The CImg Library

1.5.0

Generated by Doxygen 1.7.6.1

Wed May 16 2012 11:38:30

Contents

1 Main Page **1**

2 Module Index **3**

2.1 Modules 3

3 Namespace Index **5**

3.1 Namespace List 5

4 Class Index **7**

4.1 Class Hierarchy 7

5 Class Index **9**

5.1 Class List 9

6 Module 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 ? 13

6.1.4 What's next ? 14

6.2 CImg$<$T$>$: The image structure. 15

6.2.1 Structure overview 15

6.2.2 Image construction/destruction/copy 15

6.2.3 Image methods 15

6.2.4 Shared images 15

6.2.5 Low-level structure 15

6.3 CImgList$<$T$>$: The image list structure. 16

6.3.1 Structure overview 16
6.3.2 Image list construction/destruction/copy 16
6.3.3 Image methods 16
6.3.4 Low-level structure 16
6.4 CImgDisplay : The image display structure. 17
6.4.1 Structure overview 17
6.4.2 Image display construction/destruction/copy 17
6.4.3 Image methods 17
6.4.4 Low-level structure 17
6.5 CImgException : The library exception structure. 18
6.5.1 Structure overview 18
6.6 FAQ : Frequently Asked Questions. 19
6.6.1 FAQ Summary 19
6.6.2 1. General information and availability 19
6.6.2.1 1.1. What is the CImg Library ? 19
6.6.2.2 1.2. What platforms are supported ? 19
6.6.2.3 1.3. How is CImg distributed ? 20
6.6.2.4 1.4. What kind of people are concerned by CImg ? . . 20
6.6.2.5 1.5. What are the specificities of the CeCILL license ? 20
6.6.2.6 1.6. Who is behind CImg ? 21
6.6.3 2. C++ related questions 21
6.6.3.1 2.1 What is the level of C++ knowledge needed to use CImg ? 21
6.6.3.2 2.2 How to use CImg in my own C++ program ? . . . 21
6.6.3.3 2.3 Why is CImg entirely contained in a single header file ? 21
6.7 Setting Environment Variables 24
6.8 How to use CImg library with Visual C++ 2005 Express Edition ?. 26
6.8.1 How to use CImg library with Visual C++ 2005 Express Edition ? 26
6.9 Tutorial : Getting Started. 27
6.10 Using Drawing Functions. 30
6.10.1 Using Drawing Functions. 30
6.11 Using Image Loops. 31
6.11.1 Loops over the pixel buffer 31

6.11.2 Loops over image dimensions 31
6.11.3 Loops over interior regions and borders. 33
6.11.4 Loops using neighborhoods. 34
6.11.4.1 Neighborhood-based loops for 2D images 34
6.11.4.2 Neighborhood-based loops for 3D images 34
6.11.4.3 Defining neighborhoods 35
6.11.4.4 Using alternate variable names 35
6.11.4.5 Example codes 36
6.12 Using Display Windows. 37
6.13 How pixel data are stored with CImg. 38
6.14 Files IO in CImg. 40
6.15 Retrieving Command Line Arguments. 41
6.15.1 The cimg_usage() macro 41
6.15.2 The cimg_help() macro 41
6.15.3 The cimg_option() macro 41
6.15.4 Example of use 42
6.15.5 How to learn more about command line options ? 43

7 Namespace Documentation 45
7.1 cimg_library Namespace Reference 45
7.1.1 Detailed Description 45
7.2 cimg_library::cimg Namespace Reference 46
7.2.1 Detailed Description 55
7.2.2 Function Documentation 55
7.2.2.1 output 55
7.2.2.2 info 55
7.2.2.3 exception_mode 56
7.2.2.4 exception_mode 56
7.2.2.5 eval 56
7.2.2.6 warn 57
7.2.2.7 system 57
7.2.2.8 endianness 58
7.2.2.9 invert_endianness 58
7.2.2.10 invert_endianness 58

7.2.2.11 time 59
7.2.2.12 tic 59
7.2.2.13 toc 59
7.2.2.14 sleep 60
7.2.2.15 wait 60
7.2.2.16 rand 60
7.2.2.17 crand 61
7.2.2.18 grand 61
7.2.2.19 prand 61
7.2.2.20 mod 61
7.2.2.21 minmod 61
7.2.2.22 round 62
7.2.2.23 atof 62
7.2.2.24 strncasecmp 62
7.2.2.25 strcasecmp 63
7.2.2.26 strpare 63
7.2.2.27 strescape 64
7.2.2.28 fopen 64
7.2.2.29 fclose 65
7.2.2.30 temporary_path 65
7.2.2.31 imagemagick_path 65
7.2.2.32 graphicsmagick_path 66
7.2.2.33 medcon_path 66
7.2.2.34 ffmpeg_path 66
7.2.2.35 gzip_path 67
7.2.2.36 gunzip_path 67
7.2.2.37 dcraw_path 67
7.2.2.38 wget_path 68
7.2.2.39 curl_path 68
7.2.2.40 file_type 68
7.2.2.41 fread 68
7.2.2.42 fwrite 69
7.2.2.43 load_network_external 69
7.2.2.44 dialog 70

8 Class Documentation **71**

8.1 CImg< T > Struct Template Reference 71

8.1.1 Detailed Description 135

8.1.2 Member Typedef Documentation 137

8.1.2.1 iterator 137

8.1.2.2 const_iterator 138

8.1.2.3 value_type 138

8.1.3 Constructor & Destructor Documentation 138

8.1.3.1 ∼CImg 138

8.1.3.2 CImg 139

8.1.3.3 CImg 140

8.1.3.4 CImg 140

8.1.3.5 CImg 141

8.1.3.6 CImg 142

8.1.3.7 CImg 143

8.1.3.8 CImg 144

8.1.3.9 CImg 145

8.1.3.10 CImg 146

8.1.3.11 CImg 146

8.1.3.12 CImg 147

8.1.3.13 CImg 148

8.1.3.14 CImg 148

8.1.4 Member Function Documentation 149

8.1.4.1 assign 149

8.1.4.2 assign 149

8.1.4.3 assign 150

8.1.4.4 assign 150

8.1.4.5 assign 151

8.1.4.6 assign 151

8.1.4.7 assign 152

8.1.4.8 assign 152

8.1.4.9 assign 153

8.1.4.10 assign 153

8.1.4.11 assign 154

8.1.4.12 assign 154
8.1.4.13 assign 154
8.1.4.14 assign 155
8.1.4.15 clear 155
8.1.4.16 move_to 155
8.1.4.17 move_to 156
8.1.4.18 swap 157
8.1.4.19 empty 157
8.1.4.20 operator() 157
8.1.4.21 operator() 158
8.1.4.22 operator T ∗ 159
8.1.4.23 operator= 159
8.1.4.24 operator= 160
8.1.4.25 operator= 161
8.1.4.26 operator= 161
8.1.4.27 operator+= 161
8.1.4.28 operator+= 162
8.1.4.29 operator+= 162
8.1.4.30 operator++ 163
8.1.4.31 operator++ 163
8.1.4.32 operator+ 164
8.1.4.33 operator+ 164
8.1.4.34 operator+ 164
8.1.4.35 operator+ 165
8.1.4.36 operator-= 165
8.1.4.37 operator-= 165
8.1.4.38 operator-= 165
8.1.4.39 operator-- 165
8.1.4.40 operator-- 165
8.1.4.41 operator- 165
8.1.4.42 operator- 166
8.1.4.43 operator- 166
8.1.4.44 operator- 166
8.1.4.45 operator∗= 166

8.1.4.46 operator∗= 167
8.1.4.47 operator∗= 167
8.1.4.48 operator∗ 167
8.1.4.49 operator∗ 167
8.1.4.50 operator∗ 168
8.1.4.51 operator/= 168
8.1.4.52 operator/= 168
8.1.4.53 operator/= 168
8.1.4.54 operator/ 168
8.1.4.55 operator/ 169
8.1.4.56 operator/ 169
8.1.4.57 operator%= 169
8.1.4.58 operator%= 169
8.1.4.59 operator%= 169
8.1.4.60 operator% 169
8.1.4.61 operator% 170
8.1.4.62 operator% 170
8.1.4.63 operator&= 170
8.1.4.64 operator&= 170
8.1.4.65 operator&= 170
8.1.4.66 operator& 170
8.1.4.67 operator& 171
8.1.4.68 operator& 171
8.1.4.69 operator\|= 171
8.1.4.70 operator\|= 171
8.1.4.71 operator\|= 171
8.1.4.72 operator\| 171
8.1.4.73 operator\| 171
8.1.4.74 operator\| 172
8.1.4.75 operator^= 172
8.1.4.76 operator^= 172
8.1.4.77 operator^= 172
8.1.4.78 operator^ 172
8.1.4.79 operator^ 173

8.1.4.80 operator^ . . . 173
8.1.4.81 operator<<= . . . 173
8.1.4.82 operator<<= . . . 173
8.1.4.83 operator<<= . . . 173
8.1.4.84 operator<< . . . 173
8.1.4.85 operator<< . . . 173
8.1.4.86 operator<< . . . 174
8.1.4.87 operator>>= . . . 174
8.1.4.88 operator>>= . . . 174
8.1.4.89 operator>>= . . . 174
8.1.4.90 operator>> . . . 174
8.1.4.91 operator>> . . . 174
8.1.4.92 operator>> . . . 174
8.1.4.93 operator∼ . . . 175
8.1.4.94 operator== . . . 175
8.1.4.95 operator== . . . 175
8.1.4.96 operator== . . . 175
8.1.4.97 operator!= . . . 176
8.1.4.98 operator!= . . . 176
8.1.4.99 operator!= . . . 177
8.1.4.100 operator, . . . 177
8.1.4.101 operator, . . . 178
8.1.4.102 operator< . . . 179
8.1.4.103 pixel_type . . . 179
8.1.4.104 width . . . 179
8.1.4.105 height . . . 180
8.1.4.106 depth . . . 180
8.1.4.107 spectrum . . . 181
8.1.4.108 size . . . 181
8.1.4.109 data . . . 182
8.1.4.110 data . . . 182
8.1.4.111 offset . . . 183
8.1.4.112 begin . . . 183
8.1.4.113 end . . . 184

8.1.4.114 front 184
8.1.4.115 back 185
8.1.4.116 at 185
8.1.4.117 at 185
8.1.4.118 atX 186
8.1.4.119 atX 187
8.1.4.120 atXY 188
8.1.4.121 atXY 188
8.1.4.122 atXYZ 188
8.1.4.123 atXYZ 188
8.1.4.124 atXYZC 188
8.1.4.125 atXYZC 189
8.1.4.126 linear_atX 189
8.1.4.127 linear_atX 190
8.1.4.128 linear_atXY 190
8.1.4.129 linear_atXY 190
8.1.4.130 linear_atXYZ 191
8.1.4.131 linear_atXYZ 191
8.1.4.132 linear_atXYZC 191
8.1.4.133 linear_atXYZC 191
8.1.4.134 cubic_atX 192
8.1.4.135 cubic_atX 192
8.1.4.136 cubic_atX 193
8.1.4.137 cubic_atX 193
8.1.4.138 cubic_atXY 193
8.1.4.139 cubic_atXY 194
8.1.4.140 cubic_atXY 194
8.1.4.141 cubic_atXY 194
8.1.4.142 cubic_atXYZ 194
8.1.4.143 cubic_atXYZ 195
8.1.4.144 cubic_atXYZ 195
8.1.4.145 cubic_atXYZ 195
8.1.4.146 set_linear_atXY 195
8.1.4.147 set_linear_atXYZ 196

8.1.4.148 value_string 196
8.1.4.149 is_shared 197
8.1.4.150 is_empty 197
8.1.4.151 is_inf 197
8.1.4.152 is_nan 198
8.1.4.153 is_sameXY 198
8.1.4.154 is_sameXY 198
8.1.4.155 is_sameXY 198
8.1.4.156 is_sameXZ 198
8.1.4.157 is_sameXZ 198
8.1.4.158 is_sameXC 198
8.1.4.159 is_sameXC 199
8.1.4.160 is_sameYZ 199
8.1.4.161 is_sameYZ 199
8.1.4.162 is_sameYC 199
8.1.4.163 is_sameYC 199
8.1.4.164 is_sameZC 199
8.1.4.165 is_sameZC 200
8.1.4.166 is_sameXYZ 200
8.1.4.167 is_sameXYZ 200
8.1.4.168 is_sameXYC 200
8.1.4.169 is_sameXYC 200
8.1.4.170 is_sameXZC 200
8.1.4.171 is_sameXZC 200
8.1.4.172 is_sameYZC 201
8.1.4.173 is_sameYZC 201
8.1.4.174 is_sameXYZC 201
8.1.4.175 is_sameXYZC 201
8.1.4.176 containsXYZC 201
8.1.4.177 contains 202
8.1.4.178 contains 203
8.1.4.179 contains 203
8.1.4.180 contains 203
8.1.4.181 contains 203

8.1.4.182 is_overlapped 203
8.1.4.183 is_object3d 204
8.1.4.184 is_CImg3d 204
8.1.4.185 sqr 205
8.1.4.186 sqrt 206
8.1.4.187 exp 206
8.1.4.188 log 206
8.1.4.189 log2 207
8.1.4.190 log10 207
8.1.4.191 abs 208
8.1.4.192 sign 208
8.1.4.193 cos 208
8.1.4.194 sin 209
8.1.4.195 sinc 209
8.1.4.196 tan 210
8.1.4.197 cosh 210
8.1.4.198 sinh 210
8.1.4.199 tanh 211
8.1.4.200 acos 211
8.1.4.201 asin 212
8.1.4.202 atan 212
8.1.4.203 atan2 212
8.1.4.204 mul 213
8.1.4.205 div 214
8.1.4.206 pow 214
8.1.4.207 pow 214
8.1.4.208 pow 215
8.1.4.209 rol 215
8.1.4.210 rol 215
8.1.4.211 rol 215
8.1.4.212 ror 216
8.1.4.213 ror 216
8.1.4.214 ror 216
8.1.4.215 variance 216

8.1.4.216 variance_noise 217
8.1.4.217 eval 217
8.1.4.218 dijkstra 218
8.1.4.219 dijkstra 218
8.1.4.220 streamline 218
8.1.4.221 fill 219
8.1.4.222 round 219
8.1.4.223 noise 219
8.1.4.224 normalize 220
8.1.4.225 normalize 220
8.1.4.226 norm 221
8.1.4.227 cut 221
8.1.4.228 quantize 222
8.1.4.229 threshold 222
8.1.4.230 histogram 223
8.1.4.231 equalize 224
8.1.4.232 index 224
8.1.4.233 map 225
8.1.4.234 label 225
8.1.4.235 RGBtoHSI 226
8.1.4.236 RGBtoBayer 226
8.1.4.237 resize 226
8.1.4.238 resize_doubleXY 227
8.1.4.239 resize_tripleXY 227
8.1.4.240 shift 227
8.1.4.241 permute_axes 228
8.1.4.242 rotate 228
8.1.4.243 rotate 228
8.1.4.244 crop 229
8.1.4.245 crop 229
8.1.4.246 crop 229
8.1.4.247 crop 230
8.1.4.248 get_split 230
8.1.4.249 correlate 230

8.1.4.250 convolve . . . 230
8.1.4.251 deriche . . . 231
8.1.4.252 blur . . . 231
8.1.4.253 blur_anisotropic . . . 231
8.1.4.254 blur_bilateral . . . 232
8.1.4.255 get_gradient . . . 232
8.1.4.256 displacement . . . 233
8.1.4.257 distance . . . 233
8.1.4.258 distance . . . 233
8.1.4.259 haar . . . 233
8.1.4.260 haar . . . 233
8.1.4.261 get_elevation3d . . . 234
8.1.4.262 get_isoline3d . . . 234
8.1.4.263 get_isosurface3d . . . 235
8.1.4.264 box3d . . . 235
8.1.4.265 cone3d . . . 236
8.1.4.266 cylinder3d . . . 236
8.1.4.267 torus3d . . . 237
8.1.4.268 plane3d . . . 237
8.1.4.269 sphere3d . . . 238
8.1.4.270 ellipsoid3d . . . 238
8.1.4.271 draw_point . . . 239
8.1.4.272 draw_line . . . 239
8.1.4.273 draw_line . . . 240
8.1.4.274 draw_line . . . 241
8.1.4.275 draw_arrow . . . 241
8.1.4.276 draw_spline . . . 242
8.1.4.277 draw_spline . . . 243
8.1.4.278 draw_spline . . . 243
8.1.4.279 draw_triangle . . . 244
8.1.4.280 draw_triangle . . . 244
8.1.4.281 draw_triangle . . . 245
8.1.4.282 draw_triangle . . . 246
8.1.4.283 draw_triangle . . . 246

8.1.4.284 draw_rectangle 247
8.1.4.285 draw_rectangle 248
8.1.4.286 draw_rectangle 248
8.1.4.287 draw_circle 248
8.1.4.288 draw_circle 249
8.1.4.289 draw_ellipse 249
8.1.4.290 draw_ellipse 249
8.1.4.291 draw_ellipse 250
8.1.4.292 draw_ellipse 250
8.1.4.293 draw_image 250
8.1.4.294 draw_image 251
8.1.4.295 draw_text 251
8.1.4.296 draw_text 252
8.1.4.297 draw_quiver 252
8.1.4.298 draw_quiver 253
8.1.4.299 draw_axis 253
8.1.4.300 draw_graph 254
8.1.4.301 draw_fill 255
8.1.4.302 draw_fill 255
8.1.4.303 draw_fill 255
8.1.4.304 draw_plasma 256
8.1.4.305 draw_gaussian 256
8.1.4.306 draw_gaussian 256
8.1.4.307 draw_gaussian 257
8.1.4.308 draw_gaussian 257
8.1.4.309 draw_gaussian 257
8.1.4.310 draw_object3d 258
8.1.4.311 load 258
8.1.4.312 load_tiff 258
8.1.4.313 print 259
8.1.4.314 save 259
8.1.4.315 save_tiff 259
8.1.4.316 save_graphicsmagick_external 260
8.1.4.317 save_imagemagick_external 260

8.2 CImgDisplay Struct Reference 260
8.2.1 Detailed Description 267
8.2.2 Constructor & Destructor Documentation 268
8.2.2.1 ∼CImgDisplay 268
8.2.2.2 CImgDisplay 268
8.2.2.3 CImgDisplay 269
8.2.2.4 CImgDisplay 269
8.2.2.5 CImgDisplay 270
8.2.2.6 CImgDisplay 270
8.2.3 Member Function Documentation 270
8.2.3.1 assign 270
8.2.3.2 assign 271
8.2.3.3 assign 271
8.2.3.4 assign 271
8.2.3.5 assign 271
8.2.3.6 empty 272
8.2.3.7 operator= 272
8.2.3.8 operator= 272
8.2.3.9 operator= 272
8.2.3.10 operator bool 273
8.2.3.11 is_empty 273
8.2.3.12 is_closed 273
8.2.3.13 is_resized 273
8.2.3.14 is_moved 274
8.2.3.15 is_event 274
8.2.3.16 is_fullscreen 274
8.2.3.17 is_key 274
8.2.3.18 is_key 274
8.2.3.19 is_key 275
8.2.3.20 is_key_sequence 275
8.2.3.21 is_keyESC 276
8.2.3.22 screen_width 276
8.2.3.23 screen_height 277
8.2.3.24 width 277

8.2.3.25 height 277
8.2.3.26 normalization 277
8.2.3.27 title 278
8.2.3.28 window_width 278
8.2.3.29 window_height 279
8.2.3.30 window_x 279
8.2.3.31 window_y 279
8.2.3.32 mouse_x 279
8.2.3.33 mouse_y 280
8.2.3.34 button 280
8.2.3.35 wheel 281
8.2.3.36 key 281
8.2.3.37 released_key 282
8.2.3.38 keycode 283
8.2.3.39 frames_per_second 283
8.2.3.40 display 283
8.2.3.41 display 284
8.2.3.42 show 284
8.2.3.43 close 284
8.2.3.44 move 285
8.2.3.45 resize 285
8.2.3.46 resize 285
8.2.3.47 resize 286
8.2.3.48 resize 286
8.2.3.49 set_normalization 287
8.2.3.50 set_title 287
8.2.3.51 set_fullscreen 287
8.2.3.52 toggle_fullscreen 288
8.2.3.53 show_mouse 288
8.2.3.54 hide_mouse 289
8.2.3.55 set_mouse 289
8.2.3.56 set_button 289
8.2.3.57 set_button 289
8.2.3.58 set_wheel 290

8.2.3.59 set_wheel 290
8.2.3.60 set_key 290
8.2.3.61 set_key 291
8.2.3.62 flush 291
8.2.3.63 wait 291
8.2.3.64 render 292
8.2.3.65 paint 292
8.2.3.66 snapshot 292
8.3 CImgException Struct Reference 293
8.3.1 Detailed Description 293
8.4 CImgList< T > Struct Template Reference 294
8.4.1 Detailed Description 312
8.4.2 Member Typedef Documentation 312
8.4.2.1 iterator 312
8.4.2.2 const_iterator 313
8.4.2.3 value_type 313
8.4.3 Constructor & Destructor Documentation 313
8.4.3.1 ∼CImgList 313
8.4.3.2 CImgList 314
8.4.3.3 CImgList 314
8.4.3.4 CImgList 314
8.4.3.5 CImgList 315
8.4.3.6 CImgList 315
8.4.3.7 CImgList 316
8.4.3.8 CImgList 316
8.4.3.9 CImgList 317
8.4.3.10 CImgList 317
8.4.3.11 CImgList 317
8.4.3.12 CImgList 318
8.4.3.13 CImgList 318
8.4.3.14 CImgList 319
8.4.3.15 CImgList 319
8.4.3.16 CImgList 320
8.4.3.17 CImgList 320

8.4.3.18 CImgList 321
8.4.3.19 CImgList 321
8.4.3.20 CImgList 321
8.4.4 Member Function Documentation 322
8.4.4.1 get_shared 322
8.4.4.2 get_shared 322
8.4.4.3 assign 322
8.4.4.4 clear 322
8.4.4.5 assign 323
8.4.4.6 assign 323
8.4.4.7 assign 323
8.4.4.8 assign 324
8.4.4.9 assign 324
8.4.4.10 assign 325
8.4.4.11 assign 325
8.4.4.12 assign 325
8.4.4.13 assign 326
8.4.4.14 assign 326
8.4.4.15 assign 327
8.4.4.16 assign 327
8.4.4.17 assign 328
8.4.4.18 assign 328
8.4.4.19 assign 328
8.4.4.20 assign 329
8.4.4.21 move_to 329
8.4.4.22 move_to 329
8.4.4.23 swap 330
8.4.4.24 empty 330
8.4.4.25 operator() 330
8.4.4.26 operator() 331
8.4.4.27 operator() 331
8.4.4.28 operator CImg< T > * 331
8.4.4.29 operator= 332
8.4.4.30 operator= 332

8.4.4.31 operator= 332
8.4.4.32 operator= 333
8.4.4.33 operator+ 333
8.4.4.34 operator, ... 333
8.4.4.35 operator, ... 334
8.4.4.36 operator> 334
8.4.4.37 operator< 334
8.4.4.38 pixel_type 335
8.4.4.39 width 335
8.4.4.40 size 335
8.4.4.41 data 335
8.4.4.42 data 336
8.4.4.43 begin 336
8.4.4.44 end 336
8.4.4.45 front 336
8.4.4.46 back 336
8.4.4.47 at 337
8.4.4.48 atNXYZC 337
8.4.4.49 atNXYZC 337
8.4.4.50 atNXYZ 338
8.4.4.51 atNXYZ 338
8.4.4.52 atNXY 339
8.4.4.53 atNXY 339
8.4.4.54 atNX 340
8.4.4.55 atNX 340
8.4.4.56 atN 341
8.4.4.57 atN 341
8.4.4.58 value_string 341
8.4.4.59 is_empty 342
8.4.4.60 is_sameXYZC 342
8.4.4.61 is_sameNXYZC 342
8.4.4.62 containsNXYZC 343
8.4.4.63 containsN 343
8.4.4.64 contains 343

8.4.4.65 contains 344
8.4.4.66 contains 344
8.4.4.67 contains 344
8.4.4.68 contains 345
8.4.4.69 contains 345
8.4.4.70 contains 345
8.4.4.71 contains 345
8.4.4.72 min 346
8.4.4.73 max 346
8.4.4.74 min_max 346
8.4.4.75 min_max 346
8.4.4.76 max_min 347
8.4.4.77 insert 347
8.4.4.78 insert 347
8.4.4.79 insert 347
8.4.4.80 insert 348
8.4.4.81 insert 348
8.4.4.82 remove 348
8.4.4.83 remove 348
8.4.4.84 remove 349
8.4.4.85 images 349
8.4.4.86 get_shared_images 349
8.4.4.87 get_append 349
8.4.4.88 split 350
8.4.4.89 push_back 350
8.4.4.90 push_front 350
8.4.4.91 push_back 350
8.4.4.92 push_front 350
8.4.4.93 pop_back 350
8.4.4.94 pop_front 351
8.4.4.95 erase 351
8.4.4.96 get_select 351
8.4.4.97 get_select 351
8.4.4.98 load 352

8.4.4.99 load_cimg 352
8.4.4.100 load_cimg 352
8.4.4.101 load_cimg 352
8.4.4.102 load_cimg 353
8.4.4.103 load_parrec 353
8.4.4.104 load_yuv 353
8.4.4.105 load_yuv 354
8.4.4.106 load_ffmpeg 354
8.4.4.107 load_ffmpeg_external 354
8.4.4.108 load_gzip_external 355
8.4.4.109 load_off 355
8.4.4.110 load_tiff 355
8.4.4.111 print 355
8.4.4.112 display 356
8.4.4.113 display 356
8.4.4.114 display 356
8.4.4.115 save 357
8.4.4.116 save_ffmpeg 357
8.4.4.117 save_yuv 357
8.4.4.118 save_yuv 358
8.4.4.119 save_cimg 358
8.4.4.120 save_cimg 358
8.4.4.121 save_cimg 358
8.4.4.122 save_cimg 359
8.4.4.123 save_empty_cimg 359
8.4.4.124 save_empty_cimg 359
8.4.4.125 save_tiff 360
8.4.4.126 save_gzip_external 360
8.4.4.127 save_ffmpeg_external 360
8.4.4.128 crop_font 360
8.4.4.129 get_crop_font 360
8.4.4.130 font 361
8.4.4.131 FFT 361
8.4.4.132 FFT 361

8.4.4.133 reverse_object3d 361

Chapter 1

Main Page

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. If you have downloaded the CImg package, you actually have a local copy of these pages in the `CImg/html/reference/` directory.

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

Module Index

2.1 Modules

Here is a list of all modules:

CImg Library Overview 11
CImg<T> : The image structure. 15
CImgList<T> : The image list structure. 16
CImgDisplay : The image display structure. 17
CImgException : The library exception structure. 18
FAQ : Frequently Asked Questions. 19
Setting Environment Variables 24
How to use CImg library with Visual C++ 2005 Express Edition ?. 26
Tutorial : Getting Started. 27
Using Drawing Functions. 30
Using Image Loops. 31
Using Display Windows. 37
How pixel data are stored with CImg. 38
Files IO in CImg. 40
Retrieving Command Line Arguments. 41

Chapter 3

Namespace Index

3.1 Namespace List

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

cimg_library
Contains *all classes and functions* of the CImg library 45
cimg_library::cimg
Contains *low-level* functions and variables of the CImg Library . . . 46

Chapter 4

Class Index

4.1 Class Hierarchy

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

CImg< T > 71
CImgDisplay 260
CImgException 293
CImgList< T > 294

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), each pixel being of type `T` 71

CImgDisplay
Allow to create windows, display images on them and manage user events (keyboard, mouse and windows events) 260

CImgException
Instances of `CImgException` are thrown when errors are encountered in a `CImg` function call 293

CImgList< T >
Represent a list of images CImg<T> 294

Chapter 6

Module 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 stuffs 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 stuffs.
- 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<T> 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<T> 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 catched with a bloc `try { ..} catch (CImgException) { .. }`. 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 a 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 provide 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 avoid to handle complex dependancies and problems with library compatibility. The only thing you need is a (quite modern) C++ compiler :

- **Microsoft Visual C++ 6.0, Visual Studio.NET and Visual Express Edition** : - Use 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 C-Img-based program with g++, on Windows :

  ```
  g++ -o hello_word.exe hello_word.cpp -O2 -lgdi32
  ```

- **g++ (Linux version)** : Use the following command to compile a CImg-based program with g++, on Linux :

  ```
  g++ -o hello_word.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_word.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_word.exe hello_world.cpp -O2 -lm -lpthread -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 another compilers and encounter problems, please `write me` since maintaining compatibility is one of the priority of the CImg Library. Nevertheless, old compilers that does not respect the C++ norm 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 CImg< T > : The image structure.

Description of the CImg< T > structure

6.2.1 Structure overview

6.2.2 Image construction/destruction/copy

6.2.3 Image methods

6.2.4 Shared images

6.2.5 Low-level structure

6.3 CImgList< T > : The image list structure.

Description of the CImgList< T > structure

6.3.1 Structure overview

6.3.2 Image list construction/destruction/copy

6.3.3 Image methods

6.3.4 Low-level structure

6.4 CImgDisplay : The image display structure.

Description of the CImgDisplay structure

6.4.1 Structure overview

6.4.2 Image display construction/destruction/copy

6.4.3 Image methods

6.4.4 Low-level structure

6.5 CImgException : The library exception structure.

Description of the CImgException structure

6.5.1 Structure overview

6.6 FAQ : Frequently Asked Questions.

6.6.1 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 ?

6.6.2 1. General information and availability

6.6.2.1 1.1. What is the CImg Library ?

The CImg Library is an *open-source C++ toolkit 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 stuffs in C++ : Just include the header file *CImg.h*, and you are ready to handle images in your C++ programs.

6.6.2.2 1.2. What platforms are supported ?

CImg 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 CImg Library is compiled under these different configurations :

- PC Linux 32 bits, with g++.
- PC Windows 32 bits, with Visual C++ 6.0.
- PC Windows 32 bits, with Visual C++ Express Edition.

- Sun SPARC Solaris 32 bits, with g++.
- Mac PPC with OS X and g++.

CImg 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).

6.6.2.3 1.3. How is CImg distributed ?

The CImg Library is freely distributed as a complete .zip compressed package, hosted at the `Sourceforge servers`.

The package is distributed under the `CeCILL license`.

This package contains :

- The main library file `CImg.h` (C++ header file).
- Several C++ source code showing `examples of using CImg`.
- A complete library documentation, in `HTML` and `PDF` formats.
- Additional `library plug-ins` that can be used to extend library capabilities for specific uses.

The CImg 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 CImg package is released approximately every three months.

6.6.2.4 1.4. What kind of people are concerned by CImg ?

The CImg 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.

6.6.2.5 1.5. What are the specificities of the CeCILL license ?

The `CeCILL license` governs the use of the CImg Library. This is an *open-source* license which gives you rights to access, use, modify and redistribute the source code, under certains conditions. There are two different variants of the CeCILL license used in CImg (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* `CImg.h`. Basically, this license allows to use CImg.h in a closed-source product without forcing you to redistribute the entire software source code. Anyway, if one modifies the CImg.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 CImg 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 CImg Library.

6.6.2.6 1.6. Who is behind CImg ?

CImg has been started by David Tschumperle at the beginning of his PhD thesis, in October 1999. He is still the main coordinator of the project. Since the first release at Sourceforge, 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.

6.6.3 2. C++ related questions

6.6.3.1 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.

6.6.3.2 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;
```

6.6.3.3 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 instanciates 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 undoubtly 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. Think seriously about it, and if you have a better solution to provide, let me know so we can discuss about it.

6.7 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) or **2** (Windows-GDI display). 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` if 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.

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

6.8.1 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 liste "Show directories for", choose "include files", and add C: Files Platform SDK (adapt if needed)
  - 4. In the select liste "Show directories for", choose "library files", and add C: Files Platform SDK (adapt if needed) Edit file C: Files Visual Studio 8\V-C\VCProjectDefaults\corewin_express.vsprops (adapt if needed)
  - 6. 7. Remplace 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

6.9 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 detailled 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 than 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,v) 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 `C-Img_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.

6.10 Using Drawing Functions.

6.10.1 Using Drawing Functions.

This section tells more about drawing features in CImg images. Drawing functions list can be found in `the CImg functions list` (section **Drawing** Functions), and are all defined on a common basis. Here are the important points to understand before using drawing functions :

- Drawing is performed on the instance image. Drawing functions parameters are defined as *const* variables and return a reference to the current instance `(*this)`, so that drawing functions can be pipelined (see examples below). - Drawing is usually done in 2D color images but can be performed in 3D images with any vector-valued dimension, and with any possible pixel type.

- A color parameter is always needed to draw features in an image. The color must be defined as a C-style array whose dimension is at least

6.11 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.

6.11.1 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 end of the buffer (last pixel) till the beginning of the buffer (first 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_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);'
```

6.11.2 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,v)** : equivalent to : `for (int v = 0; v<img.-spectrum(); v++)`.

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_for-X(img,x)`.
- **cimg_forXZ(img,x,z)** : equivalent to : `cimg_forZ(img,z) cimg_for-X(img,x)`.
- **cimg_forYZ(img,y,z)** : equivalent to : `cimg_forZ(img,z) cimg_for-Y(img,y)`.
- **cimg_forXC(img,x,v)** : equivalent to : `cimg_forC(img,v) cimg_for-X(img,x)`.
- **cimg_forYC(img,y,v)** : equivalent to : `cimg_forC(img,v) cimg_for-Y(img,y)`.
- **cimg_forZC(img,z,v)** : equivalent to : `cimg_forC(img,v) cimg_for-Z(img,z)`.
- **cimg_forXYZ(img,x,y,z)** : equivalent to : `cimg_forZ(img,z) cimg_for-XY(img,x,y)`.
- **cimg_forXYC(img,x,y,v)** : equivalent to : `cimg_forC(img,v) cimg_for-XY(img,x,y)`.
- **cimg_forXZC(img,x,z,v)** : equivalent to : `cimg_forC(img,v) cimg_for-XZ(img,x,z)`.
- **cimg_forYZC(img,y,z,v)** : equivalent to : `cimg_forC(img,v) cimg_for-YZ(img,y,z)`.
- **cimg_forXYZC(img,x,y,z,v)** : equivalent to : `cimg_forC(img,v) 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,v) { img(x,y,v) = (x+y)*(v+1)/6; }
img.display("Color gradient");
```

6.11.3 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,v,n)** : Loop along the v-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,v,n)** : Loop along the z-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 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.
- 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();
```

6.11.4 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.

6.11.4.1 Neighborhood-based loops for 2D images

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

- **cimg_for2x2(img,x,y,z,v,I)** : Loop along the (x,y)-axes using a centered 2x2 neighborhood.
- **cimg_for3x3(img,x,y,z,v,I)** : Loop along the (x,y)-axes using a centered 3x3 neighborhood.
- **cimg_for4x4(img,x,y,z,v,I)** : Loop along the (x,y)-axes using a centered 4x4 neighborhood.
- **cimg_for5x5(img,x,y,z,v,I)** : 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 `v` are constants that define on which image slice and vector channel the loop must apply (usually both 0 for grayscale 2D images). Finally, `I` is the 2x2, 3x3, 4x4 or 5x5 neighborhood that will be updated with the correct pixel values during the loop (see Defining neighborhoods).

6.11.4.2 Neighborhood-based loops for 3D images

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

- **cimg_for2x2x2(img,x,y,z,v,I)** : Loop along the (x,y,z)-axes using a centered 2x2x2 neighborhood.
- **cimg_for3x3x3(img,x,y,z,v,I)** : 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. `v` 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 that will be updated with the correct pixel values during the loop (see Defining neighborhoods).

6.11.4.3 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.

6.11.4.4 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`,`Inc`,`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 additionnal 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 than the nearest valid pixel in the image (this is also called the *Neumann border condition*).

6.11.4.5 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*(Incc-Ipcc);    // Compute the derivative along the
     x-axis.
  const float iy = 0.5f*(Icnc-Icpc);    // 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), neighbor(5,5);
    // 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 !

6.12 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.

6.13 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 is 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 : A1A2-A3A4A5 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 kth 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).

6.14 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,...

6.15 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.

6.15.1 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

usage	: A string describing the program goal and usage.

Precondition

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

6.15.2 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 programm.

6.15.3 The cimg_option() macro

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

Parameters

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

Returns

`cimg_option()` returns an object that has the *same type* than 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.

6.15.4 Example of use

The code below uses the macros `cimg_usage()` and `cimg_option()`. It loads an image, smoothes it an 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 writting 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.

6.15.5 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 CImgException

  Instances of `CImgException` are thrown when errors are encountered in a `CImg` function call.
- struct CImgDisplay

  Allow to create windows, display images on them and manage user events (keyboard, mouse and windows events).
- struct CImg

  Class representing an image (up to 4 dimensions wide), each pixel being of type `T`.
- struct CImgList

  Represent 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 functions and class names collisions that could happen with the include of other C++ header files. Anyway, it should not happen often and you should reasonnably start most of your `CImg`-based programs with

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

to simplify the declaration of `CImg` Library variables 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 informations about `CImg` environement variables.
- template<typename T >
  void unused (const T &,...)

  Avoid warning messages due to unused parameters. Actually do nothing.
- unsigned int & exception_mode (const unsigned int mode)

  Set current `CImg` exception mode.
- unsigned int & exception_mode ()

  Return current `CImg` exception mode.
- double eval (const char *const expression, const double x, const double y, const double z, const double c)

  Evaluate math expression.
- 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)
- 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 siz)

  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 time ()

  Return the value of a system timer, with a millisecond precision.
- unsigned long tic ()

  Start tic/toc timer for time measurement between code instructions.
- unsigned 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().
- double rand ()

  Return a random variable between [0,1] with respect to an uniform distribution.
- double crand ()

  Return a random variable between [-1,1] with respect to an uniform distribution.
- double grand ()

  Return a random variable following a gaussian distribution and a standard deviation of 1.
- unsigned int prand (const double z)

Return a random variable following a Poisson distribution of parameter z.

- 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 sqr (const T val)

  Return square of a value.
- int xln (const int x)

  Return `1 + log_10(x)` *of a value* `x`*.*
- template<typename t1 , typename t2 >
  cimg::superset< t1, t2 >::type min (const t1 &a, const t2 &b)

  Return the minimum between two values.
- template<typename t1 , typename t2 , typename t3 >
  cimg::superset2< t1, t2, t3 >::type min (const t1 &a, const t2 &b, const t3 &c)

  Return the minimum between three values.
- template<typename t1 , typename t2 , typename t3 , typename t4 >
  cimg::superset3< t1, t2, t3, t4 >::type min (const t1 &a, const t2 &b, const t3 &c, const t4 &d)

  Return the minimum between four values.
- template<typename t1 , typename t2 >
  cimg::superset< t1, t2 >::type max (const t1 &a, const t2 &b)

  Return the maximum between two values.
- template<typename t1 , typename t2 , typename t3 >
  cimg::superset2< t1, t2, t3 >::type max (const t1 &a, const t2 &b, const t3 &c)

  Return the maximum between three values.
- template<typename t1 , typename t2 , typename t3 , typename t4 >
  cimg::superset3< t1, t2, t3, t4 >::type max (const t1 &a, const t2 &b, const t3 &c, const t4 &d)

  Return the maximum between four values.
- template<typename T >
  T sign (const T x)

  Return the sign of a value.
- template<typename T >
  unsigned int nearest_pow2 (const T x)

  Return the nearest power of 2 higher than given value.
- double sinc (const double x)

  Return the sinc of a 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.
- double log2 (const double x)

  Return base-2 logarithm of a value.
- template<typename T >
  T round (const T x, const double y=1, const int rounding_type=0)

  Return rounded value.
- char uncase (const char x)

  Convert ascii character to lower case.
- void uncase (char ∗const str)

  Convert C-string to lower case.
- double atof (const char ∗const str)

  Read value in a C-string.
- int strncasecmp (const char ∗const str1, const char ∗const str2, const int l)

  Compare the first `l` 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.
- bool strpare (char ∗const str, const char delimiter=' ', const bool is_symmetric=false, const bool is_iterative=false)

  Remove delimiters on the start and/or end of a C-string.
- void strescape (char ∗const str)

  Replace escape sequences in C-strings by their binary ascii values.
- const char ∗ basename (const char ∗const str)

  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.
- const char ∗ temporary_path (const char ∗const user_path=0, const bool reinit_path=false)

  Get/set path to store temporary files.
- const char ∗ imagemagick_path (const char ∗const user_path=0, const bool reinit_path=false)

  Get/set path to the Program Files/ directory (Windows only).
- const char ∗ graphicsmagick_path (const char ∗const user_path=0, const bool reinit_path=false)

  Get/set path to the GraphicsMagick's `gm` binary.
- const char ∗ medcon_path (const char ∗const user_path=0, const bool reinit_path=false)

  Get/set path to the XMedcon's `medcon` binary.
- const char ∗ ffmpeg_path (const char ∗const user_path=0, const bool reinit_path=false)

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

- const char ∗ gzip_path (const char ∗const user_path=0, const bool reinit_-path=false)

  Get/set path to the `gzip` binary.

- const char ∗ gunzip_path (const char ∗const user_path=0, const bool reinit_-path=false)

  Get/set path to the `gzip` binary.

- const char ∗ dcraw_path (const char ∗const user_path=0, const bool reinit_-path=false)

  Get/set path to the `dcraw` binary.

- const char ∗ wget_path (const char ∗const user_path=0, const bool reinit_-path=false)

  Get/set path to the `wget` binary.

- const char ∗ curl_path (const char ∗const user_path=0, const bool reinit_-path=false)

  Get/set path to the `curl` 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 n, char ∗const str)

  Create numbered version of a filename.

- const char ∗ file_type (std::FILE ∗const file, const char ∗const filename)

  Try to guess format from an image file.

- template<typename T >
  int fread (T ∗const ptr, const unsigned long nmemb, std::FILE ∗stream)

  Read data from file.

- template<typename T >
  int fwrite (const T ∗ptr, const unsigned long nmemb, std::FILE ∗stream)

  Write data to file.

- char ∗ load_network_external (const char ∗const filename, char ∗const filename-_local)

  Load file from network as a local temporary file.

- 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 keyENTER = 53U

  Keycode for the `ENTER` 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 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. 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 std::FILE ∗ **output** (std::FILE ∗ *file*)

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

Parameters

file	Desired output stream. Set to `0` to get the currently used output stream only.

Returns

Currently used output stream.

See also

info(), warn(), CImgException.

7.2.2.2 void **info** ( )

Print informations about `CImg` environement variables.

Note

Output is done on the default output stream.

See also

output().

7.2.2.3 unsigned int& **cimg_library::cimg::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

mode	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 (default behavior). • `3` : Do as `1` + add extra debug warnings (slow down the code !). • `4` : Do as `2` + add extra debug warnings (slow down the code !).

See also

exception_mode(), `cimg_verbosity`.

7.2.2.4 unsigned int& **cimg_library::cimg::exception_mode** ( )

Return current `CImg` exception mode.

Note

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

See also

exception_mode(unsigned int), `cimg_verbosity`.

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

Evaluate math expression.

Parameters

expression	C-string containing the formula to evaluate.
x	Value of a pre-defined variable `x`.
y	Value of a pre-defined variable `y`.
z	Value of a pre-defined variable `z`.
c	Value of a pre-defined variable `c`.

Returns

Result of the formula evaluation.

Note

Set `expression` to `0` to evaluate the latest 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.
```

7.2.2.6 void cimg_library::cimg::warn (const char *const *format*, ...)

Display a warning message on the default output stream.

Parameters

format	C-string containing the format of the message, as with `std::printf()`.

Note

If configuration macro `cimg_strict_warnings` is set, this function throws a `CImgWarningException` 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 nasty memory access.

See also

output(), info(), CImgException.

7.2.2.7 int cimg_library::cimg::system (const char *const *command*, const char *const *module_name* = 0)

Parameters

command	C-string containing the command line to execute.

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.

7.2.2.8 bool cimg_library::cimg::endianness ( )

Return the endianness of the current architecture.

Returns

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

See also

invert_endianness(T*,unsigned int), invert_endianness(T&),

7.2.2.9 void cimg_library::cimg::invert_endianness (T *const *buffer,* const unsigned long *siz*)

Reverse endianness of all elements in a memory buffer.

Parameters

`in,out`	*buffer*	Memory buffer whose endianness must be reversed.
	size	Number of buffer elements to reverse.

See also

endianness(), invert_endianness(T&).

7.2.2.10 T& cimg_library::cimg::invert_endianness (T & *a*)

Reverse endianness of a single variable.

Parameters

`in,out`	*a*	Variable to reverse.

Returns

Reference to reversed variable.

See also

endianness(), invert_endianness(T∗,unsigned int).

7.2.2.11 unsigned long cimg_library::cimg::time ( )

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

Note

The timer does not necessarily starts from `0`.

See also

tic(), toc(), sleep(), wait().

7.2.2.12 unsigned long cimg_library::cimg::tic ( )

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

Returns

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

See also

time(), toc(), sleep(), wait().

7.2.2.13 unsigned long cimg_library::cimg::toc ( )

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

Returns

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

See also

time(), tic(), sleep(), wait().

7.2.2.14 void cimg_library::cimg::sleep (const unsigned int *milliseconds*)

Sleep for a given numbers of milliseconds.

Parameters

milliseconds	Number of milliseconds to wait for.

Note

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

See also

time(), tic(), toc(), wait().

7.2.2.15 unsigned int cimg_library::cimg::wait (const unsigned int *milliseconds*)

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

Parameters

milliseconds	Number of milliseconds to wait for.

Returns

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

Note

Similar to sleep(), but the waiting time is computed with regard to the last call of wait(). It may be used to temporize your program properly, without wasting CPU time.

See also

time(), tic(), toc(), sleep().

7.2.2.16 double cimg_library::cimg::rand ( )

Return a random variable between [0,1] with respect to an uniform distribution.

See also

crand(), grand(), prand().

7.2.2.17 double cimg_library::cimg::crand ( )

Return a random variable between [-1,1] with respect to an uniform distribution.

See also

rand(), grand(), prand().

7.2.2.18 double cimg_library::cimg::grand ( )

Return a random variable following a gaussian distribution and a standard deviation of 1.

See also

rand(), crand(), prand().

7.2.2.19 unsigned int cimg_library::cimg::prand (const double *z*)

Return a random variable following a Poisson distribution of parameter z.

See also

rand(), crand(), grand().

7.2.2.20 T cimg_library::cimg::mod (const T & *x*, const T & *m*)

Return the modulo of a value.

Parameters

x	Input value.
m	Modulo value.

Note

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

7.2.2.21 T cimg_library::cimg::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.

7.2.2.22 T **cimg_library::cimg::round** (const T *x*, const double *y* = `1`, const int *rounding_type* = `0`)

Return rounded value.

Parameters

x	Value to be rounded.
y	Rounding precision.
rounding_-type	Type of rounding operation (`0` = nearest, `-1` = backward, `1` = forward).

Returns

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

7.2.2.23 double **cimg_library::cimg::atof** (const char *∗const *str*)

Read value in a C-string.

Parameters

str	C-string containing the float value to read.

Returns

Read value.

Note

Similar to `std::atof()`, but allows the retrieval of fractions from C-strings, as in *"1/2"*.

7.2.2.24 int **cimg_library::cimg::strncasecmp** (const char ∗const *str1*, const char ∗const *str2*, const int *l*)

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

Parameters

str1	C-string.
str2	C-string.
l	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).

See also

strcasecmp().

7.2.2.25 int cimg_library::cimg::strcasecmp (const char ∗const *str1*, const char ∗const *str2*)

Compare two C-strings, ignoring the case.

Parameters

str1	C-string.
str2	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).

See also

strncasecmp().

7.2.2.26 bool cimg_library::cimg::strpare (char ∗const *str*, const char *delimiter* = ' ', const bool *is_symmetric* = `false`, const bool *is_iterative* = `false`)

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

Parameters

`in,out`	*str*	C-string to work with (modified at output).
	delimiter	Delimiter character code to remove.
	is_-symmetric	Flag telling if the removal is done only if delimiters are symmetric (both at the beginning and the end of `s`).
	is_iterative	Flag telling if the removal is done if several iterations are possible.

Returns

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

See also

strescape().

7.2.2.27 void **cimg_library::cimg::strescape** (char ∗const *str*)

Replace escape sequences in C-strings by their binary ascii values.

Parameters

`in,out`	*str*	C-string to work with (modified at output).

See also

strpare().

7.2.2.28 std::FILE∗ **cimg_library::cimg::fopen** (const char ∗const *path,* const char ∗const *mode*)

Open a file.

Parameters

path	Path of the filename to open.
mode	C-string describing the opening mode.

Returns

Opened file.

Note

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

7.2.2.29 int **cimg_library::cimg::fclose** (std::FILE * *file*)

Close a file.

Parameters

file	File to close.

Returns

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

Note

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

7.2.2.30 const char* **cimg_library::cimg::temporary_path** (const char *const *user_path* = `0`, const bool *reinit_path* = `false`)

Get/set path to store temporary files.

Parameters

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

Returns

Path where temporary files can be saved.

7.2.2.31 const char* **cimg_library::cimg::imagemagick_path** (const char *const *user_path* = `0`, const bool *reinit_path* = `false`)

Get/set path to the *Program Files/* directory (Windows only).

Parameters

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

Returns

Path containing the program files. Get/set path to the ImageMagick's `convert` binary.

Parameters

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

Returns

Path containing the `convert` binary.

7.2.2.32 const char∗ cimg_library::cimg::graphicsmagick_path (const char ∗const *user_path* = `0`, const bool *reinit_path* = `false`)

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

Parameters

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

Returns

Path containing the `gm` binary.

7.2.2.33 const char∗ cimg_library::cimg::medcon_path (const char ∗const *user_path* = `0`, const bool *reinit_path* = `false`)

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

Parameters

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

Returns

Path containing the `medcon` binary.

7.2.2.34 const char∗ cimg_library::cimg::ffmpeg_path (const char ∗const *user_path* = `0`, const bool *reinit_path* = `false`)

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

Parameters

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

Returns

Path containing the `ffmpeg` binary.

7.2.2.35 const char* **cimg_library::cimg::gzip_path** (const char *const *user_path* = `0`, const bool *reinit_path* = `false`)

Get/set path to the `gzip` binary.

Parameters

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

Returns

Path containing the `gzip` binary.

7.2.2.36 const char* **cimg_library::cimg::gunzip_path** (const char *const *user_path* = `0`, const bool *reinit_path* = `false`)

Get/set path to the `gzip` binary.

Parameters

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

Returns

Path containing the `gunzip` binary.

7.2.2.37 const char* **cimg_library::cimg::dcraw_path** (const char *const *user_path* = `0`, const bool *reinit_path* = `false`)

Get/set path to the `dcraw` binary.

Parameters

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

Returns

Path containing the `dcraw` binary.

7.2.2.38 const char∗ cimg_library::cimg::wget_path (const char ∗const *user_path* = `0`, const bool *reinit_path* = `false`)

Get/set path to the `wget` binary.

Parameters

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

Returns

Path containing the `wget` binary.

7.2.2.39 const char∗ cimg_library::cimg::curl_path (const char ∗const *user_path* = `0`, const bool *reinit_path* = `false`)

Get/set path to the `curl` binary.

Parameters

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

Returns

Path containing the `curl` binary.

7.2.2.40 const char∗ cimg_library::cimg::file_type (std::FILE ∗const *file*, const char ∗const *filename*)

Try to guess format from an image file.

Parameters

file	Input file (can be `0` if `filename` is set).
filename	Input filename (can be `0` if `file` is set).

Returns

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

7.2.2.41 int cimg_library::cimg::fread (T ∗const *ptr*, const unsigned long *nmemb*, std::FILE ∗ *stream*)

Read data from file.

Parameters

`out`	*ptr*	Pointer to memory buffer that will contain the binary data read from file.
	nmemb	Number of elements to read.
	stream	File to read data from.

Returns

Number of read elements.

Note

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

7.2.2.42 int **cimg_library::cimg::fwrite** (const T ∗ *ptr*, const unsigned long *nmemb*, std::FILE ∗ *stream*)

Write data to file.

Parameters

	ptr	Pointer to memory buffer containing the binary data to write on file.
	nmemb	Number of elements to write.
`out`	*stream*	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.

7.2.2.43 char∗ **cimg_library::cimg::load_network_external** (const char ∗const *filename*, char ∗const *filename_local*)

Load file from network as a local temporary file.

Parameters

	filename	Path to the filename to read from network.
`out`	*filename_-local*	C-string containing the path to a local copy of `filename`.

Returns

Value of `filename_local`.

Note

Use external binaries `wget` or `curl` to perform. You must have one of these tools installed to be able to use this function.

7.2.2.44 int cimg_library::cimg::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

title	Title of the dialog window.
msg	Main message displayed inside the dialog window.
button1_-label	Label of the 1st button.
button2_-label	Label of the 2nd button (`0` to hide button).
button3_-label	Label of the 3rd button (`0` to hide button).
button4_-label	Label of the 4th button (`0` to hide button).
button5_-label	Label of the 5th button (`0` to hide button).
button6_-label	Label of the 6th button (`0` to hide button).
logo	Image logo displayed at the left of the main message (optional).
centering	Flag telling if the dialog window must be centered on the screen.

Returns

Indice 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 followings) 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), each pixel being of type `T`.

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

- ~CImg ()

  Destructor.
- CImg ()

  Default constructor.
- 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.

- 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.

- 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.

- 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.

- 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 ***[specialization]****.*

- CImg (const char ∗const filename)

  Construct image from an image file.

- template<typename t >
  CImg (const CImg< t > &img)

  Copy constructor.

- CImg (const CImg< T > &img)

  Copy constructor ***[specialization]****.*

- template<typename t >
  CImg (const CImg< t > &img, const bool is_shared)

  Advanced copy constructor.

- CImg (const CImg< T > &img, const bool is_shared)

  Advanced copy constructor ***[specialization]****.*

- template<typename t >
  CImg (const CImg< t > &img, const char ∗const dimensions)

  Construct image with dimensions borrowed from another image.

- 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.

- CImg (const CImgDisplay &disp)

  Construct image from a display window.

- CImg< T > & assign ()

  In-place version of the default constructor/destructor.

- 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)

  In-place version of a constructor.

- 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)

    In-place version of a constructor.
- 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,...)

    In-place version of a constructor.
- 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,...)

    In-place version of a constructor.
- 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)

    In-place version of a constructor.
- 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)

    In-place version of a constructor.
- 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)

    *In-place version of a constructor **[specialization]**.*
- template<typename t >
  CImg< 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 bool is_shared)

    In-place version of a constructor.
- CImg< 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 bool is_shared)

    *In-place version of a constructor **[specialization]**.*
- CImg< T > & assign (const char *const filename)

    In-place version of a constructor.
- template<typename t >
  CImg< T > & assign (const CImg< t > &img)

    In-place version of the default copy constructor.
- template<typename t >
  CImg< T > & assign (const CImg< t > &img, const bool is_shared)

    In-place version of the advanced copy constructor.
- template<typename t >
  CImg< T > & assign (const CImg< t > &img, const char *const dimensions)

    In-place version of a constructor.
- template<typename t >
  CImg< T > & assign (const CImg< t > &img, const char *const dimensions, const T value)

In-place version of a constructor.

- CImg< T > & assign (const CImgDisplay &disp)

  In-place version of a constructor.
- CImg< T > & clear ()

  In-place version of the default constructor.
- template<typename t >
  CImg< t > & move_to (CImg< t > &img)

  Transfer content of an image instance into another one.
- CImg< T > & move_to (CImg< T > &img)

  Transfer content of an image instance into another one ***[specialization]****.*
- 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.
- CImg< T > & swap (CImg< T > &img)

  Swap fields of two image instances.
- static CImg< T > & empty ()

  Return a reference to an empty image.

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 long wh, const unsigned long 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 long wh, const unsigned long 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 substraction operator.
- CImg< T > & operator-= (const char *const expression)

  In-place substraction operator.
- template<typename t >
  CImg< T > & operator-= (const CImg< t > &img)

  In-place substraction 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

Substraction operator.

- CImg< Tfloat > operator- (const char ∗const expression) const

  Substraction operator.

- template<typename t >
  CImg< typename cimg::superset < T, t >::type > operator- (const CImg< t > &img) const

  Substraction 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 image along specified axis.

Instance Characteristics

- 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.

- unsigned long 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]**.*
- long 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.
- 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.
- 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.
- 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.
- 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.
- 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_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_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_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 and 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 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.
- Tfloat cubic_atX (const float fx, const int y, const int z, const int c, const T out_-value, const Tfloat min_value, const Tfloat max_value) const

  Return damped 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.
- Tfloat cubic_atX (const float fx, const int y, const int z, const int c, const Tfloat min_value, const Tfloat max_value) const

  Return damped pixel value, using cubic interpolation and Neumann 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.

- Tfloat cubic_atXY (const float fx, const float fy, const int z, const int c, const T out_value, const Tfloat min_value, const Tfloat max_value) const

  Return damped pixel value, using cubic interpolation and Dirichlet boundary conditions for the X and 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.
- Tfloat cubic_atXY (const float fx, const float fy, const int z, const int c, const Tfloat min_value, const Tfloat max_value) const

  Return damped pixel value, using cubic interpolation and Neumann 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.
- Tfloat cubic_atXYZ (const float fx, const float fy, const float fz, const int c, const T out_value, const Tfloat min_value, const Tfloat max_value) const

  Return damped pixel value, using cubic interpolation and Dirichlet boundary conditions for the X,Y and Z-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.
- Tfloat cubic_atXYZ (const float fx, const float fy, const float fz, const int c, const Tfloat min_value, const Tfloat max_value) const

  Return damped pixel value, using cubic interpolation and Neumann boundary conditions for the X,Y and Z-coordinates.
- 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.
- 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.
- CImg< charT > value_string (const char separator=',', const unsigned int max_size=0) const

  Return a C-string containing a list of all values of the image instance.
- static const char ∗ pixel_type ()

  Return the type of image pixel values as a C string.

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 that of another image.
- bool is_sameXY (const CImgDisplay &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 CImg< t > &img) const

  Test if image width and depth are the same as that 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 CImg< t > &img) const

Test if image width and spectrum are the same as that 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 CImg< t > &img) const

  Test if image height and depth are the same as that 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 CImg< t > &img) const

  Test if image height and spectrum are the same as that 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 CImg< t > &img) const

  Test if image depth and spectrum are the same as that 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 CImg< t > &img) const

  Test if image width, height and depth are the same as that 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 CImg< t > &img) const

  Test if image width, height and spectrum are the same as that 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 CImg< t > &img) const

  Test if image width, depth and spectrum are the same as that 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 CImg< t > &img) const

  Test if image height, depth and spectrum are the same as that 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 CImg< t > &img) const

Test if image width, height, depth and spectrum are the same as that 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 CImg< t > &img) const

  Test if pixel buffers of instance and input images overlap.
- template<typename tp , typename tc , typename to >
  bool is_object3d (const CImgList< tp > &primitives, const CImgList< tc > &colors, const to &opacities, const bool is_full_check=true, char ∗const error_message=0) const

  Test if the set {∗`this`,`primitives`,`colors`,`opacities`} defines a valid 3d object.
- bool is_CImg3d (const bool is_full_check=true, char ∗const error_message=0) const

  Test if image instance represents a valid serialization of a 3d object.

Mathematical Functions

- CImg< T > & sqr ()

  Compute the square value of each pixel value.
- CImg< Tfloat > get_sqr () const

  Compute the square value of each pixel value ***[new-instance version]****.*
- CImg< T > & sqrt ()

  Compute the square root of each pixel value.
- CImg< Tfloat > get_sqrt () const

  Compute the square root of each pixel value ***[new-instance version]****.*
- CImg< T > & exp ()

  Compute the exponential of each pixel value.
- CImg< Tfloat > get_exp () const

  Compute the exponential of each pixel value ***[new-instance version]****.*

- CImg< T > & log ()

  Compute the logarithm of each pixel value.
- CImg< Tfloat > get_log () const

  Compute the logarithm of each pixel value ***[new-instance version]****.*
- CImg< T > & log2 ()

  Compute the base-2 logarithm of each pixel value.
- CImg< Tfloat > get_log2 () const

  Compute the base-10 logarithm of each pixel value ***[new-instance version]****.*
- CImg< T > & log10 ()

  Compute the base-10 logarithm of each pixel value.
- CImg< Tfloat > get_log10 () const

  Compute the base-10 logarithm of each pixel value ***[new-instance version]****.*
- CImg< T > & abs ()

  Compute the absolute value of each pixel value.
- CImg< Tfloat > get_abs () const

  Compute the absolute value of each pixel value ***[new-instance version]****.*
- CImg< T > & sign ()

  Compute the sign of each pixel value.
- CImg< Tfloat > get_sign () const

  Compute the sign of each pixel value ***[new-instance version]****.*
- CImg< T > & cos ()

  Compute the cosine of each pixel value.
- CImg< Tfloat > get_cos () const

  Compute the cosine of each pixel value ***[new-instance version]****.*
- CImg< T > & sin ()

  Compute the sine of each pixel value.
- CImg< Tfloat > get_sin () const

  Compute the sine of each pixel value ***[new-instance version]****.*
- CImg< T > & sinc ()

  Compute the sinc of each pixel value.
- CImg< Tfloat > get_sinc () const

  Compute the sinc of each pixel value ***[new-instance version]****.*
- CImg< T > & tan ()

  Compute the tangent of each pixel value.
- CImg< Tfloat > get_tan () const

  Compute the tangent of each pixel value ***[new-instance version]****.*
- CImg< T > & cosh ()

  Compute the hyperbolic cosine of each pixel value.
- CImg< Tfloat > get_cosh () const

  Compute the hyperbolic cosine of each pixel value ***[new-instance version]****.*
- CImg< T > & sinh ()

  Compute the hyperbolic sine of each pixel value.
- CImg< Tfloat > get_sinh () const

*Compute the hyperbolic sine of each pixel value **[new-instance version]**.*

- CImg< T > & tanh ()

  Compute the hyperbolic tangent of each pixel value.
- CImg< Tfloat > get_tanh () const

  *Compute the hyperbolic tangent of each pixel value **[new-instance version]**.*
- CImg< T > & acos ()

  Compute the arccosine of each pixel value.
- CImg< Tfloat > get_acos () const

  *Compute the arccosine of each pixel value **[new-instance version]**.*
- CImg< T > & asin ()

  Compute the arcsine of each pixel value.
- CImg< Tfloat > get_asin () const

  *Compute the arcsine of each pixel value **[new-instance version]**.*
- CImg< T > & atan ()

  Compute the arctangent of each pixel value.
- CImg< Tfloat > get_atan () const

  *Compute the arctangent of each pixel value **[new-instance version]**.*
- 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]**.*
- 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]**.*

- CImg< T > & rol (const unsigned int n=1)

  Compute the bitwise left rotation of each pixel value.

- CImg< T > get_rol (const unsigned int n=1) const

  *Compute the bitwise left rotation of each pixel value **[new-instance version]**.*

- CImg< T > & rol (const char ∗const expression)

  Compute the bitwise left rotation of each pixel value.

- CImg< T > get_rol (const char ∗const expression) const

  *Compute the bitwise left rotation of each pixel value **[new-instance version]**.*

- template<typename t >
  CImg< T > & rol (const CImg< t > &img)

  Compute the bitwise left rotation of each pixel value.

- template<typename t >
  CImg< T > get_rol (const CImg< t > &img) const

  *Compute the bitwise left rotation of each pixel value **[new-instance version]**.*

- CImg< T > & ror (const unsigned int n=1)

  Compute the bitwise right rotation of each pixel value.

- CImg< T > get_ror (const unsigned int n=1) const

  *Compute the bitwise right rotation of each pixel value **[new-instance version]**.*

- CImg< T > & ror (const char ∗const expression)

  Compute the bitwise right rotation of each pixel value.

- CImg< T > get_ror (const char ∗const expression) const

  *Compute the bitwise right rotation of each pixel value **[new-instance version]**.*

- template<typename t >
  CImg< T > & ror (const CImg< t > &img)

  Compute the bitwise right rotation of each pixel value.

- template<typename t >
  CImg< T > get_ror (const CImg< t > &img) const

  *Compute the bitwise right rotation of each pixel value **[new-instance version]**.*

- CImg< T > & min (const T val)

  Pointwise min operator between an image and a value.

- CImg< T > **get_min** (const T val) const
- template<typename t >
  CImg< T > & min (const CImg< t > &img)

  Pointwise min operator between two images.

- template<typename t >
  CImg< typename cimg::superset < T, t >::type > **get_min** (const CImg< t > &img) const
- CImg< T > & min (const char *const expression)

  Pointwise min operator between an image and a string.
- CImg< Tfloat > **get_min** (const char *const expression) const
- CImg< T > & max (const T val)

  Pointwise max operator between an image and a value.
- CImg< T > **get_max** (const T val) const
- template<typename t >
  CImg< T > & max (const CImg< t > &img)

  Pointwise max operator between two images.
- template<typename t >
  CImg< typename cimg::superset < T, t >::type > **get_max** (const CImg< t > &img) const
- CImg< T > & max (const char *const expression)

  Pointwise max operator between an image and a string.
- CImg< Tfloat > **get_max** (const char *const expression) const
- T & min ()

  Return a reference to the minimum pixel value of the image instance.
- const T & **min** () const
- T & max ()

  Return a reference to the maximum pixel value of the image instance.
- const T & **max** () const
- template<typename t >
  T & min_max (t &max_val)

  Return a reference to the minimum pixel value and return also the maximum pixel value.
- template<typename t >
  const T & **min_max** (t &max_val) const
- template<typename t >
  T & max_min (t &min_val)

  Return a reference to the maximum pixel value and return also the minimum pixel value.
- template<typename t >
  const T & **max_min** (t &min_val) const
- T kth_smallest (const unsigned int k) const

  Return the kth smallest element of the image.
- T median () const

  Return the median value of the image.
- Tdouble sum () const

  Return the sum of all the pixel values in an image.
- Tdouble mean () const

  Return the mean pixel value of the image instance.
- Tdouble variance (const unsigned int variance_method=1) const

Return the variance of the image.

- template<typename t >
  Tdouble variance_mean (const unsigned int variance_method, t &mean) const

  Return the variance and the mean of the image.
- Tdouble variance_noise (const unsigned int variance_method=2) const

  Estimate noise variance of the image instance.
- template<typename t >
  Tdouble MSE (const CImg< t > &img) const

  Compute the MSE (Mean-Squared Error) between two images.
- template<typename t >
  Tdouble PSNR (const CImg< t > &img, const Tdouble valmax=255) const

  Compute the PSNR 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) const

  Evaluate math expression.
- CImg< T > & stats (const unsigned int variance_method=1)

  Compute a statistics vector (min,max,mean,variance,xmin,ymin,zmin,cmin,xmax,ymax,zmax,cmax).
- CImg< Tdouble > **get_stats** (const unsigned int variance_method=1) const

Vector / Matrix Operations

- Tdouble magnitude (const int magnitude_type=2) const

  Return the norm of the current vector/matrix. `ntype` = norm type (0=L2, 1=L1, -1=-Linf).
- Tdouble trace () const

  Return the trace of the image, viewed as a matrix.
- Tdouble det () const

  Return the determinant of the image, viewed as a matrix.
- template<typename t >
  Tdouble dot (const CImg< t > &img) const

  Return the dot product of the current vector/matrix with the vector/matrix `img`.
- CImg< T > get_vector_at (const unsigned int x, const unsigned int y=0, const unsigned int z=0) const

  Return a new image corresponding to the vector located at (x,y,z) of the current vector-valued image.
- CImg< T > get_matrix_at (const unsigned int x=0, const unsigned int y=0, const unsigned int z=0) const

  Return a new image corresponding to the square matrix located at (x,y,z) of the current vector-valued image.
- CImg< T > get_tensor_at (const unsigned int x, const unsigned int y=0, const unsigned int z=0) const

  Return a new image corresponding to the diffusion tensor located at (x,y,z) of the current vector-valued image.
- 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 the image `vec` as the vector valued pixel located at (x,y,z) of the current vector-valued image.

- 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 the image `vec` as the square matrix-valued pixel located at (x,y,z) of the current vector-valued image.

- 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 the image `vec` as the tensor valued pixel located at (x,y,z) of the current vector-valued image.

- CImg< T > & vector ()

  Unroll all images values into a one-column vector.

- CImg< T > **get_vector** () const
- CImg< T > & matrix ()

  Realign pixel values of the image instance as a square matrix.

- CImg< T > **get_matrix** () const
- CImg< T > & tensor ()

  Realign pixel values of the image instance as a symmetric tensor.

- CImg< T > **get_tensor** () const
- CImg< T > & diagonal ()

  Return a diagonal matrix, whose diagonal coefficients are the coefficients of the input image.

- CImg< T > **get_diagonal** () const
- CImg< T > & identity_matrix ()

  Return an identity matrix having same dimension than image instance.

- CImg< T > **get_identity_matrix** () const
- CImg< T > & sequence (const T a0, const T a1)

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

- CImg< T > **get_sequence** (const T a0, const T a1) const
- CImg< T > & transpose ()

  Transpose the current matrix.

- CImg< T > **get_transpose** () const
- template<typename t >
CImg< T > & cross (const CImg< t > &img)

  Compute the cross product between two 3d vectors.

- template<typename t >
CImg< typename cimg::superset < T, t >::type > **get_cross** (const CImg< t > &img) const
- CImg< T > & invert (const bool use_LU=true)

  Invert the current matrix.

- CImg< Tfloat > **get_invert** (const bool use_LU=true) const
- CImg< T > & pseudoinvert ()

  Compute the pseudo-inverse (Moore-Penrose) of the matrix.

- CImg< Tfloat > **get_pseudoinvert** () const
- template<typename t >
  CImg< T > & solve (const CImg< t > &A)

  *Solve a linear system AX=B where B=*this.*
- template<typename t >
  CImg< typename cimg::superset2 < T, t, float >::type > **get_solve** (const C-Img< t > &A) const
- template<typename t >
  CImg< T > & solve_tridiagonal (const CImg< t > &A)

  *Solve a linear system AX=B where B=*this and A is a tridiagonal matrix A = [b0,c0,0,...; a1,b1,c1,0,... ; ... ; ...,0,aN,bN],.*
- template<typename t >
  CImg< typename cimg::superset2 < T, t, float >::type > **get_solve_tridiagonal** (const CImg< t > &A) const
- template<typename t >
  const CImg< T > & eigen (CImg< t > &val, CImg< t > &vec) const

  Compute the eigenvalues and eigenvectors of a matrix.
- CImgList< Tfloat > **get_eigen** () const
- template<typename t >
  const CImg< T > & symmetric_eigen (CImg< t > &val, CImg< t > &vec) const

  Compute the eigenvalues and eigenvectors of a symmetric matrix.
- CImgList< Tfloat > **get_symmetric_eigen** () const
- template<typename t >
  CImg< T > & sort (CImg< t > &permutations, const bool increasing=true)

  Sort values of a vector and get corresponding permutations.
- template<typename t >
  CImg< T > **get_sort** (CImg< t > &permutations, const bool increasing=true) const
- CImg< T > & sort (const bool increasing=true, const char axis=0)

  Sort image values.
- CImg< T > **get_sort** (const bool increasing=true, const char axis=0) const
- 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 a general matrix.
- CImgList< Tfloat > **get_SVD** (const bool sorting=true, const unsigned int max_-iteration=40, const float lambda=0) const
- template<typename t >
  CImg< T > & dijkstra (const unsigned int starting_node, const unsigned int ending_node, CImg< t > &previous)

  Return minimal path in a graph, using the Dijkstra algorithm.
- template<typename t >
  CImg< T > **get_dijkstra** (const unsigned int starting_node, const unsigned int ending_node, CImg< t > &previous) const
- CImg< T > & dijkstra (const unsigned int starting_node, const unsigned int ending_node=∼0U)

Return minimal path in a graph, using the Dijkstra algorithm.

- CImg< Tfloat > **get_dijkstra** (const unsigned int starting_node, const unsigned int ending_node=∼0U) const
- CImg< 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.
- template<typename tf , typename t >
  static CImg< T > dijkstra (const tf &distance, const unsigned int nb_nodes, const unsigned int starting_node, const unsigned int ending_node, CImg< t > &previous)

  Compute minimal path in a graph, using the Dijkstra algorithm.
- template<typename tf , typename t >
  static CImg< 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.
- 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)

  Return stream line of a 3d vector field.
- static CImg< floatT > streamline (const char ∗const expression, 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=true, 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 vector field.
- static CImg< T > string (const char ∗const str, const bool include_last_zero=true)

  Return an image containing the specified string.
- static CImg< T > vector (const T &a0)

  Return a vector with specified coefficients.
- static CImg< T > vector (const T &a0, const T &a1)

  Return a vector with specified coefficients.
- static CImg< T > vector (const T &a0, const T &a1, const T &a2)

  Return a vector with specified coefficients.
- static CImg< T > vector (const T &a0, const T &a1, const T &a2, const T &a3)

  Return a vector with specified coefficients.
- static CImg< T > vector (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4)

  Return a vector with specified coefficients.
- static CImg< T > vector (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5)

  Return a vector with specified coefficients.

- static CImg< T > vector (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6)

  Return a vector with specified coefficients.
- static CImg< T > vector (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7)

  Return a vector with specified coefficients.
- static CImg< T > vector (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 vector with specified coefficients.
- static CImg< T > vector (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)

  Return a vector with specified coefficients.
- static CImg< T > vector (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)

  Return a vector with specified coefficients.
- static CImg< T > vector (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)

  Return a vector with specified coefficients.
- static CImg< T > vector (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)

  Return a vector with specified coefficients.
- static CImg< T > vector (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)

  Return a vector with specified coefficients.
- static CImg< T > vector (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)

  Return a vector with specified coefficients.
- static CImg< T > vector (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 vector with specified coefficients.
- static CImg< T > matrix (const T &a0)

  Return a 1x1 square matrix with specified coefficients.
- static CImg< T > matrix (const T &a0, const T &a1, const T &a2, const T &a3)

  Return a 2x2 square matrix with specified coefficients.
- 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)

  Return a 3x3 square matrix with specified coefficients.

- 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, const T &a9, const T &a10, const T &a11, const T &a12, const T &a13, const T &a14, const T &a15)

  Return a 4x4 square matrix with specified coefficients.
- 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, 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 square matrix with specified coefficients.
- static CImg< T > tensor (const T &a1)

  Return a 1x1 symmetric matrix with specified coefficients.
- static CImg< T > tensor (const T &a1, const T &a2, const T &a3)

  Return a 2x2 symmetric matrix tensor with specified coefficients.
- static CImg< T > tensor (const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6)

  Return a 3x3 symmetric matrix with specified coefficients.
- static CImg< T > diagonal (const T &a0)

  Return a 1x1 diagonal matrix with specified coefficients.
- static CImg< T > diagonal (const T &a0, const T &a1)

  Return a 2x2 diagonal matrix with specified coefficients.
- static CImg< T > diagonal (const T &a0, const T &a1, const T &a2)

  Return a 3x3 diagonal matrix with specified coefficients.
- static CImg< T > diagonal (const T &a0, const T &a1, const T &a2, const T &a3)

  Return a 4x4 diagonal matrix with specified coefficients.
- static CImg< T > diagonal (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4)

  Return a 5x5 diagonal matrix with specified coefficients.
- static CImg< T > identity_matrix (const unsigned int N)

  Return a NxN identity matrix.
- static CImg< T > sequence (const unsigned int N, const T a0, const T a1)

  Return a N-numbered sequence vector from `a0` to `a1`.
- static CImg< 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 along the (x,y,z)-axis with an angle w.

Value Manipulation

- CImg< T > & fill (const T val)

  Fill an image by a value `val`.
- CImg< T > **get_fill** (const T val) const
- CImg< T > & fill (const T val0, const T val1)

  Fill sequentially all pixel values with values val0 and val1 respectively.

- CImg< T > **get_fill** (const T val0, const T val1) const
- CImg< T > & fill (const T val0, const T val1, const T val2)

  Fill sequentially all pixel values with values val0 and val1 and val2.
- CImg< T > **get_fill** (const T val0, const T val1, const T val2) const
- CImg< T > & fill (const T val0, const T val1, const T val2, const T val3)

  Fill sequentially all pixel values with values val0 and val1 and val2 and val3.
- CImg< T > **get_fill** (const T val0, const T val1, const T val2, const T val3) const
- CImg< T > & fill (const T val0, const T val1, const T val2, const T val3, const T val4)

  Fill sequentially all pixel values with values val0 and val1 and val2 and val3 and val4.
- CImg< T > **get_fill** (const T val0, const T val1, const T val2, const T val3, const T val4) const
- CImg< T > & fill (const T val0, const T val1, const T val2, const T val3, const T val4, const T val5)

  Fill sequentially all pixel values with values val0 and val1 and val2 and val3 and val4 and val5.
- CImg< T > **get_fill** (const T val0, const T val1, const T val2, const T val3, const T val4, const T val5) const
- CImg< T > & fill (const T val0, const T val1, const T val2, const T val3, const T val4, const T val5, const T val6)

  Fill sequentially pixel values.
- 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
- CImg< T > & 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)

  Fill sequentially pixel values.
- 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
- CImg< T > & 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)

  Fill sequentially pixel values.
- 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
- CImg< T > & 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)

  Fill sequentially pixel values.
- 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
- CImg< T > & 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)

  Fill sequentially pixel values.
- 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

- CImg< T > & 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)

  Fill sequentially pixel values.
- 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
- CImg< T > & 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)

  Fill sequentially pixel values.
- 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
- CImg< T > & 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)

  Fill sequentially pixel values.
- 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
- CImg< T > & 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)

  Fill sequentially pixel values.
- 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
- CImg< T > & 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)

  Fill sequentially pixel values.
- 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
- CImg< T > & fill (const char *const expression, const bool repeat_flag)

  Fill image values according to the given expression, which can be a formula or a list of values.
- CImg< T > **get_fill** (const char *const values, const bool repeat_values) const
- template< typename t >
  CImg< T > & fill (const CImg< t > &values, const bool repeat_values=true)

  Fill image values according to the values found in the specified image.
- template< typename t >
  CImg< T > **get_fill** (const CImg< t > &values, const bool repeat_values=true) const

- CImg< T > & fillX (const unsigned int y, const unsigned int z, const unsigned int c, const int a0,...)

  Fill image values along the X-axis at the specified pixel position (y,z,c).
- CImg< T > & **fillX** (const unsigned int y, const unsigned int z, const unsigned int c, const double a0,...)
- CImg< T > & fillY (const unsigned int x, const unsigned int z, const unsigned int c, const int a0,...)

  Fill image values along the Y-axis at the specified pixel position (x,z,c).
- CImg< T > & **fillY** (const unsigned int x, const unsigned int z, const unsigned int c, const double a0,...)
- CImg< T > & fillZ (const unsigned int x, const unsigned int y, const unsigned int c, const int a0,...)

  Fill image values along the Z-axis at the specified pixel position (x,y,c).
- CImg< T > & **fillZ** (const unsigned int x, const unsigned int y, const unsigned int c, const double a0,...)
- CImg< T > & fillC (const unsigned int x, const unsigned int y, const unsigned int z, const int a0,...)

  Fill image values along the C-axis at the specified pixel position (x,y,z).
- CImg< T > & **fillC** (const unsigned int x, const unsigned int y, const unsigned int z, const double a0,...)
- CImg< T > & discard (const T value)

  Remove specified value from the image buffer, and return resulting buffer as a one-column vector.
- CImg< T > **get_discard** (const T value) const
- template<typename t >
  CImg< T > & discard (const CImg< t > &values)

  Remove specified values sequence from the image buffer, and return resulting buffer as a one-column vector.
- template<typename t >
  CImg< T > **get_discard** (const CImg< t > &values) const
- CImg< T > & invert_endianness ()

  Invert endianness of the image buffer.
- CImg< T > **get_invert_endianness** () const
- CImg< T > & rand (const T val_min, const T val_max)

  Fill the image instance with random values between specified range.
- CImg< T > **get_rand** (const T val_min, const T val_max) const
- CImg< T > & round (const double y=1, const int rounding_type=0)

  Compute image with rounded pixel values.
- CImg< T > **get_round** (const double y=1, const unsigned int rounding_type=0) const
- CImg< T > & noise (const double sigma, const unsigned int noise_type=0)

  Add random noise to the values of the image instance.
- CImg< T > **get_noise** (const double sigma, const unsigned int noise_type=0) const
- CImg< T > & normalize (const T value_min, const T value_max)

Linearly normalize values of the image instance between `value_min` and `value-_max`.

- CImg< Tfloat > **get_normalize** (const T value_min, const T value_max) const
- CImg< T > & normalize ()

  Normalize multi-valued pixels of the image instance, with respect to their L2-norm.
- CImg< Tfloat > **get_normalize** () const
- CImg< T > & norm (const int norm_type=2)

  Compute L2-norm of each multi-valued pixel of the image instance.
- CImg< Tfloat > **get_norm** (const int norm_type=2) const
- CImg< T > & cut (const T value_min, const T value_max)

  Cut values of the image instance between `value_min` and `value_max`.
- CImg< T > **get_cut** (const T value_min, const T value_max) const
- CImg< T > & quantize (const unsigned int nb_levels, const bool keep_-range=true)

  Uniformly quantize values of the image instance into `nb_levels` levels.
- CImg< T > **get_quantize** (const unsigned int n, const bool keep_range=true) const
- CImg< T > & threshold (const T value, const bool soft_threshold=false, const bool strict_threshold=false)

  Threshold values of the image instance.
- CImg< T > **get_threshold** (const T value, const bool soft_threshold=false, const bool strict_threshold=false) const
- CImg< T > & histogram (const unsigned int nb_levels, const T value_min=(T) 0, const T value_max=(T) 0)

  Compute the histogram of the image instance.
- CImg< floatT > **get_histogram** (const unsigned int nb_levels, const T value_-min=(T) 0, const T value_max=(T) 0) const
- CImg< T > & equalize (const unsigned int nb_levels, const T value_min=(T) 0, const T value_max=(T) 0)

  Compute the histogram-equalized version of the image instance.
- CImg< T > **get_equalize** (const unsigned int nblevels, const T val_min=(T) 0, const T val_max=(T) 0) const
- template<typename t >
  CImg< T > & index (const CImg< t > &palette, const float dithering=1, const bool map_indexes=false)

  Index multi-valued pixels of the image instance, regarding to a predefined palette.
- template<typename t >
  CImg< typename CImg< t >::Tuint > **get_index** (const CImg< t > &palette, const float dithering=1, const bool map_indexes=true) const
- template<typename t >
  CImg< T > & map (const CImg< t > &palette)

  Map predefined palette on the scalar (indexed) image instance.
- template<typename t >
  CImg< t > **get_map** (const CImg< t > &palette) const
- CImg< T > & label (const bool is_high_connectivity=false, const Tfloat toler-ance=0)

Label connected components.

- CImg< unsigned long > **get_label** (const bool is_high_connectivity=false, const Tfloat tolerance=0) const
- template<typename t >
  CImg< T > & **label** (const CImg< t > &connectivity_mask, const Tfloat tolerance=0)
- template<typename t >
  CImg< unsigned long > **get_label** (const CImg< t > &connectivity_mask, const Tfloat tolerance=0) const

Color Base Management

- CImg< T > & sRGBtoRGB ()

  Convert color pixels from sRGB to RGB.
- CImg< Tfloat > **get_sRGBtoRGB** () const
- CImg< T > & RGBtosRGB ()

  Convert color pixels from RGB to sRGB.
- CImg< Tfloat > **get_RGBtosRGB** () const
- CImg< T > & RGBtoHSV ()

  Convert color pixels from RGB to HSV.
- CImg< Tfloat > **get_RGBtoHSV** () const
- CImg< T > & HSVtoRGB ()

  Convert color pixels from HSV to RGB.
- CImg< Tuchar > **get_HSVtoRGB** () const
- CImg< T > & RGBtoHSL ()

  Convert color pixels from RGB to HSL.
- CImg< Tfloat > **get_RGBtoHSL** () const
- CImg< T > & HSLtoRGB ()

  Convert color pixels from HSL to RGB.
- CImg< Tuchar > **get_HSLtoRGB** () const
- CImg< T > & RGBtoHSI ()
- CImg< Tfloat > **get_RGBtoHSI** () const
- CImg< T > & HSItoRGB ()

  Convert color pixels from HSI to RGB.
- CImg< Tfloat > **get_HSItoRGB** () const
- CImg< T > & RGBtoYCbCr ()

  Convert color pixels from RGB to YCbCr.
- CImg< Tuchar > **get_RGBtoYCbCr** () const
- CImg< T > & YCbCrtoRGB ()

  Convert color pixels from RGB to YCbCr.
- CImg< Tuchar > **get_YCbCrtoRGB** () const
- CImg< T > & RGBtoYUV ()

  Convert color pixels from RGB to YUV.
- CImg< Tfloat > **get_RGBtoYUV** () const
- CImg< T > & YUVtoRGB ()

Convert color pixels from YUV to RGB.

- CImg< Tuchar > **get_YUVtoRGB** () const
- CImg< T > & RGBtoCMY ()

  Convert color pixels from RGB to CMY.
- CImg< Tuchar > **get_RGBtoCMY** () const
- CImg< T > & CMYtoRGB ()

  Convert CMY pixels of a color image into the RGB color space.
- CImg< Tuchar > **get_CMYtoRGB** () const
- CImg< T > & CMYtoCMYK ()

  Convert color pixels from CMY to CMYK.
- CImg< Tuchar > **get_CMYtoCMYK** () const
- CImg< T > & CMYKtoCMY ()

  Convert CMYK pixels of a color image into the CMY color space.
- CImg< Tfloat > **get_CMYKtoCMY** () const
- CImg< T > & RGBtoXYZ ()

  Convert color pixels from RGB to XYZ_709.
- CImg< Tfloat > **get_RGBtoXYZ** () const
- CImg< T > & XYZtoRGB ()

  Convert XYZ_709 pixels of a color image into the RGB color space.
- CImg< Tuchar > **get_XYZtoRGB** () const
- CImg< T > & XYZtoLab ()

  Convert XYZ_709 pixels of a color image into the (L∗,a∗,b∗) color space.
- CImg< Tfloat > **get_XYZtoLab** () const
- CImg< T > & LabtoXYZ ()

  Convert Lab pixels of a color image into the XYZ color space.
- CImg< Tfloat > **get_LabtoXYZ** () const
- CImg< T > & XYZtoxyY ()

  Convert XYZ_709 pixels of a color image into the xyY color space.
- CImg< Tfloat > **get_XYZtoxyY** () const
- CImg< T > & xyYtoXYZ ()

  Convert xyY pixels of a color image into the XYZ_709 color space.
- CImg< Tfloat > **get_xyYtoXYZ** () const
- CImg< T > & RGBtoLab ()

  Convert a RGB image to a Lab one.
- CImg< Tfloat > **get_RGBtoLab** () const
- CImg< T > & LabtoRGB ()

  Convert a Lab image to a RGB one.
- CImg< Tuchar > **get_LabtoRGB** () const
- CImg< T > & RGBtoxyY ()

  Convert a RGB image to a xyY one.
- CImg< Tfloat > **get_RGBtoxyY** () const
- CImg< T > & xyYtoRGB ()

  Convert a xyY image to a RGB one.
- CImg< Tuchar > **get_xyYtoRGB** () const

- CImg< T > & RGBtoCMYK ()

  Convert a RGB image to a CMYK one.
- CImg< Tfloat > **get_RGBtoCMYK** () const
- CImg< T > & CMYKtoRGB ()

  Convert a CMYK image to a RGB one.
- CImg< Tuchar > **get_CMYKtoRGB** () const
- CImg< T > & RGBtoBayer ()

  Convert a RGB image to a Bayer-coded representation.
- CImg< T > **get_RGBtoBayer** () const
- CImg< T > & BayertoRGB (const unsigned int interpolation_type=3)

  Convert a Bayer-coded image to a RGB color image.
- CImg< Tuchar > **get_BayertoRGB** (const unsigned int interpolation_type=3) const
- static const CImg< Tuchar > & default_LUT256 ()

  Return a palette 'default' with 256 RGB entries.
- static const CImg< Tuchar > & HSV_LUT256 ()

  Return palette 'HSV' with 256 RGB entries.
- static const CImg< Tuchar > & lines_LUT256 ()

  Return palette 'lines' with 256 RGB entries.
- static const CImg< Tuchar > & hot_LUT256 ()

  Return the palette 'hot' with 256 RGB entries.
- static const CImg< Tuchar > & cool_LUT256 ()

  Return the palette 'cool' with 256 RGB entries.
- static const CImg< Tuchar > & jet_LUT256 ()

  Return palette 'jet' with 256 RGB entries.
- static const CImg< Tuchar > & flag_LUT256 ()

  Return palette 'flag' with 256 RGB entries.
- static const CImg< Tuchar > & cube_LUT256 ()

  Return palette 'cube' with 256 RGB entries.

Geometric / Spatial Manipulation

- 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 border_conditions=0, const float centering_x=0, const float centering_y=0, const float centering_z=0, const float centering_c=0)

  Resize an image.
- CImg< 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 border_conditions=0, const float centering_x=0, const float centering_y=0, const float centering_z=0, const float centering_c=0) const
- template<typename t >
  CImg< T > & resize (const CImg< t > &src, const int interpolation_type=1, const unsigned int border_conditions=0, const float centering_x=0, const float centering_y=0, const float centering_z=0, const float centering_c=0)

Resize an image.

- template<typename t >
  CImg< T > **get_resize** (const CImg< t > &src, const int interpolation_type=1, const unsigned int border_conditions=0, const float centering_x=0, const float centering_y=0, const float centering_z=0, const float centering_c=0) const
- CImg< T > & resize (const CImgDisplay &disp, const int interpolation_type=1, const unsigned int border_conditions=0, const float centering_x=0, const float centering_y=0, const float centering_z=0, const float centering_c=0)

  Resize an image.
- CImg< T > **get_resize** (const CImgDisplay &disp, const int interpolation_type=1, const unsigned int border_conditions=0, const float centering_x=0, const float centering_y=0, const float centering_z=0, const float centering_c=0) const
- CImg< T > & resize_halfXY ()

  Half-resize an image, using a special optimized filter.
- CImg< T > **get_resize_halfXY** () const
- CImg< T > & resize_doubleXY ()

  Upscale an image by a factor 2x.
- CImg< T > **get_resize_doubleXY** () const
- CImg< T > & resize_tripleXY ()

  Upscale an image by a factor 3x.
- CImg< T > **get_resize_tripleXY** () const
- CImg< T > & mirror (const char axis)

  Mirror an image along the specified axis.
- CImg< T > **get_mirror** (const char axis) const
- CImg< T > & mirror (const char ∗const axes)

  Mirror image along specified axes.
- CImg< T > **get_mirror** (const char ∗const axes) const
- CImg< T > & shift (const int deltax, const int deltay=0, const int deltaz=0, const int deltac=0, const int border_condition=0)

  Shift the image.
- CImg< T > **get_shift** (const int deltax, const int deltay=0, const int deltaz=0, const int deltac=0, const int border_condition=0) const
- CImg< T > & permute_axes (const char ∗const order)

  Permute axes order.
- CImg< T > **get_permute_axes** (const char ∗const order) const
- CImg< T > & unroll (const char axis)

  Unroll all images values into specified axis.
- CImg< T > **get_unroll** (const char axis) const
- CImg< T > & rotate (const float angle, const unsigned int border_conditions=0, const unsigned int interpolation=1)

  Rotate an image.
- CImg< T > **get_rotate** (const float angle, const unsigned int border_-conditions=0, const unsigned int interpolation=1) const
- CImg< T > & rotate (const float angle, const float cx, const float cy, const float zoom, const unsigned int border_conditions=3, const unsigned int interpola-tion=1)

Rotate an image around a center point (cx,cy).

- CImg< T > **get_rotate** (const float angle, const float cx, const float cy, const float zoom, const unsigned int border_conditions=3, const unsigned int interpolation=1) const
- template<typename t >
  CImg< T > & warp (const CImg< t > &warp, const bool is_relative=false, const bool interpolation=true, const unsigned int border_conditions=0)

  Warp an image.
- template<typename t >
  CImg< T > **get_warp** (const CImg< t > &warp, const bool is_relative=false, const bool interpolation=true, const unsigned int border_conditions=0) const
- CImg< T > & projections2d (const unsigned int x0, const unsigned int y0, const unsigned int z0)

  Return a 2d representation of a 3d image, with three slices.
- CImg< T > **get_projections2d** (const unsigned int x0, const unsigned int y0, const unsigned int z0) const
- 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 bool border_condition=false)

  Return a square region of the image.
- CImg< 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 bool border_condition=false) const
- CImg< T > & crop (const int x0, const int y0, const int z0, const int x1, const int y1, const int z1, const bool border_condition=false)

  Return a rectangular part of the image instance.
- CImg< T > **get_crop** (const int x0, const int y0, const int z0, const int x1, const int y1, const int z1, const bool border_condition=false) const
- CImg< T > & crop (const int x0, const int y0, const int x1, const int y1, const bool border_condition=false)

  Return a rectangular part of the image instance.
- CImg< T > **get_crop** (const int x0, const int y0, const int x1, const int y1, const bool border_condition=false) const
- CImg< T > & crop (const int x0, const int x1, const bool border_-condition=false)

  Return a rectangular part of the image instance.
- CImg< T > **get_crop** (const int x0, const int x1, const bool border_-condition=false) const
- CImg< T > & autocrop (const T value, const char *const axes="czyx")

  Autocrop an image, regarding of the specified backround value.
- CImg< T > **get_autocrop** (const T value, const char *const axes="czyx") const
- CImg< T > & autocrop (const T *const color, const char *const axes="zyx")

  Autocrop an image, regarding of the specified backround color.
- CImg< T > **get_autocrop** (const T *const color, const char *const axes="zyx") const
- template<typename t >
  CImg< T > & autocrop (const CImg< t > &color, const char *const axes="zyx")

Autocrop an image, regarding of the specified backround color.

- template<typename t >
  CImg< T > **get_autocrop** (const CImg< t > &color, const char ∗const axes="zyx") const
- CImg< T > & column (const int x0)

  Return one column.
- CImg< T > **get_column** (const int x0) const
- CImg< T > & columns (const int x0, const int x1)

  Return a set of columns.
- CImg< T > **get_columns** (const int x0, const int x1) const
- CImg< T > & line (const int y0)

  Return a line.
- CImg< T > **get_line** (const int y0) const
- CImg< T > & lines (const int y0, const int y1)

  Return a set of lines.
- CImg< T > **get_lines** (const int y0, const int y1) const
- CImg< T > & slice (const int z0)

  Return a slice.
- CImg< T > **get_slice** (const int z0) const
- CImg< T > & slices (const int z0, const int z1)

  Return a set of slices.
- CImg< T > **get_slices** (const int z0, const int z1) const
- CImg< T > & channel (const int c0)

  Return a channel.
- CImg< T > **get_channel** (const int c0) const
- CImg< T > & channels (const int c0, const int c1)

  Return a set of channels.
- CImg< T > **get_channels** (const int c0, const int c1) const
- 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 set of points of the image instance.
- const 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) const
- CImg< T > get_shared_lines (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 set of lines of the image instance.
- const CImg< T > **get_shared_lines** (const unsigned int y0, const unsigned int y1, const unsigned int z0=0, const unsigned int c0=0) const
- CImg< T > get_shared_line (const unsigned int y0, const unsigned int z0=0, const unsigned int c0=0)

  Return a shared-memory image referencing one particular line (y0,z0,c0) of the image instance.
- const CImg< T > **get_shared_line** (const unsigned int y0, const unsigned int z0=0, const unsigned int c0=0) const

- CImg< T > get_shared_planes (const unsigned int z0, const unsigned int z1, const unsigned int c0=0)

  Return a shared memory image referencing a set of planes (z0->z1,c0) of the image instance.

- const CImg< T > **get_shared_planes** (const unsigned int z0, const unsigned int z1, const unsigned int c0=0) const
- CImg< T > get_shared_plane (const unsigned int z0, const unsigned int c0=0)

  Return a shared-memory image referencing one plane (z0,c0) of the image instance.

- const CImg< T > **get_shared_plane** (const unsigned int z0, const unsigned int c0=0) const
- CImg< T > get_shared_channels (const unsigned int c0, const unsigned int c1)

  Return a shared-memory image referencing a set of channels (c0->c1) of the image instance.

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

  Return a shared-memory image referencing one channel c0 of the image instance.

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

  Return a shared version of the image instance.

- const CImg< T > **get_shared** () const
- CImgList< T > get_split (const char axis, const int nb=0) const

  Split image into a list.

- CImgList< T > get_split (const T value, const bool keep_values, const bool is_shared) const

  Split image into a list of one-column vectors, according to specified splitting value.

- template<typename t >
  CImgList< T > get_split (const CImg< t > &values, const bool keep_values, const bool is_shared) const

  Split image into a list of one-column vectors, according to specified sequence of splitting values.

- template<typename t >
  CImg< T > & append (const CImg< t > &img, const char axis='x', const float align=0)

  Append an image.

- CImg< T > & **append** (const CImg< T > &img, const char axis='x', const float align=0)
- 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
- CImg< T > **get_append** (const CImg< T > &img, const char axis='x', const float align=0) const

Filtering / Transforms

- template<typename t >
  CImg< T > & correlate (const CImg< t > &mask, const unsigned int border_-conditions=1, const bool is_normalized=false)

  Compute the correlation of the image instance by a mask.
- template<typename t >
  CImg< typename cimg::superset2 < T, t, float >::type > **get_correlate** (const CImg< t > &mask, const unsigned int border_conditions=1, const bool is_-normalized=false) const
- template<typename t >
  CImg< T > & convolve (const CImg< t > &mask, const unsigned int border_-conditions=1, const bool is_normalized=false)

  Compute the convolution of the image by a mask.
- template<typename t >
  CImg< typename cimg::superset2 < T, t, float >::type > **get_convolve** (const CImg< t > &mask, const unsigned int border_conditions=1, const bool is_-normalized=false) const
- template<typename t >
  CImg< T > & erode (const CImg< t > &mask, const unsigned int border_-conditions=1, const bool is_normalized=false)

  Return the erosion of the image by a structuring element.
- template<typename t >
  CImg< typename cimg::superset < T, t >::type > **get_erode** (const CImg< t > &mask, const unsigned int border_conditions=1, const bool is_normalized=false) const
- CImg< T > & erode (const unsigned int sx, const unsigned int sy, const unsigned int sz=1)

  Erode the image by a rectangular structuring element of size sx,sy,sz.
- CImg< T > **get_erode** (const unsigned int sx, const unsigned int sy, const un-signed int sz=1) const
- CImg< T > & erode (const unsigned int s)

  Erode the image by a square structuring element of size sx.
- CImg< T > **get_erode** (const unsigned int s) const
- template<typename t >
  CImg< T > & dilate (const CImg< t > &mask, const unsigned int border_-conditions=1, const bool is_normalized=false)

  Dilate the image by a structuring element.
- template<typename t >
  CImg< typename cimg::superset < T, t >::type > **get_dilate** (const CImg< t > &mask, const unsigned int border_conditions=1, const bool is_normalized=false) const
- CImg< T > & dilate (const unsigned int sx, const unsigned int sy, const unsigned int sz=1)

  Dilate the image by a rectangular structuring element of size sx,sy,sz.
- CImg< T > **get_dilate** (const unsigned int sx, const unsigned int sy, const un-signed int sz=1) const

- CImg< T > & dilate (const unsigned int s)

  Erode the image by a square structuring element of size sx.
- CImg< T > **get_dilate** (const unsigned int s) const
- template<typename t >
  CImg< T > & watershed (const CImg< t > &priority, const bool fill_lines=true)

  Compute the watershed transform, from an image instance of non-zero labels.
- template<typename t >
  CImg< T > **get_watershed** (const CImg< t > &priority, const bool fill_lines=true) const
- CImg< T > & deriche (const float sigma, const int order=0, const char axis='x', const bool cond=true)

  Compute the result of the Deriche filter.
- CImg< Tfloat > **get_deriche** (const float sigma, const int order=0, const char axis='x', const bool cond=true) const
- CImg< T > & blur (const float sigmax, const float sigmay, const float sigmaz, const bool cond=true)

  Return a blurred version of the image, using a Canny-Deriche filter.
- CImg< Tfloat > **get_blur** (const float sigmax, const float sigmay, const float sigmaz, const bool cond=true) const
- CImg< T > & blur (const float sigma, const bool cond=true)

  Return a blurred version of the image, using a Canny-Deriche filter.
- CImg< Tfloat > **get_blur** (const float sigma, const bool cond=true) const
- 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 fast_approx=1)

  Blur the image anisotropically following a field of diffusion tensors.
- template<typename t >
  CImg< T > **get_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 fast_approx=true) const
- 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 fast_approx=true)

  Blur an image following in an anisotropic way.
- CImg< T > **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 fast_approx=true) const
- CImg< T > & blur_bilateral (const float sigma_x, const float sigma_y, const float sigma_z, const float sigma_r, const int bgrid_x, const int bgrid_y, const int bgrid_z, const int bgrid_r, const bool interpolation_type=true)

  Blur an image using the bilateral filter.
- CImg< T > **get_blur_bilateral** (const float sigma_x, const float sigma_y, const float sigma_z, const float sigma_r, const int bgrid_x, const int bgrid_y, const int bgrid_z, const int bgrid_r, const bool interpolation_type=true) const

- CImg< T > & blur_bilateral (const float sigma_s, const float sigma_r, const int bgrid_s=-33, const int bgrid_r=32, const bool interpolation_type=true)

  Blur an image using the bilateral filter.
- CImg< T > **get_blur_bilateral** (const float sigma_s, const float sigma_r, const int bgrid_s=-33, const int bgrid_r=32, const bool interpolation_type=true) const
- CImg< T > & blur_patch (const float sigma_s, const float sigma_p, const unsigned int patch_size=3, const unsigned int lookup_size=4, const float smoothness=0, const bool fast_approx=true)

  Blur an image in its patch-based space.
- CImg< T > **get_blur_patch** (const float sigma_s, const float sigma_p, const unsigned int patch_size=3, const unsigned int lookup_size=4, const float smoothness=0, const bool fast_approx=true) const
- CImg< T > & blur_median (const unsigned int n)

  Apply a median filter.
- CImg< T > **get_blur_median** (const unsigned int n) const
- 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 using anisotropic shock filters or inverse diffusion.
- CImg< T > **get_sharpen** (const float amplitude, const bool sharpen_type=false, const float edge=1, const float alpha=0, const float sigma=0) const
- CImgList< Tfloat > get_gradient (const char ∗const axes=0, const int scheme=3) const

  Compute the list of images, corresponding to the XY-gradients of an image.
- CImgList< Tfloat > get_hessian (const char ∗const axes=0) const

  Return components of the Hessian matrix of an image.
- CImg< T > & laplacian ()

  Compute the laplacian of the image instance.
- CImg< Tfloat > **get_laplacian** () const
- CImg< T > & structure_tensors (const unsigned int scheme=2)

  Compute the structure tensor field of an image.
- CImg< Tfloat > **get_structure_tensors** (const unsigned int scheme=2) const
- 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)

  Return a diffusion tensor for edge-preserving anisotropic smoothing of an image.
- CImg< 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
- CImg< T > & displacement (const CImg< T > &source, const float smoothness=0.1f, const float precision=5.0f, const unsigned int nb_scales=0, const unsigned int iteration_max=10000, const bool is_backward=false)

  Estimate a displacement field between specified source image and image instance.
- CImg< Tfloat > get_displacement (const CImg< T > &source, const float smoothness=0.1f, const float precision=5.0f, const unsigned int nb_scales=0, const unsigned int iteration_max=10000, const bool is_backward=false) const

*Estimate a displacement field between specified source image and image instance **[new-instance version]**.*

- CImg< T > & distance (const T value, const unsigned int metric=2)

  Compute the distance transform according to a specified value.

- CImg< Tfloat > **get_distance** (const T value, const unsigned int metric=2) const
- template<typename t >
  CImg< T > & distance (const T value, const CImg< t > &metric_mask)

  Compute the chamfer distance transform according to a specified value, with a custom metric.

- template<typename t >
  CImg< Tfloat > **get_distance** (const T value, const CImg< t > &metric_mask) const
- CImg< T > & distance_dijkstra (const unsigned int x=0, const unsigned int y=0, const unsigned int z=0)

  Compute the distance map to one specified point.

- CImg< Tfloat > get_distance_dijkstra (const unsigned int x=0, const unsigned int y=0, const unsigned int z=0) const

  *Compute the distance map to one specified point **[new-instance version]**.*

- CImg< T > & distance_eikonal (const unsigned int nb_iterations, const float band_size=0, const float time_step=0.5f)

  Compute distance function from 0-valued isophotes by the application of an Eikonal PDE.

- CImg< Tfloat > **get_distance_eikonal** (const unsigned int nb_iterations, const float band_size=0, const float time_step=0.5f) const
- CImg< T > & haar (const char axis, const bool invert=false, const unsigned int nb_scales=1)

  Compute the Haar multiscale wavelet transform (monodimensional version).

- CImg< Tfloat > **get_haar** (const char axis, const bool invert=false, const unsigned int nb_scales=1) const
- CImg< T > & haar (const bool invert=false, const unsigned int nb_scales=1)

  Compute the Haar multiscale wavelet transform.

- CImg< Tfloat > **get_haar** (const bool invert=false, const unsigned int nb_scales=1) const
- CImgList< Tfloat > get_FFT (const char axis, const bool invert=false) const

  Compute a 1d Fast Fourier Transform, along a specified axis.

- CImgList< Tfloat > get_FFT (const bool invert=false) const

  Compute a n-d Fast-Fourier Transform.

- static void FFT (CImg< T > &real, CImg< T > &imag, const char axis, const bool invert=false)

  Compute a 1d Fast Fourier Transform, along a specified axis.

- static void FFT (CImg< T > &real, CImg< T > &imag, const bool invert=false)

  Compute a n-d Fast Fourier Transform.

3d Objects Management

- CImg< T > & shift_object3d (const float tx, const float ty=0, const float tz=0)

  Shift a 3d object.
- CImg< Tfloat > **get_shift_object3d** (const float tx, const float ty=0, const float tz=0) const
- CImg< T > & shift_object3d ()

  Shift a 3d object so that it becomes centered.
- CImg< Tfloat > **get_shift_object3d** () const
- CImg< T > & resize_object3d (const float sx, const float sy=-100, const float sz=-100)

  Resize a 3d object.
- CImg< Tfloat > **get_resize_object3d** (const float sx, const float sy=-100, const float sz=-100) const
- CImg< T > resize_object3d ()

  Resize a 3d object so that its max dimension if one.
- CImg< Tfloat > **get_resize_object3d** () const
- template<typename tf , typename tp , typename tff >
  CImg< T > & append_object3d (CImgList< tf > &primitives, const CImg< tp > &obj_vertices, const CImgList< tff > &obj_primitives)

  Append a 3d object to another one.
- 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 >::empty()) const

  Texturize primitives of a 3d object.
- template<typename tf , typename tc , typename te >
  CImg< floatT > get_elevation3d (CImgList< tf > &primitives, CImgList< tc > &colors, const CImg< te > &elevation) const

  Create and return a 3d elevation of the image instance.
- 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

  Create and return the 3d projection planes of the image instance.
- template<typename tf >
  CImg< floatT > get_isoline3d (CImgList< tf > &primitives, const float isovalue, const int size_x=-100, const int size_y=-100) const

  Create and return a isoline of the image instance as a 3d object.
- 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

  Create and return a isosurface of the image instance as a 3d object.
- template<typename tp , typename tc , typename to >
  CImg< T > & object3dtoCImg3d (const CImgList< tp > &primitives, const CImgList< tc > &colors, const to &opacities)

Convert a 3d object into a CImg3d.

- template<typename tp , typename tc >
  CImg< T > & **object3dtoCImg3d** (const CImgList< tp > &primitives, const CImgList< tc > &colors)
- template<typename tp >
  CImg< T > & **object3dtoCImg3d** (const CImgList< tp > &primitives)
- CImg< T > & **object3dtoCImg3d** ()
- template<typename tp , typename tc , typename to >
  CImg< floatT > **get_object3dtoCImg3d** (const CImgList< tp > &primitives, const CImgList< tc > &colors, const to &opacities) const
- template<typename tp , typename tc >
  CImg< floatT > **get_object3dtoCImg3d** (const CImgList< tp > &primitives, const CImgList< tc > &colors) const
- template<typename tp >
  CImg< floatT > **get_object3dtoCImg3d** (const CImgList< tp > &primitives) const
- CImg< floatT > **get_object3dtoCImg3d** () const
- template<typename tp , typename tc , typename to >
  CImg< T > get_CImg3dtoobject3d (CImgList< tp > &primitives, CImgList< tc > &colors, CImgList< to > &opacities) const

  Convert a CImg3d (one-column image) into a 3d object.
- template<typename tp , typename tc , typename to >
  CImg< T > & **CImg3dtoobject3d** (CImgList< tp > &primitives, CImgList< tc > &colors, CImgList< to > &opacities)
- 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)

  Return elevation3d of a function.
- template<typename tf >
  static CImg< floatT > **elevation3d** (CImgList< tf > &primitives, const char ∗const expression, const float x0, const float y0, const float x1, const float y1, const int sizex=256, const int sizey=256)
- 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 sizex=256, const int sizey=256)

  Return isoline as a 3d object.
- template<typename tf >
  static CImg< floatT > **isoline3d** (CImgList< tf > &primitives, const char ∗const expression, const float isovalue, const float x0, const float y0, const float x1, const float y1, const int sizex=256, const int sizey=256)
- 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)

  Return isosurface as a 3d object.

- template< typename tf >
  static CImg< floatT > **isosurface3d** (CImgList< tf > &primitives, const char ∗const expression, const float isovalue, const float x0, const float y0, const float z0, const float x1, const float y1, const float z1, const int dx=32, const int dy=32, const int dz=32)
- 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)

  Create and return a 3d box object.
- template< typename tf >
  static CImg< floatT > cone3d (CImgList< tf > &primitives, const float radius=50, const float size_z=100, const unsigned int subdivisions=24)

  Create and return a 3d cone.
- template< typename tf >
  static CImg< floatT > cylinder3d (CImgList< tf > &primitives, const float radius=50, const float size_z=100, const unsigned int subdivisions=24)

  Create and return a 3d cylinder.
- 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)

  Create and return a 3d torus.
- 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)

  Create and return a 3d XY-plane.
- template< typename tf >
  static CImg< floatT > sphere3d (CImgList< tf > &primitives, const float radius=50, const unsigned int subdivisions=3)

  Create and return a 3d sphere.
- template< typename tf , typename t >
  static CImg< floatT > ellipsoid3d (CImgList< tf > &primitives, const CImg< t > &tensor, const unsigned int subdivisions=3)

  Create and return a 3d ellipsoid.

Drawing Functions

- template< typename tc >
  CImg< T > & draw_point (const int x0, const int y0, const tc ∗const color, const float opacity=1)

  Draw a 2d colored point (pixel).
- 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 colored point (voxel).

- template<typename t , typename tc >
  CImg< T > & **draw_point** (const CImg< t > &points, const tc *const color, const float opacity=1)
- template<typename tc >
  CImg< T > & draw_line (const int x0, const int y0, const int x1, const int y1, const tc *const color, const float opacity=1, const unsigned int pattern=~0U, const bool init_hatch=true)

  Draw a 2d colored line.
- template<typename tz , typename tc >
  CImg< T > & draw_line (CImg< tz > &zbuffer, const int x0, const int y0, const float z0, const int x1, const 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 colored line, with z-buffering.
- template<typename tc >
  CImg< T > & draw_line (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, const unsigned int pattern=~0U, const bool init_hatch=true)

  Draw a 3d colored line.
- template<typename tc >
  CImg< T > & draw_line (const int x0, const int y0, const int x1, const int y1, 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 2d textured line.
- template<typename tc >
  CImg< T > & draw_line (const int x0, const int y0, const float z0, const int x1, const 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 2d textured line, with perspective correction.
- template<typename tz , typename tc >
  CImg< T > & draw_line (CImg< tz > &zbuffer, const int x0, const int y0, const float z0, const int x1, const 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 2d textured line, with z-buffering and perspective correction.
- template<typename t , typename tc >
  CImg< T > & draw_line (const CImg< t > &points, const tc *const color, const float opacity=1, const unsigned int pattern=~0U, const bool init_hatch=true)

  Draw a set of consecutive colored lines in the image instance.
- 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 colored arrow in the image instance.
- 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=0.25, const unsigned int pattern=∼0U, const bool init_hatch=true)

Draw a cubic spline curve in the image instance.

- template<typename tc >
CImg< T > & draw_spline (const int x0, const int y0, const int z0, const float u0, const float v0, const float w0, const int x1, const int y1, const int z1, const float u1, const float v1, const float w1, const tc ∗const color, const float opacity=1, const float precision=4, const unsigned int pattern=∼0U, const bool init_hatch=true)

  Draw a cubic spline curve in the image instance (for volumetric images).

- 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 cubic spline curve in the image instance.

- 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 close_-set=false, const float precision=4, const unsigned int pattern=∼0U, const bool init_hatch=true)

- template<typename tp , typename tc >
CImg< T > & draw_spline (const CImg< tp > &points, const tc ∗const color, const float opacity=1, const bool close_set=false, const float precision=4, const unsigned int pattern=∼0U, const bool init_hatch=true)

  Draw a set of consecutive colored splines in the image instance.

- 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 2d filled colored triangle.

- 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 2d outlined colored triangle.

- template<typename tz , typename tc >
CImg< T > & draw_triangle (CImg< tz > &zbuffer, const int x0, const int y0, const float z0, const int x1, const int y1, const float z1, const int x2, const int y2, const float z2, const tc ∗const color, const float opacity=1, const float bright-ness=1)

  Draw a 2d filled colored triangle, with z-buffering.

- 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 brightness0, const float brightness1, const float brightness2, const float opacity=1)

  Draw a 2d Gouraud-shaded colored triangle.

- template<typename tz , typename tc >
CImg< T > & draw_triangle (CImg< tz > &zbuffer, const int x0, const int y0,

const float z0, const int x1, const int y1, const float z1, const int x2, const int y2, const float z2, const tc ∗const color, const float brightness0, const float brightness1, const float brightness2, const float opacity=1)

Draw a 2d Gouraud-shaded colored triangle, with z-buffering.

- template<typename tc1 , typename tc2 , typename tc3 >
  CImg< T > & draw_triangle (const int x0, const int y0, const int x1, const int y1, const int x2, const int y2, const tc1 ∗const color1, const tc2 ∗const color2, const tc3 ∗const color3, const float opacity=1)

  Draw a colored triangle with interpolated colors.

- 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 CImg< tc > &texture, const int tx0, const int ty0, const int tx1, const int ty1, const int tx2, const int ty2, const float opacity=1, const float brightness=1)

  Draw a 2d textured triangle.

- template<typename tc >
  CImg< T > & draw_triangle (const int x0, const int y0, const float z0, const int x1, const int y1, const float z1, const int x2, const int y2, const float z2, const CImg< tc > &texture, const int tx0, const int ty0, const int tx1, const int ty1, const int tx2, const int ty2, const float opacity=1, const float brightness=1)

  Draw a 2d textured triangle, with perspective correction.

- template<typename tz , typename tc >
  CImg< T > & draw_triangle (CImg< tz > &zbuffer, const int x0, const int y0, const float z0, const int x1, const int y1, const float z1, const int x2, const int y2, const float z2, const CImg< tc > &texture, const int tx0, const int ty0, const int tx1, const int ty1, const int tx2, const int ty2, const float opacity=1, const float brightness=1)

  Draw a 2d textured triangle, with z-buffering and perspective correction.

- template<typename tc , typename tl >
  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 CImg< tl > &light, const int lx0, const int ly0, const int lx1, const int ly1, const int lx2, const int ly2, const float opacity=1)

  Draw a 2d Pseudo-Phong-shaded triangle.

- template<typename tz , typename tc , typename tl >
  CImg< T > & draw_triangle (CImg< tz > &zbuffer, const int x0, const int y0, const float z0, const int x1, const int y1, const float z1, const int x2, const int y2, const float z2, const tc ∗const color, const CImg< tl > &light, const int lx0, const int ly0, const int lx1, const int ly1, const int lx2, const int ly2, const float opacity=1)

  Draw a 2d Pseudo-Phong-shaded triangle, with z-buffering.

- 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 CImg< tc > &texture, const int tx0, const int ty0, const int tx1, const int ty1, const int tx2, const int ty2, const float brightness0, const float brightness1, const float brightness2, const float opacity=1)

  Draw a 2d Gouraud-shaded textured triangle.

- template<typename tc >
  CImg< T > & draw_triangle (const int x0, const int y0, const float z0, const int x1, const int y1, const float z1, const int x2, const int y2, const float z2, const CImg< tc > &texture, const int tx0, const int ty0, const int tx1, const int ty1, const int tx2, const int ty2, const float brightness0, const float brightness1, const float brightness2, const float opacity=1)

  Draw a 2d Gouraud-shaded textured triangle, with perspective correction.
- template<typename tz , typename tc >
  CImg< T > & draw_triangle (CImg< tz > &zbuffer, const int x0, const int y0, const float z0, const int x1, const int y1, const float z1, const int x2, const int y2, const float z2, const CImg< tc > &texture, const int tx0, const int ty0, const int tx1, const int ty1, const int tx2, const int ty2, const float brightness0, const float brightness1, const float brightness2, const float opacity=1)

  Draw a 2d Gouraud-shaded textured triangle, with z-buffering and perspective correction.
- template<typename tc , typename tl >
  CImg< T > & draw_triangle (const int x0, const int y0, const int x1, const int y1, const int x2, const int y2, const CImg< tc > &texture, const int tx0, const int ty0, const int tx1, const int ty1, const int tx2, const int ty2, const CImg< tl > &light, const int lx0, const int ly0, const int lx1, const int ly1, const int lx2, const int ly2, const float opacity=1)

  Draw a 2d Pseudo-Phong-shaded textured triangle.
- template<typename tc , typename tl >
  CImg< T > & draw_triangle (const int x0, const int y0, const float z0, const int x1, const int y1, const float z1, const int x2, const int y2, const float z2, const CImg< tc > &texture, const int tx0, const int ty0, const int tx1, const int ty1, const int tx2, const int ty2, const CImg< tl > &light, const int lx0, const int ly0, const int lx1, const int ly1, const int lx2, const int ly2, const float opacity=1)

  Draw a 2d Pseudo-Phong-shaded textured triangle, with perspective correction.
- template<typename tz , typename tc , typename tl >
  CImg< T > & draw_triangle (CImg< tz > &zbuffer, const int x0, const int y0, const float z0, const int x1, const int y1, const float z1, const int x2, const int y2, const float z2, const CImg< tc > &texture, const int tx0, const int ty0, const int tx1, const int ty1, const int tx2, const int ty2, const CImg< tl > &light, const int lx0, const int ly0, const int lx1, const int ly1, const int lx2, const int ly2, const float opacity=1)

  Draw a 2d Pseudo-Phong-shaded textured triangle, with z-buffering and perspective correction.
- 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 4d filled rectangle in the image instance, at coordinates (`x0`,`y0`,`z0`,`c0`)-(`x1`,`y1`,`z1`,`c1`).
- 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 3d filled colored rectangle in the image instance, at coordinates (`x0`,`y0`,`z0`)-(`x1`,`y1`,`z1`).

- 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, const unsigned int pattern)

  Draw a 3d outlined colored rectangle in the image instance.

- 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 2d filled colored rectangle in the image instance, at coordinates ($x0$*,*$y0$*)-(*$x1$*,*$y1$*).*

- 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, const unsigned int pattern)

  Draw a 2d outlined colored rectangle.

- template<typename t , typename tc >
  CImg< T > & draw_polygon (const CImg< t > &points, const tc *const color, const float opacity=1)

  Draw a filled polygon in the image instance.

- template<typename t , typename tc >
  CImg< T > & draw_polygon (const CImg< t > &points, const tc *const color, const float opacity, const unsigned int pattern)

  Draw a outlined polygon in the image instance.

- 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 circle.

- 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)

  Draw an outlined circle.

- 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 ellipse.

- 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)

  Draw a filled ellipse.

- 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 ellipse.

- 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)

Draw an outlined ellipse.

- 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.

- CImg< T > & **draw_image** (const int x0, const int y0, const int z0, const int c0, const CImg< T > &sprite, const float opacity=1)
- template<typename t >
  CImg< T > & draw_image (const int x0, const int y0, const int z0, const CImg< t > &sprite, const float opacity=1)

  Draw an image.

- template<typename t >
  CImg< T > & draw_image (const int x0, const int y0, const CImg< t > &sprite, const float opacity=1)

  Draw an image.

- template<typename t >
  CImg< T > & draw_image (const int x0, const CImg< t > &sprite, const float opacity=1)

  Draw an image.

- template<typename t >
  CImg< T > & draw_image (const CImg< t > &sprite, const float opacity=1)

  Draw an image.

- 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_valmax=1)

  Draw a sprite image in the image instance (masked version).

- template<typename ti , typename tm >
  CImg< T > & draw_image (const int x0, const int y0, const int z0, const CImg< ti > &sprite, const CImg< tm > &mask, const float opacity=1, const float mask_valmax=1)

  Draw an image.

- template<typename ti , typename tm >
  CImg< T > & draw_image (const int x0, const int y0, const CImg< ti > &sprite, const CImg< tm > &mask, const float opacity=1, const float mask_valmax=1)

  Draw an image.

- template<typename ti , typename tm >
  CImg< T > & draw_image (const int x0, const CImg< ti > &sprite, const CImg< tm > &mask, const float opacity=1, const float mask_valmax=1)

  Draw an image.

- template<typename ti , typename tm >
  CImg< T > & draw_image (const CImg< ti > &sprite, const CImg< tm > &mask, const float opacity=1, const float mask_valmax=1)

  Draw an image.

- 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 > &font,...)

  Draw a text.
- 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 > &font,...)
- 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 > &font,...)
- 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,...)

  Draw a text.
- template<typename tc >
  CImg< T > & **draw_text** (const int x0, const int y0, const char ∗const text, const tc ∗const foreground_color, const int background_color=0, const float opacity=1, const unsigned int font_height=13,...)
- template<typename tc >
  CImg< T > & **draw_text** (const int x0, const int y0, const char ∗const text, const int, const tc ∗const background_color, const float opacity=1, const unsigned int font_height=13,...)
- 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 arrows=true, const unsigned int pattern=∼0U)

  Draw a vector field in the image instance, using a colormap.
- 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 arrows=true, const unsigned int pattern=∼0U)

  Draw a vector field in the image instance, using a colormap.
- template<typename t , typename tc >
  CImg< T > & draw_axis (const CImg< t > &xvalues, 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)

  Draw a labeled horizontal axis on the image instance.
- template<typename t , typename tc >
  CImg< T > & draw_axis (const int x, const CImg< t > &yvalues, const tc ∗const color, const float opacity=1, const unsigned int pattern=∼0U, const unsigned int font_height=13, const bool allow_zero=true)

  Draw a labeled vertical axis on the image instance.

- template< typename tx , typename ty , typename tc >
  CImg< T > & draw_axes (const CImg< tx > &xvalues, const CImg< ty > &yvalues, const tc *const color, const float opacity=1, const unsigned int patternx=~0U, const unsigned int patterny=~0U, const unsigned int font_height=13, const bool allow_zero=true)

  Draw a labeled horizontal+vertical axis on the image instance.

- template< typename tc >
  CImg< T > & draw_axes (const float x0, const float x1, const float y0, const float y1, const tc *const color, const float opacity=1, const int subdivisionx=-60, const int subdivisiony=-60, const float precisionx=0, const float precisiony=0, const unsigned int patternx=~0U, const unsigned int patterny=~0U, const unsigned int font_height=13)

  Draw a labeled horizontal+vertical axis on the image instance.

- template< typename tx , typename ty , typename tc >
  CImg< T > & draw_grid (const CImg< tx > &xvalues, const CImg< ty > &yvalues, const tc *const color, const float opacity=1, const unsigned int patternx=~0U, const unsigned int patterny=~0U)

  Draw grid.

- template< typename tc >
  CImg< T > & draw_grid (const float deltax, const float deltay, const float offsetx, const float offsety, const bool invertx, const bool inverty, const tc *const color, const float opacity=1, const unsigned int patternx=~0U, const unsigned int patterny=~0U)

  Draw grid.

- 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 on the image instance.

- template< typename tc , typename t >
  CImg< T > & draw_fill (const int x, const int y, const int z, const tc *const color, const float opacity, CImg< t > &region, const float sigma=0, const bool high_connexity=false)

  Draw a 3d filled region starting from a point (x,y,\ z) in the image instance.

- template< typename tc >
  CImg< T > & draw_fill (const int x, const int y, const int z, const tc *const color, const float opacity=1, const float sigma=0, const bool high_connexity=false)

  Draw a 3d filled region starting from a point (x,y,\ z) in the image instance.

- template< typename tc >
  CImg< T > & draw_fill (const int x, const int y, const tc *const color, const float opacity=1, const float sigma=0, const bool high_connexity=false)

  Draw a 2d filled region starting from a point (x,y) in the image instance.

- CImg< T > & draw_plasma (const float alpha=1, const float beta=0, const unsigned int scale=8)

  Draw a plasma random texture.

- template<typename tc >
  CImg< T > & draw_mandelbrot (const int x0, const int y0, const int x1, const int y1, const CImg< tc > &color_palette, 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 normalized_iteration=false, const bool julia_set=false, const double paramr=0, const double parami=0)

  Draw a quadratic Mandelbrot or Julia fractal set, computed using the Escape Time Algorithm.

- template<typename tc >
  CImg< T > & draw_mandelbrot (const CImg< tc > &color_palette, 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 normalized_iteration=false, const bool julia_set=false, const double paramr=0, const double parami=0)

  Draw a quadratic Mandelbrot or Julia fractal set, computed using the Escape Time Algorithm.

- 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 in the image instance.

- 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 an anisotropic 2d gaussian function.

- template<typename tc >
  CImg< T > & draw_gaussian (const int xc, const int yc, const float r1, const float r2, const float ru, const float rv, const tc *const color, const float opacity=1)

  Draw an anisotropic 2d gaussian function.

- template<typename tc >
  CImg< T > & draw_gaussian (const float xc, const float yc, const float sigma, const tc *const color, const float opacity=1)

  Draw an isotropic 2d gaussian function.

- template<typename t , typename tc >
  CImg< T > & draw_gaussian (const float xc, const float yc, const float zc, const CImg< t > &tensor, const tc *const color, const float opacity=1)

  Draw an anisotropic 3d gaussian function.

- template<typename tc >
  CImg< T > & draw_gaussian (const float xc, const float yc, const float zc, const float sigma, const tc *const color, const float opacity=1)

  Draw an isotropic 3d 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 double_sided=false, const float focale=500, const float lightx=0, const float lighty=0, const float lightz=-5e8, const float specular_light=0.2f, const float specular_shine=0.1f)

Draw a 3d object.

- template<typename tp , typename tf , typename tc , typename to , typename tz >
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, const bool double_sided, const float focale, const float lightx, const float lighty, const float lightz, const float specular_light, const float specular_shine, CImg< tz > &zbuffer)
- 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 CImgList< to > &opacities, const unsigned int render_type=4, const bool double_sided=false, const float focale=500, const float lightx=0, const float lighty=0, const float lightz=-5e8, const float specular_light=0.2f, const float specular_shine=0.1f)
- template<typename tp , typename tf , typename tc , typename to , typename tz >
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 CImgList< to > &opacities, const unsigned int render_type, const bool double_sided, const float focale, const float lightx, const float lighty, const float lightz, const float specular_light, const float specular_shine, CImg< tz > &zbuffer)
- template<typename tp , typename tf , typename tc >
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 unsigned int render_type=4, const bool double_sided=false, const float focale=500, const float lightx=0, const float lighty=0, const float lightz=-5e8, const float specular_light=0.2f, const float specular_shine=0.1f)

  Draw a 3d object.
- template<typename tp , typename tf , typename tc , typename tz >
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 unsigned int render_type, const bool double_sided, const float focale, const float lightx, const float lighty, const float lightz, const float specular-_light, const float specular_shine, CImg< tz > &zbuffer)

Data Input

- CImg< T > & select (CImgDisplay &disp, const unsigned int feature_type=2, unsigned int ∗const XYZ=0)

  Simple interface to select a shape from an image.
- CImg< T > & select (const char ∗const title, const unsigned int feature_type=2, unsigned int ∗const XYZ=0)

  Simple interface to select a shape from an image.
- CImg< intT > get_select (CImgDisplay &disp, const unsigned int feature_type=2, unsigned int ∗const XYZ=0) const

  Simple interface to select a shape from an image.

- CImg< intT > get_select (const char ∗const title, const unsigned int feature_-type=2, unsigned int ∗const XYZ=0) const

    Simple interface to select a shape from an image.
- CImg< intT > get_select_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

    Select sub-graph in a graph.
- CImg< T > & load (const char ∗const filename)

    Load an image from a file.
- CImg< T > & load_ascii (const char ∗const filename)

    Load an image from an ASCII file.
- CImg< T > & load_ascii (std::FILE ∗const file)

    Load an image from an ASCII file.
- CImg< T > & load_dlm (const char ∗const filename)

    Load an image from a DLM file.
- CImg< T > & load_dlm (std::FILE ∗const file)

    Load an image from a DLM file.
- CImg< T > & load_bmp (const char ∗const filename)

    Load an image from a BMP file.
- CImg< T > & load_bmp (std::FILE ∗const file)

    Load an image from a BMP file.
- CImg< T > & load_jpeg (const char ∗const filename)

    Load an image from a JPEG file.
- CImg< T > & load_jpeg (std::FILE ∗const file)

    Load an image from a JPEG file.
- CImg< T > & load_magick (const char ∗const filename)

    Load an image from a file, using Magick++ library.
- CImg< T > & load_png (const char ∗const filename)

    Load an image from a PNG file.
- CImg< T > & load_png (std::FILE ∗const file)

    Load an image from a PNG file.
- CImg< T > & load_pnm (const char ∗const filename)

    Load an image from a PNM file.
- CImg< T > & load_pnm (std::FILE ∗const file)

    Load an image from a PNM file.
- CImg< T > & load_pfm (const char ∗const filename)

    Load an image from a PFM file.
- CImg< T > & load_pfm (std::FILE ∗const file)

    Load an image from a PFM file.
- CImg< T > & load_rgb (const char ∗const filename, const unsigned int dimw, const unsigned int dimh=1)

    Load an image from a RGB file.

- CImg< T > & load_rgb (std::FILE ∗const file, const unsigned int dimw, const unsigned int dimh=1)

  Load an image from a RGB file.
- CImg< T > & load_rgba (const char ∗const filename, const unsigned int dimw, const unsigned int dimh=1)

  Load an image from a RGBA file.
- CImg< T > & load_rgba (std::FILE ∗const file, const unsigned int dimw, const unsigned int dimh=1)

  Load an image from a RGBA file.
- 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)

  Load an image from a TIFF file.
- CImg< T > & load_minc2 (const char ∗const filename)

  Load an image from a MINC2 file.
- CImg< T > & load_analyze (const char ∗const filename, float ∗const voxsize=0)

  Load an image from an ANALYZE7.5/NIFTI file.
- CImg< T > & load_analyze (std::FILE ∗const file, float ∗const voxsize=0)

  Load an image from an ANALYZE7.5/NIFTI file.
- CImg< T > & load_cimg (const char ∗const filename, const char axis='z', const float align=0)

  Load an image (list) from a .cimg file.
- CImg< T > & load_cimg (std::FILE ∗const file, const char axis='z', const float align=0)

  Load an image (list) from a .cimg file.
- 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 un-signed 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 a sub-image (list) from a .cimg file.
- CImg< T > & load_cimg (std::FILE ∗const file, const unsigned int n0, const un-signed 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 a sub-image (list) from a non-compressed .cimg file.
- CImg< T > & load_inr (const char ∗const filename, float ∗const voxsize=0)

  Load an image from an INRIMAGE-4 file.
- CImg< T > & load_inr (std::FILE ∗const file, float ∗const voxsize=0)

  Load an image from an INRIMAGE-4 file.
- CImg< T > & load_exr (const char ∗const filename)

  Load an image from a EXR file.
- CImg< T > & load_pandore (const char ∗const filename)

  Load an image from a PANDORE file.
- CImg< T > & load_pandore (std::FILE ∗const file)

Load an image from a PANDORE file.

- CImg< T > & load_parrec (const char ∗const filename, const char axis='c', const float align=0)

  Load an image from a PAR-REC (Philips) file.

- CImg< T > & load_raw (const char ∗const filename, const unsigned int sizex=0, const unsigned int sizey=1, const unsigned int sizez=1, const unsigned int sizev=1, const bool multiplexed=false, const bool invert_endianness=false)

  Load an image from a .RAW file.

- CImg< T > & load_raw (std::FILE ∗const file, const unsigned int sizex=0, const unsigned int sizey=1, const unsigned int sizez=1, const unsigned int sizev=1, const bool multiplexed=false, const bool invert_endianness=false)

  Load an image from a .RAW file.

- CImg< T > & load_ffmpeg (const char ∗const filename, const unsigned int first_-frame=0, const unsigned int last_frame=∼0U, const unsigned int step_frame=1, const bool pixel_format=true, const bool resume=false, const char axis='z', const float align=0)

  Load a video sequence using FFMPEG av's libraries.

- CImg< T > & load_yuv (const char ∗const filename, const unsigned int sizex, const unsigned int sizey=1, 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', const float align=0)

  Load an image sequence from a YUV file.

- CImg< T > & load_yuv (std::FILE ∗const file, const unsigned int sizex, const unsigned int sizey=1, 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', const float align=0)

  Load an image sequence from a YUV file.

- template<typename tf , typename tc >
  CImg< T > & load_off (const char ∗const filename, CImgList< tf > &primitives, CImgList< tc > &colors)

  Load a 3d object from a .OFF file.

- template<typename tf , typename tc >
  CImg< T > & load_off (std::FILE ∗const file, CImgList< tf > &primitives, CImg-List< tc > &colors)

  Load a 3d object from a .OFF file.

- CImg< T > & load_ffmpeg_external (const char ∗const filename, const char axis='z', const float align=0)

  Load a video sequence using FFMPEG's external tool 'ffmpeg'.

- CImg< T > & load_graphicsmagick_external (const char ∗const filename)

  Load an image using GraphicsMagick's external tool 'gm'.

- CImg< T > & load_gzip_external (const char ∗const filename)

  Load a gzipped image file, using external tool 'gunzip'.

- CImg< T > & load_imagemagick_external (const char ∗const filename)

  Load an image using ImageMagick's external tool 'convert'.

- CImg< T > & load_medcon_external (const char ∗const filename)

  Load a DICOM image file, using XMedcon's external tool 'medcon'.

- CImg< T > & load_dcraw_external (const char *const filename)

  Load a RAW Color Camera image file, using external tool 'dcraw'.
- CImg< T > & load_camera (const int camera_index=-1, const unsigned int skip-_frames=0, const bool release_camera=false)

  Load an image from a camera stream, using OpenCV.
- CImg< T > & load_other (const char *const filename)

  Load an image using ImageMagick's or GraphicsMagick's executables. If failed, try to load a .cimg[z] file format.
- static CImg< T > **get_load** (const char *const filename)
- static CImg< T > **get_load_ascii** (const char *const filename)
- static CImg< T > **get_load_ascii** (std::FILE *const file)
- static CImg< T > **get_load_dlm** (const char *const filename)
- static CImg< T > **get_load_dlm** (std::FILE *const file)
- static CImg< T > **get_load_bmp** (const char *const filename)
- static CImg< T > **get_load_bmp** (std::FILE *const file)
- static CImg< T > **get_load_jpeg** (const char *const filename)
- static CImg< T > **get_load_jpeg** (std::FILE *const file)
- static CImg< T > **get_load_magick** (const char *const filename)
- static CImg< T > **get_load_png** (const char *const filename)
- static CImg< T > **get_load_png** (std::FILE *const file)
- static CImg< T > **get_load_pnm** (const char *const filename)
- static CImg< T > **get_load_pnm** (std::FILE *const file)
- static CImg< T > **get_load_pfm** (const char *const filename)
- static CImg< T > **get_load_pfm** (std::FILE *const file)
- static CImg< T > **get_load_rgb** (const char *const filename, const unsigned int dimw, const unsigned int dimh=1)
- static CImg< T > **get_load_rgb** (std::FILE *const file, const unsigned int dimw, const unsigned int dimh=1)
- static CImg< T > **get_load_rgba** (const char *const filename, const unsigned int dimw, const unsigned int dimh=1)
- static CImg< T > **get_load_rgba** (std::FILE *const file, const unsigned int dimw, const unsigned int dimh=1)
- static CImg< 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)
- static CImg< T > **get_load_analyze** (const char *const filename, float *const voxsize=0)
- static CImg< T > **get_load_analyze** (std::FILE *const file, float *const vox-size=0)
- static CImg< T > **get_load_cimg** (const char *const filename, const char axis='z', const float align=0)
- static CImg< T > **get_load_cimg** (std::FILE *const file, const char axis='z', const float align=0)
- static CImg< 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)

- static CImg< 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)
- static CImg< T > **get_load_inr** (const char ∗const filename, float ∗const voxsize=0)
- static CImg< T > **get_load_inr** (std::FILE ∗const file, float ∗voxsize=0)
- static CImg< T > **get_load_exr** (const char ∗const filename)
- static CImg< T > **get_load_pandore** (const char ∗const filename)
- static CImg< T > **get_load_pandore** (std::FILE ∗const file)
- static CImg< T > **get_load_parrec** (const char ∗const filename, const char axis='c', const float align=0)
- static CImg< T > **get_load_raw** (const char ∗const filename, const unsigned int sizex=0, const unsigned int sizey=1, const unsigned int sizez=1, const unsigned int sizev=1, const bool multiplexed=false, const bool invert_endianness=false)
- static CImg< T > **get_load_raw** (std::FILE ∗const file, const unsigned int sizex=0, const unsigned int sizey=1, const unsigned int sizez=1, const unsigned int sizev=1, const bool multiplexed=false, const bool invert_endianness=false)
- static CImg< T > **get_load_ffmpeg** (const char ∗const filename, const unsigned int first_frame=0, const unsigned int last_frame=∼0U, const unsigned int step_frame=1, const bool pixel_format=true, const bool resume=false, const char axis='z', const float align=0)
- static CImg< T > **get_load_yuv** (const char ∗const filename, const unsigned int sizex, const unsigned int sizey=1, 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', const float align=0)
- static CImg< T > **get_load_yuv** (std::FILE ∗const file, const unsigned int sizex, const unsigned int sizey=1, 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', const float align=0)
- template<typename tf , typename tc >
  static CImg< T > **get_load_off** (const char ∗const filename, CImgList< tf > &primitives, CImgList< tc > &colors)
- template<typename tf , typename tc >
  static CImg< T > **get_load_off** (std::FILE ∗const file, CImgList< tf > &primitives, CImgList< tc > &colors)
- static CImg< T > **get_load_ffmpeg_external** (const char ∗const filename, const char axis='z', const float align=0)
- static CImg< T > **get_load_graphicsmagick_external** (const char ∗const filename)
- static CImg< T > **get_load_gzip_external** (const char ∗const filename)
- static CImg< T > **get_load_imagemagick_external** (const char ∗const filename)
- static CImg< T > **get_load_medcon_external** (const char ∗const filename)
- static CImg< T > **get_load_dcraw_external** (const char ∗const filename)
- static CImg< T > **get_load_camera** (const int camera_index=-1, const unsigned int skip_frames=0, const bool release_camera=false)
- static CImg< T > **get_load_other** (const char ∗const filename)

Data Output

- const CImg< T > & print (const char *const title=0, const bool display_-stats=true) const

  Display informations about the image on the standard error output.
- const CImg< T > & display (CImgDisplay &disp) const

  Display an image into a CImgDisplay window.
- const CImg< T > & display (CImgDisplay &disp, const bool display_info) const

  Display an image in a window with a title `title`, and wait a '_is_closed' or 'keyboard' event.

  .
- const CImg< T > & display (const char *const title=0, const bool display_-info=true) const

  Display an image in a window with a title `title`, and wait a '_is_closed' or 'keyboard' event.

  .
- 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 double_sided=true, const float focale=500, const float light_x=0, const float light_y=0, const float light_z=-5000, const float specular_light=0.2f, const float specular_shine=0.1f, const bool display_-axes=true, float *const pose_matrix=0) const

  High-level interface for displaying a 3d object.
- template<typename tp , typename tf , typename tc , typename to >
  const CImg< T > & display_object3d (const char *const title, 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 double_sided=true, const float focale=500, const float light_x=0, const float light_y=0, const float light_z=-5000, const float specular_light=0.2f, const float specular_shine=0.1f, const bool display_-axes=true, float *const pose_matrix=0) const

  High-level interface for displaying a 3d object.
- template<typename tp , typename tf , typename tc >
  const CImg< T > & display_object3d (CImgDisplay &disp, const CImg< tp > &vertices, const CImgList< tf > &primitives, const CImgList< tc > &colors, const bool centering=true, const int render_static=4, const int render_motion=1, const bool double_sided=true, const float focale=500, const float light_x=0, const float light_y=0, const float light_z=-5000, const float specular_light=0.2f, const float specular_shine=0.1f, const bool display_axes=true, float *const pose_matrix=0) const

  High-level interface for displaying a 3d object.
- template<typename tp , typename tf , typename tc >
  const CImg< T > & display_object3d (const char *const title, const CImg< tp > &vertices, const CImgList< tf > &primitives, const CImgList< tc > &colors, const bool centering=true, const int render_static=4, const int render_motion=1, const bool double_sided=true, const float focale=500, const float light_x=0, const float

light_y=0, const float light_z=-5000, const float specular_light=0.2f, const float specular_shine=0.1f, const bool display_axes=true, float ∗const pose_matrix=0) const

High-level interface for displaying a 3d object.

- template<typename tp , typename tf >
const CImg< T > & display_object3d (CImgDisplay &disp, const CImg< tp > &vertices, const CImgList< tf > &primitives, const bool centering=true, const int render_static=4, const int render_motion=1, const bool double_sided=true, const float focale=500, const float light_x=0, const float light_y=0, const float light_z=-5000, const float specular_light=0.2f, const float specular_shine=0.1f, const bool display_axes=true, float ∗const pose_matrix=0) const

  High-level interface for displaying a 3d object.

- template<typename tp , typename tf >
const CImg< T > & display_object3d (const char ∗const title, const CImg< tp > &vertices, const CImgList< tf > &primitives, const bool centering=true, const int render_static=4, const int render_motion=1, const bool double_sided=true, const float focale=500, const float light_x=0, const float light_y=0, const float light_z=-5000, const float specular_light=0.2f, const float specular_shine=0.1f, const bool display_axes=true, float ∗const pose_matrix=0) const

  High-level interface for displaying a 3d object.

- template<typename tp >
const CImg< T > & display_object3d (CImgDisplay &disp, const CImg< tp > &vertices, const bool centering=true, const int render_static=4, const int render_motion=1, const bool double_sided=true, const float focale=500, const float light_x=0, const float light_y=0, const float light_z=-5000, const float specular_light=0.2f, const float specular_shine=0.1f, const bool display_axes=true, float ∗const pose_matrix=0) const

  High-level interface for displaying a 3d object.

- template<typename tp >
const CImg< T > & display_object3d (const char ∗const title, const CImg< tp > &vertices, const bool centering=true, const int render_static=4, const int render_motion=1, const bool double_sided=true, const float focale=500, const float light_x=0, const float light_y=0, const float light_z=-5000, const float specular_light=0.2f, const float specular_shine=0.1f, const bool display_axes=true, float ∗const pose_matrix=0) const

  High-level interface for displaying a 3d object.

- 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

  High-level interface for displaying a graph.

- const CImg< T > & display_graph (const char ∗const title=0, 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

  High-level interface for displaying a graph.

- const CImg< T > & save (const char ∗const filename, const int number=-1) const

Save the image as a file.

- const CImg< T > & save_ascii (const char ∗const filename) const

  Save the image as an ASCII file (ASCII Raw + simple header).
- const CImg< T > & save_ascii (std::FILE ∗const file) const

  Save the image as an ASCII file (ASCII Raw + simple header).
- const CImg< T > & save_cpp (const char ∗const filename) const

  Save the image as a CPP source file.
- const CImg< T > & save_cpp (std::FILE ∗const file) const

  Save the image as a CPP source file.
- const CImg< T > & save_dlm (const char ∗const filename) const

  Save the image as a DLM file.
- const CImg< T > & save_dlm (std::FILE ∗const file) const

  Save the image as a DLM file.
- const CImg< T > & save_bmp (const char ∗const filename) const

  Save the image as a BMP file.
- const CImg< T > & save_bmp (std::FILE ∗const file) const

  Save the image as a BMP file.
- const CImg< T > & save_jpeg (const char ∗const filename, const unsigned int quality=100) const

  Save a file in JPEG format.
- const CImg< T > & save_jpeg (std::FILE ∗const file, const unsigned int quality=100) const

  Save a file in JPEG format.
- 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.
- const CImg< T > & save_png (const char ∗const filename, const unsigned int bytes_per_pixel=0) const

  Save a file in PNG format.
- const CImg< T > & save_png (std::FILE ∗const file, const unsigned int bytes_per_pixel=0) const

  Save a file in PNG format.
- const CImg< T > & save_pnm (const char ∗const filename, const unsigned int bytes_per_pixel=0) const

  Save the image as a PNM file.
- const CImg< T > & save_pnm (std::FILE ∗const file, const unsigned int bytes_per_pixel=0) const

  Save the image as a PNM file.
- const CImg< T > & save_pnk (const char ∗const filename) const

  Save the image as a PNK file (PINK library extension of PGM).
- const CImg< T > & save_pnk (std::FILE ∗const file) const

  Save the image as a PNk file (PINK library extension of PGM).
- const CImg< T > & save_pfm (const char ∗const filename) const

  Save the image as a PFM file.

- const CImg< T > & save_pfm (std::FILE ∗const file) const

  Save the image as a PFM file.
- const CImg< T > & save_rgb (const char ∗const filename) const

  Save the image as a RGB file.
- const CImg< T > & save_rgb (std::FILE ∗const file) const

  Save the image as a RGB file.
- const CImg< T > & save_rgba (const char ∗const filename) const

  Save the image as a RGBA file.
- const CImg< T > & save_rgba (std::FILE ∗const file) const

  Save the image as a RGBA file.
- const CImg< T > & save_tiff (const char ∗const filename, const unsigned int compression=0) const

  Save a file in TIFF format.
- const CImg< T > & save_minc2 (const char ∗const filename, const char ∗const imitate_file=0) const

  Save the image as a MINC2 file.
- const CImg< T > & save_analyze (const char ∗const filename, const float ∗const voxsize=0) const

  Save the image as an ANALYZE7.5 or NIFTI file.
- const CImg< T > & save_cimg (const char ∗const filename, const bool compression=false) const

  Save the image as a .cimg file.
- const CImg< T > & **save_cimg** (std::FILE ∗const file, const bool compression=false) const
- 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

  Insert the image into an existing .cimg file, at specified coordinates.
- const CImg< 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 into an existing .cimg file, at specified coordinates.
- const CImg< T > & save_inr (const char ∗const filename, const float ∗const voxsize=0) const

  Save the image as an INRIMAGE-4 file.
- const CImg< T > & save_inr (std::FILE ∗const file, const float ∗const voxsize=0) const

  Save the image as an INRIMAGE-4 file.
- const CImg< T > & save_exr (const char ∗const filename) const

  Save the image as a EXR file.
- const CImg< T > & save_pandore (const char ∗const filename, const unsigned int colorspace=0) const

  Save the image as a PANDORE-5 file.
- const CImg< T > & save_pandore (std::FILE ∗const file, const unsigned int colorspace=0) const

Save the image as a PANDORE-5 file.

- const CImg< T > & save_raw (const char ∗const filename, const bool multiplexed=false) const

  Save the image as a RAW file.
- const CImg< T > & save_raw (std::FILE ∗const file, const bool multiplexed=false) const

  Save the image as a RAW file.
- const CImg< T > & save_ffmpeg (const char ∗const filename, const unsigned int first_frame=0, const unsigned int last_frame=∼0U, const unsigned int fps=25, const unsigned int bitrate=2048) const

  Save the image as a video sequence file, using FFMPEG library.
- const CImg< T > & save_yuv (const char ∗const filename, const bool rgb2yuv=true) const

  Save the image as a YUV video sequence file.
- const CImg< T > & save_yuv (std::FILE ∗const file, const bool rgb2yuv=true) const

  Save the image as a YUV video sequence file.
- template<typename tf , typename tc >
  const CImg< T > & save_off (const char ∗const filename, const CImgList< tf > &primitives, const CImgList< tc > &colors) const

  Save OFF files.
- template<typename tf , typename tc >
  const CImg< T > & save_off (std::FILE ∗const file, const CImgList< tf > &primitives, const CImgList< tc > &colors) const

  Save OFF files.
- const CImg< T > & save_ffmpeg_external (const char ∗const filename, const unsigned int first_frame=0, const unsigned int last_frame=∼0U, const char ∗const codec=0, const unsigned int fps=25, const unsigned int bitrate=2048) const

  Save the image as a video sequence file, using the external tool 'ffmpeg'.
- const CImg< T > & save_graphicsmagick_external (const char ∗const filename, const unsigned int quality=100) const

  Save the image using GraphicsMagick's gm.
- const CImg< T > & save_gzip_external (const char ∗const filename) const

  Save an image as a gzipped file, using external tool 'gzip'.
- const CImg< T > & save_imagemagick_external (const char ∗const filename, const unsigned int quality=100) const

  Save the image using ImageMagick's convert.
- const CImg< T > & save_medcon_external (const char ∗const filename) const

  Save an image as a Dicom file (need '(X)Medcon' : `http://xmedcon.sourceforge.net`)
- const CImg< T > & **save_other** (const char ∗const filename, const unsigned int quality=100) const
- 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 an empty .cimg file with specified dimensions.

- 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)

  Save an empty .cimg file with specified dimensions.
- static CImg< T > **logo40x38** ()

8.1.1 Detailed Description

template<typename T>struct cimg_library::CImg< T >

Class representing an image (up to 4 dimensions wide), each pixel being 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 three first 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 that tells if 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 got 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 are stored in memory, in a non interlaced 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,false);` constructs a 256x256 color image from a `unsigned char*` buffer `data_buffer` (where R,G,B channels follow each others).
  - `CImg<unsigned char> img(data_buffer,256,256,1,3,true);` constructs a 256x256 color image from a `unsigned char*` buffer `data_buffer` (where R,G,B channels are multiplexed).

The complete list of constructors can be found `here`.

Most useful functions

The `CImg<T>` class contains a lot of functions that operates on images. Some of the most useful are :

- operator()() : allows to access or write pixel values.
- display() : displays the image in a new window.

8.1.2 Member Typedef Documentation

8.1.2.1 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 methods of `CImg<T>`.

Example

```
 CImg<float> img("reference.jpg");
  // Load image from file.
 for (CImg<float>::iterator it = img.begin(), it<img.end(); ++it) *it = 0
; // Set all pixels to '0', through a CImg iterator.
 img.fill(0);
  // Do the same with a built-in method.
```

See also

const_iterator.

8.1.2.2 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 methods of `CImg<T>`.

Example

```
const CImg<float> img("reference.jpg");
  // Load image from file.
float sum = 0;
for (CImg<float>::iterator it = img.begin(), it<img.end(); ++it) sum+=*
it; // Compute sum of all pixel values, through a CImg iterator.
const float sum2 = img.sum();
  // Do the same with a built-in method.
```

See also

iterator.

8.1.2.3 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 methods. It has been mainly defined for compatibility with STL naming conventions.

8.1.3 Constructor & Destructor Documentation

8.1.3.1 ∼**CImg ( )**

Destructor.

Destroy current image instance.

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).

See also

CImg(), assign().

8.1.3.2 CImg ( )

Default constructor.

Construct a new empty image instance.

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) methods, 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.
```

See also

~CImg(), assign(), is_empty().

8.1.3.3 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.

Construct a new image instance of size `size_x` x `size_y` x `size_z` x `size_c`, with pixels of type `T`.

Parameters

size_x	: Desired image width().
size_y	: Desired image height().
size_z	: Desired image depth().
size_c	: Desired image spectrum().

Note

- It is able to create only *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'.
```

See also

CImg(unsigned int,unsigned int,unsigned int,unsigned int,T), assign(unsigned int,unsigned int,unsigned int,unsigned int).

8.1.3.4 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.

Construct a new image instance of size `size_x` x `size_y` x `size_z` x `size_c`, with pixels of type `T`, and set all pixel values to specified `value`.

Parameters

size_x	: Desired image width().
size_y	: Desired image height().
size_z	: Desired image depth().
size_c	: Desired image spectrum().
value	: Value used for initialization.

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.

See also

CImg(unsigned int,unsigned int,unsigned int,unsigned int), assign(unsigned int,unsigned int,unsigned int,unsigned int,T).

8.1.3.5 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

size_x	: Desired image width().
size_y	: Desired image height().
size_z	: Desired image depth().
size_c	: Desired image spectrum().
value0	: First value of the initialization sequence (must be an *integer*).
value1	: Second value of the initialization sequence (must be an *integer*).
...	

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();
```

See also

CImg(unsigned int,unsigned int,unsigned int,unsigned int), CImg(unsigned int,unsigned int,unsigned int,unsigned int,double,double,...), assign(unsigned int,unsigned int,unsigned int,unsigned int,int,int,...).

8.1.3.6 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

size_x	: Desired image width().
size_y	: Desired image height().
size_z	: Desired image depth().
size_c	: Desired image spectrum().
value0	: First value of the initialization sequence (must be a *double*).
value1	: Second value of the initialization sequence (must be a *double*).
...	

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' !
```

See also

CImg(unsigned int,unsigned int,unsigned int,unsigned int), CImg(unsigned int,unsigned int,unsigned int,unsigned int,int,int,...), assign(unsigned int,unsigned int,unsigned int,unsigned int,double,double,...).

8.1.3.7 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

size_x	: Desired image width().
size_y	: Desired image height().
size_z	: Desired image depth().
size_c	: Desired image spectrum().
values	: Value string describing the way pixel values are set.
repeat_-values	: Flag telling if the value filling process is periodic.

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 filled from a value sequence.
                  img2(129,129,1,3,"if(c==0,255*abs(cos(x/10)),1.8*y)",
false); // Construct image filled from a formula.
 (img1,img2).display();
```

See also

CImg(unsigned int,unsigned int,unsigned int,unsigned int), assign(unsigned int,unsigned int,unsigned int,unsigned int,const char*,bool).

8.1.3.8 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

values	: Pointer to the input memory buffer.
size_x	: Desired image width().
size_y	: Desired image height().
size_z	: Desired image depth().
size_c	: Desired image spectrum().
is_shared	: Flag telling if 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] = { 0 };
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'.
```

See also

CImg(unsigned int,unsigned int,unsigned int,unsigned int), assign(const t*,unsigned int,unsigned int,unsigned int,unsigned int,bool), is_shared().

8.1.3.9 CImg (const char *const *filename*) [explicit]

Construct image from 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

filename	: Input image filename.

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 higly 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();
```

See also

CImg(unsigned int,unsigned int,unsigned int,unsigned int), assign(const char*).

8.1.3.10 CImg (const CImg< t > & *img*)

Copy constructor.

Construct a new image instance with pixels of type `T`, as a copy of an existing `CImg<t>` instance.

Parameters

img	: 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).

See also

CImg(const CImg<t>&,bool), assign(const CImg<t>&),

8.1.3.11 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

img	: Input image to copy.
is_shared	: Desired shared state of the constructed copy.

Note

- Similar to CImg(const CImg<t>&), except that it allows 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`.

See also

CImg(const CImg<t>&), assign(const CImg<t>&,bool).

8.1.3.12 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

img	: Input image from which dimensions are borrowed.
dimensions	: 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').
```

See also

CImg(unsigned int,unsigned int,unsigned int,unsigned int), CImg(const CImg<t>&,const char*,T), assign(const CImg<t>&,const char*).

8.1.3.13 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

img	: Input image from which dimensions are borrowed.
dimensions	: String describing the image size along the X,Y,Z and V-dimensions.
value	: Value used for initialization.

Note

- Similar to CImg(const CImg<t>&,const char*), but it also fills the pixel buffer with the specified `value`.

See also

CImg(unsigned int,unsigned int,unsigned int,unsigned int,T), CImg(const CImg<t>&,const char*), assign(const CImg<t>&,const char*,T).

8.1.3.14 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

disp	: 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.

See also

CImgDisplay, assign(const CImgDisplay&).

8.1.4 Member Function Documentation

8.1.4.1 CImg<T>& assign ( )

In-place version of the default constructor/destructor.

In-place version of the default constructor CImg(). It simply resets the instance to an empty image.

Note

- It reinitializes the current image instance to a new constructed image instance.
- Memory used by the previous pixel buffer of the image instance is deallocated if necessary (i.e. if instance was not empty nor shared).
- If the image instance was shared, it is replaced by a (non-shared) empty image without a deallocation process.
- It can be useful to force memory deallocation of a pixel buffer used by an image instance, before its formal destruction.

See also

CImg(), ~CImg().

8.1.4.2 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`)

In-place version of a constructor.

In-place version of the constructor CImg(unsigned int,unsigned int,unsigned int,unsigned int).

Parameters

size_x	: Desired image width().
size_y	: Desired image height().
size_z	: Desired image depth().
size_c	: Desired image spectrum().

Note

- It reinitializes the current image instance to a new constructed image instance.

See also

assign(unsigned int,unsigned int,unsigned int,unsigned int,T). CImg(unsigned int,unsigned int,unsigned int,unsigned int).

8.1.4.3 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*)

In-place version of a constructor.

In-place version of the constructor CImg(unsigned int,unsigned int,unsigned int,unsigned int,T).

Parameters

size_x	: Desired image width().
size_y	: Desired image height().
size_z	: Desired image depth().
size_c	: Desired image spectrum().
value	: Value for initialization.

Note

- It reinitializes the current image instance to a new constructed image instance.

See also

assign(unsigned int,unsigned int,unsigned int,unsigned int), CImg(unsigned int,unsigned int,unsigned int,unsigned int,T).

8.1.4.4 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*, *...*)

In-place version of a constructor.

In-place version of the constructor CImg(unsigned int,unsigned int,unsigned int,unsigned int,int,int,...).

Parameters

size_x	: Desired image width().
size_y	: Desired image height().
size_z	: Desired image depth().
size_c	: Desired image spectrum().
value0	: First value of the initialization sequence (must be an integer).
value1	: Second value of the initialization sequence (must be an integer).
...	

Note

- It reinitializes the current image instance to a new constructed image instance.

See also

assign(unsigned int,unsigned int,unsigned int,unsigned int,double,double,...), C-Img(unsigned int,unsigned int,unsigned int,unsigned int,int,int,...).

8.1.4.5 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*, ...)

In-place version of a constructor.

In-place version of the constructor CImg(unsigned int,unsigned int,unsigned int,unsigned int,double,double,...).

Parameters

size_x	: Desired image width().
size_y	: Desired image height().
size_z	: Desired image depth().
size_c	: Desired image spectrum().
value0	: First value of the initialization sequence (must be a double).
value1	: Second value of the initialization sequence (must be a double).
...	

Note

- It reinitializes the current image instance to a new constructed image instance.

See also

assign(unsigned int,unsigned int,unsigned int,unsigned int,int,int,...), CImg(unsigned int,unsigned int,unsigned int,unsigned int,double,double,...).

8.1.4.6 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*)

In-place version of a constructor.

In-place version of the constructor CImg(unsigned int,unsigned int,unsigned int,unsigned int,const char∗,bool).

Parameters

size_x	: Desired image width().
size_y	: Desired image height().
size_z	: Desired image depth().
size_c	: Desired image spectrum().
values	: Value string describing the way pixel values are set.
repeat_-values	: Flag telling if filling process is periodic.

Note

- It reinitializes the current image instance to a new constructed image instance.

See also

CImg(unsigned int,unsigned int,unsigned int,unsigned int,const char∗,bool).

8.1.4.7 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)

In-place version of a constructor.

In-place version of the constructor CImg(const t∗,unsigned int,unsigned int,unsigned int,unsigned int).

Parameters

values	: Pointer to the input memory buffer.
size_x	: Desired image width().
size_y	: Desired image height().
size_z	: Desired image depth().
size_c	: Desired image spectrum().

Note

- It reinitializes the current image instance to a new constructed image instance.

See also

assign(const t∗,unsigned int,unsigned int,unsigned int,unsigned int,bool). CImg(const t∗,unsigned int,unsigned int,unsigned int,unsigned int,bool).

8.1.4.8 CImg<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 bool *is_shared*)

In-place version of a constructor.

In-place version of the constructor CImg(const t∗,unsigned int,unsigned int,unsigned int,unsigned int,bool).

Parameters

values	: Pointer to the input memory buffer.
size_x	: Desired image width().
size_y	: Desired image height().
size_z	: Desired image depth().

size_c	: Desired image spectrum().
is_shared	: Flag telling if input memory buffer must be shared by the current instance.

Note

- It reinitializes the current image instance to a new constructed image instance.

See also

assign(const t∗,unsigned int,unsigned int,unsigned int,unsigned int). CImg(const t∗,unsigned int,unsigned int,unsigned int,unsigned int,bool).

8.1.4.9 CImg<T>& assign (const char ∗const *filename*)

In-place version of a constructor.

In-place version of the constructor CImg(const char∗).

Parameters

filename	: Input image filename.

Note

- It reinitializes the current image instance to a new constructed image instance.
- Equivalent to load(const char∗).

See also

CImg(const char∗), load(const char∗).

8.1.4.10 CImg<T>& assign (const CImg< t > & *img*)

In-place version of the default copy constructor.

In-place version of the constructor CImg(const CImg<t>&).

Parameters

img	: Input image to copy.

Note

- It reinitializes the current image instance to a new constructed image instance.

See also

assign(const CImg<t>&,bool), CImg(const CImg<t>&).

8.1.4.11 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).

Parameters

img	: Input image to copy.
is_shared	: Desired shared state of the constructed copy.

See also

assign(const CImg<t>&), CImg(const CImg<t>&,bool).

8.1.4.12 CImg<T>& assign (const CImg< t > & *img*, const char ∗const *dimensions*)

In-place version of a constructor.

In-place version of the constructor CImg(const CImg<t>&,const char∗).

Parameters

img	: Input image from which dimensions are borrowed.
dimensions	: String describing the image size along the X,Y,Z and V-dimensions.

Note

- It reinitializes the current image instance to a new constructed image instance.

See also

assign(const CImg<t>&,const char∗,T), CImg(const CImg<t>&,const char∗).

8.1.4.13 CImg<T>& assign (const CImg< t > & *img*, const char ∗const *dimensions*, const T *value*)

In-place version of a constructor.

In-place version of the constructor CImg(const CImg<t>&,const char∗,T).

Parameters

img	: Input image from which dimensions are borrowed.
dimensions	: String describing the image size along the X,Y,Z and V-dimensions.
value	: Value for initialization.

Note

- It reinitializes the current image instance to a new constructed image instance.

See also

assign(const CImg<t>&,const char∗), CImg(const CImg<t>&,const char∗,T).

8.1.4.14 CImg<T>& assign (const CImgDisplay & *disp*)

In-place version of a constructor.

In-place version of the constructor CImg(const CImgDisplay&).

Parameters

disp	: Input `CImgDisplay`.

Note

- It reinitializes the current image instance to a new constructed image instance.

See also

CImg(const CImgDisplay&).

8.1.4.15 CImg<T>& clear ( )

In-place version of the default constructor.

Equivalent to assign().

Note

- It has been defined for compatibility with STL naming conventions.

See also

assign().

8.1.4.16 CImg<t>& move_to (CImg< t > & *img*)

Transfer content of an image instance into another one.

Transfer 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

img	: 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.
```

See also

move_to(CImgList<t>&,unsigned int), swap(CImg<T>&).

8.1.4.17 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.

Transfer 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

list	: Destination list.
pos	: Position of the newly inserted image in the list.

Note

- When optionnal parameter `pos` is ommited, the image instance is transfered 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).
```

See also

move_to(CImg<t>&), swap(CImg<T>&).

8.1.4.18 CImg<T>& swap (CImg< T > & *img*)

Swap fields of two image instances.

Parameters

img	: 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'.
```

8.1.4.19 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());
```

8.1.4.20 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 method to get/set pixel values in `CImg<T>` images.

Parameters

x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	C-coordinate of the pixel value.

Note

- Range of pixel coordinates start from `(0,0,0,0)` to `(width()-1,height()-1,depth()-1,s`
- 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 debuging 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.
```

See also

at(), atX(), atXY(), atXYZ(), atXYZC().

8.1.4.21 T& operator() (const unsigned int *x*, const unsigned int *y*, const unsigned int *z*, const unsigned int *c*, const unsigned long *wh*, const unsigned long *whd* = `0`)

Access to a pixel value.

Parameters

x	: X-coordinate of the pixel value.
y	: Y-coordinate of the pixel value.
z	: Z-coordinate of the pixel value.
c	: C-coordinate of the pixel value.
wh	: Precomputed offset, must be equal to `width()*height()`.
whd	: Precomputed offset, must be equal to `width()*height()*depth()`.

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).

See also

operator()().

8.1.4.22 operator T ∗ ( )

Implicitely cast an image into a `T*`.

Implicitely 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 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 line.
img[510] = 255;              // Set pixel value at (10,5).
```

See also

operator()().

8.1.4.23 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

value	: 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').
```

See also

fill(const T).

8.1.4.24 CImg<T>& operator= (const char *const *expression*)

Assign pixels values from a specified expression.

Initialize all pixel values from the specified string `expression`.

Parameters

expression	: 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 three
100x100 scalar images with unitialized pixel 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();
```

See also

fill(const char∗, bool), load(const char∗).

8.1.4.25 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>&).

8.1.4.26 CImg<T>& operator= (const CImgDisplay & *disp*)

Copy the content of a display window to the current image instance.

Similar to assign(const CImgDisplay&).

8.1.4.27 CImg<T>& operator+= (const t *value*)

In-place addition operator.

Add specified `value` to all pixels of an image instance.

Parameters

value	: 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();
```

See also

operator+(const t) const, operator-=(const t), operator*=(const t), operator/=(const t), operator%=(const t), operator&=(const t), operator|=(const t), operator^=(const t), operator<<=(const t), operator>>=(const t).

8.1.4.28 CImg<T>& operator+= (const char *const *expression*)

In-place addition operator.

Add values to image pixels, according to the specified string `expression`.

Parameters

expression	: 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.

See also

operator+=(const t), operator=(const char*), operator+(const char*) const, operator-=(const char*), operator*=(const char*), operator/=(const char*), operator%=(const char*), operator&=(const char*), operator|=(const char*), operator^=(const char*), operator<<=(const char*), operator>>=(const char*).

8.1.4.29 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

img	: 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 less values are available in `img`, then the values are added cyclically. 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)
const CImg<float> img2(img1.width(),img.height(),1,1,"255*(x/w)^2"); //
Construct a scalar shading (img2.spectrum()==1).
img1+=img2;                                                                  //
Add shading to each channel of 'img1'.
img1.cut(0,255);                                                             //
Prevent [0,255] overflow.
(img2,img1).display();
```

See also

operator+(const CImg<t>&) const, operator=(const CImg<t>&), operator-=(const CImg<t>&), operator*=(const CImg<t>&), operator/=(const CImg<t>&), operator%=(const CImg<t>&), operator&=(const CImg<t>&), operator|=(const CImg<t>&), operator^=(const CImg<t>&), operator<<=(const CImg<t>&), operator>>=(const CImg<t>&).

8.1.4.30 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`.

See also

operator++(int), operator--().

8.1.4.31 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.

See also

operator++(), operator--(int).

8.1.4.32 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 informations about the shared state of the input image.
- Writing (+img) is equivalent to `CImg<T>(img,false)`.

See also

CImg(const CImg<T>&), CImg(const CImg<T>&,bool), operator-() const, operator∼() const.

8.1.4.33 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.

8.1.4.34 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.

8.1.4.35 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.

8.1.4.36 CImg<T>& operator-= (const t *value*)

In-place substraction operator.

Similar to operator+=(const t), except that it performs a substraction instead of an addition.

8.1.4.37 CImg<T>& operator-= (const char ∗const *expression*)

In-place substraction operator.

Similar to operator+=(const char∗), except that it performs a substraction instead of an addition.

8.1.4.38 CImg<T>& operator-= (const CImg< t > & *img*)

In-place substraction operator.

Similar to operator+=(const CImg<t>&), except that it performs a substraction instead of an addition.

8.1.4.39 CImg<T>& operator-- ( )

In-place decrement operator (prefix).

Similar to operator++(), except that it performs a decrement instead of an increment.

8.1.4.40 CImg<T> operator-- (int)

In-place decrement operator (postfix).

Similar to operator++(int), except that it performs a decrement instead of an increment.

8.1.4.41 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();
```

See also

operator+(), operator∼().

8.1.4.42 CImg< typename cimg::superset<T,t>::type > operator- (const t *value*) const

Substraction 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.

8.1.4.43 CImg<Tfloat> operator- (const char ∗const *expression*) const

Substraction 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.

8.1.4.44 CImg< typename cimg::superset<T,t>::type > operator- (const CImg< t > & *img*) const

Substraction 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.

8.1.4.45 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.

8.1.4.46 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.

8.1.4.47 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

img	: 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].
```

See also

operator*(const CImg<t>&) const, mul().

8.1.4.48 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.

8.1.4.49 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.

8.1.4.50 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.

8.1.4.51 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.

8.1.4.52 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.

8.1.4.53 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

img	: 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.

See also

operator/(const CImg<t>&) const, operator*(const CImg<t>&) const, div().

8.1.4.54 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.

8.1.4.55 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.

8.1.4.56 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.

8.1.4.57 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.

8.1.4.58 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.

8.1.4.59 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.

8.1.4.60 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.

8.1.4.61 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.

8.1.4.62 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.

8.1.4.63 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.

8.1.4.64 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.

8.1.4.65 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.

8.1.4.66 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`.

8.1.4.67 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`.

8.1.4.68 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`.

8.1.4.69 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.

8.1.4.70 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.

8.1.4.71 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.

8.1.4.72 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`.

8.1.4.73 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`.

8.1.4.74 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`.

8.1.4.75 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.

8.1.4.76 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.

8.1.4.77 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.

8.1.4.78 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`.

8.1.4.79 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`.

8.1.4.80 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`.

8.1.4.81 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.

8.1.4.82 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.

8.1.4.83 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.

8.1.4.84 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`.

8.1.4.85 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`.

8.1.4.86 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`.

8.1.4.87 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.

8.1.4.88 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.

8.1.4.89 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.

8.1.4.90 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`.

8.1.4.91 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`.

8.1.4.92 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`.

8.1.4.93 CImg<T> operator∼ ( ) const

Bitwise inversion operator.

Similar to operator-(), except that it compute the bitwise inverse instead of the opposite value.

8.1.4.94 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

value	: Reference value to compare with.

See also

operator==(const char ∗const) const, operator==(const CImg<t>&) const, operator!=(const T) const,

8.1.4.95 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

expression	: Value string describing the way pixel values are compared.

See also

operator==(const char ∗const) const, operator==(const CImg<t>&) const, operator!=(const T) const,

8.1.4.96 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 dimensions and pixel values, and `false` otherwise.

Parameters

img	: 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.");
}
```

See also

operator!=(const CImg<t>&) const.

8.1.4.97 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

value	: Reference value to compare with.

See also

operator==(const T) const, operator!=(const CImg<t>&) const.

8.1.4.98 bool operator!= (const char ∗const *expression*) const

Test if all pixel values of an image are different from a specified expression.

Return `true` is all pixels of the image instance are different to the specified `expression`.

Parameters

expression	: Value string describing the way pixel values are compared.

See also

operator==(const char *const) const, operator==(const CImg<t>&) const, operator!=(const T) const,

8.1.4.99 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

img	: input image to compare with.

Note

- Writing `img1!=img2` is equivalent to !(img1==img2).

See also

operator==().

8.1.4.100 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

img	: 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'.
```

See also

operator,(const CImgList<t>&) const, move_to(CImgList<t>&,unsigned int). CImgList<T>::insert(const CImg<t>&,unsigned int,bool).

8.1.4.101 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

list	: 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.

See also

operator,(const CImg<t>&) const, CImgList<T>::insert(const CImgList<t>&,unsigned int,bool).

8.1.4.102 CImgList<T> operator< (const char *axis*) const

Split image along specified axis.

Return a new list of images (`CImgList` instance) containing the splitted components of the instance image along the specified axis.

Parameters

axis	: 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();
```

See also

get_split(char,int) const.

8.1.4.103 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 may contain spaces (as in "`unsigned char`").
- If the pixel type `T` does not correspond to a registered type, the string "`unknown`" is returned.

See also

value_type.

8.1.4.104 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`.

See also

height(), depth(), spectrum(), size().

8.1.4.105 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`.

See also

width(), depth(), spectrum(), size().

8.1.4.106 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`.

See also

width(), height(), spectrum(), size().

8.1.4.107 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 indifferently stands for a RGBA or 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`.

See also

width(), height(), depth(), size().

8.1.4.108 unsigned long 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));
```

See also

width(), height(), depth(), spectrum().

8.1.4.109 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.

See also

operator T∗() const, data(unsigned int,unsigned int,unsigned int,unsigned int).

8.1.4.110 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

x	: X-coordinate of the pixel value.
y	: Y-coordinate of the pixel value.
z	: Z-coordinate of the pixel value.
c	: C-coordinate of the pixel value.

Note

- Writing `img.data(x,y,z,c)` is equivalent to `&(img(x,y,z,c))`. Thus, this method has the same properties as operator()(unsigned int,unsigned int,unsigned int,unsigned int).

See also

operator()(unsigned int,unsigned int,unsigned int,unsigned int), data().

8.1.4.111 long 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

x	: X-coordinate of the pixel value.
y	: Y-coordinate of the pixel value.
z	: Z-coordinate of the pixel value.
c	: 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 method 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.
```

See also

operator()(unsigned int,unsigned int,unsigned int,unsigned int), data(unsigned int,unsigned int,unsigned int,unsigned int).

8.1.4.112 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.

See also

data().

8.1.4.113 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 an strict upper bound for a CImg<T>::iterator.

Example

```
CImg<float> img(100,100,1,3);                                          //
Define a 100x100 RGB color image.
for (CImg<float>::iterator it = img.begin(); it<img.end(); ++it)   //
'img.end()' used here as an upper bound for the iterator.
  *it = 0;
```

See also

data().

8.1.4.114 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.

See also

data(), offset(), begin().

8.1.4.115 T& back ( )

Return a reference to the last pixel value.

Note

- Writing `img.end()` 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.

See also

data(), offset(), end().

8.1.4.116 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

offset	: Offset to the desired pixel value.
out_value	: Default value returned if `offset` 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 method is slower than operator()(). Use it when you are *not* sure about the validity of the specified pixel offset.

See also

operator()(), offset(), at(int).

8.1.4.117 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

offset	: 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 method 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 method `_at(int)`.

See also

operator()(), offset(), at(int,const T).

8.1.4.118 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

x	: X-coordinate of the pixel value.
y	: Y-coordinate of the pixel value.
z	: Z-coordinate of the pixel value.
c	: C-coordinate of the pixel value.
out_value	: Default value returned if (`x`,`y`,`z`,`c`) 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 method 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.

See also

operator()(), at(int,const T). atX(int,int,int,int), atXY(int,int,int,int,const T), atXYZ(int,int,int,int,const T), atXYZC(int,int,int,int,const T).

8.1.4.119 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

x	: X-coordinate of the pixel value.
y	: Y-coordinate of the pixel value.
z	: Z-coordinate of the pixel value.
c	: 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 method 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 method `_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.

See also

operator()(), at(int), atX(int,int,int,int,const T), atXY(int,int,int,int), atXYZ(int,int,int,int), atXYZC(int,int,int,int).

8.1.4.120 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.

8.1.4.121 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 method `_atXY(int,int,int,int).`

8.1.4.122 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.

8.1.4.123 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 method `_atXYZ(int,int,int,int).`

8.1.4.124 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.

8.1.4.125 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 method `_atXYZC(int,int,int,int)`.

8.1.4.126 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 (`fx`,`y`,`z`,`c`), or a specified default value in case of out-of-bounds access along the X-axis.

Parameters

fx	: X-coordinate of the pixel value (float-valued).
y	: Y-coordinate of the pixel value.
z	: Z-coordinate of the pixel value.
c	: C-coordinate of the pixel value.
out_value	: Default value returned if (`fx`,`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 co-ordinates 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.

See also

operator()(), atX(int,int,int,int,const T), linear_atX(float,int,int,int) const, linear_atXY(float,float,int,int,const T) const, linear_atXYZ(float,float,float,int,const T) const, linear_atXYZC(float,float,float,float,const T) const.

8.1.4.127 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 (`fx`,`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

fx	: X-coordinate of the pixel value (float-valued).
y	: Y-coordinate of the pixel value.
z	: Z-coordinate of the pixel value.
c	: 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 method `_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.

See also

operator()(), atX(int,int,int,int), linear_atX(float,int,int,int,const T) const, linear_atXY(float,float,int,int) const, linear_atXYZ(float,float,float,int) const, linear_atXYZC(float,float,float,float) const.

8.1.4.128 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.

8.1.4.129 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 method `_linear_atXY(float,float,int,int)`.

8.1.4.130 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.

8.1.4.131 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 method `_linear_atXYZ(float,float,float,int)`.

8.1.4.132 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 and 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.

8.1.4.133 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 method `_linear_atXYZC(float,float,float,float)`.

8.1.4.134 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 cubicly-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.

Parameters

fx	: X-coordinate of the pixel value (float-valued).
y	: Y-coordinate of the pixel value.
z	: Z-coordinate of the pixel value.
c	: C-coordinate of the pixel value.
out_value	: 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.

See also

operator()(), atX(int,int,int,int,const T), linear_atX(float,int,int,int,const T) const, cubic_atX(float,int,int,int) const, cubic_atXY(float,float,int,int,const T) const, cubic_atXYZ(float,float,float,int,const T) const.

8.1.4.135 Tfloat cubic_atX (const float *fx*, const int *y*, const int *z*, const int *c*, const T *out_value*, const Tfloat *min_value*, const Tfloat *max_value*) const

Return damped 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 you can specify the authorized minimum and maximum of the returned value.

8.1.4.136 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 cubicly-interpolated pixel value of the image instance located at (`fx`,`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

fx	: X-coordinate of the pixel value (float-valued).
y	: Y-coordinate of the pixel value.
z	: Z-coordinate of the pixel value.
c	: 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 method `_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.

See also

operator()(), atX(int,int,int,int), linear_atX(float,int,int,int) const, cubic_atX(float,int,int,int,const T) const, cubic_atXY(float,float,int,int) const, cubic_atXYZ(float,float,float,int) const.

8.1.4.137 Tfloat **cubic_atX** (const float *fx*, const int *y*, const int *z*, const int *c*, const Tfloat *min_value*, const Tfloat *max_value*) const

Return damped pixel value, using cubic interpolation and Neumann boundary conditions for the X-coordinate.

Similar to cubic_atX(float,int,int,int) const, except that you can specify the authorized minimum and maximum of the returned value.

8.1.4.138 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.

8.1.4.139 Tfloat **cubic_atXY** (const float *fx*, const float *fy*, const int *z*, const int *c*, const T *out_value*, const Tfloat *min_value*, const Tfloat *max_value*) const

Return damped pixel value, using cubic interpolation and Dirichlet boundary conditions for the X and Y-coordinates.

Similar to cubic_atXY(float,float,int,int,const T) const, except that you can specify the authorized minimum and maximum of the returned value.

8.1.4.140 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 method `_cubic_atXY(float,float,int,int)`.

8.1.4.141 Tfloat **cubic_atXY** (const float *fx*, const float *fy*, const int *z*, const int *c*, const Tfloat *min_value*, const Tfloat *max_value*) const

Return damped pixel value, using cubic interpolation and Neumann boundary conditions for the X and Y-coordinates.

Similar to cubic_atXY(float,float,int,int) const, except that you can specify the authorized minimum and maximum of the returned value.

8.1.4.142 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.

8.1.4.143 Tfloat **cubic_atXYZ** (const float *fx*, const float *fy*, const float *fz*, const int *c*, const T *out_value*, const Tfloat *min_value*, const Tfloat *max_value*) const

Return damped pixel value, using cubic interpolation and Dirichlet boundary conditions for the X,Y and Z-coordinates.

Similar to cubic_atXYZ(float,float,float,int,const T) const, except that you can specify the authorized minimum and maximum of the returned value.

8.1.4.144 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 method `_cubic_atXYZ(float,float,float,int)`.

8.1.4.145 Tfloat **cubic_atXYZ** (const float *fx*, const float *fy*, const float *fz*, const int *c*, const Tfloat *min_value*, const Tfloat *max_value*) const

Return damped pixel value, using cubic interpolation and Neumann boundary conditions for the X,Y and Z-coordinates.

Similar to cubic_atXYZ(float,float,float,int) const, except that you can specify the authorized minimum and maximum of the returned value.

8.1.4.146 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.

Set pixel value at specified coordinates (`fx`,`fy`,`z`,`c`) in the image instance, in a way that the value is spread amongst several neighbors if the pixel coordinates are indeed float-valued.

Parameters

value	: Pixel value to set.
fx	: X-coordinate of the pixel value (float-valued).
fy	: Y-coordinate of the pixel value (float-valued).
z	: Z-coordinate of the pixel value.
c	: C-coordinate of the pixel value.
is_added	: Boolean telling if the pixel value is added to (`true`), or simply replace (`false`) the current image pixel(s).

Returns

A reference to the current image instance.

Note

- If specified coordinates are outside image bounds, no operations are performed.

See also

linear_atXY(), set_linear_atXYZ().

8.1.4.147 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.

8.1.4.148 CImg<charT> value_string (const char *separator* = ',', const unsigned int *max_size* = 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

separator	: a `char` character which specifies the separator between values in the returned C-string.
max_size	: Maximum size of the returned image.

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`.

See also

pixel_type().

8.1.4.149 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 do not own his 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 methods (see list below).

See also

CImg(const t *const,unsigned int,unsigned int,unsigned int,unsigned int,bool), C-Img(const CImg<t>&), CImg(const CImg<t>&,bool), assign(const t *const,unsigned int,unsigned int,unsigned int,unsigned int,bool), get_shared_points(), get_shared_-line(), get_shared_lines(), get_shared_plane(), get_shared_planes(), get_shared-_channel(), get_shared_channels(), get_shared().

8.1.4.150 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.

See also

CImg(), assign().

8.1.4.151 bool is_inf ( ) const

Test if image instance contains a 'inf' value.

Return `true`, if image instance contains a 'inf' value, and `false` otherwise.

See also

is_nan().

8.1.4.152 bool **is_nan** ( ) const

Test if image instance contains a 'nan' value.

Return `true`, if image instance contains a 'nan' value, and `false` otherwise.

See also

is_inf().

8.1.4.153 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.

8.1.4.154 bool **is_sameXY** (const **CImg**< t > & *img*) const

Test if image width and height are the same as that of another image.

Test if is_sameX(const CImg<t>&) const and is_sameY(const CImg<t>&) const are both verified.

8.1.4.155 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.

8.1.4.156 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.

8.1.4.157 bool **is_sameXZ** (const **CImg**< t > & *img*) const

Test if image width and depth are the same as that of another image.

Test if is_sameX(const CImg<t>&) const and is_sameZ(const CImg<t>&) const are both verified.

8.1.4.158 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.

8.1.4.159 bool is_sameXC (const CImg< t > & *img*) const

Test if image width and spectrum are the same as that of another image.

Test if is_sameX(const CImg<t>&) const and is_sameC(const CImg<t>&) const are both verified.

8.1.4.160 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.

8.1.4.161 bool is_sameYZ (const CImg< t > & *img*) const

Test if image height and depth are the same as that of another image.

Test if is_sameY(const CImg<t>&) const and is_sameZ(const CImg<t>&) const are both verified.

8.1.4.162 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.

8.1.4.163 bool is_sameYC (const CImg< t > & *img*) const

Test if image height and spectrum are the same as that of another image.

Test if is_sameY(const CImg<t>&) const and is_sameC(const CImg<t>&) const are both verified.

8.1.4.164 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.

8.1.4.165 bool **is_sameZC** (const **CImg**< t > & *img*) const

Test if image depth and spectrum are the same as that of another image.

Test if is_sameZ(const CImg<t>&) const and is_sameC(const CImg<t>&) const are both verified.

8.1.4.166 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.

8.1.4.167 bool **is_sameXYZ** (const **CImg**< t > & *img*) const

Test if image width, height and depth are the same as that of another image.

Test if is_sameXY(const CImg<t>&) const and is_sameZ(const CImg<t>&) const are both verified.

8.1.4.168 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.

8.1.4.169 bool **is_sameXYC** (const **CImg**< t > & *img*) const

Test if image width, height and spectrum are the same as that of another image.

Test if is_sameXY(const CImg<t>&) const and is_sameC(const CImg<t>&) const are both verified.

8.1.4.170 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.

8.1.4.171 bool **is_sameXZC** (const **CImg**< t > & *img*) const

Test if image width, depth and spectrum are the same as that of another image.

Test if is_sameXZ(const CImg<t>&) const and is_sameC(const CImg<t>&) const are both verified.

8.1.4.172 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.

8.1.4.173 bool is_sameYZC (const CImg< t > & *img*) const

Test if image height, depth and spectrum are the same as that of another image.

Test if is_sameYZ(const CImg<t>&) const and is_sameC(const CImg<t>&) const are both verified.

8.1.4.174 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_same-C(unsigned int) const are both verified.

8.1.4.175 bool is_sameXYZC (const CImg< t > & *img*) const

Test if image width, height, depth and spectrum are the same as that of another image.

Test if is_sameXYZ(const CImg<t>&) const and is_sameC(const CImg<t>&) const are both verified.

8.1.4.176 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

x	: X-coordinate of the pixel value.
y	: Y-coordinate of the pixel value.
z	: Z-coordinate of the pixel value.
c	: 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`.

See also

contains(const T&,t&,t&,t&,t&) const, contains(const T&,t&,t&,t&) const, contains(const T&,t&,t&) const, contains(const T&,t&) const, contains(const T&) const.

8.1.4.177 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

	pixel	: Reference to pixel value to test.
`out`	*x*	: X-coordinate of the pixel value, if test succeeds.
`out`	*y*	: Y-coordinate of the pixel value, if test succeeds.
`out`	*z*	: Z-coordinate of the pixel value, if test succeeds.
`out`	*c*	: 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);
}
```

See also

containsXYZC(int,int,int,int) const, contains(const T&,t&,t&,t&) const, contains(const T&,t&,t&) const, contains(const T&,t&) const, contains(const T&) const.

8.1.4.178 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&,t&) const, except that only the X,Y and Z-coordinates are set.

8.1.4.179 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&,t&) const, except that only the X and Y-coordinates are set.

8.1.4.180 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&,t&) const, except that only the X-coordinate is set.

8.1.4.181 bool **contains** (const T & *pixel*) const

Test if pixel value is inside image bounds.

Similar to contains(const T&,t&,t&,t&,t&) const, except that no pixel coordinates are set.

8.1.4.182 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

img	: 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");
}
```

See also

is_shared().

8.1.4.183 bool is_object3d (const CImgList< tp > & *primitives*, const CImgList< tc > & *colors*, const to & *opacities*, const bool *is_full_check* = `true`, char *const *error_message* = 0) const

Test if the set {*this,primitives,colors,opacities} defines a valid 3d object.

Return `true` is the 3d object represented by the set {*this,primitives,colors,opacities} defines a valid 3d object, and `false` otherwise. The vertex coordinates are defined by the instance image.

Parameters

	primitives	: List of primitives of the 3d object.
	colors	: List of colors of the 3d object.
	opacities	: List (or image) of opacities of the 3d object.
	is_full_check	: Boolean telling if full checking of the 3d object must be performed.
`out`	*error_-message*	: C-string to contain the error message, if the test does not succeed.

Note

- Set `is_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.

See also

is_CImg3d(), draw_object3d(), display_object3d().

8.1.4.184 bool is_CImg3d (const bool *is_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

	is_full_check	: Boolean telling if full checking of the instance must be performed.
`out`	*error_-message*	: C-string to contain the error message, if the test does not succeed.

Note

- Set `is_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.

See also

is_object3d(), object3dtoCImg3d(const CImgList<tp>&,const CImgList<tc>&,const to&), draw_object3d(), display_object3d().

8.1.4.185 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 method 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();
```

See also

get_sqr() const, sqrt(), pow(), exp(), log(), log2(), log10().

8.1.4.186 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 method 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();
```

See also

get_sqrt() const,, sqr(), pow(), exp(), log(), log2(), log10().

8.1.4.187 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 method 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.

See also

get_exp() const, sqr(), sqrt(), pow(), log(), log2(), log10().

8.1.4.188 CImg<T>& log ( )

Compute the logarithm of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its logarithm $log_e(I_{(x,y,z,c)})$.

Note

- The **[in-place version]** of this method 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.

See also

get_log() const, sqr(), sqrt(), pow(), exp(), log2(), log10().

8.1.4.189 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 method 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.

See also

get_log2(), sqr(), sqrt(), pow(), exp(), log(), log10().

8.1.4.190 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 method 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.

See also

get_log10(), sqr(), sqrt(), pow(), exp(), log(), log2().

8.1.4.191 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 method 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.

See also

get_abs(), sign().

8.1.4.192 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 $sign(I_{(x,y,z,c)})$.

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 method 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.

See also

get_sign(), abs().

8.1.4.193 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 *radian*.
- The **[in-place version]** of this method 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.

See also

get_cos(), sin(), sinc(), tan().

8.1.4.194 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 *radian*.
- The **[in-place version]** of this method 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.

See also

get_sin(), cos(), sinc(), tan().

8.1.4.195 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 $sinc(I_{(x,y,z,c)})$.

Note

- Pixel values are regarded as being exin *radian*.
- The **[in-place version]** of this method 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.

See also

get_sinc(), cos(), sin(), tan().

8.1.4.196 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 exin *radian*.
- The **[in-place version]** of this method 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.

See also

get_tan(), cos(), sin(), sinc().

8.1.4.197 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 method 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.

See also

get_cosh(), sinh(), tanh().

8.1.4.198 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 method 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.

See also

get_sinh(), cosh(), tanh().

8.1.4.199 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 method 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.

See also

get_tanh(), cosh(), sinh().

8.1.4.200 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 $acos(I_{(x,y,z,c)})$.

Note

- The **[in-place version]** of this method 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.

See also

get_acos(), asin(), atan().

8.1.4.201 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 $asin(I_{(x,y,z,c)})$.

Note

- The **[in-place version]** of this method 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.

See also

get_asin(), acos(), atan().

8.1.4.202 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 $atan(I_{(x,y,z,c)})$.

Note

- The **[in-place version]** of this method 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.

See also

get_atan(), acos(), asin().

8.1.4.203 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 $atan2(I_{(x,y,z,c)})$.

Parameters

img	: The image whose pixel values specify the second argument of the `atan2()` function.

Note

- The **[in-place version]** of this method 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();
```

See also

get_atan2(), atan().

8.1.4.204 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

img	: 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();
```

See also

get_mul(), div(), operator*=(const CImg<t>&).

8.1.4.205 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.

See also

get_div(const CImg<t>&) const, operator/=(const CImg<t>&)

8.1.4.206 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	: Used exponent.

Note

- The **[in-place version]** of this method 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();
```

See also

get_pow(double) const, pow(const char*), pow(const CImg<t>&), sqr(), sqrt(), exp(), log(), log2(), log10().

8.1.4.207 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.

See also

get_pow(const char∗) const, pow(double), pow(CImg<t>&).

8.1.4.208 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.

See also

get_pow(const CImg<t>&) const, pow(double), pow(const char∗).

8.1.4.209 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.

See also

get_rol(unsigned int) const, rol(const char∗), rol(const CImg<t>&), operator<<=(unsigned int), operator>>=(unsigned int).

8.1.4.210 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.

See also

get_rol(const char∗) const, rol(unsigned int), rol(const CImg<t>&), operator<<=(const char∗), operator>>=(const char∗).

8.1.4.211 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.

See also

get_rol(const CImg<t>&) const, rol(unsigned int), rol(const char∗), operator<<=(const CImg<t>&), operator>>=(const CImg<t>&).

8.1.4.212 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.

See also

get_ror(unsigned int) const, ror(const char∗), ror(const CImg<t>&), operator<<=(unsigned int), operator>>=(unsigned int).

8.1.4.213 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.

See also

get_ror(const char∗) const, ror(unsigned int), ror(const CImg<t>&), operator<<=(const char∗), operator>>=(const char∗).

8.1.4.214 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.

See also

get_ror(const CImg<t>&), ror(unsigned int), ror(const char ∗), operator<<=(const CImg<t>&), operator>>=(const CImg<t>&).

8.1.4.215 Tdouble variance (const unsigned int *variance_method* = 1) const

Return the variance of the image.

Parameters

variance_-method	Determines how to calculate the variance	
	0	Second moment: $v = 1/N \sum_{k=1}^{N} (x_k - \bar{x})^2 = 1/N \left(\sum_{k=1}^{N} x_k^2 - \left(\sum_{k=1}^{N} x_k \right)^2 /N \right)$ with $\bar{x} = 1/N \sum_{k=1}^{N} x_k$
	1	Best unbiased estimator: $v = \frac{1}{N-1} \sum_{k=1}^{N} (x_k - \bar{x})^2$
	2	Least median of squares
	3	Least trimmed of squares

8.1.4.216 Tdouble **variance_noise** (const unsigned int *variance_method* = 2) const

Estimate noise variance of the image instance.

Parameters

variance_-method	: method to compute the variance

Note

Because of structures such as edges in images it is recommanded to use a robust 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.

See also

variance()

8.1.4.217 double **eval** (const char *const *expression*, const double *x* = 0, const double *y* = 0, const double *z* = 0, const double *c* = 0) const

Evaluate math expression.

If you make successive evaluations on the same image and with the same expression, you can set 'expr' to 0 after the first call, to skip the math parsing step.

8.1.4.218 static CImg<T> dijkstra (const tf & *distance*, const unsigned int *nb_nodes*, const unsigned int *starting_node*, const unsigned int *ending_node*, CImg< t > & *previous*) [static]

Compute minimal path in a graph, using the Dijkstra algorithm.

Parameters

distance	An object having operator()(unsigned int i, unsigned int j) which returns distance between two nodes (i,j).
nb_nodes	Number of graph nodes.
starting_-node	Indice of the starting node.
ending_-node	Indice of the ending node (set to ∼0U to ignore ending node).
previous	Array that gives the previous node indice in the path to the starting node (optional parameter).

Returns

Array of distances of each node to the starting node.

8.1.4.219 CImg<T>& dijkstra (const unsigned int *starting_node*, const unsigned int *ending_node*, CImg< t > & *previous*)

Return minimal path in a graph, using the Dijkstra algorithm.

image instance corresponds to the adjacency matrix of the graph.

Parameters

starting_-node	Indice of the starting node.
previous	Array that gives the previous node indice in the path to the starting node (optional parameter).

Returns

Array of distances of each node to the starting node.

8.1.4.220 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

interpolation-_type	Type of interpolation (can be 0=nearest int, 1=linear, 2=2nd-order RK, 3=4th-order RK.

8.1.4.221 CImg<T>& fill (const T *val*)

Fill an image by a value `val`.

Parameters

val	= fill value

Note

All pixel values of the image instance will be initialized by `val`.

8.1.4.222 CImg<T>& round (const double *y =* `1`, const int *rounding_type =* `0`)

Compute image with rounded pixel values.

Parameters

y	Rounding precision.
rounding_-type	Roundin type, can be 0 (nearest), 1 (forward), -1(backward).

8.1.4.223 CImg<T>& noise (const double *sigma,* const unsigned int *noise_type =* `0`)

Add random noise to the values of the image instance.

Parameters

sigma	Amplitude of the random additive noise. If `sigma<0`, it stands for a percentage of the global value range.
noise_type	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();
```

8.1.4.224 CImg<T>& normalize (const T *value_min*, const T *value_max*)

Linearly normalize values of the image instance between `value_min` and `value_-max`.

Parameters

value_min	Minimum desired value of the resulting image.
value_max	Maximum desired value of the resulting image.

Returns

A reference to the modified image instance.

Note

- Function `CImg<T>::get_normalize()` is also defined. It returns a non-shared modified copy of the image instance.

Example

```
const CImg<float> img("reference.jpg"), res = img.get_normalize(160,220)
;
(img,res).display();
```

8.1.4.225 CImg<T>& normalize ( )

Normalize multi-valued pixels of the image instance, with respect to their L2-norm.

Returns

A reference to the modified image instance.

Note

- Function `CImg<T>::get_normalize()` is also defined. It returns a non-shared modified copy of the image instance.

Example

```
const CImg<float> img("reference.jpg"), res = img.get_normalize();
(img,res.normalize(0,255)).display();
```

8.1.4.226 CImg<T>& norm (const int *norm_type* = 2)

Compute L2-norm of each multi-valued pixel of the image instance.

Parameters

norm_type	Type of computed vector norm (can be `0=Linf`, `1=L1` or `2=L2`).

Returns

A reference to the modified image instance.

Note

- Function `CImg<T>::get_norm()` is also defined. It returns a non-shared modified copy of the image instance.

Example

```
const CImg<float> img("reference.jpg"), res = img.get_norm();
(img,res.normalize(0,255)).display();
```

8.1.4.227 CImg<T>& cut (const T *value_min*, const T *value_max*)

Cut values of the image instance between `value_min` and `value_max`.

Parameters

value_min	Minimum desired value of the resulting image.
value_max	Maximum desired value of the resulting image.

Returns

A reference to the modified image instance.

Note

- Function `CImg<T>::get_cut()` is also defined. It returns a non-shared modified copy of the image instance.

Example

```
const CImg<float> img("reference.jpg"), res = img.get_cut(160,220);
(img,res).display();
```

8.1.4.228 CImg<T>& quantize (const unsigned int *nb_levels*, const bool *keep_range* = `true`)

Uniformly quantize values of the image instance into `nb_levels` levels.

Parameters

nb_levels	Number of quantization levels.
keep_range	Tells if resulting values keep the same range as the original ones.

Returns

A reference to the modified image instance.

Note

- Function `CImg<T>::get_quantize()` is also defined. It returns a non-shared modified copy of the image instance.

Example

```
const CImg<float> img("reference.jpg"), res = img.get_quantize(4);
(img,res).display();
```

8.1.4.229 CImg<T>& threshold (const T *value*, const bool *soft_threshold* = `false`, const bool *strict_threshold* = `false`)

Threshold values of the image instance.

Parameters

value	Threshold value
soft_-threshold	Tells if soft thresholding must be applied (instead of hard one).
strict_-threshold	Tells if threshold value is strict.

Returns

A reference to the modified image instance. Resulting pixel values are either equal to 0 or 1.

Note

- Function `CImg<T>::get_threshold()` is also defined. It returns a non-shared modified copy of the image instance.

Example

```
const CImg<float> img("reference.jpg"), res = img.get_threshold(128);
(img,res.normalize(0,255)).display();
```

8.1.4.230 CImg<T>& histogram (const unsigned int *nb_levels*, const T *value_min* = `(T)0`, const T *value_max* = `(T)0`)

Compute the histogram of the image instance.

Parameters

nb_levels	Number of desired histogram levels.
value_min	Minimum pixel value considered for the histogram computation. All pixel values lower than `value_min` will not be counted.
value_max	Maximum pixel value considered for the histogram computation. All pixel values higher than `value_max` will not be counted.

Returns

image instance is replaced by its histogram, defined as a `CImg<T>(nb_-levels)` image.

Note

- The histogram H of an image I is the 1d function where H(x) counts the number of occurences of the value x in the image I.
- If `value_min==value_max==0` (default behavior), the function first estimates the whole range of pixel values then uses it to compute the histogram.
- The resulting histogram is always defined in 1d. Histograms of multi-valued images are not multi-dimensional.
- Function `CImg<T>::get_histogram()` is also defined. It returns a non-shared modified copy of the image instance.

Example

```
const CImg<float> img = CImg<float>("reference.jpg").histogram(256);
img.display_graph(0,3);
```

8.1.4.231 CImg<T>& equalize (const unsigned int *nb_levels*, const T *value_min* = (T)0, const T *value_max* = (T)0)

Compute the histogram-equalized version of the image instance.

Parameters

nb_levels	Number of histogram levels used for the equalization.
value_min	Minimum pixel value considered for the histogram computation. All pixel values lower than `value_min` will not be counted.
value_max	Maximum pixel value considered for the histogram computation. All pixel values higher than `value_max` will not be counted.

Returns

A reference to the modified image instance.

Note

- If `value_min==value_max==0` (default behavior), the function first estimates the whole range of pixel values then uses it to equalize the histogram.
- Function `CImg<T>::get_equalize()` is also defined. It returns a non-shared modified copy of the image instance.

Example

```
const CImg<float> img("reference.jpg"), res = img.get_equalize(256);
(img,res).display();
```

8.1.4.232 CImg<T>& index (const CImg< t > & *palette*, const float *dithering* = 1, const bool *map_indexes* = false)

Index multi-valued pixels of the image instance, regarding to a predefined palette.

Parameters

palette	Multi-valued palette used as the basis for multi-valued pixel indexing.
dithering	Level of dithering (0=disable, 1=standard level).
map_-indexes	Tell if the values of the resulting image are the palette indices or the palette vectors.

Returns

A reference to the modified image instance.

Note

- `img.index(palette,dithering,1)` is equivalent to `img.-index(palette,dithering,0).map(palette)`.
- Function `CImg<T>::get_index()` is also defined. It returns a non-shared modified copy of the image instance.

Example

```
  const CImg<float> img("reference.jpg"), palette(3,1,1,3, 0,128,255, 0,12
8,255, 0,128,255);
  const CImg<float> res = img.get_index(palette,1,true);
  (img,res).display();
```

8.1.4.233 CImg<T>& map (const CImg< t > & *palette*)

Map predefined palette on the scalar (indexed) image instance.

Parameters

palette	Multi-valued palette used for mapping the indexes.

Returns

A reference to the modified image instance.

Note

- Function `CImg<T>::get_map()` is also defined. It returns a non-shared modified copy of the image instance.

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();
```

8.1.4.234 CImg<T>& label (const bool *is_high_connectivity* = `false`, const Tfloat *tolerance* = `0`)

Label connected components.

Parameters

is_high_-connectivity	Boolean that choose between 4(false)- or 8(true)-connectivity in 2d case, and between 6(false)- or 26(true)-connectivity in 3d case.

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.

8.1.4.235 CImg<T>& RGBtoHSI ( )

Convert color pixels from RGB to HSI. Reference: "Digital Image Processing, 2nd. edition", R. Gonzalez and R. Woods. Prentice Hall, 2002.

8.1.4.236 CImg<T>& RGBtoBayer ( )

Convert a RGB image to a Bayer-coded representation.

Note

First (upper-left) pixel if the red component of the pixel color.

8.1.4.237 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 *border_conditions* = `0`, const float *centering_x* = `0`, const float *centering_y* = `0`, const float *centering_z* = `0`, const float *centering_c* = `0`)

Resize an image.

Parameters

size_x	Number of columns (new size along the X-axis).
size_y	Number of rows (new size along the Y-axis).
size_z	Number of slices (new size along the Z-axis).
size_c	Number of vector-channels (new size along the C-axis).
interpolation-_type	Method of interpolation : • -1 = no interpolation : raw memory resizing. • 0 = no interpolation : additional space is filled according to `border_condition`. • 1 = nearest-neighbor interpolation. • 2 = moving average interpolation. • 3 = linear interpolation. • 4 = grid interpolation. • 5 = bicubic interpolation. • 6 = lanczos interpolation.

border_-conditions	Border condition type.
centering_x	Set centering type (only if `interpolation_type=0`).
centering_y	Set centering type (only if `interpolation_type=0`).
centering_z	Set centering type (only if `interpolation_type=0`).
centering_c	Set centering type (only if `interpolation_type=0`).

Note

If pd[x,y,z,v]<0, it corresponds to a percentage of the original size (the default value is -100).

8.1.4.238 CImg<T>& resize_doubleXY ( )

Upscale an image by a factor 2x.

Use anisotropic upscaling algorithm described at `http://scale2x.sourceforge.net/algorithm.html`

8.1.4.239 CImg<T>& resize_tripleXY ( )

Upscale an image by a factor 3x.

Use anisotropic upscaling algorithm described at `http://scale2x.sourceforge.net/algorithm.html`

8.1.4.240 CImg<T>& shift (const int *deltax*, const int *deltay* = 0, const int *deltaz* = 0, const int *deltac* = 0, const int *border_condition* = 0)

Shift the image.

Parameters

deltax	Amount of displacement along the X-axis.
deltay	Amount of displacement along the Y-axis.
deltaz	Amount of displacement along the Z-axis.
deltac	Amount of displacement along the C-axis.
border_-condition	Border condition.

- `border_condition` can be :
  - 0 : Zero border condition (Dirichlet).
  - 1 : Nearest neighbors (Neumann).
  - 2 : Repeat Pattern (Fourier style).

8.1.4.241 CImg<T>& permute_axes (const char *const *order*)

Permute axes order.

This function permutes image axes.

Parameters

permut	= String describing the permutation (4 characters).

8.1.4.242 CImg<T>& rotate (const float *angle*, const unsigned int *border_conditions* = 0, const unsigned int *interpolation* = 1)

Rotate an image.

Parameters

angle	= rotation angle (in degrees).
cond	= rotation type. can be : • 0 = zero-value at borders • 1 = nearest pixel. • 2 = cyclic.

Note

Returned image will probably have a different size than the image instance *this.

8.1.4.243 CImg<T>& rotate (const float *angle*, const float *cx*, const float *cy*, const float *zoom*, const unsigned int *border_conditions* = 3, const unsigned int *interpolation* = 1)

Rotate an image around a center point (`cx`,`cy`).

Parameters

angle	= rotation angle (in degrees).
cx	= X-coordinate of the rotation center.
cy	= Y-coordinate of the rotation center.
zoom	= zoom.
cond	= rotation type. can be : • 0 = zero-value at borders • 1 = repeat image at borders • 2 = zero-value at borders and linear interpolation

8.1.4.244 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 bool *border_condition* = `false`)

Return a square region of the image.

Parameters

x0	= X-coordinate of the upper-left crop rectangle corner.
y0	= Y-coordinate of the upper-left crop rectangle corner.
z0	= Z-coordinate of the upper-left crop rectangle corner.
c0	= C-coordinate of the upper-left crop rectangle corner.
x1	= X-coordinate of the lower-right crop rectangle corner.
y1	= Y-coordinate of the lower-right crop rectangle corner.
z1	= Z-coordinate of the lower-right crop rectangle corner.
c1	= C-coordinate of the lower-right crop rectangle corner.
border_-condition	= Dirichlet (false) or Neumann border conditions.

8.1.4.245 CImg<T>& crop (const int *x0*, const int *y0*, const int *z0*, const int *x1*, const int *y1*, const int *z1*, const bool *border_condition* = `false`)

Return a rectangular part of the image instance.

Parameters

x0	= X-coordinate of the upper-left crop rectangle corner.
y0	= Y-coordinate of the upper-left crop rectangle corner.
z0	= Z-coordinate of the upper-left crop rectangle corner.
x1	= X-coordinate of the lower-right crop rectangle corner.
y1	= Y-coordinate of the lower-right crop rectangle corner.
z1	= Z-coordinate of the lower-right crop rectangle corner.
border_-condition	= determine the type of border condition if some of the desired region is outside the image.

8.1.4.246 CImg<T>& crop (const int *x0*, const int *y0*, const int *x1*, const int *y1*, const bool *border_condition* = `false`)

Return a rectangular part of the image instance.

Parameters

x0	= X-coordinate of the upper-left crop rectangle corner.
y0	= Y-coordinate of the upper-left crop rectangle corner.
x1	= X-coordinate of the lower-right crop rectangle corner.
y1	= Y-coordinate of the lower-right crop rectangle corner.
border_-condition	= determine the type of border condition if some of the desired region is outside the image.

8.1.4.247 CImg<T>& crop (const int *x0*, const int *x1*, const bool *border_condition* = `false`)

Return a rectangular part of the image instance.

Parameters

x0	= X-coordinate of the upper-left crop rectangle corner.
x1	= X-coordinate of the lower-right crop rectangle corner.
border_-condition	= determine the type of border condition if some of the desired region is outside the image.

8.1.4.248 CImgList<T> get_split (const CImg< t > & *values*, const bool *keep_values*, const bool *is_shared*) const

Split image into a list of one-column vectors, according to specified sequence of splitting values.

Parameters

values	The splitting pattern of values.
keep_values	Can be : • false : Discard splitting values in resulting list. • true : Keep splitting values as separate images in resulting list.

8.1.4.249 CImg<T>& correlate (const CImg< t > & *mask*, const unsigned int *border_conditions* = `1`, const bool *is_normalized* = `false`)

Compute the correlation of the image instance by a mask.

The correlation of the image instance `*this` by the mask `mask` is defined to be :

res(x,y,z) = sum_{i,j,k} (*this)(x+i,y+j,z+k)*mask(i,j,k)

Parameters

mask	= the correlation kernel.
border_-conditions	= the border condition type (0=zero, 1=dirichlet)
is_-normalized	= enable local normalization.

8.1.4.250 CImg<T>& convolve (const CImg< t > & *mask*, const unsigned int *border_conditions* = `1`, const bool *is_normalized* = `false`)

Compute the convolution of the image by a mask.

The result `res` of the convolution of an image `img` by a mask `mask` is defined to be :

res(x,y,z) = sum_{i,j,k} img(x-i,y-j,z-k)∗mask(i,j,k)

Parameters

mask	= the correlation kernel.
border_-conditions	= the border condition type (0=zero, 1=dirichlet)
is_-normalized	= enable local normalization.

8.1.4.251 CImg<T>& deriche (const float *sigma*, const int *order* = `0`, const char *axis* = `'x'`, const bool *cond* = `true`)

Compute the result of the Deriche filter.

The Canny-Deriche filter is a recursive algorithm allowing to compute blurred derivatives of order 0,1 or 2 of an image.

8.1.4.252 CImg<T>& blur (const float *sigmax*, const float *sigmay*, const float *sigmaz*, const bool *cond* = `true`)

Return a blurred version of the image, using a Canny-Deriche filter.

Blur the image with an anisotropic exponential filter (Deriche filter of order 0).

8.1.4.253 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 *fast_approx* = `1`)

Blur the image anisotropically following a field of diffusion tensors.

Parameters

G	= Field of square roots of diffusion tensors/vectors used to drive the smoothing.
amplitude	= amplitude of the smoothing.
dl	= spatial discretization.
da	= angular discretization.
gauss_prec	= precision of the gaussian function.
interpolation	Used interpolation scheme (0 = nearest-neighbor, 1 = linear, 2 = Runge-Kutta)
fast_approx	= Tell to use the fast approximation or not.

8.1.4.254 CImg<T>& blur_bilateral (const float *sigma_x*, const float *sigma_y*, const float *sigma_z*, const float *sigma_r*, const int *bgrid_x*, const int *bgrid_y*, const int *bgrid_z*, const int *bgrid_r*, const bool *interpolation_type* = `true`)

Blur an image using the bilateral filter.

Parameters

sigma_x	Amount of blur along the X-axis.
sigma_y	Amount of blur along the Y-axis.
sigma_z	Amount of blur along the Z-axis.
sigma_r	Amount of blur along the range axis.
bgrid_x	Size of the bilateral grid along the X-axis.
bgrid_y	Size of the bilateral grid along the Y-axis.
bgrid_z	Size of the bilateral grid along the Z-axis.
bgrid_r	Size of the bilateral grid along the range axis.
interpolation-_type	Use interpolation for image slicing.

Note

This algorithm uses the optimisation technique proposed by S. Paris and F. Durand, in ECCV'2006 (extended for 3d volumetric images).

8.1.4.255 CImgList<Tfloat> get_gradient (const char *const *axes* = `0`, const int *scheme* = `3`) const

Compute the list of images, corresponding to the XY-gradients of an image.

Parameters

scheme	= Numerical scheme used for the gradient computation : • -1 = Backward finite differences • 0 = Centered finite differences • 1 = Forward finite differences • 2 = Using Sobel masks • 3 = Using rotation invariant masks • 4 = Using Deriche recusrsive filter.

8.1.4.256 CImg<T>& displacement (const CImg< T > & *source*, const float *smoothness* = `0.1f`, const float *precision* = `5.0f`, const unsigned int *nb_scales* = `0`, const unsigned int *iteration_max* = `10000`, const bool *is_backward* = `false`)

Estimate a displacement field between specified source image and image instance.

Parameters

is_backward	: if false, match I2(X+U(X)) = I1(X), else match I2(X) = I1(X-U(X)).

8.1.4.257 CImg<T>& distance (const T *value*, const unsigned int *metric* = `2`)

Compute the distance transform according to a specified value.

The distance transform implementation has been 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 has then been modified to fit CImg coding style and constraints.

8.1.4.258 CImg<T>& distance (const T *value*, const CImg< t > & *metric_mask*)

Compute the chamfer distance transform according to a specified value, with a custom metric.

The algorithm code has been initially proposed by A. Meijster, and modified by D. -Tschumperlé.

8.1.4.259 CImg<T>& haar (const char *axis*, const bool *invert* = `false`, const unsigned int *nb_scales* = `1`)

Compute the Haar multiscale wavelet transform (monodimensional version).

Parameters

axis	Axis considered for the transform.
invert	Set inverse of direct transform.
nb_scales	Number of scales used for the transform.

8.1.4.260 CImg<T>& haar (const bool *invert* = `false`, const unsigned int *nb_scales* = `1`)

Compute the Haar multiscale wavelet transform.

Parameters

invert	Set inverse of direct transform.
nb_scales	Number of scales used for the transform.

8.1.4.261 CImg<floatT> get_elevation3d (CImgList< tf > & *primitives,* CImgList< tc > & *colors,* const CImg< te > & *elevation*) const

Create and return a 3d elevation of the image instance.

Parameters

out	*primitives*	The returned list of the 3d object primitives (template type *tf* should be at least *unsigned int*).
out	*colors*	The returned list of the 3d object colors.
	elevation	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,colors,img.
get_norm()*0.2);
 CImg<unsigned char>().display_object3d("Elevation3d",points3d,faces3d,
colors3d);
```

8.1.4.262 CImg<floatT> get_isoline3d (CImgList< tf > & *primitives,* const float *isovalue,* const int *size_x* = `-100`, const int *size_y* = `-100`) const

Create and return a isoline of the image instance as a 3d object.

Parameters

out	*primitives*	The returned list of the 3d object primitives (template type *tf* should be at least *unsigned int*).
	isovalue	The returned list of the 3d object colors.
	size_x	The number of subdivisions along the X-axis.
	size_y	The number of subdisivions 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

```
 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);
```

8.1.4.263 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

Create and return a isosurface of the image instance as a 3d object.

Parameters

`out`	*primitives*	The returned list of the 3d object primitives (template type *tf* should be at least *unsigned int*).
	isovalue	The returned list of the 3d object colors.
	size_x	The number of subdivisions along the X-axis.
	size_y	The number of subdisivions 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

```
 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);
```

8.1.4.264 static CImg<floatT> box3d (CImgList< tf > & *primitives*, const float *size_x* = `200`, const float *size_y* = `100`, const float *size_z* = `100`) `[static]`

Create and return a 3d box object.

Parameters

`out`	*primitives*	The returned list of the 3d object primitives (template type *tf* should be at least *unsigned int*).
	size_x	The width of the box (dimension along the X-axis).
	size_y	The height of the box (dimension along the Y-axis).
	size_z	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);
```

8.1.4.265 static CImg<floatT> cone3d (CImgList< tf > & *primitives*, const float *radius* = 50, const float *size_z* = 100, const unsigned int *subdivisions* = 24) [static]

Create and return a 3d cone.

Parameters

out	*primitives*	The returned list of the 3d object primitives (template type *tf* should be at least *unsigned int*).
	radius	The radius of the cone basis.
	size_z	The cone's height.
	subdivisions	The number of basis angular subdivisions.

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);
```

8.1.4.266 static CImg<floatT> cylinder3d (CImgList< tf > & *primitives*, const float *radius* = 50, const float *size_z* = 100, const unsigned int *subdivisions* = 24) [static]

Create and return a 3d cylinder.

Parameters

out	*primitives*	The returned list of the 3d object primitives (template type *tf* should be at least *unsigned int*).
	radius	The radius of the cylinder basis.
	size_z	The cylinder's height.
	subdivisions	The number of basis angular subdivisions.

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);
```

8.1.4.267 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]

Create and return a 3d torus.

Parameters

out	*primitives*	The returned list of the 3d object primitives (template type *tf* should be at least *unsigned int*).
	radius1	The large radius.
	radius2	The small radius.
	subdivisions1	The number of angular subdivisions for the large radius.
	subdivisions2	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<=i<=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);
```

8.1.4.268 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]

Create and return a 3d XY-plane.

Parameters

out	*primitives*	The returned list of the 3d object primitives (template type *tf* should be at least *unsigned int*).
	size_x	The width of the plane (dimension along the X-axis).

	size_y	The height of the plane (dimensions along the Y-axis).
	subdivisions-_x	The number of planar subdivisions along the X-axis.
	subdivisions-_y	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);
```

8.1.4.269 static CImg<floatT> sphere3d (CImgList< tf > & *primitives*, const float *radius* = 50, const unsigned int *subdivisions* = 3) [static]

Create and return a 3d sphere.

Parameters

out	*primitives*	The returned list of the 3d object primitives (template type *tf* should be at least *unsigned int*).
	radius	The radius of the sphere (dimension along the X-axis).
	subdivisions	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);
```

8.1.4.270 static CImg<floatT> ellipsoid3d (CImgList< tf > & *primitives*, const CImg< t > & *tensor*, const unsigned int *subdivisions* = 3) [static]

Create and return a 3d ellipsoid.

Parameters

out	*primitives*	The returned list of the 3d object primitives (template type *tf* should be at least *unsigned int*).
	tensor	The tensor which gives the shape and size of the ellipsoid.
	subdivisions	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);
```

8.1.4.271 CImg<T>& draw_point (const int *x0*, const int *y0*, const tc *const *color*, const float *opacity* = 1)

Draw a 2d colored point (pixel).

Parameters

x0	X-coordinate of the point.
y0	Y-coordinate of the point.
color	Pointer to `spectrum()` consecutive values, defining the color values.
opacity	Drawing opacity (optional).

Note

- Clipping is supported.
- 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);
```

8.1.4.272 CImg<T>& draw_line (const int *x0*, const int *y0*, const int *x1*, const int *y1*, const tc *const *color*, const float *opacity* = 1, const unsigned int *pattern* = ~0U, const bool *init_hatch* = true)

Draw a 2d colored line.

Parameters

x0	X-coordinate of the starting line point.
y0	Y-coordinate of the starting line point.
x1	X-coordinate of the ending line point.
y1	Y-coordinate of the ending line point.
color	Pointer to `spectrum()` consecutive values of type `T`, defining the drawing color.
opacity	Drawing opacity (optional).
pattern	An integer whose bits describe the line pattern (optional).
init_hatch	Flag telling if a reinitialization of the hash state must be done (optional).

Note

- Clipping is supported.
- Line routine uses Bresenham's algorithm.
- Set `init_hatch` = false to draw consecutive hatched segments without breaking the line 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);
```

8.1.4.273 CImg<T>& draw_line (const int *x0*, const int *y0*, const int *x1*, const int *y1*, 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 2d textured line.

Parameters

x0	X-coordinate of the starting line point.
y0	Y-coordinate of the starting line point.
x1	X-coordinate of the ending line point.
y1	Y-coordinate of the ending line point.
texture	Texture image defining the pixel colors.
tx0	X-coordinate of the starting texture point.
ty0	Y-coordinate of the starting texture point.
tx1	X-coordinate of the ending texture point.
ty1	Y-coordinate of the ending texture point.
opacity	Drawing opacity (optional).
pattern	An integer whose bits describe the line pattern (optional).
init_hatch	Flag telling if the hash variable must be reinitialized (optional).

Note

- Clipping is supported but not for texture coordinates.
- 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);
```

8.1.4.274 CImg<T>& draw_line (const CImg< t > & *points*, const tc ∗const *color*, const float *opacity* = `1`, const unsigned int *pattern* = `~0U`, const bool *init_hatch* = `true`)

Draw a set of consecutive colored lines in the image instance.

Parameters

points	Coordinates of vertices, stored as a list of vectors.
color	Pointer to `spectrum()` consecutive values of type `T`, defining the drawing color.
opacity	Drawing opacity (optional).
pattern	An integer whose bits describe the line pattern (optional).
init_hatch	If set to true, init hatch motif.

Note

- This function uses several call to the single CImg::draw_line() procedure, depending on the vectors size in `points`.

Example:

```
CImg<unsigned char> img(100,100,1,3,0);
const unsigned char color[] = { 255,128,64 };
CImgList<int> points;
points.insert(CImg<int>::vector(0,0)).
      .insert(CImg<int>::vector(70,10)).
      .insert(CImg<int>::vector(80,60)).
      .insert(CImg<int>::vector(10,90));
img.draw_line(points,color);
```

8.1.4.275 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 colored arrow in the image instance.

Parameters

x0	X-coordinate of the starting arrow point (tail).
y0	Y-coordinate of the starting arrow point (tail).
x1	X-coordinate of the ending arrow point (head).
y1	Y-coordinate of the ending arrow point (head).
color	Pointer to `spectrum()` consecutive values of type `T`, defining the drawing color.
angle	Aperture angle of the arrow head (optional).
length	Length of the arrow head. If negative, describes a percentage of the arrow length (optional).
opacity	Drawing opacity (optional).
pattern	An integer whose bits describe the line pattern (optional).

Note

- Clipping is supported.

8.1.4.276 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* = `0.25`, const unsigned int *pattern* = `~0U`, const bool *init_hatch* = `true`)

Draw a cubic spline curve in the image instance.

Parameters

x0	X-coordinate of the starting curve point
y0	Y-coordinate of the starting curve point
u0	X-coordinate of the starting velocity
v0	Y-coordinate of the starting velocity
x1	X-coordinate of the ending curve point
y1	Y-coordinate of the ending curve point
u1	X-coordinate of the ending velocity
v1	Y-coordinate of the ending velocity
color	Pointer to `spectrum()` consecutive values of type `T`, defining the drawing color.
precision	Curve drawing precision (optional).
opacity	Drawing opacity (optional).
pattern	An integer whose bits describe the line pattern (optional).
init_hatch	If `true`, init hatch motif.

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 serie 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);
```

8.1.4.277 CImg<T>& draw_spline (const int *x0*, const int *y0*, const int *z0*, const float *u0*, const float *v0*, const float *w0*, const int *x1*, const int *y1*, const int *z1*, const float *u1*, const float *v1*, const float *w1*, const tc *const *color*, const float *opacity* = 1, const float *precision* = 4, const unsigned int *pattern* = ~0U, const bool *init_hatch* = true)

Draw a cubic spline curve in the image instance (for volumetric images).

Note

- Similar to CImg::draw_spline() for a 3d spline in a volumetric image.

8.1.4.278 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 cubic spline curve in the image instance.

Parameters

x0	X-coordinate of the starting curve point
y0	Y-coordinate of the starting curve point
u0	X-coordinate of the starting velocity
v0	Y-coordinate of the starting velocity
x1	X-coordinate of the ending curve point
y1	Y-coordinate of the ending curve point
u1	X-coordinate of the ending velocity
v1	Y-coordinate of the ending velocity
texture	Texture image defining line pixel colors.
tx0	X-coordinate of the starting texture point.
ty0	Y-coordinate of the starting texture point.
tx1	X-coordinate of the ending texture point.
ty1	Y-coordinate of the ending texture point.

precision	Curve drawing precision (optional).
opacity	Drawing opacity (optional).
pattern	An integer whose bits describe the line pattern (optional).
init_hatch	if `true`, reinit hatch motif.

8.1.4.279 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 *brightness0*, const float *brightness1*, const float *brightness2*, const float *opacity* = `1`)

Draw a 2d Gouraud-shaded colored triangle.

Parameters

x0	= X-coordinate of the first corner in the image instance.
y0	= Y-coordinate of the first corner in the image instance.
x1	= X-coordinate of the second corner in the image instance.
y1	= Y-coordinate of the second corner in the image instance.
x2	= X-coordinate of the third corner in the image instance.
y2	= Y-coordinate of the third corner in the image instance.
color	= array of spectrum() values of type `T`, defining the global drawing color.
brightness0	= brightness of the first corner (in [0,2]).
brightness1	= brightness of the second corner (in [0,2]).
brightness2	= brightness of the third corner (in [0,2]).
opacity	= opacity of the drawing.

Note

Clipping is supported.

8.1.4.280 CImg<T>& draw_triangle (const int *x0*, const int *y0*, const int *x1*, const int *y1*, const int *x2*, const int *y2*, const CImg< tc > & *texture*, const int *tx0*, const int *ty0*, const int *tx1*, const int *ty1*, const int *tx2*, const int *ty2*, const float *opacity* = `1`, const float *brightness* = `1`)

Draw a 2d textured triangle.

Parameters

x0	= X-coordinate of the first corner in the image instance.
y0	= Y-coordinate of the first corner in the image instance.
x1	= X-coordinate of the second corner in the image instance.
y1	= Y-coordinate of the second corner in the image instance.
x2	= X-coordinate of the third corner in the image instance.
y2	= Y-coordinate of the third corner in the image instance.
texture	= texture image used to fill the triangle.
tx0	= X-coordinate of the first corner in the texture image.

ty0	= Y-coordinate of the first corner in the texture image.
tx1	= X-coordinate of the second corner in the texture image.
ty1	= Y-coordinate of the second corner in the texture image.
tx2	= X-coordinate of the third corner in the texture image.
ty2	= Y-coordinate of the third corner in the texture image.
opacity	= opacity of the drawing.
brightness	= brightness of the drawing (in [0,2]).

Note

Clipping is supported, but texture coordinates do not support clipping.

8.1.4.281 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 CImg< tl > & *light,* const int *lx0,* const int *ly0,* const int *lx1,* const int *ly1,* const int *lx2,* const int *ly2,* const float *opacity* = `1`)

Draw a 2d Pseudo-Phong-shaded triangle.

Parameters

x0	= X-coordinate of the first corner in the image instance.
y0	= Y-coordinate of the first corner in the image instance.
x1	= X-coordinate of the second corner in the image instance.
y1	= Y-coordinate of the second corner in the image instance.
x2	= X-coordinate of the third corner in the image instance.
y2	= Y-coordinate of the third corner in the image instance.
color	= array of spectrum() values of type `T`, defining the global drawing color.
light	= light image.
lx0	= X-coordinate of the first corner in the light image.
ly0	= Y-coordinate of the first corner in the light image.
lx1	= X-coordinate of the second corner in the light image.
ly1	= Y-coordinate of the second corner in the light image.
lx2	= X-coordinate of the third corner in the light image.
ly2	= Y-coordinate of the third corner in the light image.
opacity	= opacity of the drawing.

Note

Clipping is supported, but texture coordinates do not support clipping.

8.1.4.282 CImg<T>& draw_triangle (const int *x0*, const int *y0*, const int *x1*, const int *y1*, const int *x2*, const int *y2*, const CImg< tc > & *texture*, const int *tx0*, const int *ty0*, const int *tx1*, const int *ty1*, const int *tx2*, const int *ty2*, const float *brightness0*, const float *brightness1*, const float *brightness2*, const float *opacity* = 1)

Draw a 2d Gouraud-shaded textured triangle.

Parameters

x0	= X-coordinate of the first corner in the image instance.
y0	= Y-coordinate of the first corner in the image instance.
x1	= X-coordinate of the second corner in the image instance.
y1	= Y-coordinate of the second corner in the image instance.
x2	= X-coordinate of the third corner in the image instance.
y2	= Y-coordinate of the third corner in the image instance.
texture	= texture image used to fill the triangle.
tx0	= X-coordinate of the first corner in the texture image.
ty0	= Y-coordinate of the first corner in the texture image.
tx1	= X-coordinate of the second corner in the texture image.
ty1	= Y-coordinate of the second corner in the texture image.
tx2	= X-coordinate of the third corner in the texture image.
ty2	= Y-coordinate of the third corner in the texture image.
brightness0	= brightness value of the first corner.
brightness1	= brightness value of the second corner.
brightness2	= brightness value of the third corner.
opacity	= opacity of the drawing.

Note

Clipping is supported, but texture coordinates do not support clipping.

8.1.4.283 CImg<T>& draw_triangle (const int *x0*, const int *y0*, const int *x1*, const int *y1*, const int *x2*, const int *y2*, const CImg< tc > & *texture*, const int *tx0*, const int *ty0*, const int *tx1*, const int *ty1*, const int *tx2*, const int *ty2*, const CImg< tl > & *light*, const int *lx0*, const int *ly0*, const int *lx1*, const int *ly1*, const int *lx2*, const int *ly2*, const float *opacity* = 1)

Draw a 2d Pseudo-Phong-shaded textured triangle.

Parameters

x0	= X-coordinate of the first corner in the image instance.
y0	= Y-coordinate of the first corner in the image instance.
x1	= X-coordinate of the second corner in the image instance.

y1	= Y-coordinate of the second corner in the image instance.
x2	= X-coordinate of the third corner in the image instance.
y2	= Y-coordinate of the third corner in the image instance.
texture	= texture image used to fill the triangle.
tx0	= X-coordinate of the first corner in the texture image.
ty0	= Y-coordinate of the first corner in the texture image.
tx1	= X-coordinate of the second corner in the texture image.
ty1	= Y-coordinate of the second corner in the texture image.
tx2	= X-coordinate of the third corner in the texture image.
ty2	= Y-coordinate of the third corner in the texture image.
light	= light image.
lx0	= X-coordinate of the first corner in the light image.
ly0	= Y-coordinate of the first corner in the light image.
lx1	= X-coordinate of the second corner in the light image.
ly1	= Y-coordinate of the second corner in the light image.
lx2	= X-coordinate of the third corner in the light image.
ly2	= Y-coordinate of the third corner in the light image.
opacity	= opacity of the drawing.

Note

Clipping is supported, but texture coordinates do not support clipping.

8.1.4.284 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 4d filled rectangle in the image instance, at coordinates (`x0`,`y0`,`z0`,`c0`)-(`x1`,`y1`,`z1`,`c1`).

Parameters

x0	X-coordinate of the upper-left rectangle corner.
y0	Y-coordinate of the upper-left rectangle corner.
z0	Z-coordinate of the upper-left rectangle corner.
c0	C-coordinate of the upper-left rectangle corner.
x1	X-coordinate of the lower-right rectangle corner.
y1	Y-coordinate of the lower-right rectangle corner.
z1	Z-coordinate of the lower-right rectangle corner.
c1	C-coordinate of the lower-right rectangle corner.
val	Scalar value used to fill the rectangle area.
opacity	Drawing opacity (optional).

Note

- Clipping is supported.

8.1.4.285 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 3d filled colored rectangle in the image instance, at coordinates (`x0`,`y0`,`z0`)-(`x1`,`y1`,`z1`).

Parameters

x0	X-coordinate of the upper-left rectangle corner.
y0	Y-coordinate of the upper-left rectangle corner.
z0	Z-coordinate of the upper-left rectangle corner.
x1	X-coordinate of the lower-right rectangle corner.
y1	Y-coordinate of the lower-right rectangle corner.
z1	Z-coordinate of the lower-right rectangle corner.
color	Pointer to `spectrum()` consecutive values of type `T`, defining the drawing color.
opacity	Drawing opacity (optional).

Note

- Clipping is supported.

8.1.4.286 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 2d filled colored rectangle in the image instance, at coordinates (`x0`,`y0`)-(`x1`,`y1`).

Parameters

x0	X-coordinate of the upper-left rectangle corner.
y0	Y-coordinate of the upper-left rectangle corner.
x1	X-coordinate of the lower-right rectangle corner.
y1	Y-coordinate of the lower-right rectangle corner.
color	Pointer to `spectrum()` consecutive values of type `T`, defining the drawing color.
opacity	Drawing opacity (optional).

Note

- Clipping is supported.

8.1.4.287 CImg<T>& draw_circle (const int *x0*, const int *y0*, int *radius*, const tc ∗const *color*, const float *opacity* = `1`)

Draw a filled circle.

Parameters

x0	X-coordinate of the circle center.
y0	Y-coordinate of the circle center.
radius	Circle radius.
color	Array of spectrum() values of type `T`, defining the drawing color.
opacity	Drawing opacity.

Note

- Circle version of the Bresenham's algorithm is used.

8.1.4.288 CImg<T>& draw_circle (const int *x0*, const int *y0*, int *radius*, const tc *const *color*, const float *opacity*, const unsigned *int*)

Draw an outlined circle.

Parameters

x0	X-coordinate of the circle center.
y0	Y-coordinate of the circle center.
radius	Circle radius.
color	Array of spectrum() values of type `T`, defining the drawing color.
opacity	Drawing opacity.

8.1.4.289 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 ellipse.

Parameters

x0	= X-coordinate of the ellipse center.
y0	= Y-coordinate of the ellipse center.
r1	= First radius of the ellipse.
r2	= Second radius of the ellipse.
angle	= Angle of the first radius.
color	= array of spectrum() values of type `T`, defining the drawing color.
opacity	= opacity of the drawing.

8.1.4.290 CImg<T>& draw_ellipse (const int *x0*, const int *y0*, const CImg< t > & *tensor*, const tc *const *color*, const float *opacity* = `1`)

Draw a filled ellipse.

Parameters

x0	= X-coordinate of the ellipse center.
y0	= Y-coordinate of the ellipse center.
tensor	= Diffusion tensor describing the ellipse.
color	= array of spectrum() values of type `T`, defining the drawing color.
opacity	= opacity of the drawing.

8.1.4.291 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 ellipse.

Parameters

x0	= X-coordinate of the ellipse center.
y0	= Y-coordinate of the ellipse center.
r1	= First radius of the ellipse.
r2	= Second radius of the ellipse.
ru	= X-coordinate of the orientation vector related to the first radius.
rv	= Y-coordinate of the orientation vector related to the first radius.
color	= array of spectrum() values of type `T`, defining the drawing color.
pattern	= If zero, the ellipse is filled, else pattern is an integer whose bits describe the outline pattern.
opacity	= opacity of the drawing.

8.1.4.292 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*)

Draw an outlined ellipse.

Parameters

x0	= X-coordinate of the ellipse center.
y0	= Y-coordinate of the ellipse center.
tensor	= Diffusion tensor describing the ellipse.
color	= array of spectrum() values of type `T`, defining the drawing color.
opacity	= opacity of the drawing.

8.1.4.293 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

sprite	Sprite image.
x0	X-coordinate of the sprite position.
y0	Y-coordinate of the sprite position.
z0	Z-coordinate of the sprite position.
c0	C-coordinate of the sprite position.
opacity	Drawing opacity (optional).

Note

- Clipping is supported.

8.1.4.294 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_valmax* = `1`)

Draw a sprite image in the image instance (masked version).

Parameters

sprite	Sprite image.
mask	Mask image.
x0	X-coordinate of the sprite position in the image instance.
y0	Y-coordinate of the sprite position in the image instance.
z0	Z-coordinate of the sprite position in the image instance.
c0	C-coordinate of the sprite position in the image instance.
mask_-valmax	Maximum pixel value of the mask image `mask` (optional).
opacity	Drawing opacity.

Note

- Pixel values of `mask` set the opacity of the corresponding pixels in `sprite`.
- Clipping is supported.
- Dimensions along x,y and z of `sprite` and `mask` must be the same.

8.1.4.295 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 > & *font*, ...)

Draw a text.

Parameters

x0	X-coordinate of the text in the image instance.
y0	Y-coordinate of the text in the image instance.
foreground_-color	Array of spectrum() values of type `T`, defining the foreground color (0 means 'transparent').
background-_color	Array of spectrum() values of type `T`, defining the background color (0 means 'transparent').
font	Font used for drawing text.
opacity	Drawing opacity.
format	'printf'-style format string, followed by arguments.

Note

Clipping is supported.

8.1.4.296 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`, ...)

Draw a text.

Parameters

x0	X-coordinate of the text in the image instance.
y0	Y-coordinate of the text in the image instance.
foreground_-color	Array of spectrum() values of type `T`, defining the foreground color (0 means 'transparent').
background-_color	Array of spectrum() values of type `T`, defining the background color (0 means 'transparent').
font_size	Size of the font (exact match for 13,24,32,57).
opacity	Drawing opacity.
format	'printf'-style format string, followed by arguments.

Note

Clipping is supported.

8.1.4.297 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 *arrows* = `true`, const unsigned int *pattern* = `~0U`)

Draw a vector field in the image instance, using a colormap.

Parameters

flow	Image of 2d vectors used as input data.
color	Image of spectrum()-D vectors corresponding to the color of each arrow.
sampling	Length (in pixels) between each arrow.
factor	Length factor of each arrow (if <0, computed as a percentage of the maximum length).
opacity	Opacity of the drawing.
pattern	Used pattern to draw lines.

Note

Clipping is supported.

8.1.4.298 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 *arrows* = `true`, const unsigned int *pattern* = `~0U`)

Draw a vector field in the image instance, using a colormap.

Parameters

flow	Image of 2d vectors used as input data.
color	Image of spectrum()-D vectors corresponding to the color of each arrow.
sampling	Length (in pixels) between each arrow.
factor	Length factor of each arrow (if <0, computed as a percentage of the maximum length).
opacity	Opacity of the drawing.
pattern	Used pattern to draw lines.

Note

Clipping is supported.

8.1.4.299 CImg<T>& draw_axis (const CImg< t > & *xvalues*, 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`)

Draw a labeled horizontal axis on the image instance.

Parameters

xvalues	Lower bound of the x-range.
y	Y-coordinate of the horizontal axis in the image instance.
color	Array of spectrum() values of type `T`, defining the drawing color.
opacity	Drawing opacity.
pattern	Drawing pattern.
opacity_out	Drawing opacity of 'outside' axes.
allow_zero	Enable/disable the drawing of label '0' if found.

Note

if `precision==0`, precision of the labels is automatically computed.

8.1.4.300 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 on the image instance.

Parameters

data	Image containing the graph values I = f(x).
color	Array of spectrum() values of type `T`, defining the drawing color.
opacity	Drawing opacity.
plot_type	Define the type of the plot : • 0 = No plot. • 1 = Plot using segments. • 2 = Plot using cubic splines. • 3 = Plot with bars.
vertex_type	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.
ymin	Lower bound of the y-range.
ymax	Upper bound of the y-range.
pattern	Drawing pattern.

Note

- if `ymin==ymax==0`, the y-range is computed automatically from the input samples.

8.1.4.301 CImg<T>& draw_fill (const int *x*, const int *y*, const int *z*, const tc *const *color*, const float *opacity*, CImg< t > & *region*, const float *sigma* = `0`, const bool *high_connexity* = `false`)

Draw a 3d filled region starting from a point (`x`,`y`,\ z) in the image instance.

Parameters

x	X-coordinate of the starting point of the region to fill.
y	Y-coordinate of the starting point of the region to fill.
z	Z-coordinate of the starting point of the region to fill.
color	An array of spectrum() values of type `T`, defining the drawing color.
region	Image that will contain the mask of the filled region mask, as an output.
sigma	Tolerance concerning neighborhood values.
opacity	Opacity of the drawing.
high_-connexity	Tells if 8-connexity must be used (only for 2d images).

Returns

`region` is initialized with the binary mask of the filled region.

8.1.4.302 CImg<T>& draw_fill (const int *x*, const int *y*, const int *z*, const tc *const *color*, const float *opacity* = `1`, const float *sigma* = `0`, const bool *high_connexity* = `false`)

Draw a 3d filled region starting from a point (`x`,`y`,\ z) in the image instance.

Parameters

x	= X-coordinate of the starting point of the region to fill.
y	= Y-coordinate of the starting point of the region to fill.
z	= Z-coordinate of the starting point of the region to fill.
color	= an array of spectrum() values of type `T`, defining the drawing color.
sigma	= tolerance concerning neighborhood values.
opacity	= opacity of the drawing.

8.1.4.303 CImg<T>& draw_fill (const int *x*, const int *y*, const tc *const *color*, const float *opacity* = `1`, const float *sigma* = `0`, const bool *high_connexity* = `false`)

Draw a 2d filled region starting from a point (`x`,`y`) in the image instance.

Parameters

x	= X-coordinate of the starting point of the region to fill.
y	= Y-coordinate of the starting point of the region to fill.
color	= an array of spectrum() values of type `T`, defining the drawing color.
sigma	= tolerance concerning neighborhood values.
opacity	= opacity of the drawing.

8.1.4.304 CImg<T>& draw_plasma (const float *alpha* = `1`, const float *beta* = `0`, const unsigned int *scale* = `8`)

Draw a plasma random texture.

Parameters

alpha	Alpha-parameter.
beta	Beta-parameter.
scale	Scale-parameter.

8.1.4.305 CImg<T>& draw_gaussian (const float *xc*, const float *sigma*, const tc ∗const *color*, const float *opacity* = `1`)

Draw a 1d gaussian function in the image instance.

Parameters

xc	= X-coordinate of the gaussian center.
sigma	= Standard variation of the gaussian distribution.
color	= array of spectrum() values of type `T`, defining the drawing color.
opacity	= opacity of the drawing.

8.1.4.306 CImg<T>& draw_gaussian (const float *xc*, const float *yc*, const CImg< t > & *tensor*, const tc ∗const *color*, const float *opacity* = `1`)

Draw an anisotropic 2d gaussian function.

Parameters

xc	= X-coordinate of the gaussian center.
yc	= Y-coordinate of the gaussian center.
tensor	= 2x2 covariance matrix.
color	= array of spectrum() values of type `T`, defining the drawing color.
opacity	= opacity of the drawing.

8.1.4.307 CImg<T>& draw_gaussian (const float *xc*, const float *yc*, const float *sigma*, const tc ∗const *color*, const float *opacity* = `1`)

Draw an isotropic 2d gaussian function.

Parameters

xc	= X-coordinate of the gaussian center.
yc	= Y-coordinate of the gaussian center.
sigma	= standard variation of the gaussian distribution.
color	= array of spectrum() values of type `T`, defining the drawing color.
opacity	= opacity of the drawing.

8.1.4.308 CImg<T>& draw_gaussian (const float *xc*, const float *yc*, const float *zc*, const CImg< t > & *tensor*, const tc ∗const *color*, const float *opacity* = `1`)

Draw an anisotropic 3d gaussian function.

Parameters

xc	= X-coordinate of the gaussian center.
yc	= Y-coordinate of the gaussian center.
zc	= Z-coordinate of the gaussian center.
tensor	= 3x3 covariance matrix.
color	= array of spectrum() values of type `T`, defining the drawing color.
opacity	= opacity of the drawing.

8.1.4.309 CImg<T>& draw_gaussian (const float *xc*, const float *yc*, const float *zc*, const float *sigma*, const tc ∗const *color*, const float *opacity* = `1`)

Draw an isotropic 3d gaussian function.

Parameters

xc	= X-coordinate of the gaussian center.
yc	= Y-coordinate of the gaussian center.
zc	= Z-coordinate of the gaussian center.
sigma	= standard variation of the gaussian distribution.
color	= array of spectrum() values of type `T`, defining the drawing color.
opacity	= opacity of the drawing.

8.1.4.310 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 *double_sided* = `false`, const float *focale* = `500`, const float *lightx* = `0`, const float *lighty* = `0`, const float *lightz* = `-5e8`, const float *specular_light* = `0.2f`, const float *specular_shine* = `0.1f`)

Draw a 3d object.

Parameters

X	= X-coordinate of the 3d object position
Y	= Y-coordinate of the 3d object position
Z	= Z-coordinate of the 3d object position
vertices	= Image Nx3 describing 3d point coordinates
primitives	= List of P primitives
colors	= List of P color (or textures)
opacities	= Image or list of P opacities
render_type	= Render type (0=Points, 1=Lines, 2=Faces (no light), 3=Faces (flat), 4=Faces(Gouraud)
double_-sided	= Tell if object faces have two sides or are oriented.
focale	= length of the focale (0 for parallel projection)
lightx	= X-coordinate of the light
lighty	= Y-coordinate of the light
lightz	= Z-coordinate of the light
specular_-shine	= Shininess of the object

8.1.4.311 CImg<T>& load (const char ∗const *filename*)

Load an image from a file.

Parameters

filename	is the name of the image file to load.

Note

The extension of `filename` defines the file format. If no filename extension is provided, CImg<T>::get_load() will try to load a .cimg file.

8.1.4.312 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`)

Load an image from a TIFF file.

- libtiff support is enabled by defining the precompilation directive cimg_use_tif.
- When libtiff is enabled, 2D and 3D (multipage) several channel per pixel are supported for char,uchar,short,ushort,float and double pixel type.
- If cimg_use_tif is not defined at compilation time the function uses CImg<T>&load_other(const char∗).

See also

CImg<T>& load_other(const char∗)
CImg<T>& save_tiff(const char∗, const unsigned int)

8.1.4.313 const CImg<T>& print (const char ∗const *title* = `0`, const bool *display_stats* = `true`) const

Display informations about the image on the standard error output.

Parameters

title	Name for the considered image (optional).
display_-stats	Compute and display image statistics (optional).

8.1.4.314 const CImg<T>& save (const char ∗const *filename*, const int *number* = `-1`) const

Save the image as a file.

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.

8.1.4.315 const CImg<T>& save_tiff (const char ∗const *filename*, const unsigned int *compression* = `0`) const

Save a file in TIFF format.

- libtiff support is enabled by defining the precompilation directive cimg_use_tif.
- When libtiff is enabled, 2D and 3D (multipage) several channel per pixel are supported for char,uchar,short,ushort,float and double pixel type.
- If cimg_use_tif is not defined at compilation time the function uses CImg<T>&save_other(const char∗).

Parameters

compression	1:None, 2:CCITTRLE, 3:CCITTFAX3, 4:CCITTFAX4, 5:LZW, 6:JPEG

See also

CImg<T>& save_other(const char∗)
CImg<T>& load_tiff(const char∗)

8.1.4.316 const CImg<T>& save_graphicsmagick_external (const char ∗const *filename*, const unsigned int *quality* = 100) const

Save the image using GraphicsMagick's gm.

Function that saves the image for other file formats that are not natively handled by CImg, using the tool 'gm' from the GraphicsMagick package.

This is the case for all compressed image formats (GIF,PNG,JPG,TIF, ...). You need to install the GraphicsMagick package in order to get this function working properly (see `http://www.graphicsmagick.org`).

8.1.4.317 const CImg<T>& save_imagemagick_external (const char ∗const *filename*, const unsigned int *quality* = 100) const

Save the image using ImageMagick's convert.

Function that saves the image for other file formats that are not natively handled by CImg, using the tool 'convert' from the ImageMagick package.

This is the case for all compressed image formats (GIF,PNG,JPG,TIF, ...). You need to install the ImageMagick package in order to get this function working properly (see `http://www.imagemagick.org`).

8.2 CImgDisplay Struct Reference

Allow to create windows, display images on them and manage user events (keyboard, mouse and windows events).

Constructors / Destructor / Instance Management

- ∼CImgDisplay ()

  Destructor.
- CImgDisplay ()

  Create an empty display.
- 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)

Create a display with specified dimensions.

- 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)

  Create a display from an image.
- 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)

  Create a display from an image list.
- CImgDisplay (const CImgDisplay &disp)

  Create a display as a copy of an existing one.
- CImgDisplay & assign ()

  *Destructor - Empty constructor **[in-place version]**.*
- CImgDisplay & 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)

  *Create a display with specified dimensions **[in-place version]**.*
- template<typename T >
  CImgDisplay & assign (const CImg< T > &img, const char *const title=0, const unsigned int normalization=3, const bool is_fullscreen=false, const bool is_closed=false)

  *Create a display from an image **[in-place version]**.*
- template<typename T >
  CImgDisplay & assign (const CImgList< T > &list, const char *const title=0, const unsigned int normalization=3, const bool is_fullscreen=false, const bool is_closed=false)

  *Create a display from an image list **[in-place version]**.*
- CImgDisplay & assign (const CImgDisplay &disp)

  *Create a display as a copy of another one **[in-place version]**.*
- static CImgDisplay & empty ()

  Return a reference to an empty display.

Overloaded Operators

- template<typename t >
  CImgDisplay & operator= (const CImg< t > &img)

  Display image on associated window.
- template<typename t >
  CImgDisplay & operator= (const CImgList< t > &list)

  Display list of images on associated window.
- CImgDisplay & operator= (const CImgDisplay &disp)

  *Create 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_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 occured 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) const

  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.*
- bool **is_keyF1** () const
- bool **is_keyF2** () const
- bool **is_keyF3** () const
- bool **is_keyF4** () const
- bool **is_keyF5** () const
- bool **is_keyF6** () const
- bool **is_keyF7** () const
- bool **is_keyF8** () const
- bool **is_keyF9** () const
- bool **is_keyF10** () const
- bool **is_keyF11** () const
- bool **is_keyF12** () const
- bool **is_keyPAUSE** () const
- bool **is_key1** () const
- bool **is_key2** () const
- bool **is_key3** () const

- bool **is_key4** () const
- bool **is_key5** () const
- bool **is_key6** () const
- bool **is_key7** () const
- bool **is_key8** () const
- bool **is_key9** () const
- bool **is_key0** () const
- bool **is_keyBACKSPACE** () const
- bool **is_keyINSERT** () const
- bool **is_keyHOME** () const
- bool **is_keyPAGEUP** () const
- bool **is_keyTAB** () const
- bool **is_keyQ** () const
- bool **is_keyW** () const
- bool **is_keyE** () const
- bool **is_keyR** () const
- bool **is_keyT** () const
- bool **is_keyY** () const
- bool **is_keyU** () const
- bool **is_keyI** () const
- bool **is_keyO** () const
- bool **is_keyP** () const
- bool **is_keyDELETE** () const
- bool **is_keyEND** () const
- bool **is_keyPAGEDOWN** () const
- bool **is_keyCAPSLOCK** () const
- bool **is_keyA** () const
- bool **is_keyS** () const
- bool **is_keyD** () const
- bool **is_keyF** () const
- bool **is_keyG** () const
- bool **is_keyH** () const
- bool **is_keyJ** () const
- bool **is_keyK** () const
- bool **is_keyL** () const
- bool **is_keyENTER** () const
- bool **is_keySHIFTLEFT** () const
- bool **is_keyZ** () const
- bool **is_keyX** () const
- bool **is_keyC** () const
- bool **is_keyV** () const
- bool **is_keyB** () const
- bool **is_keyN** () const
- bool **is_keyM** () const
- bool **is_keySHIFTRIGHT** () const
- bool **is_keyARROWUP** () const

- bool **is_keyCTRLLEFT** () const
- bool **is_keyAPPLEFT** () const
- bool **is_keyALT** () const
- bool **is_keySPACE** () const
- bool **is_keyALTGR** () const
- bool **is_keyAPPRIGHT** () const
- bool **is_keyMENU** () const
- bool **is_keyCTRLRIGHT** () const
- bool **is_keyARROWLEFT** () const
- bool **is_keyARROWDOWN** () const
- bool **is_keyARROWRIGHT** () const
- bool **is_keyPAD0** () const
- bool **is_keyPAD1** () const
- bool **is_keyPAD2** () const
- bool **is_keyPAD3** () const
- bool **is_keyPAD4** () const
- bool **is_keyPAD5** () const
- bool **is_keyPAD6** () const
- bool **is_keyPAD7** () const
- bool **is_keyPAD8** () const
- bool **is_keyPAD9** () const
- bool **is_keyPADADD** () const
- bool **is_keyPADSUB** () const
- bool **is_keyPADMUL** () const
- bool **is_keyPADDIV** () const

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.
- unsigned int key (const unsigned int pos=0) const

  Return one entry from the pressed keys history.
- 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 >
  CImgDisplay & display (const CImg< T > &img)

  Display image on associated window.
- template<typename T >
  CImgDisplay & display (const CImgList< T > &list, const char axis='x', const float align=0)

  Display list of images on associated window.
- CImgDisplay & show ()

  Show (closed) associated window on the screen.
- CImgDisplay & close ()

  Close (visible) associated window and make it disappear from the screen.
- CImgDisplay & move (const int pos_x, const int pos_y)

  Move associated window to a new location.
- CImgDisplay & resize (const bool force_redraw=true)

  Resize display to the size of the associated window.
- CImgDisplay & resize (const int width, const int height, const bool force_redraw=true)

  Resize display to the specified size.
- template<typename T >
  CImgDisplay & resize (const CImg< T > &img, const bool force_redraw=true)

  Resize display to the size of an input image.
- CImgDisplay & resize (const CImgDisplay &disp, const bool force_redraw=true)

Resize display to the size of another CImgDisplay instance.

- CImgDisplay & set_normalization (const unsigned int normalization)

  Set normalization type.
- CImgDisplay & set_title (const char ∗const format,...)

  Set title of the associated window.
- CImgDisplay & set_fullscreen (const bool is_fullscreen, const bool force_-redraw=true)

  Enable or disable fullscreen mode.
- CImgDisplay & toggle_fullscreen (const bool force_redraw=true)

  Toggle fullscreen mode.
- CImgDisplay & 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 occuring 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 occuring on the display `disp1`.
- static void wait (CImgDisplay &disp1, CImgDisplay &disp2)

  Wait for any event occuring either on the display `disp1` or `disp2`.
- static void wait (CImgDisplay &disp1, CImgDisplay &disp2, CImgDisplay &disp3)

  Wait for any event occuring either on the display `disp1`, `disp2` or `disp3`.
- static void wait (CImgDisplay &disp1, CImgDisplay &disp2, CImgDisplay &disp3, CImgDisplay &disp4)

  Wait for any event occuring 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 occuring 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 occuring 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 occuring 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 occuring 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 occuring 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 occuring either on the display `disp1`, `disp2`, `disp3`, `disp4`, ... `disp10`.
- static void wait_all ()

  Wait for any window event occuring in any opened CImgDisplay.

8.2.1 Detailed Description

Allow to create windows, display images on them and manage user events (keyboard, mouse and windows events).

CImgDisplay methods 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 outputed each time the program is trying to call one of the CImgDisplay method.

The configuration variable `cimg_display` tells about 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 ( )

Destructor.

Note

If the associated window is visible on the screen, it is closed by the call to the destructor.

See also

CImgDisplay(), assign().

8.2.2.2 CImgDisplay ( )

Create 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); // Create new window and display image in it.
```

See also

∼CImgDisplay(), assign().

8.2.2.3 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`)

Create a display with specified dimensions.

Parameters

width	Window width.
height	Window height.
title	Window title.
normaliza-tion	Normalization type (0=none, 1=always, 2=once, 3=pixel type-dependent, see normalization()).
is_fullscreen	Flag telling if fullscreen mode is enabled.
is_closed	Flag telling if associated window is initially visible or not.

Note

A black background is initially displayed on the associated window.

See also

assign(unsigned int,unsigned int,const char∗,unsigned int,bool,bool).

8.2.2.4 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]`

Create a display from an image.

Parameters

img	Image used as a model to create the window.
title	Window title.
normaliza-tion	Normalization type (0=none, 1=always, 2=once, 3=pixel type-dependent, see normalization()).
is_fullscreen	Flag telling if fullscreen mode is enabled.
is_closed	Flag telling if associated window is initially visible or not.

Note

The pixels of the input image are initially displayed on the associated window.

See also

assign(const CImg<T>&,const char∗,unsigned int,bool,bool).

8.2.2.5 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]`

Create a display from an image list.

Parameters

list	The images list to display.
title	Window title.
normalization	Normalization type (`0`=none, `1`=always, `2`=once, `3`=pixel type-dependent, see normalization()).
is_fullscreen	Flag telling if fullscreen mode is enabled.
is_closed	Flag telling if associated window is initially visible or not.

Note

All images of the list, concatenated along the X-axis, are initially displayed on the associated window.

See also

assign(const CImgList<T>&,const char*,unsigned int,bool,bool).

8.2.2.6 CImgDisplay (const CImgDisplay & *disp*)

Create a display as a copy of an existing one.

Parameters

disp	: Display instance to copy.

Note

The pixel buffer of the input window is initially displayed on the associated window.

See also

assign(const CImgDisplay&).

8.2.3 Member Function Documentation

8.2.3.1 CImgDisplay& assign ( )

Destructor - Empty constructor **[in-place version]**.

Note

Replace the current instance by an empty display.

See also

~CImgDisplay(), CImgDisplay().

8.2.3.2 CImgDisplay& 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`)

Create a display with specified dimensions **[in-place version]**.

See also

CImgDisplay(unsigned int,unsigned int,const char*,unsigned int,bool,bool).

8.2.3.3 CImgDisplay& assign (const CImg< T > & *img*, const char *const *title* = `0`, const unsigned int *normalization* = `3`, const bool *is_fullscreen* = `false`, const bool *is_closed* = `false`)

Create a display from an image **[in-place version]**.

See also

CImgDisplay(const CImg<T>&,const char*,unsigned int,bool,bool).

8.2.3.4 CImgDisplay& assign (const CImgList< T > & *list*, const char *const *title* = `0`, const unsigned int *normalization* = `3`, const bool *is_fullscreen* = `false`, const bool *is_closed* = `false`)

Create a display from an image list **[in-place version]**.

See also

CImgDisplay(const CImgList<T>&,const char*,unsigned int,bool,bool).

8.2.3.5 CImgDisplay& assign (const CImgDisplay & *disp*)

Create a display as a copy of another one **[in-place version]**.

See also

CImgDisplay(const CImgDisplay&).

8.2.3.6 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());
```

8.2.3.7 CImgDisplay& operator= (const CImg< t > & *img*)

Display image on associated window.

Note

`disp = img` is equivalent to `disp.display(img)`.

See also

display(const CImg<T>&).

8.2.3.8 CImgDisplay& operator= (const CImgList< t > & *list*)

Display list of images on associated window.

Note

`disp = list` is equivalent to `disp.display(list)`.

See also

display(const CImgList<T>&,char,float).

8.2.3.9 CImgDisplay& operator= (const CImgDisplay & *disp*)

Create a display as a copy of another one **[in-place version]**.

Note

Equivalent to assign(const CImgDisplay&).

See also

assign(const CImgDisplay&).

8.2.3.10 operator bool ( ) const

Return `false` if display is empty, `true` otherwise.

Note

`if (disp) { ... }` is equivalent to `if (!disp.is_empty()) { ... }`.

See also

is_empty().

8.2.3.11 bool is_empty ( ) const

Return `true` if display is empty, `false` otherwise.

See also

operator bool().

8.2.3.12 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().

See also

is_event(), show(), close().

8.2.3.13 bool is_resized ( ) const

Return `true` if associated window has been resized on the screen, `false` otherwise.

See also

is_event(), resize().

8.2.3.14 bool is_moved ( ) const

Return `true` if associated window has been moved on the screen, `false` otherwise.

See also

is_event(), move().

8.2.3.15 bool is_event ( ) const

Return `true` if any event has occured on the associated window, `false` otherwise.

See also

is_closed(), is_resize(), is_moved(), is_key().

8.2.3.16 bool is_fullscreen ( ) const

Return `true` if current display is in fullscreen mode, `false` otherwise.

See also

set_fullscreen(), toggle_fullscreen().

8.2.3.17 bool is_key ( ) const

Return `true` if any key is being pressed on the associated window, `false` otherwise.

Note

The methods below do the same only for specific keys.

See also

is_event(), is_key(unsigned int) const, is_key(const char ∗) const, is_key_-sequence(), key(), release_key(), set_key(),

8.2.3.18 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

keycode	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();
}
```

See also

is_event(), is_key() const, is_key(const char *) const, is_key_sequence(), key(), released_key(), set_key(),

8.2.3.19 bool is_key (const char *const *keycode*) const

Return `true` if key specified by given keycode is being pressed on the associated window, `false` otherwise.

Parameters

keycode	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();
}
```

See also

is_event(), is_key() const, is_key(unsigned int) const, is_key_sequence(), key(), released_key(), set_key(),

8.2.3.20 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

keycodes_-sequence	Buffer of keycodes to test.
length	Number of keys in the `keycodes_sequence` buffer.
remove_-sequence	Flag telling if 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();
}
```

See also

is_event(), is_key() const, is_key(unsigned int) const, is_key(const char ∗) const, key(), released_key(), set_key(),

8.2.3.21 bool is_keyESC ( ) const

Return `true` if the `ESC` key is being pressed on the associated window, `false` otherwise.

Note

Similar methods exist for all keys managed by `CImg` (see cimg::keyESC).

See also

is_event(), is_key(unsigned int) const, is_key(const char ∗) const, is_key_-sequence(), key(), release_key(), set_key(),

8.2.3.22 static int screen_width ( ) `[static]`

Return width of the screen (current resolution along the X-axis).

See also

screen_height().

8.2.3.23 static int **screen_height** ( ) `[static]`

Return height of the screen (current resolution along the Y-axis).

See also

screen_width().

8.2.3.24 int **width** ( ) const

Return display width.

Note

The width of the display (i.e. the width of the pixel data buffer associated to the CImgDisplay instance) may be different from the actual width of the associated window.

See also

height(), window_width().

8.2.3.25 int **height** ( ) const

Return display height.

Note

The height of the display (i.e. the height of the pixel data buffer associated to the CImgDisplay instance) may be different from the actual height of the associated window.

See also

width(), window_height().

8.2.3.26 unsigned int **normalization** ( ) const

Return normalization type of the display.

The normalization type tells about how the values of an input image are normalized by the CImgDisplay to be correctly displayed. The range of values for pixels displayed on 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.

See also

set_normalization().

8.2.3.27 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.

See also

set_title().

8.2.3.28 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 to the CImgDisplay instance) may be different from the actual width of the associated window.

See also

window_height(), width().

8.2.3.29 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 to the CImgDisplay instance) may be different from the actual height of the associated window.

See also

window_width(), height().

8.2.3.30 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.

See also

window_y().

8.2.3.31 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.

See also

window_x().

8.2.3.32 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].

See also

mouse_y(), button(), wheel().

8.2.3.33 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].

See also

mouse_x(), button(), wheel().

8.2.3.34 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();
}
```

See also

mouse_x(), mouse_y(), wheel().

8.2.3.35 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 substract `1` to the wheel value.
- The returned value cumulates the number of forward of 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();
}
```

See also

mouse_x(), mouse_y(), button(), set_wheel().

8.2.3.36 unsigned int key (const unsigned int *pos* = `0`) const

Return one entry from the pressed keys history.

Parameters

pos	Indice to read from the pressed keys history (indice `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).

See also

is_event(), is_key() const, is_key(unsigned int) const, is_key(const char ∗) const, is_key_sequence(), released_key(), set_key(),

8.2.3.37 unsigned int **released_key** (const unsigned int *pos* = `0`) const

Return one entry from the released keys history.

Parameters

pos	Indice to read from the released keys history (indice `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).

See also

is_event(), is_key() const, is_key(unsigned int) const, is_key(const char ∗) const, is_key_sequence(), released_key(), set_key(),

8.2.3.38 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.
```

8.2.3.39 float frames_per_second ( )

Return the current refresh rate, in frames per second.

Note

Returns a significant value when the current instance is used to display successive frames. It measures the delay between successive calls to frames_per_second().

See also

wait(), wait(unsigned int), cimg::wait().

8.2.3.40 CImgDisplay& display (const CImg< T > & *img*)

Display image on associated window.

Parameters

img	Input image to display.

Note

This method returns immediately.

See also

display(const CImgList<T>&,char,float), CImg<T>::display(const char $*$,bool) const.

8.2.3.41 CImgDisplay& display (const CImgList< T > & *list*, const char *axis* = 'x', const float *align* = 0)

Display list of images on associated window.

Parameters

list	List of images to display.
axis	Axis used to append the images along, for the visualization (can be `x`, `y`, `z` or `c`).
align	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 method returns immediately.

See also

display(const CImg<T>&).

8.2.3.42 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.

See also

close().

8.2.3.43 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.

See also

show().

8.2.3.44 CImgDisplay& move (const int *pos_x*, const int *pos_y*)

Move associated window to a new location.

Parameters

pos_x	X-coordinate of the new window location.
pos_y	Y-coordinate of the new window location.

Note

Depending on the window manager behavior, this method may not succeed (no exceptions are thrown nevertheless).

8.2.3.45 CImgDisplay& resize (const bool *force_redraw* = `true`)

Resize display to the size of the associated window.

Parameters

force_-redraw	Flag telling if the previous window content must be updated and re-freshed as well.

Note

- Calling this method ensures that width() and window_width() become equal, as well as height() and window_height().
- The associated window is also resized to specified dimensions.

See also

width(), height(), window_width(), window_height(), resize(int,int,bool), resize(const CImg<T>&,bool), resize(const CImgDisplay&,bool).

8.2.3.46 CImgDisplay& resize (const int *width*, const int *height*, const bool *force_redraw* = `true`)

Resize display to the specified size.

Parameters

width	Requested display width.
height	Requested display height.
force_-redraw	Flag telling if the previous window content must be updated and re-freshed as well.

Note

The associated window is also resized to specified dimensions.

See also

width(), height(), resize(bool), resize(const CImg<T>&,bool), resize(const CImg-Display&,bool).

8.2.3.47 CImgDisplay& resize (const CImg< T > & *img*, const bool *force_redraw* = `true`)

Resize display to the size of an input image.

Parameters

img	Input image to take size from.
force_-redraw	Flag telling if the previous window content must be resized and updated as well.

Note

- Calling this method ensures that width() and `img.width()` become equal, as well as height() and `img.height()`.
- The associated window is also resized to specified dimensions.

See also

width(), height(), resize(bool), resize(int,int,bool), resize(const CImgDisplay&,bool).

8.2.3.48 CImgDisplay& resize (const CImgDisplay & *disp*, const bool *force_redraw* = `true`)

Resize display to the size of another CImgDisplay instance.

Parameters

disp	Input display to take size from.
force_-redraw	Flag telling if the previous window content must be resized and updated as well.

Note

- Calling this method ensures that width() and `disp.width()` become equal, as well as height() and `disp.height()`.
- The associated window is also resized to specified dimensions.

See also

width(), height(), resize(bool), resize(int,int,bool), resize(const CImg&,bool).

8.2.3.49 CImgDisplay& set_normalization (const unsigned int *normalization*)

Set normalization type.

Parameters

normalization	New normalization mode.

See also

normalization().

8.2.3.50 CImgDisplay& set_title (const char ∗const *format,* ...)

Set title of the associated window.

Parameters

format	C-string containing the format of the title, as with `std::printf()`.

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 nasty memory access.

See also

title().

8.2.3.51 CImgDisplay& set_fullscreen (const bool *is_fullscreen,* const bool *force_redraw* = `true`)

Enable or disable fullscreen mode.

Parameters

is_fullscreen	Flag telling is the fullscreen mode must be activated or not.
force_redraw	Flag telling if the previous window content must be displayed as well.

Note

- When the 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 the largest 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).

See also

toggle_fullscreen().

8.2.3.52 CImgDisplay& toggle_fullscreen (const bool *force_redraw* = `true`)

Toggle fullscreen mode.

Parameters

force_-redraw	Flag telling if the previous window content must be displayed as well.

Note

Enable fullscreen mode if it was not enabled, and disable it otherwise.

See also

set_fullscreen().

8.2.3.53 CImgDisplay& show_mouse ( )

Show mouse pointer.

Note

Depending on the window manager behavior, this method may not succeed (no exceptions are thrown nevertheless).

See also

hide_mouse(), set_mouse().

8.2.3.54 CImgDisplay& hide_mouse ( )

Hide mouse pointer.

Note

Depending on the window manager behavior, this method may not succeed (no exceptions are thrown nevertheless).

See also

show_mouse(), set_mouse().

8.2.3.55 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 method may not succeed (no exceptions are thrown nevertheless).

See also

show_mouse(), hide_mouse(), mouse_x(), mouse_y().

8.2.3.56 CImgDisplay& set_button ( )

Simulate a mouse button release event.

Note

All mouse buttons are considered released at the same time.

See also

button(), set_mouse(int,int), set_button(unsigned int,bool).

8.2.3.57 CImgDisplay& set_button (const unsigned int *button*, const bool *is_pressed* = `true`)

Simulate a mouse button press or release event.

Parameters

button	Buttons event code, where each button is associated to a single bit.
is_pressed	Flag telling if the mouse button is considered as pressed or released.

See also

button(), set_mouse(), set_button(unsigned int,bool).

8.2.3.58 CImgDisplay& set_wheel ( )

Flush all mouse wheel events.

Note

Make wheel() to return `0`, if called afterwards.

See also

wheel(), set_wheel(int).

8.2.3.59 CImgDisplay& set_wheel (const int *amplitude*)

Simulate a wheel event.

Parameters

amplitude	Amplitude of the wheel scrolling to simulate.

Note

Make wheel() to return `amplitude`, if called afterwards.

See also

wheel(), set_wheel().

8.2.3.60 CImgDisplay& set_key ( )

Flush all key events.

Note

Make key() to return `0`, if called afterwards.

See also

key(), set_key(unsigned int,bool).

8.2.3.61 CImgDisplay& set_key (const unsigned int *keycode*, const bool *is_pressed* = `true`)

Simulate a keyboard press/release event.

Parameters

keycode	Keycode of the associated key.
is_pressed	Flag telling if 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).

See also

key(), set_key().

8.2.3.62 CImgDisplay& flush ( )

Flush all display events.

Note

Remove all passed events from the current display.

See also

set_button(), set_key(), set_wheel().

8.2.3.63 CImgDisplay& wait (const unsigned int *milliseconds*)

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

Parameters

milliseconds	Number of milliseconds to wait for.

Note

Similar to cimg::wait().

See also

cimg::wait().

8.2.3.64 CImgDisplay& render (const CImg< T > & *img*)

Render image into internal display buffer.

Parameters

img	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.

See also

paint(), snapshot().

8.2.3.65 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.

See also

render(), snapshot().

8.2.3.66 const CImgDisplay& snapshot (CImg< T > & *img*) const

Take a snapshot of the associated window content.

Parameters

`out`	*img*	Output snapshot. Can be empty on input.

See also

render(), paint().

8.3 CImgException Struct Reference

Instances of `CImgException` are thrown when errors are encountered in a `CImg` function call.

Inherited by CImgArgumentException, CImgDisplayException, CImgInstance-Exception, CImgIOException, and CImgWarningException.

Public Member Functions

- const char ∗ what () const throw ()

  Return a C-string containing the error message associated to 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`. CImgException is never thrown itself. Derived classes that specify the type of errord are thrown instead. These derived classes can be :

- **CImgArgumentException** : Thrown when one argument of a called `CImg` function is invalid. This is probably one of the most thrown exception by `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 to a called `C-Img` method does not fit the function 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 occured 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 by yourself, since they are expected to be thrown only by `CImg`. 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 get 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 () { ... }` bloc, as in the following example :

```
#define "CImg.h"
using namespace cimg_library;
int main() {
  cimg::exception_mode(0);                                    // Enable
 quiet exception mode.
  try {
    ...                                                       // Here, do
 what you want to stress the CImg library.
  } catch (CImgException &e) {                                // You
 succeeded : something went wrong !
    std::fprintf(stderr,"CImg Library Error : %s",e.what());  // Display
 your custom error message.
    ...                                                       // Do what
 you want now to save the ship !
    }
  }
```

8.4 CImgList< T > Struct Template Reference

Represent a list of images CImg<T>.

Public Types

- typedef CImg< T > ∗ iterator

  Simple iterator type, to loop through each image of a list instance.

- 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 ()

    Create an empty list.
- CImgList (const unsigned int n)

    Create a list containing n 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)

    Create a list containing n images with 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)

    Create a list containing n images with specified size, and initialize pixel values.
- CImgList (const unsigned int n, const unsigned int width, const unsigned int height, const unsigned int depth, const unsigned int spectrum, const int val0, const int val1,...)

    Create a list containing n images with specified size, and initialize pixel values from a sequence of integers.
- CImgList (const unsigned int n, const unsigned int width, const unsigned int height, const unsigned int depth, const unsigned int spectrum, const double val0, const double val1,...)

    Create a list containing n images with specified size, and initialize pixel values from a sequence of doubles.
- template< typename t >
  CImgList (const unsigned int n, const CImg< t > &img, const bool is_-shared=false)

    Create a list containing n copies of the same input image.
- template< typename t >
  CImgList (const CImg< t > &img, const bool is_shared=false)

    Create a list containing one copy of an input image.
- template< typename t1 , typename t2 >
  CImgList (const CImg< t1 > &img1, const CImg< t2 > &img2, const bool is_-shared=false)

    Create a list from two images.
- 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)

    Create a list from three images.

- 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)

  Create a list from four images.
- 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)

  Create a list from five images.
- 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)

  Create a list from six images.
- 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)

  Create a list from seven images.
- 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)

  Create a list from eight images.
- template<typename t >
CImgList (const CImgList< t > &list)

  Create a list as a copy of an existing list.
- CImgList (const CImgList< T > &list)

  *Create a list as a copy of an existing list **[specialization]**.*
- template<typename t >
CImgList (const CImgList< t > &list, const bool is_shared)

  Create a list as a copy of an existing list, and force the shared state of the list elements.
- CImgList (const char ∗const filename)

  Create a list from the content of a file.
- CImgList (const CImgDisplay &disp)

  Create a list from the content of a display window.
- CImgList< T > get_shared ()

  Return a list with elements being shared copies of images in the list instance.
- const CImgList< T > get_shared () const

  *Return a list with elements being shared copies of images in the list instance **[const version]**.*
- CImgList< T > & assign ()

  *Create an empty list **[in-place version]**.*

- CImgList< T > & clear ()

  *Create an empty list **[in-place version]**.*
- CImgList< T > & assign (const unsigned int n)

  *Create a list containing n empty images **[in-place version]**.*
- 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)

  *Create a list containing n images with 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)

  *Create a list containing n images with specified size, and initialize pixel values **[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 int val0, const int val1,...)

  *Create a list containing n images with specified size, and initialize pixel values from a sequence of integers **[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 double val0, const double val1,...)

  *Create a list containing n images with specified size, and initialize pixel values from a sequence of doubles **[in-place version]**.*
- template<typename t >
  CImgList< T > & assign (const unsigned int n, const CImg< t > &img, const bool is_shared=false)

  *Create a list containing n copies of the same input image **[specialization] [in-place version]**.*
- template<typename t >
  CImgList< T > & assign (const CImg< t > &img, const bool is_shared=false)

  *Create a list containing one copy of an input 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)

  *Create a 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)

  *Create a 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)

  *Create a 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)

Create a 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)

  Create a 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)

  Create a 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)

  Create a list from eight images ***[in-place version]****.*

- template<typename t >
CImgList< T > & assign (const CImgList< t > &list, const bool is_-shared=false)

  Create a 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)

  Create a list containing n copies of the same input image ***[specialization] [in-place version]****.*

- CImgList< T > & assign (const char ∗const filename)

  Create a list from the content of a file ***[in-place version]****.*

- CImgList< T > & assign (const CImgDisplay &disp)

  Create a list from the content of a display window ***[in-place version]****.*

- template<typename t >
CImgList< t > & move_to (CImgList< t > &list)

  Transfer the content of the list instance into another list.

- template<typename t >
CImgList< t > & move_to (CImgList< t > &list, const unsigned int pos)

  Transfer the content of the list instance at a specified position in another list.

- CImgList< T > & swap (CImgList< T > &list)

  Swap all fields between list instance and another list.

- static CImgList< T > & empty ()

  Return a reference to an empty list.

Overloaded Operators

- CImg< T > & operator() (const unsigned int pos)

  Return a reference to the `pos-th` image of the list instance.
- const CImg< T > & operator() (const unsigned int pos) const

  Return a reference to the `pos-th` 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 the (x,y,z,c) pixel value of the `pos-th` 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 the (x,y,z,c) pixel value of the `pos-th` 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)

  *Create a list containing one copy of an input image **[in-place version]**.*
- template<typename t >
  CImgList< T > & operator= (const CImgList< t > &list)

  Create a list as a copy of an existing list.
- CImgList< T > & operator= (const CImgList< T > &list)

  *Create a list as a copy of an existing list **[specialization]**.*
- CImgList< T > & operator= (const char *const filename)

  *Create a list from the content of a file **[in-place version]**.*
- CImgList< T > & operator= (const CImgDisplay &disp)

  *Create a 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 CImgList< t > &list)

  Return a copy of the list instance, where all elements of input list `list` have been inserted at the end.
- CImg< T > operator> (const char axis) const

  Return image corresponding to the concatenation of all images of the instance list along specified axis.
- CImgList< T > operator< (const char axis) const

  Return 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.
- CImg< T > ∗ data ()

  Return pointer to the first image of the list.
- const CImg< T > ∗ data () const

  *Return pointer to the first image of the list **[const version]**.*
- CImg< T > ∗ data (const unsigned int pos)

  Return pointer to the pos-th image of the list.
- const CImg< T > ∗ **data** (const unsigned int l) const
- 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]**.*
- CImg< T > & front ()

  Return reference to the first image of the list.
- const CImg< T > & front () const

  *Return reference to the first image of the list **[const version]**.*
- const CImg< T > & back () const

  Return a reference to the last image of the list.
- CImg< T > & back ()

  *Return a reference to the last image of the list **[const version]**.*
- CImg< 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 to pixel value with Dirichlet boundary conditions for the three first 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 to pixel value with Dirichlet boundary conditions for the three first 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 four first 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 four first 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 three first 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 three first 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 three first 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 three first 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 two first 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 two first 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 two first 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 two first 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 first coordinates (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 first coordinates (`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 first coordinates (`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 first coordinates (`pos`) **[const version]**.*
- CImg< charT > value_string (const char separator=',', const unsigned int max_-size=0) const

  Return a C-string containing the values of all images in the instance list.
- 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

  Return `true` if list has `n` images.
- template<typename t >
  bool is_sameN (const CImgList< t > &list) const

  Return `true` if list has `n` images.
- template<typename t >
  bool **is_sameXY** (const CImg< t > &img) const
- template<typename t >
  bool **is_sameXY** (const CImgList< t > &list) const
- template<typename t >
  bool **is_sameNXY** (const unsigned int n, const CImg< t > &img) const
- template<typename t >
  bool **is_sameNXY** (const CImgList< t > &list) const
- template<typename t >
  bool **is_sameXZ** (const CImg< t > &img) const
- template<typename t >
  bool **is_sameXZ** (const CImgList< t > &list) const
- template<typename t >
  bool **is_sameNXZ** (const unsigned int n, const CImg< t > &img) const
- template<typename t >
  bool **is_sameNXZ** (const CImgList< t > &list) const
- template<typename t >
  bool **is_sameXC** (const CImg< t > &img) const
- template<typename t >
  bool **is_sameXC** (const CImgList< t > &list) const
- template<typename t >
  bool **is_sameNXC** (const unsigned int n, const CImg< t > &img) const

- template<typename t >
  bool **is_sameNXC** (const CImgList< t > &list) const
- template<typename t >
  bool **is_sameYZ** (const CImg< t > &img) const
- template<typename t >
  bool **is_sameYZ** (const CImgList< t > &list) const
- template<typename t >
  bool **is_sameNYZ** (const unsigned int n, const CImg< t > &img) const
- template<typename t >
  bool **is_sameNYZ** (const CImgList< t > &list) const
- template<typename t >
  bool **is_sameYC** (const CImg< t > &img) const
- template<typename t >
  bool **is_sameYC** (const CImgList< t > &list) const
- template<typename t >
  bool **is_sameNYC** (const unsigned int n, const CImg< t > &img) const
- template<typename t >
  bool **is_sameNYC** (const CImgList< t > &list) const
- template<typename t >
  bool **is_sameXYZ** (const CImg< t > &img) const
- template<typename t >
  bool **is_sameXYZ** (const CImgList< t > &list) const
- template<typename t >
  bool **is_sameNXYZ** (const unsigned int n, const CImg< t > &img) const
- template<typename t >
  bool **is_sameNXYZ** (const CImgList< t > &list) const
- template<typename t >
  bool **is_sameXYC** (const CImg< t > &img) const
- template<typename t >
  bool **is_sameXYC** (const CImgList< t > &list) const
- template<typename t >
  bool **is_sameNXYC** (const unsigned int n, const CImg< t > &img) const
- template<typename t >
  bool **is_sameNXYC** (const CImgList< t > &list) const
- template<typename t >
  bool **is_sameYZC** (const CImg< t > &img) const
- template<typename t >
  bool **is_sameYZC** (const CImgList< t > &list) const
- template<typename t >
  bool **is_sameNYZC** (const unsigned int n, const CImg< t > &img) const
- template<typename t >
  bool **is_sameNYZC** (const CImgList< t > &list) const
- template<typename t >
  bool **is_sameXYZC** (const CImg< t > &img) const
- template<typename t >
  bool **is_sameXYZC** (const CImgList< t > &list) const
- template<typename t >
  bool **is_sameNXYZC** (const unsigned int n, const CImg< t > &img) const

- template<typename t >
  bool **is_sameNXYZC** (const CImgList< t > &list) const
- bool **is_sameX** (const unsigned int val) const
- bool **is_sameNX** (const unsigned int n, const unsigned int val) const
- bool **is_sameY** (const unsigned int val) const
- bool **is_sameNY** (const unsigned int n, const unsigned int val) const
- bool **is_sameZ** (const unsigned int val) const
- bool **is_sameNZ** (const unsigned int n, const unsigned int val) const
- bool **is_sameC** (const unsigned int val) const
- bool **is_sameNC** (const unsigned int n, const unsigned int val) const
- bool **is_sameXY** (const unsigned int val1, const unsigned int val2) const
- bool **is_sameNXY** (const unsigned int n, const unsigned int val1, const unsigned int val2) const
- bool **is_sameXZ** (const unsigned int val1, const unsigned int val2) const
- bool **is_sameNXZ** (const unsigned int n, const unsigned int val1, const unsigned int val2) const
- bool **is_sameXC** (const unsigned int val1, const unsigned int val2) const
- bool **is_sameNXC** (const unsigned int n, const unsigned int val1, const unsigned int val2) const
- bool **is_sameYZ** (const unsigned int val1, const unsigned int val2) const
- bool **is_sameNYZ** (const unsigned int n, const unsigned int val1, const unsigned int val2) const
- bool **is_sameYC** (const unsigned int val1, const unsigned int val2) const
- bool **is_sameNYC** (const unsigned int n, const unsigned int val1, const unsigned int val2) const
- bool **is_sameZC** (const unsigned int val1, const unsigned int val2) const
- bool **is_sameNZC** (const unsigned int n, const unsigned int val1, const unsigned int val2) const
- bool **is_sameXYZ** (const unsigned int val1, const unsigned int val2, const unsigned int val3) const
- bool **is_sameNXYZ** (const unsigned int n, const unsigned int val1, const unsigned int val2, const unsigned int val3) const
- bool **is_sameXYC** (const unsigned int val1, const unsigned int val2, const unsigned int val3) const
- bool **is_sameNXYC** (const unsigned int n, const unsigned int val1, const unsigned int val2, const unsigned int val3) const
- bool **is_sameXZC** (const unsigned int val1, const unsigned int val2, const unsigned int val3) const
- bool **is_sameNXZC** (const unsigned int n, const unsigned int val1, const unsigned int val2, const unsigned int val3) const
- bool **is_sameYZC** (const unsigned int val1, const unsigned int val2, const unsigned int val3) const
- bool **is_sameNYZC** (const unsigned int n, const unsigned int val1, const unsigned int val2, const unsigned int val3) const
- bool is_sameXYZC (const unsigned int dx, const unsigned int dy, const unsigned int dz, const unsigned int dc) const

  Return `true` 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

  Return `true` 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

  Return `true` if list contains pixel located at (n,x,y,z,c).
- bool containsN (const int n) const

  Return `true` if list contains image with index $[n]$.
- template< typename t >
  bool contains (const T &pixel, t &n, t &x, t &y, t &z, t &c) const

  Return `true` 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

  Return `true` if one of the image list contains the specified referenced value.
- template< typename t >
  bool contains (const T &pixel, t &n, t &x, t &y) const

  Return `true` if one of the image list contains the specified referenced value.
- template< typename t >
  bool contains (const T &pixel, t &n, t &x) const

  Return `true` if one of the image list contains the specified referenced value.
- template< typename t >
  bool contains (const T &pixel, t &n) const

  Return `true` if one of the image list contains the specified referenced value.
- bool contains (const T &pixel) const

  Return `true` if one of the image list contains the specified referenced value.
- template< typename t >
  bool contains (const CImg< T > &img, t &n) const

  Return `true` if the list contains the image 'img'.
- bool contains (const CImg< T > &img) const

  Return `true` 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 vvalue 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 vvalue as well **[const version]**.*

- template<typename t >
  T & max_min (t &min_val)

  Return a reference to the minimum 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`.

- CImgList< 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 >
  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`.

- template<typename t >
  CImgList< T > get_insert (const unsigned int n, const CImg< 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 >
  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`.
- template<typename t >
  CImgList< T > get_insert (const CImgList< 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 >
  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.
- template<typename t >
  CImgList< T > get_insert (const unsigned int n, const CImgList< 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]**.*
- CImgList< T > & remove (const unsigned int pos1, const unsigned int pos2)

  Remove all images between from indexes.
- CImgList< T > get_remove (const unsigned int pos1, const unsigned int pos2) const

  *Remove all images between from indexes **[new-instance version]**.*
- CImgList< T > & remove (const unsigned int pos)

  Remove image at index `pos` from the image list.
- CImgList< T > get_remove (const unsigned int pos) const

  *Remove image at index `pos` from the image list **[new-instance version]**.*
- CImgList< T > & remove ()

  Remove last image.
- CImgList< T > get_remove () const

  *Remove last image **[new-instance version]**.*
- CImgList< T > & reverse ()

  Reverse list order.
- CImgList< T > get_reverse () const

  *Reverse list order **[new-instance version]**.*
- CImgList< T > & images (const unsigned int pos0, const unsigned int pos1)

  Return a sublist.
- CImgList< T > get_images (const unsigned int pos0, const unsigned int pos1) const

  *Return a sublist **[new-instance version]**.*
- CImgList< T > get_shared_images (const unsigned int pos0, const unsigned int pos1)

  Return a shared sublist.

- const CImgList< T > get_shared_images (const unsigned int pos0, const unsigned int pos1) const

    *Return a shared sublist **[new-instance version]**.*
- 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.
- CImgList< T > & split (const char axis, const int nb=0)

    Return a list where each image has been split along the specified axis.
- CImgList< T > get_split (const char axis, const int nb=0) 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

    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

    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_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.
- 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 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 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.
- CImgList< T > & load_ffmpeg (const char ∗const filename, const unsigned int first_frame=0, const unsigned int last_frame=∼0U, const unsigned int step_frame=1, const bool pixel_format=true, const bool resume=false)

  Load an image from a video file, using ffmpeg libraries.
- CImgList< T > & load_ffmpeg_external (const char ∗const filename)

  Load an image from a video file using the external tool 'ffmpeg'.
- CImgList< T > & load_gzip_external (const char ∗const filename)

  Load a gzipped list, using external tool 'gunzip'.
- template<typename tf , typename tc >
  CImgList< T > & load_off (const char ∗const filename, CImgList< tf > &primitives, CImgList< tc > &colors)

  Load a 3d object from a .OFF file.
- 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)

  Load a multi-page TIFF file.
- static CImgList< T > get_load (const char ∗const filename)

  *Load a list from a file **[new-instance version]**.*
- static CImgList< T > get_load_cimg (const char ∗const filename)

  *Load a list from a .cimg file **[new-instance version]**.*
- static CImgList< T > get_load_cimg (std::FILE ∗const file)

  *Load a list from a .cimg file **[new-instance version]**.*
- static CImgList< 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 CImgList< 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 CImgList< T > get_load_parrec (const char ∗const filename)

  Load a list from a PAR/REC (Philips) file ***[new-instance version]****.*
- static CImgList< T > get_load_yuv (const char ∗const filename, const unsigned int size_x, const unsigned int size_y=1, 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 CImgList< T > get_load_yuv (std::FILE ∗const file, const unsigned int size-_x, const unsigned int size_y=1, 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 CImgList< T > get_load_ffmpeg (const char ∗const filename, const unsigned int first_frame=0, const unsigned int last_frame=∼0U, const unsigned int step_frame=1, const bool pixel_format=true)

  Load an image from a video file, using ffmpeg libraries ***[new-instance version]****.*
- static CImgList< 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 CImgList< T > get_load_gzip_external (const char ∗const filename)

  Load a gzipped list, using external tool 'gunzip' ***[new-instance version]****.*
- template<typename tf , typename tc >
  static CImgList< T > get_load_off (const char ∗const filename, CImgList< tf > &primitives, CImgList< tc > &colors)

  Load a 3d object from a .OFF file ***[new-instance version]****.*
- static CImgList< 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)

  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 informations 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) 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) const

  Display the current CImgList instance in a new display window.
- const CImgList< T > & save (const char *const filename, const int number=-1) const

  Save a list into a file.
- const CImgList< T > & save_ffmpeg (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=∼0U, const unsigned int fps=25, const unsigned int bitrate=2048) const

  Save an image sequence, using FFMPEG library.
- const CImgList< T > & save_yuv (const char *const filename=0, const bool rgb2yuv=true) const

  Save list as a YUV image sequence file.
- const CImgList< T > & save_yuv (std::FILE *const file, const bool rgb2yuv=true) const

  Save an image sequence into a YUV file.
- const CImgList< T > & save_cimg (const char *const filename, const bool compress=false) const

  Save a list into a .cimg file.
- const CImgList< T > & save_cimg (std::FILE *file, const bool compress=false) const

  Save a list into 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 instance into 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 instance into into an existing .cimg file, at specified coordinates.
- const CImgList< T > & save_tiff (const char *const filename, const unsigned int compression=0) const

  Save a file in TIFF format.
- const CImgList< T > & save_gzip_external (const char *const filename) const

  Save a list as a gzipped file, using external tool 'gzip'.
- const CImgList< T > & save_ffmpeg_external (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=∼0U, const char *const codec=0, const unsigned int fps=25, const unsigned int bitrate=2048) const

  Save an image sequence using the external tool 'ffmpeg'.
- static bool **is_saveable** (const char *const filename)
- 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)

  Create 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)

  Create an empty .cimg file with specified dimensions.

Others

- CImgList< T > & crop_font ()

  Create an auto-cropped font (along the X axis) from a input font.
- CImgList< T > get_crop_font () const

  *Create an auto-cropped font (along the X axis) from a input font **[new-instance version]**.*
- 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 a n-d Fast Fourier Transform.
- CImgList< Tfloat > get_FFT (const bool invert=false) const

  *Compute a n-d Fast Fourier Transform **[new-instance version]**.*
- CImgList< T > & reverse_object3d ()

  Reverse primitives orientations of a 3d object.
- CImgList< T > get_reverse_object3d () const

  *Reverse primitives orientations of a 3d object **[new-instance version]**.*
- static const CImgList< T > & font (const unsigned int font_height, const bool is_variable_width=true)

  Return a CImg pre-defined font with desired size.

8.4.1 Detailed Description

template<typename T>struct cimg_library::CImgList< T >

Represent a list of images CImg<T>.

8.4.2 Member Typedef Documentation

8.4.2.1 typedef CImg<T>∗ iterator

Simple iterator type, to loop through each image of a list instance.

Note

- The `CImgList<T>::iterator` type is defined to be 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');
```

- Anyway, using loop macro `cimglist_for` will probably generate more concise code :

```
cimglist_for(list,l) list[l].mirror('x');
```

See also

const_iterator, `cimglist_apply`.

8.4.2.2 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.

See also

iterator.

8.4.2.3 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 methods. It has been mainly defined for compatibility with STL naming conventions.

8.4.3 Constructor & Destructor Documentation

8.4.3.1 ~CImgList ( )

Destructor.

Destroy current list.

Note

- Call the destructors for each image of the list.
- Destroying an empty list does actually nothing.

See also

CImgList(), assign().

8.4.3.2 CImgList ( )

Create an empty list.

Note

Constructing an empty CImgList doesn't allocate extra memory buffer.

See also

∼CImgList(), assign().

8.4.3.3 CImgList (const unsigned int *n*) `[explicit]`

Create a list containing `n` empty images.

Parameters

n	Number of empty images in the constructed instance.

Note

Useful when you know by advance the number of images you want to manage, as it will allocate the right amount of memory for the list, without needing for reallocation (that may occur when starting from an empty list and inserting several images in it).

See also

assign(unsigned int).

8.4.3.4 CImgList (const unsigned int *n*, const unsigned int *width*, const unsigned int *height* = `1`, const unsigned int *depth* = `1`, const unsigned int *spectrum* = `1`)

Create a list containing `n` images with specified size.

Parameters

n	Number of images in the constructed instance.
width	Desired images width.
height	Desired images height.
depth	Desired images depth.
spectrum	Desired number of image channels.

See also

assign(unsigned int,unsigned int,unsigned int,unsigned int,unsigned int).

8.4.3.5 CImgList (const unsigned int *n,* const unsigned int *width,* const unsigned int *height,* const unsigned int *depth,* const unsigned int *spectrum,* const T *val*)

Create a list containing `n` images with specified size, and initialize pixel values.

Parameters

n	Number of images in the constructed instance.
width	Desired images width.
height	Desired images height.
depth	Desired images depth.
spectrum	Desired number of image channels.
val	Value to initialize images pixels.

Note

Similar to CImgList(unsigned int,unsigned int,unsigned int,unsigned int,unsigned int) with a default value set for all image pixels.

See also

assign(unsigned int,unsigned int,unsigned int,unsigned int,unsigned int,const T).

8.4.3.6 CImgList (const unsigned int *n,* const unsigned int *width,* const unsigned int *height,* const unsigned int *depth,* const unsigned int *spectrum,* const int *val0,* const int *val1,* ...)

Create a list containing `n` images with specified size, and initialize pixel values from a sequence of integers.

Parameters

n	Number of images in the constructed instance.
width	Desired images width.
height	Desired images height.

depth	Desired images depth.
spectrum	Desired number of image channels.
val0	First value of an integers sequence to initialize images pixels.
val1	Secon value of an integers sequence to initialize images pixels.

Note

Similar to CImgList(unsigned int,unsigned int,unsigned int,unsigned int,unsigned int,const T) with a sequence of integers to initialize pixel values.

See also

assign(unsigned int,unsigned int,unsigned int,unsigned int,unsigned int,int,int,...).

8.4.3.7 CImgList (const unsigned int *n,* const unsigned int *width,* const unsigned int *height,* const unsigned int *depth,* const unsigned int *spectrum,* const double *val0,* const double *val1,* *...*)

Create a list containing `n` images with specified size, and initialize pixel values from a sequence of doubles.

Parameters

n	Number of images in the constructed list.
width	Desired images width.
height	Desired images height.
depth	Desired images depth.
spectrum	Desired number of image channels.
val0	First value of a doubles sequence to initialize images pixels.
val1	Second value of a doubles sequence to initialize images pixels.

Note

Similar to CImgList(unsigned int,unsigned int,unsigned int,unsigned int,unsigned int,int,int,...) with a sequence of doubles to initialize pixel values.

See also

assign(unsigned int,unsigned int,unsigned int,unsigned int,unsigned int,double,double,...).

8.4.3.8 CImgList (const unsigned int *n,* const CImg< t > & *img,* const bool *is_shared* = `false`)

Create a list containing `n` copies of the same input image.

Parameters

n	Number of images in the constructed list.
img	Input image to clone `n` times in the created list.
is_shared	Flag telling if the elements of the list are shared or non-shared copies of `img`.

See also

assign(unsigned int,const CImg<t>&,bool).

8.4.3.9 CImgList (const CImg< t > & *img*, const bool *is_shared* = `false`) `[explicit]`

Create a list containing one copy of an input image.

Parameters

img	Input image to clone in the constructed list.
is_shared	Flag telling if the element of the list is a shared or non-shared copy of `img`.

See also

assign(const CImg<t>&,bool).

8.4.3.10 CImgList (const CImg< t1 > & *img1*, const CImg< t2 > & *img2*, const bool *is_shared* = `false`)

Create a list from two images.

Parameters

img1	First input image to clone in the constructed list.
img2	Second input image to clone in the constructed list.
is_shared	Flag telling if the elements of the list are shared or non-shared copies of input images.

See also

assign(const CImg<t1>&,const CImg<t2>&,bool).

8.4.3.11 CImgList (const CImg< t1 > & *img1*, const CImg< t2 > & *img2*, const CImg< t3 > & *img3*, const bool *is_shared* = `false`)

Create a list from three images.

Parameters

img1	First input image to clone in the constructed list.
img2	Second input image to clone in the constructed list.
img3	Third input image to clone in the constructed list.
is_shared	Flag telling if the elements of the list are shared or non-shared copies of input images.

See also

assign(const CImg<t1>&,const CImg<t2>&,const CImg<t3>&,bool).

8.4.3.12 CImgList (const CImg< t1 > & *img1*, const CImg< t2 > & *img2*, const CImg< t3 > & *img3*, const CImg< t4 > & *img4*, const bool *is_shared* = `false`)

Create a list from four images.

Parameters

img1	First input image to clone in the constructed list.
img2	Second input image to clone in the constructed list.
img3	Third input image to clone in the constructed list.
img4	Fourth input image to clone in the constructed list.
is_shared	Flag telling if the elements of the list are shared or non-shared copies of input images.

See also

assign(const CImg<t1>&,const CImg<t2>&,const CImg<t3>&,const C-Img<t4>&,bool).

8.4.3.13 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`)

Create a list from five images.

Parameters

img1	First input image to clone in the constructed list.
img2	Second input image to clone in the constructed list.
img3	Third input image to clone in the constructed list.
img4	Fourth input image to clone in the constructed list.
img5	Fifth input image to clone in the constructed list.
is_shared	Flag telling if the elements of the list are shared or non-shared copies of input images.

See also

assign(const CImg<t1>&,const CImg<t2>&,const CImg<t3>&,const CImg<t4>&,const CImg<t5>&,bool).

8.4.3.14 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`)

Create a list from six images.

Parameters

img1	First input image to clone in the constructed list.
img2	Second input image to clone in the constructed list.
img3	Third input image to clone in the constructed list.
img4	Fourth input image to clone in the constructed list.
img5	Fifth input image to clone in the constructed list.
img6	Sixth input image to clone in the constructed list.
is_shared	Flag telling if the elements of the list are shared or non-shared copies of input images.

See also

assign(const CImg<t1>&,const CImg<t2>&,const CImg<t3>&,const CImg<t4>&,const CImg<t5>&,const CImg<t6>&,bool).

8.4.3.15 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`)

Create a list from seven images.

Parameters

img1	First input image to clone in the constructed list.
img2	Second input image to clone in the constructed list.
img3	Third input image to clone in the constructed list.
img4	Fourth input image to clone in the constructed list.
img5	Fifth input image to clone in the constructed list.
img6	Sixth input image to clone in the constructed list.
img7	Seventh input image to clone in the constructed list.
is_shared	Flag telling if the elements of the list are shared or non-shared copies of input images.

See also

assign(const CImg<t1>&,const CImg<t2>&,const CImg<t3>&,const CImg<t4>&,const CImg<t5>&,const CImg<t6>&,const CImg<t7>&,bool).

8.4.3.16 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`)

Create a list from eight images.

Parameters

img1	First input image to clone in the constructed list.
img2	Second input image to clone in the constructed list.
img3	Third input image to clone in the constructed list.
img4	Fourth input image to clone in the constructed list.
img5	Fifth input image to clone in the constructed list.
img6	Sixth input image to clone in the constructed list.
img7	Seventh input image to clone in the constructed list.
img8	Eighth input image to clone in the constructed list.
is_shared	Flag telling if the elements of the list are shared or non-shared copies of input images.

See also

assign(const CImg<t1>&,const CImg<t2>&,const CImg<t3>&,const CImg<t4>&,const CImg<t5>&,const CImg<t6>&,const CImg<t7>&,const CImg<t8>&,bool).

8.4.3.17 CImgList (const CImgList< t > & *list*)

Create a list as a copy of an existing list.

Parameters

list	Input list to copy.

Note

The shared state of each element of the constructed list is the same as in the input list `list`.

See also

assign(const CImgList<t>&).

8.4.3.18 CImgList (const CImgList< t > & *list*, const bool *is_shared*)

Create a list as a copy of an existing list, and force the shared state of the list elements.

Parameters

list	Input list to copy.
is_shared	Flag telling if the elements of the list are shared or non-shared copies of input images.

See also

assign(const CImgList<t>&,bool).

8.4.3.19 CImgList (const char ∗const *filename*) `[explicit]`

Create a list from the content of a file.

Parameters

filename	Filename.

Note

Some file formats support the storage of multiple images in the same file. In this case, all available images are read and stored in the constructed list.

See also

assign(const char ∗const), load(const char ∗const).

8.4.3.20 CImgList (const CImgDisplay & *disp*) `[explicit]`

Create a list from the content of a display window.

Parameters

disp	Display window to get content from.

Note

The created list contains only a single image.

See also

assign(const CImgDisplay&).

8.4.4 Member Function Documentation

8.4.4.1 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)`.

See also

assign(const CImg<T>&,bool).

8.4.4.2 const CImgList<T> get_shared ( ) const

Return a list with elements being shared copies of images in the list instance **[const version]**.

Note

Similar to get_shared() for `const` list instances.

8.4.4.3 CImgList<T>& assign ( )

Create an empty list **[in-place version]**.

Note

Replace the instance by an empty list.

See also

CImgList().

8.4.4.4 CImgList<T>& clear ( )

Create an empty list **[in-place version]**.

Equivalent to assign().

Note

It has been defined for compatibility with STL naming conventions.

See also

assign().

8.4.4.5 CImgList<T>& assign (const unsigned int *n*)

Create a list containing `n` empty images **[in-place version]**.

Parameters

n	Number of empty images in the constructed instance.

See also

CImgList(unsigned int).

8.4.4.6 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`)

Create a list containing `n` images with specified size **[in-place version]**.

Parameters

n	Number of images in the constructed instance.
width	Desired images width.
height	Desired images height.
depth	Desired images depth.
spectrum	Desired number of image channels.

See also

CImgList(unsigned int,unsigned int,unsigned int,unsigned int,unsigned int).

8.4.4.7 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*)

Create a list containing `n` images with specified size, and initialize pixel values **[in-place version]**.

Parameters

n	Number of images in the constructed instance.
width	Desired images width.
height	Desired images height.
depth	Desired images depth.
spectrum	Desired number of image channels.
val	Value to initialize images pixels.

See also

CImgList(unsigned int,unsigned int,unsigned int,unsigned int,unsigned int,const T).

8.4.4.8 CImgList<T>& assign (const unsigned int *n*, const unsigned int *width*, const unsigned int *height*, const unsigned int *depth*, const unsigned int *spectrum*, const int *val0*, const int *val1*, ...)

Create a list containing `n` images with specified size, and initialize pixel values from a sequence of integers **[in-place version]**.

Parameters

n	Number of images in the constructed instance.
width	Desired images width.
height	Desired images height.
depth	Desired images depth.
spectrum	Desired number of image channels.
val0	First value of an integers sequence to initialize images pixels.
val1	Secon value of an integers sequence to initialize images pixels.

See also

CImgList(unsigned int,unsigned int,unsigned int,unsigned int,unsigned int,int,int,...).

8.4.4.9 CImgList<T>& assign (const unsigned int *n*, const unsigned int *width*, const unsigned int *height*, const unsigned int *depth*, const unsigned int *spectrum*, const double *val0*, const double *val1*, ...)

Create a list containing `n` images with specified size, and initialize pixel values from a sequence of doubles **[in-place version]**.

Parameters

n	Number of images in the constructed list.
width	Desired images width.
height	Desired images height.
depth	Desired images depth.
spectrum	Desired number of image channels.
val0	First value of a doubles sequence to initialize images pixels.
val1	Second value of a doubles sequence to initialize images pixels.

See also

CImgList(unsigned int,unsigned int,unsigned int,unsigned int,unsigned int,double,double,...).

8.4.4.10 CImgList<T>& assign (const CImg< t > & *img*, const bool *is_shared* = `false`)

Create a list containing one copy of an input image **[in-place version]**.

Parameters

img	Input image to clone in the constructed list.
is_shared	Flag telling if the element of the list is a shared or non-shared copy of `img`.

See also

CImgList(const CImg<t>&,bool).

8.4.4.11 CImgList<T>& assign (const CImg< t1 > & *img1*, const CImg< t2 > & *img2*, const bool *is_shared* = `false`)

Create a list from two images **[in-place version]**.

Parameters

img1	First input image to clone in the constructed list.
img2	Second input image to clone in the constructed list.
is_shared	Flag telling if the elements of the list are shared or non-shared copies of input images.

See also

CImgList(const CImg<t1>&,const CImg<t2>&,bool).

8.4.4.12 CImgList<T>& assign (const CImg< t1 > & *img1*, const CImg< t2 > & *img2*, const CImg< t3 > & *img3*, const bool *is_shared* = `false`)

Create a list from three images **[in-place version]**.

Parameters

img1	First input image to clone in the constructed list.
img2	Second input image to clone in the constructed list.
img3	Third input image to clone in the constructed list.
is_shared	Flag telling if the elements of the list are shared or non-shared copies of input images.

See also

CImgList(const CImg<t1>&,const CImg<t2>&,const CImg<t3>&,bool).

8.4.4.13 CImgList<T>& assign (const CImg< t1 > & *img1*, const CImg< t2 > & *img2*, const CImg< t3 > & *img3*, const CImg< t4 > & *img4*, const bool *is_shared* = `false`)

Create a list from four images **[in-place version]**.

Parameters

img1	First input image to clone in the constructed list.
img2	Second input image to clone in the constructed list.
img3	Third input image to clone in the constructed list.
img4	Fourth input image to clone in the constructed list.
is_shared	Flag telling if the elements of the list are shared or non-shared copies of input images.

See also

CImgList(const CImg<t1>&,const CImg<t2>&,const CImg<t3>&,const C-Img<t4>&,bool).

8.4.4.14 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`)

Create a list from five images **[in-place version]**.

Parameters

img1	First input image to clone in the constructed list.
img2	Second input image to clone in the constructed list.
img3	Third input image to clone in the constructed list.
img4	Fourth input image to clone in the constructed list.
img5	Fifth input image to clone in the constructed list.
is_shared	Flag telling if the elements of the list are shared or non-shared copies of input images.

See also

CImgList(const CImg<t1>&,const CImg<t2>&,const CImg<t3>&,const C-Img<t4>&,const CImg<t5>&,bool).

8.4.4.15 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`)

Create a list from six images **[in-place version]**.

Parameters

img1	First input image to clone in the constructed list.
img2	Second input image to clone in the constructed list.
img3	Third input image to clone in the constructed list.
img4	Fourth input image to clone in the constructed list.
img5	Fifth input image to clone in the constructed list.
img6	Sixth input image to clone in the constructed list.
is_shared	Flag telling if the elements of the list are shared or non-shared copies of input images.

See also

CImgList(const CImg<t1>&,const CImg<t2>&,const CImg<t3>&,const C-Img<t4>&,const CImg<t5>&,const CImg<t6>&,bool).

8.4.4.16 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`)

Create a list from seven images **[in-place version]**.

Parameters

img1	First input image to clone in the constructed list.
img2	Second input image to clone in the constructed list.
img3	Third input image to clone in the constructed list.
img4	Fourth input image to clone in the constructed list.
img5	Fifth input image to clone in the constructed list.
img6	Sixth input image to clone in the constructed list.
img7	Seventh input image to clone in the constructed list.
is_shared	Flag telling if the elements of the list are shared or non-shared copies of input images.

See also

CImgList(const CImg<t1>&,const CImg<t2>&,const CImg<t3>&,const C-Img<t4>&,const CImg<t5>&,const CImg<t6>&,const CImg<t7>&,bool).

8.4.4.17 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`)

Create a list from eight images **[in-place version]**.

Parameters

img1	First input image to clone in the constructed list.
img2	Second input image to clone in the constructed list.
img3	Third input image to clone in the constructed list.
img4	Fourth input image to clone in the constructed list.
img5	Fifth input image to clone in the constructed list.
img6	Sixth input image to clone in the constructed list.
img7	Seventh input image to clone in the constructed list.
img8	Eighth input image to clone in the constructed list.
is_shared	Flag telling if the elements of the list are shared or non-shared copies of input images.

See also

CImgList(const CImg<t1>&,const CImg<t2>&,const CImg<t3>&,const CImg<t4>&,const CImg<t5>&,const CImg<t6>&,const CImg<t7>&,const CImg<t8>&,bool).

8.4.4.18 CImgList< T >& assign (const CImgList< t > & *list*, const bool *is_shared* = `false`)

Create a list as a copy of an existing list and force the shared state of the list elements **[in-place version]**.

Parameters

list	Input list to copy.
is_shared	Flag telling if the elements of the list are shared or non-shared copies of input images.

See also

CImgList(const CImgList<t>&,bool).

8.4.4.19 CImgList< T >& assign (const char *const *filename*)

Create a list from the content of a file **[in-place version]**.

Parameters

filename	Filename.

See also

CImgList(const char ∗const), load(const char ∗const).

8.4.4.20 CImgList<T>& assign (const CImgDisplay & *disp*)

Create a list from the content of a display window **[in-place version]**.

Parameters

disp	Display window to get content from.

See also

CImgList(const CImgDisplay&).

8.4.4.21 CImgList<t>& move_to (CImgList< t > & *list*)

Transfer the content of the list instance into another list.

Parameters

list	Destination list instance.

Note

When returning, the list instance is empty and the initial content of `list` is destroyed.

See also

move_to(CImgList<t>&,unsigned int).

8.4.4.22 CImgList<t>& move_to (CImgList< t > & *list,* const unsigned int *pos*)

Transfer the content of the list instance at a specified position in another list.

Parameters

list	Destination list instance.

Note

When returning, the list instance is empty and the initial content of `list` is preserved (only images indexes may be modified).

See also

move_to(CImgList<t>&).

8.4.4.23 CImgList<T>& swap (CImgList< T > & *list*)

Swap all fields between list instance and another list.

Parameters

list	List to swap fields with.

Note

Can be used to exchange the content of two lists in a fast way.

See also

CImg<T>::swap(CImg<T>&).

8.4.4.24 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());
```

8.4.4.25 CImg<T>& operator() (const unsigned int *pos*)

Return a reference to the `pos-th` image of the list instance.

Parameters

pos	Indice of the image element to return.

See also

operator[](unsigned int)

8.4.4.26 const CImg<T>& operator() (const unsigned int *pos*) const

Return a reference to the `pos-th` image of the list.

Parameters

pos	Indice of the image element to return.

See also

operator[](unsigned int) const

8.4.4.27 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 the (x,y,z,c) pixel value of the `pos-th` image of the list.

Parameters

pos	Indice of the image element to access.
x	X-coordinate of the pixel location.
y	Y-coordinate of the pixel location.
z	Z-coordinate of the pixel location.
c	C-coordinate of the pixel location.

Note

`list(n,x,y,z,c)` is equivalent to `list[n](x,y,z,c)`.

See also

operator()(unsigned int).

8.4.4.28 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>`.

8.4.4.29 CImgList<T>& operator= (const CImg< t > & *img*)

Create a list containing one copy of an input image **[in-place version]**.

Parameters

img	Input image to clone in the constructed list.

Note

`list = img;` is equivalent to `list.assign(img);`.

See also

assign(const CImg<t>&).

8.4.4.30 CImgList<T>& operator= (const CImgList< t > & *list*)

Create a list as a copy of an existing list.

Parameters

list	Input list to copy.

Note

`list1 = list2` is equivalent to `list1.assign(list2);`.

8.4.4.31 CImgList<T>& operator= (const char ∗const *filename*)

Create a list from the content of a file **[in-place version]**.

Parameters

filename	Filename.

Note

`list1 = filename` is equivalent to `list1.assign(filename);`.

See also

assign(const char ∗const), load(const char ∗const).

8.4.4.32 CImgList<T>& operator= (const CImgDisplay & *disp*)

Create a list from the content of a display window **[in-place version]**.

Parameters

disp	Display window to get content from.

Note

`list = disp;` is equivalent to `list.assign(disp);`.

See also

assign(const CImgDisplay&).

8.4.4.33 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.

See also

CImgList(const CImgList<T>&,bool).

8.4.4.34 CImgList<T>& operator, (const CImg< t > & *img*)

Return a copy of the list instance, where image `img` has been inserted at the end.

Parameters

img	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");
```

See also

CImg<T>::operator,(const CImg<t>&).

8.4.4.35 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

list	List inserted at the end of the instance copy.

See also

CImg<T>::operator,(const CImg<t>&).

8.4.4.36 CImg<T> operator> (const char *axis*) const

Return image corresponding to the concatenation of all images of the instance list along specified axis.

Parameters

axis	Axis used for image concatenation.

Note

`list>'x'` is equivalent to `list.get_append('x')`.

See also

get_append().

8.4.4.37 CImgList<T> operator< (const char *axis*) const

Return list corresponding to the splitting of all images of the instance list along specified axis.

Parameters

axis	Axis used for image splitting.

Note

`list<'x'` is equivalent to `list.get_split('x')`.

See also

get_split().

8.4.4.38 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 may contain spaces (as in `"unsigned char"`).
- If the pixel type `T` does not correspond to a registered type, the string `"unknown"` is returned.

See also

value_type.

8.4.4.39 int width ( ) const

Return the size of the list, i.e. the number of images contained in it.

Note

Similar to size() but returns result as a (signed) integer.

See also

size().

8.4.4.40 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 result as an unsigned integer.

See also

width().

8.4.4.41 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>`.

8.4.4.42 CImg<T>* data (const unsigned int *pos*)

Return pointer to the pos-th image of the list.

Parameters

pos	Indice of the image element to access.

Note

`list.data(n);` is equivalent to `list.data + n;`.

See also

data().

8.4.4.43 iterator begin ( )

Return iterator to the first image of the list.

See also

end().

8.4.4.44 iterator end ( )

Return iterator to one position after the last image of the list.

See also

begin().

8.4.4.45 CImg<T>& front ( )

Return reference to the first image of the list.

See also

back().

8.4.4.46 const CImg<T>& back ( ) const

Return a reference to the last image of the list.

See also

front().

8.4.4.47 CImg<T>& at (const int *pos*)

Return pos-th image of the list.

Parameters

pos	Indice of the image element to access.

8.4.4.48 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

pos	Indice of the image element to access.
x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	C-coordinate of the pixel value.
out_value	Default value returned if `offset` is outside image bounds.

Note

`list.atNXYZC(p,x,y,z,c);` is equivalent to `list[p].atXYZC(x,y,z,c);`.

See also

CImg<T>::atXYZC().

8.4.4.49 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

pos	Indice of the image element to access.
x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	C-coordinate of the pixel value.

Note

`list.atNXYZC(p,x,y,z,c);` is equivalent to `list[p].atXYZC(x,y,z,c);`.

See also

CImg<T>::atXYZC().

8.4.4.50 T& atNXYZ (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 three first coordinates (`pos`, `x`,`y`,`z`).

Parameters

pos	Indice of the image element to access.
x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	C-coordinate of the pixel value.
out_value	Default value returned if `offset` is outside image bounds.

Note

`list.atNXYZ(p,x,y,z,c);` is equivalent to `list[p].atXYZ(x,y,z,c);`.

See also

CImg<T>::atXYZ().

8.4.4.51 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 four first coordinates (`pos`, `x`,`y`,`z`).

Parameters

pos	Indice of the image element to access.
x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	C-coordinate of the pixel value.

Note

`list.atNXYZ(p,x,y,z,c);` is equivalent to `list[p].atXYZ(x,y,z,c);`.

See also

CImg<T>::atXYZ().

8.4.4.52 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 three first coordinates (`pos`, `x`,`y`).

Parameters

pos	Indice of the image element to access.
x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	C-coordinate of the pixel value.
out_value	Default value returned if `offset` is outside image bounds.

Note

`list.atNXYZ(p,x,y,z,c);` is equivalent to `list[p].atXYZ(x,y,z,c);`.

See also

CImg<T>::atXYZ().

8.4.4.53 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 three first coordinates (`pos`, `x`,`y`).

Parameters

pos	Indice of the image element to access.
x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	C-coordinate of the pixel value.
out_value	Default value returned if `offset` is outside image bounds.

Note

`list.atNXYZ(p,x,y,z,c);` is equivalent to `list[p].atXYZ(x,y,z,c);`.

See also

CImg<T>::atXYZ().

8.4.4.54 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 two first coordinates (`pos`,`x`).

Parameters

pos	Indice of the image element to access.
x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	C-coordinate of the pixel value.
out_value	Default value returned if `offset` is outside image bounds.

Note

`list.atNXYZ(p,x,y,z,c);` is equivalent to `list[p].atXYZ(x,y,z,c);`.

See also

CImg<T>::atXYZ().

8.4.4.55 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 two first coordinates (`pos`, `x`).

Parameters

pos	Indice of the image element to access.
x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	C-coordinate of the pixel value.

Note

`list.atNXYZ(p,x,y,z,c);` is equivalent to `list[p].atXYZ(x,y,z,c);`.

See also

CImg<T>::atXYZ().

8.4.4.56 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 first coordinates (`pos`).

Parameters

pos	Indice of the image element to access.
x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	C-coordinate of the pixel value.
out_value	Default value returned if `offset` is outside image bounds.

Note

`list.atNXYZ(p,x,y,z,c);` is equivalent to `list[p].atXYZ(x,y,z,c);`.

See also

CImg<T>::atXYZ().

8.4.4.57 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 first coordinates (`pos`).

Parameters

pos	Indice of the image element to access.
x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	C-coordinate of the pixel value.

Note

`list.atNXYZ(p,x,y,z,c);` is equivalent to `list[p].atXYZ(x,y,z,c);`.

See also

CImg<T>::atXYZ().

8.4.4.58 CImg<charT> value_string (const char *separator* = ',', const unsigned int *max_size* = 0) const

Return a C-string containing the values of all images in the instance list.

Parameters

separator	Character separator set between consecutive pixel values.
max_size	Maximum size of the returned string.

Note

The result is returne as a `CImg<char>` image whose pixel buffer contains the desired C-string.

8.4.4.59 bool is_empty ( ) const

Return `true` if list is empty.

See also

operator bool().

8.4.4.60 bool is_sameXYZC (const unsigned int *dx*, const unsigned int *dy*, const unsigned int *dz*, const unsigned int *dc*) const

Return `true` if dimensions of each image of the list match specified arguments.

Parameters

dx	Checked image width.
dy	Checked image height.
dz	Checked image depth.
dc	Checked image spectrum.

See also

is_sameNXYZC().

8.4.4.61 bool is_sameNXYZC (const unsigned int *n*, const unsigned int *dx*, const unsigned int *dy*, const unsigned int *dz*, const unsigned int *dc*) const

Return `true` if list dimensions match specified arguments.

Parameters

n	Number of images in the list.
dx	Checked image width.
dy	Checked image height.
dz	Checked image depth.
dc	Checked image spectrum.

See also

is_sameXYZC().

8.4.4.62 bool containsNXYZC (const int *n*, const int *x* = 0, const int *y* = 0, const int *z* = 0, const int *c* = 0) const

Return `true` if list contains pixel located at (`n`,`x`,`y`,`z`,`c`).

Parameters

n	Index of the image whom checked pixel value belong to.
x	X-coordinate of the checked pixel value.
y	Y-coordinate of the checked pixel value.
z	Z-coordinate of the checked pixel value.
c	C-coordinate of the checked pixel value.

See also

contains(), CImg<T>::containsXYZC().

8.4.4.63 bool containsN (const int *n*) const

Return `true` if list contains image with index [`n`].

Parameters

n	Index of the checked image.

See also

contains().

8.4.4.64 bool contains (const T & *pixel*, t & *n*, t & *x*, t & *y*, t & *z*, t & *c*) const

Return `true` if one image of the list contains the specified referenced value.

Parameters

	pixel	Reference to pixel value to test.
`out`	*x*	Index of image containing the pixel value, if test succeeds.
`out`	*x*	X-coordinate of the pixel value, if test succeeds.
`out`	*y*	Y-coordinate of the pixel value, if test succeeds.
`out`	*z*	Z-coordinate of the pixel value, if test succeeds.
`out`	*c*	C-coordinate of the pixel value, if test succeeds.

Note

If true, set coordinates (n,x,y,z,c).

8.4.4.65 bool contains (const T & *pixel,* t & *n,* t & *x,* t & *y,* t & *z*) const

Return `true` if one of the image list contains the specified referenced value.

Parameters

	pixel	Reference to pixel value to test.
`out`	*x*	Index of image containing the pixel value, if test succeeds.
`out`	*x*	X-coordinate of the pixel value, if test succeeds.
`out`	*y*	Y-coordinate of the pixel value, if test succeeds.
`out`	*z*	Z-coordinate of the pixel value, if test succeeds.

Note

If true, set coordinates (n,x,y,z).

8.4.4.66 bool contains (const T & *pixel,* t & *n,* t & *x,* t & *y*) const

Return `true` if one of the image list contains the specified referenced value.

Parameters

	pixel	Reference to pixel value to test.
`out`	*x*	Index of image containing the pixel value, if test succeeds.
`out`	*x*	X-coordinate of the pixel value, if test succeeds.
`out`	*y*	Y-coordinate of the pixel value, if test succeeds.

Note

If true, set coordinates (n,x,y).

8.4.4.67 bool contains (const T & *pixel,* t & *n,* t & *x*) const

Return `true` if one of the image list contains the specified referenced value.

Parameters

	pixel	Reference to pixel value to test.
`out`	*x*	Index of image containing the pixel value, if test succeeds.
`out`	*x*	X-coordinate of the pixel value, if test succeeds.

Note

If true, set coordinates (n,x).

8.4.4.68 bool contains (const T & *pixel*, t & *n*) const

Return `true` if one of the image list contains the specified referenced value.

Parameters

	pixel	Reference to pixel value to test.
`out`	*x*	Index of image containing the pixel value, if test succeeds.

Note

If true, set coordinates (n).

8.4.4.69 bool contains (const T & *pixel*) const

Return `true` if one of the image list contains the specified referenced value.

Parameters

pixel	Reference to pixel value to test.

8.4.4.70 bool contains (const CImg< T > & *img*, t & *n*) const

Return `true` if the list contains the image 'img'.

Parameters

	img	Reference to image to test.
`out`	*n*	Index of image in the list, if test succeeds.

Note

If true, returns the position (n) of the image in the list.

8.4.4.71 bool contains (const CImg< T > & *img*) const

Return `true` if the list contains the image img.

Parameters

img	Reference to image to test.

8.4.4.72 T& min ( )

Return a reference to the minimum pixel value of the instance list.

See also

max().

8.4.4.73 T& max ( )

Return a reference to the maximum pixel value of the instance list.

See also

min().

8.4.4.74 T& min_max (t & *max_val*)

Return a reference to the minimum pixel value of the instance list and return the maximum vvalue as well.

Parameters

out	*max_val*	Value of the maximum value found.

See also

max_min().

8.4.4.75 const T& min_max (t & *max_val*) const

Return a reference to the minimum pixel value of the instance list and return the maximum vvalue as well **[const version]**.

Parameters

out	*max_val*	Value of the maximum value found.

See also

max_min().

8.4.4.76 T& max_min (t & *min_val*)

Return a reference to the minimum pixel value of the instance list and return the minimum value as well.

Parameters

`out`	*min_val*	Value of the minimum value found.

See also

min_max().

8.4.4.77 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

img	Image to insert a copy to the list.
pos	Index of the insertion.
is_shared	Flag telling if the inserted image is a shared copy of `img` or not.

8.4.4.78 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

n	Number of empty images to insert.
pos	Index of the insertion.

8.4.4.79 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

n	Number of image copies to insert.
img	Image to insert by copy.

pos	Index of the insertion.
is_shared	Flag telling if inserted images are shared copies of `img` or not.

8.4.4.80 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

list	Image list to insert.
pos	Index of the insertion.
is_shared	Flag telling if inserted images are shared copies of images of `list` or not.

8.4.4.81 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

n	Number of list copies to insert.
list	Image list to insert.
pos	Index of the insertion.
is_shared	Flag telling if inserted images are shared copies of images of `list` or not.

8.4.4.82 CImgList<T>& remove (const unsigned int *pos1*, const unsigned int *pos2*)

Remove all images between from indexes.

Parameters

pos1	Starting index of the removal.
pos2	Ending index of the removal.

8.4.4.83 CImgList<T>& remove (const unsigned int *pos*)

Remove image at index `pos` from the image list.

Parameters

pos	Index of the image to remove.

8.4.4.84 CImgList<T>& remove ( )

Remove last image.

See also

remove().

8.4.4.85 CImgList<T>& images (const unsigned int *pos0*, const unsigned int *pos1*)

Return a sublist.

Parameters

pos0	Starting index of the sublist.
pos1	Ending index of the sublist.

8.4.4.86 CImgList<T> get_shared_images (const unsigned int *pos0*, const unsigned int *pos1*)

Return a shared sublist.

Parameters

pos0	Starting index of the sublist.
pos1	Ending index of the sublist.

8.4.4.87 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

axis	: specify the axis for image concatenation. Can be 'x','y','z' or 'c'.
align	: specify the alignment for image concatenation. Can be '0' (top), '0.5' (center) or '1' (bottom) for instance.

Returns

A CImg<T> image corresponding to the concatenation is returned.

8.4.4.88 CImgList<T>& split (const char *axis*, const int *nb* = 0)

Return a list where each image has been split along the specified axis.

Parameters

axis	Axis to split images along.
nb	Number of spliting parts for each image.

8.4.4.89 CImgList<T>& push_back (const CImg< t > & *img*)

Insert image at the end of the list.

Parameters

img	Image to insert.

8.4.4.90 CImgList<T>& push_front (const CImg< t > & *img*)

Insert image at the front of the list.

Parameters

img	Image to insert.

8.4.4.91 CImgList<T>& push_back (const CImgList< t > & *list*)

Insert list at the end of the current list.

Parameters

list	List to insert.

8.4.4.92 CImgList<T>& push_front (const CImgList< t > & *list*)

Insert list at the front of the current list.

Parameters

list	List to insert.

8.4.4.93 CImgList<T>& pop_back ( )

Remove last image.

See also

remove().

8.4.4.94 CImgList<T>& pop_front ( )

Remove first image.

See also

remove().

8.4.4.95 CImgList<T>& erase (const iterator *iter*)

Remove image pointed by iterator.

Parameters

iter	Iterator pointing to the image to remove.

8.4.4.96 CImg<intT> get_select (CImgDisplay & *disp*, const bool *feature_type* = `true`, const char *axis* = `'x'`, const float *align* = `0`) const

Display a simple interactive interface to select images or sublists.

Parameters

disp	Window instance to display selection and user interface.
feature_type	Can be `false` to select a single image, or `true` to select a sublist.
axis	Axis along whom images are appended for visualization.
align	Alignment setting when images have not all the same size.

Returns

A one-column vector containing the selected image indexes.

8.4.4.97 CImg<intT> get_select (const char *const *title*, const bool *feature_type* = `true`, const char *axis* = `'x'`, const float *align* = `0`) const

Display a simple interactive interface to select images or sublists.

Parameters

title	Title of a new window used to display selection and user interface.
feature_type	Can be `false` to select a single image, or `true` to select a sublist.
axis	Axis along whom images are appended for visualization.
align	Alignment setting when images have not all the same size.

Returns

A one-column vector containing the selected image indexes.

8.4.4.98 CImgList< T >& load (const char *const *filename*)

Load a list from a file.

Parameters

filename	Filename to read data from.

See also

save(const char *).

8.4.4.99 CImgList< T >& load_cimg (const char *const *filename*)

Load a list from a .cimg file.

Parameters

filename	Filename to read data from.

8.4.4.100 CImgList< T >& load_cimg (std::FILE *const *file*)

Load a list from a .cimg file.

Parameters

file	File to read data from.

8.4.4.101 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

filename	Filename to read data from.
n0	Starting index of images to read.
n1	Ending index of images to read.
x0	Starting X-coordinates of image regions to read.
y0	Starting Y-coordinates of image regions to read.

z0	Starting Z-coordinates of image regions to read.
c0	Starting C-coordinates of image regions to read.
x1	Ending X-coordinates of image regions to read.
y1	Ending Y-coordinates of image regions to read.
z1	Ending Z-coordinates of image regions to read.
c1	Ending C-coordinates of image regions to read.

8.4.4.102 CImgList<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*)

Load a sub-image list from a (non compressed) .cimg file.

Parameters

file	File to read data from.
n0	Starting index of images to read.
n1	Ending index of images to read.
x0	Starting X-coordinates of image regions to read.
y0	Starting Y-coordinates of image regions to read.
z0	Starting Z-coordinates of image regions to read.
c0	Starting C-coordinates of image regions to read.
x1	Ending X-coordinates of image regions to read.
y1	Ending Y-coordinates of image regions to read.
z1	Ending Z-coordinates of image regions to read.
c1	Ending C-coordinates of image regions to read.

8.4.4.103 CImgList<T>& load_parrec (const char ∗const *filename*)

Load a list from a PAR/REC (Philips) file.

Parameters

filename	Filename to read data from.

8.4.4.104 CImgList<T>& load_yuv (const char ∗const *filename*, const unsigned int *size_x*, const unsigned int *size_y*, 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

filename	Filename to read data from.
size_x	Width of the images.
size_y	Height of the images.
first_frame	Index of first image frame to read.
last_frame	Index of last image frame to read.
step_frame	Step applied between each frame.
yuv2rgb	Apply YUV to RGB transformation during reading.

8.4.4.105 CImgList<T>& load_yuv (std::FILE *const *file*, const unsigned int *size_x*, const unsigned int *size_y*, 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.

Parameters

filename	File to read data from.
size_x	Width of the images.
size_y	Height of the images.
first_frame	Index of first image frame to read.
last_frame	Index of last image frame to read.
step_frame	Step applied between each frame.
yuv2rgb	Apply YUV to RGB transformation during reading.

8.4.4.106 CImgList<T>& load_ffmpeg (const char *const *filename*, const unsigned int *first_frame* = `0`, const unsigned int *last_frame* = `~0U`, const unsigned int *step_frame* = `1`, const bool *pixel_format* = `true`, const bool *resume* = `false`)

Load an image from a video file, using ffmpeg libraries.

Parameters

filename	Filename to read data from.
first_frame	Index of first image frame to read.
last_frame	Index of last image frame to read.
step_frame	Step applied between each frame.

8.4.4.107 CImgList<T>& load_ffmpeg_external (const char *const *filename*)

Load an image from a video file using the external tool 'ffmpeg'.

Parameters

filename	Filename to read data from.

8.4.4.108 CImgList<T>& load_gzip_external (const char *const *filename*)

Load a gzipped list, using external tool 'gunzip'.

Parameters

filename	Filename to read data from.

8.4.4.109 CImgList<T>& load_off (const char *const *filename*, CImgList< tf > & *primitives*, CImgList< tc > & *colors*)

Load a 3d object from a .OFF file.

Parameters

	filename	Filename to read data from.
`out`	*primitives*	At return, contains the list of 3d object primitives.
`out`	*colors*	At return, contains the list of 3d object colors.

Returns

List of 3d object vertices.

8.4.4.110 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`)

Load a multi-page TIFF file.

Parameters

filename	Filename to read data from.
first_frame	Index of first image frame to read.
last_frame	Index of last image frame to read.
step_frame	Step applied between each frame.

8.4.4.111 const CImgList<T>& print (const char *const *title* = `0`, const bool *display_stats* = `true`) const

Print informations about the list on the standard output.

Parameters

title	Label set to the informations displayed.
display_-stats	Enable display of statistics.

8.4.4.112 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

disp	Reference to an existing CImgDisplay instance, where the current image list will be displayed.
axis	Specify the axis for image concatenation. Can be 'x','y','z' or 'c'.
align	Specify the alignment for image concatenation.

Returns

A reference to the current CImgList instance is returned.

Note

This function displays the list images of the current CImgList instance into an existing CImgDisplay window. Images of the list are concatenated in a single temporarly image for visualization purposes. The function returns immediately.

8.4.4.113 const CImgList<T>& display (CImgDisplay & *disp*, const bool *display_info*, const char *axis* = 'x', const float *align* = 0) const

Display the current CImgList instance in a new display window.

Parameters

title	Title of the opening display window.
axis	Axis for image concatenation. Can be 'x','y','z' or 'c'.
align	Alignment for image concatenation.

Returns

A reference to the current CImgList instance is returned.

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 concatenated in a single temporarly image for visualization purposes. The function returns when a key is pressed or the display window is closed by the user.

8.4.4.114 const CImgList<T>& display (const char *const *title* = 0, const bool *display_info* = true, const char *axis* = 'x', const float *align* = 0) const

Display the current CImgList instance in a new display window.

Parameters

title	Title of the opening display window.
display_info	Flag telling if list informations must be written on standard output.
axis	Axis for image concatenation. Can be 'x','y','z' or 'c'.
Alignment	for image concatenation.

8.4.4.115 const CImgList<T>& save (const char *const *filename,* const int *number =* `-1`) const

Save a list into a file.

Parameters

filename	Filename to write data to.
number	Number of digits used when chosen format requires the saving of multiple files.

8.4.4.116 const CImgList<T>& save_ffmpeg (const char *const *filename,* const unsigned int *first_frame =* `0`*,* const unsigned int *last_frame =* `~0U`*,* const unsigned int *fps =* `25`*,* const unsigned int *bitrate =* `2048`) const

Save an image sequence, using FFMPEG library.

Parameters

filename	Filename to write data to.
first_frame	Index of first image frame to write.
last_frame	Index of last image frame to write.
fps	Desired framerate (in frames per seconds) if chosen format supports it.
bitrate	Desired bitrate (in bits per seconds) if chosen format supports it.

8.4.4.117 const CImgList<T>& save_yuv (const char *const *filename =* `0`*,* const bool *rgb2yuv =* `true`) const

Save list as a YUV image sequence file.

Parameters

filename	Filename to write data to.
rgb2yuv	Flag telling if the RGB to YUV conversion must be done for saving.

8.4.4.118 const CImgList<T>& save_yuv (std::FILE *const *file*, const bool *rgb2yuv* = `true`) const

Save an image sequence into a YUV file.

Parameters

file	File to write data to.
rgb2yuv	Flag telling if the RGB to YUV conversion must be done for saving.

8.4.4.119 const CImgList<T>& save_cimg (const char *const *filename*, const bool *compress* = `false`) const

Save a list into a .cimg file.

Parameters

filename	Filename to write data to.
compress	Flag telling if data compression must be enabled.

8.4.4.120 const CImgList<T>& save_cimg (std::FILE * *file*, const bool *compress* = `false`) const

Save a list into a .cimg file.

Parameters

file	File to write data to.
compress	Flag telling if data compression must be enabled.

8.4.4.121 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 instance into into an existing .cimg file, at specified coordinates.

Parameters

filename	Filename to write data to.
n0	Starting index of images to write.
n1	Ending index of images to write.
x0	Starting X-coordinates of image regions to write.
y0	Starting Y-coordinates of image regions to write.
z0	Starting Z-coordinates of image regions to write.
c0	Starting C-coordinates of image regions to write.

8.4.4.122 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 instance into into an existing .cimg file, at specified coordinates.

Parameters

file	File to write data to.
n0	Starting index of images to write.
n1	Ending index of images to write.
x0	Starting X-coordinates of image regions to write.
y0	Starting Y-coordinates of image regions to write.
z0	Starting Z-coordinates of image regions to write.
c0	Starting C-coordinates of image regions to write.

8.4.4.123 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]`

Create an empty (non-compressed) .cimg file with specified dimensions.

Parameters

filename	Filename to write data to.
nb	Number of images to write.
dx	Width of images in the written file.
dy	Height of images in the written file.
dz	Depth of images in the written file.
dc	Spectrum of images in the written file.

8.4.4.124 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]`

Create an empty .cimg file with specified dimensions.

Parameters

file	File to write data to.
nb	Number of images to write.
dx	Width of images in the written file.
dy	Height of images in the written file.
dz	Depth of images in the written file.
dc	Spectrum of images in the written file.

8.4.4.125 const CImgList< T >& save_tiff (const char *const *filename*, const unsigned int *compression* = 0) const

Save a file in TIFF format.

Parameters

filename	Filename to write data to.
compression	Compression mode used to write data.

8.4.4.126 const CImgList< T >& save_gzip_external (const char *const *filename*) const

Save a list as a gzipped file, using external tool 'gzip'.

Parameters

filename	Filename to write data to.

8.4.4.127 const CImgList< T >& save_ffmpeg_external (const char *const *filename*, const unsigned int *first_frame* = 0, const unsigned int *last_frame* = ~0U, const char *const *codec* = 0, const unsigned int *fps* = 25, const unsigned int *bitrate* = 2048) const

Save an image sequence using the external tool 'ffmpeg'.

Parameters

filename	Filename to write data to.
first_frame	Index of first image frame to write.
last_frame	Index of last image frame to write.

8.4.4.128 CImgList< T >& crop_font ( )

Create an auto-cropped font (along the X axis) from a input font.

See also

get_crop_font().

8.4.4.129 CImgList< T > get_crop_font ( ) const

Create an auto-cropped font (along the X axis) from a input font **[new-instance version]**.

See also

crop_font().

8.4.4.130 static const CImgList<T>& font (const unsigned int *font_height*, const bool *is_variable_width* = `true`) [static]

Return a CImg pre-defined font with desired size.

Parameters

font_height	Height of the desired font (exact match for 11,13,17,19,24,32,38,57)
is_variable_-width	Decide if the font has a variable (`true`) or fixed (`false`) width.

8.4.4.131 CImgList<T>& FFT (const char *axis*, const bool *invert* = `false`)

Compute a 1d Fast Fourier Transform, along specified axis.

Parameters

axis	Axis along which the Fourier transform is computed.
invert	Flag telling if the direct (`false`) or inverse transform (`true`) is computed.

8.4.4.132 CImgList<T>& FFT (const bool *invert* = `false`)

Compute a n-d Fast Fourier Transform.

Parameters

invert	Flag telling if the direct (`false`) or inverse transform (`true`) is computed.

8.4.4.133 CImgList<T>& reverse_object3d ( )

Reverse primitives orientations of a 3d object.

See also

get_reverse_object3d().