -string.
-----------------	---------------------------

8.1.4.484 load_dlm()

```
template<typename T >
CImg< T > & load_dlm (
    const char *const filename )
```

Load image from a DLM file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.485 load_bmp()

```
template<typename T >
CImg< T > & load_bmp (
    const char *const filename )
```

Load image from a BMP file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.486 load_jpeg()

```
template<typename T >
CImg< T > & load_jpeg (
    const char *const filename )
```

Load image from a JPEG file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.487 load_jxl()

```
template<typename T >
CImg< T > & load_jxl (
    const char *const filename )
```

Load image from a JPEG XL file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.488 load_magick()

```
template<typename T >
CImg< T > & load_magick (
    const char *const filename )
```

Load image from a file, using Magick++ library.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.489 load_png()

```
template<typename T >
CImg< T > & load_png (
    const char *const filename,
    unsigned int *const bits_per_value = 0 )
```

Load an image from a PNG file.

This function reads a PNG file and loads its content into the current CImg<T> instance.

Parameters

	<i>filename</i>	Path to the PNG file to load, as a C-string.
out	<i>bits_per_value</i>	Number of bits used to store a scalar value in the image file.

Returns

Reference to the current image instance, now containing the loaded image.

```
// Example usage:  
CImg<unsigned char> img;  
img.load_png("image.png");  
img.display();
```

Warning

Throws a CImgIOException if the file does not exist or is not a valid PNG.

8.1.4.490 load_pnm()

```
template<typename T >  
CImg< T > & load_pnm (  
    const char *const filename )
```

Load image from a PNM file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.491 load_pfm()

```
template<typename T >  
CImg< T > & load_pfm (  
    const char *const filename )
```

Load image from a PFM file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.492 load_rgb()

```
template<typename T >  
CImg< T > & load_rgb (  
    const char *const filename,  
    const unsigned int dimw,  
    const unsigned int dimh = 1 )
```

Load image from a RGB file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>dimw</i>	Width of the image buffer.
<i>dimh</i>	Height of the image buffer.

8.1.4.493 load_rgba()

```
template<typename T >
CImg< T > & load_rgba (
    const char *const filename,
    const unsigned int dimw,
    const unsigned int dimh = 1 )
```

Load image from a RGBA file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>dimw</i>	Width of the image buffer.
<i>dimh</i>	Height of the image buffer.

8.1.4.494 load_tiff()

```
template<typename T >
CImg< T > & load_tiff (
    const char *const filename,
    const unsigned int first_frame = 0,
    const unsigned int last_frame = ~0U,
    const unsigned int step_frame = 1,
    unsigned int *const bits_per_value = 0,
    float *const voxel_size = 0,
    CImg< charT > *const description = 0 )
```

Load image from a TIFF file.

Parameters

	<i>filename</i>	Filename, as a C-string.
	<i>first_frame</i>	First frame to read (for multi-pages tiff).
	<i>last_frame</i>	Last frame to read (for multi-pages tiff).
	<i>step_frame</i>	Step value of frame reading.
out	<i>bits_per_value</i>	Number of bits used to store a scalar value in the image file.
out	<i>voxel_size</i>	Voxel size, as stored in the filename.
out	<i>description</i>	Description, as stored in the filename.

Note

- libtiff support is enabled by defining the precompilation directive `cimg_use_tiff`.
- When libtiff is enabled, 2D and 3D (multipage) several channels per pixel are supported for `char`, `uchar`, `short`, `ushort`, `float` and `double` pixel types.
- If `cimg_use_tiff` is not defined at compile time the function uses `CImg<T>` & [load_other\(const char*\)](#).

8.1.4.495 load_minc2()

```
template<typename T >
CImg< T > & load_minc2 (
    const char *const filename )
```

Load image from a MINC2 file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.496 load_analyze()

```
template<typename T >
CImg< T > & load_analyze (
    const char *const filename,
    float *const voxel_size = 0 )
```

Load image from an ANALYZE7.5/NIFTI file.

Parameters

	<i>filename</i>	Filename, as a C-string.
out	<i>voxel_size</i>	Pointer to the three voxel sizes read from the file.

8.1.4.497 load_cimg() [1/2]

```
template<typename T >
CImg< T > & load_cimg (
    const char *const filename,
    const char axis = 'z',
    const float align = 0 )
```

Load image from a .cimg[z] file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>axis</i>	Concatenation axis, if file contains multiple images. Can be { 'x' 'y' 'z' 'c' }.
<i>align</i>	Concatenation alignment.

8.1.4.498 load_cimg() [2/2]

```
template<typename T >
CImg< T > & load_cimg (
    const char *const filename,
    const unsigned int n0,
    const unsigned int n1,
    const unsigned int x0,
    const unsigned int y0,
    const unsigned int z0,
    const unsigned int c0,
    const unsigned int x1,
    const unsigned int y1,
    const unsigned int z1,
    const unsigned int c1,
    const char axis = 'z',
    const float align = 0 )
```

Load sub-images of a .cimg file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>n0</i>	Starting frame.
<i>n1</i>	Ending frame (~0U for max).
<i>x0</i>	X-coordinate of the starting sub-image vertex.
<i>y0</i>	Y-coordinate of the starting sub-image vertex.
<i>z0</i>	Z-coordinate of the starting sub-image vertex.
<i>c0</i>	C-coordinate of the starting sub-image vertex.
<i>x1</i>	X-coordinate of the ending sub-image vertex (~0U for max).
<i>y1</i>	Y-coordinate of the ending sub-image vertex (~0U for max).
<i>z1</i>	Z-coordinate of the ending sub-image vertex (~0U for max).
<i>c1</i>	C-coordinate of the ending sub-image vertex (~0U for max).
<i>axis</i>	Concatenation axis, if file contains multiple images. Can be { 'x' 'y' 'z' 'c' }.
<i>align</i>	Concatenation alignment.

8.1.4.499 load_inr()

```
template<typename T >
CImg< T > & load_inr (
    const char *const filename,
    float *const voxel_size = 0 )
```

Load image from an INRIMAGE-4 file.

Parameters

	<i>filename</i>	Filename, as a C-string.
out	<i>voxel_size</i>	Pointer to the three voxel sizes read from the file.

8.1.4.500 load_exr()

```
template<typename T >
CImg< T > & load_exr (
    const char *const filename )
```

Load image from a EXR file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.501 load_pandore()

```
template<typename T >
CImg< T > & load_pandore (
    const char *const filename )
```

Load image from a PANDORE-5 file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.502 load_parrec()

```
template<typename T >
CImg< T > & load_parrec (
    const char *const filename,
    const char axis = 'c',
    const float align = 0 )
```

Load image from a PAR-REC (Philips) file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>axis</i>	Concatenation axis, if file contains multiple images. Can be { 'x' 'y' 'z' 'c' }.
<i>align</i>	Concatenation alignment.

8.1.4.503 load_raw()

```
template<typename T >
CImg< T > & load_raw (
    const char *const filename,
    const unsigned int size_x = 0,
    const unsigned int size_y = 1,
```

```

const unsigned int size_z = 1,
const unsigned int size_c = 1,
const bool is_multiplexed = false,
const bool invert_endianness = false,
const ulongT offset = 0 )

```

Load image from a raw binary file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>size_x</i>	Width of the image buffer.
<i>size_y</i>	Height of the image buffer.
<i>size_z</i>	Depth of the image buffer.
<i>size_c</i>	Spectrum of the image buffer.
<i>is_multiplexed</i>	Indicates whether the image values are multiplexed along the C-axis.
<i>invert_endianness</i>	Indicates whether the endianness of the image buffer must be inverted.
<i>offset</i>	Starting offset of the read in the specified file.

8.1.4.504 load_yuv()

```

template<typename T >
CImg< T > & load_yuv (
    const char *const filename,
    const unsigned int size_x,
    const unsigned int size_y = 1,
    const unsigned int chroma_subsampling = 444,
    const unsigned int first_frame = 0,
    const unsigned int last_frame = ~0U,
    const unsigned int step_frame = 1,
    const bool yuv2rgb = true,
    const char axis = 'z' )

```

Load image sequence from a YUV file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>size_x</i>	Width of the frames.
<i>size_y</i>	Height of the frames.
<i>chroma_subsampling</i>	Type of chroma subsampling. Can be { 420 422 444 }.
<i>first_frame</i>	Index of the first frame to read.
<i>last_frame</i>	Index of the last frame to read.
<i>step_frame</i>	Step value for frame reading.
<i>yuv2rgb</i>	Indicates whether the YUV to RGB transform must be applied.
<i>axis</i>	Concatenation axis, if file contains multiple images. Can be { 'x' 'y' 'z' 'c' }.

8.1.4.505 load_off()

```
template<typename T >
template<typename tf , typename tc >
CImg< T > & load_off (
    CImgList< tf > & primitives,
    CImgList< tc > & colors,
    const char *const filename )
```

Load 3D object from a .OFF file.

Parameters

out	<i>primitives</i>	Primitives data of the 3D object.
out	<i>colors</i>	Colors data of the 3D object.
	<i>filename</i>	Filename, as a C-string.

8.1.4.506 load_video()

```
template<typename T >
CImg< T > & load_video (
    const char *const filename,
    const unsigned int first_frame = 0,
    const unsigned int last_frame = ~0U,
    const unsigned int step_frame = 1,
    const char axis = 'z',
    const float align = 0 )
```

Load image sequence from a video file, using OpenCV library.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>first_frame</i>	Index of the first frame to read.
<i>last_frame</i>	Index of the last frame to read.
<i>step_frame</i>	Step value for frame reading.
<i>axis</i>	Alignment axis.
<i>align</i>	Concatenation alignment.

8.1.4.507 load_ffmpeg_external()

```
template<typename T >
CImg< T > & load_ffmpeg_external (
    const char *const filename,
    const char axis = 'z',
    const float align = 0 )
```

Load image sequence using FFMPEG's external tool 'ffmpeg'.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>axis</i>	Concatenation axis, if file contains multiple images. Can be { 'x' 'y' 'z' 'c' }.
<i>align</i>	Concatenation alignment.

8.1.4.508 load_gif_external()

```
template<typename T >
CImg< T > & load_gif_external (
    const char *const filename,
    const char axis = 'z',
    const float align = 0 )
```

Load gif file, using Imagemagick or GraphicsMagicks's external tools.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>axis</i>	Concatenation axis, if file contains multiple images. Can be { 'x' 'y' 'z' 'c' }.
<i>align</i>	Concatenation alignment.

8.1.4.509 load_heif()

```
template<typename T >
CImg< T > & load_heif (
    const char *const filename )
```

Load image from a HEIC file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.510 load_webp()

```
template<typename T >
CImg< T > & load_webp (
    const char *const filename )
```

Load image from a WebP file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.511 load_graphicsmagick_external()

```
template<typename T >
CImg< T > & load_graphicsmagick_external (
    const char *const filename )
```

Load image using GraphicsMagick's external tool 'gm'.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.512 load_gzip_external()

```
template<typename T >
CImg< T > & load_gzip_external (
    const char *const filename )
```

Load gzipped image file, using external tool 'gunzip'.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.513 load_imagemagick_external()

```
template<typename T >
CImg< T > & load_imagemagick_external (
    const char *const filename )
```

Load image using ImageMagick's external tool 'convert'.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.514 load_medcon_external()

```
template<typename T >
CImg< T > & load_medcon_external (
    const char *const filename )
```

Load image from a DICOM file, using Medcon's external tool 'medcon'.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.515 load_pdf_external()

```
template<typename T >
CImg< T > & load_pdf_external (
    const char *const filename,
    const unsigned int resolution = 400 )
```

Load image from a .pdf file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>resolution</i>	Image resolution.

8.1.4.516 load_dcraw_external()

```
template<typename T >
CImg< T > & load_dcraw_external (
    const char *const filename )
```

Load image from a RAW Color Camera file, using external tool 'dcraw'.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.517 load_camera()

```
template<typename T >
CImg< T > & load_camera (
    const unsigned int camera_index = 0,
    const unsigned int capture_width = 0,
    const unsigned int capture_height = 0,
    const unsigned int skip_frames = 0,
    const bool release_camera = true )
```

Captures an image from a connected camera device (requires OpenCV).

This function allows you to directly acquire an image from a camera device (e.g., webcam) connected to the system. The captured image is stored in the current CImg<T> instance.

Parameters

<i>camera_index</i>	Index of the camera to capture images from (from 0 to 63).
<i>capture_width</i>	Width of the desired image ('0' stands for default value).
<i>capture_height</i>	Height of the desired image ('0' stands for default value).
<i>skip_frames</i>	Number of frames to skip before the capture.
<i>release_camera</i>	Indicates whether the camera resource must be released at the end of the function.

Returns

Reference to the current image instance, now containing the captured camera frame.

```
// Example usage:
CImg<unsigned char> img;
img.load_camera(); // Capture an image from the default camera
img.display();
```

Note

Requires your code to be linked with the OpenCV library (macro `cimg_use_opencv` enabled).

Warning

Throws a `CImgIOException` if no camera is found or if the capture fails.

8.1.4.518 load_other()

```
template<typename T >
CImg< T > & load_other (
    const char *const filename )
```

Load image using various non-native ways.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.519 print()

```
template<typename T >
const CImg< T > & print (
    const char *const title = 0,
    const bool display_stats = true ) const
```

Display information about the image data.

Parameters

<i>title</i>	Name for the considered image.
<i>display_stats</i>	Specifies whether to compute and display image statistics.

8.1.4.520 display() [1/3]

```
template<typename T >
const CImg< T > & display (
    CImgDisplay & disp ) const
```

Display image into a `CImgDisplay` window.

Parameters

<i>disp</i>	Display window.
-------------	-----------------

8.1.4.521 display() [2/3]

```
template<typename T >
const CImg< T > & display (
    CImgDisplay & disp,
    const bool display_info,
    unsigned int *const XYZ = 0,
    const bool exit_on_anykey = false ) const
```

Display image into a [CImgDisplay](#) window, in an interactive way.

Parameters

	<i>disp</i>	Display window.
	<i>display_info</i>	Indicates whether image information is displayed on the standard output.
in, out	<i>XYZ</i>	Contains the XYZ coordinates at start / exit of the function.
	<i>exit_on_anykey</i>	Exit function when any key is pressed.

8.1.4.522 display() [3/3]

```
template<typename T >
const CImg< T > & display (
    const char *const title = 0,
    const bool display_info = true,
    unsigned int *const XYZ = 0,
    const bool exit_on_anykey = false ) const
```

Display image into an interactive window.

Parameters

	<i>title</i>	Window title
	<i>display_info</i>	Indicates whether image information is displayed on the standard output.
in, out	<i>XYZ</i>	Contains the XYZ coordinates at start / exit of the function.
	<i>exit_on_anykey</i>	Exit function when any key is pressed.

8.1.4.523 display_object3d()

```
template<typename T >
template<typename tp , typename tf , typename tc , typename to >
const CImg< T > & display_object3d (
    CImgDisplay & disp,
    const CImg< tp > & vertices,
```

```

const CImgList< tf > & primitives,
const CImgList< tc > & colors,
const to & opacities,
const bool centering = true,
const int render_static = 4,
const int render_motion = 1,
const bool is_double_sided = false,
const float focal = 700,
const float light_x = 0,
const float light_y = 0,
const float light_z = -5e8f,
const float specular_lightness = 0.25f,
const float specular_shininess = 0.1f,
const bool display_axes = true,
float *const pose_matrix = 0,
const bool exit_on_anykey = false ) const

```

Display a 3D object in an interactive window.

Parameters

<i>disp</i>	Display window.
<i>vertices</i>	Vertices data of the 3D object.
<i>primitives</i>	Primitives data of the 3D object.
<i>colors</i>	Colors data of the 3D object.
<i>opacities</i>	Opacities data of the 3D object.
<i>centering</i>	Indicates whether the 3D object should be centered for display.
<i>render_static</i>	Rendering mode.
<i>render_motion</i>	Rendering mode when the 3D object is in motion.
<i>is_double_sided</i>	Indicates whether the object primitives are double-sided.
<i>focal</i>	Focal length (0 for parallel projection).
<i>light_x</i>	X-coordinate of the light source.
<i>light_y</i>	Y-coordinate of the light source.
<i>light_z</i>	Z-coordinate of the light source.
<i>specular_lightness</i>	Amount of specular light.
<i>specular_shininess</i>	Shininess of the object material.
<i>display_axes</i>	Indicates whether the 3D axes are displayed.
<i>pose_matrix</i>	Pointer to 12 values defining a 3D pose (as a 4x3 matrix).
<i>exit_on_anykey</i>	Exit function when any key is pressed.

8.1.4.524 display_graph()

```

template<typename T >
const CImg< T > & display_graph (
    CImgDisplay & disp,
    const unsigned int plot_type = 1,
    const unsigned int vertex_type = 1,
    const char *const labelx = 0,
    const double xmin = 0,
    const double xmax = 0,
    const char *const labely = 0,
    const double ymin = 0,

```

```
const double ymax = 0,
const bool exit_on_anykey = false ) const
```

Display 1D graph in an interactive window.

Parameters

<i>disp</i>	Display window.
<i>plot_type</i>	Plot type. Can be { 0=points 1=segments 2=splines 3=bars }.
<i>vertex_type</i>	Vertex type.
<i>labelx</i>	Title for the horizontal axis, as a C-string.
<i>xmin</i>	Minimum value along the X-axis.
<i>xmax</i>	Maximum value along the X-axis.
<i>labely</i>	Title for the vertical axis, as a C-string.
<i>ymin</i>	Minimum value along the Y-axis.
<i>ymax</i>	Maximum value along the Y-axis.
<i>exit_on_anykey</i>	Exit function when any key is pressed.

8.1.4.525 save()

```
template<typename T >
const CImg< T > & save (
    const char *const filename,
    const int number = -1,
    const unsigned int digits = 6 ) const
```

Save the image as a file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>number</i>	When positive, represents an index added to the filename. Otherwise, no number is added.
<i>digits</i>	Number of digits used for adding the number to the filename.

Note

- The used file format is defined by the file extension in the filename *filename*.
- Parameter *number* can be used to add a 6-digit number to the filename before saving.

8.1.4.526 save_ascii()

```
template<typename T >
const CImg< T > & save_ascii (
    const char *const filename ) const
```

Save the image as an ASCII file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.527 save_cpp()

```
template<typename T >
const CImg< T > & save_cpp (
    const char *const filename ) const
```

Save the image as a .cpp source file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.528 save_dlm()

```
template<typename T >
const CImg< T > & save_dlm (
    const char *const filename ) const
```

Save the image as a DLM file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.529 save_bmp()

```
template<typename T >
const CImg< T > & save_bmp (
    const char *const filename ) const
```

Save the image as a BMP file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.530 save_webp()

```
template<typename T >
const CImg< T > & save_webp (
```

```
const char *const filename,
const int quality = 100 ) const
```

Save the image as a WebP file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>quality</i>	Image quality (in %)

8.1.4.531 save_jpeg()

```
template<typename T >
const CImg< T > & save_jpeg (
    const char *const filename,
    const unsigned int quality = 100 ) const
```

Save the image as a JPEG file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>quality</i>	Image quality (in %)

8.1.4.532 save_jxl()

```
template<typename T >
const CImg< T > & save_jxl (
    const char *const filename,
    const float distance = 1.0f,
    const unsigned int bytes_per_pixel = 0 ) const
```

Save the image as a JPEG XL file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>distance</i>	Sets the level for lossy compression: lower = higher quality. Range: 0 .. 25. 0.0 = mathematically lossless
<i>bytes_per_pixel</i>	Force the number of bytes per pixels for the saving, when possible.

8.1.4.533 save_magick()

```
template<typename T >
const CImg< T > & save_magick (
    const char *const filename,
    const unsigned int bytes_per_pixel = 0 ) const
```

Save the image, using built-in ImageMagick++ library.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>bytes_per_pixel</i>	Force the number of bytes per pixel for the saving, when possible.

8.1.4.534 save_png()

```
template<typename T >
const CImg< T > & save_png (
    const char *const filename,
    const unsigned int bytes_per_pixel = 0 ) const
```

Save the image as a PNG file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>bytes_per_pixel</i>	Force the number of bytes per pixels for the saving, when possible.

8.1.4.535 save_pnm()

```
template<typename T >
const CImg< T > & save_pnm (
    const char *const filename,
    const unsigned int bytes_per_pixel = 0 ) const
```

Save the image as a PNM file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>bytes_per_pixel</i>	Force the number of bytes per pixels for the saving.

8.1.4.536 save_pnk()

```
template<typename T >
const CImg< T > & save_pnk (
    const char *const filename ) const
```

Save the image as a PNK file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.537 save_pfm()

```
template<typename T >
const CImg< T > & save_pfm (
    const char *const filename ) const
```

Save the image as a PFM file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.538 save_rgb()

```
template<typename T >
const CImg< T > & save_rgb (
    const char *const filename ) const
```

Save the image as a RGB file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.539 save_rgba()

```
template<typename T >
const CImg< T > & save_rgba (
    const char *const filename ) const
```

Save the image as a RGBA file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.540 save_tiff()

```
template<typename T >
const CImg< T > & save_tiff (
    const char *const filename,
    const unsigned int compression_type = 0,
    const float *const voxel_size = 0,
    const char *const description = 0,
    const bool use_bigtiff = true ) const
```

Save the image as a TIFF file.

Parameters

	<i>filename</i>	Filename, as a C-string.
	<i>compression_type</i>	Type of data compression. Can be { 0=None 1=ADOBE_DEFLATE 2=CCITT_T4 3=CCITT_T6 4=CCITTFAX3 5=CCITTFAX4 6=CCITTRLE 7=CCITTRLEW 8=DCS 9=DEFLATE 10=IT8BL 11=IT8CTPAD 12=IT8LW 13=IT8MP 14=JBIG 15=JP2000 16=JPEG 17=JXL 18=LERC 19=LZMA 20=LZW 21=NEXT 22=OJPEG 23=PACKBITS 24=PIXARFILM 25=PIXARLOG 26=SGILOG 27=SGILOG24 28=T43 29=T85 30=THUNDERSCAN 31=WEBP 32=ZSTD }
out	<i>voxel_size</i>	Voxel size, to be stored in the file metadata.
out	<i>description</i>	Description, to be stored in the file metadata.
	<i>use_bigtiff</i>	Indicates whether the file is saved as BigTIFF (>4 GB) or not.

Note

- libtiff support is enabled by defining the precompilation directive `cimg_use_tiff`.
- When libtiff is enabled, 2D and 3D (multipage) several channels per pixel are supported for `char`, `uchar`, `short`, `ushort`, `float` and `double` pixel types.
- If `cimg_use_tiff` is not defined at compile time the function uses `CImg<T>&save_other(const char*)`.

8.1.4.541 `save_minc2()`

```
template<typename T >
const CImg< T > & save_minc2 (
    const char *const filename,
    const char *const imitate_file = 0 ) const
```

Save the image as a MINC2 file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>imitate_file</i>	If non-zero, reference filename, as a C-string, to borrow header from.

8.1.4.542 `save_analyze()`

```
template<typename T >
const CImg< T > & save_analyze (
    const char *const filename,
    const float *const voxel_size = 0 ) const
```

Save the image as an ANALYZE7.5 or NIFTI file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>voxel_size</i>	Pointer to 3 consecutive values that tell about the voxel sizes along the X,Y and Z dimensions.

8.1.4.543 save_cimg() [1/2]

```
template<typename T >
const CImg< T > & save_cimg (
    const char *const filename,
    const bool is_compressed = false ) const
```

Save the image as a .cimg file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>is_compressed</i>	Indicates whether the file contains compressed image data.

8.1.4.544 save_cimg() [2/2]

```
template<typename T >
const CImg< T > & save_cimg (
    const char *const filename,
    const unsigned int n0,
    const unsigned int x0,
    const unsigned int y0,
    const unsigned int z0,
    const unsigned int c0 ) const
```

Save the image as a sub-image into an existing .cimg file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>n0</i>	Index of the image inside the file.
<i>x0</i>	X-coordinate of the sub-image location.
<i>y0</i>	Y-coordinate of the sub-image location.
<i>z0</i>	Z-coordinate of the sub-image location.
<i>c0</i>	C-coordinate of the sub-image location.

8.1.4.545 save_empty_cimg() [1/2]

```
template<typename T >
static void save_empty_cimg (
    const char *const filename,
    const unsigned int dx,
    const unsigned int dy = 1,
    const unsigned int dz = 1,
    const unsigned int dc = 1 ) [static]
```

Save a blank image as a .cimg file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>dx</i>	Width of the image.
<i>dy</i>	Height of the image.
<i>dz</i>	Depth of the image.
<i>dc</i>	Number of channels of the image.

Note

- All pixel values of the saved image are set to 0.
- Use this function to save large images without having to instantiate and allocate them.

8.1.4.546 `save_empty_cimg()` [2/2]

```
template<typename T >
static void save_empty_cimg (
    std::FILE *const file,
    const unsigned int dx,
    const unsigned int dy = 1,
    const unsigned int dz = 1,
    const unsigned int dc = 1 ) [static]
```

Same as `save_empty_cimg(const char *,unsigned int,unsigned int,unsigned int,unsigned int)` with a file stream argument instead of a filename string.

8.1.4.547 `save_inr()`

```
template<typename T >
const CImg< T > & save_inr (
    const char *const filename,
    const float *const voxel_size = 0 ) const
```

Save the image as an INRIMAGE-4 file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>voxel_size</i>	Pointer to 3 values specifying the voxel sizes along the X,Y and Z dimensions.

8.1.4.548 `save_exr()`

```
template<typename T >
const CImg< T > & save_exr (
    const char *const filename ) const
```

Save the image as an OpenEXR file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

Note

The OpenEXR file format is [described here](#).

8.1.4.549 save_pandore() [1/2]

```
template<typename T >
const CImg< T > & save_pandore (
    const char *const filename,
    const unsigned int colorspace = 0 ) const
```

Save the image as a Pandore-5 file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>colorspace</i>	Colorspace data field in output file (see Pandore file specifications for more information).

8.1.4.550 save_pandore() [2/2]

```
template<typename T >
const CImg< T > & save_pandore (
    std::FILE *const file,
    const unsigned int colorspace = 0 ) const
```

Same as `save_pandore(const char *,unsigned int) const` with a file stream argument instead of a filename string.

8.1.4.551 save_raw() [1/2]

```
template<typename T >
const CImg< T > & save_raw (
    const char *const filename,
    const bool is_multiplexed = false ) const
```

Save the image as a raw data file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>is_multiplexed</i>	Indicates whether the image channels are stored in a multiplexed way (<code>true</code>) or not (<code>false</code>).

Note

The .raw format does not store the image dimensions in the output file, so you have to keep track of them somewhere to be able to read the file correctly afterwards.

8.1.4.552 save_raw() [2/2]

```
template<typename T >
const CImg< T > & save_raw (
    std::FILE *const file,
    const bool is_multiplexed = false ) const
```

Same as [save_raw\(const char *,bool\) const](#) with a file stream argument instead of a filename string.

8.1.4.553 save_yuv() [1/2]

```
template<typename T >
const CImg< T > & save_yuv (
    const char *const filename,
    const unsigned int chroma_subsampling = 444,
    const bool is_rgb = true ) const
```

Save the image as a .yuv video file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>chroma_subsampling</i>	Type of chroma subsampling. Can be { 420 422 444 }.
<i>is_rgb</i>	Indicates whether pixel values of the instance image are RGB-coded (<code>true</code>) or YUV-coded (<code>false</code>).

Note

Each slice of the instance image is considered to be a single frame of the output video file.

8.1.4.554 save_yuv() [2/2]

```
template<typename T >
const CImg< T > & save_yuv (
    std::FILE *const file,
    const unsigned int chroma_subsampling = 444,
    const bool is_rgb = true ) const
```

Same as [save_yuv\(const char*,const unsigned int,const bool\) const](#) with a file stream argument instead of a filename string.

8.1.4.555 save_off() [1/2]

```
template<typename T >
template<typename tf , typename tc >
const CImg< T > & save_off (
    const CImgList< tf > & primitives,
    const CImgList< tc > & colors,
    const char *const filename ) const
```

Save 3D object as an Object File Format (.off) file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>primitives</i>	List of 3D object primitives.
<i>colors</i>	List of 3D object colors.

Note

- Instance image contains the vertices data of the 3D object.
- Textured, transparent or sphere-shaped primitives cannot be managed by the .off file format. Such primitives will be lost or simplified during file saving.
- The .off file format is [described here](#).

8.1.4.556 save_off() [2/2]

```
template<typename T >
template<typename tf , typename tc >
const CImg< T > & save_off (
    const CImgList< tf > & primitives,
    const CImgList< tc > & colors,
    std::FILE *const file ) const
```

Same as `save_off(const CImgList<tf>&,const CImgList<tc>&,const char*) const` with a file stream argument instead of a filename string.

8.1.4.557 save_video()

```
template<typename T >
const CImg< T > & save_video (
    const char *const filename,
    const unsigned int fps = 25,
    const char * codec = 0,
    const bool keep_open = false ) const
```

Save volumetric image as a video (using the OpenCV library when available).

Parameters

<i>filename</i>	Filename to write data to.
<i>fps</i>	Number of frames per second.
<i>codec</i>	Type of compression (See http://www.fourcc.org/codecs.php to see available codecs).
<i>keep_open</i>	Indicates whether the video writer associated with the specified filename must be kept open (to allow frames to be added in the same file afterwards).

8.1.4.558 save_ffmpeg_external()

```
template<typename T >
const CImg< T > & save_ffmpeg_external (
    const char *const filename,
    const unsigned int fps = 25,
    const char *const codec = 0,
    const unsigned int bitrate = 2048 ) const
```

Save volumetric image as a video, using ffmpeg external binary.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>fps</i>	Video framerate.
<i>codec</i>	Video codec, as a C-string.
<i>bitrate</i>	Video bitrate.

Note

- Each slice of the instance image is considered to be a single frame of the output video file.
- This function uses `ffmpeg`, an external executable binary provided by `FFmpeg`. It must be installed for the function to succeed.

8.1.4.559 save_gzip_external()

```
template<typename T >
const CImg< T > & save_gzip_external (
    const char *const filename ) const
```

Save the image using gzip external binary.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

Note

This function uses `gzip`, an external executable binary provided by `gzip`. It must be installed for the function to succeed.

8.1.4.560 save_graphicmagick_external()

```
template<typename T >
const CImg< T > & save_graphicmagick_external (
    const char *const filename,
    const unsigned int quality = 100 ) const
```

Save the image using GraphicsMagick's external binary.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>quality</i>	Image quality (expressed in percent), when the file format supports it.

Note

This function uses `gm`, an external executable binary provided by [GraphicsMagick](#). It must be installed for the function to succeed.

8.1.4.561 `save_imagemagick_external()`

```
template<typename T >
const CImg< T > & save_imagemagick_external (
    const char *const filename,
    const unsigned int quality = 100 ) const
```

Save the image using ImageMagick's external binary.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>quality</i>	Image quality (expressed in percent), when the file format supports it.

Note

This function uses `convert`, an external executable binary provided by [ImageMagick](#). It must be installed for the function to succeed.

8.1.4.562 `save_medcon_external()`

```
template<typename T >
const CImg< T > & save_medcon_external (
    const char *const filename ) const
```

Save the image as a Dicom file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

Note

This function uses `medcon`, an external executable binary provided by [\(X\)Medcon](#). It must be installed for the function to succeed.

8.1.4.563 save_other()

```
template<typename T >
const CImg< T > & save_other (
    const char *const filename,
    const unsigned int quality = 100 ) const
```

Parameters

<i>filename</i>	Filename, as a C-string.
<i>quality</i>	Image quality (expressed in percent), when the file format supports it.

Note

- The filename extension indicates the desired file format.
- This function tries to save the instance image as a file, using external tools from [ImageMagick](#) or [GraphicsMagick](#). At least one of these tool must be installed for the function to succeed.
- It is recommended to use the generic function `save(const char*, int) const` instead, as it can handle some file formats natively.

8.1.4.564 get_serialize()

```
template<typename T >
CImg< ucharT > get_serialize (
    const bool is_compressed = false,
    const unsigned int header_size = 0 ) const
```

Serialize a `CImg<T>` instance into a raw `CImg<unsigned char>` buffer.

Parameters

<i>is_compressed</i>	Indicates that zlib compression must be used for serialization (this requires 'cimg_use_zlib' been enabled).
<i>header_size</i>	Reserve empty bytes as a starting header.

8.2 CImgDisplay Struct Reference

Allow the creation of windows, display images on them and manage user events (keyboard, mouse and windows events).

```
#include <CImg.h>
```

Constructors / Destructor / Instance Management

- `~CImgDisplay ()`
Destructor.
- `CImgDisplay ()`

Construct an empty display.

- `ClmgDisplay (const unsigned int width, const unsigned int height, const char *const title=0, const unsigned int normalization=3, const bool is_fullscreen=false, const bool is_closed=false)`

Construct a display with specified dimensions.

- `template<typename T >
ClmgDisplay (const Clmg< T > &img, const char *const title=0, const unsigned int normalization=3, const bool is_fullscreen=false, const bool is_closed=false)`

Construct a display from an image.

- `template<typename T >
ClmgDisplay (const ClmgList< T > &list, const char *const title=0, const unsigned int normalization=3, const bool is_fullscreen=false, const bool is_closed=false)`

Construct a display from an image list.

- `ClmgDisplay (const ClmgDisplay &disp)`

Construct a display as a copy of an existing one.

- `ClmgDisplay & assign ()`

Destructor - Empty constructor [in-place version].

- `ClmgDisplay & assign (const unsigned int width, const unsigned int height, const char *const title=0, const unsigned int normalization=3, const bool is_fullscreen=false, const bool is_closed=false)`

Construct a display with specified dimensions [in-place version].

- `template<typename T >
ClmgDisplay & assign (const Clmg< T > &img, const char *const title=0, const unsigned int normalization=3, const bool is_fullscreen=false, const bool is_closed=false)`

Construct a display from an image [in-place version].

- `template<typename T >
ClmgDisplay & assign (const ClmgList< T > &list, const char *const title=0, const unsigned int normalization=3, const bool is_fullscreen=false, const bool is_closed=false)`

Construct a display from an image list [in-place version].

- `ClmgDisplay & assign (const ClmgDisplay &disp)`

Construct a display as a copy of another one [in-place version].

- `template<typename T >
static void screenshot (Clmg< T > &img)`

Take a screenshot.

- `static ClmgDisplay & empty ()`

Return a reference to an empty display.

- `static const ClmgDisplay & const_empty ()`

Return a reference to an empty display [const version].

Overloaded Operators

- `template<typename t >
ClmgDisplay & operator= (const Clmg< t > &img)`

Display image on associated window.

- `template<typename t >
ClmgDisplay & operator= (const ClmgList< t > &list)`

Display list of images on associated window.

- `ClmgDisplay & operator= (const ClmgDisplay &disp)`

Construct a display as a copy of another one [in-place version].

- `operator bool () const`

Return false if display is empty, true otherwise.

Instance Checking

- **bool is_empty () const**
Return *true* if display is empty, *false* otherwise.
- **bool is_closed () const**
Return *true* if display is closed (i.e. not visible on the screen), *false* otherwise.
- **bool is_visible () const**
Return *true* if display is visible (i.e. not closed by the user), *false* otherwise.
- **bool is_resized () const**
Return *true* if associated window has been resized on the screen, *false* otherwise.
- **bool is_moved () const**
Return *true* if associated window has been moved on the screen, *false* otherwise.
- **bool is_event () const**
Return *true* if any event has occurred on the associated window, *false* otherwise.
- **bool is_fullscreen () const**
Return *true* if current display is in fullscreen mode, *false* otherwise.
- **bool is_key () const**
Return *true* if any key is being pressed on the associated window, *false* otherwise.
- **bool is_key (const unsigned int keycode) const**
Return *true* if key specified by given keycode is being pressed on the associated window, *false* otherwise.
- **bool & is_key (const char *const keycode)**
Return *true* if key specified by given keycode is being pressed on the associated window, *false* otherwise.
- **bool is_key_sequence (const unsigned int *const keycodes_sequence, const unsigned int length, const bool remove_sequence=false)**
Return *true* if specified key sequence has been typed on the associated window, *false* otherwise.
- **bool is_keyESC () const**
Return *true* if the *ESC* key is being pressed on the associated window, *false* otherwise.

Instance Characteristics

- **int width () const**
Return display width.
- **int height () const**
Return display height.
- **unsigned int normalization () const**
Return normalization type of the display.
- **const char * title () const**
Return title of the associated window as a C-string.
- **int window_width () const**
Return width of the associated window.
- **int window_height () const**
Return height of the associated window.
- **int window_x () const**
Return X-coordinate of the associated window.
- **int window_y () const**
Return Y-coordinate of the associated window.
- **int mouse_x () const**
Return X-coordinate of the mouse pointer.
- **int mouse_y () const**
Return Y-coordinate of the mouse pointer.

- `unsigned int button () const`
Return current state of the mouse buttons.
- `int wheel () const`
Return current state of the mouse wheel.
- `const unsigned int & key (const unsigned int pos=0) const`
Return one entry from the pressed keys history.
- `const unsigned int & released_key (const unsigned int pos=0) const`
Return one entry from the released keys history.
- `float frames_per_second ()`
Return the current refresh rate, in frames per second.
- `static int screen_width ()`
Return width of the screen (current resolution along the X-axis).
- `static int screen_height ()`
Return height of the screen (current resolution along the Y-axis).
- `static unsigned int keycode (const char *const keycode)`
Return keycode corresponding to the specified string.

Window Manipulation

- `template<typename T >`
`ClmgDisplay & display (const Clmg< T > &img)`
Display image on associated window.
- `template<typename T >`
`ClmgDisplay & display (const ClmgList< T > &list, const char axis='x', const float align=0)`
Display list of images on associated window.
- `ClmgDisplay & show ()`
Show (closed) associated window on the screen.
- `ClmgDisplay & close ()`
Close (visible) associated window and make it disappear from the screen.
- `ClmgDisplay & move (const int pos_x, const int pos_y)`
Move associated window to a new location.
- `ClmgDisplay & resize (const bool force_redraw=true)`
Resize display to the size of the associated window.
- `ClmgDisplay & resize (const int width, const int height, const bool force_redraw=true)`
Resize display to the specified size.
- `template<typename T >`
`ClmgDisplay & resize (const Clmg< T > &img, const bool force_redraw=true)`
Resize display to the size of an input image.
- `ClmgDisplay & resize (const ClmgDisplay &disp, const bool force_redraw=true)`
*Resize display to the size of another *ClmgDisplay* instance.*
- `ClmgDisplay & set_normalization (const unsigned int normalization)`
Set normalization type.
- `ClmgDisplay & set_title (const char *const format,...)`
Set title of the associated window.
- `ClmgDisplay & set_fullscreen (const bool is_fullscreen, const bool force_redraw=true)`
Enable or disable fullscreen mode.
- `ClmgDisplay & toggle_fullscreen (const bool force_redraw=true)`
Toggle fullscreen mode.
- `ClmgDisplay & show_mouse ()`
Show mouse pointer.

- [CImgDisplay & hide_mouse \(\)](#)
Hide mouse pointer.
- [CImgDisplay & set_mouse \(const int pos_x, const int pos_y\)](#)
Move mouse pointer to a specified location.
- [CImgDisplay & set_button \(\)](#)
Simulate a mouse button release event.
- [CImgDisplay & set_button \(const unsigned int button, const bool is_pressed=true\)](#)
Simulate a mouse button press or release event.
- [CImgDisplay & set_wheel \(\)](#)
Flush all mouse wheel events.
- [CImgDisplay & set_wheel \(const int amplitude\)](#)
Simulate a wheel event.
- [CImgDisplay & set_key \(\)](#)
Flush all key events.
- [CImgDisplay & set_key \(const unsigned int keycode, const bool is_pressed=true\)](#)
Simulate a keyboard press/release event.
- [CImgDisplay & flush \(\)](#)
Flush all display events.
- [CImgDisplay & wait \(\)](#)
Wait for any user event occurring on the current display.
- [CImgDisplay & wait \(const unsigned int milliseconds\)](#)
Wait for a given number of milliseconds since the last call to [wait\(\)](#).
- [template<typename T > CImgDisplay & render \(const CImg< T > &img\)](#)
Render image into internal display buffer.
- [CImgDisplay & paint \(\)](#)
Paint internal display buffer on associated window.
- [template<typename T > const CImgDisplay & snapshot \(CImg< T > &img\) const](#)
Take a snapshot of the associated window content.
- [static void wait \(CImgDisplay &disp1\)](#)
*Wait for any event occurring on the display *disp1*.*
- [static void wait \(CImgDisplay &disp1, CImgDisplay &disp2\)](#)
*Wait for any event occurring either on the display *disp1* or *disp2*.*
- [static void wait \(CImgDisplay &disp1, CImgDisplay &disp2, CImgDisplay &disp3\)](#)
*Wait for any event occurring either on the display *disp1*, *disp2* or *disp3*.*
- [static void wait \(CImgDisplay &disp1, CImgDisplay &disp2, CImgDisplay &disp3, CImgDisplay &disp4\)](#)
*Wait for any event occurring either on the display *disp1*, *disp2*, *disp3* or *disp4*.*
- [static void wait \(CImgDisplay &disp1, CImgDisplay &disp2, CImgDisplay &disp3, CImgDisplay &disp4, CImgDisplay &disp5\)](#)
*Wait for any event occurring either on the display *disp1*, *disp2*, *disp3*, *disp4* or *disp5*.*
- [static void wait \(CImgDisplay &disp1, CImgDisplay &disp2, CImgDisplay &disp3, CImgDisplay &disp4, CImgDisplay &disp5, CImgDisplay &disp6\)](#)
*Wait for any event occurring either on the display *disp1*, *disp2*, *disp3*, *disp4*, ... *disp6*.*
- [static void wait \(CImgDisplay &disp1, CImgDisplay &disp2, CImgDisplay &disp3, CImgDisplay &disp4, CImgDisplay &disp5, CImgDisplay &disp6, CImgDisplay &disp7\)](#)
*Wait for any event occurring either on the display *disp1*, *disp2*, *disp3*, *disp4*, ... *disp7*.*
- [static void wait \(CImgDisplay &disp1, CImgDisplay &disp2, CImgDisplay &disp3, CImgDisplay &disp4, CImgDisplay &disp5, CImgDisplay &disp6, CImgDisplay &disp7, CImgDisplay &disp8\)](#)
*Wait for any event occurring either on the display *disp1*, *disp2*, *disp3*, *disp4*, ... *disp8*.*
- [static void wait \(CImgDisplay &disp1, CImgDisplay &disp2, CImgDisplay &disp3, CImgDisplay &disp4, CImgDisplay &disp5, CImgDisplay &disp6, CImgDisplay &disp7, CImgDisplay &disp8, CImgDisplay &disp9\)](#)

Wait for any event occurring either on the display *disp1*, *disp2*, *disp3*, *disp4*, ... *disp9*.

- `static void wait (CImgDisplay &disp1, CImgDisplay &disp2, CImgDisplay &disp3, CImgDisplay &disp4, CImgDisplay &disp5, CImgDisplay &disp6, CImgDisplay &disp7, CImgDisplay &disp8, CImgDisplay &disp9, CImgDisplay &disp10)`

Wait for any event occurring either on the display *disp1*, *disp2*, *disp3*, *disp4*, ... *disp10*.

- `static void wait_all ()`

Wait for any window event occurring in any opened *CImgDisplay*.

- `template<typename T >`
`static void screenshot (const int x0, const int y0, const int x1, const int y1, CImg< T > &img)`

Take a snapshot of the current screen content.

8.2.1 Detailed Description

Allow the creation of windows, display images on them and manage user events (keyboard, mouse and windows events).

CImgDisplay functions rely on a low-level graphic library to perform: it can be either **X-Window** (X11, for Unix-based systems) or **GDI32** (for Windows-based systems). If both libraries are missing, *CImgDisplay* will not be able to display images on screen, and will enter a minimal mode where warning messages will be outputted each time the program is trying to call one of the *CImgDisplay* function.

The configuration variable `cimg_display` indicates the graphic library used. It is set automatically by *CImg* when one of these graphic libraries has been detected. But, you can override its value if necessary. Valid choices are:

- 0: Disable display capabilities.
- 1: Use **X-Window** (X11) library.
- 2: Use **GDI32** library.

Remember to link your program against **X11** or **GDI32** libraries if you use *CImgDisplay*.

8.2.2 Constructor & Destructor Documentation

8.2.2.1 ~CImgDisplay()

```
~CImgDisplay ( )
```

Destructor.

Note

If the associated window is visible on the screen, it is closed by the call to the destructor.

8.2.2.2 CImgDisplay() [1/5]

```
CImgDisplay ( )
```

Construct an empty display.

Note

Constructing an empty [CImgDisplay](#) instance does not make a window appearing on the screen, until display of valid data is performed.

Example

```
CImgDisplay disp; // Does actually nothing
...
disp.display(img); // Construct new window and display image in it
```

8.2.2.3 CImgDisplay() [2/5]

```
CImgDisplay (
    const unsigned int width,
    const unsigned int height,
    const char *const title = 0,
    const unsigned int normalization = 3,
    const bool is_fullscreen = false,
    const bool is_closed = false )
```

Construct a display with specified dimensions.

Parameters

<i>width</i>	Window width.
<i>height</i>	Window height.
<i>title</i>	Window title.
<i>normalization</i>	Normalization type (0=none, 1=always, 2=once, 3=pixel type-dependent, see normalization()).
<i>is_fullscreen</i>	Indicates whether fullscreen mode is enabled.
<i>is_closed</i>	Indicates whether associated window is initially visible.

Note

A black background is initially displayed on the associated window.

8.2.2.4 CImgDisplay() [3/5]

```
template<typename T >
CImgDisplay (
    const CImg< T > & img,
    const char *const title = 0,
    const unsigned int normalization = 3,
```

```
const bool is_fullscreen = false,  
const bool is_closed = false ) [explicit]
```

Construct a display from an image.

Parameters

<i>img</i>	Image used as a model to create the window.
<i>title</i>	Window title.
<i>normalization</i>	Normalization type (0=none, 1=always, 2=once, 3=pixel type-dependent, see normalization()).
<i>is_fullscreen</i>	Indicates whether fullscreen mode is enabled.
<i>is_closed</i>	Indicates whether associated window is initially visible.

Note

The pixels of the input image are initially displayed on the associated window.

8.2.2.5 CImgDisplay() [4/5]

```
template<typename T >
CImgDisplay (
    const CImgList< T > & list,
    const char *const title = 0,
    const unsigned int normalization = 3,
    const bool is_fullscreen = false,
    const bool is_closed = false ) [explicit]
```

Construct a display from an image list.

Parameters

<i>list</i>	The images list to display.
<i>title</i>	Window title.
<i>normalization</i>	Normalization type (0=none, 1=always, 2=once, 3=pixel type-dependent, see normalization()).
<i>is_fullscreen</i>	Indicates whether fullscreen mode is enabled.
<i>is_closed</i>	Indicates whether associated window is initially visible.

Note

All images of the list, appended along the X-axis, are initially displayed on the associated window.

8.2.2.6 CImgDisplay() [5/5]

```
CImgDisplay (
    const CImgDisplay & disp )
```

Construct a display as a copy of an existing one.

Parameters

<i>disp</i>	Display instance to copy.
-------------	---------------------------

Note

The pixel buffer of the input window is initially displayed on the associated window.

8.2.3 Member Function Documentation**8.2.3.1 screenshot() [1/2]**

```
template<typename T >
static void screenshot (
    CImg< T > & img ) [static]
```

Take a screenshot.

Parameters

out	<i>img</i>	Output screenshot. Can be empty on input
-----	------------	--

8.2.3.2 assign()

```
CImgDisplay & assign ( )
```

Destructor - Empty constructor **[in-place version]**.

Note

Replace the current instance by an empty display.

8.2.3.3 empty()

```
static CImgDisplay & empty ( ) [static]
```

Return a reference to an empty display.

Note

Can be useful for writing function prototypes where one of the argument (of type [CImgDisplay&](#)) must have a default value.

Example

```
void foo(CImgDisplay& disp=CImgDisplay::empty());
```

8.2.3.4 operator=() [1/3]

```
template<typename t >
CImgDisplay & operator= (
    const CImg< t > & img )
```

Display image on associated window.

Note

`disp = img` is equivalent to `disp.display(img)`.

8.2.3.5 operator=() [2/3]

```
template<typename t >
CImgDisplay & operator= (
    const CImgList< t > & list )
```

Display list of images on associated window.

Note

`disp = list` is equivalent to `disp.display(list)`.

8.2.3.6 operator=() [3/3]

```
CImgDisplay & operator= (
    const CImgDisplay & disp )
```

Construct a display as a copy of another one **[in-place version]**.

Note

Equivalent to `assign(const CImgDisplay&)`.

8.2.3.7 operator bool()

```
operator bool ( ) const
```

Return `false` if display is empty, `true` otherwise.

Note

`if (disp) { ... }` is equivalent to `if (!disp.is_empty()) { ... }`.

8.2.3.8 is_closed()

```
bool is_closed ( ) const
```

Return `true` if display is closed (i.e. not visible on the screen), `false` otherwise.

Note

- When a user physically closes the associated window, the display is set to closed.
- A closed display is not destroyed. Its associated window can be show again on the screen using `show()`.

8.2.3.9 is_key() [1/3]

```
bool is_key ( ) const
```

Return `true` if any key is being pressed on the associated window, `false` otherwise.

Note

The functions below do the same only for specific keys.

8.2.3.10 is_key() [2/3]

```
bool is_key (
    const unsigned int keycode ) const
```

Return `true` if key specified by given keycode is being pressed on the associated window, `false` otherwise.

Parameters

<i>keycode</i>	Keycode to test.
----------------	------------------

Note

Keycode constants are defined in the `cimg` namespace and are architecture-dependent. Use them to ensure your code stay portable (see [cimg::keyESC](#)).

Example

```
CImgDisplay disp(400,400);
while (!disp.is_closed()) {
    if (disp.key(cimg::keyTAB)) { ... } // Equivalent to 'if (disp.is_keyTAB())'
    disp.wait();
}
```

8.2.3.11 `is_key()` [3/3]

```
bool & is_key (
    const char *const keycode )
```

Return `true` if key specified by given keycode is being pressed on the associated window, `false` otherwise.

Parameters

<i>keycode</i>	C-string containing the keycode label of the key to test.
----------------	---

Note

Use it when the key you want to test can be dynamically set by the user.

Example

```
CImgDisplay disp(400,400);
const char *const keycode = "TAB";
while (!disp.is_closed()) {
    if (disp.is_key(keycode)) { ... } // Equivalent to 'if (disp.is_keyTAB())'
    disp.wait();
}
```

8.2.3.12 `is_key_sequence()`

```
bool is_key_sequence (
    const unsigned int *const keycodes_sequence,
    const unsigned int length,
    const bool remove_sequence = false )
```

Return `true` if specified key sequence has been typed on the associated window, `false` otherwise.

Parameters

<i>keycodes_sequence</i>	Buffer of keycodes to test.
<i>length</i>	Number of keys in the <i>keycodes_sequence</i> buffer.
<i>remove_sequence</i>	Indicates whether the key sequence must be removed from the key history, if found.

Note

Keycode constants are defined in the `cimg` namespace and are architecture-dependent. Use them to ensure your code stay portable (see [cimg::keyESC](#)).

Example

```
CImgDisplay disp(400,400);
const unsigned int key_seq[] = { cimg::keyCTRLLEFT, cimg::keyD };
while (!disp.is_closed()) {
    if (disp.is_key_sequence(key_seq,2)) { ... } // Test for the 'CTRL+D' keyboard event
    disp.wait();
}
```

8.2.3.13 is_keyESC()

```
bool is_keyESC ( ) const
```

Return `true` if the ESC key is being pressed on the associated window, `false` otherwise.

Note

Similar functions exist for all keys managed by `CImg` (see [cimg::keyESC](#)).

8.2.3.14 width()

```
int width ( ) const
```

Return display width.

Note

The width of the display (i.e. the width of the pixel data buffer associated with the [CImgDisplay](#) instance) may be different from the actual width of the associated window.

8.2.3.15 height()

```
int height ( ) const
```

Return display height.

Note

The height of the display (i.e. the height of the pixel data buffer associated with the [CImgDisplay](#) instance) may be different from the actual height of the associated window.

8.2.3.16 normalization()

```
unsigned int normalization ( ) const
```

Return normalization type of the display.

The normalization type indicates how the values of an input image are normalized by the [CImgDisplay](#) to be correctly displayed. The range of values for pixels displayed on the screen is $[0, 255]$. If the range of values of the data to display is different, a normalization may be required for displaying the data in a correct way. The normalization type can be one of:

- 0: Value normalization is disabled. It is then assumed that all input data to be displayed by the [CImgDisplay](#) instance have values in range $[0, 255]$.
- 1: Value normalization is always performed (this is the default behavior). Before displaying an input image, its values will be (virtually) stretched in range $[0, 255]$, so that the contrast of the displayed pixels will be maximum. Use this mode for images whose minimum and maximum values are not prescribed to known values (e.g. float-valued images). Note that when normalized versions of images are computed for display purposes, the actual values of these images are not modified.
- 2: Value normalization is performed once (on the first image display), then the same normalization coefficients are kept for next displayed frames.
- 3: Value normalization depends on the pixel type of the data to display. For integer pixel types, the normalization is done regarding the minimum/maximum values of the type (no normalization occurs then for `unsigned char`). For float-valued pixel types, the normalization is done regarding the minimum/maximum value of the image data instead.

8.2.3.17 title()

```
const char * title ( ) const
```

Return title of the associated window as a C-string.

Note

Window title may be not visible, depending on the used window manager or if the current display is in fullscreen mode.

8.2.3.18 window_width()

```
int window_width ( ) const
```

Return width of the associated window.

Note

The width of the display (i.e. the width of the pixel data buffer associated with the [CImgDisplay](#) instance) may be different from the actual width of the associated window.

8.2.3.19 window_height()

```
int window_height ( ) const
```

Return height of the associated window.

Note

The height of the display (i.e. the height of the pixel data buffer associated with the [CImgDisplay](#) instance) may be different from the actual height of the associated window.

8.2.3.20 window_x()

```
int window_x ( ) const
```

Return X-coordinate of the associated window.

Note

The returned coordinate corresponds to the location of the upper-left corner of the associated window.

8.2.3.21 window_y()

```
int window_y ( ) const
```

Return Y-coordinate of the associated window.

Note

The returned coordinate corresponds to the location of the upper-left corner of the associated window.

8.2.3.22 mouse_x()

```
int mouse_x ( ) const
```

Return X-coordinate of the mouse pointer.

Note

- If the mouse pointer is outside window area, `-1` is returned.
- Otherwise, the returned value is in the range `[0, width\(\)-1]`.

8.2.3.23 mouse_y()

```
int mouse_y ( ) const
```

Return Y-coordinate of the mouse pointer.

Note

- If the mouse pointer is outside window area, `-1` is returned.
- Otherwise, the returned value is in the range `[0, height\(\)-1]`.

8.2.3.24 button()

```
unsigned int button ( ) const
```

Return current state of the mouse buttons.

Note

Three mouse buttons can be managed. If one button is pressed, its corresponding bit in the returned value is set:

- bit 0 (value 0x1): State of the left mouse button.
- bit 1 (value 0x2): State of the right mouse button.
- bit 2 (value 0x4): State of the middle mouse button.

Several bits can be activated if more than one button are pressed at the same time.

Example

```
CImgDisplay disp(400,400);
while (!disp.is_closed()) {
    if (disp.button() & 1) { // Left button clicked
        ...
    }
    if (disp.button() & 2) { // Right button clicked
        ...
    }
    if (disp.button() & 4) { // Middle button clicked
        ...
    }
    disp.wait();
}
```

8.2.3.25 wheel()

```
int wheel ( ) const
```

Return current state of the mouse wheel.

Note

- The returned value can be positive or negative depending on whether the mouse wheel has been scrolled forward or backward.
- Scrolling the wheel forward add 1 to the wheel value.
- Scrolling the wheel backward subtract 1 from the wheel value.
- The returned value cumulates the number of forward or backward scrolls since the creation of the display, or since the last reset of the wheel value (using [set_wheel\(\)](#)). It is strongly recommended to quickly reset the wheel counter when an action has been performed regarding the current wheel value. Otherwise, the returned wheel value may be for instance 0 despite the fact that many scrolls have been done (as many in forward as in backward directions).

Example

```
CImgDisplay disp(400,400);
while (!disp.is_closed()) {
    if (disp.wheel()) {
        int counter = disp.wheel(); // Read the state of the mouse wheel
        ... // Do what you want with 'counter'
        disp.set_wheel(); // Reset the wheel value to 0
    }
    disp.wait();
}
```


8.2.3.26 key()

```
const unsigned int & key (  
    const unsigned int pos = 0 ) const
```

Return one entry from the pressed keys history.

Parameters

<i>pos</i>	Index to read from the pressed keys history (index 0 corresponds to latest entry).
------------	--

Returns

Keycode of a pressed key or 0 for a released key.

Note

- Each [CImgDisplay](#) stores a history of the pressed keys in a buffer of size 128. When a new key is pressed, its keycode is stored in the pressed keys history. When a key is released, 0 is put instead. This means that up to the 64 last pressed keys may be read from the pressed keys history. When a new value is stored, the pressed keys history is shifted so that the latest entry is always stored at position 0.
- Keycode constants are defined in the `cimg` namespace and are architecture-dependent. Use them to ensure your code stay portable (see [cimg::keyESC](#)).

8.2.3.27 released_key()

```
const unsigned int & released_key (
    const unsigned int pos = 0 ) const
```

Return one entry from the released keys history.

Parameters

<i>pos</i>	Index to read from the released keys history (index 0 corresponds to latest entry).
------------	---

Returns

Keycode of a released key or 0 for a pressed key.

Note

- Each [CImgDisplay](#) stores a history of the released keys in a buffer of size 128. When a new key is released, its keycode is stored in the pressed keys history. When a key is pressed, 0 is put instead. This means that up to the 64 last released keys may be read from the released keys history. When a new value is stored, the released keys history is shifted so that the latest entry is always stored at position 0.
- Keycode constants are defined in the `cimg` namespace and are architecture-dependent. Use them to ensure your code stay portable (see [cimg::keyESC](#)).

8.2.3.28 keycode()

```
static unsigned int keycode (
    const char *const keycode ) [static]
```

Return keycode corresponding to the specified string.

Note

Keycode constants are defined in the cimg namespace and are architecture-dependent. Use them to ensure your code stay portable (see [cimg::keyESC](#)).

Example

```
const unsigned int keyTAB = CImgDisplay::keycode("TAB"); // Return cimg::keyTAB
```

8.2.3.29 frames_per_second()

```
float frames_per_second ( )
```

Return the current refresh rate, in frames per second.

Note

Returns a meaningful value when the current instance is used to display successive frames. It measures the delay between successive calls to [frames_per_second\(\)](#).

8.2.3.30 display() [1/2]

```
template<typename T >
CImgDisplay & display (
    const CImg< T > & img )
```

Display image on associated window.

Parameters

<i>img</i>	Input image to display.
------------	-------------------------

Note

This function returns immediately.

8.2.3.31 display() [2/2]

```
template<typename T >
CImgDisplay & display (
    const CImgList< T > & list,
    const char axis = 'x',
    const float align = 0 )
```

Display list of images on associated window.

Parameters

<i>list</i>	List of images to display.
<i>axis</i>	Axis along which to append the images for the visualization (can be x, y, z or c).
<i>align</i>	Relative position of aligned images when displaying lists with images of different sizes (0 for upper-left, 0.5 for centering and 1 for lower-right).

Note

This function returns immediately.

8.2.3.32 show()

```
CImgDisplay & show ( )
```

Show (closed) associated window on the screen.

Note

- Force the associated window of a display to be visible on the screen, even if it has been closed before.
- Using `show()` on a visible display does nothing.

8.2.3.33 close()

```
CImgDisplay & close ( )
```

Close (visible) associated window and make it disappear from the screen.

Note

- A closed display only means the associated window is not visible anymore. This does not mean the display has been destroyed. Use `show()` to make the associated window reappear.
- Using `close()` on a closed display does nothing.

8.2.3.34 move()

```
CImgDisplay & move (
    const int pos_x,
    const int pos_y )
```

Move associated window to a new location.

Parameters

<code>pos↔ _x</code>	X-coordinate of the new window location.
<code>pos↔ _y</code>	Y-coordinate of the new window location.

Note

Depending on the window manager behavior, this function may not succeed (no exceptions are thrown nevertheless).

8.2.3.35 `resize()` [1/4]

```
CImgDisplay & resize (
    const bool force_redraw = true )
```

Resize display to the size of the associated window.

Parameters

<i>force_redraw</i>	Indicates whether the previous window content must be updated and refreshed as well.
---------------------	--

Note

- Calling this function ensures that `width()` and `window_width()` become equal, as well as `height()` and `window_height()`.
- The associated window is also resized to specified dimensions.

8.2.3.36 `resize()` [2/4]

```
CImgDisplay & resize (
    const int width,
    const int height,
    const bool force_redraw = true )
```

Resize display to the specified size.

Parameters

<i>width</i>	Requested display width.
<i>height</i>	Requested display height.
<i>force_redraw</i>	Indicates whether the previous window content must be updated and refreshed as well.

Note

The associated window is also resized to specified dimensions.

8.2.3.37 `resize()` [3/4]

```
template<typename T >
CImgDisplay & resize (
    const CImg< T > & img,
    const bool force_redraw = true )
```

Resize display to the size of an input image.

Parameters

<i>img</i>	Input image to take size from.
<i>force_redraw</i>	Indicates whether the previous window content must be resized and updated as well.

Note

- Calling this function ensures that `width()` and `img.width()` become equal, as well as `height()` and `img.height()`.
- The associated window is also resized to specified dimensions.

8.2.3.38 `resize()` [4/4]

```
CImgDisplay &resize (
    const CImgDisplay & disp,
    const bool force_redraw = true )
```

Resize display to the size of another `CImgDisplay` instance.

Parameters

<i>disp</i>	Input display to take size from.
<i>force_redraw</i>	Indicates whether the previous window content must be resized and updated as well.

Note

- Calling this function ensures that `width()` and `disp.width()` become equal, as well as `height()` and `disp.height()`.
- The associated window is also resized to specified dimensions.

8.2.3.39 `set_normalization()`

```
CImgDisplay &set_normalization (
    const unsigned int normalization )
```

Set normalization type.

Parameters

<i>normalization</i>	New normalization mode.
----------------------	-------------------------

8.2.3.40 `set_title()`

```
CImgDisplay &set_title (
    const char *const format,
    ... )
```

Set title of the associated window.

Parameters

<i>format</i>	C-string containing the format of the title, as with <code>std::printf()</code> .
---------------	---

Warning

As the first argument is a format string, it is highly recommended to write
`disp.set_title("%s", window_title);`

instead of

`disp.set_title(window_title);`

if `window_title` can be arbitrary, to prevent memory access violations (or undefined behavior).

8.2.3.41 `set_fullscreen()`

```
CImgDisplay & set_fullscreen (
    const bool is_fullscreen,
    const bool force_redraw = true )
```

Enable or disable fullscreen mode.

Parameters

<i>is_fullscreen</i>	Indicates whether fullscreen mode must be activated.
<i>force_redraw</i>	Indicates whether the previous window content must be displayed as well.

Note

- When fullscreen mode is enabled, the associated window fills the entire screen but the size of the current display is not modified.
- The screen resolution may be switched to fit the associated window size and ensure it appears as large as possible. For X-Window (X11) users, the configuration flag `cimg_use_xrandr` has to be set to allow the screen resolution change (requires the X11 extensions to be enabled).

8.2.3.42 `toggle_fullscreen()`

```
CImgDisplay & toggle_fullscreen (
    const bool force_redraw = true )
```

Toggle fullscreen mode.

Parameters

<i>force_redraw</i>	Indicates whether the previous window content must be displayed as well.
---------------------	--

Note

Enable fullscreen mode if it was not enabled, and disable it otherwise.

8.2.3.43 `show_mouse()`

```
CImgDisplay & show_mouse ( )
```

Show mouse pointer.

Note

Depending on the window manager behavior, this function may not succeed (no exceptions are thrown nevertheless).

8.2.3.44 hide_mouse()

```
CImgDisplay & hide_mouse ( )
```

Hide mouse pointer.

Note

Depending on the window manager behavior, this function may not succeed (no exceptions are thrown nevertheless).

8.2.3.45 set_mouse()

```
CImgDisplay & set_mouse (
    const int pos_x,
    const int pos_y )
```

Move mouse pointer to a specified location.

Note

Depending on the window manager behavior, this function may not succeed (no exceptions are thrown nevertheless).

8.2.3.46 set_button() [1/2]

```
CImgDisplay & set_button ( )
```

Simulate a mouse button release event.

Note

All mouse buttons are considered released at the same time.

8.2.3.47 set_button() [2/2]

```
CImgDisplay & set_button (
    const unsigned int button,
    const bool is_pressed = true )
```

Simulate a mouse button press or release event.

Parameters

<i>button</i>	Buttons event code, where each button is associated with a single bit.
<i>is_pressed</i>	Indicates whether the mouse button is considered as pressed or released.

8.2.3.48 set_wheel() [1/2]

```
CImgDisplay & set_wheel ( )
```

Flush all mouse wheel events.

Note

Make `wheel()` to return 0, if called afterwards.

8.2.3.49 set_wheel() [2/2]

```
CImgDisplay & set_wheel (
    const int amplitude )
```

Simulate a wheel event.

Parameters

<i>amplitude</i>	Amplitude of the wheel scrolling to simulate.
------------------	---

Note

Make `wheel()` to return `amplitude`, if called afterwards.

8.2.3.50 set_key() [1/2]

```
CImgDisplay & set_key ( )
```

Flush all key events.

Note

Make `key()` to return 0, if called afterwards.

8.2.3.51 set_key() [2/2]

```
CImgDisplay & set_key (
    const unsigned int keycode,
    const bool is_pressed = true )
```

Simulate a keyboard press/release event.

Parameters

<i>keycode</i>	Keycode of the associated key.
<i>is_pressed</i>	Indicates whether the key is considered as pressed or released.

Note

Keycode constants are defined in the `cimg` namespace and are architecture-dependent. Use them to ensure your code stay portable (see [cimg::keyESC](#)).

8.2.3.52 flush()

```
CImgDisplay & flush ( )
```

Flush all display events.

Note

Remove all passed events from the current display.

8.2.3.53 wait()

```
CImgDisplay & wait (
    const unsigned int milliseconds )
```

Wait for a given number of milliseconds since the last call to [wait\(\)](#).

Parameters

<i>milliseconds</i>	Number of milliseconds to wait for.
---------------------	-------------------------------------

Note

Similar to `cimg::wait()`.

8.2.3.54 render()

```
template<typename T >
CImgDisplay & render (
    const CImg< T > & img )
```

Render image into internal display buffer.

Parameters

<i>img</i>	Input image data to render.
------------	-----------------------------

Note

- Convert image data representation into the internal display buffer (architecture-dependent structure).
- The content of the associated window is not modified, until [paint\(\)](#) is called.
- Should not be used for common [CImgDisplay](#) uses, since [display\(\)](#) is more useful.

8.2.3.55 paint()

```
CImgDisplay & paint ( )
```

Paint internal display buffer on associated window.

Note

- Update the content of the associated window with the internal display buffer, e.g. after a [render\(\)](#) call.
- Should not be used for common [CImgDisplay](#) uses, since [display\(\)](#) is more useful.

8.2.3.56 screenshot() [2/2]

```
template<typename T >
static void screenshot (
    const int x0,
    const int y0,
    const int x1,
    const int y1,
    CImg< T > & img ) [static]
```

Take a snapshot of the current screen content.

Parameters

	<i>x0</i>	X-coordinate of the upper left corner.
	<i>y0</i>	Y-coordinate of the upper left corner.
	<i>x1</i>	X-coordinate of the lower right corner.
	<i>y1</i>	Y-coordinate of the lower right corner.
out	<i>img</i>	Output screenshot. Can be empty on input

8.2.3.57 snapshot()

```
template<typename T >
const CImgDisplay & snapshot (
    CImg< T > & img ) const
```

Take a snapshot of the associated window content.

Parameters

out	<i>img</i>	Output snapshot. Can be empty on input.
-----	------------	---

8.3 CImgException Struct Reference

Instances of [CImgException](#) are thrown when errors are encountered in a `CImg` function call.

```
#include <CImg.h>
```

Inherits exception.

Inherited by `CImgArgumentException`, `CImgDisplayException`, `CImgIOException`, `CImgInstanceException`, and `CImgWarningException`.

Public Member Functions

- `const char * what () const throw ()`

Return a C-string containing the error message associated with the thrown exception.

8.3.1 Detailed Description

Instances of [CImgException](#) are thrown when errors are encountered in a `CImg` function call.

Overview

[CImgException](#) is the base class of all exceptions thrown by `CImg` (except **`CImgAbortException`**). [CImgException](#) is never thrown directly. Derived classes that specify the type of error are thrown instead. These classes can be:

- **`CImgAbortException`**: Thrown when a computationally-intensive function is aborted by an external signal. This is the only `non-derived` exception class.
- **`CImgArgumentException`**: Thrown when one argument of a called `CImg` function is invalid. This is probably one of the most commonly thrown exceptions in `CImg`. For instance, the following example throws a `CImgArgumentException`:

```
CImg<float> img(100,100,1,3); // Define a 100x100 color image with float-valued pixels
img.mirror('e'); // Try to mirror image along the (non-existing) 'e'-axis
```
- **`CImgDisplayException`**: Thrown when something went wrong during the display of images in [CImgDisplay](#) instances.
- **`CImgInstanceException`**: Thrown when an instance associated with a called `CImg` function does not meet the function's requirements. For instance, the following example throws a `CImgInstanceException`:

```
const CImg<float> img; // Define an empty image
const float value = img.at(0); // Try to read first pixel value (does not exist)
```
- **`CImgIOException`**: Thrown when an error occurred when trying to load or save image files. This happens when trying to read files that do not exist or with invalid formats. For instance, the following example throws a `CImgIOException`:

```
const CImg<float> img("missing_file.jpg"); // Try to load a file that does not exist
```
- **`CImgWarningException`**: Thrown only if configuration macro `cimg_strict_warnings` is set, and when a `CImg` function has to display a warning message (see [cimg::warn\(\)](#)).

It is not recommended to throw [CImgException](#) instances manually, as they are intended to be thrown only by the `CImg` library. When an error occurs in a library function call, `CImg` may display error messages on the screen or on the standard output, depending on the current `CImg` exception mode. The `CImg` exception mode can be queried and set by functions [cimg::exception_mode\(\)](#) and [cimg::exception_mode\(unsigned int\)](#).

Exceptions handling

In all cases, when an error occurs in CImg, an instance of the corresponding exception class is thrown. This may lead the program to break (this is the default behavior), but you can bypass this behavior by handling the exceptions by yourself, using a usual `try { ... } catch ( ) { ... }` block, as in the following example:

```
#include "CImg.h"
using namespace cimg_library;
int main() {
    cimg::exception_mode(0); // Enable quiet exception mode
    try {
        ... // Here, do what you want to stress CImg
    } catch (CImgException& e) { // You succeeded: something went wrong!
        std::fprintf(stderr, "CImg Library Error: %s", e.what()); // Display your custom error message
        ... // Perform necessary recovery or cleanup steps here
    }
}
```

8.4 CImgList< T > Struct Template Reference

Represents a list of images CImg<T>.

```
#include <CImg.h>
```

Public Types

- `typedef CImg< T > * iterator`
Simple iterator type to loop through each image of a list.
- `typedef const CImg< T > * const_iterator`
Simple const iterator type, to loop through each image of a `const` list instance.
- `typedef T value_type`
Pixel value type.

Constructors / Destructor / Instance Management

- `~CImgList ()`
Destructor.
- `CImgList ()`
Default constructor.
- `CImgList (const unsigned int n)`
Construct list containing empty images.
- `CImgList (const unsigned int n, const unsigned int width, const unsigned int height=1, const unsigned int depth=1, const unsigned int spectrum=1)`
Construct list containing images of specified size.
- `CImgList (const unsigned int n, const unsigned int width, const unsigned int height, const unsigned int depth, const unsigned int spectrum, const T &val)`
Construct list containing images of specified size, and initialize pixel values.
- `template<typename t >`
`CImgList (const unsigned int n, const CImg< t > &img, const bool is_shared=false)`
Construct list containing copies of an input image.
- `template<typename t >`
`CImgList (const CImg< t > &img, const bool is_shared=false)`
Construct list from one image.

- `template<typename t1 , typename t2 >`
`ClmgList (const Clmg< t1 > &img1, const Clmg< t2 > &img2, const bool is_shared=false)`
Construct list from two images.
- `template<typename t1 , typename t2 , typename t3 >`
`ClmgList (const Clmg< t1 > &img1, const Clmg< t2 > &img2, const Clmg< t3 > &img3, const bool is_←
shared=false)`
Construct list from three images.
- `template<typename t1 , typename t2 , typename t3 , typename t4 >`
`ClmgList (const Clmg< t1 > &img1, const Clmg< t2 > &img2, const Clmg< t3 > &img3, const Clmg< t4
> &img4, const bool is_shared=false)`
Construct list from four images.
- `template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 >`
`ClmgList (const Clmg< t1 > &img1, const Clmg< t2 > &img2, const Clmg< t3 > &img3, const Clmg< t4
> &img4, const Clmg< t5 > &img5, const bool is_shared=false)`
Construct list from five images.
- `template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 , typename t6 >`
`ClmgList (const Clmg< t1 > &img1, const Clmg< t2 > &img2, const Clmg< t3 > &img3, const Clmg< t4
> &img4, const Clmg< t5 > &img5, const Clmg< t6 > &img6, const bool is_shared=false)`
Construct list from six images.
- `template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 , typename t6 , typename t7 >`
`ClmgList (const Clmg< t1 > &img1, const Clmg< t2 > &img2, const Clmg< t3 > &img3, const Clmg<
t4 > &img4, const Clmg< t5 > &img5, const Clmg< t6 > &img6, const Clmg< t7 > &img7, const bool
is_shared=false)`
Construct list from seven images.
- `template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 , typename t6 , typename t7 , typename t8 >`
`ClmgList (const Clmg< t1 > &img1, const Clmg< t2 > &img2, const Clmg< t3 > &img3, const Clmg< t4
> &img4, const Clmg< t5 > &img5, const Clmg< t6 > &img6, const Clmg< t7 > &img7, const Clmg< t8
> &img8, const bool is_shared=false)`
Construct list from eight images.
- `template<typename t >`
`ClmgList (const ClmgList< t > &list)`
Construct list copy.
- **ClmgList** (`const ClmgList< T > &list`)
Construct list copy [specialization].
- `template<typename t >`
`ClmgList (const ClmgList< t > &list, const bool is_shared)`
Construct list copy, and force the shared state of the list elements.
- `ClmgList (const char *const filename)`
Construct list by reading the content of a file.
- `ClmgList (const ClmgDisplay &disp)`
Construct list from the content of a display window.
- `ClmgList< T > get_shared ()`
Return a list with elements being shared copies of images in the list instance.
- `const ClmgList< T > get_shared () const`
Return a list with elements being shared copies of images in the list instance [const version].
- `ClmgList< T > & assign ()`
Destructor [in-place version].
- `ClmgList< T > & clear ()`
Destructor [in-place version].
- `ClmgList< T > & assign (const unsigned int n)`
Construct list containing empty images [in-place version].
- `ClmgList< T > & assign (const unsigned int n, const unsigned int width, const unsigned int height=1, const
unsigned int depth=1, const unsigned int spectrum=1)`

Construct list containing images of specified size **[in-place version]**.

- `CImgList< T > & assign (const unsigned int n, const unsigned int width, const unsigned int height, const unsigned int depth, const unsigned int spectrum, const T &val)`

Construct list containing images of specified size, and initialize pixel values **[in-place version]**.

- `template<typename t >
CImgList< T > & assign (const unsigned int n, const CImg< t > &img, const bool is_shared=false)`

Construct list containing copies of an input image **[in-place version]**.

- `template<typename t >
CImgList< T > & assign (const CImg< t > &img, const bool is_shared=false)`

Construct list from one image **[in-place version]**.

- `template<typename t1 , typename t2 >
CImgList< T > & assign (const CImg< t1 > &img1, const CImg< t2 > &img2, const bool is_shared=false)`

Construct list from two images **[in-place version]**.

- `template<typename t1 , typename t2 , typename t3 >
CImgList< T > & assign (const CImg< t1 > &img1, const CImg< t2 > &img2, const CImg< t3 > &img3, const bool is_shared=false)`

Construct list from three images **[in-place version]**.

- `template<typename t1 , typename t2 , typename t3 , typename t4 >
CImgList< T > & assign (const CImg< t1 > &img1, const CImg< t2 > &img2, const CImg< t3 > &img3, const CImg< t4 > &img4, const bool is_shared=false)`

Construct list from four images **[in-place version]**.

- `template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 >
CImgList< T > & assign (const CImg< t1 > &img1, const CImg< t2 > &img2, const CImg< t3 > &img3, const CImg< t4 > &img4, const CImg< t5 > &img5, const bool is_shared=false)`

Construct list from five images **[in-place version]**.

- `template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 , typename t6 >
CImgList< T > & assign (const CImg< t1 > &img1, const CImg< t2 > &img2, const CImg< t3 > &img3, const CImg< t4 > &img4, const CImg< t5 > &img5, const CImg< t6 > &img6, const bool is_shared=false)`

Construct list from six images **[in-place version]**.

- `template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 , typename t6 , typename t7 >
CImgList< T > & assign (const CImg< t1 > &img1, const CImg< t2 > &img2, const CImg< t3 > &img3, const CImg< t4 > &img4, const CImg< t5 > &img5, const CImg< t6 > &img6, const CImg< t7 > &img7, const bool is_shared=false)`

Construct list from seven images **[in-place version]**.

- `template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 , typename t6 , typename t7 , typename t8 >
CImgList< T > & assign (const CImg< t1 > &img1, const CImg< t2 > &img2, const CImg< t3 > &img3, const CImg< t4 > &img4, const CImg< t5 > &img5, const CImg< t6 > &img6, const CImg< t7 > &img7, const CImg< t8 > &img8, const bool is_shared=false)`

Construct list from eight images **[in-place version]**.

- `template<typename t >
CImgList< T > & assign (const CImgList< t > &list, const bool is_shared=false)`

Construct list as a copy of an existing list and force the shared state of the list elements **[in-place version]**.

- `CImgList< T > & assign (const CImgList< T > &list, const bool is_shared=false)`

Construct list as a copy of an existing list and force shared state of elements **[in-place version] [specialization]**.

- `CImgList< T > & assign (const char *const filename)`

Construct list by reading the content of a file **[in-place version]**.

- `CImgList< T > & assign (const CImgDisplay &disp)`

Construct list from the content of a display window **[in-place version]**.

- `template<typename t >
CImgList< t > & move_to (CImgList< t > &list)`

Transfers the content of the list instance to another list.

- `template<typename t >
CImgList< t > & move_to (CImgList< t > &list, const unsigned int pos)`

- Transfers the content of the list instance at a specified position in another list.*

• `CImgList< T > & swap (CImgList< T > &list)`

Swap all fields between two list instances.
- `static CImgList< T > & empty ()`

Return a reference to an empty list.
- `static const CImgList< T > & const_empty ()`

Return a reference to an empty list [const version].

Overloaded Operators

- `CImg< T > & operator() (const unsigned int pos)`

Return a reference to one image element of the list.
- `const CImg< T > & operator() (const unsigned int pos) const`

Return a reference to one image of the list.
- `T & operator() (const unsigned int pos, const unsigned int x, const unsigned int y=0, const unsigned int z=0, const unsigned int c=0)`

Return a reference to one pixel value of one image of the list.
- `const T & operator() (const unsigned int pos, const unsigned int x, const unsigned int y=0, const unsigned int z=0, const unsigned int c=0) const`

Return a reference to one pixel value of one image of the list [const version].
- `operator CImg< T > * ()`

Return pointer to the first image of the list.
- `operator const CImg< T > * () const`

Return pointer to the first image of the list [const version].
- `template<typename t >`
`CImgList< T > & operator= (const CImg< t > &img)`
Construct list from one image [in-place version].
- `template<typename t >`
`CImgList< T > & operator= (const CImgList< t > &list)`
Construct list from another list.
- `CImgList< T > & operator= (const CImgList< T > &list)`

Construct list from another list [specialization].
- `CImgList< T > & operator= (const char *const filename)`

Construct list by reading the content of a file [in-place version].
- `CImgList< T > & operator= (const CImgDisplay &disp)`

Construct list from the content of a display window [in-place version].
- `CImgList< T > operator+ () const`

Return a non-shared copy of a list.
- `template<typename t >`
`CImgList< T > & operator, (const CImg< t > &img)`
Return a copy of the list instance, where image `img` has been inserted at the end.
- `template<typename t >`
`CImgList< T > operator, (const CImg< t > &img) const`
Return a copy of the list instance, where image `img` has been inserted at the end [const version].
- `template<typename t >`
`CImgList< T > & operator, (const CImgList< t > &list)`
Return a copy of the list instance, where all elements of input list `list` have been inserted at the end.
- `template<typename t >`
`CImgList< T > & operator, (const CImgList< t > &list) const`
Return a copy of the list instance, where all elements of input `list` have been inserted at the end [const version].
- `CImg< T > operator> (const char axis) const`

Return the image corresponding to the concatenation of all images of the instance list along specified axis.
- `CImgList< T > operator< (const char axis) const`

Return the list corresponding to the splitting of all images of the instance list along specified axis.

Instance Characteristics

- `int width () const`
Return the size of the list, i.e. the number of images contained in it.
- `unsigned int size () const`
Return the size of the list, i.e. the number of images contained in it.
- `Clmg< T > * data ()`
Return pointer to the first image of the list.
- `const Clmg< T > * data () const`
Return pointer to the first image of the list **[const version]**.
- `Clmg< T > * data (const unsigned int pos)`
Return pointer to the pos-th image of the list.
- `iterator begin ()`
Return iterator to the first image of the list.
- `const_iterator begin () const`
Return iterator to the first image of the list **[const version]**.
- `iterator end ()`
Return iterator to one position after the last image of the list.
- `const_iterator end () const`
Return iterator to one position after the last image of the list **[const version]**.
- `Clmg< T > & front ()`
Return reference to the first image of the list.
- `const Clmg< T > & front () const`
Return reference to the first image of the list **[const version]**.
- `const Clmg< T > & back () const`
Return a reference to the last image of the list.
- `Clmg< T > & back ()`
Return a reference to the last image of the list **[const version]**.
- `Clmg< T > & at (const int pos)`
Return pos-th image of the list.
- `T & atNXYZC (const int pos, const int x, const int y, const int z, const int c, const T &out_value)`
Access to pixel value with Dirichlet boundary conditions.
- `T atNXYZC (const int pos, const int x, const int y, const int z, const int c, const T &out_value) const`
Access to pixel value with Dirichlet boundary conditions **[const version]**.
- `T & atNXYZC (const int pos, const int x, const int y, const int z, const int c)`
Access to pixel value with Neumann boundary conditions.
- `T atNXYZC (const int pos, const int x, const int y, const int z, const int c) const`
Access to pixel value with Neumann boundary conditions **[const version]**.
- `T & atNXYZ (const int pos, const int x, const int y, const int z, const int c, const T &out_value)`
Access pixel value with Dirichlet boundary conditions for the 3 coordinates (pos, x,y,z).
- `T atNXYZ (const int pos, const int x, const int y, const int z, const int c, const T &out_value) const`
Access pixel value with Dirichlet boundary conditions for the 3 coordinates (pos, x,y,z) **[const version]**.
- `T & atNXYZ (const int pos, const int x, const int y, const int z, const int c=0)`
Access to pixel value with Neumann boundary conditions for the 4 coordinates (pos, x,y,z).
- `T atNXYZ (const int pos, const int x, const int y, const int z, const int c=0) const`
Access to pixel value with Neumann boundary conditions for the 4 coordinates (pos, x,y,z) **[const version]**.
- `T & atNXY (const int pos, const int x, const int y, const int z, const int c, const T &out_value)`
Access to pixel value with Dirichlet boundary conditions for the 3 coordinates (pos, x,y).
- `T atNXY (const int pos, const int x, const int y, const int z, const int c, const T &out_value) const`
Access to pixel value with Dirichlet boundary conditions for the 3 coordinates (pos, x,y) **[const version]**.
- `T & atNXY (const int pos, const int x, const int y, const int z=0, const int c=0)`

- Access to pixel value with Neumann boundary conditions for the 3 coordinates (pos, x, y).*

 - **T atNXY** (const int pos, const int x, const int y, const int z=0, const int c=0) const

Access to pixel value with Neumann boundary conditions for the 3 coordinates (pos, x, y) [const version].

- **T & atNX** (const int pos, const int x, const int y, const int z, const int c, const T &out_value)

Access to pixel value with Dirichlet boundary conditions for the 2 coordinates (pos, x).

- **T atNX** (const int pos, const int x, const int y, const int z, const int c, const T &out_value) const

Access to pixel value with Dirichlet boundary conditions for the 2 coordinates (pos, x) [const version].

- **T & atNX** (const int pos, const int x, const int y=0, const int z=0, const int c=0)

Access to pixel value with Neumann boundary conditions for the 2 coordinates (pos, x).

- **T atNX** (const int pos, const int x, const int y=0, const int z=0, const int c=0) const

Access to pixel value with Neumann boundary conditions for the 2 coordinates (pos, x) [const version].

- **T & atN** (const int pos, const int x, const int y, const int z, const int c, const T &out_value)

Access to pixel value with Dirichlet boundary conditions for the coordinate (pos).

- **T atN** (const int pos, const int x, const int y, const int z, const int c, const T &out_value) const

Access to pixel value with Dirichlet boundary conditions for the coordinate (pos) [const version].

- **T & atN** (const int pos, const int x=0, const int y=0, const int z=0, const int c=0)

Return pixel value with Neumann boundary conditions for the coordinate (pos).

- **T atN** (const int pos, const int x=0, const int y=0, const int z=0, const int c=0) const

Return pixel value with Neumann boundary conditions for the coordinate (pos) [const version].

- **static const char * pixel_type** ()

Return the type of image pixel values as a C string.

Instance Checking

- **bool is_empty** () const

Return true if list is empty.

- **bool is_sameN** (const unsigned int size_n) const

Test if number of image elements is equal to specified value.

- **template<typename t >**
bool is_sameN (const CImgList< t > &list) const

Test if number of image elements is equal between two images lists.

- **bool is_sameXYZC** (const unsigned int dx, const unsigned int dy, const unsigned int dz, const unsigned int dc) const

Test if dimensions of each image of the list match specified arguments.

- **bool is_sameNXYZC** (const unsigned int n, const unsigned int dx, const unsigned int dy, const unsigned int dz, const unsigned int dc) const

Test if list dimensions match specified arguments.

- **bool containsNXYZC** (const int n, const int x=0, const int y=0, const int z=0, const int c=0) const

Test if list contains one particular pixel location.

- **bool containsN** (const int n) const

Test if list contains image with specified index.

- **template<typename t >**
bool contains (const T &pixel, t &n, t &x, t &y, t &z, t &c) const

Test if one image of the list contains the specified referenced value.

- **template<typename t >**
bool contains (const T &pixel, t &n, t &x, t &y, t &z) const

Test if one image of the list contains the specified referenced value.

- **template<typename t >**
bool contains (const T &pixel, t &n, t &x, t &y) const

Test if one image of the list contains the specified referenced value.

- `template<typename t >`
`bool contains (const T &pixel, t &n, t &x) const`
Test if one image of the list contains the specified referenced value.
- `template<typename t >`
`bool contains (const T &pixel, t &n) const`
Test if one image of the list contains the specified referenced value.
- `bool contains (const T &pixel) const`
Test if one image of the list contains the specified referenced value.
- `template<typename t >`
`bool contains (const CImg< T > &img, t &n) const`
Test if the list contains the image 'img'.
- `bool contains (const CImg< T > &img) const`
Test if the list contains the image img.

Mathematical Functions

- `T & min ()`
Return a reference to the minimum pixel value of the instance list.
- `const T & min () const`
Return a reference to the minimum pixel value of the instance list [const version].
- `T & max ()`
Return a reference to the maximum pixel value of the instance list.
- `const T & max () const`
Return a reference to the maximum pixel value of the instance list [const version].
- `template<typename t >`
`T & min_max (t &max_val)`
Return a reference to the minimum pixel value of the instance list and return the maximum value as well.
- `template<typename t >`
`const T & min_max (t &max_val) const`
Return a reference to the minimum pixel value of the instance list and return the maximum value as well [const version].
- `template<typename t >`
`T & max_min (t &min_val)`
Return a reference to the maximum pixel value of the instance list and return the minimum value as well.
- `template<typename t >`
`const T & max_min (t &min_val) const`
Return a reference to the minimum pixel value of the instance list and return the minimum value as well [const version].

List Manipulation

- `template<typename t >`
`CImgList< T > & insert (const CImg< t > &img, const unsigned int pos=~0U, const bool is_shared=false)`
Insert a copy of the image img into the current image list, at position pos.
- `CImgList< T > & insert (const CImg< T > &img, const unsigned int pos=~0U, const bool is_shared=false)`
Insert a copy of the image img into the current image list, at position pos [specialization].
- `template<typename t >`
`CImgList< T > & get_insert (const CImg< t > &img, const unsigned int pos=~0U, const bool is_shared=false)`
`const`
Insert a copy of the image img into the current image list, at position pos [new-instance version].
- `CImgList< T > & insert (const unsigned int n, const unsigned int pos=~0U)`

- Insert n empty images img into the current image list, at position pos .*

 - `ClmgList< T > get_insert (const unsigned int n, const unsigned int pos=~0U) const`

Insert n empty images img into the current image list, at position pos [new-instance version].
- `template<typename t >`
`ClmgList< T > & insert (const unsigned int n, const Clmg< t > &img, const unsigned int pos=~0U, const bool is_shared=false)`

Insert n copies of the image img into the current image list, at position pos .
- `template<typename t >`
`ClmgList< T > get_insert (const unsigned int n, const Clmg< t > &img, const unsigned int pos=~0U, const bool is_shared=false) const`

Insert n copies of the image img into the current image list, at position pos [new-instance version].
- `template<typename t >`
`ClmgList< T > & insert (const ClmgList< t > &list, const unsigned int pos=~0U, const bool is_shared=false)`

Insert a copy of the image list $list$ into the current image list, starting from position pos .
- `template<typename t >`
`ClmgList< T > get_insert (const ClmgList< t > &list, const unsigned int pos=~0U, const bool is_shared=false) const`

Insert a copy of the image list $list$ into the current image list, starting from position pos [new-instance version].
- `template<typename t >`
`ClmgList< T > & insert (const unsigned int n, const ClmgList< t > &list, const unsigned int pos=~0U, const bool is_shared=false)`

Insert n copies of the list $list$ at position pos of the current list.
- `template<typename t >`
`ClmgList< T > get_insert (const unsigned int n, const ClmgList< t > &list, const unsigned int pos=~0U, const bool is_shared=false) const`

Insert n copies of the list $list$ at position pos of the current list [new-instance version].
- `ClmgList< T > & remove (const unsigned int pos1, const unsigned int pos2)`

Remove all images between from indexes.
- `ClmgList< T > get_remove (const unsigned int pos1, const unsigned int pos2) const`

Remove all images between from indexes [new-instance version].
- `ClmgList< T > & remove (const unsigned int pos)`

Remove image at index pos from the image list.
- `ClmgList< T > get_remove (const unsigned int pos) const`

Remove image at index pos from the image list [new-instance version].
- `ClmgList< T > & remove ()`

Remove last image.
- `ClmgList< T > get_remove () const`

Remove last image [new-instance version].
- `ClmgList< T > & reverse ()`

Reverse list order.
- `ClmgList< T > get_reverse () const`

Reverse list order [new-instance version].
- `ClmgList< T > & images (const unsigned int pos0, const unsigned int pos1)`

Return a sublist.
- `ClmgList< T > get_images (const unsigned int pos0, const unsigned int pos1) const`

Return a sublist [new-instance version].
- `ClmgList< T > get_shared_images (const unsigned int pos0, const unsigned int pos1)`

Return a shared sublist.
- `const ClmgList< T > get_shared_images (const unsigned int pos0, const unsigned int pos1) const`

Return a shared sublist [new-instance version].
- `Clmg< T > get_append (const char axis, const float align=0) const`

Return a single image which is the concatenation of all images of the current `ClmgList` instance.

- `CImgList< T > & split (const char axis, const int nb=-1)`
Return a list where each image has been split along the specified axis.
- `CImgList< T > get_split (const char axis, const int nb=-1, const unsigned int max_parts=~0U) const`
Return a list where each image has been split along the specified axis [*new-instance version*].
- `template<typename t >`
`CImgList< T > & push_back (const CImg< t > &img)`
Insert image at the end of the list.
- `template<typename t >`
`CImgList< T > & push_front (const CImg< t > &img)`
Insert image at the front of the list.
- `template<typename t >`
`CImgList< T > & push_back (const CImgList< t > &list)`
Insert list at the end of the current list.
- `template<typename t >`
`CImgList< T > & push_front (const CImgList< t > &list)`
Insert list at the front of the current list.
- `CImgList< T > & pop_back ()`
Remove last image.
- `CImgList< T > & pop_front ()`
Remove first image.
- `CImgList< T > & erase (const iterator iter)`
Remove image pointed by iterator.

Data Input

- `CImg< intT > get_select (CImgDisplay &disp, const bool feature_type=true, const char axis='x', const float align=0, const bool exit_on_anykey=false) const`
Display a simple interactive interface to select images or sublists.
- `CImg< intT > get_select (const char *const title, const bool feature_type=true, const char axis='x', const float align=0, const bool exit_on_anykey=false) const`
Display a simple interactive interface to select images or sublists.
- `CImgList< T > & load (const char *const filename)`
Load a list from a file.
- `CImgList< T > & load_cimg (const char *const filename)`
Load a list from a .cimg file.
- `CImgList< T > & load_cimg (std::FILE *const file)`
Load a list from a .cimg file.
- `CImgList< T > & load_cimg (const char *const filename, const unsigned int n0, const unsigned int n1, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0, const unsigned int x1, const unsigned int y1, const unsigned int z1, const unsigned int c1)`
Load a sublist list from a (non compressed) .cimg file.
- `CImgList< T > & load_parrec (const char *const filename)`
Load a list from a PAR/REC (Philips) file.
- `CImgList< T > & load_yuv (const char *const filename, const unsigned int size_x, const unsigned int size_y, const unsigned int chroma_subsampling=444, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool yuv2rgb=true)`
Load a list from a YUV image sequence file.
- `CImgList< T > & load_yuv (std::FILE *const file, const unsigned int size_x, const unsigned int size_y, const unsigned int chroma_subsampling=444, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool yuv2rgb=true)`
Load a list from an image sequence YUV file [*in-place version*].

- `ClmgList< T > & load_video (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1)`
Load an image from a video file, using OpenCV library.
- `ClmgList< T > & load_ffmpeg_external (const char *const filename)`
Load an image from a video file using the external tool 'ffmpeg'.
- `ClmgList< T > & load_gif_external (const char *const filename)`
Load gif file, using ImageMagick or GraphicsMagick's external tools.
- `ClmgList< T > & load_gzip_external (const char *const filename)`
Load a gzipped list, using external tool 'gunzip'.
- `ClmgList< T > & load_tiff (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, unsigned int *const bits_per_value=0, float *const voxel_size=0, Clmg< charT > *const description=0)`
Load images from a TIFF file.
- `static ClmgList< T > get_load (const char *const filename)`
*Load a list from a file **[new-instance version]**.*
- `static ClmgList< T > get_load_cimg (const char *const filename)`
*Load a list from a .cimg file **[new-instance version]**.*
- `static ClmgList< T > get_load_cimg (std::FILE *const file)`
*Load a list from a .cimg file **[new-instance version]**.*
- `static ClmgList< T > get_load_cimg (const char *const filename, const unsigned int n0, const unsigned int n1, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0, const unsigned int x1, const unsigned int y1, const unsigned int z1, const unsigned int c1)`
*Load a sublist list from a (non compressed) .cimg file **[new-instance version]**.*
- `static ClmgList< T > get_load_cimg (std::FILE *const file, const unsigned int n0, const unsigned int n1, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0, const unsigned int x1, const unsigned int y1, const unsigned int z1, const unsigned int c1)`
*Load a sub-image list from a (non compressed) .cimg file **[new-instance version]**.*
- `static ClmgList< T > get_load_parrec (const char *const filename)`
*Load a list from a PAR/REC (Philips) file **[new-instance version]**.*
- `static ClmgList< T > get_load_yuv (const char *const filename, const unsigned int size_x, const unsigned int size_y=1, const unsigned int chroma_subsampling=444, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool yuv2rgb=true)`
*Load a list from a YUV image sequence file **[new-instance version]**.*
- `static ClmgList< T > get_load_yuv (std::FILE *const file, const unsigned int size_x, const unsigned int size_y=1, const unsigned int chroma_subsampling=444, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool yuv2rgb=true)`
*Load a list from an image sequence YUV file **[new-instance version]**.*
- `static ClmgList< T > get_load_video (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1)`
*Load an image from a video file, using OpenCV library **[new-instance version]**.*
- `static ClmgList< T > get_load_ffmpeg_external (const char *const filename)`
*Load an image from a video file using the external tool 'ffmpeg' **[new-instance version]**.*
- `static ClmgList< T > get_load_gif_external (const char *const filename)`
*Load gif file, using ImageMagick or GraphicsMagick's external tools **[new-instance version]**.*
- `static ClmgList< T > get_load_gzip_external (const char *const filename)`
*Load a gzipped list, using external tool 'gunzip' **[new-instance version]**.*
- `static ClmgList< T > get_load_tiff (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, unsigned int *const bits_per_value=0, float *const voxel_size=0, Clmg< charT > *const description=0)`
*Load a multi-page TIFF file **[new-instance version]**.*

Data Output

- `const CImgList< T > & print (const char *const title=0, const bool display_stats=true) const`
Print information about the list on the standard output.
- `const CImgList< T > & display (CImgDisplay &disp, const char axis='x', const float align=0) const`
Display the current CImgList instance in an existing CImgDisplay window (by reference).
- `const CImgList< T > & display (CImgDisplay &disp, const bool display_info, const char axis='x', const float align=0, unsigned int *const XYZ=0, const bool exit_on_anykey=false) const`
Display the current CImgList instance in a new display window.
- `const CImgList< T > & display (const char *const title=0, const bool display_info=true, const char axis='x', const float align=0, unsigned int *const XYZ=0, const bool exit_on_anykey=false) const`
Display the current CImgList instance in a new display window.
- `const CImgList< T > & save (const char *const filename, const int number=-1, const unsigned int digits=6) const`
Save the list into a file.
- `const CImgList< T > & save_gif_external (const char *const filename, const float fps=25, const unsigned int nb_loops=0)`
Save the image sequence as a GIF animated file.
- `const CImgList< T > & save_yuv (const char *const filename=0, const unsigned int chroma_subsampling=444, const bool is_rgb=true) const`
Save the list as a YUV image sequence file.
- `const CImgList< T > & save_yuv (std::FILE *const file, const unsigned int chroma_subsampling=444, const bool is_rgb=true) const`
Save the image sequence into a YUV file.
- `const CImgList< T > & save_cimg (const char *const filename, const bool is_compressed=false) const`
Save the list as a .cimg file.
- `const CImgList< T > & save_cimg (std::FILE *file, const bool is_compressed=false) const`
Save the list as a .cimg file.
- `const CImgList< T > & save_cimg (const char *const filename, const unsigned int n0, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0) const`
Insert the image list instance into an existing .cimg file, at specified coordinates.
- `const CImgList< T > & save_cimg (std::FILE *const file, const unsigned int n0, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0) const`
Insert the image list instance into an existing .cimg file, at specified coordinates.
- `const CImgList< T > & save_tiff (const char *const filename, const unsigned int compression_type=0, const float *const voxel_size=0, const char *const description=0, const bool use_bigtiff=true) const`
Save the list as a TIFF file.
- `const CImgList< T > & save_gzip_external (const char *const filename) const`
Save the list as a gzipped file using the external tool 'gzip'.
- `const CImgList< T > & save_video (const char *const filename, const unsigned int fps=25, const char *const codec=0, const bool keep_open=false) const`
Save the image sequence (using the OpenCV library when available).
- `const CImgList< T > & save_ffmpeg_external (const char *const filename, const unsigned int fps=25, const char *const codec=0, const unsigned int bitrate=2048) const`
Save the image sequence, using the external tool 'ffmpeg'.
- `CImg< ucharT > get_serialize (const bool is_compressed=false, const unsigned int header_size=0) const`
Serialize a CImgList< T > instance into a raw CImg< unsigned char > buffer.
- `static bool is_saveable (const char *const filename)`
Indicates whether an image list can be saved as one single file or not.
- `static void save_empty_cimg (const char *const filename, const unsigned int nb, const unsigned int dx, const unsigned int dy=1, const unsigned int dz=1, const unsigned int dc=1)`
Save an empty (non-compressed) .cimg file with specified dimensions.

- `static void save_empty_cimg (std::FILE *const file, const unsigned int nb, const unsigned int dx, const unsigned int dy=1, const unsigned int dz=1, const unsigned int dc=1)`
Save an empty `.cimg` file with specified dimensions.
- `template<typename t >`
`static CImgList< T > get_unserialize (const CImg< t > &buffer, const unsigned int header_size=0)`
Unserialize a `CImg<unsigned char>` serialized buffer into a `CImgList< T>` list.

Others

- `CImgList< T > & FFT (const char axis, const bool invert=false)`
Compute a 1D Fast Fourier Transform, along specified axis.
- `CImgList< Tfloat > get_FFT (const char axis, const bool invert=false) const`
Compute a 1-D Fast Fourier Transform, along specified axis *[new-instance version]*.
- `CImgList< T > & FFT (const bool invert=false)`
Compute n-D Fast Fourier Transform.
- `CImgList< Tfloat > get_FFT (const bool invert=false) const`
Compute n-D Fast Fourier Transform *[new-instance version]*.
- `CImgList< T > & reverse_object3d ()`
Reverse the orientations of a 3D object's primitives.
- `CImgList< T > get_reverse_object3d () const`
Reverse the orientations of a 3D object's primitives *[new-instance version]*.
- `static const CImgList< ucharT > & font (const unsigned int requested_height, const bool is_variable_width=true)`
Return a predefined `CImg` font with the requested height.

8.4.1 Detailed Description

```
template<typename T>
struct cimg_library::CImgList< T >
```

Represents a list of images `CImg<T>`.

8.4.2 Member Typedef Documentation

8.4.2.1 iterator

```
template<typename T >
typedef CImg<T>* iterator
```

Simple iterator type to loop through each image of a list.

Note

- The `CImgList<T>::iterator` type is defined as a `CImg<T>*`.
- You may use it like this:

```
CImgList<> list; // Assuming this image list is not empty
for (CImgList<>::iterator it = list.begin(); it<list.end(); ++it) (*it).mirror('x');
```
- Using the loop macro `cimglist_for` is another (more concise) alternative:

```
cimglist_for(list,1) list[1].mirror('x');
```


8.4.2.2 const_iterator

```
template<typename T >
typedef const CImg<T>* const_iterator
```

Simple const iterator type, to loop through each image of a `const` list instance.

Note

- The `CImgList<T>::const_iterator` type is defined to be a `const CImg<T>*`.
- Similar to `CImgList<T>::iterator`, but for constant list instances.

8.4.2.3 value_type

```
template<typename T >
typedef T value_type
```

Pixel value type.

Refer to the pixels value type of the images in the list.

Note

- The `CImgList<T>::value_type` type of a `CImgList<T>` is defined to be a `T`. It is then similar to `CImg<T>::value_type`.
- `CImgList<T>::value_type` is actually not used in `CImg` functions. It has been mainly defined for compatibility with STL naming conventions.

8.4.3 Constructor & Destructor Documentation

8.4.3.1 ~CImgList()

```
template<typename T >
~CImgList ( )
```

Destructor.

Destroy current list instance.

Note

- Any allocated buffer is deallocated.
- Destroying an empty list does nothing actually.

8.4.3.2 CImgList() [1/17]

```
template<typename T >
CImgList ( )
```

Default constructor.

Construct a new empty list instance.

Note

- An empty list has no pixel data and its dimension `width()` is set to 0, as well as its image buffer pointer `data()`.
- An empty list may be reassigned afterwards, with the family of the `assign()` functions. In all cases, the type of pixels stays `T`.

8.4.3.3 CImgList() [2/17]

```
template<typename T >
CImgList (
    const unsigned int n ) [explicit]
```

Construct list containing empty images.

Parameters

<i>n</i>	Number of empty images.
----------	-------------------------

Note

Useful when you know in advance the number of images you want to manage, as it will allocate the right amount of memory for the list, without needs for reallocation (that may occur when starting from an empty list and inserting several images in it).

8.4.3.4 CImgList() [3/17]

```
template<typename T >
CImgList (
    const unsigned int n,
    const unsigned int width,
    const unsigned int height = 1,
    const unsigned int depth = 1,
    const unsigned int spectrum = 1 )
```

Construct list containing images of specified size.

Parameters

<i>n</i>	Number of images.
<i>width</i>	Width of images.
<i>height</i>	Height of images.
<i>depth</i>	Depth of images.
<i>spectrum</i>	Number of channels of images.

Note

Pixel values are not initialized and may probably contain garbage.

8.4.3.5 CImgList() [4/17]

```
template<typename T >
CImgList (
    const unsigned int n,
    const unsigned int width,
    const unsigned int height,
    const unsigned int depth,
    const unsigned int spectrum,
    const T & val )
```

Construct list containing images of specified size, and initialize pixel values.

Parameters

<i>n</i>	Number of images.
<i>width</i>	Width of images.
<i>height</i>	Height of images.
<i>depth</i>	Depth of images.
<i>spectrum</i>	Number of channels of images.
<i>val</i>	Initialization value for images pixels.

8.4.3.6 CImgList() [5/17]

```
template<typename T >
template<typename t >
CImgList (
    const unsigned int n,
    const CImg< t > & img,
    const bool is_shared = false )
```

Construct list containing copies of an input image.

Parameters

<i>n</i>	Number of images.
<i>img</i>	Input image to copy in the constructed list.
<i>is_shared</i>	Indicates whether the elements of the list are shared or non-shared copies of <i>img</i> .

8.4.3.7 CImgList() [6/17]

```
template<typename T >
template<typename t >
CImgList (
```

```
const CImg< t > & img,
const bool is_shared = false ) [explicit]
```

Construct list from one image.

Parameters

<i>img</i>	Input image to copy in the constructed list.
<i>is_shared</i>	Indicates whether the element of the list is a shared or non-shared copy of <i>img</i> .

8.4.3.8 CImgList() [7/17]

```
template<typename T >
template<typename t1 , typename t2 >
CImgList (
    const CImg< t1 > & img1,
    const CImg< t2 > & img2,
    const bool is_shared = false )
```

Construct list from two images.

Parameters

<i>img1</i>	First input image to copy in the constructed list.
<i>img2</i>	Second input image to copy in the constructed list.
<i>is_shared</i>	Indicates whether the elements of the list are shared or non-shared copies of input images.

8.4.3.9 CImgList() [8/17]

```
template<typename T >
template<typename t1 , typename t2 , typename t3 >
CImgList (
    const CImg< t1 > & img1,
    const CImg< t2 > & img2,
    const CImg< t3 > & img3,
    const bool is_shared = false )
```

Construct list from three images.

Parameters

<i>img1</i>	First input image to copy in the constructed list.
<i>img2</i>	Second input image to copy in the constructed list.
<i>img3</i>	Third input image to copy in the constructed list.
<i>is_shared</i>	Indicates whether the elements of the list are shared or non-shared copies of input images.

8.4.3.10 CImgList() [9/17]

```
template<typename T >
template<typename t1 , typename t2 , typename t3 , typename t4 >
CImgList (
    const CImg< t1 > & img1,
    const CImg< t2 > & img2,
    const CImg< t3 > & img3,
    const CImg< t4 > & img4,
    const bool is_shared = false )
```

Construct list from four images.

Parameters

<i>img1</i>	First input image to copy in the constructed list.
<i>img2</i>	Second input image to copy in the constructed list.
<i>img3</i>	Third input image to copy in the constructed list.
<i>img4</i>	Fourth input image to copy in the constructed list.
<i>is_shared</i>	Indicates whether the elements of the list are shared or non-shared copies of input images.

8.4.3.11 CImgList() [10/17]

```
template<typename T >
template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 >
CImgList (
    const CImg< t1 > & img1,
    const CImg< t2 > & img2,
    const CImg< t3 > & img3,
    const CImg< t4 > & img4,
    const CImg< t5 > & img5,
    const bool is_shared = false )
```

Construct list from five images.

Parameters

<i>img1</i>	First input image to copy in the constructed list.
<i>img2</i>	Second input image to copy in the constructed list.
<i>img3</i>	Third input image to copy in the constructed list.
<i>img4</i>	Fourth input image to copy in the constructed list.
<i>img5</i>	Fifth input image to copy in the constructed list.
<i>is_shared</i>	Indicates whether the elements of the list are shared or non-shared copies of input images.

8.4.3.12 CImgList() [11/17]

```
template<typename T >
template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 , typename t6 >
CImgList (
```

```

const CImg< t1 > & img1,
const CImg< t2 > & img2,
const CImg< t3 > & img3,
const CImg< t4 > & img4,
const CImg< t5 > & img5,
const CImg< t6 > & img6,
const bool is_shared = false )

```

Construct list from six images.

Parameters

<i>img1</i>	First input image to copy in the constructed list.
<i>img2</i>	Second input image to copy in the constructed list.
<i>img3</i>	Third input image to copy in the constructed list.
<i>img4</i>	Fourth input image to copy in the constructed list.
<i>img5</i>	Fifth input image to copy in the constructed list.
<i>img6</i>	Sixth input image to copy in the constructed list.
<i>is_shared</i>	Indicates whether the elements of the list are shared or non-shared copies of input images.

8.4.3.13 CImgList() [12/17]

```

template<typename T >
template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 , typename t6 ,
typename t7 >
CImgList (
    const CImg< t1 > & img1,
    const CImg< t2 > & img2,
    const CImg< t3 > & img3,
    const CImg< t4 > & img4,
    const CImg< t5 > & img5,
    const CImg< t6 > & img6,
    const CImg< t7 > & img7,
    const bool is_shared = false )

```

Construct list from seven images.

Parameters

<i>img1</i>	First input image to copy in the constructed list.
<i>img2</i>	Second input image to copy in the constructed list.
<i>img3</i>	Third input image to copy in the constructed list.
<i>img4</i>	Fourth input image to copy in the constructed list.
<i>img5</i>	Fifth input image to copy in the constructed list.
<i>img6</i>	Sixth input image to copy in the constructed list.
<i>img7</i>	Seventh input image to copy in the constructed list.
<i>is_shared</i>	Indicates whether the elements of the list are shared or non-shared copies of input images.

8.4.3.14 CImgList() [13/17]

```

template<typename T >
template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 , typename t6 ,
typename t7 , typename t8 >
CImgList (
    const CImg< t1 > & img1,
    const CImg< t2 > & img2,
    const CImg< t3 > & img3,
    const CImg< t4 > & img4,
    const CImg< t5 > & img5,
    const CImg< t6 > & img6,
    const CImg< t7 > & img7,
    const CImg< t8 > & img8,
    const bool is_shared = false )

```

Construct list from eight images.

Parameters

<i>img1</i>	First input image to copy in the constructed list.
<i>img2</i>	Second input image to copy in the constructed list.
<i>img3</i>	Third input image to copy in the constructed list.
<i>img4</i>	Fourth input image to copy in the constructed list.
<i>img5</i>	Fifth input image to copy in the constructed list.
<i>img6</i>	Sixth input image to copy in the constructed list.
<i>img7</i>	Seventh input image to copy in the constructed list.
<i>img8</i>	Eighth input image to copy in the constructed list.
<i>is_shared</i>	Indicates whether the elements of the list are shared or non-shared copies of input images.

8.4.3.15 CImgList() [14/17]

```

template<typename T >
template<typename t >
CImgList (
    const CImgList< t > & list )

```

Construct list copy.

Parameters

<i>list</i>	Input list to copy.
-------------	---------------------

Note

The shared state of each element of the constructed list is kept the same as in `list`.

8.4.3.16 CImgList() [15/17]

```

template<typename T >
template<typename t >

```

```
CImgList (
    const CImgList< t > & list,
    const bool is_shared )
```

Construct list copy, and force the shared state of the list elements.

Parameters

<i>list</i>	Input list to copy.
<i>is_shared</i>	Indicates whether the elements of the list are shared or non-shared copies of input images.

8.4.3.17 CImgList() [16/17]

```
template<typename T >
CImgList (
    const char *const filename ) [explicit]
```

Construct list by reading the content of a file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.4.3.18 CImgList() [17/17]

```
template<typename T >
CImgList (
    const CImgDisplay & disp ) [explicit]
```

Construct list from the content of a display window.

Parameters

<i>disp</i>	Display window to get content from.
-------------	-------------------------------------

Note

Constructed list contains a single image only.

8.4.4 Member Function Documentation

8.4.4.1 get_shared()

```
template<typename T >
CImgList< T > get_shared ( )
```

Return a list with elements being shared copies of images in the list instance.

Note

`list2 = list1.get_shared()` is equivalent to `list2.assign(list1,true)`.

8.4.4.2 assign() [1/16]

```
template<typename T >
CImgList< T > & assign ( )
```

Destructor **[in-place version]**.

See also

[CImgList\(\)](#).

8.4.4.3 clear()

```
template<typename T >
CImgList< T > & clear ( )
```

Destructor **[in-place version]**.

Equivalent to [assign\(\)](#).

Note

Only here for compatibility with STL naming conventions.

8.4.4.4 assign() [2/16]

```
template<typename T >
CImgList< T > & assign (
    const unsigned int n )
```

Construct list containing empty images **[in-place version]**.

See also

[CImgList\(unsigned int\)](#).

8.4.4.5 assign() [3/16]

```
template<typename T >
CImgList< T > & assign (
    const unsigned int n,
    const unsigned int width,
    const unsigned int height = 1,
    const unsigned int depth = 1,
    const unsigned int spectrum = 1 )
```

Construct list containing images of specified size **[in-place version]**.

See also

[CImgList\(unsigned int, unsigned int, unsigned int, unsigned int, unsigned int\)](#).

8.4.4.6 assign() [4/16]

```
template<typename T >
CImgList< T > & assign (
    const unsigned int n,
    const unsigned int width,
    const unsigned int height,
    const unsigned int depth,
    const unsigned int spectrum,
    const T & val )
```

Construct list containing images of specified size, and initialize pixel values **[in-place version]**.

See also

[CImgList\(unsigned int, unsigned int, unsigned int, unsigned int, unsigned int, const T\).](#)

8.4.4.7 assign() [5/16]

```
template<typename T >
template<typename t >
CImgList< T > & assign (
    const unsigned int n,
    const CImg< t > & img,
    const bool is_shared = false )
```

Construct list containing copies of an input image **[in-place version]**.

See also

[CImgList\(unsigned int, const CImg<t>&, bool\).](#)

8.4.4.8 assign() [6/16]

```
template<typename T >
template<typename t >
CImgList< T > & assign (
    const CImg< t > & img,
    const bool is_shared = false )
```

Construct list from one image **[in-place version]**.

See also

[CImgList\(const CImg<t>&, bool\).](#)

8.4.4.9 assign() [7/16]

```
template<typename T >
template<typename t1 , typename t2 >
CImgList< T > & assign (
    const CImg< t1 > & img1,
    const CImg< t2 > & img2,
    const bool is_shared = false )
```

Construct list from two images **[in-place version]**.

See also

[CImgList\(const CImg<t>&, const CImg<t>&, bool\).](#)

8.4.4.10 assign() [8/16]

```
template<typename T >
template<typename t1 , typename t2 , typename t3 >
CImgList< T > & assign (
    const CImg< t1 > & img1,
    const CImg< t2 > & img2,
    const CImg< t3 > & img3,
    const bool is_shared = false )
```

Construct list from three images **[in-place version]**.

See also

[CImgList\(const CImg<t>&, const CImg<t>&, const CImg<t>&, bool\).](#)

8.4.4.11 assign() [9/16]

```
template<typename T >
template<typename t1 , typename t2 , typename t3 , typename t4 >
CImgList< T > & assign (
    const CImg< t1 > & img1,
    const CImg< t2 > & img2,
    const CImg< t3 > & img3,
    const CImg< t4 > & img4,
    const bool is_shared = false )
```

Construct list from four images **[in-place version]**.

See also

[CImgList\(const CImg<t>&, const CImg<t>&, const CImg<t>&, const CImg<t>&, bool\).](#)

8.4.4.12 assign() [10/16]

```
template<typename T >
template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 >
CImgList< T > & assign (
    const CImg< t1 > & img1,
    const CImg< t2 > & img2,
    const CImg< t3 > & img3,
    const CImg< t4 > & img4,
    const CImg< t5 > & img5,
    const bool is_shared = false )
```

Construct list from five images **[in-place version]**.

See also

[CImgList\(const CImg<t>&, const CImg<t>&, const CImg<t>&, const CImg<t>&, const CImg<t>&, bool\).](#)

8.4.4.13 assign() [11/16]

```
template<typename T >
template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 , typename t6 >
CImgList< T > & assign (
    const CImg< t1 > & img1,
    const CImg< t2 > & img2,
    const CImg< t3 > & img3,
    const CImg< t4 > & img4,
    const CImg< t5 > & img5,
    const CImg< t6 > & img6,
    const bool is_shared = false )
```

Construct list from six images **[in-place version]**.

See also

[CImgList\(const CImg<t>&,const CImg<t>&,const CImg<t>&,const CImg<t>&,const CImg<t>&,const CImg<t>&, bool\).](#)

8.4.4.14 assign() [12/16]

```
template<typename T >
template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 , typename t6 ,
typename t7 >
CImgList< T > & assign (
    const CImg< t1 > & img1,
    const CImg< t2 > & img2,
    const CImg< t3 > & img3,
    const CImg< t4 > & img4,
    const CImg< t5 > & img5,
    const CImg< t6 > & img6,
    const CImg< t7 > & img7,
    const bool is_shared = false )
```

Construct list from seven images **[in-place version]**.

See also

[CImgList\(const CImg<t>&,const CImg<t>&,const CImg<t>&,const CImg<t>&,const CImg<t>&,const CImg<t>&, const C](#)

8.4.4.15 assign() [13/16]

```
template<typename T >
template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 , typename t6 ,
typename t7 , typename t8 >
CImgList< T > & assign (
    const CImg< t1 > & img1,
    const CImg< t2 > & img2,
    const CImg< t3 > & img3,
    const CImg< t4 > & img4,
    const CImg< t5 > & img5,
    const CImg< t6 > & img6,
    const CImg< t7 > & img7,
    const CImg< t8 > & img8,
    const bool is_shared = false )
```

Construct list from eight images **[in-place version]**.

See also

[CImgList\(const CImg<t>&,const CImg<t>&,const CImg<t>&,const CImg<t>&,const CImg<t>&,const CImg<t>&,const C](#)

8.4.4.16 assign() [14/16]

```
template<typename T >
template<typename t >
CImgList< T > & assign (
    const CImgList< t > & list,
    const bool is_shared = false )
```

Construct list as a copy of an existing list and force the shared state of the list elements **[in-place version]**.

See also

[CImgList\(const CImgList<t>&, bool is_shared\).](#)

8.4.4.17 assign() [15/16]

```
template<typename T >
CImgList< T > & assign (
    const char *const filename )
```

Construct list by reading the content of a file **[in-place version]**.

See also

[CImgList\(const char *const\).](#)

8.4.4.18 assign() [16/16]

```
template<typename T >
CImgList< T > & assign (
    const CImgDisplay & disp )
```

Construct list from the content of a display window **[in-place version]**.

See also

[CImgList\(const CImgDisplay&\)](#).

8.4.4.19 move_to() [1/2]

```
template<typename T >
template<typename t >
CImgList< t > & move_to (
    CImgList< t > & list )
```

Transfers the content of the list instance to another list.

Parameters

<i>list</i>	Destination list.
-------------	-------------------

Note

When returning, the current list instance is empty and the initial content of `list` is destroyed.

8.4.4.20 move_to() [2/2]

```
template<typename T >
template<typename t >
CImgList< t > & move_to (
    CImgList< t > & list,
    const unsigned int pos )
```

Transfers the content of the list instance at a specified position in another list.

Parameters

<i>list</i>	Destination list.
<i>pos</i>	Index of the insertion in the list.

Note

When returning, the list instance is empty and the initial content of `list` is preserved (only images indexes may be modified).

8.4.4.21 swap()

```
template<typename T >
CImgList< T > & swap (
    CImgList< T > & list )
```

Swap all fields between two list instances.

Parameters

<i>list</i>	List to swap fields with.
-------------	---------------------------

Note

Can be used to exchange the content of two lists in a fast way.

8.4.4.22 empty()

```
template<typename T >
static CImgList< T > & empty ( ) [static]
```

Return a reference to an empty list.

Note

Can be used to define default values in a function taking a CImgList<T> as an argument.

```
void f(const CImgList<char>& list=CImgList<char>::empty());
```

8.4.4.23 operator>() [1/3]

```
template<typename T >
CImg< T > & operator() (
    const unsigned int pos )
```

Return a reference to one image element of the list.

Parameters

<i>pos</i>	Index of the image element.
------------	-----------------------------

8.4.4.24 operator>() [2/3]

```
template<typename T >
const CImg< T > & operator() (
    const unsigned int pos ) const
```

Return a reference to one image of the list.

Parameters

<i>pos</i>	Index of the image element.
------------	-----------------------------

8.4.4.25 operator>() [3/3]

```
template<typename T >
T & operator() (
    const unsigned int pos,
    const unsigned int x,
    const unsigned int y = 0,
    const unsigned int z = 0,
    const unsigned int c = 0 )
```

Return a reference to one pixel value of one image of the list.

Parameters

<i>pos</i>	Index of the image element.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.

Note

`list(n, x, y, z, c)` is equivalent to `list[n](x, y, z, c)`.

8.4.4.26 operator CImg< T > *()

```
template<typename T >
operator CImg< T > * ( )
```

Return pointer to the first image of the list.

Note

Images in a list are stored as a buffer of `CImg<T>`.

8.4.4.27 operator=() [1/4]

```
template<typename T >
template<typename t >
CImgList< T > & operator= (
    const CImg< t > & img )
```

Construct list from one image **[in-place version]**.

Parameters

<i>img</i>	Input image to copy in the constructed list.
------------	--

Note

`list = img;` is equivalent to `list.assign(img);`.

8.4.4.28 operator=() [2/4]

```
template<typename T >
template<typename t >
CImgList< T > & operator= (
    const CImgList< t > & list )
```

Construct list from another list.

Parameters

<i>list</i>	Input list to copy.
-------------	---------------------

Note

`list1 = list2` is equivalent to `list1.assign(list2);`.

8.4.4.29 operator=() [3/4]

```
template<typename T >
CImgList< T > & operator= (
    const char *const filename )
```

Construct list by reading the content of a file **[in-place version]**.

See also

[CImgList\(const char *const\).](#)

8.4.4.30 operator=() [4/4]

```
template<typename T >
CImgList< T > & operator= (
    const CImgDisplay & disp )
```

Construct list from the content of a display window **[in-place version]**.

See also

[CImgList\(const CImgDisplay&\).](#)

8.4.4.31 operator+()

```
template<typename T >
CImgList< T > operator+ ( ) const
```

Return a non-shared copy of a list.

Note

`+list` is equivalent to `CImgList<T>(list, false)`. It forces the copy to have non-shared elements.

8.4.4.32 operator,() [1/2]

```
template<typename T >
template<typename t >
CImgList< T > & operator, (
    const CImg< t > & img )
```

Return a copy of the list instance, where image `img` has been inserted at the end.

Parameters

<i>img</i>	Image inserted at the end of the instance copy.
------------	---

Note

Define a convenient way to create temporary lists of images, as in the following code:

```
(img1,img2,img3,img4).display("My four images");
```

8.4.4.33 operator,() [2/2]

```
template<typename T >
template<typename t >
CImgList< T > & operator, (
    const CImgList< t > & list )
```

Return a copy of the list instance, where all elements of input list `list` have been inserted at the end.

Parameters

<i>list</i>	List inserted at the end of the instance copy.
-------------	--

8.4.4.34 operator>()

```
template<typename T >
CImg< T > operator> (
    const char axis ) const
```

Return the image corresponding to the concatenation of all images of the instance list along specified axis.

Parameters

<i>axis</i>	Concatenation axis. Can be { 'x' 'y' 'z' 'c' }.
-------------	---

Note

`list>'x'` is equivalent to `list.get_append('x')`.

8.4.4.35 operator<()>

```
template<typename T >
CImgList< T > operator< (
    const char axis ) const
```

Return the list corresponding to the splitting of all images of the instance list along specified axis.

Parameters

<i>axis</i>	Axis used for image splitting.
-------------	--------------------------------

Note

`list<'x'` is equivalent to `list.get_split('x')`.

8.4.4.36 pixel_type()

```
template<typename T >
static const char * pixel_type ( ) [static]
```

Return the type of image pixel values as a C string.

Return a `char*` string containing the usual type name of the image pixel values (i.e. a stringified version of the template parameter `T`).

Note

- The returned string does not contain any spaces.
- If the pixel type `T` does not correspond to a registered type, the string "unknown" is returned.

8.4.4.37 width()

```
template<typename T >
int width ( ) const
```

Return the size of the list, i.e. the number of images contained in it.

Note

Similar to [size\(\)](#) but returns the result as a (signed) integer.

8.4.4.38 size()

```
template<typename T >
unsigned int size ( ) const
```

Return the size of the list, i.e. the number of images contained in it.

Note

Similar to [width\(\)](#) but returns the result as an unsigned integer.

8.4.4.39 data() [1/2]

```
template<typename T >
CImg< T > * data ( )
```

Return pointer to the first image of the list.

Note

Images in a list are stored as a buffer of `CImg<T>`.

8.4.4.40 data() [2/2]

```
template<typename T >
CImg< T > * data (
    const unsigned int pos )
```

Return pointer to the pos-th image of the list.

Parameters

<i>pos</i>	Index of the image element to access.
------------	---------------------------------------

Note

`list.data(n) ;` is equivalent to `list.data + n ;`.

8.4.4.41 at()

```
template<typename T >
CImg< T > & at (
    const int pos )
```

Return pos-th image of the list.

Parameters

<i>pos</i>	Index of the image element to access.
------------	---------------------------------------

8.4.4.42 atNXYZC() [1/2]

```
template<typename T >
T & atNXYZC (
    const int pos,
    const int x,
    const int y,
    const int z,
    const int c,
    const T & out_value )
```

Access to pixel value with Dirichlet boundary conditions.

Parameters

<i>pos</i>	Index of the image element to access.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.
<i>out_value</i>	Default value returned if <i>offset</i> is outside image bounds.

Note

`list.atNXYZC(p, x, y, z, c) ;` is equivalent to `list[p].atXYZC(x, y, z, c) ;`.

8.4.4.43 atNXYZC() [2/2]

```
template<typename T >
T & atNXYZC (
    const int pos,
    const int x,
    const int y,
    const int z,
    const int c )
```

Access to pixel value with Neumann boundary conditions.

Parameters

<i>pos</i>	Index of the image element to access.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.

Note

`list.atNXYZC(p, x, y, z, c) ;` is equivalent to `list[p].atXYZC(x, y, z, c) ;`.

8.4.4.44 atNXYZ() [1/2]

```
template<typename T >
T & atNXYZ (
    const int pos,
    const int x,
    const int y,
    const int z,
    const int c,
    const T & out_value )
```

Access pixel value with Dirichlet boundary conditions for the 3 coordinates (pos, x,y,z).

Parameters

<i>pos</i>	Index of the image element to access.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.
<i>out_value</i>	Default value returned if <i>offset</i> is outside image bounds.

Note

`list.atNXYZ(p, x, y, z, c) ;` is equivalent to `list[p].atXYZ(x, y, z, c) ;`.

8.4.4.45 atNXYZ() [2/2]

```
template<typename T >
T & atNXYZ (
    const int pos,
    const int x,
    const int y,
    const int z,
    const int c = 0 )
```

Access to pixel value with Neumann boundary conditions for the 4 coordinates (pos, x,y,z).

Parameters

<i>pos</i>	Index of the image element to access.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.

Note

`list.atNXYZ (p, x, y, z, c) ;` is equivalent to `list [p] .atXYZ (x, y, z, c) ;`.

8.4.4.46 atNXY() [1/2]

```
template<typename T >
T & atNXY (
    const int pos,
    const int x,
    const int y,
    const int z,
    const int c,
    const T & out_value )
```

Access to pixel value with Dirichlet boundary conditions for the 3 coordinates (pos, x,y).

Parameters

<i>pos</i>	Index of the image element to access.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.
<i>out_value</i>	Default value returned if <i>offset</i> is outside image bounds.

Note

`list.atNXYZ (p, x, y, z, c) ;` is equivalent to `list [p] .atXYZ (x, y, z, c) ;`.

8.4.4.47 atNXY() [2/2]

```
template<typename T >
T & atNXY (
    const int pos,
    const int x,
    const int y,
    const int z = 0,
    const int c = 0 )
```

Access to pixel value with Neumann boundary conditions for the 3 coordinates (pos, x,y).

Parameters

<i>pos</i>	Index of the image element to access.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.

Note

`list.atNXYZ(p, x, y, z, c) ;` is equivalent to `list[p].atXYZ(x, y, z, c) ;`.

8.4.4.48 atNX() [1/2]

```
template<typename T >
T & atNX (
    const int pos,
    const int x,
    const int y,
    const int z,
    const int c,
    const T & out_value )
```

Access to pixel value with Dirichlet boundary conditions for the 2 coordinates (pos,x).

Parameters

<i>pos</i>	Index of the image element to access.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.
<i>out_value</i>	Default value returned if <i>offset</i> is outside image bounds.

Note

`list.atNXYZ(p, x, y, z, c) ;` is equivalent to `list[p].atXYZ(x, y, z, c) ;`.

8.4.4.49 atNX() [2/2]

```
template<typename T >
T & atNX (
    const int pos,
    const int x,
    const int y = 0,
    const int z = 0,
    const int c = 0 )
```

Access to pixel value with Neumann boundary conditions for the 2 coordinates (pos, x).

Parameters

<i>pos</i>	Index of the image element to access.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.

Note

`list.atNXYZ(p, x, y, z, c) ;` is equivalent to `list[p].atXYZ(x, y, z, c) ;`.

8.4.4.50 atN() [1/2]

```
template<typename T >
T & atN (
    const int pos,
    const int x,
    const int y,
    const int z,
    const int c,
    const T & out_value )
```

Access to pixel value with Dirichlet boundary conditions for the coordinate (`pos`).

Parameters

<i>pos</i>	Index of the image element to access.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.
<i>out_value</i>	Default value returned if <code>offset</code> is outside image bounds.

Note

`list.atNXYZ(p, x, y, z, c) ;` is equivalent to `list[p].atXYZ(x, y, z, c) ;`.

8.4.4.51 atN() [2/2]

```
template<typename T >
T & atN (
    const int pos,
    const int x = 0,
    const int y = 0,
    const int z = 0,
    const int c = 0 )
```

Return pixel value with Neumann boundary conditions for the coordinate (`pos`).

Parameters

<i>pos</i>	Index of the image element to access.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.

Note

`list.atNXYZ(p, x, y, z, c) ;` is equivalent to `list[p].atXYZ(x, y, z, c) ;`.

8.4.4.52 is_sameN() [1/2]

```
template<typename T >
bool is_sameN (
    const unsigned int size_n ) const
```

Test if number of image elements is equal to specified value.

Parameters

<i>size_↔ _n</i>	Number of image elements to test.
----------------------	-----------------------------------

8.4.4.53 is_sameN() [2/2]

```
template<typename T >
template<typename t >
bool is_sameN (
    const CImgList< t > & list ) const
```

Test if number of image elements is equal between two images lists.

Parameters

<i>list</i>	Input list to compare with.
-------------	-----------------------------

8.4.4.54 is_sameXYZC()

```
template<typename T >
bool is_sameXYZC (
    const unsigned int dx,
    const unsigned int dy,
    const unsigned int dz,
    const unsigned int dc ) const
```

Test if dimensions of each image of the list match specified arguments.

Parameters

<i>dx</i>	Checked image width.
<i>dy</i>	Checked image height.
<i>dz</i>	Checked image depth.
<i>dc</i>	Checked image spectrum.

8.4.4.55 is_sameNXYZC()

```
template<typename T >
bool is_sameNXYZC (
    const unsigned int n,
    const unsigned int dx,
    const unsigned int dy,
    const unsigned int dz,
    const unsigned int dc ) const
```

Test if list dimensions match specified arguments.

Parameters

<i>n</i>	Number of images in the list.
<i>dx</i>	Checked image width.
<i>dy</i>	Checked image height.
<i>dz</i>	Checked image depth.
<i>dc</i>	Checked image spectrum.

8.4.4.56 containsNXYZC()

```
template<typename T >
bool containsNXYZC (
    const int n,
    const int x = 0,
    const int y = 0,
    const int z = 0,
    const int c = 0 ) const
```

Test if list contains one particular pixel location.

Parameters

<i>n</i>	Index of the image whom checked pixel value belong to.
<i>x</i>	X-coordinate of the checked pixel value.
<i>y</i>	Y-coordinate of the checked pixel value.
<i>z</i>	Z-coordinate of the checked pixel value.
<i>c</i>	C-coordinate of the checked pixel value.

8.4.4.57 containsN()

```
template<typename T >
bool containsN (
    const int n ) const
```

Test if list contains image with specified index.

Parameters

<i>n</i>	Index of the checked image.
----------	-----------------------------

8.4.4.58 contains() [1/8]

```

template<typename T >
template<typename t >
bool contains (
    const T & pixel,
    t & n,
    t & x,
    t & y,
    t & z,
    t & c ) const

```

Test if one image of the list contains the specified referenced value.

Parameters

	<i>pixel</i>	Reference to pixel value to test.
out	<i>n</i>	Index of image containing the pixel value, if test succeeds.
out	<i>x</i>	X-coordinate of the pixel value, if test succeeds.
out	<i>y</i>	Y-coordinate of the pixel value, if test succeeds.
out	<i>z</i>	Z-coordinate of the pixel value, if test succeeds.
out	<i>c</i>	C-coordinate of the pixel value, if test succeeds.

Note

If true, set coordinates (n,x,y,z,c).

8.4.4.59 contains() [2/8]

```

template<typename T >
template<typename t >
bool contains (
    const T & pixel,
    t & n,
    t & x,
    t & y,
    t & z ) const

```

Test if one image of the list contains the specified referenced value.

Parameters

	<i>pixel</i>	Reference to pixel value to test.
out	<i>n</i>	Index of image containing the pixel value, if test succeeds.
out	<i>x</i>	X-coordinate of the pixel value, if test succeeds.
out	<i>y</i>	Y-coordinate of the pixel value, if test succeeds.
out	<i>z</i>	Z-coordinate of the pixel value, if test succeeds.

Note

If true, set coordinates (n,x,y,z).

8.4.4.60 contains() [3/8]

```
template<typename T >
template<typename t >
bool contains (
    const T & pixel,
    t & n,
    t & x,
    t & y ) const
```

Test if one image of the list contains the specified referenced value.

Parameters

	<i>pixel</i>	Reference to pixel value to test.
out	<i>n</i>	Index of image containing the pixel value, if test succeeds.
out	<i>x</i>	X-coordinate of the pixel value, if test succeeds.
out	<i>y</i>	Y-coordinate of the pixel value, if test succeeds.

Note

If true, set coordinates (n,x,y).

8.4.4.61 contains() [4/8]

```
template<typename T >
template<typename t >
bool contains (
    const T & pixel,
    t & n,
    t & x ) const
```

Test if one image of the list contains the specified referenced value.

Parameters

	<i>pixel</i>	Reference to pixel value to test.
out	<i>n</i>	Index of image containing the pixel value, if test succeeds.
out	<i>x</i>	X-coordinate of the pixel value, if test succeeds.

Note

If true, set coordinates (n,x).

8.4.4.62 contains() [5/8]

```
template<typename T >
template<typename t >
bool contains (
    const T & pixel,
    t & n ) const
```

Test if one image of the list contains the specified referenced value.

Parameters

	<i>pixel</i>	Reference to pixel value to test.
out	<i>n</i>	Index of image containing the pixel value, if test succeeds.

Note

If true, set coordinates (n).

8.4.4.63 contains() [6/8]

```
template<typename T >
bool contains (
    const T & pixel ) const
```

Test if one image of the list contains the specified referenced value.

Parameters

<i>pixel</i>	Reference to pixel value to test.
--------------	-----------------------------------

8.4.4.64 contains() [7/8]

```
template<typename T >
template<typename t >
bool contains (
    const CImg< T > & img,
    t & n ) const
```

Test if the list contains the image 'img'.

Parameters

	<i>img</i>	Reference to image to test.
out	<i>n</i>	Index of image in the list, if test succeeds.

Note

If true, returns the position (n) of the image in the list.

8.4.4.65 contains() [8/8]

```
template<typename T >
bool contains (
    const CImg< T > & img ) const
```

Test if the list contains the image img.

Parameters

<i>img</i>	Reference to image to test.
------------	-----------------------------

8.4.4.66 min_max() [1/2]

```
template<typename T >
template<typename t >
T & min_max (
    t & max_val )
```

Return a reference to the minimum pixel value of the instance list and return the maximum value as well.

Parameters

out	<i>max_val</i>	Value of the maximum value found.
-----	----------------	-----------------------------------

8.4.4.67 min_max() [2/2]

```
template<typename T >
template<typename t >
const T & min_max (
    t & max_val ) const
```

Return a reference to the minimum pixel value of the instance list and return the maximum value as well **[const version]**.

Parameters

out	<i>max_val</i>	Value of the maximum value found.
-----	----------------	-----------------------------------

8.4.4.68 max_min()

```
template<typename T >
template<typename t >
```

```
T & max_min (
    t & min_val )
```

Return a reference to the maximum pixel value of the instance list and return the minimum value as well.

Parameters

out	<i>min_val</i>	Value of the minimum value found.
-----	----------------	-----------------------------------

8.4.4.69 insert() [1/5]

```
template<typename T >
template<typename t >
CImgList< T > & insert (
    const CImg< t > & img,
    const unsigned int pos = ~0U,
    const bool is_shared = false )
```

Insert a copy of the image *img* into the current image list, at position *pos*.

Parameters

<i>img</i>	Image to insert a copy to the list.
<i>pos</i>	Index of the insertion.
<i>is_shared</i>	Indicates whether the inserted image is a shared copy of <i>img</i> .

8.4.4.70 insert() [2/5]

```
template<typename T >
CImgList< T > & insert (
    const unsigned int n,
    const unsigned int pos = ~0U )
```

Insert *n* empty images *img* into the current image list, at position *pos*.

Parameters

<i>n</i>	Number of empty images to insert.
<i>pos</i>	Index of the insertion.

8.4.4.71 insert() [3/5]

```
template<typename T >
template<typename t >
CImgList< T > & insert (
    const unsigned int n,
    const CImg< t > & img,
```



```
const unsigned int pos = ~0U,
const bool is_shared = false )
```

Insert *n* copies of the image *img* into the current image list, at position *pos*.

Parameters

<i>n</i>	Number of image copies to insert.
<i>img</i>	Image to insert by copy.
<i>pos</i>	Index of the insertion.
<i>is_shared</i>	Indicates whether inserted images are shared copies of <i>img</i> .

8.4.4.72 insert() [4/5]

```
template<typename T >
template<typename t >
CImgList< T > & insert (
    const CImgList< t > & list,
    const unsigned int pos = ~0U,
    const bool is_shared = false )
```

Insert a copy of the image list *list* into the current image list, starting from position *pos*.

Parameters

<i>list</i>	Image list to insert.
<i>pos</i>	Index of the insertion.
<i>is_shared</i>	Indicates whether inserted images are shared copies of images of <i>list</i> .

8.4.4.73 insert() [5/5]

```
template<typename T >
template<typename t >
CImgList< T > & insert (
    const unsigned int n,
    const CImgList< t > & list,
    const unsigned int pos = ~0U,
    const bool is_shared = false )
```

Insert *n* copies of the list *list* at position *pos* of the current list.

Parameters

<i>n</i>	Number of list copies to insert.
<i>list</i>	Image list to insert.
<i>pos</i>	Index of the insertion.
<i>is_shared</i>	Indicates whether inserted images are shared copies of images of <i>list</i> .

8.4.4.74 remove() [1/2]

```
template<typename T >
CImgList< T > & remove (
    const unsigned int pos1,
    const unsigned int pos2 )
```

Remove all images between from indexes.

Parameters

<i>pos1</i>	Starting index of the removal.
<i>pos2</i>	Ending index of the removal.

8.4.4.75 remove() [2/2]

```
template<typename T >
CImgList< T > & remove (
    const unsigned int pos )
```

Remove image at index `pos` from the image list.

Parameters

<i>pos</i>	Index of the image to remove.
------------	-------------------------------

8.4.4.76 images()

```
template<typename T >
CImgList< T > & images (
    const unsigned int pos0,
    const unsigned int pos1 )
```

Return a sublist.

Parameters

<i>pos0</i>	Starting index of the sublist.
<i>pos1</i>	Ending index of the sublist.

8.4.4.77 get_shared_images()

```
template<typename T >
CImgList< T > get_shared_images (
    const unsigned int pos0,
    const unsigned int pos1 )
```

Return a shared sublist.

Parameters

<i>pos0</i>	Starting index of the sublist.
<i>pos1</i>	Ending index of the sublist.

8.4.4.78 get_append()

```
template<typename T >
CImg< T > get_append (
    const char axis,
    const float align = 0 ) const
```

Return a single image which is the concatenation of all images of the current [CImgList](#) instance.

Parameters

<i>axis</i>	Concatenation axis. Can be { 'x' 'y' 'z' 'c' }.
<i>align</i>	Concatenation alignment.

8.4.4.79 split()

```
template<typename T >
CImgList< T > & split (
    const char axis,
    const int nb = -1 )
```

Return a list where each image has been split along the specified axis.

Parameters

<i>axis</i>	Axis to split images along.
<i>nb</i>	Number of split parts for each image.

8.4.4.80 push_back() [1/2]

```
template<typename T >
template<typename t >
CImgList< T > & push_back (
    const CImg< t > & img )
```

Insert image at the end of the list.

Parameters

<i>img</i>	Image to insert.
------------	------------------

8.4.4.81 push_front() [1/2]

```
template<typename T >
template<typename t >
CImgList< T > & push_front (
    const CImg< t > & img )
```

Insert image at the front of the list.

Parameters

<i>img</i>	Image to insert.
------------	------------------

8.4.4.82 push_back() [2/2]

```
template<typename T >
template<typename t >
CImgList< T > & push_back (
    const CImgList< t > & list )
```

Insert list at the end of the current list.

Parameters

<i>list</i>	List to insert.
-------------	-----------------

8.4.4.83 push_front() [2/2]

```
template<typename T >
template<typename t >
CImgList< T > & push_front (
    const CImgList< t > & list )
```

Insert list at the front of the current list.

Parameters

<i>list</i>	List to insert.
-------------	-----------------

8.4.4.84 erase()

```
template<typename T >
CImgList< T > & erase (
    const iterator iter )
```

Remove image pointed by iterator.

Parameters

<i>iter</i>	Iterator pointing to the image to remove.
-------------	---

8.4.4.85 `get_select()` [1/2]

```
template<typename T >
CImg< intT > get_select (
    CImgDisplay & disp,
    const bool feature_type = true,
    const char axis = 'x',
    const float align = 0,
    const bool exit_on_anykey = false ) const
```

Display a simple interactive interface to select images or sublists.

Parameters

<i>disp</i>	Window instance to display selection and user interface.
<i>feature_type</i>	Can be <code>false</code> to select a single image, or <code>true</code> to select a sublist.
<i>axis</i>	Axis along whom images are appended for visualization.
<i>align</i>	Alignment setting when images have not all the same size.
<i>exit_on_anykey</i>	Exit function when any key is pressed.

Returns

A one-column vector containing the selected image indexes.

8.4.4.86 `get_select()` [2/2]

```
template<typename T >
CImg< intT > get_select (
    const char *const title,
    const bool feature_type = true,
    const char axis = 'x',
    const float align = 0,
    const bool exit_on_anykey = false ) const
```

Display a simple interactive interface to select images or sublists.

Parameters

<i>title</i>	Title of a new window used to display selection and user interface.
<i>feature_type</i>	Can be <code>false</code> to select a single image, or <code>true</code> to select a sublist.
<i>axis</i>	Axis along whom images are appended for visualization.
<i>align</i>	Alignment setting when images have not all the same size.
<i>exit_on_anykey</i>	Exit function when any key is pressed.

Returns

A one-column vector containing the selected image indexes.

8.4.4.87 load()

```
template<typename T >
CImgList< T > & load (
    const char *const filename )
```

Load a list from a file.

Parameters

<i>filename</i>	Filename to read data from.
-----------------	-----------------------------

8.4.4.88 load_cimg() [1/3]

```
template<typename T >
CImgList< T > & load_cimg (
    const char *const filename )
```

Load a list from a .cimg file.

Parameters

<i>filename</i>	Filename to read data from.
-----------------	-----------------------------

8.4.4.89 load_cimg() [2/3]

```
template<typename T >
CImgList< T > & load_cimg (
    std::FILE *const file )
```

Load a list from a .cimg file.

Parameters

<i>file</i>	File to read data from.
-------------	-------------------------

8.4.4.90 load_cimg() [3/3]

```
template<typename T >
CImgList< T > & load_cimg (
    const char *const filename,
    const unsigned int n0,
    const unsigned int n1,
```

```

const unsigned int x0,
const unsigned int y0,
const unsigned int z0,
const unsigned int c0,
const unsigned int x1,
const unsigned int y1,
const unsigned int z1,
const unsigned int c1 )

```

Load a sublist list from a (non compressed) .cimg file.

Parameters

<i>filename</i>	Filename to read data from.
<i>n0</i>	Starting index of images to read (~0U for max).
<i>n1</i>	Ending index of images to read (~0U for max).
<i>x0</i>	Starting X-coordinates of image regions to read.
<i>y0</i>	Starting Y-coordinates of image regions to read.
<i>z0</i>	Starting Z-coordinates of image regions to read.
<i>c0</i>	Starting C-coordinates of image regions to read.
<i>x1</i>	Ending X-coordinates of image regions to read (~0U for max).
<i>y1</i>	Ending Y-coordinates of image regions to read (~0U for max).
<i>z1</i>	Ending Z-coordinates of image regions to read (~0U for max).
<i>c1</i>	Ending C-coordinates of image regions to read (~0U for max).

8.4.4.91 load_parrec()

```

template<typename T >
CImgList< T > & load_parrec (
    const char *const filename )

```

Load a list from a PAR/REC (Philips) file.

Parameters

<i>filename</i>	Filename to read data from.
-----------------	-----------------------------

8.4.4.92 load_yuv()

```

template<typename T >
CImgList< T > & load_yuv (
    const char *const filename,
    const unsigned int size_x,
    const unsigned int size_y,
    const unsigned int chroma_subsampling = 444,
    const unsigned int first_frame = 0,
    const unsigned int last_frame = ~0U,
    const unsigned int step_frame = 1,
    const bool yuv2rgb = true )

```

Load a list from a YUV image sequence file.

Parameters

<i>filename</i>	Filename to read data from.
<i>size_x</i>	Width of the images.
<i>size_y</i>	Height of the images.
<i>chroma_subsampling</i>	Type of chroma subsampling. Can be { 420 422 444 }.
<i>first_frame</i>	Index of first image frame to read.
<i>last_frame</i>	Index of last image frame to read.
<i>step_frame</i>	Step applied between each frame.
<i>yuv2rgb</i>	Apply YUV to RGB transformation during reading.

8.4.4.93 load_video()

```
template<typename T >
CImgList< T > & load_video (
    const char *const filename,
    const unsigned int first_frame = 0,
    const unsigned int last_frame = ~0U,
    const unsigned int step_frame = 1 )
```

Load an image from a video file, using OpenCV library.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>first_frame</i>	Index of the first frame to read.
<i>last_frame</i>	Index of the last frame to read (can be higher than the actual number of frames, e.g. '~0U').
<i>step_frame</i>	Step value for frame reading.

Note

If `step_frame==0`, the current video stream is forced to be released (without any frames read).

8.4.4.94 load_ffmpeg_external()

```
template<typename T >
CImgList< T > & load_ffmpeg_external (
    const char *const filename )
```

Load an image from a video file using the external tool 'ffmpeg'.

Parameters

<i>filename</i>	Filename to read data from.
-----------------	-----------------------------

8.4.4.95 load_gif_external()

```
template<typename T >
CImgList< T > & load_gif_external (
    const char *const filename )
```

Load gif file, using ImageMagick or GraphicsMagick's external tools.

Parameters

<i>filename</i>	Filename to read data from.
-----------------	-----------------------------

8.4.4.96 load_gzip_external()

```
template<typename T >
CImgList< T > & load_gzip_external (
    const char *const filename )
```

Load a gzipped list, using external tool 'gunzip'.

Parameters

<i>filename</i>	Filename to read data from.
-----------------	-----------------------------

8.4.4.97 load_tiff()

```
template<typename T >
CImgList< T > & load_tiff (
    const char *const filename,
    const unsigned int first_frame = 0,
    const unsigned int last_frame = ~0U,
    const unsigned int step_frame = 1,
    unsigned int *const bits_per_value = 0,
    float *const voxel_size = 0,
    CImg< charT > *const description = 0 )
```

Load images from a TIFF file.

Parameters

	<i>filename</i>	Filename to read data from.
	<i>first_frame</i>	Index of first image frame to read.
	<i>last_frame</i>	Index of last image frame to read.
	<i>step_frame</i>	Step applied between each frame.
out	<i>bits_per_value</i>	Number of bits used to store a scalar value in the image file.
out	<i>voxel_size</i>	Voxel size, as stored in the filename.
out	<i>description</i>	Description, as stored in the filename.

8.4.4.98 print()

```
template<typename T >
const CImgList< T > & print (
    const char *const title = 0,
    const bool display_stats = true ) const
```

Print information about the list on the standard output.

Parameters

<i>title</i>	Label set to the information displayed.
<i>display_stats</i>	Indicates whether image statistics must be computed and displayed.

8.4.4.99 display() [1/3]

```
template<typename T >
const CImgList< T > & display (
    CImgDisplay & disp,
    const char axis = 'x',
    const float align = 0 ) const
```

Display the current [CImgList](#) instance in an existing [CImgDisplay](#) window (by reference).

Parameters

<i>disp</i>	Reference to an existing CImgDisplay instance, where the current image list will be displayed.
<i>axis</i>	Concatenation axis. Can be { 'x' 'y' 'z' 'c' }.
<i>align</i>	Concatenation alignment.

Note

This function displays the list images of the current [CImgList](#) instance into an existing [CImgDisplay](#) window. Images of the list are appended in a single temporary image for visualization purposes. The function returns immediately.

8.4.4.100 display() [2/3]

```
template<typename T >
const CImgList< T > & display (
    CImgDisplay & disp,
    const bool display_info,
    const char axis = 'x',
    const float align = 0,
    unsigned int *const XYZ = 0,
    const bool exit_on_anykey = false ) const
```

Display the current [CImgList](#) instance in a new display window.

Parameters

	<i>disp</i>	Display window.
	<i>display_info</i>	Indicates whether image information is displayed on the standard output.
	<i>axis</i>	Alignment axis for images viewing.
	<i>align</i>	Concatenation alignment.
<i>in, out</i>	<i>XYZ</i>	Contains the XYZ coordinates at start / exit of the function.
	<i>exit_on_anykey</i>	Exit function when any key is pressed.

Note

This function opens a new window with a specific title and displays the list images of the current [CImgList](#) instance into it. Images of the list are appended in a single temporary image for visualization purposes. The function returns when a key is pressed or the display window is closed by the user.

8.4.4.101 **display()** [3/3]

```
template<typename T >
const CImgList< T > & display (
    const char *const title = 0,
    const bool display_info = true,
    const char axis = 'x',
    const float align = 0,
    unsigned int *const XYZ = 0,
    const bool exit_on_anykey = false ) const
```

Display the current [CImgList](#) instance in a new display window.

Parameters

	<i>title</i>	Title of the opening display window.
	<i>display_info</i>	Indicates whether list information must be written on standard output.
	<i>axis</i>	Concatenation axis. Can be { 'x' 'y' 'z' 'c' }.
	<i>align</i>	Concatenation alignment.
<i>in, out</i>	<i>XYZ</i>	Contains the XYZ coordinates at start / exit of the function.
	<i>exit_on_anykey</i>	Exit function when any key is pressed.

8.4.4.102 **save()**

```
template<typename T >
const CImgList< T > & save (
    const char *const filename,
    const int number = -1,
    const unsigned int digits = 6 ) const
```

Save the list into a file.

Parameters

<i>filename</i>	Filename to write data to.
<i>number</i>	When positive, represents an index added to the filename. Otherwise, no number is added.
<i>digits</i>	Number of digits used for adding the number to the filename.

8.4.4.103 is_saveable()

```
template<typename T >
static bool is_saveable (
    const char *const filename ) [static]
```

Indicates whether an image list can be saved as one single file or not.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

Returns

`true` if the file format supports multiple images, `false` otherwise.

8.4.4.104 save_gif_external()

```
template<typename T >
const CImgList< T > & save_gif_external (
    const char *const filename,
    const float fps = 25,
    const unsigned int nb_loops = 0 )
```

Save the image sequence as a GIF animated file.

Parameters

<i>filename</i>	Filename to write data to.
<i>fps</i>	Number of desired frames per second.
<i>nb_loops</i>	Number of loops (0 for infinite looping).

8.4.4.105 save_yuv() [1/2]

```
template<typename T >
const CImgList< T > & save_yuv (
    const char *const filename = 0,
    const unsigned int chroma_subsampling = 444,
    const bool is_rgb = true ) const
```

Save the list as a YUV image sequence file.

Parameters

<i>filename</i>	Filename to write data to.
<i>chroma_subsampling</i>	Type of chroma subsampling. Can be { 420 422 444 }.
<i>is_rgb</i>	Indicates whether the RGB to YUV conversion must be done for saving.

8.4.4.106 save_yuv() [2/2]

```
template<typename T >
const CImgList< T > & save_yuv (
    std::FILE *const file,
    const unsigned int chroma_subsampling = 444,
    const bool is_rgb = true ) const
```

Save the image sequence into a YUV file.

Parameters

<i>file</i>	File to write data to.
<i>chroma_subsampling</i>	Type of chroma subsampling. Can be { 420 422 444 }.
<i>is_rgb</i>	Indicates whether the RGB to YUV conversion must be done for saving.

8.4.4.107 save_cimg() [1/4]

```
template<typename T >
const CImgList< T > & save_cimg (
    const char *const filename,
    const bool is_compressed = false ) const
```

Save the list as a .cimg file.

Parameters

<i>filename</i>	Filename to write data to.
<i>is_compressed</i>	Indicates whether data compression must be enabled.

8.4.4.108 save_cimg() [2/4]

```
template<typename T >
const CImgList< T > & save_cimg (
    std::FILE * file,
    const bool is_compressed = false ) const
```

Save the list as a .cimg file.

Parameters

<i>file</i>	File to write data to.
<i>is_compressed</i>	Indicates whether data compression must be enabled.

8.4.4.109 save_cimg() [3/4]

```
template<typename T >
const CImgList< T > & save_cimg (
    const char *const filename,
    const unsigned int n0,
    const unsigned int x0,
    const unsigned int y0,
    const unsigned int z0,
    const unsigned int c0 ) const
```

Insert the image list instance into an existing .cimg file, at specified coordinates.

Parameters

<i>filename</i>	Filename to write data to.
<i>n0</i>	Starting index of images to write.
<i>x0</i>	Starting X-coordinates of image regions to write.
<i>y0</i>	Starting Y-coordinates of image regions to write.
<i>z0</i>	Starting Z-coordinates of image regions to write.
<i>c0</i>	Starting C-coordinates of image regions to write.

8.4.4.110 save_cimg() [4/4]

```
template<typename T >
const CImgList< T > & save_cimg (
    std::FILE *const file,
    const unsigned int n0,
    const unsigned int x0,
    const unsigned int y0,
    const unsigned int z0,
    const unsigned int c0 ) const
```

Insert the image list instance into an existing .cimg file, at specified coordinates.

Parameters

<i>file</i>	File to write data to.
<i>n0</i>	Starting index of images to write.
<i>x0</i>	Starting X-coordinates of image regions to write.
<i>y0</i>	Starting Y-coordinates of image regions to write.
<i>z0</i>	Starting Z-coordinates of image regions to write.
<i>c0</i>	Starting C-coordinates of image regions to write.

8.4.4.111 save_empty_cimg() [1/2]

```
template<typename T >
static void save_empty_cimg (
    const char *const filename,
```

```

const unsigned int nb,
const unsigned int dx,
const unsigned int dy = 1,
const unsigned int dz = 1,
const unsigned int dc = 1 ) [static]

```

Save an empty (non-compressed) .cimg file with specified dimensions.

Parameters

<i>filename</i>	Filename to write data to.
<i>nb</i>	Number of images to write.
<i>dx</i>	Width of images in the written file.
<i>dy</i>	Height of images in the written file.
<i>dz</i>	Depth of images in the written file.
<i>dc</i>	Spectrum of images in the written file.

8.4.4.112 save_empty_cimg() [2/2]

```

template<typename T >
static void save_empty_cimg (
    std::FILE *const file,
    const unsigned int nb,
    const unsigned int dx,
    const unsigned int dy = 1,
    const unsigned int dz = 1,
    const unsigned int dc = 1 ) [static]

```

Save an empty .cimg file with specified dimensions.

Parameters

<i>file</i>	File to write data to.
<i>nb</i>	Number of images to write.
<i>dx</i>	Width of images in the written file.
<i>dy</i>	Height of images in the written file.
<i>dz</i>	Depth of images in the written file.
<i>dc</i>	Spectrum of images in the written file.

8.4.4.113 save_tiff()

```

template<typename T >
const CImgList< T > & save_tiff (
    const char *const filename,
    const unsigned int compression_type = 0,
    const float *const voxel_size = 0,
    const char *const description = 0,
    const bool use_bigtiff = true ) const

```

Save the list as a TIFF file.

Parameters

<i>filename</i>	Filename to write data to.
<i>compression_type</i>	Compression mode used to write data.
<i>voxel_size</i>	Voxel size, to be stored in the file metadata.
<i>description</i>	Description, to be stored in the file metadata.
<i>use_bigtiff</i>	Indicates whether the file is saved as BigTIFF (>4 GB) or not.

8.4.4.114 save_gzip_external()

```
template<typename T >
const CImgList< T > & save_gzip_external (
    const char *const filename ) const
```

Save the list as a gzipped file using the external tool 'gzip'.

Parameters

<i>filename</i>	Filename to write data to.
-----------------	----------------------------

8.4.4.115 save_video()

```
template<typename T >
const CImgList< T > & save_video (
    const char *const filename,
    const unsigned int fps = 25,
    const char * codec = 0,
    const bool keep_open = false ) const
```

Save the image sequence (using the OpenCV library when available).

Parameters

<i>filename</i>	Filename to write data to.
<i>fps</i>	Number of frames per second.
<i>codec</i>	Type of compression (See http://www.fourcc.org/codecs.php to see available codecs).
<i>keep_open</i>	Indicates whether the video writer associated with the specified filename must be kept open (to allow frames to be added in the same file afterwards).

8.4.4.116 save_ffmpeg_external()

```
template<typename T >
const CImgList< T > & save_ffmpeg_external (
    const char *const filename,
    const unsigned int fps = 25,
    const char *const codec = 0,
    const unsigned int bitrate = 2048 ) const
```


Save the image sequence, using the external tool 'ffmpeg'.

Parameters

<i>filename</i>	Filename to write data to.
<i>fps</i>	Number of frames per second.
<i>codec</i>	Type of compression.
<i>bitrate</i>	Output bitrate

8.4.4.117 get_serialize()

```
template<typename T >
CImg< ucharT > get_serialize (
    const bool is_compressed = false,
    const unsigned int header_size = 0 ) const
```

Serialize a CImgList<T> instance into a raw CImg<unsigned char> buffer.

Parameters

<i>is_compressed</i>	Indicates that zlib compression must be used for serialization (this requires 'cimg_use_zlib' been enabled).
<i>header_size</i>	Reserve empty bytes as a starting header.

8.4.4.118 font()

```
template<typename T >
static const CImgList< ucharT > & font (
    const unsigned int requested_height,
    const bool is_variable_width = true ) [static]
```

Return a predefined CImg font with the requested height.

Parameters

<i>requested_height</i>	Height of the desired font (exact match for 13,23,53,103).
<i>is_variable_width</i>	Decides if the font has a variable (<code>true</code>) or fixed (<code>false</code>) width.

Note

Beware, the returned reference is valid only until the next call to this function!

8.4.4.119 FFT() [1/2]

```
template<typename T >
CImgList< T > & FFT (
```

```
const char axis,  
const bool invert = false )
```

Compute a 1D Fast Fourier Transform, along specified axis.

Parameters

<i>axis</i>	Axis along which the Fourier transform is computed.
<i>invert</i>	Indicates whether the direct (<i>false</i>) or inverse transform (<i>true</i>) is computed.

8.4.4.120 FFT() [2/2]

```
template<typename T >
CImgList< T > & FFT (
    const bool invert = false )
```

Compute n-D Fast Fourier Transform.

Parameters

<i>invert</i>	Indicates whether the direct (<i>false</i>) or inverse transform (<i>true</i>) is computed.
---------------	---

Index

- ~CImg
 - CImg< T >, [116](#)
- ~CImgDisplay
 - CImgDisplay, [348](#)
- ~CImgList
 - CImgList< T >, [383](#)
- abs
 - CImg< T >, [183](#)
- abscut
 - CImg< T >, [224](#)
- acos
 - CImg< T >, [186](#)
- acosh
 - CImg< T >, [187](#)
- append
 - CImg< T >, [245](#)
- append_object3d
 - CImg< T >, [269](#)
- asin
 - CImg< T >, [186](#)
- asinh
 - CImg< T >, [187](#)
- assign
 - CImg< T >, [125–127](#)
 - CImgDisplay, [352](#)
 - CImgList< T >, [390–395](#)
- at
 - CImg< T >, [158](#), [159](#)
 - CImgList< T >, [402](#)
- atan
 - CImg< T >, [186](#)
- atan2
 - CImg< T >, [186](#)
- atanh
 - CImg< T >, [187](#)
- atN
 - CImgList< T >, [407](#)
- atNX
 - CImgList< T >, [406](#)
- atNXY
 - CImgList< T >, [405](#)
- atNXYZ
 - CImgList< T >, [404](#)
- atNXYZC
 - CImgList< T >, [403](#)
- atof
 - cimg_library::cimg, [46](#)
- atX
 - CImg< T >, [159](#), [160](#)
- atXY
 - CImg< T >, [161](#)
- atXYZ
 - CImg< T >, [161](#)
- atXYZC
 - CImg< T >, [162](#)
- autocrop
 - CImg< T >, [239](#)
- back
 - CImg< T >, [158](#)
- begin
 - CImg< T >, [157](#)
- blur
 - CImg< T >, [254](#)
- blur_anisotropic
 - CImg< T >, [255](#)
- blur_bilateral
 - CImg< T >, [256](#)
- blur_box
 - CImg< T >, [257](#), [258](#)
- blur_guided
 - CImg< T >, [258](#)
- blur_median
 - CImg< T >, [260](#)
- blur_patch
 - CImg< T >, [260](#)
- box3d
 - CImg< T >, [277](#)
- boxfilter
 - CImg< T >, [257](#)
- button
 - CImgDisplay, [357](#)
- CImg
 - CImg< T >, [116–124](#)
- CImg Library Overview, [11](#)
- CImg< T >, [61](#)
 - ~CImg, [116](#)
 - abs, [183](#)
 - abscut, [224](#)
 - acos, [186](#)
 - acosh, [187](#)
 - append, [245](#)
 - append_object3d, [269](#)
 - asin, [186](#)
 - asinh, [187](#)
 - assign, [125–127](#)
 - at, [158](#), [159](#)
 - atan, [186](#)

atan2, 186
atanh, 187
atX, 159, 160
atXY, 161
atXYZ, 161
atXYZC, 162
autocrop, 239
back, 158
begin, 157
blur, 254
blur_anisotropic, 255
blur_bilateral, 256
blur_box, 257, 258
blur_guided, 258
blur_median, 260
blur_patch, 260
box3d, 277
boxfilter, 257
CImg, 116–124
CImg3dtoobject3d, 281
clear, 128
closing, 250
columns, 239
cone3d, 277
const_iterator, 115
contains, 178, 179
containsXYZC, 177
convolve, 247
cool_LUT256, 229
correlate, 245
cos, 183
cosh, 185
crop, 237
cross, 204
cube_LUT256, 230
cubic_atX, 166, 167
cubic_atX_c, 166, 167
cubic_atXY, 168
cubic_atXY_c, 168
cubic_atXYZ, 169
cubic_atXYZ_c, 169
cubic_atXYZ_p, 170
cumulate, 248
cut, 223
cylinder3d, 278
data, 156
default_LUT256, 228
depth, 155
deriche, 253
diagonal, 203
diffusion_tensors, 262
dijkstra, 210
dilate, 249, 250
discard, 219
displacement, 262
display, 327, 328
display_graph, 329
display_object3d, 328
distance, 264
distance_dijkstra, 264
distance_eikonal, 266
div, 188
dot, 200
draw_arrow, 287
draw_axes, 307
draw_axis, 306, 307
draw_circle, 301, 302
draw_ellipse, 299–301
draw_fill, 309
draw_gaussian, 311, 312
draw_graph, 308
draw_grid, 308
draw_image, 302
draw_line, 283–287
draw_mandelbrot, 310
draw_object3d, 312
draw_plasma, 310
draw_point, 282
draw_polygon, 299
draw_quiver, 305, 306
draw_rectangle, 297, 298
draw_spline, 288–290
draw_text, 303, 304
draw_triangle, 290–296
eigen, 205
elevation3d, 276
ellipsoid3d, 280
empty, 129
end, 157
equalize, 226
erode, 248, 249
eval, 198, 199
exp, 181
FFT, 267, 268
fill, 216–218
fill_from_values, 217
fillC, 219
fillX, 218
fillY, 218
fillZ, 219
flag_LUT256, 229
front, 158
get_channel, 241
get_channels, 241
get_column, 239
get_eigen, 205
get_elevation3d, 271
get_FFT, 267
get_gradient, 261
get_hessian, 261
get_isoline3d, 272
get_isosurface3d, 274
get_matrix_at, 201
get_projections2d, 237
get_projections3d, 272
get_rows, 240

get_serialize, 343
get_shared_channel, 244
get_shared_channels, 244
get_shared_points, 242
get_shared_row, 243
get_shared_rows, 242
get_shared_slice, 244
get_shared_slices, 243
get_slice, 240
get_slices, 240
get_split, 244, 245
get_stats, 200
get_SVD, 208
get_symmetric_eigen, 207
get_tensor_at, 201
get_vector_at, 201
haar, 266, 267
height, 154
histogram, 225
hot_LUT256, 229
HSV_LUT256, 228
identity_matrix, 203, 215
index, 226
invert, 204
is_CImg3d, 180
is_empty, 172
is_inf, 172
is_nan, 172
is_object3d, 180
is_overlapped, 179
is_sameXC, 174
is_sameXY, 173
is_sameXYC, 176
is_sameXYZ, 175
is_sameXYZC, 177
is_sameXZ, 173
is_sameXZC, 176
is_sameYC, 174, 175
is_sameYZ, 174
is_sameYZC, 176, 177
is_sameZC, 175
is_shared, 172
isoline3d, 273
isosurface3d, 275
iterator, 115
jet_LUT256, 229
kth_smallest, 196
label, 227, 228
linear_atX, 162, 163
linear_atXY, 164
linear_atXYZ, 164, 165
linear_atXYZC, 165
lines_LUT256, 228
load, 314
load_analyze, 319
load_ascii, 315
load_bmp, 315
load_camera, 326
load_cimg, 319, 320
load_dcraw_external, 326
load_dlm, 315
load_exr, 320
load_ffmpeg_external, 323
load_gif_external, 324
load_graphicmagick_external, 324
load_gzip_external, 325
load_heif, 324
load_imagemagick_external, 325
load_inr, 320
load_jpeg, 315
load_jxl, 316
load_magick, 316
load_medcon_external, 325
load_minc2, 319
load_off, 322
load_other, 327
load_pandore, 321
load_parrec, 321
load_pdf_external, 326
load_pfm, 317
load_png, 316
load_pnm, 317
load_raw, 321
load_rgb, 317
load_rgba, 318
load_tiff, 318
load_video, 323
load_webp, 324
load_yuv, 322
log, 182
log10, 182
log2, 182
magnitude, 200
map, 227
matchpatch, 263
matrix, 214
max, 193
max_min, 196
maxabs, 195
min, 191
min_max, 196
minabs, 194
mirror, 233
move_to, 128
MSE, 198
mul, 188
noise, 222
norm, 223
normalize, 222, 223
object3dtoCImg3d, 281
offset, 157
opening, 252
operator T*, 131
operator!=, 150, 151
operator<, 153
operator<<, 147

operator<=, 146, 147
operator>, 148, 149
operator>=, 148
operator(), 130, 131
operator+, 135, 136
operator++, 135
operator+=, 133, 134
operator,, 151, 153
operator-, 137, 138
operator--, 137
operator-=, 136
operator/, 141
operator/=: 140
operator=, 132, 133
operator==, 149, 150
operator%, 142
operator%=: 141, 142
operator&, 143
operator&=: 142, 143
operator*, 139
operator*=, 138
operator~, 149
operator^, 146
operator^=: 145
operator|, 144, 145
operator|=, 144
otsu, 224
permute_axes, 234
pixel_type, 154
plane3d, 279
pow, 188, 189
print, 327
project_matrix, 209
PSNR, 198
QR, 209
quantize, 224
rand, 220
resize, 230–232
resize_doubleXY, 232
resize_object3d, 269
resize_tripleXY, 233
RGBtoXYZ, 230
rol, 189, 190
ror, 190
rotate, 234–236
rotate_object3d, 268
rotation_matrix, 216
round, 220
row, 240
row_vector, 211, 212
save, 330
save_analyze, 335
save_ascii, 330
save_bmp, 331
save_cimg, 336
save_cpp, 331
save_dlm, 331
save_empty_cimg, 336, 337
save_exr, 337
save_ffmpeg_external, 341
save_graphicmagick_external, 341
save_gzip_external, 341
save_imagemagick_external, 342
save_inr, 337
save_jpeg, 332
save_jxl, 332
save_magick, 332
save_medcon_external, 342
save_minc2, 335
save_off, 339, 340
save_other, 342
save_pandore, 338
save_pfm, 333
save_png, 333
save_pnk, 333
save_pnm, 333
save_raw, 338, 339
save_rgb, 334
save_rgba, 334
save_tiff, 334
save_video, 340
save_webp, 331
save_yuv, 339
select, 314
sequence, 203, 216
set_linear_atX, 170
set_linear_atXY, 171
set_linear_atXYZ, 171
set_matrix_at, 202
set_tensor_at, 202
set_vector_at, 202
sharpen, 261
shift, 233
shift_object3d, 268, 269
sign, 183
sin, 184
sinc, 184
sinh, 185
size, 155
solve, 204
solve_tridiagonal, 204
sort, 207
spectrum, 155
sphere3d, 280
sqr, 181
sqrt, 181
streamline, 241
string, 211
structure_tensors, 262
SVD, 208
swap, 129
symmetric_eigen, 205
tan, 184
tanh, 185
tensor, 215
texturize_object3d, 271

- threshold, [225](#)
- torus3d, [279](#)
- transpose, [203](#)
- unroll, [234](#)
- value_string, [171](#)
- value_type, [115](#)
- vanvliet, [253](#)
- variance, [196](#)
- variance_mean, [197](#)
- variance_noise, [197](#)
- vector, [213](#)
- warp, [236](#)
- watershed, [252](#)
- width, [154](#)
- XYZtoRGB, [230](#)
- CImg3dtoobject3d
 - CImg< T >, [281](#)
- cimg_library, [29](#)
- cimg_library::cimg, [30](#)
 - atof, [46](#)
 - create_directory, [41](#)
 - curl_path, [53](#)
 - date, [52](#), [53](#)
 - dcraw_path, [54](#)
 - dialog, [60](#)
 - endianness, [43](#)
 - eval, [41](#)
 - exception_mode, [40](#)
 - fclose, [50](#)
 - fdate, [51](#), [52](#)
 - fempty, [58](#)
 - ffmpeg_path, [54](#)
 - files, [59](#)
 - fopen, [49](#)
 - fread, [57](#)
 - fsize, [51](#)
 - ftype, [58](#)
 - fwrite, [57](#)
 - graphicsmagick_path, [54](#)
 - gunzip_path, [55](#)
 - gzip_path, [55](#)
 - imagemagick_path, [55](#)
 - info, [39](#)
 - invert_endianness, [43](#)
 - is_directory, [50](#)
 - is_file, [50](#)
 - load_network, [59](#)
 - medcon_path, [56](#)
 - minmod, [45](#)
 - mod, [45](#)
 - openmp_mode, [40](#)
 - output, [39](#)
 - path_exists, [51](#)
 - round, [45](#)
 - sleep, [44](#)
 - split_filename, [57](#)
 - strcasecmp, [47](#)
 - strellipse, [48](#)
 - strncasecmp, [47](#)
 - strpare, [48](#)
 - strunescape, [49](#)
 - strwindows_reserved, [49](#)
 - system, [42](#)
 - temporary_path, [56](#)
 - tic, [44](#)
 - time, [43](#)
 - toc, [44](#)
 - wait, [44](#)
 - warn, [42](#)
 - wave, [46](#)
 - wget_path, [56](#)
- CImgDisplay, [343](#)
 - ~CImgDisplay, [348](#)
 - assign, [352](#)
 - button, [357](#)
 - CImgDisplay, [348](#), [349](#), [351](#)
 - close, [362](#)
 - display, [361](#)
 - empty, [352](#)
 - flush, [368](#)
 - frames_per_second, [361](#)
 - height, [355](#)
 - hide_mouse, [366](#)
 - is_closed, [353](#)
 - is_key, [353](#), [354](#)
 - is_key_sequence, [354](#)
 - is_keyESC, [355](#)
 - key, [358](#)
 - keycode, [360](#)
 - mouse_x, [357](#)
 - mouse_y, [357](#)
 - move, [362](#)
 - normalization, [355](#)
 - operator bool, [353](#)
 - operator=, [352](#), [353](#)
 - paint, [369](#)
 - released_key, [360](#)
 - render, [368](#)
 - resize, [362–364](#)
 - screenshot, [352](#), [369](#)
 - set_button, [366](#)
 - set_fullscreen, [365](#)
 - set_key, [367](#)
 - set_mouse, [366](#)
 - set_normalization, [364](#)
 - set_title, [364](#)
 - set_wheel, [367](#)
 - show, [362](#)
 - show_mouse, [365](#)
 - snapshot, [369](#)
 - title, [356](#)
 - toggle_fullscreen, [365](#)
 - wait, [368](#)
 - wheel, [358](#)
 - width, [355](#)
 - window_height, [356](#)

- window_width, 356
 - window_x, 357
 - window_y, 357
- CImgException, 370
- CImgList
 - CImgList< T >, 383–390
- CImgList< T >, 371
 - ~CImgList, 383
 - assign, 390–395
 - at, 402
 - atN, 407
 - atNX, 406
 - atNXY, 405
 - atNXYZ, 404
 - atNXYZC, 403
 - CImgList, 383–390
 - clear, 391
 - const_iterator, 382
 - contains, 410–413
 - containsN, 409
 - containsNXYZC, 409
 - data, 402
 - display, 424, 425
 - empty, 397
 - erase, 418
 - FFT, 431, 433
 - font, 431
 - get_append, 417
 - get_select, 419
 - get_serialize, 431
 - get_shared, 390
 - get_shared_images, 416
 - images, 416
 - insert, 414, 415
 - is_sameN, 408
 - is_sameNXYZC, 408
 - is_sameXYZC, 408
 - is_saveable, 426
 - iterator, 382
 - load, 420
 - load_cimg, 420
 - load_ffmpeg_external, 422
 - load_gif_external, 422
 - load_gzip_external, 423
 - load_parrec, 421
 - load_tiff, 423
 - load_video, 422
 - load_yuv, 421
 - max_min, 413
 - min_max, 413
 - move_to, 396
 - operator CImg< T > *, 398
 - operator<, 401
 - operator>, 400
 - operator(), 397, 398
 - operator+, 399
 - operator., 400
 - operator=, 398, 399
 - pixel_type, 401
 - print, 423
 - push_back, 417, 418
 - push_front, 417, 418
 - remove, 415, 416
 - save, 425
 - save_cimg, 427, 428
 - save_empty_cimg, 428, 429
 - save_ffmpeg_external, 430
 - save_gif_external, 426
 - save_gzip_external, 430
 - save_tiff, 429
 - save_video, 430
 - save_yuv, 426, 427
 - size, 401
 - split, 417
 - swap, 396
 - value_type, 383
 - width, 401
- clear
 - CImg< T >, 128
 - CImgList< T >, 391
- close
 - CImgDisplay, 362
- closing
 - CImg< T >, 250
- columns
 - CImg< T >, 239
- cone3d
 - CImg< T >, 277
- const_iterator
 - CImg< T >, 115
 - CImgList< T >, 382
- contains
 - CImg< T >, 178, 179
 - CImgList< T >, 410–413
- containsN
 - CImgList< T >, 409
- containsNXYZC
 - CImgList< T >, 409
- containsXYZC
 - CImg< T >, 177
- convolve
 - CImg< T >, 247
- cool_LUT256
 - CImg< T >, 229
- correlate
 - CImg< T >, 245
- cos
 - CImg< T >, 183
- cosh
 - CImg< T >, 185
- create_directory
 - cimg_library::cimg, 41
- crop
 - CImg< T >, 237
- cross
 - CImg< T >, 204

- cube_LUT256
 - Clmg< T >, 230
- cubic_atX
 - Clmg< T >, 166, 167
- cubic_atX_c
 - Clmg< T >, 166, 167
- cubic_atXY
 - Clmg< T >, 168
- cubic_atXY_c
 - Clmg< T >, 168
- cubic_atXYZ
 - Clmg< T >, 169
- cubic_atXYZ_c
 - Clmg< T >, 169
- cubic_atXYZ_p
 - Clmg< T >, 170
- cumulate
 - Clmg< T >, 248
- curl_path
 - cimg_library::cimg, 53
- cut
 - Clmg< T >, 223
- cylinder3d
 - Clmg< T >, 278
- data
 - Clmg< T >, 156
 - ClmgList< T >, 402
- date
 - cimg_library::cimg, 52, 53
- dcraw_path
 - cimg_library::cimg, 54
- default_LUT256
 - Clmg< T >, 228
- depth
 - Clmg< T >, 155
- deriche
 - Clmg< T >, 253
- diagonal
 - Clmg< T >, 203
- dialog
 - cimg_library::cimg, 60
- diffusion_tensors
 - Clmg< T >, 262
- dijkstra
 - Clmg< T >, 210
- dilate
 - Clmg< T >, 249, 250
- discard
 - Clmg< T >, 219
- displacement
 - Clmg< T >, 262
- display
 - Clmg< T >, 327, 328
 - ClmgDisplay, 361
 - ClmgList< T >, 424, 425
- display_graph
 - Clmg< T >, 329
- display_object3d
 - Clmg< T >, 328
- distance
 - Clmg< T >, 264
- distance_dijkstra
 - Clmg< T >, 264
- distance_eikonal
 - Clmg< T >, 266
- div
 - Clmg< T >, 188
- dot
 - Clmg< T >, 200
- draw_arrow
 - Clmg< T >, 287
- draw_axes
 - Clmg< T >, 307
- draw_axis
 - Clmg< T >, 306, 307
- draw_circle
 - Clmg< T >, 301, 302
- draw_ellipse
 - Clmg< T >, 299–301
- draw_fill
 - Clmg< T >, 309
- draw_gaussian
 - Clmg< T >, 311, 312
- draw_graph
 - Clmg< T >, 308
- draw_grid
 - Clmg< T >, 308
- draw_image
 - Clmg< T >, 302
- draw_line
 - Clmg< T >, 283–287
- draw_mandelbrot
 - Clmg< T >, 310
- draw_object3d
 - Clmg< T >, 312
- draw_plasma
 - Clmg< T >, 310
- draw_point
 - Clmg< T >, 282
- draw_polygon
 - Clmg< T >, 299
- draw_quiver
 - Clmg< T >, 305, 306
- draw_rectangle
 - Clmg< T >, 297, 298
- draw_spline
 - Clmg< T >, 288–290
- draw_text
 - Clmg< T >, 303, 304
- draw_triangle
 - Clmg< T >, 290–296
- eigen
 - Clmg< T >, 205
- elevation3d
 - Clmg< T >, 276
- ellipsoid3d

- CImg< T >, 280
- empty
 - CImg< T >, 129
 - CImgDisplay, 352
 - CImgList< T >, 397
- end
 - CImg< T >, 157
- endianness
 - cimg_library::cimg, 43
- equalize
 - CImg< T >, 226
- erase
 - CImgList< T >, 418
- erode
 - CImg< T >, 248, 249
- eval
 - CImg< T >, 198, 199
 - cimg_library::cimg, 41
- exception_mode
 - cimg_library::cimg, 40
- exp
 - CImg< T >, 181
- FAQ : Frequently Asked Questions., 13
- fclose
 - cimg_library::cimg, 50
- fdate
 - cimg_library::cimg, 51, 52
- fempty
 - cimg_library::cimg, 58
- ffmpeg_path
 - cimg_library::cimg, 54
- FFT
 - CImg< T >, 267, 268
 - CImgList< T >, 431, 433
- files
 - cimg_library::cimg, 59
- Files IO in CImg., 25
- fill
 - CImg< T >, 216–218
- fill_from_values
 - CImg< T >, 217
- fillC
 - CImg< T >, 219
- fillX
 - CImg< T >, 218
- fillY
 - CImg< T >, 218
- fillZ
 - CImg< T >, 219
- flag_LUT256
 - CImg< T >, 229
- flush
 - CImgDisplay, 368
- font
 - CImgList< T >, 431
- fopen
 - cimg_library::cimg, 49
- frames_per_second
 - CImgDisplay, 361
- fread
 - cimg_library::cimg, 57
- front
 - CImg< T >, 158
- fsize
 - cimg_library::cimg, 51
- ftype
 - cimg_library::cimg, 58
- fwrite
 - cimg_library::cimg, 57
- get_append
 - CImgList< T >, 417
- get_channel
 - CImg< T >, 241
- get_channels
 - CImg< T >, 241
- get_column
 - CImg< T >, 239
- get_eigen
 - CImg< T >, 205
- get_elevation3d
 - CImg< T >, 271
- get_FFT
 - CImg< T >, 267
- get_gradient
 - CImg< T >, 261
- get_hessian
 - CImg< T >, 261
- get_isoline3d
 - CImg< T >, 272
- get_isosurface3d
 - CImg< T >, 274
- get_matrix_at
 - CImg< T >, 201
- get_projections2d
 - CImg< T >, 237
- get_projections3d
 - CImg< T >, 272
- get_rows
 - CImg< T >, 240
- get_select
 - CImgList< T >, 419
- get_serialize
 - CImg< T >, 343
 - CImgList< T >, 431
- get_shared
 - CImgList< T >, 390
- get_shared_channel
 - CImg< T >, 244
- get_shared_channels
 - CImg< T >, 244
- get_shared_images
 - CImgList< T >, 416
- get_shared_points
 - CImg< T >, 242
- get_shared_row
 - CImg< T >, 243

- get_shared_rows
 - Clmg< T >, 242
- get_shared_slice
 - Clmg< T >, 244
- get_shared_slices
 - Clmg< T >, 243
- get_slice
 - Clmg< T >, 240
- get_slices
 - Clmg< T >, 240
- get_split
 - Clmg< T >, 244, 245
- get_stats
 - Clmg< T >, 200
- get_SVD
 - Clmg< T >, 208
- get_symmetric_eigen
 - Clmg< T >, 207
- get_tensor_at
 - Clmg< T >, 201
- get_vector_at
 - Clmg< T >, 201
- graphicsmagick_path
 - cimg_library::cimg, 54
- gunzip_path
 - cimg_library::cimg, 55
- gzip_path
 - cimg_library::cimg, 55
- haar
 - Clmg< T >, 266, 267
- height
 - Clmg< T >, 154
 - ClmgDisplay, 355
- hide_mouse
 - ClmgDisplay, 366
- histogram
 - Clmg< T >, 225
- hot_LUT256
 - Clmg< T >, 229
- How pixel data are stored with Clmg., 24
- How to use Clmg library with Visual C++ 2005 Express Edition ?, 17
- HSV_LUT256
 - Clmg< T >, 228
- identity_matrix
 - Clmg< T >, 203, 215
- imagemagick_path
 - cimg_library::cimg, 55
- images
 - ClmgList< T >, 416
- index
 - Clmg< T >, 226
- info
 - cimg_library::cimg, 39
- insert
 - ClmgList< T >, 414, 415
- invert
 - Clmg< T >, 204
- invert_endianness
 - cimg_library::cimg, 43
- is_Clmg3d
 - Clmg< T >, 180
- is_closed
 - ClmgDisplay, 353
- is_directory
 - cimg_library::cimg, 50
- is_empty
 - Clmg< T >, 172
- is_file
 - cimg_library::cimg, 50
- is_inf
 - Clmg< T >, 172
- is_key
 - ClmgDisplay, 353, 354
- is_key_sequence
 - ClmgDisplay, 354
- is_keyESC
 - ClmgDisplay, 355
- is_nan
 - Clmg< T >, 172
- is_object3d
 - Clmg< T >, 180
- is_overlapped
 - Clmg< T >, 179
- is_sameN
 - ClmgList< T >, 408
- is_sameNXYZC
 - ClmgList< T >, 408
- is_sameXC
 - Clmg< T >, 174
- is_sameXY
 - Clmg< T >, 173
- is_sameXYC
 - Clmg< T >, 176
- is_sameXYZ
 - Clmg< T >, 175
- is_sameXYZC
 - Clmg< T >, 177
 - ClmgList< T >, 408
- is_sameXZ
 - Clmg< T >, 173
- is_sameXZC
 - Clmg< T >, 176
- is_sameYC
 - Clmg< T >, 174, 175
- is_sameYZ
 - Clmg< T >, 174
- is_sameYZC
 - Clmg< T >, 176, 177
- is_sameZC
 - Clmg< T >, 175
- is_saveable
 - ClmgList< T >, 426
- is_shared
 - Clmg< T >, 172

- isoline3d
 - Clmg< T >, 273
- isosurface3d
 - Clmg< T >, 275
- iterator
 - Clmg< T >, 115
 - ClmgList< T >, 382
- jet_LUT256
 - Clmg< T >, 229
- key
 - ClmgDisplay, 358
- keycode
 - ClmgDisplay, 360
- kth_smallest
 - Clmg< T >, 196
- label
 - Clmg< T >, 227, 228
- linear_atX
 - Clmg< T >, 162, 163
- linear_atXY
 - Clmg< T >, 164
- linear_atXYZ
 - Clmg< T >, 164, 165
- linear_atXYZC
 - Clmg< T >, 165
- lines_LUT256
 - Clmg< T >, 228
- load
 - Clmg< T >, 314
 - ClmgList< T >, 420
- load_analyze
 - Clmg< T >, 319
- load_ascii
 - Clmg< T >, 315
- load_bmp
 - Clmg< T >, 315
- load_camera
 - Clmg< T >, 326
- load_cimg
 - Clmg< T >, 319, 320
 - ClmgList< T >, 420
- load_dcraw_external
 - Clmg< T >, 326
- load_dlm
 - Clmg< T >, 315
- load_exr
 - Clmg< T >, 320
- load_ffmpeg_external
 - Clmg< T >, 323
 - ClmgList< T >, 422
- load_gif_external
 - Clmg< T >, 324
 - ClmgList< T >, 422
- load_graphicsmagick_external
 - Clmg< T >, 324
- load_gzip_external
 - Clmg< T >, 325
- load_heif
 - Clmg< T >, 324
- load_imagemagick_external
 - Clmg< T >, 325
- load_inr
 - Clmg< T >, 320
- load_jpeg
 - Clmg< T >, 315
- load_jxl
 - Clmg< T >, 316
- load_magick
 - Clmg< T >, 316
- load_medcon_external
 - Clmg< T >, 325
- load_minc2
 - Clmg< T >, 319
- load_network
 - cimg_library::cimg, 59
- load_off
 - Clmg< T >, 322
- load_other
 - Clmg< T >, 327
- load_pandore
 - Clmg< T >, 321
- load_parrec
 - Clmg< T >, 321
 - ClmgList< T >, 421
- load_pdf_external
 - Clmg< T >, 326
- load_pfm
 - Clmg< T >, 317
- load_png
 - Clmg< T >, 316
- load_pnm
 - Clmg< T >, 317
- load_raw
 - Clmg< T >, 321
- load_rgb
 - Clmg< T >, 317
- load_rgba
 - Clmg< T >, 318
- load_tiff
 - Clmg< T >, 318
 - ClmgList< T >, 423
- load_video
 - Clmg< T >, 323
 - ClmgList< T >, 422
- load_webp
 - Clmg< T >, 324
- load_yuv
 - Clmg< T >, 322
 - ClmgList< T >, 421
- log
 - Clmg< T >, 182
- log10
 - Clmg< T >, 182

log2
 CImg< T >, 182
 magnitude
 CImg< T >, 200
 map
 CImg< T >, 227
 matchpatch
 CImg< T >, 263
 matrix
 CImg< T >, 214
 max
 CImg< T >, 193
 max_min
 CImg< T >, 196
 CImgList< T >, 413
 maxabs
 CImg< T >, 195
 medcon_path
 cimg_library::cimg, 56
 min
 CImg< T >, 191
 min_max
 CImg< T >, 196
 CImgList< T >, 413
 minabs
 CImg< T >, 194
 minmod
 cimg_library::cimg, 45
 mirror
 CImg< T >, 233
 mod
 cimg_library::cimg, 45
 mouse_x
 CImgDisplay, 357
 mouse_y
 CImgDisplay, 357
 move
 CImgDisplay, 362
 move_to
 CImg< T >, 128
 CImgList< T >, 396
 MSE
 CImg< T >, 198
 mul
 CImg< T >, 188
 noise
 CImg< T >, 222
 norm
 CImg< T >, 223
 normalization
 CImgDisplay, 355
 normalize
 CImg< T >, 222, 223
 object3dtoCImg3d
 CImg< T >, 281
 offset
 CImg< T >, 157
 opening
 CImg< T >, 252
 openmp_mode
 cimg_library::cimg, 40
 operator bool
 CImgDisplay, 353
 operator CImg< T > *
 CImgList< T >, 398
 operator T*
 CImg< T >, 131
 operator!=
 CImg< T >, 150, 151
 operator<
 CImg< T >, 153
 CImgList< T >, 401
 operator<<
 CImg< T >, 147
 operator<<=
 CImg< T >, 146, 147
 operator>
 CImgList< T >, 400
 operator>>
 CImg< T >, 148, 149
 operator>>=
 CImg< T >, 148
 operator()
 CImg< T >, 130, 131
 CImgList< T >, 397, 398
 operator+
 CImg< T >, 135, 136
 CImgList< T >, 399
 operator++
 CImg< T >, 135
 operator+=
 CImg< T >, 133, 134
 operator,
 CImg< T >, 151, 153
 CImgList< T >, 400
 operator-
 CImg< T >, 137, 138
 operator--
 CImg< T >, 137
 operator-=
 CImg< T >, 136
 operator/
 CImg< T >, 141
 operator/=
 CImg< T >, 140
 operator=
 CImg< T >, 132, 133
 CImgDisplay, 352, 353
 CImgList< T >, 398, 399
 operator==
 CImg< T >, 149, 150
 operator%
 CImg< T >, 142
 operator%=

- CImg< T >, 141, 142
- operator&
 - CImg< T >, 143
- operator&=
 - CImg< T >, 142, 143
- operator*
 - CImg< T >, 139
- operator*=
 - CImg< T >, 138
- operator~
 - CImg< T >, 149
- operator^
 - CImg< T >, 146
- operator^=
 - CImg< T >, 145
- operator|
 - CImg< T >, 144, 145
- operator|=
 - CImg< T >, 144
- otsu
 - CImg< T >, 224
- output
 - cimg_library::cimg, 39
- paint
 - CImgDisplay, 369
- path_exists
 - cimg_library::cimg, 51
- permute_axes
 - CImg< T >, 234
- pixel_type
 - CImg< T >, 154
 - CImgList< T >, 401
- plane3d
 - CImg< T >, 279
- pow
 - CImg< T >, 188, 189
- print
 - CImg< T >, 327
 - CImgList< T >, 423
- project_matrix
 - CImg< T >, 209
- PSNR
 - CImg< T >, 198
- push_back
 - CImgList< T >, 417, 418
- push_front
 - CImgList< T >, 417, 418
- QR
 - CImg< T >, 209
- quantize
 - CImg< T >, 224
- rand
 - CImg< T >, 220
- released_key
 - CImgDisplay, 360
- remove
 - CImgList< T >, 415, 416
- render
 - CImgDisplay, 368
- resize
 - CImg< T >, 230–232
 - CImgDisplay, 362–364
- resize_doubleXY
 - CImg< T >, 232
- resize_object3d
 - CImg< T >, 269
- resize_tripleXY
 - CImg< T >, 233
- Retrieving Command Line Arguments., 25
- RGBtoXYZ
 - CImg< T >, 230
- rol
 - CImg< T >, 189, 190
- ror
 - CImg< T >, 190
- rotate
 - CImg< T >, 234–236
- rotate_object3d
 - CImg< T >, 268
- rotation_matrix
 - CImg< T >, 216
- round
 - CImg< T >, 220
 - cimg_library::cimg, 45
- row
 - CImg< T >, 240
- row_vector
 - CImg< T >, 211, 212
- save
 - CImg< T >, 330
 - CImgList< T >, 425
- save_analyze
 - CImg< T >, 335
- save_ascii
 - CImg< T >, 330
- save_bmp
 - CImg< T >, 331
- save_cimg
 - CImg< T >, 336
 - CImgList< T >, 427, 428
- save_cpp
 - CImg< T >, 331
- save_dlm
 - CImg< T >, 331
- save_empty_cimg
 - CImg< T >, 336, 337
 - CImgList< T >, 428, 429
- save_exr
 - CImg< T >, 337
- save_ffmpeg_external
 - CImg< T >, 341
 - CImgList< T >, 430
- save_gif_external
 - CImgList< T >, 426

save_graphicsmagick_external
 CImg< T >, 341

save_gzip_external
 CImg< T >, 341
 CImgList< T >, 430

save_imagemagick_external
 CImg< T >, 342

save_inr
 CImg< T >, 337

save_jpeg
 CImg< T >, 332

save_jxl
 CImg< T >, 332

save_magick
 CImg< T >, 332

save_medcon_external
 CImg< T >, 342

save_minc2
 CImg< T >, 335

save_off
 CImg< T >, 339, 340

save_other
 CImg< T >, 342

save_pandore
 CImg< T >, 338

save_pfm
 CImg< T >, 333

save_png
 CImg< T >, 333

save_pnk
 CImg< T >, 333

save_pnm
 CImg< T >, 333

save_raw
 CImg< T >, 338, 339

save_rgb
 CImg< T >, 334

save_rgba
 CImg< T >, 334

save_tiff
 CImg< T >, 334
 CImgList< T >, 429

save_video
 CImg< T >, 340
 CImgList< T >, 430

save_webp
 CImg< T >, 331

save_yuv
 CImg< T >, 339
 CImgList< T >, 426, 427

screenshot
 CImgDisplay, 352, 369

select
 CImg< T >, 314

sequence
 CImg< T >, 203, 216

set_button
 CImgDisplay, 366

set_fullscreen
 CImgDisplay, 365

set_key
 CImgDisplay, 367

set_linear_atX
 CImg< T >, 170

set_linear_atXY
 CImg< T >, 171

set_linear_atXYZ
 CImg< T >, 171

set_matrix_at
 CImg< T >, 202

set_mouse
 CImgDisplay, 366

set_normalization
 CImgDisplay, 364

set_tensor_at
 CImg< T >, 202

set_title
 CImgDisplay, 364

set_vector_at
 CImg< T >, 202

set_wheel
 CImgDisplay, 367

Setting Environment Variables, 16

sharpen
 CImg< T >, 261

shift
 CImg< T >, 233

shift_object3d
 CImg< T >, 268, 269

show
 CImgDisplay, 362

show_mouse
 CImgDisplay, 365

sign
 CImg< T >, 183

sin
 CImg< T >, 184

sinc
 CImg< T >, 184

sinh
 CImg< T >, 185

size
 CImg< T >, 155
 CImgList< T >, 401

sleep
 cimg_library::cimg, 44

snapshot
 CImgDisplay, 369

solve
 CImg< T >, 204

solve_tridiagonal
 CImg< T >, 204

sort
 CImg< T >, 207

spectrum
 CImg< T >, 155

- sphere3d
 - Clmg< T >, 280
- split
 - ClmgList< T >, 417
- split_filename
 - cimg_library::cimg, 57
- sqr
 - Clmg< T >, 181
- sqrt
 - Clmg< T >, 181
- strcasecmp
 - cimg_library::cimg, 47
- streamline
 - Clmg< T >, 241
- strellipse
 - cimg_library::cimg, 48
- string
 - Clmg< T >, 211
- strncasecmp
 - cimg_library::cimg, 47
- strpare
 - cimg_library::cimg, 48
- structure_tensors
 - Clmg< T >, 262
- strunescape
 - cimg_library::cimg, 49
- strwindows_reserved
 - cimg_library::cimg, 49
- SVD
 - Clmg< T >, 208
- swap
 - Clmg< T >, 129
 - ClmgList< T >, 396
- symmetric_eigen
 - Clmg< T >, 205
- system
 - cimg_library::cimg, 42
- tan
 - Clmg< T >, 184
- tanh
 - Clmg< T >, 185
- temporary_path
 - cimg_library::cimg, 56
- tensor
 - Clmg< T >, 215
- texturize_object3d
 - Clmg< T >, 271
- The Clmg Library, 1
- threshold
 - Clmg< T >, 225
- tic
 - cimg_library::cimg, 44
- time
 - cimg_library::cimg, 43
- title
 - ClmgDisplay, 356
- toc
 - cimg_library::cimg, 44
- toggle_fullscreen
 - ClmgDisplay, 365
- torus3d
 - Clmg< T >, 279
- transpose
 - Clmg< T >, 203
- Tutorial : Getting Started., 17
- unroll
 - Clmg< T >, 234
- Using Display Windows., 23
- Using Image Loops., 19
- value_string
 - Clmg< T >, 171
- value_type
 - Clmg< T >, 115
 - ClmgList< T >, 383
- vanvliet
 - Clmg< T >, 253
- variance
 - Clmg< T >, 196
- variance_mean
 - Clmg< T >, 197
- variance_noise
 - Clmg< T >, 197
- vector
 - Clmg< T >, 213
- wait
 - cimg_library::cimg, 44
 - ClmgDisplay, 368
- warn
 - cimg_library::cimg, 42
- warp
 - Clmg< T >, 236
- watershed
 - Clmg< T >, 252
- wave
 - cimg_library::cimg, 46
- wget_path
 - cimg_library::cimg, 56
- wheel
 - ClmgDisplay, 358
- width
 - Clmg< T >, 154
 - ClmgDisplay, 355
 - ClmgList< T >, 401
- window_height
 - ClmgDisplay, 356
- window_width
 - ClmgDisplay, 356
- window_x
 - ClmgDisplay, 357
- window_y
 - ClmgDisplay, 357
- XYZtoRGB
 - Clmg< T >, 230