## Contents

1 Main Page .................................................. 1

2 Module Index ............................................. 3
   2.1 Modules ............................................. 3

3 Namespace Index ........................................... 5
   3.1 Namespace List ....................................... 5

4 Hierarchical Index ....................................... 7
   4.1 Class Hierarchy ....................................... 7

5 Class Index ................................................ 9
   5.1 Class List ............................................ 9

6 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 ? ................................. 12
      6.1.4 What's next ? .................................... 13
   6.2 FAQ: Frequently Asked Questions. ................... 14
      6.2.1 FAQ Summary ..................................... 14
      6.2.2 1. General information and availability .......... 14
         6.2.2.1 1.1. What is the CImg Library ? ............ 14
         6.2.2.2 1.2. What platforms are supported ? ......... 14
         6.2.2.3 1.3. How is CImg distributed ? .............. 15
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2.2.4</td>
<td>1.4. What kind of people are concerned by CImg?</td>
<td>15</td>
</tr>
<tr>
<td>6.2.2.5</td>
<td>1.5. What are the specificities of the CeCILL license?</td>
<td>15</td>
</tr>
<tr>
<td>6.2.2.6</td>
<td>1.6. Who is behind CImg?</td>
<td>15</td>
</tr>
<tr>
<td>6.2.3</td>
<td>2. C++ related questions</td>
<td>16</td>
</tr>
<tr>
<td>6.2.3.1</td>
<td>2.1 What is the level of C++ knowledge needed to use CImg?</td>
<td>16</td>
</tr>
<tr>
<td>6.2.3.2</td>
<td>2.2 How to use CImg in my own C++ program?</td>
<td>16</td>
</tr>
<tr>
<td>6.2.3.3</td>
<td>2.3 Why is CImg entirely contained in a single header file?</td>
<td>16</td>
</tr>
<tr>
<td>6.2.4</td>
<td>3. Other resources</td>
<td>17</td>
</tr>
<tr>
<td>6.2.4.1</td>
<td>3.1 Translations</td>
<td>17</td>
</tr>
<tr>
<td>6.3</td>
<td>Setting Environment Variables</td>
<td>18</td>
</tr>
<tr>
<td>6.4</td>
<td>How to use CImg library with Visual C++ 2005 Express Edition?</td>
<td>19</td>
</tr>
<tr>
<td>6.4.1</td>
<td>How to use CImg library with Visual C++ 2005 Express Edition?</td>
<td>19</td>
</tr>
<tr>
<td>6.5</td>
<td>Tutorial: Getting Started</td>
<td>20</td>
</tr>
<tr>
<td>6.6</td>
<td>Using Image Loops</td>
<td>22</td>
</tr>
<tr>
<td>6.6.1</td>
<td>Loops over the pixel buffer</td>
<td>22</td>
</tr>
<tr>
<td>6.6.2</td>
<td>Loops over image dimensions</td>
<td>23</td>
</tr>
<tr>
<td>6.6.3</td>
<td>Loops over interior regions and borders</td>
<td>24</td>
</tr>
<tr>
<td>6.6.4</td>
<td>Loops using neighborhoods</td>
<td>24</td>
</tr>
<tr>
<td>6.6.4.1</td>
<td>Neighborhood-based loops for 2D images</td>
<td>25</td>
</tr>
<tr>
<td>6.6.4.2</td>
<td>Neighborhood-based loops for 3D images</td>
<td>25</td>
</tr>
<tr>
<td>6.6.4.3</td>
<td>Defining neighborhoods</td>
<td>25</td>
</tr>
<tr>
<td>6.6.4.4</td>
<td>Using alternate variable names</td>
<td>26</td>
</tr>
<tr>
<td>6.6.4.5</td>
<td>Example codes</td>
<td>27</td>
</tr>
<tr>
<td>6.7</td>
<td>Using Display Windows</td>
<td>28</td>
</tr>
<tr>
<td>6.8</td>
<td>How pixel data are stored with CImg</td>
<td>29</td>
</tr>
<tr>
<td>6.9</td>
<td>Files IO in CImg</td>
<td>30</td>
</tr>
<tr>
<td>6.10</td>
<td>Retrieving Command Line Arguments</td>
<td>31</td>
</tr>
<tr>
<td>6.10.1</td>
<td>The cimg_usage() macro</td>
<td>31</td>
</tr>
<tr>
<td>6.10.2</td>
<td>The cimg_help() macro</td>
<td>31</td>
</tr>
<tr>
<td>6.10.3</td>
<td>The cimg_option() macro</td>
<td>31</td>
</tr>
<tr>
<td>6.10.4</td>
<td>Example of use</td>
<td>32</td>
</tr>
<tr>
<td>6.10.5</td>
<td>How to learn more about command line options?</td>
<td>32</td>
</tr>
</tbody>
</table>
7 Namespace Documentation

7.1 cimg_library Namespace Reference ........................................... 33

7.1.1 Detailed Description ............................................................... 33

7.2 cimg_library::cimg Namespace Reference ....................................... 34

7.2.1 Detailed Description ............................................................... 42

7.2.2 Function Documentation ......................................................... 42

7.2.2.1 output() ............................................................... 42

7.2.2.2 info() ............................................................... 43

7.2.2.3 exception_mode() [1/2] ..................................................... 43

7.2.2.4 exception_mode() [2/2] ..................................................... 43

7.2.2.5 openmp_mode() ............................................................. 44

7.2.2.6 eval() ............................................................... 44

7.2.2.7 warn() ............................................................... 45

7.2.2.8 system() ............................................................... 45

7.2.2.9 endianness() ............................................................... 46

7.2.2.10 invert_endianness() [1/2] .................................................. 46

7.2.2.11 invert_endianness() [2/2] .................................................. 46

7.2.2.12 time() ............................................................... 46

7.2.2.13 tic() ............................................................... 47

7.2.2.14 toc() ............................................................... 47

7.2.2.15 sleep() ............................................................... 47

7.2.2.16 wait() ............................................................... 48

7.2.2.17 mod() ............................................................... 48

7.2.2.18 minmod() ............................................................... 48

7.2.2.19 round() ............................................................... 49

7.2.2.20 atof() ............................................................... 49

7.2.2.21 strncasecmp() ............................................................ 50

7.2.2.22 strcasecmp() ............................................................. 50

7.2.2.23 strellipsize() [1/2] ......................................................... 51

7.2.2.24 strellipsize() [2/2] ........................................................ 51
CONTENTS

7.2.2.25 strpare() ................................................................. 51
7.2.2.26 strwindows_reserved() ................................................ 52
7.2.2.27 strunescape() ........................................................... 52
7.2.2.28 fopen() ................................................................. 52
7.2.2.29 fclose() ................................................................. 53
7.2.2.30 is_directory() ........................................................... 53
7.2.2.31 is_file() ................................................................. 53
7.2.2.32 fsize() ................................................................. 54
7.2.2.33 fdate() [1/2] ............................................................. 54
7.2.2.34 fdate() [2/2] ............................................................. 55
7.2.2.35 date() [1/2] ............................................................... 55
7.2.2.36 date() [2/2] ............................................................... 55
7.2.2.37 temporary_path() ....................................................... 56
7.2.2.38 imagemagick_path() .................................................... 56
7.2.2.39 graphicsmagick_path() ................................................ 57
7.2.2.40 medcon_path() .......................................................... 57
7.2.2.41 ffmpeg_path() .......................................................... 58
7.2.2.42 gzip_path() .............................................................. 58
7.2.2.43 gunzip_path() ........................................................... 58
7.2.2.44 dcraw_path() ............................................................ 59
7.2.2.45 wget_path() ............................................................. 59
7.2.2.46 curl_path() .............................................................. 59
7.2.2.47 split_filename() ......................................................... 60
7.2.2.48 fread() ................................................................. 60
7.2.2.49 fwrite() ................................................................. 61
7.2.2.50 fempty() ................................................................. 61
7.2.2.51 ftype() ................................................................. 61
7.2.2.52 load_network() .......................................................... 62
7.2.2.53 files() ................................................................. 62
7.2.2.54 dialog() ................................................................. 63
8 Class Documentation

8.1 CImg< T > Struct Template Reference ........................................... 65

8.1.1 Detailed Description ........................................................................... 121

8.1.2 Member Typedef Documentation .......................................................... 123

8.1.2.1 iterator .............................................................................................. 123

8.1.2.2 const_iterator .................................................................................... 123

8.1.2.3 value_type .......................................................................................... 124

8.1.3 Constructor & Destructor Documentation ............................................. 124

8.1.3.1 ~CImg() ............................................................................................ 124

8.1.3.2 CImg() [1/13] .................................................................................... 124

8.1.3.3 CImg() [2/13] .................................................................................... 125

8.1.3.4 CImg() [3/13] .................................................................................... 125

8.1.3.5 CImg() [4/13] .................................................................................... 126

8.1.3.6 CImg() [5/13] .................................................................................... 127

8.1.3.7 CImg() [6/13] .................................................................................... 128

8.1.3.8 CImg() [7/13] .................................................................................... 129

8.1.3.9 CImg() [8/13] .................................................................................... 129

8.1.3.10 CImg() [9/13] .................................................................................. 130

8.1.3.11 CImg() [10/13] ................................................................................. 131

8.1.3.12 CImg() [11/13] ............................................................................... 131

8.1.3.13 CImg() [12/13] ............................................................................... 132

8.1.3.14 CImg() [13/13] ............................................................................... 132

8.1.4 Member Function Documentation ....................................................... 133

8.1.4.1 assign() [1/13] ................................................................................ 133

8.1.4.2 assign() [2/13] ................................................................................ 133

8.1.4.3 assign() [3/13] ................................................................................ 133

8.1.4.4 assign() [4/13] ................................................................................ 133

8.1.4.5 assign() [5/13] ................................................................................ 134

8.1.4.6 assign() [6/13] ................................................................................ 134

8.1.4.7 assign() [7/13] ................................................................................ 134
8.1.4.38 operator--() [1/2] ................................. 145
8.1.4.39 operator--() [2/2] ................................. 145
8.1.4.40 operator-() [1/4] ................................. 145
8.1.4.41 operator-() [2/4] ................................. 145
8.1.4.43 operator-() [4/4] ................................. 146
8.1.4.44 operator\*() [1/3] ................................. 146
8.1.4.45 operator\*() [2/3] ................................. 146
8.1.4.46 operator\*() [3/3] ................................. 146
8.1.4.47 operator\*() [1/3] ................................. 147
8.1.4.48 operator\*() [2/3] ................................. 147
8.1.4.49 operator\*() [3/3] ................................. 147
8.1.4.50 operator/=() [1/3] ................................. 147
8.1.4.51 operator/=() [2/3] ................................. 148
8.1.4.52 operator/=() [3/3] ................................. 148
8.1.4.53 operator/() [1/3] ................................. 148
8.1.4.54 operator/() [2/3] ................................. 148
8.1.4.55 operator/() [3/3] ................................. 149
8.1.4.56 operator\%=() [1/3] ................................. 149
8.1.4.57 operator\%=() [2/3] ................................. 149
8.1.4.58 operator\%=() [3/3] ................................. 149
8.1.4.59 operator\%=() [1/3] ................................. 149
8.1.4.60 operator\%=() [2/3] ................................. 150
8.1.4.61 operator\%=() [3/3] ................................. 150
8.1.4.62 operator &=() [1/3] ................................. 150
8.1.4.63 operator &=() [2/3] ................................. 150
8.1.4.64 operator &=() [3/3] ................................. 150
8.1.4.65 operator &=() [1/3] ................................. 151
8.1.4.66 operator &=() [2/3] ................................. 151
8.1.4.67 operator &=() [3/3] ................................. 151
<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>=()</td>
<td>151</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>151</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>152</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>152</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>152</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>152</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>152</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>152</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>152</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>153</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>153</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>153</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>153</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>154</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>154</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>154</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>154</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>154</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>154</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>155</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>155</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>155</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>155</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>155</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>156</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>156</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>156</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>156</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>156</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>157</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>157</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>157</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>157</td>
</tr>
<tr>
<td>**</td>
<td>=()</td>
<td>158</td>
</tr>
</tbody>
</table>
8.1.4.128 linear_atXY() ........................................ 174
8.1.4.129 linear_atXYZ() ...................................... 174
8.1.4.130 linear_atXYZ() ...................................... 175
8.1.4.131 linear_atXYZC() .................................. 175
8.1.4.132 linear_atXYZC() .................................. 175
8.1.4.133 cubic_atX() ......................................... 176
8.1.4.134 cubic_atX_c() ...................................... 176
8.1.4.135 cubic_atX() ......................................... 177
8.1.4.136 cubic_atX_c() ...................................... 177
8.1.4.137 cubic_atXY() ......................................... 178
8.1.4.138 cubic_atXY_c() .................................. 178
8.1.4.139 cubic_atXY() ......................................... 178
8.1.4.140 cubic_atXY_c() .................................. 179
8.1.4.141 cubic_atXYZ() ...................................... 179
8.1.4.142 cubic_atXYZ_c() .................................. 179
8.1.4.143 cubic_atXYZ() ...................................... 179
8.1.4.144 cubic_atXYZ_c() .................................. 180
8.1.4.145 cubic_atXYZ_p() .................................. 180
8.1.4.146 set_linear_atX() .................................. 180
8.1.4.147 set_linear_atXY() .................................. 181
8.1.4.148 set_linear_atXYZ() .................................. 181
8.1.4.149 value_string() ...................................... 181
8.1.4.150 is_shared() ......................................... 182
8.1.4.151 is_empty() ........................................ 182
8.1.4.152 is_inf() ........................................ 182
8.1.4.153 is_nan() ........................................ 183
8.1.4.154 is_sameXY() ...................................... 183
8.1.4.155 is_sameXY() ...................................... 183
8.1.4.156 is_sameXY() ...................................... 183
8.1.4.157 is_sameXZ() ..................................... 183
<p>| 8.1.4.188 exp() | .................................................. | 192 |
| 8.1.4.189 log() | .................................................. | 192 |
| 8.1.4.190 log2() | .................................................. | 193 |
| 8.1.4.191 log10() | .................................................. | 193 |
| 8.1.4.192 abs() | .................................................. | 193 |
| 8.1.4.193 sign() | .................................................. | 194 |
| 8.1.4.194 cos() | .................................................. | 194 |
| 8.1.4.195 sin() | .................................................. | 194 |
| 8.1.4.196 sinc() | .................................................. | 195 |
| 8.1.4.197 tan() | .................................................. | 195 |
| 8.1.4.198 cosh() | .................................................. | 195 |
| 8.1.4.199 sinh() | .................................................. | 196 |
| 8.1.4.200 tanh() | .................................................. | 196 |
| 8.1.4.201 acos() | .................................................. | 196 |
| 8.1.4.202 asin() | .................................................. | 197 |
| 8.1.4.203 atan() | .................................................. | 197 |
| 8.1.4.204 atan2() | .................................................. | 197 |
| 8.1.4.205 acosh() | .................................................. | 198 |
| 8.1.4.206 asinh() | .................................................. | 198 |
| 8.1.4.207 atanh() | .................................................. | 198 |
| 8.1.4.208 mul() | .................................................. | 198 |
| 8.1.4.209 div() | .................................................. | 199 |
| 8.1.4.210 pow() ([1/3]) | .................................................. | 199 |
| 8.1.4.211 pow() ([2/3]) | .................................................. | 200 |
| 8.1.4.212 pow() ([3/3]) | .................................................. | 200 |
| 8.1.4.213 rol() ([1/3]) | .................................................. | 200 |
| 8.1.4.214 rol() ([2/3]) | .................................................. | 200 |
| 8.1.4.215 rol() ([3/3]) | .................................................. | 200 |
| 8.1.4.216 ror() ([1/3]) | .................................................. | 201 |
| 8.1.4.217 ror() ([2/3]) | .................................................. | 201 |</p>
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1.4.218 ror()</td>
<td>201</td>
</tr>
<tr>
<td>8.1.4.219 min()</td>
<td>201</td>
</tr>
<tr>
<td>8.1.4.220 min()</td>
<td>201</td>
</tr>
<tr>
<td>8.1.4.221 min()</td>
<td>202</td>
</tr>
<tr>
<td>8.1.4.222 max()</td>
<td>202</td>
</tr>
<tr>
<td>8.1.4.223 max()</td>
<td>202</td>
</tr>
<tr>
<td>8.1.4.224 max()</td>
<td>204</td>
</tr>
<tr>
<td>8.1.4.225 min_max()</td>
<td>204</td>
</tr>
<tr>
<td>8.1.4.226 max_min()</td>
<td>204</td>
</tr>
<tr>
<td>8.1.4.227 kth_smallest()</td>
<td>205</td>
</tr>
<tr>
<td>8.1.4.228 variance()</td>
<td>205</td>
</tr>
<tr>
<td>8.1.4.229 variance_mean()</td>
<td>205</td>
</tr>
<tr>
<td>8.1.4.230 variance_noise()</td>
<td>206</td>
</tr>
<tr>
<td>8.1.4.231 MSE()</td>
<td>206</td>
</tr>
<tr>
<td>8.1.4.232 PSNR()</td>
<td>206</td>
</tr>
<tr>
<td>8.1.4.233 eval()</td>
<td>207</td>
</tr>
<tr>
<td>8.1.4.234 eval()</td>
<td>207</td>
</tr>
<tr>
<td>8.1.4.235 eval()</td>
<td>208</td>
</tr>
<tr>
<td>8.1.4.236 magnitude()</td>
<td>208</td>
</tr>
<tr>
<td>8.1.4.237 dot()</td>
<td>209</td>
</tr>
<tr>
<td>8.1.4.238 get_vector_at()</td>
<td>209</td>
</tr>
<tr>
<td>8.1.4.239 get_matrix_at()</td>
<td>209</td>
</tr>
<tr>
<td>8.1.4.240 get_tensor_at()</td>
<td>210</td>
</tr>
<tr>
<td>8.1.4.241 set_vector_at()</td>
<td>210</td>
</tr>
<tr>
<td>8.1.4.242 set_matrix_at()</td>
<td>210</td>
</tr>
<tr>
<td>8.1.4.243 set_tensor_at()</td>
<td>212</td>
</tr>
<tr>
<td>8.1.4.244 vector()</td>
<td>212</td>
</tr>
<tr>
<td>8.1.4.245 diagonal()</td>
<td>212</td>
</tr>
<tr>
<td>8.1.4.246 identity_matrix()</td>
<td>213</td>
</tr>
<tr>
<td>8.1.4.247 sequence()</td>
<td>213</td>
</tr>
<tr>
<td>Function Name</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>transpose()</td>
<td>213</td>
</tr>
<tr>
<td>cross()</td>
<td>213</td>
</tr>
<tr>
<td>invert()</td>
<td>214</td>
</tr>
<tr>
<td>solve()</td>
<td>214</td>
</tr>
<tr>
<td>solve_tridiagonal()</td>
<td>214</td>
</tr>
<tr>
<td>eigen()</td>
<td>215</td>
</tr>
<tr>
<td>get_eigen()</td>
<td>215</td>
</tr>
<tr>
<td>symmetric_eigen()</td>
<td>215</td>
</tr>
<tr>
<td>get_symmetric_eigen()</td>
<td>216</td>
</tr>
<tr>
<td>sort() [1/2]</td>
<td>216</td>
</tr>
<tr>
<td>sort() [2/2]</td>
<td>216</td>
</tr>
<tr>
<td>SVD()</td>
<td>217</td>
</tr>
<tr>
<td>get_SVD()</td>
<td>217</td>
</tr>
<tr>
<td>dijkstra() [1/2]</td>
<td>218</td>
</tr>
<tr>
<td>dijkstra() [2/2]</td>
<td>218</td>
</tr>
<tr>
<td>string()</td>
<td>219</td>
</tr>
<tr>
<td>vector() [2/5]</td>
<td>219</td>
</tr>
<tr>
<td>vector() [3/5]</td>
<td>219</td>
</tr>
<tr>
<td>vector() [4/5]</td>
<td>220</td>
</tr>
<tr>
<td>vector() [5/5]</td>
<td>220</td>
</tr>
<tr>
<td>matrix() [1/3]</td>
<td>220</td>
</tr>
<tr>
<td>matrix() [2/3]</td>
<td>221</td>
</tr>
<tr>
<td>matrix() [3/3]</td>
<td>221</td>
</tr>
<tr>
<td>tensor()</td>
<td>222</td>
</tr>
<tr>
<td>identity_matrix() [2/2]</td>
<td>222</td>
</tr>
<tr>
<td>sequence() [2/2]</td>
<td>222</td>
</tr>
<tr>
<td>rotation_matrix()</td>
<td>223</td>
</tr>
<tr>
<td>fill() [1/4]</td>
<td>223</td>
</tr>
<tr>
<td>fill() [2/4]</td>
<td>223</td>
</tr>
<tr>
<td>fill() [3/4]</td>
<td>224</td>
</tr>
</tbody>
</table>
8.1.4.308 XYZtoRGB() .................................................. 236
8.1.4.309 resize() [1/3] ................................................. 236
8.1.4.310 resize() [2/3] ................................................. 237
8.1.4.311 resize() [3/3] ................................................. 238
8.1.4.312 resize_doubleXY() ........................................... 238
8.1.4.313 resize_tripleXY() ........................................... 239
8.1.4.314 mirror() [1/2] ............................................... 239
8.1.4.315 mirror() [2/2] ............................................... 239
8.1.4.316 shift() .......................................................... 239
8.1.4.317 permute_axes() .............................................. 240
8.1.4.318 unroll() ....................................................... 240
8.1.4.319 rotate() [1/4] ................................................. 240
8.1.4.320 rotate() [2/4] ................................................. 241
8.1.4.323 warp() .......................................................... 244
8.1.4.324 get_projections2d() ....................................... 244
8.1.4.325 crop() .......................................................... 244
8.1.4.326 autocrop() .................................................... 245
8.1.4.327 get_column() ............................................... 245
8.1.4.328 columns() .................................................... 245
8.1.4.329 row() ........................................................... 247
8.1.4.330 get_rows() ................................................... 247
8.1.4.331 get_slice() .................................................. 247
8.1.4.332 get_slices() .................................................. 248
8.1.4.333 get_channel() ............................................... 248
8.1.4.334 get_channels() ............................................. 248
8.1.4.335 streamline() ................................................ 248
8.1.4.336 get_shared_points() ...................................... 249
8.1.4.337 get_shared_rows() ....................................... 250
8.1.4.428 draw_triangle() [7/9] .................................................. 296
8.1.4.429 draw_triangle() [8/9] .................................................. 296
8.1.4.430 draw_triangle() [9/9] .................................................. 297
8.1.4.431 draw_rectangle() [1/3] ................................................. 298
8.1.4.432 draw_rectangle() [2/3] ................................................. 299
8.1.4.433 draw_rectangle() [3/3] ................................................. 299
8.1.4.434 draw_polygon() ........................................................... 300
8.1.4.435 draw_ellipse() [1/4] .................................................... 300
8.1.4.436 draw_ellipse() [2/4] .................................................... 301
8.1.4.437 draw_ellipse() [3/4] .................................................... 301
8.1.4.439 draw_circle() [1/2] ..................................................... 302
8.1.4.440 draw_circle() [2/2] ..................................................... 303
8.1.4.441 draw_image() [1/2] ..................................................... 303
8.1.4.442 draw_image() [2/2] ..................................................... 304
8.1.4.443 draw_text() [1/4] ....................................................... 304
8.1.4.444 draw_text() [2/4] ....................................................... 305
8.1.4.446 draw_text() [4/4] ....................................................... 306
8.1.4.447 draw_quiver() [1/2] .................................................... 306
8.1.4.448 draw_quiver() [2/2] .................................................... 307
8.1.4.449 draw_axis() [1/2] ..................................................... 307
8.1.4.450 draw_axis() [2/2] ..................................................... 308
8.1.4.451 draw_axes() .............................................................. 308
8.1.4.452 draw_grid() .............................................................. 309
8.1.4.453 draw_graph() ............................................................ 309
8.1.4.454 draw_fill() .............................................................. 310
8.1.4.455 draw_plasma() .......................................................... 311
8.1.4.456 draw_mandelbrot() ..................................................... 311
8.1.4.457 draw_gaussian() [1/2] ............................................... 312
8.1.4.458 draw_gaussian() [2/2] .............................................. 313
8.1.4.459 draw_object3d() .................................................. 313
8.1.4.460 select() ................................................................ 314
8.1.4.461 load() .................................................................. 314
8.1.4.462 load_ascii() ............................................................ 315
8.1.4.463 load_dlm() .............................................................. 315
8.1.4.464 load_bmp() .............................................................. 315
8.1.4.465 load_jpeg() .............................................................. 316
8.1.4.466 load_magick() ......................................................... 316
8.1.4.467 load_png() ............................................................... 316
8.1.4.468 load_pnm() .............................................................. 316
8.1.4.469 load_pfm() .............................................................. 318
8.1.4.470 load_rgb() ............................................................... 318
8.1.4.471 load_rgba() ............................................................. 318
8.1.4.472 load_tiff() ............................................................... 319
8.1.4.473 load_minc2() ........................................................... 319
8.1.4.474 load_analyze() ........................................................ 319
8.1.4.475 load_cimg() [1/2] ..................................................... 320
8.1.4.476 load_cimg() [2/2] ..................................................... 320
8.1.4.477 load_inr() ............................................................... 321
8.1.4.478 load_exr() ............................................................... 321
8.1.4.479 load_pandore() ....................................................... 322
8.1.4.480 load_parrec() ........................................................ 322
8.1.4.481 load_raw() ............................................................. 322
8.1.4.482 load_yuv() ............................................................. 323
8.1.4.483 load_off() .............................................................. 323
8.1.4.484 load_video() ........................................................... 324
8.1.4.485 load_ffmpeg_external() ............................................ 324
8.1.4.486 load_gif_external() ............................................... 325
8.1.4.487 load_graphicsmagick_external() ......................... 325
8.1.4.488 load_gzip_external() ........................................... 325
8.1.4.489 load_imagemagick_external() ................................... 325
8.1.4.490 load_medcon_external() ......................................... 326
8.1.4.491 load_dcraw_external() .......................................... 326
8.1.4.492 load_camera() .................................................. 326
8.1.4.493 load_other() .................................................... 327
8.1.4.494 print() .......................................................... 327
8.1.4.495 display() [1/3] .................................................. 327
8.1.4.496 display() [2/3] .................................................. 327
8.1.4.497 display() [3/3] .................................................. 328
8.1.4.498 display_object3d() ............................................. 328
8.1.4.499 display_graph() ................................................ 329
8.1.4.500 save() .......................................................... 330
8.1.4.501 save_ascii() .................................................... 330
8.1.4.502 save_cpp() ...................................................... 331
8.1.4.503 save_dlm() ...................................................... 331
8.1.4.504 save_bmp() ...................................................... 331
8.1.4.505 save_jpeg() ...................................................... 331
8.1.4.506 save_magick() .................................................. 332
8.1.4.507 save_png() ...................................................... 332
8.1.4.508 save_pnm() ...................................................... 332
8.1.4.509 save_pnk() ...................................................... 333
8.1.4.510 save_pfm() ...................................................... 333
8.1.4.511 save_rgb() ...................................................... 333
8.1.4.512 save_rgba() .................................................... 333
8.1.4.513 save_tiff() ...................................................... 334
8.1.4.514 save_minc2() .................................................... 334
8.1.4.515 save_analyze() .................................................. 335
8.1.4.516 save_cimg() [1/2] .............................................. 335
8.1.4.517 save_cimg() [2/2] .............................................. 335
8.1.4.518 save_empty_cimg() [1/2] ......................................................... 336
8.1.4.519 save_empty_cimg() [2/2] ......................................................... 336
8.1.4.520 save_inr() ................................................................. 337
8.1.4.521 save_exr() ................................................................. 337
8.1.4.522 save_pandore() [1/2] ......................................................... 337
8.1.4.523 save_pandore() [2/2] ......................................................... 338
8.1.4.524 save_raw() [1/2] ................................................................. 338
8.1.4.525 save_raw() [2/2] ................................................................. 338
8.1.4.526 save_yuv() [1/2] ................................................................. 338
8.1.4.527 save_yuv() [2/2] ................................................................. 339
8.1.4.528 save_off() [1/2] ................................................................. 339
8.1.4.529 save_off() [2/2] ................................................................. 340
8.1.4.530 save_video() ................................................................. 340
8.1.4.531 save_ffmpeg_external() ................................................................. 340
8.1.4.532 save_gzip_external() ................................................................. 341
8.1.4.533 save_graphicsmagick_external() ......................................................... 341
8.1.4.534 save_imagemagick_external() ......................................................... 341
8.1.4.535 save_medcon_external() ................................................................. 342
8.1.4.536 save_other() ................................................................. 342
8.1.4.537 get_serialize() ................................................................. 343

8.2 CImgDisplay Struct Reference ......................................................... 343

8.2.1 Detailed Description ........................................................................ 349

8.2.2 Constructor & Destructor Documentation ......................................................... 349

8.2.2.1 ~CImgDisplay() ........................................................................ 350
8.2.2.2 CImgDisplay() [1/5] ........................................................................ 350
8.2.2.3 CImgDisplay() [2/5] ........................................................................ 350
8.2.2.4 CImgDisplay() [3/5] ........................................................................ 351
8.2.2.5 CImgDisplay() [4/5] ........................................................................ 351
8.2.2.6 CImgDisplay() [5/5] ........................................................................ 352

8.2.3 Member Function Documentation ......................................................... 352
8.3.31 display() ........................................ 362
8.3.32 show() ........................................ 363
8.3.33 close() ........................................ 363
8.3.34 move() ........................................ 363
8.3.35 resize() [1/4] ................................. 364
8.3.36 resize() [2/4] ................................. 364
8.3.37 resize() [3/4] ................................. 365
8.3.38 resize() [4/4] ................................. 365
8.3.39 set_normalization() .............................. 365
8.3.40 set_title() ..................................... 366
8.3.41 set_fullscreen() ............................... 366
8.3.42 toggle_fullscreen() ............................. 367
8.3.43 show_mouse() .................................. 367
8.3.44 hide_mouse() .................................. 367
8.3.45 set_mouse() .................................... 368
8.3.46 set_button() [1/2] ......................... 368
8.3.47 set_button() [2/2] ......................... 368
8.3.48 set_wheel() [1/2] ............................. 368
8.3.49 set_wheel() [2/2] ............................. 369
8.3.50 set_key() [1/2] ............................... 369
8.3.51 set_key() [2/2] ............................... 369
8.3.52 flush() ........................................ 370
8.3.53 wait() .......................................... 370
8.3.54 render() ....................................... 370
8.3.55 paint() ................................-------- 371
8.3.56 screenshot() [2/2] ......................... 371
8.3.57 snapshot() .................................... 371

8.3 CImgException Struct Reference ........................................ 372
8.3.1 Detailed Description ............................................. 372

8.4 CImgList<T> Struct Template Reference .................................. 373
8.4.1 Detailed Description ........................................ 387
8.4.2 Member Typedef Documentation ............................ 387
  8.4.2.1 iterator ............................................. 387
  8.4.2.2 const_iterator ................................... 387
  8.4.2.3 value_type ......................................... 388
8.4.3 Constructor & Destructor Documentation .................. 388
  8.4.3.1 ~CImgList() ........................................ 388
  8.4.3.2 CImgList() [1/19] ................................ 388
  8.4.3.3 CImgList() [2/19] ................................ 388
  8.4.3.4 CImgList() [3/19] ................................ 389
  8.4.3.5 CImgList() [4/19] ................................ 389
  8.4.3.6 CImgList() [5/19] ................................ 390
  8.4.3.7 CImgList() [6/19] ................................ 390
  8.4.3.8 CImgList() [7/19] ................................ 391
  8.4.3.9 CImgList() [8/19] ................................ 391
  8.4.3.10 CImgList() [9/19] ................................ 392
  8.4.3.11 CImgList() [10/19] ................................ 392
  8.4.3.12 CImgList() [11/19] ................................ 392
  8.4.3.13 CImgList() [12/19] ................................ 393
  8.4.3.14 CImgList() [13/19] ................................ 393
  8.4.3.15 CImgList() [14/19] ................................ 394
  8.4.3.16 CImgList() [15/19] ................................ 394
  8.4.3.17 CImgList() [16/19] ................................ 395
  8.4.3.18 CImgList() [17/19] ................................ 395
  8.4.3.19 CImgList() [18/19] ................................ 396
  8.4.3.20 CImgList() [19/19] ................................ 396
8.4.4 Member Function Documentation .......................... 396
  8.4.4.1 get_shared() ........................................ 396
  8.4.4.2 assign() [1/18] ..................................... 397
  8.4.4.3 clear() .............................................. 397
8.4.4.34 operator,() [1/2] ................................. 406
8.4.4.35 operator,() [2/2] ................................. 407
8.4.4.36 operator>() ..................................... 407
8.4.4.37 operator<( ) ..................................... 407
8.4.4.38 pixel_type() ..................................... 408
8.4.4.39 width() .......................................... 408
8.4.4.40 size() ............................................ 408
8.4.4.41 data() [1/2] ...................................... 409
8.4.4.42 data() [2/2] ...................................... 409
8.4.4.43 at() ............................................... 409
8.4.4.44 atNXYZC() [1/2] ................................. 409
8.4.4.45 atNXYZC() [2/2] ................................. 410
8.4.4.46 atNXYZ() [1/2] ................................. 410
8.4.4.47 atNXYZ() [2/2] ................................. 411
8.4.4.48 atNXY() [1/2] ................................. 412
8.4.4.49 atNXY() [2/2] ................................. 412
8.4.4.50 atNX() [1/2] ................................. 413
8.4.4.51 atNX() [2/2] ................................. 413
8.4.4.52 atN() [1/2] ................................. 414
8.4.4.53 atN() [2/2] ................................. 414
8.4.4.54 is_sameN() [1/2] ................................. 415
8.4.4.55 is_sameN() [2/2] ................................. 415
8.4.4.56 is_sameXYZC() ................................. 415
8.4.4.57 is_sameNXYZC() ................................. 415
8.4.4.58 containsNXYZC() ................................. 416
8.4.4.59 containsN() ................................. 416
8.4.4.60 contains() [1/8] ................................. 417
8.4.4.61 contains() [2/8] ................................. 417
8.4.4.62 contains() [3/8] ................................. 418
8.4.4.63 contains() [4/8] ................................. 418
<table>
<thead>
<tr>
<th>Section</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.4.4.64</td>
<td>contains()</td>
<td>419</td>
</tr>
<tr>
<td>8.4.4.65</td>
<td>contains()</td>
<td>419</td>
</tr>
<tr>
<td>8.4.4.66</td>
<td>contains()</td>
<td>419</td>
</tr>
<tr>
<td>8.4.4.67</td>
<td>contains()</td>
<td>420</td>
</tr>
<tr>
<td>8.4.4.68</td>
<td>min_max()</td>
<td>420</td>
</tr>
<tr>
<td>8.4.4.69</td>
<td>min_max()</td>
<td>420</td>
</tr>
<tr>
<td>8.4.4.70</td>
<td>max_min()</td>
<td>420</td>
</tr>
<tr>
<td>8.4.4.71</td>
<td>insert()</td>
<td>421</td>
</tr>
<tr>
<td>8.4.4.72</td>
<td>insert()</td>
<td>421</td>
</tr>
<tr>
<td>8.4.4.73</td>
<td>insert()</td>
<td>421</td>
</tr>
<tr>
<td>8.4.4.74</td>
<td>insert()</td>
<td>422</td>
</tr>
<tr>
<td>8.4.4.75</td>
<td>insert()</td>
<td>422</td>
</tr>
<tr>
<td>8.4.4.76</td>
<td>remove()</td>
<td>422</td>
</tr>
<tr>
<td>8.4.4.77</td>
<td>remove()</td>
<td>423</td>
</tr>
<tr>
<td>8.4.4.78</td>
<td>images()</td>
<td>423</td>
</tr>
<tr>
<td>8.4.4.79</td>
<td>get_shared_images()</td>
<td>423</td>
</tr>
<tr>
<td>8.4.4.80</td>
<td>get_append()</td>
<td>424</td>
</tr>
<tr>
<td>8.4.4.81</td>
<td>split()</td>
<td>424</td>
</tr>
<tr>
<td>8.4.4.82</td>
<td>push_back()</td>
<td>424</td>
</tr>
<tr>
<td>8.4.4.83</td>
<td>push_front()</td>
<td>425</td>
</tr>
<tr>
<td>8.4.4.84</td>
<td>push_back()</td>
<td>425</td>
</tr>
<tr>
<td>8.4.4.85</td>
<td>push_front()</td>
<td>425</td>
</tr>
<tr>
<td>8.4.4.86</td>
<td>erase()</td>
<td>425</td>
</tr>
<tr>
<td>8.4.4.87</td>
<td>get_select()</td>
<td>426</td>
</tr>
<tr>
<td>8.4.4.88</td>
<td>get_select()</td>
<td>426</td>
</tr>
<tr>
<td>8.4.4.89</td>
<td>load()</td>
<td>427</td>
</tr>
<tr>
<td>8.4.4.90</td>
<td>load_cimg()</td>
<td>427</td>
</tr>
<tr>
<td>8.4.4.91</td>
<td>load_cimg()</td>
<td>427</td>
</tr>
<tr>
<td>8.4.4.92</td>
<td>load_cimg()</td>
<td>427</td>
</tr>
<tr>
<td>8.4.4.93</td>
<td>load_parrec()</td>
<td>428</td>
</tr>
</tbody>
</table>
This is the reference documentation of the CImg Library, the C++ template image processing library. This documentation have been generated using the tool doxygen. It contains a detailed description of all classes and functions of the CImg Library.

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

You may be interested also in the presentation slides presenting an overview of the CImg Library capabilities.
Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

- CImg Library Overview .......................................................... 11
- FAQ : Frequently Asked Questions. ........................................ 14
- Setting Environment Variables .............................................. 18
- How to use CImg library with Visual C++ 2005 Express Edition? ........................................................... 19
- Tutorial : Getting Started ...................................................... 20
- Using Image Loops. ............................................................... 22
- Using Display Windows ......................................................... 28
- How pixel data are stored with CImg ...................................... 29
- Files IO in CImg ................................................................. 30
- Retrieving Command Line Arguments .................................... 31
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

- **cimg_library::cimg**
  Contains low-level functions and variables of the CImg Library
Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

- CImg<T> ......................................................... 65
- CImg<charT> ................................................. 65
- CImg<doubleT> ............................................... 65
- CImg<floatT> .................................................. 65
- CImg<intT> ..................................................... 65
- CImg<uintT> ................................................... 65
- CImg<ulongT> .................................................. 65
- CImgDisplay .................................................. 343
- CImgException .............................................. 372
- CImgList<T> ................................................... 373
- CImgList<boolT> ............................................. 373
- CImgList<charT> ............................................. 373
- CImgList<doubleT> ......................................... 373
- CImgList<ulongT> ............................................ 373
Chapter 5

Class Index

5.1 Class List

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

\texttt{CImg< T >}
Class representing an image (up to 4 dimensions wide), each pixel being of type \( T \) ........... 65

\texttt{CImgDisplay}
Allow the creation of windows, display images on them and manage user events (keyboard, mouse and windows events) ......................................................... 343

\texttt{CImgException}
Instances of \texttt{CImgException} are thrown when errors are encountered in a \texttt{CImg} function call 372

\texttt{CImgList< T >}
Represent a list of images \texttt{CImg< T >} .................................................. 373
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 stuff in C++.

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

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

The CImg Library is structured as follows :

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

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

- The namespace cimg_library::cimg defines a set of low-level functions and variables used by the library. Documented functions in this namespace can be safely used in your own program. But, never use the cimg_library::cimg namespace as a default namespace, since it contains functions whose names are already defined in the standard C/C++ library.
• The class `cimg_library::CImg` represents images up to 4-dimensions wide, containing pixels of type `T` (template parameter). This is actually the main class of the library.

• The class `cimg_library::CImgList` represents lists of `cimg_library::CImg<T>` images. It can be used for instance to store different frames of an image sequence.

• The class `cimg_library::CImgDisplay` is able to display images or image lists into graphical display windows. As you may guess, the code of this class is highly system-dependent but this is transparent for the programmer, as environment variables are automatically set by the CImg library (see also Setting Environment Variables).

• The class `cimg_library::CImgException` (and its subclasses) are used by the library to throw exceptions when errors occur. Those exceptions can be caught with a `try { ..} catch (CImgException) { .. }` block. Subclasses define precisely the type of encountered errors.

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

6.1.2 CImg version of "Hello world".

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

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

int main() {
    CImg<unsigned char>(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 :

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

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

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

6.1.3 How to compile ?

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

• **Microsoft Visual Studio.NET and Visual Express Edition** : Use the project files and solution files provided in the CImg Library package (directory 'compilation') to see how it works.
• **Intel ICL compiler**: Use the following command to compile a CImg-based program with ICL:

```
icl /Ox hello_world.cpp user32.lib gdi32.lib
```

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

```
g++ -o hello_word.exe hello_world.cpp -O2 -limgdi32
```

• **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 -lns1 -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 -I/usr/X11R6/include -L/usr/X11R6/lib -lm -lpthread -lX11
```

• **Dev-Cpp**: Use the project file provided in the CImg library package to see how it works.

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

### 6.1.4 What's next?

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

6.2.1 FAQ Summary

- General information and availability
  - What is the CImg Library ?
  - What platforms are supported ?
  - How is CImg distributed ?
  - What kind of people are concerned by CImg ?
  - What are the specificities of the CeCILL license ?
  - Who is behind CImg ?

- C++ related questions
  - What is the level of C++ knowledge needed to use CImg ?
  - How to use CImg in my own C++ program ?
  - Why is CImg entirely contained in a single header file ?

- Other resources
  - Translations

6.2.2 1. General information and availability

6.2.2.1 1.1. What is the CImg Library ?

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

6.2.2.2 1.2. What platforms are supported ?

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

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).

6.2.2.3 1.3. How is CImg distributed ?

The CImg Library is freely distributed as a complete .zip compressed package, hosted at the CImg server. 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 PDF format.
- 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 rhythm of release. A new version of the CImg package is released approximately every three months.

6.2.2.4 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.

6.2.2.5 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 certain 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 CeCILL-C and CeCILL licenses before releasing a software based on the CImg Library.

6.2.2.6 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, 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.

Generated by Doxygen
6.2.3 2. C++ related questions

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

The Clmg 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 Clmg class (defining the pixel type of the images). The design is simple but clean, making the library accessible even for non professional C++ programmers, while proposing strong extension capabilities for C++ experts.

6.2.3.2 2.2 How to use Clmg 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 Clmg images :

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

6.2.3.3 2.3 Why is Clmg 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 Clmg.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 Clmg 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, Clmg neither. Clmg 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 Clmg 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 Clmg, the two most important being CImg<T> and ClmgList<T> representing respectively an image and a collection of images. But contrary to the STL library, these two Clmg classes are strongly inter-dependent. All Clmg 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 Clmg. 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 ClmgDisplay of the Clmg library, by setting the compilation macro cimg_display to 0, avoiding thus to compile this class if you don't use display capabilities of Clmg 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 ClmgException, which is only few lines long and is obviously required in almost all methods of Clmg. Including this one is mandatory.
As a consequence, having a single header file instead of several ones is just a way for you to avoid including all of them, without any consequences on compilation time. This is both good technical and practical reasons to do like this.

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

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

6.2.4 3. Other resources

6.2.4.1 3.1 Translations

This FAQ has been translated to Serbo-Croatian language by Web Geeks.
6.3 Setting Environment Variables

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

```c
#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 "<tt>CImg.h</tt>" file. An include of the plugin file is performed in the CImg<T> class. If cimg_plugin is not specified (default), no include is done.

- **cimglist_plugin**: Same as cimg_plugin, but to add features to the CImgList<T> class.

- **cimgdisplay_plugin**: Same as cimg_plugin, but to add features to the CImgDisplay<T> class.

All these compilation variables can be checked, using the function `cimg_library::cimg::info()`, which displays a list of the different configuration variables and their values on the standard error output.
6.4 How to use CImg library with Visual C++ 2005 Express Edition ?

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

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

- Download CImg library
- Download and install Visual C++ 2005 Express Edition
- Download and install Microsoft Windows SDK
- Configure Visual C++ to take into account Microsoft SDK
  - 1. Go to menu "Tools -> options"
  - 2. Select option "Projects and Solutions -> VC++ Directories"
  - 3. In the select liste "Show directories for", choose "include files", and add C:\Program Files\Microsoft Platform SDK\Include (adapt if needed)
  - 4. In the select liste "Show directories for", choose "library files", and add C:\Program Files\Microsoft Platform SDK\Lib (adapt if needed) Edit file C:\Program Files\Microsoft Visual Studio 8\VC\VCProject Defaults\corewin_express.vsprops (adapt if needed)
  - 6. 7. 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
6.5 Tutorial: Getting Started.

Let's start to write our first program to get the idea. This will demonstrate how to load and create images, as well as handle image display and mouse events. Assume we want to load a color image lena.jpg, smooth it, display it in a window, 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:

```cpp
#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,image.width()-1,y,0,0),red,1,1,0,255,0);
            visu.draw_graph(image.get_crop(0,y,0,1,image.width()-1,y,0,1),green,1,1,0,255,0);
            visu.draw_graph(image.get_crop(0,y,0,2,image.width()-1,y,0,2),blue,1,1,0,255,0).display(draw_disp);
        }
    }
    return 0;
}
```

Here is a screenshot of the resulting program:

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

```cpp
#include "CImg.h"
```

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

```cpp
using namespace cimg_library;
```

Use the library namespace to ease the declarations afterward.

```cpp
int main() {
```

Definition of the main function.

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

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

```cpp
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 \( \text{image} = \text{image.get\_blur}(2.5); \) (more expensive, since it needs an additional copy operation).

```cpp
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.

```cpp
while (!main_disp.is_closed() && !draw_disp.is_closed()) {
    main_disp.wait();
}
```

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

```cpp
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,0,1,image.width()-1,y,0,1),red,1,0,256,0);
    
    visu.fill(0).draw_graph(image.get_crop(0,y,0,0,image.width()-1,y,0,0),red,1,0,256,0);
    
    visu.fill(0).draw_graph(image.get_crop(0,y,0,2,image.width()-1,y,0,2),green,1,0,256,0);
    
    visu.fill(0).draw_graph(image.get_crop(0,y,0,3,image.width()-1,y,0,3),blue,1,0,256,0);
    
    visu.fill(0).draw_graph(image.get_crop(0,y,0,0,image.width()-1,y,0,0),red,1,0,256,0);
    
    visu.fill(0).draw_graph(image.get_crop(0,y,0,1,image.width()-1,y,0,1),green,1,0,256,0);
    
    visu.fill(0).draw_graph(image.get_crop(0,y,0,2,image.width()-1,y,0,2),blue,1,0,256,0);
}
```

This line illustrates the pipeline property of most of the CImg class functions. The first function \( \text{fill}(0) \) 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 \( \text{draw\_graph()} \) which draws a plot in the image visu. The plot data are given by another image (the first argument of \( \text{draw\_graph()} \)). In this case, the given image is the red-component of the line \( y \) of the original image, retrieved by the function \( \text{get\_crop()} \) which returns a sub-image of the image image. Remember that images coordinates are 4D \( (x,y,z,c) \) and for color images, the R,G,B channels are respectively given by \( v=0, \ v=1 \) and \( v=2 \).

```cpp
visu.draw_graph(image.get_crop(0,y,0,1,image.width()-1,y,0,1),green,1,0,256,0); visu.draw_graph(image.get_crop(0,y,0,2,image.width()-1,y,0,2),blue,1,0,256,0); visu.draw_graph(image.get_crop(0,y,0,3,image.width()-1,y,0,3),red,1,0,256,0);
```

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

```cpp
visu.draw_graph(image.get_crop(0,y,0,1,image.width()-1,y,0,1),green,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 \( \text{display()} \) that just paints the image visu in the corresponding display window.

```cpp
...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 \( \text{CImg\_demo.cpp} \) which contains small and various examples of what you can do with the CImg Library. All CImg classes are used in this source, and the code can be easily modified to see what happens.
6.6 Using Image Loops.

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

- Loops over the pixel buffer
- Loops over image dimensions
- Loops over interior regions and borders.
- Loops using neighborhoods.

6.6.1 Loops over the pixel buffer

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

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


```cpp
CImg<float> img(320,200);
cimg_for(img,ptr,float) { *ptr=0; } // Equivalent to 'img.fill(0);'
```

- `cimg_rof(img,ptr,T)`: This macro does the same as `cimg_for()` but from the end to the beginning of the pixel buffer.

- `cimg_foroff(img,off)`: This macro loops over the pixel data buffer of the image `img`, using an offset, starting from the beginning of the buffer (first pixel, `off=0`) till the end of the buffer (last pixel value, `off = img.size()-1`).
  - `img` must be a (non empty) `cimg_library::CImg<T>` image of pixels `T`.
  - `off` is an inner-loop variable, only defined inside the scope of the loop.

Here is an example of use:

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

6.6.2 Loops over image dimensions

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

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

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

- `cimg_forXY(img,x,y)`: equivalent to:
  ```cpp
  cimg_forY(img,y)cimg_forX(img,x).
  ```
- `cimg_forXZ(img,x,z)`: equivalent to:
  ```cpp
  cimg_forZ(img,z)cimg_forX(img,x).
  ```
- `cimg_forYZ(img,y,z)`: equivalent to:
  ```cpp
  cimg_forZ(img,z)cimg_forY(img,y).
  ```
- `cimg_forXC(img,x,c)`: equivalent to:
  ```cpp
  cimg_forC(img,c)cimg_forX(img,x).
  ```
- `cimg_forYC(img,y,c)`: equivalent to:
  ```cpp
  cimg_forC(img,c)cimg_forY(img,y).
  ```
- `cimg_forZC(img,z,c)`: equivalent to:
  ```cpp
  cimg_forC(img,c)cimg_forZ(img,z).
  ```
- `cimg_forXYZ(img,x,y,z)`: equivalent to:
  ```cpp
  cimg_forZ(img,z)cimg_forXY(img,x,y).
  ```
- `cimg_forXYC(img,x,y,c)`: equivalent to:
  ```cpp
  cimg_forC(img,c)cimg_forXY(img,x,y).
  ```
- `cimg_forXZC(img,x,z,c)`: equivalent to:
  ```cpp
  cimg_forC(img,c)cimg_forXZ(img,x,z).
  ```
- `cimg_forYZC(img,y,z,c)`: equivalent to:
  ```cpp
  cimg_forC(img,c)cimg_forYZ(img,y,z).
  ```
- `cimg_forXYZC(img,x,y,z,c)`: equivalent to:
  ```cpp
  cimg_forC(img,c)cimg_forXYZ(img,x,y,z).
  ```

For all these loops, `x`, `y`, `z` and `v` are inner-defined variables only visible inside the scope of the loop. They don’t have to be defined before the call of the macro.

- `img` must be a (non empty) `cimg_library::CImg` image.

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

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

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

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

And also:

- `cimg_for_borderX(img,x,n)` : Loop along the x-axis, only for pixels inside a border of $n$ pixels wide.
- `cimg_for_borderY(img,y,n)` : Loop along the y-axis, only for pixels inside a border of $n$ pixels wide.
- `cimg_for_borderZ(img,z,n)` : Loop along the z-axis, only for pixels inside a border of $n$ pixels wide.
- `cimg_for_borderC(img,c,n)` : Loop along the c-axis, only for pixels inside a border of $n$ pixels wide.
- `cimg_for_borderXY(img,x,y,n)` : Loop along the (x,y)-axes, only for pixels inside a border of $n$ pixels wide.
- `cimg_for_borderXYZ(img,x,y,z,n)` : Loop along the (x,y,z)-axes, only for pixels inside a border of $n$ pixels wide.

For all these loops, $x$, $y$, $z$ and $c$ are inner-defined variables only visible inside the scope of the loop. They don't have to be defined before the call of the macro.

- `img` must be a (non empty) `cimg_library::CImg` image.
- The constant $n$ stands for the size of the border.

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

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

6.6.4 Loops using neighborhoods.

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

6.6.4.1 Neighborhood-based loops for 2D images

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

- \texttt{cimg\_for2x2(img,x,y,z,c,I,T)}: Loop along the (x,y)-axes using a centered 2x2 neighborhood.
- \texttt{cimg\_for3x3(img,x,y,z,c,I,T)}: Loop along the (x,y)-axes using a centered 3x3 neighborhood.
- \texttt{cimg\_for4x4(img,x,y,z,c,I,T)}: Loop along the (x,y)-axes using a centered 4x4 neighborhood.
- \texttt{cimg\_for5x5(img,x,y,z,c,I,T)}: Loop along the (x,y)-axes using a centered 5x5 neighborhood.

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

6.6.4.2 Neighborhood-based loops for 3D images

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

- \texttt{cimg\_for2x2x2(img,x,y,z,c,I,T)}: Loop along the (x,y,z)-axes using a centered 2x2x2 neighborhood.
- \texttt{cimg\_for3x3x3(img,x,y,z,c,I,T)}: Loop along the (x,y,z)-axes using a centered 3x3x3 neighborhood.

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

6.6.4.3 Defining neighborhoods

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

For instance, a 3x3 neighborhood can be defined either as a 'float[9]' or a 'CImg<float>(3,3)' variable.
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, 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 additional indices are introduced: a (stands for 'after') and b (stands for 'before'), so that:

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

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

6.6.4.5 Example codes

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

```cpp
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.

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

As you can see, explaining the use of the CImg neighborhood macros is actually more difficult than using them!
6.7 Using Display Windows.

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

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

- **0**: No pixel normalization is performed when displaying an image. This is the fastest process, but you must be sure your displayed image have pixel values inside the range \([0,255]\).

- **1**: Pixel value normalization is done for each new image display. Image pixels are not modified themselves, only displayed pixels are normalized.

- **2**: Pixel value normalization is done for the first image display, then the normalization parameters are kept and used for all the next image displays.
6.8 How pixel data are stored with CImg.

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

A CImg\(<T>\) is basically a 4th-dimensional array (width,height,depth,dim), and its pixel data are stored linearly in a single memory buffer of general size \((\text{width} \times \text{height} \times \text{depth} \times \text{dim})\). 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\*\text{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 instance image, with a classical scanline run.

So, a color image with \(\text{dim}=3\) and \(\text{depth}=1\), will be stored in memory as:

\[
\text{R1R2R3R4R5R6......G1G2G3G4G5G6.......B1B2B3B4B5B6..... (i.e following a 'planar' structure)}
\]

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 \((1\times5\times1\times1)\) CImg\(<T>\) (column vector A) will be stored as: A1A2A3A4A5 where A1 = img\((0,0)\), A2 = img\((0,1)\), ... , A5 = img\((0,4)\).

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

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

\[
\text{img.get_shared_channel(0).blur(2); img.get_shared_channels(1,2).mirror('x');}
\]

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

The CImg Library can NATIVELY handle the following file formats:

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

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

The CImg library offers facilities to retrieve command line arguments in a console-based program, as it is a commonly needed operation. Three macros \texttt{cimg\_usage()}, \texttt{cimg\_help()} and \texttt{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 \texttt{-h} or \texttt{-help} will automatically display the program usage, followed by the list of requested options.

6.10.1 The \texttt{cimg\_usage()} macro

The macro \texttt{cimg\_usage(usage)} may be used to describe the program goal and usage. It is generally inserted one time after the \texttt{int main(int argc,char **argv)} definition.

Parameters

| usage | A string describing the program goal and usage. |

Precondition

The function where \texttt{cimg\_usage()} is used must have correctly defined \texttt{argc} and \texttt{argv} variables.

6.10.2 The \texttt{cimg\_help()} macro

The macro \texttt{cimg\_help(str)} will display the string \texttt{str} only if the \texttt{-help} or \texttt{-help} option are invoked when running the program.

6.10.3 The \texttt{cimg\_option()} macro

The macro \texttt{cimg\_option(name,default,usage)} may be used to retrieve an option value from the command line.

Parameters

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

Returns

\texttt{cimg\_option()} returns an object that has the \texttt{same type} as the default value \texttt{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 \texttt{default}. Warning, this can be confusing in some situations (look at the end of the next section).

Precondition

The function where \texttt{cimg\_option()} is used must have correctly defined \texttt{argc} and \texttt{argv} variables.
6.10.4 Example of use

The code below uses the macros \texttt{cimg\_usage()} and \texttt{cimg\_option()}. It loads an image, smoothes it and quantifies it with a specified number of values.

```cpp
#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",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 \texttt{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 \texttt{cimg\_option(option,default,usage)} is defined by the type of \texttt{default}, undesired casts may appear when writing code such as:

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

In this case, \texttt{sigma} will always be equal to an integer (since the default value \texttt{0} is an integer). When passing a float value on the command line, a \texttt{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 \texttt{0.0} as the default value in this case.

6.10.5 How to learn more about command line options ?

You should take a look at the examples \texttt{examples/gmic.cpp} provided in the CImg Library package. This is a command line based image converter which intensively uses the \texttt{cimg\_option()} and \texttt{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

• cimg

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

Classes

• struct CImg

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

• struct CImgDisplay

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

• struct CImgException

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

• struct CImgList

  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 inclusion of other C++ header files. Anyway, it should not happen often and you should reasonably start most of your CImg-based programs with

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

to simplify the declaration of CImg Library objects afterwards.
Contains low-level functions and variables of the CImg Library.

Functions

- `std::FILE * output (std::FILE *file)`
  Get/set default output stream for the CImg library messages.
- `void info ()`
  Print information about CImg environment variables.
- `template<typename T > void unused (const T &,...)`
  Avoid warning messages due to unused parameters. Do nothing actually.
- `unsigned int & exception_mode (const unsigned int mode)`
  Set current CImg exception mode.
- `unsigned int & exception_mode ()`
  Return current CImg exception mode.
- `unsigned int openmp_mode (const unsigned int mode)`
  Set current CImg openmp mode.
- `unsigned int openmp_mode ()`
  Return current CImg openmp mode.
- `int dialog (const char *const title, const char *const msg, const char *const button1_label, const char *const button2_label, const char *const button3_label, const char *const button4_label, const char *const button5←_label, const char *const button6_label, const bool is_centered)`
  Display a simple dialog box, and wait for the user's response [specialization].
- `double eval (const char *const expression, const double x, const double y, const double z, const double c)`
  Evaluate math expression.
- `void warn (const char *const format,...)`
  Display a warning message on the default output stream.
- `int system (const char *const command, const char *const module_name=0, const bool is_verbose=false)`
  Exchange values of variables a and b.
- `template<typename T > void swap (T &a, T &b)`
  Exchange values of variables (a1,a2) and (b1,b2).
- `template<typename T1 , typename T2 > void swap (T1 &a1, T1 &b1, T2 &a2, T2 &b2)`
  Exchange values of variables (a1,a2,a3) and (b1,b2,b3).
- `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,a4) and (b1,b2,b3,a4).
- `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,a3,a4) and (b1,b2,b3,a4).
- `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,a3,a4,a5) and (b1,b2,b3,a4,a5).
Exchange values of variables (a₁,a₂,....,a₆) and (b₁,b₂,....,b₆).

```cpp
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).

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

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

- bool endianness ()
  
  Return the endianness of the current architecture.

- template<typename T >
  
  void invert_endianness (T *const buffer, const unsigned long size)
  
  Reverse endianness of all elements in a memory buffer.

- template<typename T >
  
  T & invert_endianness (T &a)
  
  Reverse endianness of a single variable.

- unsigned long 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.

- long wait (const unsigned int milliseconds)
  
  Wait for a given number of milliseconds since the last call to wait().

- template<typename T ,
             typename t >
  
  T cut (const T &val, const t &val_min, const t &val_max)
  
  Cut (i.e. clamp) value in specified interval.

- template<typename T >
  
  T 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.

- double acosh (const double x)
  
  Return hyperbolic arcosine of a value.

- double asinh (const double x)
  
  Return hyperbolic arcsine of a value.

- double atanh (const double x)
  
  Return hyperbolic arctangent of a value.

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

- double log2 (const double x)
  
  Return base-2 logarithm of a value.

- template<typename T >
  
  T sqr (const T &val)
  
  Return square of a value.
• template<typename T >
  double cb( const T &x)
  
  \textit{Return cubic root of a value.}

• template<typename t >
  t min( const t &a, const t &b, const t &c)
  
  \textit{Return the minimum between three values.}

• template<typename t >
  t min( const t &a, const t &b, const t &c, const t &d)
  
  \textit{Return the minimum between four values.}

• template<typename t >
  t max( const t &a, const t &b, const t &c)
  
  \textit{Return the maximum between three values.}

• template<typename t >
  t max( const t &a, const t &b, const t &c, const t &d)
  
  \textit{Return the maximum between four values.}

• template<typename T >
  T sign( const T &x)
  
  \textit{Return the sign of a value.}

• template<typename T >
  unsigned long nearest_pow2( const T &x)
  
  \textit{Return the nearest power of 2 higher than given value.}

• template<typename T >
  T mod( const T &x, const T &m)
  
  \textit{Return the modulo of a value.}

• template<typename T >
  T minmod( const T &a, const T &b)
  
  \textit{Return the min-mod of two values.}

• template<typename T >
  T round( const T &x, const double y, const int rounding_type=0)
  
  \textit{Return rounded value.}

• template<typename T >
  T hypot( const T x, const T y)
  
  \textit{Return }\sqrt{x^2 + y^2}.\textit{ }

• double factorial( const int n)
  
  \textit{Calculate factorial of }n.\textit{ }

• double permutations( const int k, const int n, const bool with_order)
  
  \textit{Return the number of permutations of }k\textit{ objects in a set of }n\textit{ objects.}

• double fibonacci( const int n)
  
  \textit{Calculate fibonacci number.}

• long gcd( long a, long b)
  
  \textit{Calculate greatest common divisor.}

• char lowercase( const char x)
  
  \textit{Convert Ascii character to lower case.}

• void lowercase( char *const str)
  
  \textit{Convert C-string to lower case.}

• char uppercase( const char x)
  
  \textit{Convert Ascii character to upper case.}

• void uppercase( char *const str)
  
  \textit{Convert C-string to upper case.}

• bool is_blank( const char c)
  
  \textit{Return }true\textit{ if input character is blank (space, tab, or non-printable character).}

• 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.

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

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

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

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

- `void strwindows_reserved (char *const str, const char c='_')`
  Replace reserved characters (for Windows filename) by another character.

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

- `const char * basename (const char *const s, const char separator='/')`
  Return the basename of a filename.

- `std::FILE * fopen (const char *const path, const char *const mode)`
  Open a file.

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

- `int fseek (FILE *stream, long offset, int origin)`
  Version of 'fseek()' that supports >64bits offsets everywhere (for Windows).

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

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

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

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

- `template<typename T >
  int fdate (const char *const path, T *attr, const unsigned int nb_attr)`
  Get last write time of a given file or directory (multiple-attributes version).

- `template<typename T >
  int date (T *attr, const unsigned int nb_attr)`
  Get current local time (multiple-attributes version).

- `int date (unsigned int attr)`
  Get current local time (single-attribute version).

- `const char * temporary_path (const char *const user_path, const bool reinit_path)`
  Get/set path to store temporary files.

- `const char * imagemagick_path (const char *const user_path, const bool reinit_path)`
  Get/set path to the Program Files/ directory (Windows only).

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

- `const char * medcon_path (const char *const user_path, const bool reinit_path)`
  Get/set path to the XMedcon's medcon binary.
• const char * ffmpeg_path (const char * const user_path, const bool reinit_path)
  Get/set path to the FFmpeg’s ffmpeg binary.
• const char * gzip_path (const char * const user_path, const bool reinit_path)
  Get/set path to the gzip binary.
• const char * gunzip_path (const char * const user_path, const bool reinit_path)
  Get/set path to the gunzip binary.
• const char * dcraw_path (const char * const user_path, const bool reinit_path)
  Get/set path to the dcraw binary.
• const char * wget_path (const char * const user_path, const bool reinit_path)
  Get/set path to the wget binary.
• const char * curl_path (const char * const user_path, const bool reinit_path)
  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 digits, char * const str)
  Generate a numbered version of a filename.
• template<typename T >
  size_t fread (T * const ptr, const size_t nmemb, std::FILE *stream)
  Read data from file.
• template<typename T >
  size_t fwrite (const T * ptr, const size_t nmemb, std::FILE *stream)
  Write data to file.
• void fempty (std::FILE * const file, const char * const filename)
  Create an empty file.
• const char * ftype (std::FILE * const file, const char * const filename)
  Try to guess format from an image file.
• char * load_network (const char * const url, char * const filename_local, const unsigned int timeout, const
  bool try_fallback, const char * const referer)
  Load file from network as a local temporary file.
• const char * option (const char * const name, const int argc, const char * const argv, const char * const
  _default, const char * const usage, const bool reset_static)
  Return options specified on the command line.
• CImgList< CImg< t > >
  files (const char * const path, const bool is_pattern=false, const unsigned int mode=2,
  const bool include_path=false)
  Return list of files/directories in specified directory.
• template<typename t >
  int dialog (const char * const title, const char * const msg, char * const button1_label, 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)
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)
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 PADDIV 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 output()

```cpp
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.

7.2.2.2 info()

void info ( )

Print information about CImg environment variables.

Note
Output is done on the default output stream.

7.2.2.3 exception_mode( ) [1/2]

unsigned int & 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

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

7.2.2.4 exception_mode() [2/2]

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

Return current CImg exception mode.

Note
By default, return the value of configuration macro cimg_verbosity
7.2.2.5 openmp_mode()

```c
unsigned int cimg_library::cimg::openmp_mode (const unsigned int mode)
```

Set current CImg openmp mode.

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

**Parameters**

<table>
<thead>
<tr>
<th>mode</th>
<th>Desired openmp mode. Possible values are:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Never parallelize.</td>
</tr>
<tr>
<td>1</td>
<td>Always parallelize.</td>
</tr>
<tr>
<td>2</td>
<td>Adaptive parallelization mode (default behavior).</td>
</tr>
</tbody>
</table>

7.2.2.6 eval()

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

Evaluate math expression.

**Parameters**

<table>
<thead>
<tr>
<th>expression</th>
<th>C-string describing the formula to evaluate.</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Value of the pre-defined variable x.</td>
</tr>
<tr>
<td>y</td>
<td>Value of the pre-defined variable y.</td>
</tr>
<tr>
<td>z</td>
<td>Value of the pre-defined variable z.</td>
</tr>
<tr>
<td>c</td>
<td>Value of the pre-defined variable c.</td>
</tr>
</tbody>
</table>

**Returns**

Result of the formula evaluation.

**Note**

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

**Example**

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

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 CImgWarning exception instead.

Warning

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

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

instead of

cimg::warn(warning_message);

if warning_message can be arbitrary, to prevent nasty memory access.

7.2.2.8 system()

int cimg_library::cimg::system (  
    const char *const command,  
    const char *const module_name = 0,  
    const bool is_verbose = false )

Parameters

| command   | C-string containing the command line to execute. |
| module_name | Module name. |

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.
7.2.2.9 endianness()

bool cimg_library::cimg::endianness ( )

Return the endianness of the current architecture.

Returns

false for Little Endian or true for Big Endian.

7.2.2.10 invert_endianness() [1/2]

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

Reverse endianness of all elements in a memory buffer.

Parameters

<table>
<thead>
<tr>
<th>in, out</th>
<th>buffer</th>
<th>Memory buffer whose endianness must be reversed.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>size</td>
<td>Number of buffer elements to reverse.</td>
</tr>
</tbody>
</table>

7.2.2.11 invert_endianness() [2/2]

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.

7.2.2.12 time()

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.

7.2.2.13 tic()

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()).

7.2.2.14 toc()

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().

7.2.2.15 sleep()

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 resources during the sleeping time. It can be used to temporize your program properly, without wasting CPU time.
7.2.2.16 wait()

```cpp
long 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**

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

7.2.2.17 mod()

```cpp
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.

7.2.2.18 minmod()

```cpp
T cimg_library::cimg::minmod ( const T & a, const T & b )
```

Return the min-mod of two values.
Note

\[ \text{\texttt{minmod}}(a, b) \text{ is defined to be:} \]

\[ \begin{align*}
\text{\texttt{minmod}}(a, b) &= \text{\texttt{min}}(a, b), \text{ if } a \text{ and } b \text{ have the same sign.} \\
\text{\texttt{minmod}}(a, b) &= 0, \text{ if } a \text{ and } b \text{ have different signs.}
\end{align*} \]

### 7.2.2.19 round() \[ T \text{\texttt{cimg}}::\text{\texttt{round}} ( \]

\[ \begin{align*}
&\text{\texttt{const T \& x}}, \\
&\text{\texttt{const double y}}, \\
&\text{\texttt{const int rounding_type = 0}}) \\
\end{align*} \]

Return rounded value.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>Value to be rounded.</td>
</tr>
<tr>
<td>( y )</td>
<td>Rounding precision.</td>
</tr>
<tr>
<td>( \text{rounding_type} )</td>
<td>Type of rounding operation ((0 = \text{nearest}, -1 = \text{backward}, 1 = \text{forward})).</td>
</tr>
</tbody>
</table>

Returns

Rounded value, having the same type as input value \( x \).

### 7.2.2.20 atof()

\[ \text{\texttt{double cimg}}::\text{\texttt{atof}} ( \]

\[ \begin{align*}
&\text{\texttt{const char \*const str}}) \end{align*} \]

Read value in a C-string.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{str} )</td>
<td>C-string containing the float value to read.</td>
</tr>
</tbody>
</table>

Returns

Read value.

Note

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

Generated by Doxygen
### 7.2.2.21 strncasecmp()

```cpp
def strncasecmp(cstr1, cstr2, l):
    return 0
```

*Compare the first \( l \) characters of two C-strings, ignoring the case.*

**Parameters**

<table>
<thead>
<tr>
<th>( str1 )</th>
<th>C-string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>( str2 )</td>
<td>C-string.</td>
</tr>
<tr>
<td>( l )</td>
<td>Number of characters to compare.</td>
</tr>
</tbody>
</table>

**Returns**

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

**Note**

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

### 7.2.2.22 strcasecmp()

```cpp
def strcasecmp(cstr1, cstr2):
    return 0
```

*Compare two C-strings, ignoring the case.*

**Parameters**

<table>
<thead>
<tr>
<th>( str1 )</th>
<th>C-string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>( str2 )</td>
<td>C-string.</td>
</tr>
</tbody>
</table>

**Returns**

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

**Note**

This function has to be defined since it is not provided by all C++-compilers (not ANSI).
7.2.2.23 strellipsize() [1/2]

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

Ellipsizes a string.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str</code></td>
<td>C-string.</td>
</tr>
<tr>
<td><code>l</code></td>
<td>Max number of characters.</td>
</tr>
<tr>
<td><code>is_ending</code></td>
<td>Tell if the dots are placed at the end or at the center of the ellipsized string.</td>
</tr>
</tbody>
</table>

7.2.2.24 strellipsize() [2/2]

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

Ellipsizes a string.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str</code></td>
<td>C-string.</td>
</tr>
<tr>
<td><code>res</code></td>
<td>Output C-string.</td>
</tr>
<tr>
<td><code>l</code></td>
<td>Max number of characters.</td>
</tr>
<tr>
<td><code>is_ending</code></td>
<td>Tell if the dots are placed at the end or at the center of the ellipsized string.</td>
</tr>
</tbody>
</table>

7.2.2.25 strpare()

```cpp
bool cimg_library::cimg::strpare (  
    char *const str,  
    const char delimiter,  
    const bool is_symmetric,  
    const bool is_iterative )
```

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str</code></td>
<td>C-string to work with (modified at output).</td>
</tr>
<tr>
<td><code>delimiter</code></td>
<td>Delimiter character code to remove.</td>
</tr>
<tr>
<td><code>is_symmetric</code></td>
<td>Tells if the removal is done only if delimiters are symmetric (both at the beginning and the end of <code>str</code>).</td>
</tr>
<tr>
<td><code>is_iterative</code></td>
<td>Tells if the removal is done if several iterations are possible.</td>
</tr>
</tbody>
</table>
Returns

    true if delimiters have been removed, false otherwise.

7.2.2.26 strwindows_reserved()

void cimg_library::cimg::strwindows_reserved (  
    char *const str,  
    const char c = '_')  

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

Parameters

<table>
<thead>
<tr>
<th>in, out</th>
<th>str</th>
<th>C-string to work with (modified at output).</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>c</td>
<td>Replacement character.</td>
</tr>
</tbody>
</table>

7.2.2.27 strunescape()

void cimg_library::cimg::strunescape (  
    char *const str )  

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

Parameters

<table>
<thead>
<tr>
<th>in, out</th>
<th>str</th>
<th>C-string to work with (modified at output).</th>
</tr>
</thead>
</table>

7.2.2.28 fopen()

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

Open a file.

Parameters

<table>
<thead>
<tr>
<th>path</th>
<th>Path of the filename to open.</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>C-string describing the opening mode.</td>
</tr>
</tbody>
</table>
Returns

Opened file.

Note

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

7.2.2.29  fclose()

```cpp
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

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

7.2.2.30  is_directory()

```cpp
bool cimg_library::cimg::is_directory ( const char * const path )
```

Check if a path is a directory.

Parameters

- `path` | Specified path to test.

7.2.2.31  is_file()

```cpp
bool cimg_library::cimg::is_file ( const char * const path )
```
Check if a path is a file.

**Parameters**

| path | Specified path to test. |

### 7.2.2.32 fsize()

```cpp
clong long cimg_library::cimg::fsize (  
    const char *const filename )
```

Get file size.

**Parameters**

| filename | Specified filename to get size from. |

**Returns**

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

### 7.2.2.33 fdate() [1/2]

```cpp
cint cimg_library::cimg::fdate (  
    const char *const path,  
    T *attr,  
    const unsigned int nb_attr )
```

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

**Parameters**

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

**Returns**

Latest read attribute.
7.2.2.34  

```c
int cimg_library::cimg::fdate (
    const char * const path,
    unsigned int attr )
```

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

**Parameters**

<table>
<thead>
<tr>
<th>path</th>
<th>Specified path to get attributes from.</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr</td>
<td>Type of requested time attributes. Can be { 0=year</td>
</tr>
</tbody>
</table>

**Returns**

Specified attribute or -1 if an error occurred.

7.2.2.35  

```c
int cimg_library::cimg::date ( 
    T * attr,
    const unsigned int nb_attr )
```

Get current local time (multiple-attributes version).

**Parameters**

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

**Returns**

Latest read attribute.

7.2.2.36  

```c
int cimg_library::cimg::date ( 
    unsigned int attr )
```

Get current local time (single-attribute version).
### Parameters

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

### Returns

Specified attribute or -1 if an error occurred.

#### 7.2.2.37 temporary_path()

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

Get/set path to store temporary files.

### Parameters

<table>
<thead>
<tr>
<th>user_path</th>
<th>Specified path, or 0 to get the path currently used.</th>
</tr>
</thead>
<tbody>
<tr>
<td>reinit_path</td>
<td>Force path to be recalculated (may take some time).</td>
</tr>
</tbody>
</table>

### Returns

Path where temporary files can be saved.

#### 7.2.2.38 imagemagick_path()

```c
const char * imagemagick_path (  
    const char *const user_path,  
    const bool reinit_path )
```

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

### Parameters

<table>
<thead>
<tr>
<th>user_path</th>
<th>Specified path, or 0 to get the path currently used.</th>
</tr>
</thead>
<tbody>
<tr>
<td>reinit_path</td>
<td>Force path to be recalculated (may take some time).</td>
</tr>
</tbody>
</table>

### Returns

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

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

Returns

Path containing the `convert` binary.

### 7.2.2.39 graphicsmagick_path()

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

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

Parameters

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

Returns

Path containing the `gm` binary.

### 7.2.2.40 medcon_path()

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

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

Parameters

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

Returns

Path containing the `medcon` binary.
### 7.2.2.41 ffmpeg_path()

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

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>user_path</code></td>
<td>Specified path, or 0 to get the path currently used.</td>
</tr>
<tr>
<td><code>reinit_path</code></td>
<td>Force path to be recalculated (may take some time).</td>
</tr>
</tbody>
</table>

**Returns**

Path containing the `ffmpeg` binary.

### 7.2.2.42 gzip_path()

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

Get/set path to the `gzip` binary.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>user_path</code></td>
<td>Specified path, or 0 to get the path currently used.</td>
</tr>
<tr>
<td><code>reinit_path</code></td>
<td>Force path to be recalculated (may take some time).</td>
</tr>
</tbody>
</table>

**Returns**

Path containing the `gzip` binary.

### 7.2.2.43 gunzip_path()

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

Get/set path to the `gunzip` binary.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>user_path</code></td>
<td>Specified path, or 0 to get the path currently used.</td>
</tr>
<tr>
<td><code>reinit_path</code></td>
<td>Force path to be recalculated (may take some time).</td>
</tr>
</tbody>
</table>

**Returns**

Path containing the `gunzip` binary.
Returns

Path containing the \texttt{gunzip} binary.

### 7.2.2.44 \texttt{dcraw\_path()}

\begin{verbatim}
const char * dcraw_path ( 
    const char *const user_path, 
    const bool reinit_path )
\end{verbatim}

Get/set path to the \texttt{dcraw} binary.

\textbf{Parameters}

<table>
<thead>
<tr>
<th>user_path</th>
<th>Specified path, or 0 to get the path currently used.</th>
</tr>
</thead>
<tbody>
<tr>
<td>reinit_path</td>
<td>Force path to be recalculated (may take some time).</td>
</tr>
</tbody>
</table>

Returns

Path containing the \texttt{dcraw} binary.

### 7.2.2.45 \texttt{wget\_path()}

\begin{verbatim}
const char * wget_path ( 
    const char *const user_path, 
    const bool reinit_path )
\end{verbatim}

Get/set path to the \texttt{wget} binary.

\textbf{Parameters}

<table>
<thead>
<tr>
<th>user_path</th>
<th>Specified path, or 0 to get the path currently used.</th>
</tr>
</thead>
<tbody>
<tr>
<td>reinit_path</td>
<td>Force path to be recalculated (may take some time).</td>
</tr>
</tbody>
</table>

Returns

Path containing the \texttt{wget} binary.

### 7.2.2.46 \texttt{curl\_path()}

\begin{verbatim}
const char * curl_path ( 
    const char *const user_path, 
    const bool reinit_path )
\end{verbatim}

Get/set path to the \texttt{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.

### 7.2.2.47 split_filename()

```c++
const char* cimg_library::cimg::split_filename (const char* const filename,
                                              char* const body = 0)
```

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

filename and body must not overlap!

### 7.2.2.48 fread()

```c++
size_t cimg_library::cimg::fread (T *const ptr,
                                 const size_t nmemb,
                                 std::FILE * const 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

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

```cpp
csize_t cimg_library::cimg::fwrite (const T * ptr,
const size_t nmemb,
std::FILE * stream )
```

Write data to file.

**Parameters**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ptr</strong></td>
<td>Pointer to memory buffer containing the binary data to write on file.</td>
<td></td>
</tr>
<tr>
<td><strong>nmemb</strong></td>
<td>Number of elements to write.</td>
<td></td>
</tr>
<tr>
<td><strong>stream</strong></td>
<td>File to write data on.</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

Number of written elements.

**Note**

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

7.2.2.50 fempty()

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

Create an empty file.

**Parameters**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>file</strong></td>
<td>Input file (can be 0 if filename is set).</td>
</tr>
<tr>
<td><strong>filename</strong></td>
<td>Filename, as a C-string (can be 0 if file is set).</td>
</tr>
</tbody>
</table>

7.2.2.51 ftype()

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

Try to guess format from an image file.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Input file (can be 0 if filename is set).</td>
</tr>
<tr>
<td>filename</td>
<td>Filename, as a C-string (can be 0 if file is set).</td>
</tr>
</tbody>
</table>

Returns

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

7.2.2.52 load_network()

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

Load file from network as a local temporary file.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>URL of the filename, as a C-string.</td>
</tr>
<tr>
<td>filename_local</td>
<td>C-string containing the path to a local copy of filename.</td>
</tr>
<tr>
<td>timeout</td>
<td>Maximum time (in seconds) authorized for downloading the file from the URL.</td>
</tr>
<tr>
<td>try_fallback</td>
<td>When using libcurl, tells using system calls as fallbacks in case of libcurl failure.</td>
</tr>
<tr>
<td>referer</td>
<td>Referer used, as a C-string.</td>
</tr>
</tbody>
</table>

Returns

Value of filename_local.

Note

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

7.2.2.53 files()

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

Return list of files/directories in specified directory.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Path to the directory. Set to 0 for current directory.</td>
</tr>
<tr>
<td>is_pattern</td>
<td>Tell if specified path has a matching pattern in it.</td>
</tr>
<tr>
<td>mode</td>
<td>Output type, can be primary { 0=files only</td>
</tr>
<tr>
<td>include_path</td>
<td>Tell if path must be included in resulting filenames.</td>
</tr>
</tbody>
</table>

Returns

A list of filenames.

7.2.2.54 dialog()

```cpp
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

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

Returns

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

- Up to 6 buttons can be defined in the dialog window.
- The function returns when a user clicked one of the button or closed the dialog window.
- If a button text is set to 0, the corresponding button (and the following) will not appear in the dialog box. At least one button must be specified.
Chapter 8

Class Documentation

8.1 CImg< T > Struct Template Reference

Class representing an image (up to 4 dimensions wide), 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 ()
  
  Destroy image.

- CImg ()
  
  Construct empty image.

- CImg (const unsigned int size_x, const unsigned int size_y=1, const unsigned int size_z=1, const unsigned int size_c=1)

  Construct image with specified size.

- CImg (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const T &value)

  Construct image with specified size and initialize pixel values.

- CImg (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const int value0, const int value1,...)

  Construct image with specified size and initialize pixel values from a sequence of integers.

- 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.

• 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 [specialization].

• template<typename t >
  CImg (const t ∗const values, const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const char ∗const axes_order)
  Construct image from memory buffer with specified size and pixel ordering scheme.

• CImg (const char ∗const filename)
  Construct image from reading an image file.

• template<typename t >
  CImg (const CImg< t > &img)
  Construct image copy.

• CImg (const T &value)
  Construct image copy [specialization].

• template<typename t >
  CImg (const CImg< t > &img, const bool is_shared)
  Advanced copy constructor.

• CImg< T > & assign ()
  Construct empty image [in-place version].

• CImg< T > & assign (const unsigned int size_x, const unsigned int size_y=1, const unsigned int size_z=1, const unsigned int size_c=1)
  Construct image with specified size [in-place version].

• CImg< T > & assign (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const T &value)
  Construct image with specified size and initialize pixel values [in-place version].

• CImg< T > & assign (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const int value0, const int value1, ...)
  Construct image with specified size and initialize pixel values from a sequence of integers [in-place version].

• CImg< T > & assign (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const double value0, const double value1, ...)
  Construct image with specified size and initialize pixel values from a sequence of doubles [in-place version].

• CImg< T > & assign (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const char ∗const values, const bool repeat_values)
  Construct image with specified size and initialize pixel values from a value string [in-place version].

• template<typename t >
  CImg< T > & assign (const t ∗const values, const unsigned int size_x, const unsigned int size_y=1, const unsigned int size_z=1, const unsigned int size_c=1)
  Construct image with specified size and initialize pixel values from a memory buffer [in-place version].
• CImg<T>& assign (const T*const values, const unsigned int size_x, const unsigned int size_y=1, const unsigned int size_z=1, const unsigned int size_c=1)
  Construct image with specified size and initialize pixel values from a memory buffer [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)
  Construct image from memory buffer with specified size and pixel ordering scheme.

• CImg<T>& assign (const char*const filename)
  Construct image from reading an image file [in-place version].

• template<typename t>
  CImg<T>& assign (const CImg<t>&img)
  Construct image copy [in-place version].

• 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)
  Construct image with dimensions borrowed from another image [in-place version].

• CImg<T>& swap (CImg<T>&img)
  Swap fields of two image instances.

• static CImg<T>& empty ()
  Return a reference to an empty image.

• static const CImg<T>& const_empty ()
  Return a reference to an empty image [const version].
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 ulongT wh, const ulongT 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 ulongT wh, const ulongT 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 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<tpename 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<tpename cimg::superset<T, t>::type> operator+ (const CImg<T>& img) const
  
  Addition operator.

- CImg<T> & operator+= (const t value)
8.1 CImg\textless{} T \textgreater{} Struct Template Reference

In-place subtraction operator.
\begin{itemize}
  \item CImg\textless{} T \textgreater{} \& operator\textasciimacron{} (const char *const expression)  
    In-place subtraction operator.
  \item template<typename t >
    CImg\textless{} T \textgreater{} \& operator\textasciimacron{} (const CImg\textless{} t \textgreater{} &img)  
    In-place subtraction operator.
  \item CImg\textless{} T \textgreater{} \& operator\textasciimacron{} ()  
    In-place decrement operator (prefix).
  \item CImg\textless{} T \textgreater{} \& operator\textasciimacron{} (int)  
    In-place decrement operator (postfix).
  \item CImg\textless{} T \textgreater{} operator\textasciimacron{} () const  
    Replace each pixel by its opposite value.
  \item template<typename t >
    CImg\textless{} t \textgreater{}::type operator\textasciimacron{} (const t value) const  
    Subtraction operator.
  \item CImg\textless{} Tfloat \textgreater{} operator\textasciimacron{} (const char *const expression) const  
    Subtraction operator.
  \item template<typename t >
    CImg\textless{} t \textgreater{}::type operator\textasciimacron{} (const CImg\textless{} t \textgreater{} &img) const  
    Subtraction operator.
  \item template<typename t >
    CImg\textless{} t \textgreater{} \& operator\textasciimacron{}= (const t value)  
    In-place multiplication operator.
  \item CImg\textless{} Tfloat \textgreater{} \& operator\textasciimacron{} (const char *const expression) const  
    In-place multiplication operator.
  \item template<typename t >
    CImg\textless{} t \textgreater{} \& operator\textasciimacron{}= (const CImg\textless{} t \textgreater{} &img) const  
    In-place multiplication operator.
  \item template<typename t >
    CImg\textless{} t \textgreater{} \& operator\textasciilast{} (const t value)  
    Multiplication operator.
  \item CImg\textless{} Tfloat \textgreater{} \& operator\textasciilast{} (const char *const expression) const  
    Multiplication operator.
  \item template<typename t >
    CImg\textless{} t \textgreater{} \& operator\textasciilast{}= (const CImg\textless{} t \textgreater{} &img) const  
    Multiplication operator.
  \item template<typename t >
    CImg\textless{} T \textgreater{} \& operator\textasciilast{} (const t value)  
    In-place division operator.
  \item CImg\textless{} Tfloat \textgreater{} \& operator\textasciilast{} (const char *const expression) const  
    In-place division operator.
  \item template<typename t >
    CImg\textless{} T \textgreater{} \& operator\textasciilast{}= (const CImg\textless{} t \textgreater{} &img) const  
    In-place division operator.
  \item template<typename t >
    CImg\textless{} t \textgreater{} \& operator\textasciilast{} (const t value)  
    Division operator.
  \item CImg\textless{} Tfloat \textgreater{} \& operator\textasciilast{} (const char *const expression) const  
    Division operator.
  \item template<typename t >
    CImg\textless{} t \textgreater{} \& operator\textasciilast{} (const CImg\textless{} t \textgreater{} &img) const  
    Division operator.
\end{itemize}
• 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.
• 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.
• 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.
• CImg< T > & (const t value) const
  Bitwise AND operator.
• CImg< T > & (const char * const expression) const
  Bitwise AND operator.
• template<typename t >
  CImg< T > & (const CImg< t > & img) const
  Bitwise AND operator.
• template<typename t >
  CImg< T > & (const t value)
  In-place bitwise OR operator.
• CImg< T > & (const char * const expression)
  In-place bitwise OR operator.
• template<typename t >
  CImg< T > & (const CImg< t > & img)
  In-place bitwise OR operator.
• CImg< T > (const t value) const
  Bitwise OR operator.
• CImg< T > (const char * const expression) const
  Bitwise OR operator.
• template<typename t >
  CImg< T > (const CImg< t > & img) const
  Bitwise OR operator.
• template<typename t >
  CImg< T > & (const t value)
  In-place bitwise XOR operator.
• CImg< T > & (const char * const expression)
  In-place bitwise XOR operator.
In-place bitwise XOR operator.

- `template<typename t> CImg<T>& operator^=(const CImg< &img)`
  - In-place bitwise XOR operator.

- `template<typename t> CImg<T> operator^ (const t value) const`
  - Bitwise XOR operator.

- `template<typename t> CImg<T> operator^ (const char *const expression) const`
  - Bitwise XOR operator.

In-place bitwise left shift operator.

- `template<typename t> CImg<T>& operator<=(const t value)`
  - In-place bitwise left shift operator.

- `template<typename t> CImg<T> operator<=(const char *const expression)`
  - In-place bitwise left shift operator.

- `template<typename t> CImg<T> operator<=(const CImg< &img)`
  - In-place bitwise left shift operator.

Bitwise right shift operator.

- `template<typename t> CImg<T> operator>=(const t value)`
  - In-place bitwise right shift operator.

- `template<typename t> CImg<T> operator>=(const char *const expression)`
  - In-place bitwise right shift operator.

- `template<typename t> CImg<T> operator>=(const CImg< &img)`
  - In-place bitwise right shift operator.

Bitwise inversion operator.

- `template<typename t> bool operator== (const t value)`
  - Test if all pixels of an image have the same value.

- `template<typename t> bool operator== (const char *const expression)`
  - Test if all pixel values of an image follow a specified expression.
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 Clmg< 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.

- ClmgList< 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.

- ulongT size () const
  Return the total number of pixel values.

- T * data ()
  Return a pointer to the first pixel value.

- const T * data () const
  Return a pointer to the first pixel value [const version].

- T * data (const unsigned int x, const unsigned int y=0, const unsigned int z=0, const unsigned int c=0)
  Return a pointer to a located pixel value.

- const T * data (const unsigned int x, const unsigned int y=0, const unsigned int z=0, const unsigned int c=0) const
  Return a pointer to a located pixel value [const version].

- longT offset (const int x, const int y=0, const int z=0, const int c=0) const
  Return the offset to a located pixel value, with respect to the beginning of the pixel buffer.

- iterator begin ()
  Return a Clmg< T >::iterator pointing to the first pixel value.

- const_iterator begin () const
  Return a Clmg< T >::iterator pointing to the first value of the pixel buffer [const version].

- iterator end ()
  Return a Clmg< T >::iterator pointing next to the last pixel value.

- const_iterator end () const
  Return a Clmg< T >::iterator pointing next to the last pixel value [const version].

- T & front ()
  Return a reference to the first pixel value.
• const T & front () const
  Return a reference to the first pixel value [const version].

• T & back ()
  Return a reference to the last pixel value.

• const T & back () const
  Return a reference to the last pixel value [const version].

• T & at (const int offset, const T &out_value)
  Access to a pixel value at a specified offset, using Dirichlet boundary conditions.

• T at (const int offset, const T &out_value) const
  Access to a pixel value at a specified offset, using Dirichlet boundary conditions [const version].

• T & at (const int offset)
  Access to a pixel value at a specified offset, using Neumann boundary conditions.

• const T & at (const int offset) const
  Access to a pixel value at a specified offset, using Neumann boundary conditions [const version].

• T & atX (const int x, const int y, const int z, const int c, const T &out_value)
  Access to a pixel value, using Dirichlet boundary conditions for the X-coordinate.

• T atX (const int x, const int y, const int z, const int c, const T &out_value) const
  Access to a pixel value, using Dirichlet boundary conditions for the X-coordinate [const version].

• T & atX (const int x, const int y=0, const int z=0, const int c=0)
  Access to a pixel value, using Neumann boundary conditions for the X-coordinate.

• const T & atX (const int x, const int y=0, const int z=0, const int c=0) const
  Access to a pixel value, using Neumann boundary conditions for the X-coordinate [const version].

• T & atXY (const int x, const int y, const int z, const int c, const T &out_value)
  Access to a pixel value, using Dirichlet boundary conditions for the X and Y-coordinates.

• T atXY (const int x, const int y, const int z, const int c, const T &out_value) const
  Access to a pixel value, using Dirichlet boundary conditions for the X and Y coordinates [const version].

• T & atXY (const int x, const int y, const int z=0, const int c=0)
  Access to a pixel value, using Neumann boundary conditions for the X and Y-coordinates.

• const T & atXY (const int x, const int y, const int z=0, const int c=0) const
  Access to a pixel value, using Neumann boundary conditions for the X and Y-coordinates [const version].

• T & atXYZ (const int x, const int y, const int z, const int c, const T &out_value)
  Access to a pixel value, using Dirichlet boundary conditions for the X,Y and Z-coordinates.

• T atXYZ (const int x, const int y, const int z, const int c, const T &out_value) const
  Access to a pixel value, using Dirichlet boundary conditions for the X,Y and Z-coordinates [const version].

• T & atXYZ (const int x, const int y, const int z, const int c)
  Access to a pixel value, using Neumann boundary conditions for the X,Y and Z-coordinates.

• const T & atXYZ (const int x, const int y, const int z, const int c) const
  Access to a pixel value, using Neumann boundary conditions [const version].

• Tfloat linear_atX (const float fx, const int y, const int z, const int c, const T &out_value) const
  Return pixel value, using linear interpolation and Dirichlet boundary conditions for the X-coordinate.

• Tfloat linear_atX (const float fx, const int y=0, const int z=0, const int c=0) const
  Return pixel value, using linear interpolation and Neumann boundary conditions for the X-coordinate.

• Tfloat linear_atX_p (const float fx, const int y=0, const int z=0, const int c=0) const
Return pixel value, using linear interpolation and periodic boundary conditions for the X-coordinate.

- Tfloat `linear_atXY (const float fx, const float fy, const int z, const int c, const T &out_value) const`
  
- Tfloat `linear_atXY (const float fx, const float fy, const int z=0, const int c=0) const`
  
- Tfloat `linear_atXY_p (const float fx, const float fy, const int z=0, const int c=0) const`
  
Return pixel value, using linear interpolation and Dirichlet boundary conditions for the X and Y-coordinates.

- Tfloat `linear_atXY (const float fx, const float fy, const int z, const int c, const T &out_value) const`
  
- Tfloat `linear_atXY (const float fx, const float fy, const int z=0, const int c=0) const`
  
- Tfloat `linear_atXY_p (const float fx, const float fy, const int z=0, const int c=0) const`
  
Return pixel value, using linear interpolation and Neumann boundary conditions for the X and Y-coordinates.

- Tfloat `linear_atXY (const float fx, const float fy, const int z, const int c, const T &out_value) const`
  
- Tfloat `linear_atXY (const float fx, const float fy, const int z=0, const int c=0) const`
  
- Tfloat `linear_atXY_p (const float fx, const float fy, const int z=0, const int c=0) const`
  
Return pixel value, using linear interpolation and Dirichlet boundary conditions for the X, Y, and Z-coordinates.

- Tfloat `linear_atXY (const float fx, const float fy, const int z, const int c, const T &out_value) const`
  
- Tfloat `linear_atXY (const float fx, const float fy, const int z=0, const int c=0) const`
  
- Tfloat `linear_atXY_p (const float fx, const float fy, const int z=0, const int c=0) const`
  
Return pixel value, using linear interpolation and Neumann boundary conditions for the X, Y, and Z-coordinates.

- Tfloat `linear_atXYZ (const float fx, const float fy, const float fz, const int c, const T &out_value) const`
  
- Tfloat `linear_atXYZ (const float fx, const float fy, const float fz=0, const int c=0) const`
  
- Tfloat `linear_atXYZ_p (const float fx, const float fy, const float fz=0, const int c=0) 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, const float fz, const int c, const T &out_value) const`
  
- Tfloat `linear_atXYZ (const float fx, const float fy, const float fz=0, const int c=0) const`
  
- Tfloat `linear_atXYZ_p (const float fx, const float fy, const float fz=0, const int c=0) const`
  
Return pixel value, using linear interpolation and Dirichlet boundary conditions for all X, Y, Z, and C-coordinates.

- Tfloat `linear_at XYZC (const float fx, const float fy, const float fz, const float fc, const T &out_value) const`
  
- Tfloat `linear_at XYZC (const float fx, const float fy, const float fz, const float fc=0) const`
  
- Tfloat `linear_at XYZC_p (const float fx, const float fy, const float fz, const float fc=0) const`
  
Return pixel value, using linear interpolation and Dirichlet 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`
  
- Tfloat `cubic_atX (const float fx, const int y=0, const int z=0, const int c=0) const`
  
- Tfloat `cubic_atX_p (const float fx, const int y=0, const int z=0, const int c=0) const`
  
Return pixel value, using cubic interpolation and Dirichlet boundary conditions for the X-coordinate.

- T `cubic_atX (const float fx, const int y, const int z, const int c, const T &out_value) const`
  
- T `cubic_atX (const float fx, const int y=0, const int z=0, const int c=0) const`
  
- T `cubic_atX_p (const float fx, const int y=0, const int z=0, const int c=0) const`
  
Return pixel value, using cubic interpolation and Neumann boundary conditions for the X-coordinate.

- T `cubic_atX (const float fx, const int y, const int z, const int c) const`
  
- T `cubic_atX (const float fx, const int y=0, const int z=0, const int c) const`
  
- T `cubic atX_p (const float fx, const int y=0, const int z=0, const int c) const`
  
Return pixel value, using cubic interpolation and Dirichlet boundary conditions for the X-coordinate.

- T `cubic atX_c (const float fx, const int y, const int z, const int c, const T &out_value) const`
  
- T `cubic atX_c (const float fx, const int y=0, const int z=0, const int c) const`
  
- T `cubic atX_p_c (const float fx, const int y, const int z, const int c) const`
  
Return pixel value, using cubic interpolation and Neumann boundary conditions for the X-coordinate.

- T `cubic atXY (const float fx, const float fy, const int z, const int c, const T &out_value) const`
  
- T `cubic atXY (const float fx, const float fy, const int z=0, const int c=0) const`
  
- T `cubic atXY_p (const float fx, const float fy, const int z=0, const int c=0) const`
  
Return pixel value, using cubic interpolation and Dirichlet boundary conditions for the X and Y-coordinates.

- T `cubic atXY (const float fx, const float fy, const int z, const int c, const T &out_value) const`
  
- T `cubic atXY (const float fx, const float fy, const int z=0, const int c=0) const`
  
- T `cubic atXY_p (const float fx, const float fy, const int z=0, const int c=0) const`
  
Return pixel value, using cubic interpolation and Neumann boundary conditions for the X and Y-coordinates.

- T `cubic atXY (const float fx, const float fy, const int z, const int c) const`
  
- T `cubic atXY (const float fx, const float fy, const int z=0, const int c) const`
  
- T `cubic atXY_p (const float fx, const float fy, const int z=0, const int c) const`
  
Return pixel value, using cubic interpolation and Dirichlet boundary conditions for the X and Y-coordinates.

- T `cubic atXYZ (const float fx, const float fy, const float fz, const int c, const T &out_value) const`
  
- T `cubic atXYZ (const float fx, const float fy, const float fz=0, const int c=0) const`
  
- T `cubic atXYZ_p (const float fx, const float fy, const float fz=0, const int c=0) const`
  
Return pixel value, using cubic interpolation and Dirichlet boundary conditions for the X, Y, and Z-coordinates.

- T `cubic atXYZ (const float fx, const float fy, const float fz, const int c, const T &out_value) const`
  
- T `cubic atXYZ (const float fx, const float fy, const float fz=0, const int c=0) const`
  
- T `cubic atXYZ_p (const float fx, const float fy, const float fz=0, const int c=0) const`
  
Return pixel value, using cubic interpolation and Neumann boundary conditions for the X, Y, and Z-coordinates.

- T `cubic atXYZ (const float fx, const float fy, const float fz, const int c) const`
  
- T `cubic atXYZ (const float fx, const float fy, const float fz=0, const int c) const`
  
- T `cubic atXYZ_p (const float fx, const float fy, const float fz=0, const int c) const`
  
Return pixel value, using cubic interpolation and Neumann boundary conditions for the X, Y, and Z-coordinates.
8.1 Clmg< T > Struct Template Reference

- T cubic_atXYZ_pc (const float fx, const float fy, const float fz, const int c) const
- Clmg< T > & set_linear_atX (const T &value, const float fx, const int y=0, const int z=0, const int c=0, const bool is_added=false)
  
  Set pixel value, using linear interpolation for the X-coordinates.
- Clmg< T > & set_linear_atXY (const T &value, const float fx, const float fy=0, const int z=0, const int c=0, const bool is_added=false)
  
  Set pixel value, using linear interpolation for the X and Y-coordinates.
- Clmg< T > & set_linear_atXYZ (const T &value, const float fx, const float fy=0, const float fz=0, const int c=0, const bool is_added=false)
  
  Set pixel value, using linear interpolation for the X, Y and Z-coordinates.
- Clmg< charT > value_string (const char separator=',', const unsigned int max_size=0, const char ∗ format=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 Clmg< t > &img) const
  
  Test if image width is equal to specified value.
- bool is_sameX (const ClmgDisplay &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 Clmg< t > &img) const
  
  Test if image height is equal to specified value.
- bool is_sameY (const ClmgDisplay &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 Clmg< t > &img) const
  
  Test if image depth is equal to specified value.
- bool is_sameY (const ClmgDisplay &disp) 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 Clmg< 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 Clmg< t &img) const
  Test if image width and height are the same as that of another image.

• bool is_sameXY (const ClmgDisplay &disp) const
  Test if image width and height are the same as that of an existing display window.

• bool is_sameXZ (const unsigned int size_x, const unsigned int size_z) const
  Test if image width and depth are equal to specified values.

• template<typename t >
  bool is_sameXZ (const Clmg< t &img) const
  Test if image width and depth are the same as 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 Clmg< 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 Clmg< 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 Clmg< 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 Clmg< 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 Clmg< 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 Clmg< 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 Clmg< 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 Clmg< 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 Clmg< 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 CImg< t >&img) 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) 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 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 full_check=true, char ∗const error_message=0) const
  Test if image instance represents a valid serialization of a 3D object.

Mathematical Functions

• Clmg< T > & sqr ()
  Compute the square value of each pixel value.
• Clmg< Tfloat > get_sqr () const
• Clmg< T > & sqrt ()
  Compute the square root of each pixel value.
• Clmg< Tfloat > get_sqrt () const
• Clmg< T > & exp ()
  Compute the exponential of each pixel value.
• Clmg< Tfloat > get_exp () const
• Clmg< T > & log ()
  Compute the logarithm of each pixel value.
• Clmg< Tfloat > get_log () const
• Clmg< T > & log2 ()
  Compute the base-2 logarithm of each pixel value.
• Clmg< Tfloat > get_log2 () const
• Clmg< T > & log10 ()
  Compute the base-10 logarithm of each pixel value.
• CImg< Tfloat > get_log10 () const
  Compute the absolute value of each pixel value.

• CImg< Tfloat > get_abs () const
• CImg< T > & abs ()
  Compute the sign of each pixel value.

• CImg< Tfloat > get_sign () const
• CImg< T > & sign ()
  Compute the cosine of each pixel value.

• CImg< Tfloat > get_cos () const
• CImg< T > & cos ()
  Compute the sine of each pixel value.

• CImg< Tfloat > get_sin () const
• CImg< T > & sin ()
  Compute the sinc of each pixel value.

• CImg< Tfloat > get_sinc () const
• CImg< T > & sinc ()
  Compute the tangent of each pixel value.

• CImg< Tfloat > get_tan () const
• CImg< T > & tan ()
  Compute the hyperbolic cosine of each pixel value.

• CImg< Tfloat > get_cosh () const
• CImg< T > & cosh ()
  Compute the hyperbolic sine of each pixel value.

• CImg< Tfloat > get_sinh () const
• CImg< T > & sinh ()
  Compute the hyperbolic tangent of each pixel value.

• CImg< Tfloat > get_tanh () const
• CImg< T > & tanh ()
  Compute the arccosine of each pixel value.

• CImg< Tfloat > get_acos () const
• CImg< T > & acos ()
  Compute the arcsine of each pixel value.

• CImg< Tfloat > get_asin () const
• CImg< T > & asin ()
  Compute the arctangent of each pixel value.

• CImg< Tfloat > get_atan () const
• CImg< T > & atan ()
  Compute the arctangent2 of each pixel value.

• CImg< Tfloat > get_atan2 (const CImg< T > &img) const
  Compute the arctangent2 of each pixel value.

• CImg< Tfloat > get_acosh () const
• CImg< T > & acosh ()
  Compute the hyperbolic arccosine of each pixel value.

• CImg< Tfloat > get_acos () const
• CImg< T > & acos ()
  Compute the hyperbolic arcsine of each pixel value.

• CImg< Tfloat > get_asinh () const
• CImg< T > & asinh ()
  Compute the hyperbolic arctangent of each pixel value.
• template<typename t >
  CImg<T> & mul (const CImg<T> &img)
  In-place pointwise multiplication.
• template<typename t >
  CImg<typename cimg::superset<T, t>::type>& get_mul (const CImg<t> &img) const
  In-place pointwise multiplication [new-instance version].
• template<typename t >
  CImg<T> & div (const CImg<T> &img)
  In-place pointwise division.
• template<typename t >
  CImg<typename cimg::superset<T, t>::type>& get_div (const CImg<t> &img) const
  In-place pointwise division [new-instance version].
• CImg<T> & pow (const double p)
  Raise each pixel value to a specified power.
• CImg<Tfloat> get_pow (const double p) const
  Raise each pixel value to a specified power [new-instance version].
• CImg<T> & pow (const char *const expression)
  Raise each pixel value to a power, specified from an expression.
• CImg<Tfloat> get_pow (const char *const expression) const
  Raise each pixel value to a power, specified from an expression [new-instance version].
• template<typename t >
  CImg<T> & 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> & 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].
• 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 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].
• \texttt{template<\text{typename } t>}
  \texttt{CImg< }T \texttt{>} \texttt{get\_ror (const CImg< }t \texttt{>} \&img) const
  \textit{Compute the bitwise right rotation of each pixel value [new-instance version].}

• \texttt{CImg< }T \texttt{>} \& \texttt{min (const }T \texttt{&value)}
  \textit{Pointwise min operator between instance image and a value.}

• \texttt{CImg< }T \texttt{>} \texttt{get\_min (const }T \texttt{&value) const}
  \textit{Pointwise min operator between instance image and a value [new-instance version].}

• \texttt{template<\text{typename } t>}
  \texttt{CImg< }T \texttt{>} \& \texttt{min (const CImg< }t \texttt{>} \&img)}
  \textit{Pointwise min operator between two images.}

• \texttt{template<\text{typename } t>}
  \texttt{CImg< }\text{typename cimg::superset< }T, t \text{>::type } \&img \texttt{)
  \textit{Pointwise min operator between two images [new-instance version].}

• \texttt{CImg< }T \texttt{>} \& \texttt{min (const char *const expression)}
  \textit{Pointwise min operator between an image and an expression.}

• \texttt{CImg< }T \texttt{>} \& \texttt{max (const }T \texttt{&value)}
  \textit{Pointwise max operator between instance image and a value.}

• \texttt{CImg< }T \texttt{>} \texttt{get\_max (const }T \texttt{&value) const}
  \textit{Pointwise max operator between instance image and a value [new-instance version].}

• \texttt{CImg< }T \texttt{>} \& \texttt{min (const char *const expression)}
  \textit{Pointwise max operator between an image and an expression.}

• \texttt{CImg< }T \texttt{>} \& \texttt{max (const char *const expression) const}
  \textit{Pointwise max operator between an image and an expression [new-instance version].}

• \texttt{T \& min ()}
  \textit{Return a reference to the minimum pixel value.}

• \texttt{const T \& min () const}
  \textit{Return a reference to the minimum pixel value [const version].}

• \texttt{T \& max ()}
  \textit{Return a reference to the maximum pixel value.}

• \texttt{const T \& max () const}
  \textit{Return a reference to the maximum pixel value [const version].}

• \texttt{template<\text{typename } t>}
  \texttt{T \& min\_max (t \&max\_val)}
  \textit{Return a reference to the minimum pixel value as well as the maximum pixel value.}

• \texttt{template<\text{typename } t>}
  \texttt{const T \& min\_max (t \&max\_val) const}
  \textit{Return a reference to the minimum pixel value as well as the maximum pixel value [const version].}

• \texttt{template<\text{typename } t>}
  \texttt{T \& max\_min (t \&min\_val)}
  \textit{Return a reference to the maximum pixel value as well as the minimum pixel value.}

• \texttt{template<\text{typename } t>}
  \texttt{const T \& max\_min (t \&min\_val) const}
  \textit{Return a reference to the maximum pixel value as well as the minimum pixel value [const version].}

• \texttt{T \texttt{kth\_smallest (const ulongT k) const}
Return the kth smallest pixel value.

- `T median()` const
  Return the median pixel value.

- `double product()` const
  Return the product of all the pixel values.

- `double sum()` const
  Return the sum of all the pixel values.

- `double mean()` const
  Return the average pixel value.

- `double variance(const unsigned int variance_method=1)` const
  Return the variance of the pixel values.

- `template<typename t>`
  `double variance_mean(const unsigned int variance_method, t &mean)` const
  Return the variance as well as the average of the pixel values.

- `double variance_noise(const unsigned int variance_method=2)` const
  Return estimated variance of the noise.

- `template<typename t>`
  `double MSE(const CImg<T> &img)` const
  Compute the MSE (Mean-Squared Error) between two images.

- `template<typename t>`
  `double PSNR(const CImg<T> &img, const double max_value=255)` const
  Compute the PSNR (Peak Signal-to-Noise Ratio) between two images.

- `double eval(const char *const expression, const double x=0, const double y=0, const double z=0, const double c=0, const CImgList<T> *const list_inputs=0, CImgList<T> *const list_outputs=0)`
  Evaluate math formula.

- `double eval(const char *const expression, const double x=0, const double y=0, const double z=0, const double c=0, const CImgList<T> *const list_inputs=0, CImgList<T> *const list_outputs=0) const`
  Evaluate math formula [const version].

- `template<typename t>`
  `void eval(CImg<T>&output, const char *const expression, const double x=0, const double y=0, const double z=0, const double c=0, const CImgList<T> *const list_inputs=0, CImgList<T> *const list_outputs=0)`
  Evaluate math formula.

- `template<typename t>`
  `void eval(CImg<T>&output, const char *const expression, const double x=0, const double y=0, const double z=0, const double c=0, const CImgList<T> *const list_inputs=0, CImgList<T> *const list_outputs=0) const`
  Evaluate math formula [const version].

- `CImg<T>eval(const char *const expression, const CImg<T>&xyzc, const CImgList<T> *const list_inputs=0, CImgList<T> *const list_outputs=0)`
  Evaluate math formula on a set of variables.

- `template<typename t>`
  `CImg<T>eval(const char *const expression, const CImg<T>&xyzc, const CImgList<T> *const list_inputs=0, CImgList<T> *const list_outputs=0) const`
  Evaluate math formula on a set of variables [const version].

- `CImg<T> &get_stats(const unsigned int variance_method=1)` const
  Compute statistics vector from the pixel values.

- `CImg<T> &stats(const unsigned int variance_method=1)`
  Compute statistics vector from the pixel values [in-place version].
Vector / Matrix Operations

- double magnitude (const int magnitude_type=2) const
  Compute norm of the image, viewed as a matrix.
- double trace () const
  Compute the trace of the image, viewed as a matrix.
- double det () const
  Compute the determinant of the image, viewed as a matrix.
- template<typename t >
  double dot (const CImg< t >&img) const
  Compute the dot product between instance and argument, viewed as matrices.
- CImg< T > get_vector_at (const unsigned int x, const unsigned int y=0, const unsigned int z=0) const
  Get vector-valued pixel located at specified position.
- CImg< T > get_matrix_at (const unsigned int x=0, const unsigned int y=0, const unsigned int z=0) const
  Get (square) matrix-valued pixel located at specified position.
- CImg< T > get_tensor_at (const unsigned int x, const unsigned int y=0, const unsigned int z=0) const
  Get tensor-valued pixel located at specified position.
- template<typename t >
  CImg< T >& set_vector_at (const CImg< t >&vec, const unsigned int x, const unsigned int y=0, const unsigned int z=0)
  Set vector-valued pixel at specified position.
- template<typename t >
  CImg< T >& set_matrix_at (const CImg< t >&mat, const unsigned int x=0, const unsigned int y=0, const unsigned int z=0)
  Set (square) matrix-valued pixel at specified position.
- template<typename t >
  CImg< T >& set_tensor_at (const CImg< t >&ten, const unsigned int x=0, const unsigned int y=0, const unsigned int z=0)
  Set tensor-valued pixel at specified position.
- CImg< T > & vector ()
  Unroll pixel values along axis $y$.
- CImg< T > & vector () const
  Unroll pixel values along axis $y$ [new-instance version].
- CImg< T > & matrix ()
  Resize image to become a scalar square matrix.
- CImg< T > & matrix () const
  Resize image to become a scalar square matrix [new-instance version].
- CImg< T > & tensor ()
  Resize image to become a symmetric tensor.
- CImg< T > & tensor () const
  Resize image to become a symmetric tensor [new-instance version].
- CImg< T > & diagonal ()
  Resize image to become a diagonal matrix.
- CImg< T > & diagonal () const
  Resize image to become a diagonal matrix [new-instance version].
- CImg< T > & identity_matrix ()
  Replace the image by an identity matrix.
- CImg< T > & identity_matrix () const
  Replace the image by an identity matrix [new-instance version].
- CImg< T > & sequence (const T &a0, const T &a1)
  Fill image with a linear sequence of values.
- CImg< T > & get_sequence (const T &a0, const T &a1) const
Fill image with a linear sequence of values [new-instance version].

- \texttt{CImg< T > \& transpose ()}
  - Transpose the image, viewed as a matrix.

- \texttt{CImg< T > \get\_transpose () const}
  - Transpose the image, viewed as a matrix [new-instance version].

- \texttt{template<typename t >}
  - \texttt{CImg< T > \& \cross (const CImg< t > \&img)}
    - Compute the cross product between two \(1 \times 3\) images, viewed as 3D vectors.

- \texttt{template<typename cimg::superset< T, t >::type >}
  - \texttt{get\_cross (const CImg< t >\&img)} const
    - Compute the cross product between two \(1 \times 3\) images, viewed as 3D vectors [new-instance version].

- \texttt{CImg< T > \& invert (const bool use\_LU=true)}
  - Invert the instance image, viewed as a matrix.

- \texttt{CImg< Tfloat > \get\_invert (const bool use\_LU=true) const}
  - Invert the instance image, viewed as a matrix [new-instance version].

- \texttt{CImg< T > \& pseudoinvert ()}
  - Compute the Moore-Penrose pseudo-inverse of the instance image, viewed as a matrix.

- \texttt{CImg< Tfloat > \get\_pseudoinvert () const}
  - Compute the Moore-Penrose pseudo-inverse of the instance image, viewed as a matrix [new-instance version].

- \texttt{template<typename t >}
  - \texttt{CImg< T > \& solve (const CImg< t >\&A)}
    - Solve a system of linear equations.

- \texttt{template<typename cimg::superset2< T, t, float >::type >}
  - \texttt{get\_solve (const CImg< t >\&A)} const
    - Solve a system of linear equations [new-instance version].

- \texttt{template<typename t >}
  - \texttt{CImg< T > \& solve\_tridiagonal (const CImg< t >\&A)}
    - Solve a tridiagonal system of linear equations.

- \texttt{template<typename cimg::superset2< T, t, float >::type >}
  - \texttt{get\_solve\_tridiagonal (const CImg< t >\&A)} const
    - Solve a tridiagonal system of linear equations [new-instance version].

- \texttt{template<typename t >}
  - \texttt{const CImg< T > \& \eigen (CImg< t >\&val, CImg< t >\&vec) const}
    - Compute eigenvalues and eigenvectors of the instance image, viewed as a matrix.

- \texttt{CImgList< Tfloat > \get\_eigen () const}
  - Compute eigenvalues and eigenvectors of the instance image, viewed as a matrix.

- \texttt{template<typename t >}
  - \texttt{const CImg< T > \& \symmetric\_eigen (CImg< t >\&val, CImg< t >\&vec) const}
    - Compute eigenvalues and eigenvectors of the instance image, viewed as a symmetric matrix.

- \texttt{CImgList< Tfloat > \get\_symmetric\_eigen () const}
  - Compute eigenvalues and eigenvectors of the instance image, viewed as a symmetric matrix.

- \texttt{template<typename t >}
  - \texttt{CImg< T > \& \sort (CImg< t >\&permutations, const bool is\_increasing=true)}
    - Sort pixel values and get sorting permutations.

- \texttt{template<typename t >}
  - \texttt{CImg< T > \get\_sort (CImg< t >\&permutations, const bool is\_increasing=true) const}
    - Sort pixel values and get sorting permutations [new-instance version].

- \texttt{CImg< T > \& \sort (const bool is\_increasing=true, const char axis=0)}
  - Sort pixel values.

- \texttt{CImg< T > \get\_sort (const bool is\_increasing=true, const char axis=0) const}
  - Sort pixel values [new-instance version].
• \textbf{template\textless t\textgreater{} const CImg\textless{}t\textgreater{} & SVD (CImg\textless{}t\textgreater{} &U, CImg\textless{}t\textgreater{} &S, CImg\textless{}t\textgreater{} &V, const bool sorting=true, const unsigned int max\_iteration=40, const float lambda=0) const}

Compute the SVD of the instance image, viewed as a general matrix.

• \textbf{CImgList\textless{}Tfloat\textgreater{} get\_SVD (const bool sorting=true, const unsigned int max\_iteration=40, const float lambda=0) const}

Compute the SVD of the instance image, viewed as a general matrix.

• \textbf{template\textless{}t\textgreater{} CImg\textless{}t\textgreater{} & dijkstra (const unsigned int starting\_node, const unsigned int ending\_node, CImg\textless{}t\textgreater{} &previous\_node)}

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

• \textbf{template\textless{}t\textgreater{} CImg\textless{}t\textgreater{} & get\_dijkstra (const unsigned int starting\_node, const unsigned int ending\_node, CImg\textless{}t\textgreater{} &previous\_node) const}

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

• \textbf{CImg\textless{}T\textgreater{} & dijkstra (const unsigned int starting\_node, const unsigned int ending\_node=size\_U)}

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

• \textbf{CImg\textless{}Tfloat\textgreater{} get\_dijkstra (const unsigned int starting\_node, const unsigned int ending\_node=size\_U) const}

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

• \textbf{template\textless{}t\textgreater{} template\textless{}tf, t\textgreater{} static CImg\textless{}T\textgreater{} dijkstra (const tf &distance, const unsigned int nb\_nodes, const unsigned int starting\_node, const unsigned int ending\_node, CImg\textless{}t\textgreater{} &previous\_node)}

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

• \textbf{template\textless{}tf, t\textgreater{} static CImg\textless{}T\textgreater{} dijkstra (const tf &distance, const unsigned int nb\_nodes, const unsigned int starting\_node, const unsigned int ending\_node=size\_U)}

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

• \textbf{static CImg\textless{}T\textgreater{} string (const char* const str, const bool is\_last\_zero=true, const bool is\_shared=false)}

Return an image containing the Ascii codes of the specified string.

• \textbf{static CImg\textless{}T\textgreater{} T > vector (const T &a0)}

Return a 1x1 image containing specified value.

• \textbf{static CImg\textless{}T\textgreater{} T > vector (const T &a0, const T &a1)}

Return a 1x2 image containing specified values.

• \textbf{static CImg\textless{}T\textgreater{} T > vector (const T &a0, const T &a1, const T &a2)}

Return a 1x3 image containing specified values.

• \textbf{static CImg\textless{}T\textgreater{} T > vector (const T &a0, const T &a1, const T &a2, const T &a3)}

Return a 1x4 image containing specified values.

• \textbf{static CImg\textless{}T\textgreater{} T > vector (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4)}

Return a 1x5 image containing specified values.

• \textbf{static CImg\textless{}T\textgreater{} T > vector (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5)}

Return a 1x6 image containing specified values.

• \textbf{static CImg\textless{}T\textgreater{} 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 1x7 image containing specified values.

• \textbf{static CImg\textless{}T\textgreater{} 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 1x8 image containing specified values.

• \textbf{static CImg\textless{}T\textgreater{} 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 1x9 image containing specified values.

• \textbf{static CImg\textless{}T\textgreater{} 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 1x10 image containing specified values.
8.1 Clmg< T > Struct Template Reference

- `static Clmg< 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 1x1 image containing specified values.

- `static Clmg< 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 1x2 image containing specified values.

- `static Clmg< 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 1x3 image containing specified values.

- `static Clmg< 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 1x4 image containing specified values.

- `static Clmg< T > matrix (const T &a0)`
  
  Return a 1x1 matrix containing specified coefficients.

- `static Clmg< T > matrix (const T &a0, const T &a1, const T &a2, const T &a3)`
  
  Return a 2x2 matrix containing specified coefficients.

- `static Clmg< T > matrix (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7, const T &a8)`
  
  Return a 3x3 matrix containing specified coefficients.

- `static Clmg< T > matrix (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7, const T &a8, const T &a9, const T &a10, const T &a11, const T &a12, const T &a13, const T &a14, const T &a15)`
  
  Return a 4x4 matrix containing specified coefficients.

- `static Clmg< T > tensor (const T &a0)`
  
  Return a 1x1 symmetric matrix containing specified coefficients.

- `static Clmg< T > tensor (const T &a0, const T &a1, const T &a2, const T &a3)`
  
  Return a 2x2 symmetric matrix tensor containing specified coefficients.

- `static Clmg< T > diagonal (const T &a0)`
  
  Return a 1x1 diagonal matrix containing specified coefficients.

- `static Clmg< T > diagonal (const T &a0, const T &a1)`
  
  Return a 2x2 diagonal matrix containing specified coefficients.
• static 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 from an { axis + angle } or a quaternion.

Value Manipulation

• CImg<T> & fill (const T &val)

  Fill all pixel values with specified value.

• CImg<T> get_fill (const T &val) const

  Fill all pixel values with specified value [new-instance version].

• CImg<T> & fill (const T &val0, const T &val1)

  Fill sequentially all pixel values with specified values.

• CImg<T> get_fill (const T &val0, const T &val1) const

  Fill sequentially all pixel values with specified values [new-instance version].

• CImg<T> & fill (const T &val0, const T &val1, const T &val2)

  Fill sequentially all pixel values with specified values [new-instance version].

• CImg<T> get_fill (const T &val0, const T &val1, const T &val2) const

  Fill sequentially all pixel values with specified values [new-instance version].

• CImg<T> & fill (const T &val0, const T &val1, const T &val2, const T &val3)

  Fill sequentially all pixel values with specified values [new-instance version].

• CImg<T> get_fill (const T &val0, const T &val1, const T &val2, const T &val3) const

  Fill sequentially all pixel values with specified values [new-instance version].

• CImg<T> & fill (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4)

  Fill sequentially all pixel values with specified values [new-instance version].

• CImg<T> get_fill (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4) const

  Fill sequentially all pixel values with specified values [new-instance version].

• 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 specified values [new-instance version].

• CImg<T> get_fill (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5) const

  Fill sequentially all pixel values with specified values [new-instance version].

• 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 all pixel values with specified values [new-instance version].

• CImg<T> get_fill (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6) const

  Fill sequentially all pixel values with specified values [new-instance version].

• 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 all pixel values with specified values [overloading].

• 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

  Fill sequentially all pixel values with specified values [new-instance version].

• 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 all pixel values with specified values [overloading].

• 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

  Fill sequentially all pixel values with specified values [new-instance version].

• 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 all pixel values with specified values [overloading].
• `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
  *Fill sequentially all pixel values with specified values [new-instance version].*

• `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
  *Fill sequentially all pixel values with specified values [overloading].*

• `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
  *Fill sequentially all pixel values with specified values [new-instance version].*

• `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
  *Fill sequentially all pixel values with specified values [overloading].*

• `CImg<T> get_fill` (const `T` &val0, const `T` &val1, const `T` &val2, const `T` &val3, const `T` &val4, const `T` &val5, const `T` &val6, const `T` &val7, const `T` &val8, const `T` &val9, const `T` &val10) const
  *Fill sequentially all pixel values with specified values [new-instance version].*

• `CImg<T> & fill` (const `T` &val0, const `T` &val1, const `T` &val2, const `T` &val3, const `T` &val4, const `T` &val5, const `T` &val6, const `T` &val7, const `T` &val8, const `T` &val9, const `T` &val10, const `T` &val11) const
  *Fill sequentially all pixel values with specified values [overloading].*

• `CImg<T> get_fill` (const `T` &val0, const `T` &val1, const `T` &val2, const `T` &val3, const `T` &val4, const `T` &val5, const `T` &val6, const `T` &val7, const `T` &val8, const `T` &val9, const `T` &val10, const `T` &val11, const `T` &val12) const
  *Fill sequentially all pixel values with specified values [new-instance version].*

• `CImg<T> & fill` (const `T` &val0, const `T` &val1, const `T` &val2, const `T` &val3, const `T` &val4, const `T` &val5, const `T` &val6, const `T` &val7, const `T` &val8, const `T` &val9, const `T` &val10, const `T` &val11, const `T` &val12, const `T` &val13) const
  *Fill sequentially all pixel values with specified values [overloading].*

• `CImg<T> get_fill` (const `T` &val0, const `T` &val1, const `T` &val2, const `T` &val3, const `T` &val4, const `T` &val5, const `T` &val6, const `T` &val7, const `T` &val8, const `T` &val9, const `T` &val10, const `T` &val11, const `T` &val12, const `T` &val13) const
  *Fill sequentially all pixel values with specified values [new-instance version].*

• `CImg<T> & fill` (const `T` &val0, const `T` &val1, const `T` &val2, const `T` &val3, const `T` &val4, const `T` &val5, const `T` &val6, const `T` &val7, const `T` &val8, const `T` &val9, const `T` &val10, const `T` &val11, const `T` &val12, const `T` &val13, const `T` &val14) const
  *Fill sequentially all pixel values with specified values [overloading].*

• `CImg<T> get_fill` (const `T` &val0, const `T` &val1, const `T` &val2, const `T` &val3, const `T` &val4, const `T` &val5, const `T` &val6, const `T` &val7, const `T` &val8, const `T` &val9, const `T` &val10, const `T` &val11, const `T` &val12, const `T` &val13, const `T` &val14, const `T` &val15) const
  *Fill sequentially all pixel values with specified values [new-instance version].*

• `CImg<T> & fill` (const `T` &val0, const `T` &val1, const `T` &val2, const `T` &val3, const `T` &val4, const `T` &val5, const `T` &val6, const `T` &val7, const `T` &val8, const `T` &val9, const `T` &val10, const `T` &val11, const `T` &val12, const `T` &val13, const `T` &val14, const `T` &val15) const
  *Fill sequentially all pixel values with specified values [overloading].*

• `CImg<T> get_fill` (const `T` &val0, const `T` &val1, const `T` &val2, const `T` &val3, const `T` &val4, const `T` &val5, const `T` &val6, const `T` &val7, const `T` &val8, const `T` &val9, const `T` &val10, const `T` &val11, const `T` &val12, const `T` &val13, const `T` &val14, const `T` &val15) const
  *Fill sequentially all pixel values with specified values [new-instance version].*

• `CImg<T> & fill` (const char *const expression, const bool repeat_values, const bool allow_formula=true, const `ClmgList<T>` *const list_inputs=0, `ClmgList<T>` *const list_outputs=0) const
  *Fill sequentially all pixel values with specified values [overloading].*
Fill sequentially pixel values according to a given expression.

- \texttt{Clmg\langle T \rangle \ get\_fill} (const char \texttt{*const expression, const bool repeat\_values, const bool allow\_formula=1, const ClmgList\langle T \rangle \texttt{*const list\_inputs=0, ClmgList\langle T \rangle \texttt{*const list\_outputs=0) const}

  Fill sequentially pixel values according to a given expression [new-instance version].

- \texttt{template<typename t > CImg\langle T \rangle\ get\_fill} (const char \texttt{*const expression, const bool repeat\_values, const bool allow\_formula=1, const ClmgList\langle T \rangle \texttt{*const list\_inputs=0, ClmgList\langle T \rangle \texttt{*const list\_outputs=0) const}

  Fill sequentially pixel values according to the values found in another image.

- \texttt{template<typename t > CImg\langle T \rangle\ get\_fill} (const CImg\langle t \rangle\& values, const bool repeat\_values=\texttt{true})

  Fill sequentially pixel values according to the values found in another image [new-instance version].

- \texttt{template<typename t > CImg\langle T \rangle\ get\_fill\_X} (const unsigned int y, const unsigned int z, const unsigned int c, const int a0,...)

  Fill pixel values along the X-axis at a specified pixel position.

- \texttt{template<typename t > CImg\langle T \rangle\ get\_fill\_Y} (const unsigned int x, const unsigned int z, const unsigned int c, const double a0,...)

  Fill pixel values along the Y-axis at a specified pixel position.

- \texttt{template<typename t > CImg\langle T \rangle\ get\_fill\_Z} (const unsigned int x, const unsigned int y, const unsigned int c, const double a0,...)

  Fill pixel values along the Z-axis at a specified pixel position.

- \texttt{template<typename t > CImg\langle T \rangle\ get\_fill\_C} (const unsigned int x, const unsigned int y, const unsigned int z, const double a0,...)

  Fill pixel values along the C-axis at a specified pixel position.

- \texttt{template<typename t > CImg\langle T \rangle\ get\_discard} (const CImg\langle t \rangle\& values, const char axis=0)

  Discard specified sequence of values in the image buffer, along a specific axis.

- \texttt{template<typename t > CImg\langle T \rangle\ get\_discard\_X} (const unsigned int y, const unsigned int z, const unsigned int c, const int a0,...)

  Discard neighboring duplicates in the image buffer, along the specified axis.

- \texttt{template<typename t > CImg\langle T \rangle\ get\_discard\_Y} (const unsigned int x, const unsigned int z, const unsigned int c, const double a0,...)

  Discard neighboring duplicates in the image buffer, along the specified axis [new-instance version].

- \texttt{template<typename t > CImg\langle T \rangle\ get\_discard\_Z} (const unsigned int x, const unsigned int y, const unsigned int c, const double a0,...)

  Discard neighboring duplicates in the image buffer, along the specified axis [overloading].

- \texttt{template<typename t > CImg\langle T \rangle\ get\_discard\_C} (const unsigned int x, const unsigned int y, const unsigned int z, const double a0,...)

  Discard neighboring duplicates in the image buffer, along the specified axis [overloading].

- \texttt{CImg\langle T \rangle\ invert\_endianess} ()

  Invert endianness of all pixel values.

- \texttt{CImg\langle T \rangle\ invert\_endianess\_X} () const

  Invert endianness of all pixel values [new-instance version].

- \texttt{CImg\langle T \rangle\ rand} (const T \texttt{*val\_min, const T *val\_max})

  Fill image with random values in specified range.

- \texttt{CImg\langle T \rangle\ rand\_X} (const T \texttt{*val\_min, const T *val\_max}) const

  Fill image with random values in specified range [new-instance version].

- \texttt{CImg\langle T \rangle\ round} (const double y=1, const int rounding\_type=0)

  Round pixel values.

- \texttt{CImg\langle T \rangle\ round\_X} (const double y=1, const unsigned int rounding\_type=0) const

  Round pixel values [new-instance version].

- \texttt{CImg\langle T \rangle\ noise} (const double sigma, const unsigned int noise\_type=0)

  Add random noise to pixel values.

- \texttt{CImg\langle T \rangle\ noise\_X} (const double sigma, const unsigned int noise\_type=0) const

  Add random noise to pixel values [new-instance version].
- `CImg< T > & normalize (const T &min_value, const T &max_value)`
  Linearly normalize pixel values.
- `CImg< Tfloat > get_normalize (const T &min_value, const T &max_value) const`
  Linearly normalize pixel values [new-instance version].
- `CImg< T > & normalize ()`
  Normalize multi-valued pixels of the image instance, with respect to their L2-norm.
- `CImg< Tfloat > get_normalize () const`
  Normalize multi-valued pixels of the image instance [new-instance version].
- `CImg< T > & norm (const int norm_type=2)`
  Compute Lp-norm of each multi-valued pixel of the image instance.
- `CImg< Tfloat > get_norm (const int norm_type=2) const`
  Compute L2-norm of each multi-valued pixel of the image instance [new-instance version].
- `CImg< T > & cut (const T &min_value, const T &max_value)`
  Cut pixel values in specified range.
- `CImg< T > get_cut (const T &min_value, const T &max_value) const`
  Cut pixel values in specified range [new-instance version].
- `CImg< T > & quantize (const unsigned int nb_levels, const bool keep_range=true)`
  Uniformly quantize pixel values.
- `CImg< T > get_quantize (const unsigned int n, const bool keep_range=true) const`
  Uniformly quantize pixel values [new-instance version].
- `CImg< T > & threshold (const T &value, const bool soft_threshold=false, const bool strict_threshold=false)`
  Threshold pixel values.
- `CImg< T > get_threshold (const T &value, const bool soft_threshold=false, const bool strict_threshold=false) const`
  Threshold pixel values [new-instance version].
- `CImg< T > & histogram (const unsigned int nb_levels, const T &min_value, const T &max_value)`
  Compute the histogram of pixel values.
- `CImg< T > & histogram (const unsigned int nb_levels)`
  Compute the histogram of pixel values [overloading].
- `CImg< ulongT > get_histogram (const unsigned int nb_levels, const T &min_value, const T &max_value) const`
  Compute the histogram of pixel values [new-instance version].
- `CImg< ulongT > get_histogram (const unsigned int nb_levels)`
  Compute the histogram of pixel values [new-instance version].
- `CImg< T > & equalize (const unsigned int nb_levels, const T &min_value, const T &max_value)`
  Equalize histogram of pixel values.
- `CImg< T > & equalize (const unsigned int nb_levels)`
  Equalize histogram of pixel values [overloading].
- `CImg< T > & get_equalize (const unsigned int nb_levels, const T &val_min, const T &val_max)`
  Equalize histogram of pixel values [new-instance version].
- `CImg< T > & get_equalize (const unsigned int nblevels)`
  Equalize histogram of pixel values [new-instance version].
- `template<typename t > CImg< t > & index (const CImg< t > &colormap, const float dithering=1, const bool map_indexes=false)`
  Index multi-valued pixels regarding to a specified colormap.
- `template<typename t > CImg< t >::Tuint get_index (const CImg< t > &colormap, const float dithering=1, const bool map_indexes=true) const`
  Index multi-valued pixels regarding to a specified colormap [new-instance version].
- `template<typename t > CImg< t > & map (const CImg< t > &colormap, const unsigned int boundary_conditions=0)`
Map predefined colormap on the scalar (indexed) image instance.

- \texttt{template<typename t>}
  \texttt{CImg\<t\>\ get\_map\ (const CImg\<t\>& colormap, const unsigned int boundary\_conditions=0) const}
  Map predefined colormap on the scalar (indexed) image instance \texttt{[new-instance version]}.

- \texttt{CImg\<T\>\& label\ (const bool is\_high\_connectivity=false, const Tfloat tolerance=0) const}
  Label connected components.

- \texttt{CImg\<ulongT\>\ get\_label\ (const bool is\_high\_connectivity=false, const Tfloat tolerance=0) const}
  Label connected components \texttt{[new-instance version]}.

Color Base Management

- \texttt{CImg\<T\>\& sRGBtoRGB ()}
  Convert pixel values from sRGB to RGB color spaces.

- \texttt{CImg\<Tfloat\>\ get\_sRGBtoRGB () const}
  Convert pixel values from sRGB to RGB color spaces \texttt{[new-instance version]}.

- \texttt{CImg\<T\>\& RGBtosRGB ()}
  Convert pixel values from RGB to sRGB color spaces.

- \texttt{CImg\<Tfloat\>\ get\_RGBtosRGB () const}
  Convert pixel values from RGB to sRGB color spaces \texttt{[new-instance version]}.

- \texttt{CImg\<T\>\& RGBtoHSI ()}
  Convert pixel values from RGB to HSI color spaces.

- \texttt{CImg\<Tfloat\>\ get\_RGBtoHSI () const}
  Convert pixel values from RGB to HSI color spaces \texttt{[new-instance version]}.

- \texttt{CImg\<T\>\& RGBtoYCbCr ()}
  Convert pixel values from RGB to YCbCr color spaces.

- \texttt{CImg\<Tuchar\>\ get\_HSVtoRGB () const}
  Convert pixel values from HSV to RGB color spaces \texttt{[new-instance version]}.

- \texttt{CImg\<T\>\& HSLtoRGB ()}
  Convert pixel values from HSL to RGB color spaces.

- \texttt{CImg\<Tuchar\>\ get\_HSLtoRGB () const}
  Convert pixel values from HSL to RGB color spaces \texttt{[new-instance version]}.

- \texttt{CImg\<T\>\& RGBtoHSL ()}
  Convert pixel values from RGB to HSL color spaces.

- \texttt{CImg\<T\>\& HSLtoRGB ()}
  Convert pixel values from HSL to RGB color spaces.

- \texttt{CImg\<T\>\& RGBtoHSV ()}
  Convert pixel values from RGB to HSV color spaces.

- \texttt{CImg\<T\>\& HSVtoRGB ()}
  Convert pixel values from HSV to RGB color spaces.

- \texttt{CImg\<T\>\& HSVtoRGB ()}
  Convert pixel values from HSV to RGB color spaces \texttt{[new-instance version]}.
Convert pixel values from RGB to YCbCr color spaces.

- `CImg< Tuchar > get_RGBtoYCbCr () const
  Convert pixel values from RGB to YCbCr color spaces [new-instance version].

- `CImg< T > & YCbCrtoRGB ()
  Convert pixel values from YCbCr to RGB color spaces.

- `CImg< Tuchar > get_YCbCrtoRGB () const
  Convert pixel values from YCbCr to RGB color spaces [new-instance version].

- `CImg< T > & RGBtoYUV ()
  Convert pixel values from RGB to YUV color spaces.

- `CImg< Tfloat > get_RGBtoYUV () const
  Convert pixel values from RGB to YUV color spaces [new-instance version].

- `CImg< T > & YUVtoRGB ()
  Convert pixel values from YUV to RGB color spaces.

- `CImg< Tuchar > get_YUVtoRGB () const
  Convert pixel values from YUV to RGB color spaces [new-instance version].

- `CImg< T > & CMYtoRGB ()
  Convert pixel values from CMY to RGB color spaces.

- `CImg< Tuchar > get_RGBtoCMY () const
  Convert pixel values from RGB to CMY color spaces [new-instance version].

- `CImg< T > & CMYtoCMYK ()
  Convert pixel values from CMY to CMYK color spaces.

- `CImg< Tuchar > get_CMYtoCMYK () const
  Convert pixel values from CMY to CMYK color spaces [new-instance version].

- `CImg< T > & CMYtoRGB ()
  Convert pixel values from CMY to RGB color spaces.

- `CImg< Tuchar > get_CMYtoRGB () const
  Convert pixel values from CMY to RGB color spaces [new-instance version].

- `CImg< T > & CMYtoCMYK ()
  Convert pixel values from CMY to CMYK color spaces.

- `CImg< Tuchar > get_CMYtoCMYK () const
  Convert pixel values from CMY to CMYK color spaces [new-instance version].

- `CImg< Tfloat > get_CMYtoCMYK () const
  Convert pixel values from CMY to CMYK color spaces [new-instance version].

- `CImg< T > & RGBtoXYZ (const bool use_D65=true)
  Convert pixel values from RGB to XYZ color spaces.

- `CImg< Tfloat > get_RGBtoXYZ (const bool use_D65=true) const
  Convert pixel values from RGB to XYZ color spaces [new-instance version].

- `CImg< T > & XYZtoRGB (const bool use_D65=true)
  Convert pixel values from XYZ to RGB color spaces.

- `CImg< Tfloat > get_XYZtoRGB (const bool use_D65=true) const
  Convert pixel values from XYZ to RGB color spaces [new-instance version].

- `CImg< T > & XYZtoLab (const bool use_D65=true)
  Convert pixel values from XYZ to Lab color spaces.

- `CImg< Tfloat > get_XYZtoLab (const bool use_D65=true) const
  Convert pixel values from XYZ to Lab color spaces [new-instance version].

- `CImg< T > & LabtoXYZ (const bool use_D65=true)
  Convert pixel values from Lab to XYZ color spaces.

- `CImg< Tfloat > get_LabtoXYZ (const bool use_D65=true) const
  Convert pixel values from Lab to XYZ color spaces [new-instance version].

- `CImg< T > & XYZtoxyY ()
  Convert pixel values from XYZ to xyY color spaces.

- `CImg< Tfloat > get_XYZtoxyY () const
  Convert pixel values from XYZ to xyY color spaces [new-instance version].
• \texttt{CImg\<\textbackslash T\> \& xyYtoXYZ ()}
  
  \textit{Convert pixel values from xyY pixels to XYZ color spaces.}

• \texttt{CImg\<\textbackslash Tfloat\> get\_xyYtoXYZ () const}
  
  \textit{Convert pixel values from xyY pixels to XYZ color spaces [new-instance version].}

• \texttt{CImg\<\textbackslash T\> \& \& RGBtoLab (const bool use\_D65=true)}
  
  \textit{Convert pixel values from RGB to Lab color spaces.}

• \texttt{CImg\<\textbackslash Tfloat\> get\_RGBtoLab (const bool use\_D65=true) const}
  
  \textit{Convert pixel values from RGB to Lab color spaces [new-instance version].}

• \texttt{CImg\<\textbackslash T\> \& LabtoRGB (const bool use\_D65=true)}
  
  \textit{Convert pixel values from Lab to RGB color spaces.}

• \texttt{CImg\<\textbackslash Tchar\> get\_LabtoRGB (const bool use\_D65=true) const}
  
  \textit{Convert pixel values from Lab to RGB color spaces [new-instance version].}

• \texttt{CImg\<\textbackslash Tfloat\> get\_RGBtoxyY (const bool use\_D65=true) const}
  
  \textit{Convert pixel values from RGB to xyY color spaces [new-instance version].}

• \texttt{CImg\<\textbackslash T\> \& xyYtoRGB (const bool use\_D65=true)}
  
  \textit{Convert pixel values from xyY to RGB color spaces.}

• \texttt{CImg\<\textbackslash Tuchar\> get\_xyYtoRGB (const bool use\_D65=true) const}
  
  \textit{Convert pixel values from xyY to RGB color spaces [new-instance version].}

• \texttt{static const CImg\<\textbackslash Tuchar\> \& default\_LUT256 ()}
  
  \textit{Return colormap "default", containing 256 colors entries in RGB.}

• \texttt{static const CImg\<\textbackslash Tuchar\> \& HSV\_LUT256 ()}
  
  \textit{Return colormap "HSV", containing 256 colors entries in RGB.}

• \texttt{static const CImg\<\textbackslash Tuchar\> \& lines\_LUT256 ()}
  
  \textit{Return colormap "lines", containing 256 colors entries in RGB.}

• \texttt{static const CImg\<\textbackslash Tuchar\> \& hot\_LUT256 ()}
  
  \textit{Return colormap "hot", containing 256 colors entries in RGB.}

• \texttt{static const CImg\<\textbackslash Tuchar\> \& cool\_LUT256 ()}
  
  \textit{Return colormap "cool", containing 256 colors entries in RGB.}

• \texttt{static const CImg\<\textbackslash Tuchar\> \& jet\_LUT256 ()}
  
  \textit{Return colormap "jet", containing 256 colors entries in RGB.}

• \texttt{static const CImg\<\textbackslash Tuchar\> \& flag\_LUT256 ()}
  
  \textit{Return colormap "flag", containing 256 colors entries in RGB.}

• \texttt{static const CImg\<\textbackslash Tuchar\> \& cube\_LUT256 ()}
  
  \textit{Return colormap "cube", containing 256 colors entries in RGB.}
Geometric / Spatial Manipulation

- **CImg** & **resize** (const int _size_\_x, const int _size_\_y=100, const int _size_\_z=100, const int _size_\_c=100, const int _interpolation_type=1, const unsigned int _boundary_conditions=0, const float _centering_\_x=0, const float _centering_\_y=0, const float _centering_\_z=0, const float _centering_\_c=0)

  Resize image to new dimensions.

- **CImg** & **get_resize** (const int _size_\_x, const int _size_\_y=100, const int _size_\_z=100, const int _size_\_c=100, const int _interpolation_type=1, const unsigned int _boundary_conditions=0, const float _centering_\_x=0, const float _centering_\_y=0, const float _centering_\_z=0, const float _centering_\_c=0) const

  Resize image to new dimensions [new-instance version].

-.template<typename t >

  - **CImg** & **resize** (const CImg< t > &src, const int _interpolation_type=1, const unsigned int _boundary_conditions=0, const float _centering_\_x=0, const float _centering_\_y=0, const float _centering_\_z=0, const float _centering_\_c=0)

  Resize image to dimensions of another image.

- template<typename t >

  - **CImg** & **get_resize** (const CImg< t > &src, const int _interpolation_type=1, const unsigned int _boundary_conditions=0, const float _centering_\_x=0, const float _centering_\_y=0, const float _centering_\_z=0, const float _centering_\_c=0) const

  Resize image to dimensions of another image [new-instance version].

- **CImg** & **resize** (const CImgDisplay &disp, const int _interpolation_type=1, const unsigned int _boundary_conditions=0, const float _centering_\_x=0, const float _centering_\_y=0, const float _centering_\_z=0, const float _centering_\_c=0)

  Resize image to dimensions of a display window.

- **CImg** & **get_resize** (const CImgDisplay &disp, const int _interpolation_type=1, const unsigned int _boundary_conditions=0, const float _centering_\_x=0, const float _centering_\_y=0, const float _centering_\_z=0, const float _centering_\_c=0) const

  Resize image to dimensions of a display window [new-instance version].

- **CImg** & **resize_halfXY** ()

  Resize image to half-size along XY axes, using an optimized filter.

- **CImg** & **get_resize_halfXY** () const

  Resize image to half-size along XY axes, using an optimized filter [new-instance version].

- **CImg** & **resize_doubleXY** ()

  Resize image to double-size, using the Scale2X algorithm.

- **CImg** & **get_resize_doubleXY** () const

  Resize image to double-size, using the Scale2X algorithm [new-instance version].

- **CImg** & **resize_tripleXY** ()

  Resize image to triple-size, using the Scale3X algorithm.

- **CImg** & **get_resize_tripleXY** () const

  Resize image to triple-size, using the Scale3X algorithm [new-instance version].

- **CImg** & **mirror** (const char _axis)

  Mirror image content along specified axis.

- **CImg** & **get_mirror** (const char ∗const _axes) const

  Mirror image content along specified axes.

- **CImg** & **shift** (const int _delta_\_x, const int _delta_\_y=0, const int _delta_\_z=0, const int _delta_\_c=0, const unsigned int _boundary_conditions=0)

  Shift image content.

- **CImg** & **get_shift** (const int _delta_\_x, const int _delta_\_y=0, const int _delta_\_z=0, const int _delta_\_c=0, const unsigned int _boundary_conditions=0) const
Shift image content [new-instance version].

- `CImg< T > & permute_axes (const char *const axes_order)`
  Permutes axes order.

- `CImg< T > & get_permute_axes (const char *const axes_order) const`
  Permutes axes order [new-instance version].

- `CImg< T > & unroll (const char axis)`
  Unrolls pixel values along the specified axis.

- `CImg< T > & get_unroll (const char axis) const`
  Unrolls pixel values along the specified axis [new-instance version].

- `CImg< T > & rotate (const float angle, const unsigned int interpolation=1, const unsigned int boundary_conditions=0)`
  Rotate image with an arbitrary angle.

- `CImg< T > & get_rotate (const float angle, const unsigned int interpolation=1, const unsigned int boundary_conditions=0) const`
  Rotate image with an arbitrary angle [new-instance version].

- `CImg< T > & rotate (const float u, const float v, const float w, const float angle, const unsigned int interpolation, const unsigned int boundary_conditions)`
  Rotate volumetric image with an arbitrary angle and axis.

- `CImg< T > & get_rotate (const float u, const float v, const float w, const float angle, const unsigned int interpolation, const unsigned int boundary_conditions) const`
  Rotate volumetric image with an arbitrary angle and axis [new-instance version].

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

- `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 unsigned int boundary_conditions=0) const`
  Crop image region [new-instance version].

- `CImg< T > & warp (const CImg< t > &p_warp, const unsigned int mode=0, const unsigned int interpolation=1, const unsigned int boundary_conditions=0)`
  Warps image content by a warping field.

- `template<typename t >
  CImg< T > & warp (const CImg< t > &p_warp, const unsigned int mode=0, const unsigned int interpolation=1, const unsigned int boundary_conditions=0)`
  Warps image content by a warping field [new-instance version].

- `CImg< T > & get_warp (const CImg< t > &p_warp, const unsigned int mode=0, const unsigned int interpolation=1, const unsigned int boundary_conditions=0) const`
  Warps image content by a warping field [new-instance version].

- `CImg< T > & get_projections2d (const unsigned int x0, const unsigned int y0, const unsigned int z0)`
  Generates a 2D representation of a 3D image, with XY, XZ and YZ views.

- `CImg< T > & projections2d (const unsigned int x0, const unsigned int y0, const unsigned int z0)`
  Generates a 2D representation of a 3D image, with XY, XZ and YZ views [in-place version].

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

- `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 unsigned int boundary_conditions=0) const`
  Crops image region [new-instance version].

- `CImg< T > & crop (const int x0, const int y0, const int z0, const int x1, const int y1, const int z1, const unsigned int boundary_conditions=0)`
  Crops image region [new-instance version].
Crop image region [overloading].

- \texttt{CImg<T> get_crop (const int x0, const int y0, const int z0, const int x1, const int y1, const int z1, const unsigned int boundary_conditions=0) const} 
  Crop image region [new-instance version].

- \texttt{CImg<T> & crop (const int x0, const int y0, const int x1, const int y1, const unsigned int boundary_conditions=0)} 
  Crop image region [overloading].

- \texttt{CImg<T> get_crop (const int x0, const int y0, const int x1, const int y1, const unsigned int boundary_conditions=0) const} 
  Crop image region [new-instance version].

- \texttt{CImg<T> & crop (const int x0, const int x1, const unsigned int boundary_conditions=0)} 
  Crop image region [overloading].

- \texttt{CImg<T> get_crop (const int x0, const int x1, const unsigned int boundary_conditions=0) const} 
  Crop image region [new-instance version].

- \texttt{CImg<T> & autocrop (const T &value, const char *const axes="czyx")} 
  Autocrop image region, regarding the specified background value.

- \texttt{CImg<T> get_autocrop (const T &value, const char *const axes="czyx") const} 
  Autocrop image region, regarding the specified background value [new-instance version].

- \texttt{CImg<T> & autocrop (const T &const color=0, const char *const axes="zyx")} 
  Autocrop image region, regarding the specified background color.

- \texttt{CImg<T> get_autocrop (const T &const color=0, const char *const axes="zyx") const} 
  Autocrop image region, regarding the specified background color [new-instance version].

- \texttt{CImg<T> get_column (const int x0) const} 
  Return specified image column.

- \texttt{CImg<T> & column (const int x0)} 
  Return specified image column [in-place version].

- \texttt{CImg<T> & columns (const int x0, const int x1)} 
  Return specified range of image columns.

- \texttt{CImg<T> get_columns (const int x0, const int x1) const} 
  Return specified range of image columns [in-place version].

- \texttt{CImg<T> get_row (const int y0) const} 
  Return specified image row.

- \texttt{CImg<T> & row (const int y0)} 
  Return specified image row [in-place version].

- \texttt{CImg<T> get_rows (const int y0, const int y1) const} 
  Return specified range of image rows.

- \texttt{CImg<T> & rows (const int y0, const int y1)} 
  Return specified range of image rows [in-place version].

- \texttt{CImg<T> get_slice (const int z0) const} 
  Return specified image slice.

- \texttt{CImg<T> & slice (const int z0)} 
  Return specified image slice [in-place version].

- \texttt{CImg<T> get_slices (const int z0, const int z1) const} 
  Return specified range of image slices.

- \texttt{CImg<T> & slices (const int z0, const int z1)} 
  Return specified range of image slices [in-place version].

- \texttt{CImg<T> get_channel (const int c0) const} 
  Return specified image channel.

- \texttt{CImg<T> & channel (const int c0)} 
  Return specified image channel [in-place version].

- \texttt{CImg<T> get_channels (const int c0, const int c1) const} 
  Return specified range of image channels.
Return specified range of image channels.

- **CImg**<\ T > & channels (const int c0, const int c1)
  Return specified range of image channels [in-place version].
- **CImg**<\ T > floatT > get_streamline (const float x, const float y, const float z, const float float d=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.
- **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
  Return a shared-memory image referencing a range of pixels of the image instance.
- **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
  Return a shared-memory image referencing a range of pixels of the image instance [const version].
- **CImg**<\ T > get_shared_rows (const unsigned int y0, const unsigned int y1, const unsigned int z0=0, const unsigned int c0=0) const
  Return a shared-memory image referencing a range of rows of the image instance.
- **CImg**<\ T > get_shared_rows (const unsigned int y0, const unsigned int y1, const unsigned int z0=0, const unsigned int c0=0) const
  Return a shared-memory image referencing a range of rows of the image instance [const version].
- **CImg**<\ T > get_shared_row (const unsigned int y0, const unsigned int z0=0, const unsigned int c0=0) const
  Return a shared-memory image referencing one row of the image instance.
- **CImg**<\ T > get_shared_row (const unsigned int y0, const unsigned int z0=0, const unsigned int c0=0) const
  Return a shared-memory image referencing one row of the image instance [const version].
- **CImg**<\ T > get_shared_slices (const unsigned int z0, const unsigned int z1, const unsigned int c0=0) const
  Return a shared-memory image referencing a range of slices of the image instance.
- **CImg**<\ T > get_shared_slices (const unsigned int z0, const unsigned int z1, const unsigned int c0=0) const
  Return a shared-memory image referencing a range of slices of the image instance [const version].
- **CImg**<\ T > get_shared_slice (const unsigned int z0, const unsigned int c0=0)
  Return a shared-memory image referencing one slice of the image instance.
- **CImg**<\ T > get_shared_slice (const unsigned int z0, const unsigned int c0=0)
  Return a shared-memory image referencing one slice of the image instance [const version].
- **CImg**<\ T > get_shared_channels (const unsigned int c0, const unsigned int c1)
  Return a shared-memory image referencing a range of channels of the image instance.
- **CImg**<\ T > get_shared_channels (const unsigned int c0, const unsigned int c1)
  Return a shared-memory image referencing a range of channels of the image instance [const version].
- **CImg**<\ T > get_shared_channel (const unsigned int c0)
  Return a shared-memory image referencing one channel of the image instance.
- **CImg**<\ T > get_shared_channel (const unsigned int c0)
  Return a shared-memory image referencing one channel of the image instance [const version].
- **CImg**<\ T > get_shared ()
  Return a shared-memory version of the image instance.
- **CImg**<\ T > get_shared ()
  Return a shared-memory version of the image instance [const version].
- **CImgList**<\ T > get_split (const char axis, const int nb=-1) const
  Split image into a list along specified axis.
- **CImgList**<\ T > get_split (const CImg<\ T > &values, const char axis=0, const bool keep_values=true) const
  Split image into a list of sub-images, according to a specified splitting value sequence and optionally axis.
- **CImg**<\ T > & append (const CImg<\ T > &img, const char axis='x', const float float align=0)
Append two images along specified axis.

- \texttt{CImg<T> & append (const CImg<T> &img, const char axis='x', const float align=0)}
  Append two images along specified axis \textbf{[specialization]}.

  \begin{verbatim}
  template<
  \texttt{typename t >}
  CImg< t >
  \end{verbatim}

- \texttt{CImg<T> & append (const CImg<T> &img, const char axis='x', const float align=0) const}
  Append two images along specified axis \textbf{[const version]}.

Filtering / Transforms

- \texttt{template<typename t func >
  static CImg< floatT > > streamline (const func &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, const float align=0) const
  Return streamline of a 3D vector field.

- \texttt{static CImg< floatT > > streamline (const char axis='x', const float align=0)}
  Return streamline of a 3D vector field \textbf{[overloading]}.
• `CImg< T > & cumulate (const char axis=0)
  Cumulate image values, optionally along specified axis.

• `CImg< Tlong > & cumulate (const char axis=0) const
  Cumulate image values, optionally along specified axis [new-instance version].

• `CImg< T > & cumulate (const char *const axes)
  Cumulate image values, along specified axes.

• `CImg< Tlong > & cumulate (const char *const axes) const
  Cumulate image values, along specified axes [new-instance version].

• template<typename t>
  CImg< T > & erode (const CImg< t > &kernel, const bool boundary_conditions=true, const bool is_real=false)
  Erode image by a structuring element.

• template<typename t>
  CImg< typename cimg::superset< T, t >::type > get_erode (const CImg< t > &kernel, const bool boundary_conditions=true, const bool is_real=false) const
  Erode image by a structuring element [new-instance version].

• CImg< T > & erode (const unsigned int sx, const unsigned int sy, const unsigned int sz=1)
  Erode image by a rectangular structuring element of specified size.

• CImg< T > get_erode (const unsigned int sx, const unsigned int sy, const unsigned int sz=1) const
  Erode image by a rectangular structuring element of specified size [new-instance version].

• CImg< T > & erode (const unsigned int s)
  Erode the image by a square structuring element of specified size.

• CImg< T > get_erode (const unsigned int s) const
  Erode the image by a square structuring element of specified size [new-instance version].

• template<typename t>
  CImg< T > & dilate (const CImg< t > &kernel, const bool boundary_conditions=true, const bool is_real=false)
  Dilate image by a structuring element.

• template<typename t>
  CImg< typename cimg::superset< T, t >::type > get_dilate (const CImg< t > &kernel, const bool boundary_conditions=true, const bool is_real=false) const
  Dilate image by a structuring element [new-instance version].

• CImg< T > & dilate (const unsigned int sx, const unsigned int sy, const unsigned int sz=1)
  Dilate image by a rectangular structuring element of specified size.

• CImg< T > get_dilate (const unsigned int sx, const unsigned int sy, const unsigned int sz=1) const
  Dilate image by a rectangular structuring element of specified size [new-instance version].

• CImg< T > & dilate (const unsigned int s)
  Dilate image by a square structuring element of specified size.

• CImg< T > get_dilate (const unsigned int s) const
  Dilate image by a square structuring element of specified size [new-instance version].

• template<typename t>
  CImg< T > & watershed (const CImg< t > &priority, const bool is_high_connectivity=false)
  Compute watershed transform.

• template<typename t>
  CImg< T > get_watershed (const CImg< t > &priority, const bool is_high_connectivity=false) const
  Compute watershed transform [new-instance version].

• CImg< T > & deriche (const float sigma, const unsigned int order=0, const char axis='x', const bool boundary_conditions=true)
  Apply recursive Deriche filter.

• CImg< Tfloat > & get_deriche (const float sigma, const unsigned int order=0, const char axis='x', const bool boundary_conditions=true) const
  Apply recursive Deriche filter [new-instance version].
• \texttt{CImg\<T\>\& vanvliet} (const float sigma, const unsigned int order, const char axis='x', const bool boundary\_conditions=true)
  
  \textit{Van Vliet recursive Gaussian filter.}

• \texttt{CImg\<T\>float get\_vanvliet} (const float sigma, const unsigned int order, const char axis='x', const bool boundary\_conditions=true, const bool is\_gaussian=false)
  
  Blur image using Van Vliet recursive Gaussian filter. \textbf{[new-instance version].}

• \texttt{CImg\<T\>\& blur} (const float sigma\_x, const float sigma\_y, const float sigma\_z, const bool boundary\_conditions=true, const bool is\_gaussian=false)
  
  Blur image.

• \texttt{CImg\<T\>float get\_blur} (const float sigma\_x, const float sigma\_y, const float sigma\_z, const bool boundary\_conditions=true, const bool is\_gaussian=false) const
  
  Blur image \textbf{[new-instance version].}

• \texttt{CImg\<T\>\& blur} (const float sigma, const bool boundary\_conditions=true, const bool is\_gaussian=false)
  
  Blur image isotropically.

• \texttt{CImg\<T\>float get\_blur} (const float sigma, const bool boundary\_conditions=true, const bool is\_gaussian=false) const
  
  Blur image \textbf{[new-instance version].}

• \texttt{CImg\<T\>\& blur\_anisotropic} (const \texttt{CImg\<t\>}\&G, const float amplitude=60, const float dl=0.8f, const float da=30, const float gauss\_prec=2, const unsigned int interpolation\_type=0, const bool is\_fast\_approx=1)
  
  Blur image anisotropically, directed by a field of diffusion tensors.

• \texttt{template<typename t >}
  
  \texttt{CImg\<T\>\& blur\_anisotropic} (const \texttt{CImg\<t\>}\&G, const float amplitude=60, const float dl=0.8f, const float da=30, const float gauss\_prec=2, const unsigned int interpolation\_type=0, const bool is\_fast\_approx=true) const
  
  Blur image anisotropically, directed by a field of diffusion tensors \textbf{[new-instance version].}

• \texttt{CImg\<T\>\& blur\_anisotropic} (const \texttt{CImg\<t\>}\&G, const float amplitude, const float sharpness=0.7f, const float anisotropy=0.6f, const float alpha=0.6f, const float sigma=1.1f, const float dl=0.8f, const float da=30, const float gauss\_prec=2, const unsigned int interpolation\_type=0, const bool is\_fast\_approx=true)
  
  Blur image anisotropically, in an edge-preserving way.

• \texttt{template<typename t >}
  
  \texttt{CImg\<T\>\& blur\_anisotropic} (const \texttt{CImg\<t\>}\&G, const float amplitude, const float sharpness=0.7f, const float anisotropy=0.6f, const float alpha=0.6f, const float sigma=1.1f, const float dl=0.8f, const float da=30, const float gauss\_prec=2, const unsigned int interpolation\_type=0, const bool is\_fast\_approx=true) const
  
  Blur image anisotropically, in an edge-preserving way \textbf{[new-instance version].}

• \texttt{CImg\<T\>\& blur\_bilateral} (const \texttt{CImg\<t\>}\&guide, const float sigma\_x, const float sigma\_y, const float sigma\_z, const float sigma\_r, const float sampling\_x, const float sampling\_y, const float sampling\_z, const float sampling\_r)
  
  Blur image, with the joint bilateral filter.

• \texttt{template<typename t >}
  
  \texttt{CImg\<T\>\& blur\_bilateral} (const \texttt{CImg\<t\>}\&guide, const float sigma\_x, const float sigma\_y, const float sigma\_z, const float sigma\_r, const float sampling\_s=0, const float sampling\_r=0)
  
  Blur image, with the joint bilateral filter \textbf{[new-instance version].}

• \texttt{template<typename t >}
  
  \texttt{CImg\<T\>\& blur\_bilateral} (const \texttt{CImg\<t\>}\&guide, const float sigma\_s, const float sigma\_r, const float sampling\_s=0, const float sampling\_r=0) const
  
  Blur image using the joint bilateral filter.

• \texttt{CImg\<T\>\& blur\_bilateral} (const \texttt{CImg\<t\>}\&guide, const float sigma\_s, const float sigma\_r, const float sampling\_s=0, const float sampling\_r=0) const
  
  Blur image using the bilateral filter \textbf{[new-instance version].}

• \texttt{CImg\<T\>\& boxfilter} (const float boxsize, const int order, const char axis='x', const bool boundary\_conditions=true, const unsigned int nb\_iter=1)
  
  Blur image with a box filter. \textbf{[new-instance version].}
**CImg<Tfloat>* get_boxfilter** (const float boxsize, const int order, const char axis='x', const bool boundary_conditions=true, const unsigned int nb_iter=1) const

Blur image with a box filter.

**CImg<Tfloat>& blur_box** (const float boxsize_x, const float boxsize_y, const float boxsize_z, const bool boundary_conditions=true, const unsigned int nb_iter=1)

Blur image with a box filter [new-instance version].

**CImg<Tfloat>& blur_box** (const float boxsize_x, const float boxsize_y, const float boxsize_z, const bool boundary_conditions=true)

Blur image with a box filter.

**CImg<Tfloat>& blur_box** (const float boxsize, const bool boundary_conditions=true, const unsigned int nb_iter=1)

Blur image with a box filter [new-instance version].

**CImg<Tfloat>& blur_guided** (const CImg<t>&guide, const float radius, const float regularization)

Blur image, with the image guided filter.

**CImg<Tfloat>& get_blur_guided** (const CImg<t>&guide, const float radius, const float regularization) const

Blur image, with the image guided filter [new-instance version].

**CImg<Tfloat>& blur_patch** (const CImg<t>&guide, const float sigma_s, const float sigma_r, const unsigned int patch_size=3, const unsigned int lookup_size=4, const bool is_fast_approx=true)

Blur image using patch-based space.

**CImg<Tfloat>& get_blur_patch** (const CImg<t>&guide, const float sigma_s, const float sigma_r, const unsigned int patch_size=3, const unsigned int lookup_size=4, const float smoothness=0, const bool is_fast_approx=true) const

Blur image using patch-based space [new-instance version].

**CImg<Tfloat>& blur_patch** (const float sigma_s, const float sigma_r, const unsigned int patch_size=3, const unsigned int lookup_size=4, const float smoothness=0, const bool is_fast_approx=true)

Blur image using patch-based space [simplification].

**CImg<Tfloat>& sharpen** (const float amplitude, const bool sharpen_type=false, const float edge=1, const float alpha=0, const float sigma=0)

Sharpen image.

**CImg<Tfloat>& get_sharpen** (const float amplitude, const bool sharpen_type=false, const float edge=1, const float alpha=0, const float sigma=0) const

Sharpen image [new-instance version].

**CImgList<Tfloat>* get_gradient** (const char*const axes=0, const int scheme=0) const

Return image gradient.

**CImgList<Tfloat>* get_hessian** (const char*const axes=0) const

Return image hessian.

**CImg<T>& laplacian** ()

Compute image Laplacian.

**CImg<Tfloat>& get_laplacian** () const

Compute image Laplacian [new-instance version].
• Clmg::<T> & structure_tensors (const bool is_fwbw_scheme=false)
  
  Compute the structure tensor field of an image.

• Clmg::<T> float > get_structure_tensors (const bool is_fwbw_scheme=false) const
  
  Compute the structure tensor field of an image [new-instance version].

• Clmg::<T> & diffusion_tensors (const float sharpness=0.7f, const float anisotropy=0.6f, const float alpha=0.6f, const float sigma=1.1f, const bool is_sqrt=false)
  
  Compute field of diffusion tensors for edge-preserving smoothing.

• Clmg::<T> float > get_diffusion_tensors (const float sharpness=0.7f, const float anisotropy=0.6f, const float alpha=0.6f, const float sigma=1.1f, const bool is_sqrt=false) const
  
  Compute field of diffusion tensors for edge-preserving smoothing [new-instance version].

• Clmg::<T> & displacement (const Clmg::<T> &source, const float smoothness=0.1f, const float precision=5.1, const unsigned int nb_scales=0, const unsigned int iteration_max=10000, const bool is_backward=false, const Clmg::<floatT> &guide=Clmg::<floatT>::const_empty())
  
  Estimate displacement field between two images.

• Clmg::<floatT> float > get_displacement (const Clmg::<T> &source, const float smoothness=0.1f, const float precision=5.1, const unsigned int nb_scales=0, const unsigned int iteration_max=10000, const bool is_backward=false, const Clmg::<floatT> &guide=Clmg::<floatT>::const_empty()) const
  
  Estimate displacement field between two images [new-instance version].

* template<typename t1 , typename t2 >
  
  Clmg::<T> & matchpatch (const Clmg::<T> &patch_image, const unsigned int patch_width, const unsigned int patch_height, const unsign long int nb_randoms, const float occ_penalization, const Clmg::<T>::const_empty()) const
  
  Compute correspondence map between two images, a patch-matching algorithm.

  template<typename t >
  
  Clmg::<T> & matchpatch (const Clmg::<T> &patch_image, const unsigned int patch_width, const unsigned int patch_height, const unsign long int nb_randoms=5, const float occ_penalization=0, const unsigned int nb_iterations=5, const unsigned int nb_randoms=5, const float occ_penalization=0, const Clmg::<T>::const_empty()) const
  
  Compute correspondence map between two images, using the patch-match algorithm [new-instance version].

  template<typename t >
  
  Clmg::<T> & matchpatch (const Clmg::<T> &patch_image, const unsigned int patch_width, const unsigned int patch_height, const unsign long int nb_randoms=5, const float occ_penalization=0, const unsigned int nb_iterations=5, const unsigned int nb_randoms=5, const float occ_penalization=0, const Clmg::<T>::const_empty()) const
  
  Compute correspondence map between two images, using the patch-match algorithm [overloading].

• Clmg::<T> & distance (const T &value, const unsigned int metric=2)
  
  Compute Euclidean distance function to a specified value.

• Clmg::<T> float > get_distance (const T &value, const unsigned int metric=2) const
  
  Compute distance to a specified value [new-instance version].

* template<typename t >
  
  Clmg::<T> & distance (const T &value, const Clmg::<T>::const_empty()) const
  
  Compute chamfer distance to a specified value, with a custom metric.

* template<typename t >
  
  Clmg::<T> float > get_distance (const T &value, const Clmg::<T>::const_empty()) const
  
  Compute chamfer distance to a specified value, with a custom metric [new-instance version].

* template<typename t , typename to >
  
  Clmg::<T> & distance_dijkstra (const T &value, const Clmg::<T>::const_empty()) const
  
  Compute distance to a specified value, according to a custom metric (use dijkstra algorithm).
• template<typename t , typename to >
  CImg< typename cimg::superset< t, long >::type >
  get_distance_dijkstra (const T &value, const CImg< t > &metric,
  const bool is_high_connectivity, CImg< to > &return_path) const
  
  Compute distance map to a specified value, according to a custom metric (use dijkstra algorithm) [new-instance version].

• template<typename t >
  CImg< T > & distance_dijkstra (const T &value, const CImg< t > &metric,
  const bool is_high_connectivity=false)
  
  Compute distance map to a specified value, according to a custom metric (use dijkstra algorithm). [overloading].

• template<typename t >
  CImg< Tfloat > get_distance_dijkstra (const T &value, const CImg< t > &metric,
  const bool is_high_connectivity=false) const
  
  Compute distance map to a specified value, according to a custom metric (use dijkstra algorithm). [new-instance version].

• template<typename t >
  CImg< T > & distance_eikonal (const T &value, const CImg< t > &metric)
  
  Compute distance map to one source point, according to a custom metric (use fast marching algorithm).

• template<typename t >
  CImg< Tfloat > get_distance_eikonal (const T &value, const CImg< t > &metric) const
  
  Compute distance map to one source point, according to a custom metric (use fast marching algorithm).

• CImg< Tfloat > get_distance_eikonal (const unsigned int nb_iterations, const float band_size=0, const float time_step=0.5f) const
  
  Compute distance function to 0-valued isophotes, using the Eikonal PDE [new-instance version].

• CImg< Tfloat > get_distance_eikonal (const unsigned int nb_iterations, const float band_size=0, const float time_step=0.5f) const
  
  Compute distance function to 0-valued isophotes, using the Eikonal PDE [new-instance version].

• CImg< T > & haar (const char axis, const bool invert=false, const unsigned int nb_scales=1)
  
  Compute Haar multiscale wavelet transform.

• CImg< Tfloat > haar (const char axis, const bool invert=false, const unsigned int nb_scales=1) const
  
  Compute Haar multiscale wavelet transform [new-instance version].

• CImg< Tfloat > get_haar (const char axis, const bool invert=false, const unsigned int nb_scales=1) const
  
  Compute Haar multiscale wavelet transform [overloading].

• CImg< Tfloat > get_haar (const bool invert=false, const unsigned int nb_scales=1) const
  
  Compute Haar multiscale wavelet transform [new-instance version].

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

• CImgList< Tfloat > get_FFT (const bool is_inverse=false) const
  
  Compute n-D Fast Fourier Transform.

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

• static void FFT (CImg< T > &real, CImg< T > &imag, const bool is_inverse=false, const unsigned int nb_threads=0)
  
  Compute n-D Fast Fourier Transform.

3D Objects Management

• CImg< T > & shift_object3d (const float tx, const float ty=0, const float tz=0)
  
  Shift 3D object's vertices.

• CImg< Tfloat > get_shift_object3d (const float tx, const float ty=0, const float tz=0) const
  
  Shift 3D object's vertices [new-instance version].

• CImg< T > & shift_object3d ()
  
  Shift 3D object's vertices [overloading].

Generated by Doxygen
8.1 Clmg< T > Struct Template Reference

Shift 3D object's vertices, so that it becomes centered.
- Clmg< Tfloat > get_shift_object3d () const
  Shift 3D object's vertices, so that it becomes centered [new-instance version].
- Clmg< T > & resize_object3d (const float sx, const float sy=-100, const float sz=-100)
  Resize 3D object.
- Clmg< Tfloat > get_resize_object3d (const float sx, const float sy=-100, const float sz=-100) const
  Resize 3D object [new-instance version].
- Clmg< T > resize_object3d ()
  Resize 3D object to unit size.
- Clmg< Tfloat > & resize_object3d (const float sx, const float sy=-100, const float sz=-100)
  Resize 3D object to unit size [new-instance version].
- template<typename tf , typename tp , typename tt >
  Clmg< T > append_object3d (ClmgList< tf > &primitives, const Clmg< tp > &obj_vertices, const CImg< tf > &obj_primitives)
  Merge two 3D objects together.
- template<typename tp , typename tc , typename tt , typename tx >
  const Clmg< T > & texturize_object3d (ClmgList< tp > &primitives, ClmgList< tc > &colors, const Clmg< tt > &texture, const Clmg< tx >::const_empty()) const
  Texturize primitives of a 3D object.
- template<typename tf , typename tc , typename te >
  CImg< floatT > get_elevation3d (ClmgList< tf > &primitives, ClmgList< tc > &colors, const Clmg< te > &elevation) const
  Generate a 3D elevation of the image instance.
- template<typename tf >
  CImg< floatT > get_projections3d (ClmgList< tf > &primitives, ClmgList< tc > &colors, const unsigned int x0, const unsigned int y0, const unsigned int z0, const bool normalize_colors=false) const
  Generate the 3D projection planes of the image instance.
- template<typename tf >
  Clmg< floatT > get_isoline3d (ClmgList< tf > &primitives, const float isovalue, const int size_x=-100, const int size_y=-100) const
  Generate a isoline of the image instance as a 3D object.
- template<typename tf >
  Clmg< floatT > get_isosurface3d (ClmgList< tf > &primitives, const float isovalue, const int size_x=-100, const int size_y=-100, const int size_z=-100) const
  Generate an isosurface of the image instance as a 3D object.
- template<typename tp , typename tc , typename to >
  Clmg< T > object3dtoCImg3d (const ClmgList< tp > &primitives, const ClmgList< tc > &colors, const to &opacities, const bool full_check=true)
  Convert 3D object into a Clmg3d representation.
- template<typename tp , typename tc >
  CImg< T > & object3dtoCImg3d (const ClmgList< tp > &primitives, const ClmgList< tc > &colors, const bool full_check=true)
  Convert 3D object into a Clmg3d representation [overloading].
- template<typename tp >
  Clmg< T > & object3dtoCImg3d (const ClmgList< tp > &primitives, const bool full_check=true)
  Convert 3D object into a Clmg3d representation [overloading].
- Clmg< T > & object3dtoCImg3d (const bool full_check=true)
  Convert 3D object into a Clmg3d representation [overloading].
- template<typename tp , typename tc , typename to >
  Clmg< floatT > get_object3dtoCImg3d (const ClmgList< tp > &primitives, const ClmgList< tc > &colors, const to &opacities, const bool full_check=true) const
  Convert 3D object into a Clmg3d representation [new-instance version].

Generated by Doxygen
• template<typename tp , typename tc >
  CImg<floatT> get_object3dtoCImg3d (const CImgList<tp>& primitives, const CImgList<tc>& colors, const bool full_check=true) const
  Convert 3D object into a CImg3d representation [overloading].

• template<typename tp >
  CImg<floatT> get_object3dtoCImg3d (const CImgList<tp>& primitives, const bool full_check=true) const
  Convert 3D object into a CImg3d representation [overloading].

• CImg<floatT> get_object3dtoCImg3d (const bool full_check=true) const
  Convert 3D object into a CImg3d representation [overloading].

• template<typename tp , typename tc , typename to >
  CImg<T> CImg3dtoobject3d (CImgList<tp>& primitives, CImgList<tc>& colors, CImgList<to>& opacities, const bool full_check=true)
  Convert CImg3d representation into a 3D object.

• template<typename tp , typename tc , typename to >
  CImg<T> get_CImg3dtoobject3d (CImgList<tp>& primitives, CImgList<tc>& colors, CImgList<to>& opacities, const bool full_check=true) const
  Convert CImg3d representation into a 3D object [new-instance version].

• 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)
  Compute 3D elevation of a function as a 3D object.

• 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 size_x=256, const int size_y=256)
  Compute 3D elevation of a function, as a 3D object [overloading].

• template<typename tf , typename tfunc >
  static CImg<floatT> isoline3d (CImgList<tf>& primitives, const tfunc &func, const float isovalue, const float x0, const float y0, const float x1, const float y1, const int size_x=256, const int size_y=256)
  Compute 0-isolines of a function, 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 dx=32, const int dy=32, const int dz=32)
  Compute isolines of a function, as a 3D object [overloading].

• 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)
  Compute isosurface of a function, 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)
  Compute isosurface of a function, as a 3D object [overloading].

• 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)
  Generate 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)
  Generate 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)
8.1 CImg< T > Struct Template Reference 105

Generate 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)

  Generate 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)

  Generate a 3D XY-plane.

- template<typename tf , typename t >
  static CImg< floatT > ellipsoid3d (CImgList< tf >&primitives, const CImg< t >&tensor, const unsigned int subdivisions=3)

  Generate a 3D ellipsoid.

Drawing Functions

- template<typename tc >
  CImg< T > & draw_point (const int x0, const int y0, const int z0, const tc ∗const color, const float opacity=1)

  Draw a 3D point.

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

  Draw a 2D point [simplification].

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

  Draw a 2D line.

- template<typename tz , typename tc >
  CImg< T > & draw_line (CImg< tz >&zbuffer, int x0, int y0, int x1, int y1, const float z0, const float z1, const tc ∗const color, const float float opacity=1, const unsigned int pattern=~0U, const bool init_hatch=true)

  Draw a 2D line, with z-buffering.

- template<typename tc >
  CImg< T > & draw_line (int x0, int y0, int x1, int y1, const CImg< tc >&texture, int tx0, int ty0, int tx1, int ty1, const float float opacity=1, const unsigned int pattern=~0U, const bool init_hatch=true)

  Draw a textured 2D line.

- template<typename tc >
  CImg< T > & draw_line (int x0, int y0, const float z0, int x1, int y1, const float z1, const CImg< tc >&texture, const int tx0, const int ty0, const int tx1, const int ty1, const float float opacity=1, const unsigned int pattern=~0U, const bool init_hatch=true)

  Draw a textured 2D line, with perspective correction.

- template<typename tz , typename tc >
  CImg< T > & draw_line (CImg< tz >&zbuffer, int x0, int y0, const float z0, int x1, int y1, const float z1, const CImg< tc >&texture, const int tx0, const int ty0, const int tx1, const int ty1, const float float opacity=1, const unsigned int pattern=~0U, const bool init_hatch=true)

  Draw a textured 2D line, with perspective correction and z-buffering.
Draw a set of consecutive lines.

- template<typename tc >
  CImg< T > & draw_arrow (const int x0, const int y0, const int x1, const int y1, const tc *const color, const float opacity=1, const float angle=30, const float length=-10, const unsigned int pattern=~0U)

  Draw a 2D arrow.

- template<typename tc >
  CImg< T > & draw_spline (const int x0, const int y0, const int x1, const int y1, const float u0, const float v0, const int x2, const int y2, const float u1, const float v1, const CImg< T > &texture, const float opacity=1, const float precision=4, const unsigned int pattern=~0U, const bool init_hatch=true)

  Draw a老头 2D spline.

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

  Draw a set of consecutive splines.

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

  Draw a set of consecutive splines [overloading].

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

  Draw a filled 2D triangle, with z-buffering.

- template<typename tc >
  CImg< T > & draw_triangle (int x0, int y0, int x1, int y1, int x2, int y2, const tc *const color, const float opacity=1, const float brightness=1)

  Draw a Gouraud-shaded 2D triangle.

- template<typename tz , 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, int tx0, int ty0, int tx1, int ty1, int tx2, int ty2, const float opacity=1, const float brightness=1)

  Draw a color-interpolated 2D triangle.

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

  Draw a textured 2D triangle.

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

  Draw a Gouraud-shaded 2D triangle, with z-buffering [overloading].
Draw a 2D textured triangle, with perspective correction.

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

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

- template<typename tc , typename tl >
  - CImg<T> & draw_triangle (int x0, int y0, int y1, int x2, int y2, const tc &const color, const CImg<Tl> &tl > &light, int lx0, int ly0, int lx1, int ly1, int lx2, int ly2, const float opacity=1)
• template<typename t, 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 outlined 2D rectangle [overloading].
• template<typename tp , typename tc >
  CImg<T> & draw_polygon (const CImg<tp>&points, const tc *const color, const float opacity=1)
  Draw a filled 2D polygon.
• 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 2D or 3D polygon [overloading].
• template<typename tc >
  CImg<T> & draw_ellipse (const int x0, const int y0, const float r1, const float r2, const float angle, const tc *const color, const float opacity=1)
  Draw a filled 2D ellipse.
• 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 a filled 2D ellipse [overloading].
• 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)
  Draw an image [specialization].
• 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 [overloading].
• template<typename t >
  CImg<T> & draw_image (const int x0, const CImg<t>&sprite, const float opacity=1)
  Draw an image [overloading].


- \template<typename ti , typename tm >
  \Clmg<T> & draw_image (const int x0, const int y0, const int z0, const int c0, const \Clmg<Ti> &sprite, const \Clmg<T> &mask, const float opacity=1, const float mask_max_value=1)

  Draw a masked image.

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

  Draw a masked image [overloading].

- \template<typename ti , typename tm >
  \Clmg<T> & draw_image (const int x0, const int y0, const \Clmg<Ti> &sprite, const \Clmg<T> &mask, const float opacity=1, const float mask_max_value=1)

  Draw an image.

- \template<typename tc1 , typename tc2 , typename t >
  \Clmg<T> & draw_text (const int x0, const int y0, const char *const text, const tc1 *const foreground_color, const tc2 *const background_color, const float opacity, const \Clmg<List<T>> &font,...)

  Draw a text string.

- \template<typename tc , typename t >
  \Clmg<T> & draw_text (const int x0, const int y0, const char *const text, const int, const tc *const background_color, const float opacity=1, const unsigned int font_height=13,...)

  Draw a text string [overloading].

- \template<typename tc1 , typename tc2 >
  \Clmg<T> & draw_text (const int x0, const int y0, const char *const text, const int, const tc1 *const foreground_color, const tc2 *const background_color=0, const float opacity=1, const unsigned int font_height=13,...)

  Draw a text string [overloading].

- \template<typename t1 , typename t2 >
  \Clmg<T> & draw_quiver (const \Clmg<T1> &flow, const t2 *const color, const float opacity=1, const unsigned int sampling=25, const float factor=-20, const bool is_arrow=true, const unsigned int pattern=~0U)

  Draw a 2D vector field.

- \template<typename t1 , typename t2 >
  \Clmg<T> & draw_quiver (const \Clmg<T1> &flow, const \Clmg<T2> &color, const float opacity=1, const unsigned int sampling=25, const float factor=-20, const bool is_arrow=true, const unsigned int pattern=~0U)

  Draw a 2D vector field, using a field of colors.

---

*Generated by Doxygen*
• template<typename t , typename tc >
  CImg< T > & draw_axis (const CImg< t > &values_x, const int y, const tc *const color, const float opacity=1,
  const unsigned int pattern~0U, const unsigned int font_height=13, const bool allow_zero=true, const float
  round_x=0)
  
  Draw a labeled horizontal axis.
• template<typename t , typename tc >
  CImg< T > & draw_axis (const int x, const CImg< t > &values_y, const tc *const color, const float opacity=1,
  const unsigned int pattern~0U, const unsigned int font_height=13, const bool allow_zero=true, const float
  round_y=0)
  
  Draw a labeled vertical axis.
• template<typename tx , typename ty , typename tc >
  CImg< T > & draw_axes (const CImg< tx > &values_x, const CImg< ty > &values_y, const tc *const color,
  const float opacity=1, const unsigned int pattern_x~0U, const unsigned int pattern_y~0U, const unsigned
  int font_height=13, const bool allow_zero=true, const float round_x=0, const float round_y=0)
  
  Draw labeled horizontal and vertical axes.
• 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 pattern_x~0U, const unsigned int pattern_y~0U, const unsigned int
  font_height=13)
  
  Draw labeled horizontal and vertical axes [overloading].
• template<typename tx , typename ty , typename tc >
  CImg< T > & draw_grid (const CImg< tx > &values_x, const CImg< ty > &values_y, const tc *const color,
  const float opacity=1, const unsigned int pattern_x~0U, const unsigned int pattern_y~0U)
  
  Draw 2D grid.
• template<typename tc >
  CImg< T > & draw_grid (const float delta_x, const float delta_y, const float offsetx, const float offsety, const
  bool invertx, const bool inverty, const tc *const color, const float opacity=1, const unsigned int pattern_x~0U,
  const unsigned int pattern_y~0U)
  
  Draw 2D grid [simplification].
• 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 1D graph.
• template<typename tc , typename t >
  CImg< T > & draw_fill (const int x0, const int y0, const int z0, const tc *const color, const float opacity,
  CImg< t > &region, const float tolerance=0, const bool is_high_connectivity=false)
  
  Draw filled 3D region with the flood fill algorithm.
• template<typename tc >
  CImg< T > & draw_fill (const int x0, const int y0, const int z0, const tc *const color, const float opacity=1,
  const float tolerance=0, const bool is_high_connexity=false)
  
  Draw filled 2D region with the flood fill algorithm [simplification].
• template<typename tc >
  CImg< T > & draw_fill (const int x0, const int y0, const int z0, const tc *const color, const float opacity=1,
  const float tolerance=0, const bool is_high_connectivity=false)
  
  Draw filled 2D region with the flood fill algorithm [simplification].
• template<typename tc >
  CImg< T > & draw_plasma (const float alpha=1, const float beta=0, const unsigned int scale=8)
  
  Draw a random plasma texture.
• template<typename tc >
  CImg< T > & draw_mandelbrot (const int x0, const int y0, const int x1, const int y1, const CImg< tc > &colormap,
  const float opacity=1, const double z0r=-2, const double z0i=-2, const double z1r=2, const double z1i=2, const unsigned int
  iteration_max=255, const bool is_normalizedIteration=false, const bool is_julia_set=false, const double param_r=0,
  const double param_i=0)
  
  Draw a quadratic Mandelbrot or Julia 2D fractal.
8.1 Clmg< T > Struct Template Reference

```
• template<typename tc>
  Clmg< T > & draw_mandelbrot (const Clmg< T > &colormap, const float opacity=1, const double z0r=-2, const double z0i=-2, const double z1r=-2, const double z1i=-2, const unsigned int iteration_max=255, const bool is_normalized_iteration=false, const bool is_julia_set=false, const double param_r=0, const double param_i=0)

  Draw a quadratic Mandelbrot or Julia 2D fractal [overloading].

• template<typename tc>
  Clmg< T > & draw_gaussian (const float xc, const float sigma, const Clmg< T > &tensor, const float color, const float opacity=1)

  Draw a 2D gaussian function.

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

  Draw a 2D gaussian function [overloading].

• template<typename tc>
  Clmg< T > & draw_gaussian (const int xc, const int yc, const float r1, const float r2, const float ru, const float ru, const float rv, const float rc, const Clmg< T > &tensor, const float color, const float opacity=1)

  Draw a 2D gaussian function [overloading].

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

  Draw a 2D gaussian function [overloading].

• template<typename tc>
  Clmg< T > & draw_gaussian (const float xc, const float yc, const float zc, const Clmg< T > &tensor, const float color, const float opacity=1)

  Draw a 2D gaussian function [overloading].

• template<typename tp, typename tf, typename tc, typename to, typename tz>
  Clmg< T > & draw_mandelbrot (const CImgList &zbuffer)

  Draw a 3D gaussian function.

• template<typename tp, typename tf, typename tc, typename to, typename tz>
  Clmg< T > & draw_object3d (const float x0, const float y0, const float z0, const CImgList &zbuffer)

  Draw a 3D object.
```

Generated by Doxygen
Draw a 3D object [simplification].

- 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 is_double_sided=false, const float focale=700, const float lightx=0, const float lighty=0, const float lightz=-5e8, const float specular_lightness=0.2f, const float specular_shininess=0.1f, const float g_opacity=1)

  Draw a 3D object [simplification].

- 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 is_double_sided, const float focale, const float lightx, const float lighty, const float lightz, const float specular_lightness, const float specular_shininess, const float g_opacity, CImg<tz>& zbuffer)

  Draw a 3D object [simplification].

Data Input

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

  Launch 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, const bool exit_on_anykey=false, const bool is_deep_selection_default=false)

  Simple interface to select a shape from an image [overloading].

- CImg<intT>& get_select (CImgDisplay &disp, const unsigned int feature_type=2, unsigned int ∗const X←YZ=0, const bool exit_on_anykey=false, const bool is_deep_selection_default=false) const

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

- CImg<intT>& get_select (const char ∗const title, const unsigned int feature_type=2, unsigned int ∗const XYZ=0, const bool exit_on_anykey=false, const bool is_deep_selection_default=false) const

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

- 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 bool exit_on_anykey=false) const

  Select sub-graph in a graph.

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

  Load image from a file.

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

  Load image from an Ascii file.

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

  Load image from an Ascii file [overloading].

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

  Load image from a DLM file.

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

  Load image from a DLM file [overloading].

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

  Load image from a BMP file.

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

  Load image from a BMP file [overloading].

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

  Load image from a JPEG file.

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

  Load image from a JPEG file [overloading].

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

  Load image from a file, using Magick++ library.
• CImg< T > & load_png (const char * const filename, unsigned int * const bits_per_pixel=0)  
  Load image from a PNG file.
• CImg< T > & load_png (std::FILE * const file, unsigned int * const bits_per_pixel=0)  
  Load image from a PNG file [overloading].
• CImg< T > & load_pnm (const char * const filename)  
  Load image from a PNM file.
• CImg< T > & load_pnm (std::FILE * const file)  
  Load image from a PNM file [overloading].
• CImg< T > & load_pfm (const char * const filename)  
  Load image from a PFM file.
• CImg< T > & load_pfm (std::FILE * const file)  
  Load image from a PFM file [overloading].
• CImg< T > & load_rgb (const char * const filename, const unsigned int dimw, const unsigned int dimh=1)  
  Load image from a RGB file.
• CImg< T > & load_rgb (std::FILE * const file, const unsigned int dimw, const unsigned int dimh=1)  
  Load image from a RGB file [overloading].
• CImg< T > & load_rgba (const char * const filename, const unsigned int dimw, const unsigned int dimh=1)  
  Load image from a RGBA file.
• CImg< T > & load_rgba (std::FILE * const file, const unsigned int dimw, const unsigned int dimh=1)  
  Load image from a RGBA file [overloading].
• 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, float * const voxel_size=0, CImg< charT > * const description=0)  
  Load image from a TIFF file.
• CImg< T > & load_minc2 (const char * const filename)  
  Load image from a MINC2 file.
• CImg< T > & load_analyze (const char * const filename, float * const voxel_size=0)  
  Load image from an ANALYZE7.5/NIFTI file.
• CImg< T > & load_analyze (std::FILE * const file, float * const voxel_size=0)  
  Load image from an ANALYZE7.5/NIFTI file [overloading].
• CImg< T > & load_cimg (const char * const filename, const char * const axis='z', const float align=0)  
  Load sub-images of a .cimg file.
• CImg< T > & load_cimg (std::FILE * const file, const char * const axis='z', const float align=0)  
  Load sub-images of a .cimg file [overloading].
• CImg< T > & load_inr (const char * const filename, float * const voxel_size=0)  
  Load image from an INRIMAGE-4 file.
• CImg< T > & load_inr (std::FILE * const file, float * const voxel_size=0)  
  Load image from an INRIMAGE-4 file [overloading].
• CImg< T > & load_exr (const char * const filename)  
  Load image from an EXR file.
• CImg< T > & load_pandore (const char * const filename)  
  Load image from a PANDORE-5 file.
Load image from a PANDORE-5 file [overloading].
- \texttt{CImg< T > & load_parrec} (const char *const filename, const char axis='c', const float align=0)
  
  
  Load image from a PAR-REC (Philips) file.
- \texttt{CImg< T > & load_raw} (const char *const filename, const char axis='c', const float align=0)
  
  
  Load image from a raw binary file.
- \texttt{CImg< T > & load_yuv} (const char *const filename, const unsigned int size_x, const unsigned int chroma_subsampling=444, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const char axis='z')
  
  Load image sequence from a YUV file.
- \texttt{CImg< T > & load_3doff} (CImgList< tf >&primitives, CImgList< tc >&colors, const char *const filename)
  
  Load 3D object from a .OFF file.
- \texttt{CImg< T > & load_ffmpeg} (std::FILE *const file)
  
  Load image sequence using FFMPEG's external tool 'ffmpeg'.
- \texttt{CImg< T > & load_gif} (const char *const filename, const char axis='z', const float align=0)
  
  Load gif file, using Imagemagick or GraphicsMagicks's external tools.
- \texttt{CImg< T > & load_graphicsmagick} (const char *const filename)
  
  Load image using GraphicsMagick's external tool 'gm'.
- \texttt{CImg< T > & load_gzip} (const char *const filename)
  
  Load gzipped image file, using external tool 'gunzip'.
- \texttt{CImg< T > & load_imagemagick} (const char *const filename)
  
  Load image using ImageMagick's external tool 'convert'.
- \texttt{CImg< T > & load_medcon} (const char *const filename)
  
  Load image from a DICOM file, using XMedcon's external tool 'medcon'.
- \texttt{CImg< T > & load_dcraw} (const char *const filename)
  
  Load image from a RAW Color Camera file, using external tool 'dcraw'.
- \texttt{CImg< T > & load_camera} (const unsigned int camera_index=0, const unsigned int capture_width=0, const unsigned int capture_height=0, const unsigned int skip_frames=0, const bool release_camera=true)
  
  Load image from a camera stream, using OpenCV.
- \texttt{CImg< T > & load_other} (const char *const filename)
  
  Load image using various non-native ways.
- \texttt{static CImg< T > get_load} (const char *const filename)
  
  Load image from a file [new-instance version].
- \texttt{static CImg< T > get_load_ascii} (const char *const filename)
  
  Load image from an Ascii file [in-place version].
- \texttt{static CImg< T > get_load_ascii} (std::FILE *const file)
Load image from an Ascii file [new-instance version].

- static CImg< T > get_load_dlm (const char * const filename)
- static CImg< T > get_load_dlm (std::FILE * const file)

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

- static CImg< T > get_load_bmp (const char * const filename)
- static CImg< T > get_load_bmp (std::FILE * const file)

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

- static CImg< T > get_load_jpeg (const char * const filename)
- static CImg< T > get_load_jpeg (std::FILE * const file)

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

- static CImg< T > get_load_magick (const char * const filename)
- static CImg< T > get_load_png (const char * const filename, unsigned int * const bits_per_pixel=0)
- static CImg< T > get_load_png (std::FILE * const file, unsigned int * const bits_per_pixel=0)

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

- static CImg< T > get_load_pnm (const char * const filename)
- static CImg< T > get_load_pnm (std::FILE * const file)

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

- static CImg< T > get_load_pfm (const char * const filename)
- static CImg< T > get_load_pfm (std::FILE * const file)

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

- 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)

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

- 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)

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

- 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, float * const voxel_size=0, CImg< charT > * const description=0)
- static CImg< T > get_load_minc2 (const char * const filename)
- static CImg< T > get_load_analyze (const char * const filename, float * const voxel_size=0)
- static CImg< T > get_load_analyze (std::FILE * const file, float * const voxel_size=0)

Load image from an ANALYZE7.5/NIFTI file [new-instance version].

- static 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)

Load image from a .cimg[z] file [new-instance version].
• 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)

Load sub-images of a .cimg file [new-instance version].
• 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)

Load sub-images of a .cimg file [new-instance version].
• static CImg< T > get_load_inr (const char *const filename, float *const voxel_size=0)

Load image from an INRIMAGE-4 file [new-instance version].
• static CImg< T > get_load_inr (std::FILE *const file, float *voxel_size=0)

Load image from an INRIMAGE-4 file [new-instance version].
• static CImg< T > get_load_off (CImgList *primitives, CImgList *colors, std::FILE *const file)

Load 3D object from a .OFF file [new-instance version].
• static CImg< T > get_load_off (const char *const filename, CImgList< tf > &primitives, CImgList< tc > &colors, const char *const filename)

Load 3D object from a .OFF file [new-instance version].
• static CImg< T > get_load_pandore (const char *const filename)

Load image from a PANDORE-5 file [new-instance version].
• static CImg< T > get_load_pandore (std::FILE *const file)

Load image from a PANDORE-5 file [new-instance version].
• static CImg< T > get_load_raw (const char *const filename, const char axis='c', const float align=0)

Load image from a raw binary file [new-instance version].
• static CImg< T > get_load_raw (std::FILE *const file, const unsigned int size_x=0, const unsigned int size_y=1, const unsigned int size_z=1, const unsigned int size_c=1, const bool is_multiplexed=false, const bool invert_endianness=false, const ulongT offset=0)

Load image from a raw binary file [new-instance version].
• static CImg< T > get_load_yuv (const char *const filename, float *const voxel_size=0)

Load image sequence from a YUV file [new-instance version].
• static CImg< T > get_load_yuv (std::FILE *const file, const unsigned int size_x=0, const unsigned int size_y=1, const unsigned int size_z=1, const unsigned int size_c=1, const bool is_multiplexed=false, const bool invert_endianness=false, const ulongT offset=0)

Load image sequence from a YUV file [new-instance version].
• template<typename tf, typename tc>
static CImg< T > get_load_off (CImgList< tf > &primitives, CImgList< tc > &colors, const char *const filename)

Load 3D object from a .OFF file [new-instance version].
• template<typename tf, typename tc>
static CImg< T > get_load_off (CImgList< tf > &primitives, CImgList< tc > &colors, std::FILE *const file)

Load 3D object from a .OFF file [new-instance version].
• static CImg< T > get_load_video (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool yuv2rgb=true, const char axis='c')

Load image sequence from a video file, using OpenCV library [new-instance version].
• static CImg< T > get_load_ffmpeg_external (const char *const filename, const char axis='z', const float align=0)

Load image sequence using FFmpeg's external tool 'ffmpeg' [new-instance version].
• static CImg< T > get_load_gif_external (const char *const filename, const char axis='z', const float align=0)

Load gif file, using ImageMagick or GraphicsMagick's external tool 'convert' [new-instance version].
• static CImg< T > get_load_graphicsmagick_external (const char *const filename)

Load gif file, using ImageMagick or GraphicsMagick's external tool 'convert' [new-instance version].
8.1 Clmg< T > Struct Template Reference

Load image using GraphicsMagick's external tool 'gm' [new-instance version].
- static Clmg< T > get_load_gzip_external (const char *const filename)
  Load gzipped image file, using external tool 'gunzip' [new-instance version].
- static Clmg< T > get_load_imagemagick_external (const char *const filename)
  Load image using ImageMagick's external tool 'convert' [new-instance version].
- static Clmg< T > get_load_medcon_external (const char *const filename)
  Load image from a DICOM file, using XMedcon's external tool 'medcon' [new-instance version].
- static Clmg< T > get_load_dcraw_external (const char *const filename)
  Load image from a RAW Color Camera file, using external tool 'dcraw' [new-instance version].
- static Clmg< T > get_load_camera (const unsigned int camera_index=0, const unsigned int capture_width=0, const unsigned int capture_height=0, const unsigned int skip_frames=0, const bool release_camera=true)
  Load image from a camera stream, using OpenCV [new-instance version].
- static Clmg< T > get_load_other (const char *const filename)
  Load image using various non-native ways [new-instance version].

Data Output

- const Clmg< T > & print (const char *const title=0, const bool display_stats=true) const
  Display information about the image data.
- const Clmg< T > & display (ClmgDisplay &disp) const
  Display image into a ClmgDisplay window.
- const Clmg< T > & display (ClmgDisplay &disp, const bool display_info, unsigned int *const XYZ=0, const bool exit_on_anykey=false) const
  Display image into a ClmgDisplay window, in an interactive way.
- const Clmg< T > & display (const char *const title=0, const bool display_info=true, unsigned int *const XYZ=0, const bool exit_on_anykey=false) const
  Display image into an interactive window.
- template<typename tp , typename tf , typename tc , typename to>
  const Clmg< T > & display_object3d (ClmgDisplay &disp, const Clmg< tp > &vertices, const ClmgList< tf > &primitives, const ClmgList< tc > &colors, const to &opacities, const bool centering=true, const int render_static=4, const int render_motion=1, const bool is_double_sided=true, const float focale=700, const float light_x=0, const float light_y=0, const float light_z=-5e8f, const float specular_lightness=0.2f, const float specular_shininess=0.1f, const bool display_axes=true, float *const pose_matrix=0, const bool exit_on_anykey=false) const
  Display object 3D in an interactive window.
- template<typename tp , typename tf , typename tc>
  const Clmg< T > & display_object3d (ClmgDisplay &disp, const Clmg< tp > &vertices, const ClmgList< tf > &primitives, const ClmgList< tc > &colors, const bool centering=true, const int render_static=4, const int render_motion=1, const bool is_double_sided=true, const float focale=700, const float light_x=0, const float light_y=0, const float light_z=-5e8f, const float specular_lightness=0.2f, const float specular_shininess=0.1f, const bool display_axes=true, float *const pose_matrix=0, const bool exit_on_anykey=false) const
  Display object 3D in an interactive window [simplification].
• 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 boolor, const int render_4), const int render_5), const bool is_double_sided=true, const float focale=700, const float light_x=0, const float light_y=0, const float light_z=-5e8f, const float specular_lightness=0.2f, const float specular_shininess=0.1f, const bool boolor display_axes=true, float =const pose_matrix=0, const bool boolor exit_on_anykey=false) const

  Display object 3D in an interactive window [simplification].

• template<typename tp, typename tf>
  const CImg< T > & display_object3d (CImgDisplay &disp, const CImg< tp > &vertices, const CImgList< tf > &primitives, const bool boolor, const int render_4), const int render_5), const bool is_double_sided=true, const float focale=700, const float light_x=0, const float light_y=0, const float light_z=-5e8f, const float specular_lightness=0.2f, const float specular_shininess=0.1f, const bool boolor display_axes=true, float =const pose_matrix=0, const bool boolor exit_on_anykey=false) const

  Display object 3D in an interactive window [simplification].

• template<typename tp>
  const CImg< T > & display_object3d (CImgDisplay &disp, const CImg< tp > &vertices, const CImgList< tf > &primitives, const bool boolor, const int render_4), const int render_5), const bool is_double_sided=true, const float focale=700, const float light_x=0, const float light_y=0, const float light_z=-5e8f, const float specular_lightness=0.2f, const float specular_shininess=0.1f, const bool boolor display_axes=true, float =const pose_matrix=0, const bool boolor exit_on_anykey=false) const

  Display object 3D in an interactive window [simplification].

• template<typename tp>
  const CImg< T > & display_object3d (CImgDisplay &disp, const CImg< tp > &vertices, const CImgList< tf > &primitives, const bool boolor, const int render_4), const int render_5), const bool is_double_sided=true, const float focale=700, const float light_x=0, const float light_y=0, const float light_z=-5e8f, const float specular_lightness=0.2f, const float specular_shininess=0.1f, const bool boolor display_axes=true, float =const pose_matrix=0, const bool boolor exit_on_anykey=false) const

  Display object 3D in an interactive window [simplification].

• template<typename tp>
  const CImg< T > & display_graph (CImgDisplay &disp, const unsigned int plot_type=1, const unsigned int digits=6) const

  Save image as a file.

• template<typename tp>
  const CImg< T > & save (const char *const filename, const int number=-1, const unsigned int digits=6) const

  Save image as an Ascii file.

• template<typename tp>
  const CImg< T > & save_ascii (const char *const filename) const

  Save image as an Ascii file [overloading].

• template<typename tp>
  const CImg< T > & save_cpp (const char *const filename) const

  Save image as a .cpp source file.

• template<typename tp>
  const CImg< T > & save_dlm (const char *const filename) const

  Save image as a DLM file.
• const CImg<T> & save_dlm (std::FILE *const file) const
  Save image as a DLM file \[overloading\].
• const CImg<T> & save_bmp (const char *const filename) const
  Save image as a BMP file.
• const CImg<T> & save_bmp (std::FILE *const file) const
  Save image as a BMP file \[overloading\].
• const CImg<T> & save_jpeg (const char *const filename, const unsigned int quality=100) const
  Save image as a JPEG file.
• const CImg<T> & save_jpeg (std::FILE *const file, const unsigned int quality=100) const
  Save image as a JPEG file \[overloading\].
• const CImg<T> & save_magick (const char *const filename, const unsigned int bytes_per_pixel=0) const
  Save image, using built-in ImageMagick++ library.
• const CImg<T> & save_png (const char *const filename, const unsigned int bytes_per_pixel=0) const
  Save image as a PNG file.
• const CImg<T> & save_png (std::FILE *const file, const unsigned int bytes_per_pixel=0) const
  Save image as a PNG file \[overloading\].
• const CImg<T> & save_pnm (const char *const filename, const unsigned int bytes_per_pixel=0) const
  Save image as a PNM file.
• const CImg<T> & save_pnm (std::FILE *const file, const unsigned int bytes_per_pixel=0) const
  Save image as a PNM file \[overloading\].
• const CImg<T> & save_pnk (const char *const filename) const
  Save image as a PNK file.
• const CImg<T> & save_pnk (std::FILE *const file) const
  Save image as a PNK file \[overloading\].
• const CImg<T> & save_pfm (const char *const filename) const
  Save image as a PFM file.
• const CImg<T> & save_pfm (std::FILE *const file) const
  Save image as a PFM file \[overloading\].
• const CImg<T> & save_rgb (const char *const filename) const
  Save image as a RGB file.
• const CImg<T> & save_rgb (std::FILE *const file) const
  Save image as a RGB file \[overloading\].
• const CImg<T> & save_rgba (const char *const filename) const
  Save image as a RGBA file.
• const CImg<T> & save_rgba (std::FILE *const file) const
  Save image as a RGBA file \[overloading\].
• const CImg<T> & save_tiff (const char *const filename, const unsigned int compression_type=0, const float *const voxel_size=0, const char *const description=0, const bool use_bigtiff=true) const
  Save image as a TIFF file.
• const CImg<T> & save_minc2 (const char *const filename, const char *const imitate_file=0) const
  Save image as a MINC2 file.
• const CImg<T> & save_analyze (const char *const filename, const float *const voxel_size=0) const
  Save image as an ANALYZE7.5 or NIFTI file.
• const CImg<T> & save_cimg (const char *const filename, const bool is_compressed=false) const
  Save image as a .cimg file.
• const CImg<T> & save_cimg (std::FILE *const file, const bool is_compressed=false) const
  Save image as a .cimg file \[overloading\].
• const CImg<T> & save_cimg (const char *const filename, const unsigned int n0, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0) const
  Save image as a sub-image into an existing .cimg file.
• 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
  
  Save image as a sub-image into an existing .cimg file [overloading].
• const CImg<T> & save_inr (const char *const filename, const float *const voxel_size=0) const
  
  Save image as an INRIMAGE-4 file.
• const CImg<T> & save_inr (std::FILE *const file, const float *const voxel_size=0) const
  
  Save image as an INRIMAGE-4 file [overloading].
• const CImg<T> & save_exr (const char *const filename) const
  
  Save image as an OpenEXR file.
• const CImg<T> & save_pandore (const char *const filename, const unsigned int colorspace=0) const
  
  Save image as a Pandore-5 file.
• const CImg<T> & save_pandore (std::FILE *const file, const unsigned int colorspace=0) const
  
  Save image as a Pandore-5 file [overloading].
• const CImg<T> & save_raw (const char *const filename, const bool is_multiplexed=false) const
  
  Save image as a raw data file.
• const CImg<T> & save_raw (std::FILE *const file, const bool is_multiplexed=false) const
  
  Save image as a raw data file [overloading].
• const CImg<T> & save_yuv (const char *const filename, const unsigned int chroma_subsampling=444, const bool is_rgb=true) const
  
  Save image as a .yuv video file.
• const CImg<T> & save_yuv (std::FILE *const file, const unsigned int chroma_subsampling=444, const bool is_rgb=true) const
  
  Save image as a .yuv video file [overloading].
• template<typename tf , typename tc >
const CImg<T> & save_off (const CImgList<tf> &primitives, const CImgList<tc> &colors, const char *const filename) const
  
  Save 3D object as an Object File Format (.off) file.
• template<typename tf , typename tc >
const CImg<T> & save_off (const CImgList<tf> &primitives, const CImgList<tc> &colors, std::FILE *const file) const
  
  Save 3D object as an Object File Format (.off) file [overloading].
• const CImg<T> & save_video (const char *const filename, const unsigned int fps=25, const char *const codec=0, const bool keep_open=false) const
  
  Save volumetric image as a video, using the OpenCV library.
• const CImg<T> & save_ffmpeg_external (const char *const filename, const unsigned int fps=25, const char *const codec=0, const unsigned int bitrate=2048) const
  
  Save volumetric image as a video, using ffmpeg external binary.
• const CImg<T> & save_gzip_external (const char *const filename) const
  
  Save image using gzip external binary.
• const CImg<T> & save_graphicsmagick_external (const char *const filename, const unsigned int quality=100) const
  
  Save image using GraphicsMagick's external binary.
• const CImg<T> & save_imagemagick_external (const char *const filename, const unsigned int quality=100) const
  
  Save image using ImageMagick's external binary.
• const CImg<T> & save_medcon_external (const char *const filename) const
  
  Save image as a Dicom file.
• const CImg<T> & save_other (const char *const filename, const unsigned int quality=100) const
• CImg<ucharT> get_serialize (const bool is_compressed=false) const
  
  Serialize a CImg<ucharT> instance into a raw CImg<ucharT> buffer.
• 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) const
8.1 CImg< T > Struct Template Reference

Save blank image as a .cimg file.

- static void save_empty_cimg (std::FILE ∗const file, const unsigned int dx, const unsigned int dy=1, const
  unsigned int dz=1, const unsigned int dc=1)
  
  Save blank image as a .cimg file [overloading].

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 de-
signed 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);` constructs a 256x256 color image from a unsigned char* buffer data_buffer (where R,G,B channels follow each others).

The complete list of constructors can be found [here](#).

Most useful functions

The `CImg<T>` class contains a lot of functions that operates on images. Some of the most useful are:

- **operator()**: Read or write pixel values.
- **display()**: displays the image in a new window.
8.1.2 Member Typedef Documentation

8.1.2.1 iterator

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

```cpp
CImg<float> img("reference.jpg"); // Load image from file
for (CImg<float>::iterator it = img.begin(), it<img.end(); ++it) *it = 0;
```

8.1.2.2 const_iterator

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

```cpp
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;
```
8.1.2.3 value_type

typedef T value_type

Pixel value type.

Refer to the type of the pixel values of an image instance.

Note

- The \texttt{CImg\textless{}T\textgreater{}::value\_type} type of a \texttt{CImg\textless{}T\textgreater{}} is defined to be a \texttt{T}.
- \texttt{CImg\textless{}T\textgreater{}::value\_type} is actually not used in \texttt{CImg} methods. It has been mainly defined for compatibility with STL naming conventions.

8.1.3 Constructor & Destructor Documentation

8.1.3.1 \texttt{\~CImg()}

\texttt{\~CImg()}\n
Destroy image.

Note

- The pixel buffer \texttt{data()} is deallocated if necessary, e.g. for non-empty and non-shared image instances.
- Destroying an empty or shared image does nothing actually.

Warning

- When destroying a non-shared image, make sure that you will not operate on a remaining shared image that shares its buffer with the destroyed instance, in order to avoid further invalid memory access (to a deallocated buffer).

8.1.3.2 \texttt{CImg() [1/13]}

\texttt{CImg()}

Construct empty image.

Note

- An empty image has no pixel data and all of its dimensions \texttt{width()}, \texttt{height()}, \texttt{depth()}, \texttt{spectrum()} are set to 0, as well as its pixel buffer pointer \texttt{data()}.
- An empty image may be re-assigned afterwards, e.g. with the family of \texttt{assign(unsigned int,unsigned int,unsigned int,unsigned int)} methods, or by \texttt{operator=(const CImg\textless{}T\textgreater{}&)}. In all cases, the type of pixels stays \texttt{T}.
- An empty image is never shared.

Example

\begin{verbatim}
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
\end{verbatim}
8.1.3.3 CImg() [2/13]

CImg (  
  const unsigned int size_x,  
  const unsigned int size_y = 1,  
  const unsigned int size_z = 1,  
  const unsigned int size_c = 1 ) [explicit]

Construct image with specified size.

Parameters

<table>
<thead>
<tr>
<th>size→_x</th>
<th>Image width().</th>
</tr>
</thead>
<tbody>
<tr>
<td>size→_y</td>
<td>Image height().</td>
</tr>
<tr>
<td>size→_z</td>
<td>Image depth().</td>
</tr>
<tr>
<td>size→_c</td>
<td>Image spectrum() (number of channels).</td>
</tr>
</tbody>
</table>

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'
```

8.1.3.4 CImg() [3/13]

CImg (  
  const unsigned int size_x,  
  const unsigned int size_y,  
  const unsigned int size_z,  
  const unsigned int size_c,  
  const T & value )

Construct image with specified size and initialize pixel values.
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size_x</td>
<td>Image width()</td>
</tr>
<tr>
<td>size_y</td>
<td>Image height()</td>
</tr>
<tr>
<td>size_z</td>
<td>Image depth()</td>
</tr>
<tr>
<td>size_c</td>
<td>Image spectrum() (number of channels)</td>
</tr>
<tr>
<td>value</td>
<td>Initialization value.</td>
</tr>
</tbody>
</table>

### Note

- Similar to CImg(unsigned int,unsigned int,unsigned int,unsigned int), but it also fills the pixel buffer with the specified value.

### Warning

- It cannot be used to construct a vector-valued image and initialize it with vector-valued pixels (e.g. RGB vector, for color images). For this task, you may use fillC() after construction.

### 8.1.3.5 CImg() [4/13]

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 \times size\_y \times size\_z \times size\_c, with pixels of type T, and initialize pixel values from the specified sequence of integers value0,value1,...

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>size_x</td>
<td>Image width()</td>
<td></td>
</tr>
<tr>
<td>size_y</td>
<td>Image height()</td>
<td></td>
</tr>
<tr>
<td>size_z</td>
<td>Image depth()</td>
<td></td>
</tr>
<tr>
<td>size_c</td>
<td>Image spectrum() (number of channels)</td>
<td></td>
</tr>
<tr>
<td>value0</td>
<td>First value of the initialization sequence (must be an integer).</td>
<td></td>
</tr>
<tr>
<td>value1</td>
<td>Second value of the initialization sequence (must be an integer).</td>
<td></td>
</tr>
</tbody>
</table>
8.1 CImg<T> Struct Template Reference

Note

• Similar to \texttt{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 \texttt{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

\begin{verbatim}
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();
\end{verbatim}

8.1.3.6 CImg() [5/13]

\texttt{CImg (}
const unsigned int \texttt{size}_x,
const unsigned int \texttt{size}_y,
const unsigned int \texttt{size}_z,
const unsigned int \texttt{size}_c,
const double \texttt{value}_0,
const double \texttt{value}_1,
...\texttt{)}

Construct image with specified size and initialize pixel values from a sequence of doubles.

Construct a new image instance of size \texttt{size}_x \texttt{x size}_y \texttt{x size}_z \texttt{x size}_c, with pixels of type \texttt{T}, and initialize pixel values from the specified sequence of doubles \texttt{value}_0, \texttt{value}_1, ...

Parameters

\begin{tabular}{|l|l|}
\hline
\texttt{size}_x & Image width(). \\
\hline
\texttt{size}_y & Image height(). \\
\hline
\texttt{size}_z & Image depth(). \\
\hline
\texttt{size}_c & Image spectrum() (number of channels). \\
\hline
\texttt{value}_0 & First value of the initialization sequence (must be a \texttt{double}). \\
\hline
\texttt{value}_1 & Second value of the initialization sequence (must be a \texttt{double}). \\
\hline
\ldots & \\
\hline
\end{tabular}

Note

• Similar to \texttt{CImg(unsigned int, unsigned int, unsigned int, unsigned int, int, int, ...)}, but takes a sequence of double values instead of integers.

Generated by Doxygen
Warning

- You must specify exactly \(dx \times dy \times dz \times dc\) 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:

```cpp
const CImg<float> img(2,2,1,1, 0.5,0.5,255,255); // FAIL: The two last arguments are 'int', not 'double'!
```

### 8.1.3.7 CImg()

**CImg**

```cpp
 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 \(size_x \times size_y \times size_z \times size_c\), with pixels of type \(T\), and initializes pixel values from the specified string \(values\).

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size_x</td>
<td>Image width().</td>
</tr>
<tr>
<td>size_y</td>
<td>Image height().</td>
</tr>
<tr>
<td>size_z</td>
<td>Image depth().</td>
</tr>
<tr>
<td>size_c</td>
<td>Image spectrum() (number of channels).</td>
</tr>
<tr>
<td>values</td>
<td>Value string describing the way pixel values are set.</td>
</tr>
<tr>
<td>repeat_values</td>
<td>Tells if the value filling process is repeated over the image.</td>
</tr>
</tbody>
</table>

**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 "\(\cos(x/10) \times \sin(y/20)\)". 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**

```cpp
const CImg<float> img1(129,129,1,3,"0,64,128,192,255",true), // Construct image from a value sequence
    img2(129,129,1,3,"if(c==0,255*abs(cos(x/10)),1.8*y)",false); // Construct image from a formula
(img1,img2).display();
```
8.1.3.8 CImg() [7/13]

```cpp
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 \(size_x \times size_y \times size_z \times size_c\), with pixels of type \(T\), and initializes pixel values from the specified \(t\) memory buffer.

**Parameters**

<table>
<thead>
<tr>
<th>values</th>
<th>Pointer to the input memory buffer.</th>
</tr>
</thead>
<tbody>
<tr>
<td>size_x</td>
<td>Image width().</td>
</tr>
<tr>
<td>size_y</td>
<td>Image height().</td>
</tr>
<tr>
<td>size_z</td>
<td>Image depth().</td>
</tr>
<tr>
<td>size_c</td>
<td>Image spectrum() (number of channels).</td>
</tr>
<tr>
<td>is_shared</td>
<td>Tells if input memory buffer must be shared by the current instance.</td>
</tr>
</tbody>
</table>

**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**

```cpp
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'
```

8.1.3.9 CImg() [8/13]

```cpp
CImg(
    const char *const filename ) [explicit]
```

Construct image from reading an image file.

Construct a new image instance with pixels of type \(T\), and initialize pixel values with the data read from an image file.
Parameters

| filename   | Filename, as a C-string. |

Note

- Similar to `CImg(unsigned int, unsigned int, unsigned int, unsigned int)`, but it reads the image dimensions and pixel values from the specified image file.
- The recognition of the image file format by CImg highly depends on the tools installed on your system and on the external libraries you used to link your code against.
- Considered pixel type `T` should better fit the file format specification, or data loss may occur during file load (e.g. constructing a `CImg<unsigned char>` from a float-valued image file).
- A `CImgIOException` is thrown when the specified `filename` cannot be read, or if the file format is not recognized.

Example

```cpp
const CImg<float> img("reference.jpg");
img.display();
```

8.1.3.10 CImg()

CImg (const CImg<t>& img)

Construct image copy.

Construct a new image instance with pixels of type `T`, as a copy of an existing `CImg<t>` instance.

Parameters

| 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).
### 8.1.3.11 CImg() [10/13]

```cpp
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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>img</td>
<td>Input image to copy.</td>
</tr>
<tr>
<td>is_shared</td>
<td>Tells about the shared state of the constructed copy.</td>
</tr>
</tbody>
</table>

**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.

### 8.1.3.12 CImg() [11/13]

```cpp
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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>img</td>
<td>Input image from which dimensions are borrowed.</td>
</tr>
<tr>
<td>dimensions</td>
<td>C-string describing the image size along the X,Y,Z and C-dimensions.</td>
</tr>
</tbody>
</table>

**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

```cpp
const CImg<float> img1(256,128,1,3), // 'img1' is a 256x128x1x3 image
img2(img1,"xyzc"), // 'img2' is a 256x128x1x3 image
img3(img1,"y,x,z,c"), // 'img3' is a 128x256x1x3 image
img4(img1,"c,x,y,3",0), // 'img4' is a 3x128x256x3 image (with pixels initialized to '0')
```

8.1.3.13 CImg()

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>img</code></td>
<td>Input image from which dimensions are borrowed.</td>
</tr>
<tr>
<td><code>dimensions</code></td>
<td>String describing the image size along the X,Y,Z and V-dimensions.</td>
</tr>
<tr>
<td><code>value</code></td>
<td>Value used for initialization.</td>
</tr>
</tbody>
</table>

Note

- Similar to CImg(const CImg<T> &, const char *), but it also fills the pixel buffer with the specified value.

8.1.3.14 CImg()

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>disp</code></td>
<td>Input display window.</td>
</tr>
</tbody>
</table>

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).
8.1 CImg< T > Struct Template Reference

- The image pixels are read as 8-bits RGB values.

8.1.4 Member Function Documentation

8.1.4.1 assign() [1/13]

CImg< T > & assign()

Construct empty image [in-place version].

In-place version of the default constructor CImg(). It simply resets the instance to an empty image.

8.1.4.2 assign() [2/13]

CImg< T > & assign(
    const unsigned int size_x,
    const unsigned int size_y = 1,
    const unsigned int size_z = 1,
    const unsigned int size_c = 1
)

Construct image with specified size [in-place version].

In-place version of the constructor CImg(unsigned int,unsigned int,unsigned int,unsigned int).

8.1.4.3 assign() [3/13]

CImg< T > & assign(
    const unsigned int size_x,
    const unsigned int size_y,
    const unsigned int size_z,
    const unsigned int size_c,
    const T & value
)

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

In-place version of the constructor CImg(unsigned int,unsigned int,unsigned int,unsigned int,T).

8.1.4.4 assign() [4/13]

CImg< T > & assign(
    const unsigned int size_x,
    const unsigned int size_y,
    const unsigned int size_z,
    const unsigned int size_c,
    const int value0,
    const int value1,
    ...
)

Construct image with specified size and initialize pixel values from a sequence of integers [in-place version].

In-place version of the constructor CImg(unsigned int,unsigned int,unsigned int,unsigned int,int,int,...).
8.1.4.5 assign() [5/13]

CImg<T>& assign (  
    const unsigned int size_x,  
    const unsigned int size_y,  
    const unsigned int size_z,  
    const unsigned int size_c,  
    const double value0,  
    const double value1,  
    ...  )

Construct image with specified size and initialize pixel values from a sequence of doubles [in-place version].

In-place version of the constructor CImg(unsigned int,unsigned int,unsigned int,unsigned int,double,double,...).

8.1.4.6 assign() [6/13]

CImg<T>& assign (  
    const unsigned int size_x,  
    const unsigned int size_y,  
    const unsigned int size_z,  
    const unsigned int size_c,  
    const char ∗const values,  
    const bool repeat_values )

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

In-place version of the constructor CImg(unsigned int,unsigned int,unsigned int,unsigned int,const char ∗,bool).

8.1.4.7 assign() [7/13]

CImg<T>& assign (  
    const t ∗const values,  
    const unsigned int size_x,  
    const unsigned int size_y = 1,  
    const unsigned int size_z = 1,  
    const unsigned int size_c = 1  )

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

In-place version of the constructor CImg(const t ∗,unsigned int,unsigned int,unsigned int,unsigned int).

8.1.4.8 assign() [8/13]

CImg<T>& assign (  
    const char ∗const filename )

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

In-place version of the constructor CImg(const char∗).
8.1.4.9 assign() [9/13]

CImg<T>& assign (const CImg<T> & img )

Construct image copy [in-place version].

In-place version of the constructor CImg(const CImg<T>&).

8.1.4.10 assign() [10/13]

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

In-place version of the advanced copy constructor.

In-place version of the constructor CImg(const CImg<T>&,bool).

8.1.4.11 assign() [11/13]

CImg<T>& assign (const CImg<T> & img, const char∗ const dimensions )

Construct image with dimensions borrowed from another image [in-place version].

In-place version of the constructor CImg(const CImg<T>&,const char∗).

8.1.4.12 assign() [12/13]

CImg<T>& assign (const CImg<T> & img, const char∗ const dimensions, const T & value )

Construct image with dimensions borrowed from another image and initialize pixel values [in-place version].

In-place version of the constructor CImg(const CImg<T>&,const char∗,T).

8.1.4.13 assign() [13/13]

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

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

In-place version of the constructor CImg(const CImgDisplay&).
8.1.4.14 clear()

CImg<\text{T}>\& clear()

Construct empty image [in-place version].

Equivalent to assign().

Note

- It has been defined for compatibility with STL naming conventions.

8.1.4.15 move_to() [1/2]

CImg<\text{t}>\& move_to (CImg<\text{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

\begin{itemize}
  \item \textit{img} \hspace{1cm} Destination image.
\end{itemize}

Note

- Pixel types \text{T} and \text{t} of source and destination images can be different, though the process is designed to be instantaneous when \text{T} and \text{t} are the same.

Example

\begin{verbatim}
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
\end{verbatim}

8.1.4.16 move_to() [2/2]

CImgList<\text{t}>\& move_to (CImgList<\text{t}>\& list,
const unsigned int pos = \text{\~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 \text{pos} in specified \text{CImgList<\text{t}>} instance.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>Destination list.</td>
</tr>
<tr>
<td>pos</td>
<td>Position of the newly inserted image in the list.</td>
</tr>
</tbody>
</table>

Note

- When optional parameter `pos` is omitted, the image instance is transferred as a new image at the end of the specified `list`.
- It is convenient to sequentially insert new images into image lists, with no additional copies of memory buffer.

Example

```cpp
CImgList<float> list; // Construct an empty image list
CImg<float> img("reference.jpg"); // Read image from filename
img.move_to(list); // Transfer image content as a new item in the list (no buffer copy)
```

8.1.4.17 swap()

```cpp
CImg<T>& swap (CImg<T>& img)
```

Swap fields of two image instances.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>img</td>
<td>Image to swap fields with.</td>
</tr>
</tbody>
</table>

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

```cpp
CImg<float> img1("lena.jpg"),
    img2("milla.jpg");
img1.swap(img2); // Now, 'img1' is 'milla' and 'img2' is 'lena'
```

8.1.4.18 empty()

```cpp
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.

```cpp
void f(const int x=0, const int y=0, const CImg<float>& img=CImg<float>::empty());
```
8.1.4.19  operator() [1/2]

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 \textit{const}, whether the image instance is \textit{const} or not. This is the standard method to get/set pixel values in \texttt{CImg<T>} images.

\textbf{Parameters}

<table>
<thead>
<tr>
<th>x</th>
<th>X-coordinate of the pixel value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>Y-coordinate of the pixel value.</td>
</tr>
<tr>
<td>z</td>
<td>Z-coordinate of the pixel value.</td>
</tr>
<tr>
<td>c</td>
<td>C-coordinate of the pixel value.</td>
</tr>
</tbody>
</table>

\textbf{Note}

- Range of pixel coordinates start from \((0,0,0,0)\) to \((\text{width()} - 1, \text{height()} - 1, \text{depth()} - 1, \text{spectrum()} - 1)\).
- Due to the particular arrangement of the pixel buffers defined in \texttt{CImg}, you can omit one coordinate if the corresponding dimension is equal to 1. For instance, pixels of a 2D image (\texttt{depth()} equal to 1) can be accessed by \texttt{img(x,y,c)} instead of \texttt{img(x,y,0,c)}.

\textbf{Warning}

- There is \textit{no} boundary checking done in this operator, to make it as fast as possible. You \textit{must} take care of out-of-bounds access by yourself, if necessary. For debugging purposes, you may want to define macro \texttt{'cimg\_verbosity'\textasciitilde3} 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.

\textbf{Example}

\begin{verbatim}
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
\end{verbatim}

8.1.4.20  operator() [2/2]

T& operator() (  
    const unsigned int x,  
    const unsigned int y,  
    const unsigned int z,  
    const unsigned int c,  
    const ulongT wh,  
    const ulongT whd = 0  
)

Access to a pixel value.
Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>X-coordinate of the pixel value.</td>
</tr>
<tr>
<td>y</td>
<td>Y-coordinate of the pixel value.</td>
</tr>
<tr>
<td>z</td>
<td>Z-coordinate of the pixel value.</td>
</tr>
<tr>
<td>c</td>
<td>C-coordinate of the pixel value.</td>
</tr>
<tr>
<td>wh</td>
<td>Precomputed offset, must be equal to width()∗height().</td>
</tr>
<tr>
<td>whd</td>
<td>Precomputed offset, must be equal to width()∗height()∗depth().</td>
</tr>
</tbody>
</table>

Note

- Similar to (but faster than) operator(). It uses precomputed offsets to optimize memory access. You may use it to optimize the reading/writing of several pixel values in the same image (e.g. in a loop).

8.1.4.21 operator T*( )

operator T* ( )

Implicitly cast an image into a T*.

Implicitly cast a CImg<T> instance into a T* or const T* pointer, whether the image instance is const or not. The returned pointer points on the first value of the image pixel buffer.

Note

- It simply returns the pointer data() to the pixel buffer.
- This implicit conversion is convenient to test the empty state of images (data() being 0 in this case), e.g.

```cpp
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
    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.

```cpp
CImg<float> img(100,100);
const float value = img[99]; // Access to value of the last pixel on the first row
img[510] = 255; // Set pixel value at (10,5)
```

8.1.4.22 operator=() [1/4]

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

```cpp
CImg<char> img(100,100); // Declare image (with garbage values)
img = 0;                 // Set all pixel values to '0'
img = 1.2;               // Set all pixel values to '1' (cast of '1.2' as a 'char')
```

8.1.4.23 `operator=` [2/4]

```cpp
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

```cpp
CImg<float> img1(100,100), img2(img1), img3(img1); // Declare 3 scalar images 100x100 with uninitialized values
img1 = "0,50,100,150,200,250,200,150,100,50"; // Set pixel values of 'img1' from a value sequence
img2 = "15*{(x*y)%25}";                      // Set pixel values of 'img2' from a formula
img3 = "reference.jpg";                      // Set pixel values of 'img3' from a file (image size is modified)
{img1, img2, img3}.display();
```
8.1.4.24 operator= [3/4]

```
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>&).

8.1.4.25 operator= [4/4]

```
CImg<T>& operator= (const CImgDisplay & disp)
```

Copy the content of a display window to the current image instance.

Similar to assign(const CImgDisplay&).

8.1.4.26 operator+= [1/3]

```
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();
```
8.1.4.27 \texttt{operator+=()} [2/3]

\begin{verbatim}
CImg\texttt{\langle T\rangle \& \ operator+= \{
    \text{const char } \ast\text{const expression }
\}}
\end{verbatim}

In-place addition operator.

Add values to image pixels, according to the specified string \texttt{expression}.

\textbf{Parameters}

\begin{tabular}{|l|l|}
\hline
\texttt{expression} & Value string describing the way pixel values are added. \\
\hline
\end{tabular}

\textbf{Note}

\begin{itemize}
\item Similar to \texttt{operator=(const char \&)}, except that it adds values to the pixels of the current image instance, instead of assigning them.
\end{itemize}

8.1.4.28 \texttt{operator+=()} [3/3]

\begin{verbatim}
CImg\texttt{\langle T\rangle \& \ operator+= \{
    \text{const CImg\langle t \rangle } \&\ \text{img }
\}}
\end{verbatim}

In-place addition operator.

Add values to image pixels, according to the values of the input image \texttt{img}.

\textbf{Parameters}

\begin{tabular}{|l|l|}
\hline
\texttt{img} & Input image to add. \\
\hline
\end{tabular}

\textbf{Note}

\begin{itemize}
\item The size of the image instance is never modified.
\item It is not mandatory that input image \texttt{img} has the same size as the image instance. If less values are available in \texttt{img}, then the values are added periodically. For instance, adding one WxH scalar image (\texttt{spectrumb} equal to 1) to one WxH color image (\texttt{spectrumb} equal to 3) means each color channel will be incremented with the same values at the same locations.
\end{itemize}

\textbf{Example}

\begin{verbatim}
CImg\texttt{\langle float\rangle \ img1("reference.jpg"); // Load a RGB color image (img1.spectrum()==3)
    \ \ // Construct a scalar shading (img2.spectrum()==1).
    \ const CImg\texttt{\langle float\rangle \ img2(img1.width(),img.height(),1,1,"255*(x/w)^2");}
    \ img1+=img2; // Add shading to each channel of 'img1'
    \ img1.cut(0,255); // Prevent [0,255] overflow
    \ (img2,img1).display();
\end{verbatim}
8.1.4.29 `operator++()` [1/2]

\[ \text{CImg} < T > & \text{operator++} ( ) \]

In-place increment operator (prefix).

Add 1 to all image pixels, and return a reference to the current incremented image instance.

**Note**

- Writing `++img` is equivalent to `img+=1`.

8.1.4.30 `operator++()` [2/2]

\[ \text{CImg} < T > \text{operator++} ( \text{int} ) \]

In-place increment operator (postfix).

Add 1 to all image pixels, and return a new copy of the initial (pre-incremented) image instance.

**Note**

- Use the prefixed version `operator++()` if you don't need a copy of the initial (pre-incremented) image instance, since a useless image copy may be expensive in terms of memory usage.

8.1.4.31 `operator+()` [1/4]

\[ \text{CImg} < T > \text{operator+} ( ) \text{const} \]

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 > (\text{const CImg} < T > \&)` returns a shared copy of a shared input image, and it may be not desirable to work on a regular copy (e.g. for a resize operation) if you have no information about the shared state of the input image.

- Writing `(+img)` is equivalent to `CImg < T > (img, false)`.
8.1.4.32 operator+() [2/4]

\begin{verbatim}
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.
\end{verbatim}

8.1.4.33 operator+() [3/4]

\begin{verbatim}
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.
\end{verbatim}

8.1.4.34 operator+() [4/4]

\begin{verbatim}
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.
\end{verbatim}

8.1.4.35 operator-=() [1/3]

\begin{verbatim}
CImg<T>& operator-= ( const t value )

In-place subtraction operator.

Similar to operator+=(const t), except that it performs a subtraction instead of an addition.
\end{verbatim}

8.1.4.36 operator-=() [2/3]

\begin{verbatim}
CImg<T>& operator-= ( const char *const expression )

In-place subtraction operator.

Similar to operator+=(const char*), except that it performs a subtraction instead of an addition.
\end{verbatim}
8.1.4.37 operator-() [3/3]

```
CImg<T>& operator- ( const CImg<T>& img )
```

In-place subtraction operator.

Similar to `operator+=(const CImg<T>&)`, except that it performs a subtraction instead of an addition.

8.1.4.38 operator--() [1/2]

```
CImg<T>& operator-- ( )
```

In-place decrement operator (prefix).

Similar to `operator++()`, except that it performs a decrement instead of an increment.

8.1.4.39 operator--() [2/2]

```
CImg<T> operator-- ( int )
```

In-place decrement operator (postfix).

Similar to `operator++(int)`, except that it performs a decrement instead of an increment.

8.1.4.40 operator-() [1/4]

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

Replace each pixel by its opposite value.

Note

- If the computed opposite values are out-of-range, they are treated as with standard C++ numeric types. For instance, the `unsigned char` opposite of 1 is 255.

Example

```
const CImg<unsigned char> img1("reference.jpg"); // Load a RGB color image
img2 = -img1; // Compute its opposite (in 'unsigned char')
(img1, img2).display();
```

8.1.4.41 operator-() [2/4]

```
CImg<typename cimg::superset<T,t>::type > operator- ( const t value ) const
```

Subtraction operator.

Similar to `operator-=(const t)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type `T`, if necessary.
8.1.4.42 **operator-()** [3/4]

```cpp
CImg<Tfloat> operator- ( const char *const expression ) const
```

Subtraction operator.

Similar to `operator-=` (const char*), except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type `T`, if necessary.

8.1.4.43 **operator-()** [4/4]

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

Subtraction operator.

Similar to `operator-=` (const CImg<T>&), except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type `T`, if necessary.

8.1.4.44 **operator\*=()** [1/3]

```cpp
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.

8.1.4.45 **operator\*=()** [2/3]

```cpp
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.

8.1.4.46 **operator\*=()** [3/3]

```cpp
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**

<table>
<thead>
<tr>
<th><code>img</code></th>
<th>Second operand of the matrix multiplication.</th>
</tr>
</thead>
</table>

Generated by Doxygen
8.1 CImg< T > Struct Template Reference

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

```cpp
CImg<float> A(2,2,1,1, 1,2,3,4); // Construct 2x2 matrix A = [1,2;3,4]
const CImg<float> X(1,2,1,1, 1,2); // Construct 1x2 vector X = [1;2]
A*=X; // Assign matrix multiplication A*X to 'A'
// 'A' is now a 1x2 vector whose values are [5;11].
```

8.1.4.47 `operator*( )` [1/3]

```cpp
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.

8.1.4.48 `operator*( )` [2/3]

```cpp
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.

8.1.4.49 `operator*( )` [3/3]

```cpp
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.

8.1.4.50 `operator/=()` [1/3]

```cpp
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.
8.1.4.51 \texttt{operator/()} [2/3]

\texttt{CImg\textless{}T\textgreater{}\& operator/ \{ \\
\quad \text{const char } \ast \text{const expression } \}}

In-place division operator.

Similar to \texttt{operator\+=(const char\*)}, except that it performs a division instead of an addition.

8.1.4.52 \texttt{operator/()} [3/3]

\texttt{CImg\textless{}T\textgreater{}\& operator/ \{ \\
\quad \text{const CImg\textless{} t } \& \text{ img } \}}

In-place division operator.

Replace the image instance by the (right) matrix division between the image instance and the specified matrix \texttt{img}.

Parameters

- \texttt{img} Second operand of the matrix division.

Note

- It does \textit{not} compute a pointwise division between two images. For this purpose, use \texttt{div(const CImg\textless{}T\textgreater{}\&)} instead.
- It returns the matrix operation \texttt{A*inverse(img)}.
- The size of the image instance can be modified by this operator.

8.1.4.53 \texttt{operator/()} [1/3]

\texttt{CImg\textless{}typename cimg::superset\lt{}T,t\gt{}::type \textgreater{} operator/ \{( \\
\quad \text{const t value } \}} \text{const}

Division operator.

Similar to \texttt{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 \texttt{T}, if necessary.

8.1.4.54 \texttt{operator/()} [2/3]

\texttt{CImg\textless{}T\textgreater{}float operator/ \{ \\
\quad \text{const char } \ast \text{const expression } \}} \text{const}

Division operator.

Similar to \texttt{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 \texttt{T}, if necessary.
8.1.4.55  \texttt{\textbf{operator/}}  \[3/3\]

\begin{verbatim}
CImg< typename cimg::superset<T,t>::type > operator/ (  
    const CImg< t >& img ) const
\end{verbatim}

Division operator.

Similar to \texttt{\textbf{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.

8.1.4.56  \texttt{\textbf{operator\%=}}  \[1/3\]

\begin{verbatim}
CImg<T>& operator\%= {  
    const t value }
\end{verbatim}

In-place modulo operator.

Similar to \texttt{\textbf{operator\+=}=(const t)}, except that it performs a modulo operation instead of an addition.

8.1.4.57  \texttt{\textbf{operator\%=}}  \[2/3\]

\begin{verbatim}
CImg<T>& operator\%= {  
    const char *const expression }
\end{verbatim}

In-place modulo operator.

Similar to \texttt{\textbf{operator\+=}=(const char*)}, except that it performs a modulo operation instead of an addition.

8.1.4.58  \texttt{\textbf{operator\%=}}  \[3/3\]

\begin{verbatim}
CImg<T>& operator\%= {  
    const CImg< t >& img }
\end{verbatim}

In-place modulo operator.

Similar to \texttt{\textbf{operator\+=}=(const CImg<T>&)}, except that it performs a modulo operation instead of an addition.

8.1.4.59  \texttt{\textbf{operator\%}}  \[1/3\]

\begin{verbatim}
CImg< typename cimg::superset<T,t>::type > operator\% {  
    const t value } const
\end{verbatim}

Modulo operator.

Similar to \texttt{\textbf{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.
8.1.4.60  operator%() [2/3]

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.

8.1.4.61  operator%() [3/3]

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.

8.1.4.62  operator &=() [1/3]

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.

8.1.4.63  operator &=() [2/3]

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.

8.1.4.64  operator &=() [3/3]

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.
8.1.4.65 operator &() [1/3]

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.

8.1.4.66 operator &() [2/3]

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.

8.1.4.67 operator &() [3/3]

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.

8.1.4.68 operator "|=() [1/3]

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.

8.1.4.69 operator "|=() [2/3]

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.
8.1.4.70 \texttt{operator\texttt{\|}=() [3/3]}

\texttt{CImg<\tt T>& operator\texttt{|=} (const \texttt{CImg<\tt T>&} \texttt{img})}

In-place bitwise OR operator.

Similar to \texttt{operator\texttt{+}=(const \texttt{CImg<\tt T>&})}, except that it performs a bitwise OR operation instead of an addition.

8.1.4.71 \texttt{operator\texttt{\|}() [1/3]}

\texttt{CImg<\tt T> operator\texttt{|} (const \tt value) const}

Bitwise OR operator.

Similar to \texttt{operator\texttt{|}=(const \tt)}, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is \texttt{\tt}.

8.1.4.72 \texttt{operator\texttt{\|}() [2/3]}

\texttt{CImg<\tt T> operator\texttt{|} (const \texttt{char }*\texttt{expression}) const}

Bitwise OR operator.

Similar to \texttt{operator\texttt{|}=(const \texttt{char}* )}, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is \texttt{\tt}.

8.1.4.73 \texttt{operator\texttt{\|}() [3/3]}

\texttt{CImg<\tt T> operator\texttt{|} (const \texttt{CImg<\tt T>&} \texttt{img}) const}

Bitwise OR operator.

Similar to \texttt{operator\texttt{|}=(const \texttt{CImg<\tt T>&})}, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is \texttt{\tt}.

8.1.4.74 \texttt{operator\texttt{^}=() [1/3]}

\texttt{CImg<\tt T>& operator\texttt{^}= (const \tt value )}

In-place bitwise XOR operator.

Similar to \texttt{operator\texttt{+=}(const \tt)}, except that it performs a bitwise XOR operation instead of an addition.

Warning

- It does \textit{not} compute the \textit{power} of pixel values. For this purpose, use \texttt{pow(const \tt)} instead.
8.1.4.75 operator\^{}() [2/3]

\texttt{CImg<T>\& operator\^{} = (const char \*const expression)}

In-place bitwise XOR operator.

Similar to \texttt{operator+=(const char\*)}, except that it performs a bitwise XOR operation instead of an addition.

Warning

- It does \textit{not} compute the \textit{power} of pixel values. For this purpose, use \texttt{pow(const char\*)} instead.

8.1.4.76 operator\^{}() [3/3]

\texttt{CImg<T>\& operator\^{} = (const CImg<T>& img)}

In-place bitwise XOR operator.

Similar to \texttt{operator+=(const CImg<T>&)}, except that it performs a bitwise XOR operation instead of an addition.

Warning

- It does \textit{not} compute the \textit{power} of pixel values. For this purpose, use \texttt{pow(const CImg<T>&)} instead.

8.1.4.77 operator\(^{}()) [1/3]

\texttt{CImg<T> operator\(^{} (const t value) const}

Bitwise XOR operator.

Similar to \texttt{operator\(^{}=(const t)}, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is \texttt{T}.

8.1.4.78 operator\(^{}()) [2/3]

\texttt{CImg<T> operator\(^{} (const char \*const expression) const}

Bitwise XOR operator.

Similar to \texttt{operator\(^{}=(const char\*)}, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is \texttt{T}.

Generated by Doxygen
8.1.4.79  \texttt{operator\^{}() [3/3]}

\begin{verbatim}
CImg\textless T\textgreater{} \texttt{operator\^{} (}
    \texttt{const CImg\textless t \textgreater{} & img) const}
\end{verbatim}

Bitwise XOR operator.

Similar to \texttt{operator\^{}=(const CImg\textless t\textgreater{}&)}, 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  \texttt{operator<<() [1/3]}

\begin{verbatim}
CImg\textless T\textgreater{}& \texttt{operator\<< (}
    \texttt{const t value )}
\end{verbatim}

In-place bitwise left shift operator.

Similar to \texttt{operator+=(const t)}, except that it performs a bitwise left shift instead of an addition.

8.1.4.81  \texttt{operator<<() [2/3]}

\begin{verbatim}
CImg\textless T\textgreater{}& \texttt{operator\<< (}
    \texttt{const char *const expression )}
\end{verbatim}

In-place bitwise left shift operator.

Similar to \texttt{operator+=(const char*)}, except that it performs a bitwise left shift instead of an addition.

8.1.4.82  \texttt{operator<<() [3/3]}

\begin{verbatim}
CImg\textless T\textgreater{}& \texttt{operator\<< (}
    \texttt{const CImg\textless t \textgreater{} & img )}
\end{verbatim}

In-place bitwise left shift operator.

Similar to \texttt{operator+=(const CImg\textless t\textgreater{}&)}, except that it performs a bitwise left shift instead of an addition.

8.1.4.83  \texttt{operator<<() [1/3]}

\begin{verbatim}
CImg\textless T\textgreater{} \texttt{operator\<< (}
    \texttt{const t value ) const}
\end{verbatim}

Bitwise left shift operator.

Similar to \texttt{operator<<=(const t)}, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is \( T \).
8.1.4.84 operator\(<\() [2/3]$

\texttt{CImg<\(T\)} & \texttt{operator\(<\() \{
\text{const char *const expression } \text{ const}
\}

Bitwise left shift operator.

Similar to \texttt{operator\(<\()=(\text{const char*}), \text{ except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is } T.\)

8.1.4.85 operator\(<\() [3/3]$

\texttt{CImg<\(T\)} & \texttt{operator\(<\() \{
\text{const CImg< } t > & \text{ img } \text{ const}
\}

Bitwise left shift operator.

Similar to \texttt{operator\(<\()=(\text{const CImg<}\(t\)\&), \text{ 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 operator\(\geq\)() [1/3]$

\texttt{CImg<\(T\)} & \texttt{operator\(\geq\)() \{
\text{const } t \text{ value }
\}

In-place bitwise right shift operator.

Similar to \texttt{operator\(+=\)(\text{const } t), \text{ except that it performs a bitwise right shift instead of an addition.}\)

8.1.4.87 operator\(\geq\)() [2/3]$

\texttt{CImg<\(T\)} & \texttt{operator\(\geq\)() \{
\text{const char *const expression }
\}

In-place bitwise right shift operator.

Similar to \texttt{operator\(+=\)(\text{const char*}), \text{ except that it performs a bitwise right shift instead of an addition.}\)

8.1.4.88 operator\(\geq\)() [3/3]$

\texttt{CImg<\(T\)} & \texttt{operator\(\geq\)() \{
\text{const CImg< } t > & \text{ img }
\}

In-place bitwise right shift operator.

Similar to \texttt{operator\(+=\)(\text{const CImg<}\(t\)\&), \text{ except that it performs a bitwise right shift instead of an addition.}\)
8.1.4.89 operator>>() [1/3]

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.

8.1.4.90 operator>>() [2/3]

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.

8.1.4.91 operator>>() [3/3]

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.

8.1.4.92 operator~()  

CImg<T> operator~ ( ) const

Bitwise inversion operator.

Similar to operator-(), except that it compute the bitwise inverse instead of the opposite value.

8.1.4.93 operator==() [1/3]

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. |
8.1.4.94 \texttt{operator==()} [2/3]

\begin{verbatim}
bool operator== (  
  const char *const expression ) const
\end{verbatim}

Test if all pixel values of an image follow a specified expression.  
Return \texttt{true} if all pixels of the image instance are equal to the specified \texttt{expression}.

\textbf{Parameters}

\begin{itemize}
  \item \texttt{expression} Value string describing the way pixel values are compared.
\end{itemize}

8.1.4.95 \texttt{operator==()} [3/3]

\begin{verbatim}
bool operator== (  
  const CImg< t > & img ) const
\end{verbatim}

Test if two images have the same size and values.  
Return \texttt{true} if the image instance and the input image \texttt{img} have the same dimensions and pixel values, and \texttt{false} otherwise.

\textbf{Parameters}

\begin{itemize}
  \item \texttt{img} Input image to compare with.
\end{itemize}

\textbf{Note}

\begin{itemize}
  \item The pixel buffer pointers \texttt{data()} of the two compared images do not have to be the same for \texttt{operator==()} to return \texttt{true}. Only the dimensions and the pixel values matter. Thus, the comparison can be \texttt{true} even for different pixel types \texttt{T} and \texttt{t}.
\end{itemize}

\textbf{Example}

\begin{verbatim}
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.");
}
\end{verbatim}

8.1.4.96 \texttt{operator!=()} [1/3]

\begin{verbatim}
bool operator!=(  
  const t value ) const
\end{verbatim}

Test if pixels of an image are all different from a value.  
Return \texttt{true} if all pixels of the image instance are different than the specified \texttt{value}.  

\textbf{Example}

\begin{verbatim}
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.");
}
\end{verbatim}
Parameters

| value | Reference value to compare with. |

8.1.4.97  operator"!=()  [2/3]

```cpp
bool operator!= (  
    const char *const expression ) const
```

Test if all pixel values of an image are different from a specified expression.

Return `true` if all pixels of the image instance are different to the specified `expression`.

Parameters

| expression | Value string describing the way pixel values are compared. |

8.1.4.98  operator"!=()  [3/3]

```cpp
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)`.  

8.1.4.99  operator,()  [1/2]

```cpp
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 \( \text{img} \) 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

```cpp
const CImg<float> img1("reference.jpg"),
    img2 = img1.get_mirror('x'),
    img3 = img2.get_blur(5);
const CImgList<float> list = (img1,img2); // Create list of two elements from 'img1' and 'img2'
{list,img3}.display(); // Display image list containing copies of 'img1','img2'
    and 'img3'
```

8.1.4.100 operator,() [2/2]

CImgList\( < \text{typename cimg::superset}\,< T , t > ::\text{type} > \) 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 \( \text{list} \), from positions [1] to [list.size()].

Parameters

| list | Input image list that will be appended to the image instance. |
8.1 CImg< T > Struct Template Reference

8.1.4.101 \texttt{operator<()}

\begin{verbatim}
CImgList< T > operator< ( 
    const char axis ) const
\end{verbatim}

Split image along specified axis.

Return a new list of images (CImgList instance) containing the split components of the instance image along the specified axis.

\textbf{Parameters}

\begin{tabular}{|l|}
\hline
axis \hspace{1cm} Splitting axis (can be 'x', 'y', 'z' or 'c') \\
\hline
\end{tabular}

\textbf{Note}

- Similar to \texttt{get_split(char,int) const}, with default second argument.

\textbf{Example}

\begin{verbatim}
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();
\end{verbatim}

8.1.4.102 \texttt{pixel_type()}

\begin{verbatim}
static const char* pixel_type () [static]
\end{verbatim}

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).

\textbf{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.
8.1.4.103 width()

int width() const

Return the number of image columns.

Return the image width, i.e. the image dimension along the X-axis.

Note

- The width() of an empty image is equal to 0.
- width() is typically equal to 1 when considering images as vectors for matrix calculations.
- width() returns an int, although the image width is internally stored as an unsigned int. Using an int is safer and prevents arithmetic traps possibly encountered when doing calculations involving unsigned int variables. Access to the initial unsigned int variable is possible (though not recommended) by (*this)._width.

8.1.4.104 height()

int height() const

Return the number of image rows.

Return the image height, i.e. the image dimension along the Y-axis.

Note

- The height() of an empty image is equal to 0.
- height() returns an int, although the image height is internally stored as an unsigned int. Using an int is safer and prevents arithmetic traps possibly encountered when doing calculations involving unsigned int variables. Access to the initial unsigned int variable is possible (though not recommended) by (*this)._height.

8.1.4.105 depth()

int depth() const

Return the number of image slices.

Return the image depth, i.e. the image dimension along the Z-axis.

Note

- The depth() of an empty image is equal to 0.
- depth() is typically equal to 1 when considering usual 2D images. When depth() > 1, the image is said to be volumetric.
- depth() returns an int, although the image depth is internally stored as an unsigned int. Using an int is safer and prevents arithmetic traps possibly encountered when doing calculations involving unsigned int variables. Access to the initial unsigned int variable is possible (though not recommended) by (*this)._depth.
8.1.4.106  spectrum()

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 stand 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.

8.1.4.107  size()

ulongT size() const

Return the total number of pixel values.

Return width() * height() * depth() * spectrum(), i.e. the total number of values of type T in the pixel buffer of the image instance.

Note

- The size() of an empty image is equal to 0.
- The allocated memory size for a pixel buffer of a non-shared CImg<T> instance is equal to size() * sizeof(T).

Example

```cpp
const CImg<float> img(100,100,1,3); // Construct new 100x100 color image
if (img.size() == 30000) // Test succeeds
    std::printf("Pixel buffer uses %lu bytes", img.size() * sizeof(float));
```
8.1.4.108 data() [1/2]

T* data ()

Return a pointer to the first pixel value.

Return a T*, or a const T* pointer to the first value in the pixel buffer of the image instance, whether the instance is const or not.

Note

- The data() of an empty image is equal to 0 (null pointer).
- The allocated pixel buffer for the image instance starts from data() and goes to data() + size() - 1 (included).
- To get the pointer to one particular location of the pixel buffer, use data(unsigned int, unsigned int, unsigned int, unsigned int) instead.

8.1.4.109 data() [2/2]

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>X-coordinate of the pixel value.</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>Y-coordinate of the pixel value.</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>Z-coordinate of the pixel value.</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>C-coordinate of the pixel value.</td>
<td></td>
</tr>
</tbody>
</table>

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).

8.1.4.110 offset()

longT offset ( 
    const int  x,
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

```cpp
const CImg<float> img(100, 100, 1, 3); // Define a 100x100 RGB-color image
const long off = img.offset(10, 10, 0, 2); // Get the offset of the blue value of the pixel located at (10,10)
const float val = img[off]; // Get the blue value of this pixel
```

### 8.1.4.111 begin()

**iterator** begin ()

Return a `CImg<T>::iterator` pointing to the first pixel value.

### Note

- Equivalent to `data()`.
- It has been mainly defined for compatibility with STL naming conventions.

### 8.1.4.112 end()

**iterator** end ()

Return a `CImg<T>::iterator` pointing next to the last pixel value.

### Note

- Writing `img.end()` is equivalent to `img.data() + img.size()`.
- It has been mainly defined for compatibility with STL naming conventions.
Warning

- The returned iterator actually points to a value located outside the acceptable bounds of the pixel buffer. Trying to read or write the content of the returned iterator will probably result in a crash. Use it mainly as a strict upper bound for a `CImg<T>::iterator`.

Example

```cpp
CImg<float> img(100,100,1,3); // Define a 100x100 RGB color image
// 'img.end()' used below as an upper bound for the iterator.
for (CImg<float>::iterator it = img.begin(); it<img.end(); ++it)
  *it = 0;
```

8.1.4.113 front()

T& front ()

Return a reference to the first pixel value.

Note

- Writing `img.front()` is equivalent to `img[0]`, or `img(0,0,0,0)`.
- It has been mainly defined for compatibility with STL naming conventions.

8.1.4.114 back()

T& back ()

Return a reference to the last pixel value.

Note

- Writing `img.back()` is equivalent to `img[img.size() - 1]`, or `img(img.width()-1,img.height()-1,img.depth()-1,img.spectrum()-1)`.
- It has been mainly defined for compatibility with STL naming conventions.

8.1.4.115 at() [1/2]

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

<table>
<thead>
<tr>
<th>offset</th>
<th>Offset to the desired pixel value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out_value</td>
<td>Default value returned if offset is outside image bounds.</td>
</tr>
</tbody>
</table>

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.

8.1.4.116 at() [2/2]

```cpp
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)`.

8.1.4.117 atX() [1/2]

```cpp
t& atX(
    const int x,
    const int y,
    const int z,
    const int c,
    const T & out_value
)
```

Generated by Doxygen
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

<table>
<thead>
<tr>
<th>x</th>
<th>X-coordinate of the pixel value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>Y-coordinate of the pixel value.</td>
</tr>
<tr>
<td>z</td>
<td>Z-coordinate of the pixel value.</td>
</tr>
<tr>
<td>c</td>
<td>C-coordinate of the pixel value.</td>
</tr>
<tr>
<td>out_value</td>
<td>Default value returned if ((x,y,z,c)) is outside image bounds.</td>
</tr>
</tbody>
</table>

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.

8.1.4.118 atX() [2/2]

```cpp
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

<table>
<thead>
<tr>
<th>x</th>
<th>X-coordinate of the pixel value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>Y-coordinate of the pixel value.</td>
</tr>
<tr>
<td>z</td>
<td>Z-coordinate of the pixel value.</td>
</tr>
<tr>
<td>c</td>
<td>C-coordinate of the pixel value.</td>
</tr>
</tbody>
</table>

Note

- Similar to `at(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)`. 

Generated by Doxygen
Warning

- There is no boundary checking performed for the Y,Z and C-coordinates, so they must be inside image bounds.

8.1.4.119 **atXY()** [1/2]

```cpp
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.

8.1.4.120 **atXY()** [2/2]

```cpp
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 `_atX< Y(int,int,int,int)`.

8.1.4.121 **atXYZ()** [1/2]

```cpp
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,const T)`, except that boundary checking is performed both on X,Y and Z-coordinates.
8.1.4.122 atXYZ() [2/2]

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 _atXY←Z(int,int,int,int).

8.1.4.123 atXYZC() [1/2]

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,const T), except that boundary checking is performed on all X,Y,Z and C-coordinates.

8.1.4.124 atXYZC() [2/2]

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), 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 _atXY←C(int,int,int,int).
8.1.4.125  linear_atX()  [1/2]

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fx</td>
<td>X-coordinate of the pixel value (float-valued).</td>
</tr>
<tr>
<td>y</td>
<td>Y-coordinate of the pixel value.</td>
</tr>
<tr>
<td>z</td>
<td>Z-coordinate of the pixel value.</td>
</tr>
<tr>
<td>c</td>
<td>C-coordinate of the pixel value.</td>
</tr>
<tr>
<td>out_value</td>
<td>Default value returned if ((fx,y,z,c)) is outside image bounds.</td>
</tr>
</tbody>
</table>

Note

- Similar to \(\text{atX}(\text{int, int, int, const } T)\), except that the returned pixel value is approximated by a linear interpolation along the X-axis, if corresponding coordinates are not integers.
- The type of the returned pixel value is extended to \(\text{float}\), if the pixel type \(T\) is not float-valued.

Warning

- There is \textit{no} boundary checking performed for the \(Y, Z\) and \(C\)-coordinates, so they must be inside image bounds.

8.1.4.126 \texttt{linear\_atX() \[2/2\]}

\[
\text{float linear\_atX (}
\text{const float } fx, \\
\text{const int } y = 0, \\
\text{const int } z = 0, \\
\text{const int } c = 0 ) \text{ const}
\]

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fx</td>
<td>X-coordinate of the pixel value (float-valued).</td>
</tr>
<tr>
<td>y</td>
<td>Y-coordinate of the pixel value.</td>
</tr>
<tr>
<td>z</td>
<td>Z-coordinate of the pixel value.</td>
</tr>
<tr>
<td>c</td>
<td>C-coordinate of the pixel value.</td>
</tr>
</tbody>
</table>

Note

- Similar to \texttt{linear\_atX(float, 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 \textit{not} empty, you may rather use the slightly faster method \texttt{\_linear\_atX(float, int, int, int)}. 

Generated by Doxygen
Warning

- There is no boundary checking performed for the Y, Z and C-coordinates, so they must be inside image bounds.

8.1.4.127 linear_atXY() [1/2]

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.

8.1.4.128 linear_atXY() [2/2]

Tfloat linear_atXY (const float fx, const float fy, const float fz, const int c, const T & out_value ) 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).

8.1.4.129 linear_atXYZ() [1/2]

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.
8.1.4.130  \texttt{linear_atXYZ}() [2/2]

\begin{verbatim}
Tfloat linear_atXYZ (  
    const float fx,  
    const float fy = 0,  
    const float fz = 0,  
    const int c = 0 ) const
\end{verbatim}

Return pixel value, using linear interpolation and Neumann boundary conditions for the X,Y and Z-coordinates.

Similar to \texttt{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 \textit{not} empty, you may rather use the slightly faster method \texttt{\_linear\_atXYZ(float,float,float,int)}.

8.1.4.131  \texttt{linear_atXYZC}() [1/2]

\begin{verbatim}
Tfloat linear_atXYZC (  
    const float fx,  
    const float fy,  
    const float fz,  
    const float fc,  
    const T & out_value ) const
\end{verbatim}

Return pixel value, using linear interpolation and Dirichlet boundary conditions for all X,Y,Z,C-coordinates.

Similar to \texttt{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.

8.1.4.132  \texttt{linear_atXYZC}() [2/2]

\begin{verbatim}
Tfloat linear_atXYZC (  
    const float fx,  
    const float fy = 0,  
    const float fz = 0,  
    const float fc = 0 ) const
\end{verbatim}

Return pixel value, using linear interpolation and Neumann boundary conditions for all X,Y,Z and C-coordinates.

Similar to \texttt{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 \textit{not} empty, you may rather use the slightly faster method \texttt{\_linear\_atXYZC(float,float,float,float)}.
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. The cubic interpolation uses Hermite splines.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fx</strong></td>
<td>X-coordinate of the pixel value (float-valued).</td>
</tr>
<tr>
<td><strong>y</strong></td>
<td>Y-coordinate of the pixel value.</td>
</tr>
<tr>
<td><strong>z</strong></td>
<td>Z-coordinate of the pixel value.</td>
</tr>
<tr>
<td><strong>c</strong></td>
<td>C-coordinate of the pixel value.</td>
</tr>
<tr>
<td><strong>out_value</strong></td>
<td>Default value returned if ((fx, y, z, c)) is outside image bounds.</td>
</tr>
</tbody>
</table>

**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.

Return clamped pixel value, using cubic interpolation and Dirichlet boundary conditions for the X-coordinate.

Similar to `cubic_atX(float,int,int,int,const T) const`, except that the return value is clamped to stay in the min/max range of the datatype \(T\).
8.1.4.135 **cubic_atX** [2/2]

```cpp
T float 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. The cubic interpolation uses Hermite splines.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(fx)</td>
<td>X-coordinate of the pixel value (float-valued).</td>
</tr>
<tr>
<td>(y)</td>
<td>Y-coordinate of the pixel value.</td>
</tr>
<tr>
<td>(z)</td>
<td>Z-coordinate of the pixel value.</td>
</tr>
<tr>
<td>(c)</td>
<td>C-coordinate of the pixel value.</td>
</tr>
</tbody>
</table>

**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.

8.1.4.136 **cubic_atX_c** [2/2]

```cpp
T cubic_atX_c ( 
    const float fx, 
    const int y, 
    const int z, 
    const int c ) const
```

Return clamped pixel value, using cubic interpolation and Neumann boundary conditions for the X-coordinate.

Similar to `cubic_atX(float,int,int,int) const`, except that the return value is clamped to stay in the min/max range of the datatype \(T\).
8.1.4.137 cubic_atXY() [1/2]

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,const T) const, except that the cubic interpolation and boundary checking are achieved both for X and Y-coordinates.

8.1.4.138 cubic_atXY_c() [1/2]

T cubic_atXY_c {
    const float fx,
    const float fy,
    const int z,
    const int c,
    const T & out_value } const

Return clamped pixel value, using cubic interpolation and Dirichlet boundary conditions for the X,Y-coordinates.

Similar to cubic_atX(float,float,int,int,const T) const, except that the return value is clamped to stay in the min/max range of the datatype T.

8.1.4.139 cubic_atXY() [2/2]

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).
8.1.4.140 cubic_atXY_c() [2/2]

T cubic_atXY_c (const float fx, const float fy, const int z, const int c) const

Return clamped pixel value, using cubic interpolation and Neumann boundary conditions for the X,Y-coordinates.
Similar to cubic_atXY(float,float,int,int) const, except that the return value is clamped to stay in the min/max range of the datatype T.

8.1.4.141 cubic_atXYZ() [1/2]

T float 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.

8.1.4.142 cubic_atXYZ_c() [1/2]

T cubic_atXYZ_c (const float fx, const float fy, const float fz, const int c, const T & out_value) const

Return clamped pixel value, using cubic interpolation and Dirichlet boundary conditions for the XYZ-coordinates.
Similar to cubic_atXYZ(float,float,float,int,const T) const, except that the return value is clamped to stay in the min/max range of the datatype T.

8.1.4.143 cubic_atXYZ() [2/2]

T float 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).
8.1.4.144 cubic_atXYZ_c()

T cubic_atXYZ_c (const float fx, const float fy, const float fz, const int c) const

Return clamped pixel value, using cubic interpolation and Neumann boundary conditions for the XYZ-coordinates.

Similar to cubic_atXYZ(float,float,float,int) const, except that the return value is clamped to stay in the min/max range of the datatype T.

8.1.4.145 cubic_atXYZ_p()

T float cubic_atXYZ_p (const float fx, const float fy, const float fz, const int c = 0) const

Return pixel value, using cubic interpolation and Neumann boundary conditions for the X,Y and Z-coordinates.

Similar to cubic_atX(float,int,int,int) const, except that the cubic interpolation and boundary checking are achieved both for X,Y and Z-coordinates.

Note

- If you know your image instance is not empty, you may rather use the slightly faster method _cubic←_atXYZ(float,float,float,int).

8.1.4.146 set_linear_atX()

CImg<T>& set_linear_atX (const T & value, const float fx, const int y = 0, const int z = 0, const int c = 0, const bool is_added = false)

Set pixel value, using linear interpolation for the X-coordinates.

Set pixel value at specified coordinates (fx,y,z,c) in the image instance, in a way that the value is spread amongst several neighbors if the pixel coordinates are float-valued.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Pixel value to set.</td>
</tr>
<tr>
<td>fx</td>
<td>X-coordinate of the pixel value (float-valued).</td>
</tr>
<tr>
<td>y</td>
<td>Y-coordinate of the pixel value.</td>
</tr>
<tr>
<td>z</td>
<td>Z-coordinate of the pixel value.</td>
</tr>
<tr>
<td>c</td>
<td>C-coordinate of the pixel value.</td>
</tr>
<tr>
<td>is_added</td>
<td>Tells if the pixel value is added to (true), or simply replace (false) the current image pixel(s).</td>
</tr>
</tbody>
</table>
Returns

A reference to the current image instance.

Note

• Calling this method with out-of-bounds coordinates does nothing.

8.1.4.147 set_linear_atXY()

```cpp
CImg<T>& set_linear_atXY(
    const T & value,
    const float fx,
    const float fy = 0,
    const int z = 0,
    const int c = 0,
    const bool is_added = false)
```

Set pixel value, using linear interpolation for the X and Y-coordinates.

Similar to `set_linear_atX(const T&, float, int, int, int, bool)`, except that the linear interpolation is achieved both for X and Y-coordinates.

8.1.4.148 set_linear_atXYZ()

```cpp
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.

8.1.4.149 value_string()

```cpp
CImg<charT> value_string(
    const char separator = ',',
    const unsigned int max_size = 0,
    const char *const format = 0) const
```

Return a C-string containing a list of all values of the image instance.

Return a new `CImg<`char`>` image whose buffer `data()` is a `char*` string describing the list of all pixel values of the image instance (written in base 10), separated by specified `separator` character.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>separator</td>
<td>A char character which specifies the separator between values in the returned C-string.</td>
</tr>
<tr>
<td>max_size</td>
<td>Maximum size of the returned image (or 0 if no limits are set).</td>
</tr>
<tr>
<td>format</td>
<td>For float/double-values, tell the printf format used to generate the Ascii representation of the numbers (or 0 for default representation).</td>
</tr>
</tbody>
</table>

Note

- The returned image is never empty.
- For an empty image instance, the returned string is "".
- If max_size is equal to 0, there are no limits on the size of the returned string.
- Otherwise, if the maximum number of string characters is exceeded, the value string is cut off and terminated by character '\0'. In that case, the returned image size is max_size + 1.

8.1.4.150 is_shared()

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).

8.1.4.151 is_empty()

bool is_empty() const

Test if image instance is empty.

Return true, if image instance is empty, i.e. does not contain any pixel values, has dimensions 0 x 0 x 0 x 0 and a pixel buffer pointer set to 0 (null pointer), and false otherwise.

8.1.4.152 is_inf()

bool is_inf() const

Test if image instance contains a 'inf' value.

Return true, if image instance contains a 'inf' value, and false otherwise.
8.1.4.153  is_nan()

```cpp
bool is_nan() const
```

Test if image instance contains a NaN value.

Return `true`, if image instance contains a NaN value, and `false` otherwise.

8.1.4.154  is_sameXY(1/3)

```cpp
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.

8.1.4.155  is_sameXY(2/3)

```cpp
bool is_sameXY(const Clmg<t> & img) const
```

Test if image width and height are the same as that of another image.

Test if `is_sameX(const Clmg<t> &) const` and `is_sameY(const Clmg<t> &) const` are both verified.

8.1.4.156  is_sameXY(3/3)

```cpp
bool is_sameXY(const ClmgDisplay & disp) const
```

Test if image width and height are the same as that of an existing display window.

Test if `is_sameX(const ClmgDisplay&) const` and `is_sameY(const ClmgDisplay&) const` are both verified.

8.1.4.157  is_sameXZ(1/2)

```cpp
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.
8.1.4.158 is_sameXZ() [2/2]

```cpp
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.

8.1.4.159 is_sameXC() [1/2]

```cpp
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.

8.1.4.160 is_sameXC() [2/2]

```cpp
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.

8.1.4.161 is_sameYZ() [1/2]

```cpp
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.

8.1.4.162 is_sameYZ() [2/2]

```cpp
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 CImg< T > Struct Template Reference 185

8.1.4.163 is_sameYC() [1/2]

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.

8.1.4.164 is_sameYC() [2/2]

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.

8.1.4.165 is_sameZC() [1/2]

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.

8.1.4.166 is_sameZC() [2/2]

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.

8.1.4.167 is_sameXYZ() [1/2]

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.
is_sameXYZ() [2/2]

```cpp
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.

is_sameXYC() [1/2]

```cpp
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.

is_sameXYC() [2/2]

```cpp
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.

is_sameXZC() [1/2]

```cpp
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.

is_sameXZC() [2/2]

```cpp
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.
8.1.4.173 is_sameYZC() [1/2]

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_sameZY(unsigned int,unsigned int) const and is_sameC(unsigned int) const are both verified.

8.1.4.174 is_sameYZC() [2/2]

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.

8.1.4.175 is_sameXYZC() [1/2]

bool is_sameXYZC (  
    const unsigned int size_x,  
    const unsigned int size_y,  
    const unsigned int size_z,  
    const unsigned int size_c ) const

Test if image width, height, depth and spectrum are equal to specified values.
Test if is_sameXYZ(unsigned int,unsigned int,unsigned int) const and is_sameC(unsigned int) const are both verified.

8.1.4.176 is_sameXYZC() [2/2]

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.

8.1.4.177 containsXYZC()

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 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 \leq x \leq \text{width}() - 1.
  - 0 \leq y \leq \text{height}() - 1.
  - 0 \leq z \leq \text{depth}() - 1.
  - 0 \leq c \leq \text{spectrum}() - 1.

8.1.4.178 contains()[3/5]

```cpp
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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 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:

```cpp
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 \%lu refers to pixel located at (%u,%u,%u,%u).\n",  
    offset,x,y,z,c);
}
```
8.1.4.179  contains() [2/5]

bool contains (  
    const T & pixel,  
    t & x,  
    t & y,  
    t & z ) const

Test if pixel value is inside image bounds and get its X,Y and Z-coordinates.

Similar to contains(const T&,t&,t&,t&) const, except that only the X,Y and Z-coordinates are set.

8.1.4.180  contains() [3/5]

bool contains (  
    const T & pixel,  
    t & x,  
    t & y ) const

Test if pixel value is inside image bounds and get its X and Y-coordinates.

Similar to contains(const T&,t&,t&,t&) const, except that only the X and Y-coordinates are set.

8.1.4.181  contains() [4/5]

bool contains (  
    const T & pixel,  
    t & x ) const

Test if pixel value is inside image bounds and get its X-coordinate.

Similar to contains(const T&,t&,t&,t&) const, except that only the X-coordinate is set.

8.1.4.182  contains() [5/5]

bool contains (  
    const T & pixel ) const

Test if pixel value is inside image bounds.

Similar to contains(const T&,t&,t&,t&) const, except that no pixel coordinates are set.

8.1.4.183  is_overlapped()

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

\[ \text{img} \quad \text{Input image to compare with.} \]

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

```cpp
const CImg<float> img1("reference.jpg"), // Load RGB-color image
    img2 = img1.get_shared_channel(1); // Get shared version of the green channel
if (img1.is_overlapped(img2)) { // Test succeeds, 'img1' and 'img2' overlaps
    std::printf("Buffers overlap!\n");
}
```

8.1.4.184 is_object3d()

```cpp
bool is_object3d (const CImgList<tp> & primitives,
                 const CImgList<tc> & colors,
                 const to & opacities,
                 const bool full_check = true,
                 char *const error_message = 0) const
```

Test if the set \{*this,primitives,colors,opacities\} defines a valid 3D object.

Return true 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>primitives</td>
<td>List of primitives of the 3D object.</td>
</tr>
<tr>
<td>colors</td>
<td>List of colors of the 3D object.</td>
</tr>
<tr>
<td>opacities</td>
<td>List (or image) of opacities of the 3D object.</td>
</tr>
<tr>
<td>full_check</td>
<td>Tells if full checking of the 3D object must be performed.</td>
</tr>
<tr>
<td>error_message</td>
<td>C-string to contain the error message, if the test does not succeed.</td>
</tr>
</tbody>
</table>

Note

- Set full_checking to false to speed-up the 3D object checking. In this case, only the size of each 3D object component is checked.
- Size of the string error_message should be at least 128-bytes long, to be able to contain the error message.
8.1.4.185 is_CImg3d()

```cpp
bool is_CImg3d (const bool full_check = true,
               char *const error_message = 0) const
```

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

Return true if the image instance represents a valid serialization of a 3D object, and false otherwise.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>full_check</td>
<td>Tells if full checking of the instance must be performed.</td>
</tr>
<tr>
<td>error_message</td>
<td>C-string to contain the error message, if the test does not succeed.</td>
</tr>
</tbody>
</table>

**Note**

- Set `full_check` to false to speed-up the 3D object checking. In this case, only the size of each 3D object component is checked.
- Size of the string `error_message` should be at least 128-bytes long, to be able to contain the error message.

8.1.4.186 sqr()

```cpp
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 \( I^2(x,y,z,c) \).

**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**

```cpp
const CImg<float> img("reference.jpg");
{img, img.get_sqr().normalize(0,255)}.display();
```
8.1.4.187  \texttt{sqrt()}  \\
\texttt{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 $\sqrt{I_{(x,y,z,c)}}$.  \\
\textbf{Note}  \\
• The \textbf{[in-place version]} of this method statically casts the computed values to the pixel type $T$.  \\
• The \textbf{[new-instance version]} returns a \texttt{CImg<float>} image, if the pixel type $T$ is not float-valued.  \\
\textbf{Example}  \\
\begin{verbatim}
const CImg<float> img("reference.jpg");
{img, img.get_sqrt().normalize(0,255)}.display();
\end{verbatim}  \\

8.1.4.188  \texttt{exp()}  \\
\texttt{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 $e^{I_{(x,y,z,c)}}$.  \\
\textbf{Note}  \\
• The \textbf{[in-place version]} of this method statically casts the computed values to the pixel type $T$.  \\
• The \textbf{[new-instance version]} returns a \texttt{CImg<float>} image, if the pixel type $T$ is not float-valued.  \\

8.1.4.189  \texttt{log()}  \\
\texttt{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 $\log_e(I_{(x,y,z,c)})$.  \\
\textbf{Note}  \\
• The \textbf{[in-place version]} of this method statically casts the computed values to the pixel type $T$.  \\
• The \textbf{[new-instance version]} returns a \texttt{CImg<float>} image, if the pixel type $T$ is not float-valued.
8.1.4.190  \texttt{log2()} \\
\texttt{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 \texttt{CImg<float>} image, if the pixel type $T$ is not float-valued.

8.1.4.191  \texttt{log10()} \\
\texttt{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 \texttt{CImg<float>} image, if the pixel type $T$ is not float-valued.

8.1.4.192  \texttt{abs()} \\
\texttt{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 \texttt{CImg<float>} image, if the pixel type $T$ is not float-valued.
8.1.4.193  sign()

CImg<
T
>
& sign ( )

Compute the sign of each pixel value.

Replace each pixel value \( I_{(x,y,z,c)} \) of the image instance by its sign \( \text{sign}(I_{(x,y,z,c)}) \).

Note

- The sign is set to:
  - 1 if pixel value is strictly positive.
  - -1 if pixel value is strictly negative.
  - 0 if pixel value is equal to 0.
- The [in-place version] of this 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.

8.1.4.194  cos()

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.

8.1.4.195  sin()

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.
8.1.4.196 \texttt{sinc()}

\texttt{CImg<T>\& \texttt{sinc ( )}}

Compute the \texttt{sinc} of each pixel value.

Replace each pixel value \(I_{(x,y,z,c)}\) of the image instance by its \(\text{sinc}(I_{(x,y,z,c)})\).

\textbf{Note}

- Pixel values are regarded as being \textit{ex in radian}.
- The [\texttt{in-place version}] of this method statically casts the computed values to the pixel type \(T\).
- The [\texttt{new-instance version}] returns a \texttt{CImg<float>} image, if the pixel type \(T\) is \textit{not} float-valued.

8.1.4.197 \texttt{tan()}

\texttt{CImg<T>\& \texttt{tan ( )}}

Compute the \texttt{tangent} of each pixel value.

Replace each pixel value \(I_{(x,y,z,c)}\) of the image instance by its \(\text{tangent}(I_{(x,y,z,c)})\).

\textbf{Note}

- Pixel values are regarded as being \textit{ex in radian}.
- The [\texttt{in-place version}] of this method statically casts the computed values to the pixel type \(T\).
- The [\texttt{new-instance version}] returns a \texttt{CImg<float>} image, if the pixel type \(T\) is \textit{not} float-valued.

8.1.4.198 \texttt{cosh()}

\texttt{CImg<T>\& \texttt{cosh ( )}}

Compute the \texttt{hyperbolic cosine} of each pixel value.

Replace each pixel value \(I_{(x,y,z,c)}\) of the image instance by its \(\text{hyperbolic cosine \texttt{cosh}}(I_{(x,y,z,c)})\).

\textbf{Note}

- The [\texttt{in-place version}] of this method statically casts the computed values to the pixel type \(T\).
- The [\texttt{new-instance version}] returns a \texttt{CImg<float>} image, if the pixel type \(T\) is \textit{not} float-valued.
8.1.4.199 sinh()

CImg<\text{T}>4 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 \textbf{[in-place version]} of this method statically casts the computed values to the pixel type \( \text{T} \).
- The \textbf{[new-instance version]} returns a CImg<float> image, if the pixel type \( \text{T} \) is \textit{not} float-valued.

8.1.4.200 tanh()

CImg<\text{T}>4 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 \textbf{[in-place version]} of this method statically casts the computed values to the pixel type \( \text{T} \).
- The \textbf{[new-instance version]} returns a CImg<float> image, if the pixel type \( \text{T} \) is \textit{not} float-valued.

8.1.4.201 acos()

CImg<\text{T}>4 acos ( )

Compute the arccosine of each pixel value.

Replace each pixel value \( I_{(x,y,z,c)} \) of the image instance by its arccosine \( \text{acos}(I_{(x,y,z,c)}) \).

Note

- The \textbf{[in-place version]} of this method statically casts the computed values to the pixel type \( \text{T} \).
- The \textbf{[new-instance version]} returns a CImg<float> image, if the pixel type \( \text{T} \) is \textit{not} float-valued.
8.1.4.202 \texttt{asin()}

\texttt{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 \(\text{asin}(I(x,y,z,c))\).

\textbf{Note}

- The [in-place version] of this method statically casts the computed values to the pixel type \(T\).
- The [new-instance version] returns a \texttt{CImg<float>} image, if the pixel type \(T\) is \textit{not} float-valued.

8.1.4.203 \texttt{atan()}

\texttt{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 \(\text{atan}(I(x,y,z,c))\).

\textbf{Note}

- The [in-place version] of this method statically casts the computed values to the pixel type \(T\).
- The [new-instance version] returns a \texttt{CImg<float>} image, if the pixel type \(T\) is \textit{not} float-valued.

8.1.4.204 \texttt{atan2()}

\texttt{CImg<T>& atan2 ( const CImg< t >& img )}

Compute the arctangent2 of each pixel value.

Replace each pixel value \(I(x,y,z,c)\) of the image instance by its arctangent2 \(\text{atan2}(I(x,y,z,c))\).

\textbf{Parameters}

\begin{verbatim}
img  Image whose pixel values specify the second argument of the \texttt{atan2()} function.
\end{verbatim}

\textbf{Note}

- The [in-place version] of this method statically casts the computed values to the pixel type \(T\).
- The [new-instance version] returns a \texttt{CImg<float>} image, if the pixel type \(T\) is \textit{not} float-valued.
Example

```cpp
const CImg<float>
    img_x(100,100,1,1,"x-w/2",false), // Define an horizontal centered gradient, from '-width/2' to 'width/2'
    img_y(100,100,1,1,"y-h/2",false), // Define a vertical centered gradient, from '-height/2' to 'height/2'
    img_atan2 = img_y.get_atan2(img_x); // Compute atan2(y,x) for each pixel value
{img_x,img_y,img_atan2}.display();
```

8.1.4.205 `acosh()`

CImg<T> & acosh()

Compute the hyperbolic arccosine of each pixel value.
Replace each pixel value \( I_{(x,y,z,c)} \) of the image instance by its arccosine \( \text{acosh}(I_{(x,y,z,c)}) \).

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.

8.1.4.206 `asinh()`

CImg<T> & asinh()

Compute the hyperbolic arcsine of each pixel value.
Replace each pixel value \( I_{(x,y,z,c)} \) of the image instance by its hyperbolic arcsine \( \text{asinh}(I_{(x,y,z,c)}) \).

Note:

- The [in-place version] of this 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.

8.1.4.207 `atanh()`

CImg<T> & atanh()

Compute the hyperbolic arctangent of each pixel value.
Replace each pixel value \( I_{(x,y,z,c)} \) of the image instance by its hyperbolic arctangent \( \text{atanh}(I_{(x,y,z,c)}) \).

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.

8.1.4.208 `mul()`

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

\[ \text{img} \] Input image, as the second operand of the multiplication.

Note

- Similar to operator+=(const CImg& img), 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&) instead.

Example

```cpp
CImg<float> img("reference.jpg");
shade(img.width, img.height(), 1, 1, "-(x-w/2)^2-(y-h/2)^2", false);
shade.normalize(0, 1);
(img, shade, img.get_mul(shade)).display();
```

8.1.4.209 div()

\[ \text{CImg<}T\text{>} & \text{div (const CImg<}T\text{>} & img) } \]

In-place pointwise division.

Similar to mul(const CImg& img), except that it performs a pointwise division instead of a multiplication.

8.1.4.210 pow() [1/3]

\[ \text{CImg<}T\text{>} & \text{pow (const double p) } \]

Raise each pixel value to a specified power.

Replace each pixel value \( I(x,y,z,c) \) of the image instance by its power \( I^p(x,y,z,c) \).

Parameters

\[ p \] Exponent value.

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

```cpp
const CImg<float> img0("reference.jpg"); // Load reference color image
```
200 Class Documentation

```c
img1 = (img0/255).pow(1.8)*=255; // Compute gamma correction, with gamma = 1.8
img2 = (img0/255).pow(0.5)*=255; // Compute gamma correction, with gamma = 0.5
{img0,img1,img2}.display();
```

8.1.4.211 pow() [2/3]

```c
CImg<T>& pow (  
  const char *const expression )
```

Raise each pixel value to a power, specified from an expression.
Similar to operator+=(const char*), except it performs a pointwise exponentiation instead of an addition.

8.1.4.212 pow() [3/3]

```c
CImg<T>& pow (  
  const CImg<T>& img )
```

Raise each pixel value to a power, pointwisely specified from another image.
Similar to operator+=(const CImg<T>& img), except that it performs an exponentiation instead of an addition.

8.1.4.213 rol() [1/3]

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

Compute the bitwise left rotation of each pixel value.
Similar to operator<<(unsigned int), except that it performs a left rotation instead of a left shift.

8.1.4.214 rol() [2/3]

```c
CImg<T>& rol (  
  const char *const expression )
```

Compute the bitwise left rotation of each pixel value.
Similar to operator<<(const char*), except that it performs a left rotation instead of a left shift.

8.1.4.215 rol() [3/3]

```c
CImg<T>& rol (  
  const CImg<T>& img )
```

Compute the bitwise left rotation of each pixel value.
Similar to operator<<(const CImg<T>&), except that it performs a left rotation instead of a left shift.
8.1.4.216 \texttt{ror()} [1/3]

\begin{verbatim}
CImg\langle T \rangle \& \texttt{ror} (
    \text{const unsigned int} \ n = 1
)
\end{verbatim}

Compute the bitwise right rotation of each pixel value.

Similar to operator \texttt{\textgreater\textgreater}=(\text{unsigned int}), except that it performs a right rotation instead of a right shift.

8.1.4.217 \texttt{ror()} [2/3]

\begin{verbatim}
CImg\langle T \rangle \& \texttt{ror} (
    \text{const char} \* \text{const} \ expression
)
\end{verbatim}

Compute the bitwise right rotation of each pixel value.

Similar to \texttt{operator\textgreater\textgreater}=(\text{const char*}), except that it performs a right rotation instead of a right shift.

8.1.4.218 \texttt{ror()} [3/3]

\begin{verbatim}
CImg\langle T \rangle \& \texttt{ror} (
    \text{const CImg\langle t \rangle \&} \ img\n)
\end{verbatim}

Compute the bitwise right rotation of each pixel value.

Similar to \texttt{operator\textgreater\textgreater}=(\text{const CImg\langle t \rangle \&}), except that it performs a right rotation instead of a right shift.

8.1.4.219 \texttt{min()} [1/3]

\begin{verbatim}
CImg\langle T \rangle \& \texttt{min} (
    \text{const T} \& \ \text{value}\n)
\end{verbatim}

Pointwise min operator between instance image and a value.

Parameters

\begin{tabular}{|c|c|}
\hline
\texttt{val} & Value used as the reference argument of the min operator. \\
\hline
\end{tabular}

Note

Replace each pixel value $I(x,y,z,c)$ of the image instance by $\text{min}(I(x,y,z,c), \text{val})$.

8.1.4.220 \texttt{min()} [2/3]

\begin{verbatim}
CImg\langle T \rangle \& \texttt{min} (
    \text{const CImg\langle t \rangle \&} \ img\n)
\end{verbatim}

Pointwise min operator between two images.
Parameters

| **img** | Image used as the reference argument of the min operator. |

Note

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by $\min(I_{(x,y,z,c)}, \text{img}_{(x,y,z,c)})$.

8.1.4.221 **min()** [3/3]

```cpp
cimg<T>& min (  
    const char *const expression  )
```

Pointwise min operator between an image and an expression.

Parameters

| **expression** | Math formula as a C-string. |

Note

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by $\min(I_{(x,y,z,c)}, \text{expr}_{(x,y,z,c)})$.

8.1.4.222 **max()** [1/3]

```cpp
cimg<T>& max (  
    const T & value  )
```

Pointwise max operator between instance image and a value.

Parameters

| **val** | Value used as the reference argument of the max operator. |

Note

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by $\max(I_{(x,y,z,c)}, \text{val})$.

8.1.4.223 **max()** [2/3]

```cpp
cimg<T>& max (  
    const cimg< t > & img  )
```
Pointwise max operator between two images.
Parameters

\[ \text{img} \] Image used as the reference argument of the max operator.

Note

Replace each pixel value \( I_{(x,y,z,c)} \) of the image instance by \( \max(I_{(x,y,z,c)}, \text{img}_{(x,y,z,c)}) \).

8.1.4.224 max()

```cpp
CImg<\text{T},1> & max ( const char *const expression )
```

Pointwise max operator between an image and an expression.

Parameters

\[ \text{expression} \] Math formula as a C-string.

Note

Replace each pixel value \( I_{(x,y,z,c)} \) of the image instance by \( \max(I_{(x,y,z,c)}, \text{expr}_{(x,y,z,c)}) \).

8.1.4.225 min_max()

```cpp
\text{T} & min_max ( t & \text{max_val} )
```

Return a reference to the minimum pixel value as well as the maximum pixel value.

Parameters

\[ \text{out} \quad \text{max_val} \] Maximum pixel value.

8.1.4.226 max_min()

```cpp
\text{T} & max_min ( t & \text{min_val} )
```

Return a reference to the maximum pixel value as well as the minimum pixel value.
Parameters

| out | min_val | Minimum pixel value |

8.1.4.227  kth_smallest()

\[ T \text{ kth_smallest} (\text{const ulongT k}) \text{ const} \]

Return the kth smallest pixel value.

Parameters

| k | Rank of the search smallest element |

8.1.4.228  variance()

double variance (\text{const unsigned int variance_method = 1}) \text{ const}

Return the variance of the pixel values.

Parameters

<table>
<thead>
<tr>
<th>variance_method</th>
<th>Method used to estimate the variance. Can be:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Second moment, computed as [ 1/N \sum_{k=1}^{N} (x_k - \bar{x})^2 = 1/N \left( \sum_{k=1}^{N} x_k^2 - \left( \sum_{k=1}^{N} x_k \right)^2 / N \right) \text{ with } \bar{x} = 1/N \sum_{k=1}^{N} x_k. ]</td>
</tr>
<tr>
<td>1</td>
<td>Best unbiased estimator, computed as [ \frac{1}{N-1} \sum_{k=1}^{N} (x_k - \bar{x})^2. ]</td>
</tr>
<tr>
<td>2</td>
<td>Least median of squares.</td>
</tr>
<tr>
<td>3</td>
<td>Least trimmed of squares.</td>
</tr>
</tbody>
</table>

8.1.4.229  variance_mean()

double variance_mean (\text{const unsigned int variance_method, t & mean}) \text{ const}

Return the variance as well as the average of the pixel values.
Parameters

<table>
<thead>
<tr>
<th></th>
<th>variance_method</th>
<th>Method used to estimate the variance (see variance(const unsigned int) const).</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>mean</td>
<td>Average pixel value.</td>
</tr>
</tbody>
</table>

8.1.4.230 variance_noise()

double variance_noise (const unsigned int variance_method = 2) const

Return estimated variance of the noise.

Parameters

| variance_method | Method used to compute the variance (see variance(const unsigned int) const). |

Note

Because of structures such as edges in images it is recommended to use 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 $\sigma$ is the noise variance.

8.1.4.231 MSE()

double MSE (const CImg< t >& img ) const

Compute the MSE (Mean-Squared Error) between two images.

Parameters

| img | Image used as the second argument of the MSE operator. |

8.1.4.232 PSNR()

double PSNR (const CImg< t >& img, const double max_value = 255 ) const

Compute the PSNR (Peak Signal-to-Noise Ratio) between two images.
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>img</code></td>
<td>Image used as the second argument of the PSNR operator.</td>
</tr>
<tr>
<td><code>max_value</code></td>
<td>Maximum theoretical value of the signal.</td>
</tr>
</tbody>
</table>

#### 8.1.4.233 `eval()` [1/3]

```c
double eval {
    const char *const expression,
    const double x = 0,
    const double y = 0,
    const double z = 0,
    const double c = 0,
    const CImgList< T > *const list_inputs = 0,
    CImgList< T > *const list_outputs = 0 )
```

Evaluate math formula.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>expression</code></td>
<td>Math formula, as a C-string.</td>
</tr>
<tr>
<td><code>x</code></td>
<td>Value of the pre-defined variable <code>x</code>.</td>
</tr>
<tr>
<td><code>y</code></td>
<td>Value of the pre-defined variable <code>y</code>.</td>
</tr>
<tr>
<td><code>z</code></td>
<td>Value of the pre-defined variable <code>z</code>.</td>
</tr>
<tr>
<td><code>c</code></td>
<td>Value of the pre-defined variable <code>c</code>.</td>
</tr>
<tr>
<td><code>list_inputs</code></td>
<td>A list of input images attached to the specified math formula.</td>
</tr>
<tr>
<td><code>list_outputs</code></td>
<td>A pointer to a list of output images attached to the specified math formula.</td>
</tr>
</tbody>
</table>

#### 8.1.4.234 `eval()` [2/3]

```c
void eval ( 
    CImg< T >::output,
    const char *const expression,
    const double x = 0,
    const double y = 0,
    const double z = 0,
    const double c = 0,
    const CImgList< T > *const list_inputs = 0,
    CImgList< T > *const list_outputs = 0 )
```

Evaluate math formula.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>output</code></td>
<td>Contains values of output vector returned by the evaluated expression (or is empty if the returned type is scalar).</td>
</tr>
</tbody>
</table>
### Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>Math formula, as a C-string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Value of the pre-defined variable x.</td>
</tr>
<tr>
<td>y</td>
<td>Value of the pre-defined variable y.</td>
</tr>
<tr>
<td>z</td>
<td>Value of the pre-defined variable z.</td>
</tr>
<tr>
<td>c</td>
<td>Value of the pre-defined variable c.</td>
</tr>
<tr>
<td>list_inputs</td>
<td>A list of input images attached to the specified math formula.</td>
</tr>
<tr>
<td>out</td>
<td>list_outputs</td>
</tr>
</tbody>
</table>

### `eval()` [3/3]

```cpp
CImg<doubleT> eval (
    const char *const expression,
    const CImg<doubleT> & xyzc,
    const CImgList<doubleT> *const list_inputs = 0,
    CImgList<doubleT> *const list_outputs = 0 )
```

Evaluate math formula on a set of variables.

### Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>Math formula, as a C-string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>xyzc</td>
<td>Set of values (x,y,z,c) used for the evaluation.</td>
</tr>
<tr>
<td>list_inputs</td>
<td>A list of input images attached to the specified math formula.</td>
</tr>
<tr>
<td>out</td>
<td>list_outputs</td>
</tr>
</tbody>
</table>

### `magnitude()`

```cpp
double magnitude ( 
    const int magnitude_type = 2 ) const
```

Compute norm of the image, viewed as a matrix.

### Parameters

<table>
<thead>
<tr>
<th>magnitude_type</th>
<th>Norm type. Can be:</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Linf-norm</td>
</tr>
<tr>
<td>0</td>
<td>L0-norm</td>
</tr>
<tr>
<td>1</td>
<td>L1-norm</td>
</tr>
<tr>
<td>2</td>
<td>L2-norm</td>
</tr>
</tbody>
</table>
8.1.4.237  dot()

double dot (  
    const CImg< T > & img ) const

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

Parameters

| img | Image used as a second argument of the dot product. |

8.1.4.238  get_vector_at()

CImg< T > get_vector_at (  
    const unsigned int x,  
    const unsigned int y = 0,  
    const unsigned int z = 0 ) const

Get vector-valued pixel located at specified position.

Parameters

| x | X-coordinate of the pixel value. |
| y | Y-coordinate of the pixel value. |
| z | Z-coordinate of the pixel value. |

8.1.4.239  get_matrix_at()

CImg< T > get_matrix_at (  
    const unsigned int x = 0,  
    const unsigned int y = 0,  
    const unsigned int z = 0 ) const

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

Parameters

| x | X-coordinate of the pixel value. |
| y | Y-coordinate of the pixel value. |
| z | Z-coordinate of the pixel value. |
Note
- The `spectrum()` of the image must be a square.

8.1.4.240  `get_tensor_at()`

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

Get tensor-valued pixel located at specified position.

**Parameters**

| x  | X-coordinate of the pixel value. |
| y  | Y-coordinate of the pixel value. |
| z  | Z-coordinate of the pixel value. |

8.1.4.241  `set_vector_at()`

```cpp
CImg<T>& set_vector_at (  
  const CImg< T >& vec,  
  const unsigned int x,  
  const unsigned int y = 0,  
  const unsigned int z = 0 )
```

Set vector-valued pixel at specified position.

**Parameters**

| vec   | Vector to put on the instance image. |
| x     | X-coordinate of the pixel value.     |
| y     | Y-coordinate of the pixel value.     |
| z     | Z-coordinate of the pixel value.     |

8.1.4.242  `set_matrix_at()`

```cpp
CImg<T>& set_matrix_at (  
  const CImg< T >& mat,  
  const unsigned int x = 0,  
  const unsigned int y = 0,  
  const unsigned int z = 0 )
```
Set (square) matrix-valued pixel at specified position.
8.1.4.243 set_tensor_at()

```cpp
CImg< T > & set_tensor_at ( const CImg< T > & ten, const unsigned int x = 0, const unsigned int y = 0, const unsigned int z = 0 )
```

Set tensor-valued pixel at specified position.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ten</code></td>
<td>Tensor to put on the instance image.</td>
</tr>
<tr>
<td><code>x</code></td>
<td>X-coordinate of the pixel value.</td>
</tr>
<tr>
<td><code>y</code></td>
<td>Y-coordinate of the pixel value.</td>
</tr>
<tr>
<td><code>z</code></td>
<td>Z-coordinate of the pixel value.</td>
</tr>
</tbody>
</table>

8.1.4.244 vector() [1/5]

```cpp
CImg< T > & vector ( )
```

Unroll pixel values along axis `y`.

Note

Equivalent to

```
unroll('y');
```

8.1.4.245 diagonal()

```cpp
CImg< T > & diagonal ( )
```

Resize image to become a diagonal matrix.

Note

Transform the image as a diagonal matrix so that each of its initial value becomes a diagonal coefficient.
8.1.4.246 identity_matrix() [1/2]

CImg&lt;T&gt;& identity_matrix ( )

Replace the image by an identity matrix.

Note
If the instance image is not square, it is resized to a square matrix using its maximum dimension as a reference.

8.1.4.247 sequence() [1/2]

CImg&lt;T&gt;& sequence ( const T & a0, const T & a1 )

Fill image with a linear sequence of values.

Parameters

<table>
<thead>
<tr>
<th>a0</th>
<th>Starting value of the sequence.</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>Ending value of the sequence.</td>
</tr>
</tbody>
</table>

8.1.4.248 transpose()

CImg&lt;T&gt;& transpose ( )

Transpose the image, viewed as a matrix.

Note
Equivalent to

```
permute_axes("yxzc");
```

8.1.4.249 cross()

CImg&lt;T&gt;& cross ( const CImg&lt; t & img )

Compute the cross product between two 1x3 images, viewed as 3D vectors.
Parameters

\texttt{img} \quad \text{Image used as the second argument of the cross product.}

\textbf{Note}

The first argument of the cross product is \texttt{*this}.

8.1.4.250 \quad \texttt{invert()}

\begin{verbatim}
CImg<T>& invert {
    const bool use_LU = true
}
\end{verbatim}

Invert the instance image, viewed as a matrix.

\textbf{Parameters}

\texttt{use_LU} \quad \text{Choose the inverting algorithm. Can be:}

\begin{itemize}
    \item \texttt{true}: LU-based matrix inversion.
    \item \texttt{false}: SVD-based matrix inversion.
\end{itemize}

8.1.4.251 \quad \texttt{solve()}

\begin{verbatim}
CImg<T>& solve {
    const CImg< t > & A
}
\end{verbatim}

Solve a system of linear equations.

\textbf{Parameters}

\texttt{A} \quad \text{Matrix of the linear system.}

\textbf{Note}

Solve \(AX = B\) where \(B = \texttt{*this}\).

8.1.4.252 \quad \texttt{solve_tridiagonal()}

\begin{verbatim}
CImg<T>& solve_tridiagonal {
    const CImg< t > & A
}
\end{verbatim}

Solve a tridiagonal system of linear equations.
### Parameters

| A | Coefficients of the tridiagonal system. A is a tridiagonal matrix \( A = [ b_0,c_0,0,\ldots; a_1,b_1,c_1,0,\ldots; \ldots; 0,a_N,b_N ] \), stored as a 3 columns matrix |

### Note

Solve \( AX=B \) where \( B=*this \), using the Thomas algorithm.

#### 8.1.4.253  
**eigen()**

```cpp
const CImg<T>& eigen {
  CImg<T> & val,
  CImg<T> & vec } const
```

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

**Parameters**

| **out**  | **val** | Vector of the estimated eigenvalues, in decreasing order. |
| **out**  | **vec** | Matrix of the estimated eigenvectors, sorted by columns. |

#### 8.1.4.254  
**get_eigen()**

```cpp
CImgList<Tfloat> get_eigen ( ) const
```

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

**Returns**

A list of two images \([val; vec]\), whose meaning is similar as in \( eigen(CImg<t>&,CImg<t>&) \) const.

#### 8.1.4.255  
**symmetric_eigen()**

```cpp
const CImg<T>& symmetric_eigen {
  CImg<T> & val,
  CImg<T> & vec } const
```

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

---

Generated by Doxygen
Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>val</th>
<th>Vector of the estimated eigenvalues, in decreasing order.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>vec</td>
<td>Matrix of the estimated eigenvectors, sorted by columns.</td>
</tr>
</tbody>
</table>

8.1.4.256 get_symmetric_eigen()

CImgList<Tfloat> get_symmetric_eigen () const

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

Returns

A list of two images \([\text{val}; \text{vec}]\), whose meaning are similar as in `symmetric_eigen(CImg<T>&, CImg<T>&) const`. 

8.1.4.257 sort() [1/2]

CImg<T>& sort (CImg<T>& permutations, const bool is_increasing = true)

Sort pixel values and get sorting permutations.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>permutations</th>
<th>Permutation map used for the sorting.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>is_increasing</td>
<td>Tells if pixel values are sorted in an increasing (true) or decreasing (false) way.</td>
</tr>
</tbody>
</table>

8.1.4.258 sort() [2/2]

CImg<T>& sort (const bool is_increasing = true, const char axis = 0)

Sort pixel values.

Parameters

| is_increasing | Tells if pixel values are sorted in an increasing (true) or decreasing (false) way. |
Parameters

<table>
<thead>
<tr>
<th>axis</th>
<th>Tells if the value sorting must be done along a specific axis. Can be:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• 0</td>
<td>All pixel values are sorted, independently on their initial position.</td>
</tr>
<tr>
<td>• 'x'</td>
<td>Image columns are sorted, according to the first value in each column.</td>
</tr>
<tr>
<td>• 'y'</td>
<td>Image rows are sorted, according to the first value in each row.</td>
</tr>
<tr>
<td>• 'z'</td>
<td>Image slices are sorted, according to the first value in each slice.</td>
</tr>
<tr>
<td>• 'c'</td>
<td>Image channels are sorted, according to the first value in each channel.</td>
</tr>
</tbody>
</table>

### 8.1.4.259 SVD()

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

Compute the SVD of the instance image, viewed as a general matrix.

Compute the SVD decomposition \( \text{this} = U \times S \times V' \) where \( U \) and \( V \) are orthogonal matrices and \( S \) is a diagonal matrix. \( V' \) denotes the matrix transpose of \( V \).

#### Parameters

<table>
<thead>
<tr>
<th>out U</th>
<th>First matrix of the SVD product.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out S</td>
<td>Coefficients of the second (diagonal) matrix of the SVD product. These coefficients are stored as a vector.</td>
</tr>
<tr>
<td>out V</td>
<td>Third matrix of the SVD product.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sorting</th>
<th>Tells if the diagonal coefficients are sorted (in decreasing order).</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_iteration</td>
<td>Maximum number of iterations considered for the algorithm convergence.</td>
</tr>
<tr>
<td>lambda</td>
<td>Epsilon used for the algorithm convergence.</td>
</tr>
</tbody>
</table>

#### Note

The instance matrix can be computed from \( U, S \) and \( V \) by

```cpp
const CImg< A;  // Input matrix (assumed to contain some values)
CImg< U,S,V;  
A.SVD(U,S,V)
```

### 8.1.4.260 get_SVD()

```cpp
CImgList<Tfloat> get_SVD {
    const bool sorting = true,
```
const unsigned int max_iteration = 40,
const float lambda = 0 ) const

Compute the SVD of the instance image, viewed as a general matrix.

Returns
A list of three images \([U; S; V]\), whose meaning is similar as in \(\text{SVD}(\text{CImg}<t>&,\text{CImg}<t>&,\text{CImg}<t>&,\text{bool},\text{unsigned int},\text{float})\) const.

8.1.4.261 dijkstra() [1/2]

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_node } [static]

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

Parameters

<table>
<thead>
<tr>
<th>distance</th>
<th>An object having (\text{operator()}(\text{unsigned int} i, \text{unsigned int} j)) which returns distance between two nodes ((i,j)).</th>
</tr>
</thead>
<tbody>
<tr>
<td>nb_nodes</td>
<td>Number of graph nodes.</td>
</tr>
<tr>
<td>starting_node</td>
<td>Index of the starting node.</td>
</tr>
<tr>
<td>ending_node</td>
<td>Index of the ending node (set to ~0U to ignore ending node).</td>
</tr>
<tr>
<td>previous_node</td>
<td>Array that gives the previous node index in the path to the starting node (optional parameter).</td>
</tr>
</tbody>
</table>

Returns
Array of distances of each node to the starting node.

8.1.4.262 dijkstra() [2/2]

CImg<T>& dijkstra {
const unsigned int starting_node,
const unsigned int ending_node,
CImg< t > & previous_node }

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

Parameters

<table>
<thead>
<tr>
<th>starting_node</th>
<th>Index of the starting node.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ending_node</td>
<td>Index of the ending node.</td>
</tr>
<tr>
<td>previous_node</td>
<td>Array that gives the previous node index in the path to the starting node (optional parameter).</td>
</tr>
</tbody>
</table>
Returns

Array of distances of each node to the starting node.

Note

image instance corresponds to the adjacency matrix of the graph.

---

8.1.4.263 string()

```cpp
static CImg<T> string (const char *const str, const bool is_last_zero = true, const bool is_shared = false) [static]
```

Return an image containing the Ascii codes of the specified string.

**Parameters**

<table>
<thead>
<tr>
<th>str</th>
<th>input C-string to encode as an image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_last_zero</td>
<td>Tells if the ending '0' character appear in the resulting image.</td>
</tr>
<tr>
<td>is_shared</td>
<td>Return result that shares its buffer with str.</td>
</tr>
</tbody>
</table>

---

8.1.4.264 vector() [2/5]

```cpp
static CImg<T> vector (const T & a0) [static]
```

Return a 1x1 image containing specified value.

**Parameters**

| a0     | First vector value. |

---

8.1.4.265 vector() [3/5]

```cpp
static CImg<T> vector (const T & a0, const T & a1) [static]
```

Return a 1x2 image containing specified values.
Parameters

<table>
<thead>
<tr>
<th></th>
<th>First vector value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td></td>
</tr>
<tr>
<td>a1</td>
<td>Second vector value.</td>
</tr>
</tbody>
</table>

8.1.4.266  vector() [4/5]

```cpp
static CImg<T> vector(
    const T & a0,
    const T & a1,
    const T & a2 ) [static]
```

Return a 1x3 image containing specified values.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>First vector value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td></td>
</tr>
<tr>
<td>a1</td>
<td>Second vector value.</td>
</tr>
<tr>
<td>a2</td>
<td>Third vector value.</td>
</tr>
</tbody>
</table>

8.1.4.267  vector() [5/5]

```cpp
static CImg<T> vector(
    const T & a0,
    const T & a1,
    const T & a2,
    const T & a3 ) [static]
```

Return a 1x4 image containing specified values.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>First vector value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td></td>
</tr>
<tr>
<td>a1</td>
<td>Second vector value.</td>
</tr>
<tr>
<td>a2</td>
<td>Third vector value.</td>
</tr>
<tr>
<td>a3</td>
<td>Fourth vector value.</td>
</tr>
</tbody>
</table>

8.1.4.268  matrix() [1/3]

```cpp
static CImg<T> matrix(
    const T & a0 ) [static]
```

Return a 1x1 matrix containing specified coefficients.
Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>a0</strong></td>
<td>First matrix value.</td>
</tr>
</tbody>
</table>

Note

Equivalent to `vector(const T&)`.

8.1.4.269 matrix() [2/3]

```cpp
static CImg<T> matrix (
    const T & a0,
    const T & a1,
    const T & a2,
    const T & a3 ) [static]
```

Return a 2x2 matrix containing specified coefficients.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>a0</strong></td>
<td>First matrix value.</td>
</tr>
<tr>
<td><strong>a1</strong></td>
<td>Second matrix value.</td>
</tr>
<tr>
<td><strong>a2</strong></td>
<td>Third matrix value.</td>
</tr>
<tr>
<td><strong>a3</strong></td>
<td>Fourth matrix value.</td>
</tr>
</tbody>
</table>

8.1.4.270 matrix() [3/3]

```cpp
static CImg<T> matrix (
    const T & a0,
    const T & a1,
    const T & a2,
    const T & a3,
    const T & a4,
    const T & a5,
    const T & a6,
    const T & a7,
    const T & a8 ) [static]
```

Return a 3x3 matrix containing specified coefficients.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>a0</strong></td>
<td>First matrix value.</td>
</tr>
<tr>
<td><strong>a1</strong></td>
<td>Second matrix value.</td>
</tr>
<tr>
<td><strong>a2</strong></td>
<td>Third matrix value.</td>
</tr>
<tr>
<td><strong>a3</strong></td>
<td>Fourth matrix value.</td>
</tr>
<tr>
<td><strong>a4</strong></td>
<td>Fifth matrix value.</td>
</tr>
</tbody>
</table>
8.1.4.271 tensor()

```cpp
class CImg<T> { 
    static CImg<T> tensor ( 
        const T & a0 ) [static]
```

Return a 1x1 symmetric matrix containing specified coefficients.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>a0</strong></td>
<td>First matrix value.</td>
</tr>
</tbody>
</table>

Note

Equivalent to `vector(const T&)`.

8.1.4.272 identity_matrix() [2/2]

```cpp
class CImg<T> { 
    static CImg<T> identity_matrix ( 
        const unsigned int N ) [static]
```

Return a NxN identity matrix.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>N</strong></td>
<td>Dimension of the matrix.</td>
</tr>
</tbody>
</table>

8.1.4.273 sequence() [2/2]

```cpp
class CImg<T> { 
    static CImg<T> sequence ( 
        const unsigned int N, 
        const T & a0, 
        const T & a1 ) [static]
```

Return a N-numbered sequence vector from `a0` to `a1`. 
8.1 CImg< T > Struct Template Reference

Parameters

<table>
<thead>
<tr>
<th>N</th>
<th>Size of the resulting vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td>Starting value of the sequence.</td>
</tr>
<tr>
<td>a1</td>
<td>Ending value of the sequence.</td>
</tr>
</tbody>
</table>

8.1.4.274 rotation_matrix()

```cpp
static CImg< T > rotation_matrix ( const float x, const float y, const float z, const float w, const bool is_quaternion = false ) [static]
```

Return a 3x3 rotation matrix from an ( axis + angle ) or a quaternion.

Parameters

| x | X-coordinate of the rotation axis, or first quaternion coordinate. |
| y | Y-coordinate of the rotation axis, or second quaternion coordinate. |
| z | Z-coordinate of the rotation axis, or third quaternion coordinate. |
| w | Angle of the rotation axis (in degree), or fourth quaternion coordinate. |
| is_quaternion | Tell is the four arguments denotes a set ( axis + angle ) or a quaternion (x,y,z,w). |

8.1.4.275 fill() [1/4]

```cpp
CImg< T > & fill ( const T & val )
```

Fill all pixel values with specified value.

Parameters

| val | Fill value. |

8.1.4.276 fill() [2/4]

```cpp
CImg< T > & fill ( const T & val0, const T & val1 )
```

Fill sequentially all pixel values with specified values.
Parameters

<table>
<thead>
<tr>
<th>val0</th>
<th>First fill value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>val1</td>
<td>Second fill value.</td>
</tr>
</tbody>
</table>

### 8.1.4.277 fill()

```cpp
CImg<T>& fill (const char *const expression, const bool repeat_values, const bool allow_formula = true, const CImgList<T> *const list_inputs = 0, CImgList<T> *const list_outputs = 0)
```

Fill sequentially pixel values according to a given expression.

Parameters

| expression | C-string describing a math formula, or a sequence of values. |
| repeat_values | In case a list of values is provided, tells if this list must be repeated for the filling. |
| allow_formula | Tells that mathematical formulas are authorized for the filling. |
| list_inputs | In case of a mathematical expression, attach a list of images to the specified expression. |
| list_outputs | In case of a math expression, list of images atatched to the specified expression. |

### 8.1.4.278 fill()

```cpp
CImg<T>& fill (const CImg<t>& values, const bool repeat_values = true)
```

Fill sequentially pixel values according to the values found in another image.

Parameters

| values | Image containing the values used for the filling. |
| repeat_values | In case there are less values than necessary in values, tells if these values must be repeated for the filling. |

### 8.1.4.279 fillX()

```cpp
CImg<T>& fillX (const unsigned int y, const unsigned int x, const unsigned int width, const unsigned int height)
```

```cpp
CImg<T>& fillX (const CImg<T>& values, const unsigned int y, const unsigned int x, const unsigned int width, const unsigned int height)
```
const unsigned int z,  
const unsigned int c,  
const int a0,  
... )

Fill pixel values along the X-axis at a specified pixel position.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>Y-coordinate of the filled column.</td>
</tr>
<tr>
<td>z</td>
<td>Z-coordinate of the filled column.</td>
</tr>
<tr>
<td>c</td>
<td>C-coordinate of the filled column.</td>
</tr>
<tr>
<td>a0</td>
<td>First fill value.</td>
</tr>
</tbody>
</table>

8.1.4.280 fillY()

CImg<T>& fillY (  
    const unsigned int x,  
    const unsigned int z,  
    const unsigned int c,  
    const int a0,  
    ... )

Fill pixel values along the Y-axis at a specified pixel position.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>X-coordinate of the filled row.</td>
</tr>
<tr>
<td>z</td>
<td>Z-coordinate of the filled row.</td>
</tr>
<tr>
<td>c</td>
<td>C-coordinate of the filled row.</td>
</tr>
<tr>
<td>a0</td>
<td>First fill value.</td>
</tr>
</tbody>
</table>

8.1.4.281 fillZ()

CImg<T>& fillZ (  
    const unsigned int x,  
    const unsigned int y,  
    const unsigned int c,  
    const int a0,  
    ... )

Fill pixel values along the Z-axis at a specified pixel position.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>X-coordinate of the filled slice.</td>
</tr>
<tr>
<td>y</td>
<td>Y-coordinate of the filled slice.</td>
</tr>
<tr>
<td>c</td>
<td>C-coordinate of the filled slice.</td>
</tr>
<tr>
<td>a0</td>
<td>First fill value.</td>
</tr>
</tbody>
</table>
8.1.4.282 fillC()

CImg<T>& fillC {
    const unsigned int x,
    const unsigned int y,
    const unsigned int z,
    const int a0,
    ...
}

Fill pixel values along the C-axis at a specified pixel position.

Parameters

| x  | X-coordinate of the filled channel. |
| y  | Y-coordinate of the filled channel. |
| z  | Z-coordinate of the filled channel. |
| a0 | First filling value.               |

8.1.4.283 discard()

CImg<T>& discard {
    const CImg< t > & values,
    const char axis = 0
}

Discard specified sequence of values in the image buffer, along a specific axis.

Parameters

| values | Sequence of values to discard. |
| axis   | Axis along which the values are discarded. If set to 0 (default value) the method does it for all the buffer values and returns a one-column vector. |

Note

Discarded values will change the image geometry, so the resulting image is returned as a one-column vector.

8.1.4.284 rand()

CImg<T>& rand {
    const T & val_min,
    const T & val_max
}

Fill image with random values in specified range.
8.1 CImg< T > Struct Template Reference

Parameters

<table>
<thead>
<tr>
<th>val_min</th>
<th>Minimal authorized random value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>val_max</td>
<td>Maximal authorized random value.</td>
</tr>
</tbody>
</table>

Note

Random variables are uniformly distributed in [val_min, val_max].

8.1.4.285 round()

CImg< T > & round {
    const double y = 1,
    const int rounding_type = 0
}

Round pixel values.

Parameters

<table>
<thead>
<tr>
<th>y</th>
<th>Rounding precision.</th>
</tr>
</thead>
<tbody>
<tr>
<td>rounding_type</td>
<td>Rounding type. Can be:</td>
</tr>
<tr>
<td></td>
<td>• -1: Backward.</td>
</tr>
<tr>
<td></td>
<td>• 0: Nearest.</td>
</tr>
<tr>
<td></td>
<td>• 1: Forward.</td>
</tr>
</tbody>
</table>

8.1.4.286 noise()

CImg< T > & noise {
    const double sigma,
    const unsigned int noise_type = 0
}

Add random noise to pixel values.

Parameters

<table>
<thead>
<tr>
<th>sigma</th>
<th>Amplitude of the random additive noise. If sigma&lt;0, it stands for a percentage of the global value range.</th>
</tr>
</thead>
<tbody>
<tr>
<td>noise_type</td>
<td>Type of additive noise (can be 0=gaussian, 1=uniform, 2=Salt and Pepper, 3=Poisson or 4=Rician).</td>
</tr>
</tbody>
</table>
Returns

A reference to the modified image instance.

Note

- For Poisson noise (\texttt{noise\_type=3}), parameter \texttt{sigma} is ignored, as Poisson noise only depends on the image value itself.
- Function \texttt{CImg\textless T\textgreater::get\_noise()} is also defined. It returns a non-shared modified copy of the image instance.

Example

```cpp
class CImgTypedebug commented (
    const CImg<
        int, int> &
    )
    return (true);
8.1.4.289 norm()

\texttt{CImg< T > \& norm (}
\texttt{const int norm\_type = 2 )}

Compute Lp-norm of each multi-valued pixel of the image instance.
Parameters

| norm_type | Type of computed vector norm (can be \(-1=\text{Linf}\), or greater or equal than 0). |

Example

```cpp
const CImg<float> img("reference.jpg"); res = img.get_norm();
{img, res.normalize(0, 255)}.display();
```

8.1.4.290 cut()

```cpp
CImg<T>& cut {
    const T & min_value,
    const T & max_value
}
```

Cut pixel values in specified range.

Parameters

| min_value | Minimum desired value of the resulting image. |
| max_value | Maximum desired value of the resulting image. |

Example

```cpp
const CImg<float> img("reference.jpg"); res = img.get_cut(160, 220);
{img, res}.display();
```

8.1.4.291 quantize()

```cpp
CImg<T>& quantize {
    const unsigned int nb_levels,
    const bool keep_range = true
}
```

Uniformly quantize pixel values.

Parameters

| nb_levels | Number of quantization levels. |
| keep_range | Tells if resulting values keep the same range as the original ones. |

Example

```cpp
const CImg<float> img("reference.jpg"); res = img.get_quantize(4);
{img, res}.display();
```
8.1.4.292 threshold()

CImg<T>& threshold ( 
    const T & value, 
    const bool soft_threshold = false, 
    const bool strict_threshold = false )

Threshold pixel values.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Threshold value</td>
</tr>
<tr>
<td>soft_threshold</td>
<td>Tells if soft thresholding must be applied (instead of hard one).</td>
</tr>
<tr>
<td>strict_threshold</td>
<td>Tells if threshold value is strict.</td>
</tr>
</tbody>
</table>

Example

```cpp
const CImg<float> img("reference.jpg");
const CImg<float> res = img.get_threshold(128);
(img, res).display();
```

8.1.4.293 histogram()

CImg<T>& histogram ( 
    const unsigned int nb_levels, 
    const T & min_value, 
    const T & max_value )

Compute the histogram of pixel values.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nb_levels</td>
<td>Number of desired histogram levels.</td>
</tr>
<tr>
<td>min_value</td>
<td>Minimum pixel value considered for the histogram computation. All pixel values lower than min_value will not be counted.</td>
</tr>
<tr>
<td>max_value</td>
<td>Maximum pixel value considered for the histogram computation. All pixel values higher than max_value will not be counted.</td>
</tr>
</tbody>
</table>

Note

- The histogram H of an image I is the 1D function where H(x) counts the number of occurrences of the value x in the image I.
- The resulting histogram is always defined in 1D. Histograms of multi-valued images are not multi-dimensional.

Example

```cpp
const CImg<float> img = CImg<float>("reference.jpg").histogram(256);
img.display_graph(0, 3);`
8.1.4.294  

**equalize()**

```cpp
CImg<T>& equalize(
    const unsigned int nb_levels,
    const T & min_value,
    const T & max_value
)
```

Equalize histogram of pixel values.

**Parameters**

<table>
<thead>
<tr>
<th>nb_levels</th>
<th>Number of histogram levels used for the equalization.</th>
</tr>
</thead>
<tbody>
<tr>
<td>min_value</td>
<td>Minimum pixel value considered for the histogram computation. All pixel values lower than min_value will not be counted.</td>
</tr>
<tr>
<td>max_value</td>
<td>Maximum pixel value considered for the histogram computation. All pixel values higher than max_value will not be counted.</td>
</tr>
</tbody>
</table>

**Example**

```cpp
const CImg<float> img("reference.jpg");
const CImg<float> res = img.get_equalize(256);
(img,res).display();
```

8.1.4.295  

**index()**

```cpp
CImg<T>& index(
    const CImg<T> & colormap,
    const float dithering = 1,
    const bool map_indexes = false
)
```

Index multi-valued pixels regarding to a specified colormap.

**Parameters**

<table>
<thead>
<tr>
<th>colormap</th>
<th>Multi-valued colormap used as the basis for multi-valued pixel indexing.</th>
</tr>
</thead>
<tbody>
<tr>
<td>dithering</td>
<td>Level of dithering (0=disable, 1=standard level).</td>
</tr>
<tr>
<td>map_indexes</td>
<td>Tell if the values of the resulting image are the colormap indices or the colormap vectors.</td>
</tr>
</tbody>
</table>

**Note**

- `img.index(colormap,dithering,1)` is equivalent to `img.index(colormap,dithering,0).map(colormap)`.

**Example**

```cpp
const CImg<float> img("reference.jpg");
const CImg<float> colormap(3,1,1,3, 0,128,255, 0,128,255, 0,128,255);
const CImg<float> res = img.get_index(colormap,1,true);
(img,res).display();
```
8.1.4.296 **map()**

```cpp
CImg<T>& map(
    const CImg<T> & colormap,
    const unsigned int boundary_conditions = 0)
```

Map predefined colormap on the scalar (indexed) image instance.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>colormap</code></td>
<td>Multi-valued colormap used for mapping the indexes.</td>
</tr>
<tr>
<td><code>boundary_conditions</code></td>
<td>Boundary conditions. Can be { 0=dirichlet</td>
</tr>
</tbody>
</table>

**Example**

```cpp
const CImg<float> img("reference.jpg"),
    colormap1(3,1,1,3, 0,128,255, 0,128,255, 0,128,255),
    colormap2(3,1,1,3, 255,0,0, 0,255,0, 0,0,255),
    res = img.get_index(colormap1,0).map(colormap2);
(img,res).display();
```

8.1.4.297 **label()** [1/2]

```cpp
CImg<T>& label(
    const bool is_high_connectivity = false,
    const Tfloat tolerance = 0)
```

Label connected components.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>is_high_connectivity</code></td>
<td>Boolean that choose between 4(false)- or 8(true)-connectivity in 2D case, and between 6(false)- or 26(true)-connectivity in 3D case.</td>
</tr>
<tr>
<td><code>tolerance</code></td>
<td>Tolerance used to determine if two neighboring pixels belong to the same region.</td>
</tr>
</tbody>
</table>

**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.

8.1.4.298 **label()** [2/2]

```cpp
CImg<T>& label(
    const CImg<T> & connectivity_mask,
    const Tfloat tolerance = 0)
```

Label connected components [overloading].
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connectivity_mask</td>
<td>Mask of the neighboring pixels.</td>
</tr>
<tr>
<td>tolerance</td>
<td>Tolerance used to determine if two neighboring pixels belong to the same region.</td>
</tr>
</tbody>
</table>

8.1.4.299  default_LUT256()

static const CImg<Tuchar>& default_LUT256() [static]

Return colormap "default", containing 256 colors entries in RGB.

Returns

The following 256x1x1x3 colormap is returned:

8.1.4.300  HSV_LUT256()

static const CImg<Tuchar>& HSV_LUT256() [static]

Return colormap "HSV", containing 256 colors entries in RGB.

Returns

The following 256x1x1x3 colormap is returned:

8.1.4.301  lines_LUT256()

static const CImg<Tuchar>& lines_LUT256() [static]

Return colormap "lines", containing 256 colors entries in RGB.

Returns

The following 256x1x1x3 colormap is returned:
8.1.4.302  hot_LUT256()

static const CImg<Tuchar>& hot_LUT256() [static]

Return colormap "hot", containing 256 colors entries in RGB.

Returns
The following 256x1x1x3 colormap is returned:

8.1.4.303  cool_LUT256()

static const CImg<Tuchar>& cool_LUT256() [static]

Return colormap "cool", containing 256 colors entries in RGB.

Returns
The following 256x1x1x3 colormap is returned:

8.1.4.304  jet_LUT256()

static const CImg<Tuchar>& jet_LUT256() [static]

Return colormap "jet", containing 256 colors entries in RGB.

Returns
The following 256x1x1x3 colormap is returned:

8.1.4.305  flag_LUT256()

static const CImg<Tuchar>& flag_LUT256() [static]

Return colormap "flag", containing 256 colors entries in RGB.

Returns
The following 256x1x1x3 colormap is returned:
8.1.4.306  cube_LUT256()

static const CImg<Tuchar>& cube_LUT256() [static]

Return colormap "cube", containing 256 colors entries in RGB.

Returns

The following 256x1x1x3 colormap is returned:

8.1.4.307  RGBtoXYZ()

CImg<T>& RGBtoXYZ (const bool use_D65 = true)

Convert pixel values from RGB to XYZ color spaces.

Parameters

| use_D65 | Tell to use the D65 illuminant (D50 otherwise). |

8.1.4.308  XYZtoRGB()

CImg<T>& XYZtoRGB (const bool use_D65 = true)

Convert pixel values from XYZ to RGB color spaces.

Parameters

| use_D65 | Tell to use the D65 illuminant (D50 otherwise). |

8.1.4.309  resize()[1/3]

CImg<T>& resize (const int size_x,
const int size_y = -100,
const int size_z = -100,
const int size_c = -100,
const int interpolation_type = 1,
const unsigned int boundary_conditions = 0,
const float centering_x = 0,
const float centering_y = 0,
const float centering_z = 0,
const float centering_c = 0 )

Resize image to new dimensions.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>size_x</code></td>
<td>Number of columns (new size along the X-axis).</td>
</tr>
<tr>
<td><code>size_y</code></td>
<td>Number of rows (new size along the Y-axis).</td>
</tr>
<tr>
<td><code>size_z</code></td>
<td>Number of slices (new size along the Z-axis).</td>
</tr>
<tr>
<td><code>size_c</code></td>
<td>Number of vector-channels (new size along the C-axis).</td>
</tr>
<tr>
<td><code>interpolation_type</code></td>
<td>Method of interpolation:</td>
</tr>
<tr>
<td></td>
<td>• -1 = no interpolation: raw memory resizing.</td>
</tr>
<tr>
<td></td>
<td>• 0 = no interpolation: additional space is filled according to <code>boundary_conditions</code>.</td>
</tr>
<tr>
<td></td>
<td>• 1 = nearest-neighbor interpolation.</td>
</tr>
<tr>
<td></td>
<td>• 2 = moving average interpolation.</td>
</tr>
<tr>
<td></td>
<td>• 3 = linear interpolation.</td>
</tr>
<tr>
<td></td>
<td>• 4 = grid interpolation.</td>
</tr>
<tr>
<td></td>
<td>• 5 = cubic interpolation.</td>
</tr>
<tr>
<td></td>
<td>• 6 = lanczos interpolation.</td>
</tr>
<tr>
<td><code>boundary_conditions</code></td>
<td>Type of boundary conditions used if necessary.</td>
</tr>
<tr>
<td><code>centering_x</code></td>
<td>Set centering type (only if <code>interpolation_type</code>=0).</td>
</tr>
<tr>
<td><code>centering_y</code></td>
<td>Set centering type (only if <code>interpolation_type</code>=0).</td>
</tr>
<tr>
<td><code>centering_z</code></td>
<td>Set centering type (only if <code>interpolation_type</code>=0).</td>
</tr>
<tr>
<td><code>centering_c</code></td>
<td>Set centering type (only if <code>interpolation_type</code>=0).</td>
</tr>
</tbody>
</table>

Note

If pd[x,y,z,v]<0, it corresponds to a percentage of the original size (the default value is -100).

8.1.4.310 resize() [2/3]

CImg<T> & resize (  
    const CImg<T> & src,
    const int interpolation_type = 1,
    const unsigned int boundary_conditions = 0,
    const float centering_x = 0,
    const float centering_y = 0,
    const float centering_z = 0,
    const float centering_c = 0 )

Resize image to dimensions of another image.
Parameters

<table>
<thead>
<tr>
<th>src</th>
<th>Reference image used for dimensions.</th>
</tr>
</thead>
<tbody>
<tr>
<td>interpolation_type</td>
<td>Interpolation method.</td>
</tr>
<tr>
<td>boundary_conditions</td>
<td>Boundary conditions.</td>
</tr>
<tr>
<td>centering_x</td>
<td>Set centering type (only if interpolation_type=0).</td>
</tr>
<tr>
<td>centering_y</td>
<td>Set centering type (only if interpolation_type=0).</td>
</tr>
<tr>
<td>centering_z</td>
<td>Set centering type (only if interpolation_type=0).</td>
</tr>
<tr>
<td>centering_c</td>
<td>Set centering type (only if interpolation_type=0).</td>
</tr>
</tbody>
</table>

8.1.4.311 resize() [3/3]

CImg<T>& resize(
    const CImgDisplay & disp,
    const int interpolation_type = 1,
    const unsigned int boundary_conditions = 0,
    const float centering_x = 0,
    const float centering_y = 0,
    const float centering_z = 0,
    const float centering_c = 0 )

Resize image to dimensions of a display window.

Parameters

<table>
<thead>
<tr>
<th>disp</th>
<th>Reference display window used for dimensions.</th>
</tr>
</thead>
<tbody>
<tr>
<td>interpolation_type</td>
<td>Interpolation method.</td>
</tr>
<tr>
<td>boundary_conditions</td>
<td>Boundary conditions.</td>
</tr>
<tr>
<td>centering_x</td>
<td>Set centering type (only if interpolation_type=0).</td>
</tr>
<tr>
<td>centering_y</td>
<td>Set centering type (only if interpolation_type=0).</td>
</tr>
<tr>
<td>centering_z</td>
<td>Set centering type (only if interpolation_type=0).</td>
</tr>
<tr>
<td>centering_c</td>
<td>Set centering type (only if interpolation_type=0).</td>
</tr>
</tbody>
</table>

8.1.4.312 resize_doubleXY()

CImg<T>& resize_doubleXY() { }

Resize image to double-size, using the Scale2X algorithm.

Note

Use anisotropic upscaling algorithm described here.
8.1.4.313  resize_tripleXY()

CImg<T>& resize_tripleXY ( )

Resize image to triple-size, using the Scale3X algorithm.

Note
Use anisotropic upscaling algorithm described here.

8.1.4.314  mirror() [1/2]

CImg<T>& mirror ( const char axis )

Mirror image content along specified axis.

Parameters
axis  Mirror axis

8.1.4.315  mirror() [2/2]

CImg<T>& mirror ( const char *const axes )

Mirror image content along specified axes.

Parameters
axes  Mirror axes, as a C-string.

Note
axes may contain multiple characters, e.g. "xyz"

8.1.4.316  shift()

CImg<T>& shift ( const int delta_x, const int delta_y = 0,
const int delta_z = 0,
const int delta_c = 0,
const unsigned int boundary_conditions = 0
)

Shift image content.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delta_x</td>
<td>Amount of displacement along the X-axis.</td>
</tr>
<tr>
<td>delta_y</td>
<td>Amount of displacement along the Y-axis.</td>
</tr>
<tr>
<td>delta_z</td>
<td>Amount of displacement along the Z-axis.</td>
</tr>
<tr>
<td>delta_c</td>
<td>Amount of displacement along the C-axis.</td>
</tr>
<tr>
<td>boundary_conditions</td>
<td>Boundary conditions. Can be { 0=dirichlet</td>
</tr>
</tbody>
</table>

8.1.4.317 permute_axes()

CImg\langle T\rangle \& permute_axes (  
   const char *const axes_order )

Permute axes order.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axes_order</td>
<td>Axes permutations, as a C-string of 4 characters. This function permutes image content regarding the specified axes permutation.</td>
</tr>
</tbody>
</table>

8.1.4.318 unroll()

CImg\langle T\rangle \& unroll (  
   const char axis )

Unroll pixel values along specified axis.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axis</td>
<td>Unroll axis (can be 'x', 'y', 'z' or 'c').</td>
</tr>
</tbody>
</table>

8.1.4.319 rotate() [1/4]

CImg\langle T\rangle \& rotate (  
   const float angle,
const unsigned int interpolation = 1,
const unsigned int boundary_conditions = 0)

Rotate image with arbitrary angle.

Parameters

<table>
<thead>
<tr>
<th>angle</th>
<th>Rotation angle, in degrees.</th>
</tr>
</thead>
<tbody>
<tr>
<td>interpolation</td>
<td>Type of interpolation. Can be { 0=nearest</td>
</tr>
<tr>
<td>boundary_conditions</td>
<td>Boundary conditions. Can be { 0=dirichlet</td>
</tr>
</tbody>
</table>

Note

The size of the image is modified.

8.1.4.320 rotate() [2/4]

CImg<T> rotate (  
  const float angle,
  const float cx,
  const float cy,
  const unsigned int interpolation,
  const unsigned int boundary_conditions = 0)

Rotate image with arbitrary angle, around a center point.

Parameters

<table>
<thead>
<tr>
<th>angle</th>
<th>Rotation angle, in degrees.</th>
</tr>
</thead>
<tbody>
<tr>
<td>cx</td>
<td>X-coordinate of the rotation center.</td>
</tr>
<tr>
<td>cy</td>
<td>Y-coordinate of the rotation center.</td>
</tr>
<tr>
<td>interpolation</td>
<td>Type of interpolation, { 0=nearest</td>
</tr>
<tr>
<td>boundary_conditions</td>
<td>Boundary conditions, { 0=dirichlet</td>
</tr>
</tbody>
</table>

8.1.4.321 rotate() [3/4]

CImg<T> rotate (  
  const float u,
  const float v,
  const float w,
  const float angle,
  const unsigned int interpolation,
  const unsigned int boundary_conditions )
Rotate volumetric image with arbitrary angle and axis.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( u )</td>
<td>X-coordinate of the 3D rotation axis.</td>
</tr>
<tr>
<td>( v )</td>
<td>Y-coordinate of the 3D rotation axis.</td>
</tr>
<tr>
<td>( w )</td>
<td>Z-coordinate of the 3D rotation axis.</td>
</tr>
<tr>
<td>( \text{angle} )</td>
<td>Rotation angle, in degrees.</td>
</tr>
<tr>
<td>( \text{interpolation} )</td>
<td>Type of interpolation. Can be { 0=nearest</td>
</tr>
<tr>
<td>( \text{boundary_conditions} )</td>
<td>Boundary conditions. Can be { 0=dirichlet</td>
</tr>
</tbody>
</table>

Note

Most of the time, size of the image is modified.

8.1.4.322 rotate() [4/4]

CImg<T> rotate {
    const float \( u \),
    const float \( v \),
    const float \( w \),
    const float \( \text{angle} \),
    const float \( \text{cx} \),
    const float \( \text{cy} \),
    const float \( \text{cz} \),
    const unsigned int \( \text{interpolation} = 1 \),
    const unsigned int \( \text{boundary\_conditions} = 0 \) 
}

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( u )</td>
<td>X-coordinate of the 3D rotation axis.</td>
</tr>
<tr>
<td>( v )</td>
<td>Y-coordinate of the 3D rotation axis.</td>
</tr>
<tr>
<td>( w )</td>
<td>Z-coordinate of the 3D rotation axis.</td>
</tr>
<tr>
<td>( \text{angle} )</td>
<td>Rotation angle, in degrees.</td>
</tr>
<tr>
<td>( \text{cx} )</td>
<td>X-coordinate of the rotation center.</td>
</tr>
<tr>
<td>( \text{cy} )</td>
<td>Y-coordinate of the rotation center.</td>
</tr>
<tr>
<td>( \text{cz} )</td>
<td>Z-coordinate of the rotation center.</td>
</tr>
<tr>
<td>( \text{interpolation} )</td>
<td>Type of interpolation. Can be { 0=nearest</td>
</tr>
<tr>
<td>( \text{boundary_conditions} )</td>
<td>Boundary conditions. Can be { 0=dirichlet</td>
</tr>
</tbody>
</table>

Note

Most of the time, size of the image is modified.
8.1.4.323 warp()

CImg<T>& warp ( 
    const CImg<T>& p_warp, 
    const unsigned int mode = 0, 
    const unsigned int interpolation = 1, 
    const unsigned int boundary_conditions = 0 )

Warp image content by a warping field.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>warp</td>
<td>Warping field.</td>
</tr>
<tr>
<td>mode</td>
<td>Can be { 0=backward-absolute</td>
</tr>
<tr>
<td>interpolation</td>
<td>Can be { 0=nearest</td>
</tr>
<tr>
<td>boundary_conditions</td>
<td>Boundary conditions { 0=dirichlet</td>
</tr>
</tbody>
</table>

8.1.4.324 get_projections2d()

CImg<T> get_projections2d ( 
    const unsigned int x0, 
    const unsigned int y0, 
    const unsigned int z0 ) const

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>X-coordinate of the projection point.</td>
</tr>
<tr>
<td>y0</td>
<td>Y-coordinate of the projection point.</td>
</tr>
<tr>
<td>z0</td>
<td>Z-coordinate of the projection point.</td>
</tr>
</tbody>
</table>

8.1.4.325 crop()

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

Crop image region.
8.1 CImg< T > Struct Template Reference

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. |

boundary_conditions = Can be { 0=dirichlet | 1=neumann | 2=periodic | 3=mirror }.

8.1.4.326 autopdag()  

CImg<T> autopdag (  
    const T *const color = 0,  
    const char *const axes = "zyx" )

Autocrop image region, regarding the specified background color.

Parameters

| color | Color used for the crop. If 0, color is guessed. |
| axes  | Axes used for the crop. |

8.1.4.327 get_column()  

CImg<T> get_column (  
    const int x0 ) const

Return specified image column.

Parameters

| x0 | Image column. |

8.1.4.328 columns()  

CImg<T> columns (  
    const int x0,  
    const int x1 )

Generated by Doxygen
Return specified range of image columns.
8.1 CImg< T > Struct Template Reference

Parameters

<table>
<thead>
<tr>
<th>x0</th>
<th>Starting image column.</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>Ending image column.</td>
</tr>
</tbody>
</table>

8.1.4.329 row()

CImg< T > & row (  
  const int y0)

Return specified image row [in-place version].

Parameters

| y0 | Image row. |

8.1.4.330 get_rows()

CImg< T > get_rows (  
  const int y0,    
  const int y1) const

Return specified range of image rows.

Parameters

| y0 | Starting image row. |
| y1 | Ending image row.   |

8.1.4.331 get_slice()

CImg< T > get_slice (  
  const int z0) const

Return specified image slice.

Parameters

| z0 | Image slice. |

Generated by Doxygen
8.1.4.332 get_slices()

```cpp
CImg<T> get_slices(
    const int z0,
    const int z1 ) const
```

Return specified range of image slices.

**Parameters**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>z0</code></td>
<td>Starting image slice.</td>
</tr>
<tr>
<td><code>z1</code></td>
<td>Ending image slice.</td>
</tr>
</tbody>
</table>

8.1.4.333 get_channel()

```cpp
CImg<T> get_channel(
    const int c0 ) const
```

Return specified image channel.

**Parameters**

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>c0</code></td>
</tr>
</tbody>
</table>

8.1.4.334 get_channels()

```cpp
CImg<T> get_channels(
    const int c0,
    const int c1 ) const
```

Return specified range of image channels.

**Parameters**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>c0</code></td>
<td>Starting image channel.</td>
</tr>
<tr>
<td><code>c1</code></td>
<td>Ending image channel.</td>
</tr>
</tbody>
</table>

8.1.4.335 streamline()

```cpp
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 streamline of a 3D vector field.

Parameters

<table>
<thead>
<tr>
<th>func</th>
<th>Vector field function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>X-coordinate of the starting point of the streamline.</td>
</tr>
<tr>
<td>y</td>
<td>Y-coordinate of the starting point of the streamline.</td>
</tr>
<tr>
<td>z</td>
<td>Z-coordinate of the starting point of the streamline.</td>
</tr>
<tr>
<td>L</td>
<td>Streamline length.</td>
</tr>
<tr>
<td>dl</td>
<td>Streamline length increment.</td>
</tr>
<tr>
<td>interpolation_type</td>
<td>Type of interpolation. Can be { 0=nearest int</td>
</tr>
<tr>
<td>is_backward_tracking</td>
<td>Tells if the streamline is estimated forward or backward.</td>
</tr>
<tr>
<td>is_oriented_only</td>
<td>Tells if the direction of the vectors must be ignored.</td>
</tr>
<tr>
<td>x0</td>
<td>X-coordinate of the first bounding-box vertex.</td>
</tr>
<tr>
<td>y0</td>
<td>Y-coordinate of the first bounding-box vertex.</td>
</tr>
<tr>
<td>z0</td>
<td>Z-coordinate of the first bounding-box vertex.</td>
</tr>
<tr>
<td>x1</td>
<td>X-coordinate of the second bounding-box vertex.</td>
</tr>
<tr>
<td>y1</td>
<td>Y-coordinate of the second bounding-box vertex.</td>
</tr>
<tr>
<td>z1</td>
<td>Z-coordinate of the second bounding-box vertex.</td>
</tr>
</tbody>
</table>

8.1.4.336 get_shared_points()

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

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

Parameters

| x0 | X-coordinate of the starting pixel. |
Parameters

<table>
<thead>
<tr>
<th>x1</th>
<th>X-coordinate of the ending pixel.</th>
</tr>
</thead>
<tbody>
<tr>
<td>y0</td>
<td>Y-coordinate.</td>
</tr>
<tr>
<td>z0</td>
<td>Z-coordinate.</td>
</tr>
<tr>
<td>c0</td>
<td>C-coordinate.</td>
</tr>
</tbody>
</table>

8.1.4.337  get_shared_rows()

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

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

Parameters

<table>
<thead>
<tr>
<th>y0</th>
<th>Y-coordinate of the starting row.</th>
</tr>
</thead>
<tbody>
<tr>
<td>y1</td>
<td>Y-coordinate of the ending row.</td>
</tr>
<tr>
<td>z0</td>
<td>Z-coordinate.</td>
</tr>
<tr>
<td>c0</td>
<td>C-coordinate.</td>
</tr>
</tbody>
</table>

8.1.4.338  get_shared_row()

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

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

Parameters

<table>
<thead>
<tr>
<th>y0</th>
<th>Y-coordinate.</th>
</tr>
</thead>
<tbody>
<tr>
<td>z0</td>
<td>Z-coordinate.</td>
</tr>
<tr>
<td>c0</td>
<td>C-coordinate.</td>
</tr>
</tbody>
</table>

8.1.4.339  get_shared_slices()

```cpp
CImg<T> get_shared_slices (  
    const unsigned int z0,  
```
8.1 CImg< T > Struct Template Reference

const unsigned int z1,
const unsigned int c0 = 0 )

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

Parameters

<table>
<thead>
<tr>
<th>z0</th>
<th>Z-coordinate of the starting slice.</th>
</tr>
</thead>
<tbody>
<tr>
<td>z1</td>
<td>Z-coordinate of the ending slice.</td>
</tr>
<tr>
<td>c0</td>
<td>C-coordinate.</td>
</tr>
</tbody>
</table>

8.1.4.340 get_shared_slice()

CImg< T > get_shared_slice (  
const unsigned int z0,
const unsigned int c0 = 0 )

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

Parameters

<table>
<thead>
<tr>
<th>z0</th>
<th>Z-coordinate.</th>
</tr>
</thead>
<tbody>
<tr>
<td>c0</td>
<td>C-coordinate.</td>
</tr>
</tbody>
</table>

8.1.4.341 get_shared_channels()

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

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

Parameters

<table>
<thead>
<tr>
<th>c0</th>
<th>C-coordinate of the starting channel.</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>C-coordinate of the ending channel.</td>
</tr>
</tbody>
</table>

8.1.4.342 get_shared_channel()

CImg< T > get_shared_channel (  
const unsigned int c0 )

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

| c0    | C-coordinate. |

8.1.4.343 get_split() [1/2]

CImgList<T> get_split (  
const char axis,  
const int nb = -1 ) const

Split image into a list along specified axis.

Parameters

| axis | Splitting axis. Can be { 'x' | 'y' | 'z' | 'c' }. |
| nb   | Number of split parts. |

Note

- If nb==0, instance image is split into blocs of egal values along the specified axis.
- If nb<=0, instance image is split into blocs of -nb pixel wide.
- If nb>0, instance image is split into nb blocs.

8.1.4.344 get_split() [2/2]

CImgList<T> get_split (  
const CImg<t>& values,  
const char axis = 0,  
const bool keep_values = true ) const

Split image into a list of sub-images, according to a specified splitting value sequence and optionally axis.

Parameters

| values | Splitting value sequence. |
| axis   | Axis along which the splitting is performed. Can be '0' to ignore axis. |
| keep_values | Tells if the splitting sequence must be kept in the split blocs. |

8.1.4.345 append()}

CImg<T>& append (  
const CImg<t>& img,  
...
Append two images along specified axis.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>img</strong></td>
<td>Image to append with instance image.</td>
</tr>
<tr>
<td><strong>axis</strong></td>
<td>Appending axis. Can be ( 'x'</td>
</tr>
<tr>
<td><strong>align</strong></td>
<td>Append alignment in [0,1].</td>
</tr>
</tbody>
</table>

8.1.4.346 correlate()

Clmg<T>& correlate {
    const Clmg<T>& kernel,
    const unsigned int boundary_conditions = 1,
    const bool is_normalized = false,
    const unsigned int channel_mode = 1,
    const unsigned int xcenter = ~0U,
    const unsigned int ycenter = ~0U,
    const unsigned int zcenter = ~0U,
    const unsigned int xstart = 0,
    const unsigned int ystart = 0,
    const unsigned int zstart = 0,
    const unsigned int xend = ~0U,
    const unsigned int yend = ~0U,
    const unsigned int zend = ~0U,
    const float xstride = 1,
    const float ystride = 1,
    const float zstride = 1,
    const float xdilation = 1,
    const float ydilation = 1,
    const float zdilation = 1
}

Correlate image by a kernel.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>kernel</strong></td>
<td>= the correlation kernel.</td>
</tr>
<tr>
<td><strong>boundary_conditions</strong></td>
<td>Boundary condition. Can be { 0=dirichlet</td>
</tr>
<tr>
<td><strong>is_normalized</strong></td>
<td>= enable local normalization.</td>
</tr>
<tr>
<td><strong>channel</strong></td>
<td>= channel processing mode. Can be { 0=sum inputs</td>
</tr>
<tr>
<td><strong>xcenter</strong></td>
<td>X-coordinate of the kernel center (~0U means 'centered').</td>
</tr>
<tr>
<td><strong>xstart</strong></td>
<td>Starting X-coordinate of the instance image.</td>
</tr>
<tr>
<td><strong>xend</strong></td>
<td>Ending X-coordinate of the instance image.</td>
</tr>
<tr>
<td><strong>xstride</strong></td>
<td>Stride along the X-axis.</td>
</tr>
<tr>
<td><strong>xdilation</strong></td>
<td>Dilation along the X-axis.</td>
</tr>
<tr>
<td><strong>ycenter</strong></td>
<td>Y-coordinate of the kernel center (~0U means 'centered').</td>
</tr>
<tr>
<td><strong>ystart</strong></td>
<td>Starting Y-coordinate of the instance image.</td>
</tr>
<tr>
<td><strong>yend</strong></td>
<td>Ending Y-coordinate of the instance image.</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ystride</td>
<td>Stride along the Y-axis.</td>
</tr>
<tr>
<td>ydilation</td>
<td>Dilation along the Y-axis.</td>
</tr>
<tr>
<td>zcenter</td>
<td>Z-coordinate of the kernel center ((\sim0U) means 'centered').</td>
</tr>
<tr>
<td>zstart</td>
<td>Starting Z-coordinate of the instance image.</td>
</tr>
<tr>
<td>zend</td>
<td>Ending Z-coordinate of the instance image.</td>
</tr>
<tr>
<td>zstride</td>
<td>Stride along the Z-axis.</td>
</tr>
<tr>
<td>zdilation</td>
<td>Dilation along the Z-axis.</td>
</tr>
</tbody>
</table>

Note

- The correlation of the image instance 
  \(\star\) this by the kernel \(\text{kernel}\) is defined to be:
  \(\text{res}(x,y,z) = \sum_{i,j,k}^{\star\text{this}}(x+i-c_x, y+j-c_y, z+k-c_z) \ast \text{kernel}(i,j,k)\).

8.1.4.347 convolve()

```cpp
CImg<

Convolve image by a kernel.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kernel</td>
<td>= the correlation kernel.</td>
</tr>
<tr>
<td>boundary_conditions</td>
<td>Boundary condition. Can be { 0=dirichlet</td>
</tr>
<tr>
<td>is_normalized</td>
<td>= enable local normalization.</td>
</tr>
<tr>
<td>channel</td>
<td>mode Channel processing mode. Can be { 0=sum inputs</td>
</tr>
<tr>
<td>xcenter</td>
<td>X-coordinate of the kernel center ((\sim0U) means 'centered').</td>
</tr>
<tr>
<td>xstart</td>
<td>Starting X-coordinate of the instance image.</td>
</tr>
<tr>
<td>xend</td>
<td>Ending X-coordinate of the instance image.</td>
</tr>
<tr>
<td>xstride</td>
<td>Stride along the X-axis.</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xdilation</td>
<td>Dilation along the X-axis.</td>
</tr>
<tr>
<td>ycenter</td>
<td>Y-coordinate of the kernel center ((\sim 0\text{U}) means 'centered').</td>
</tr>
<tr>
<td>ystart</td>
<td>Starting Y-coordinate of the instance image.</td>
</tr>
<tr>
<td>yend</td>
<td>Ending Y-coordinate of the instance image.</td>
</tr>
<tr>
<td>ystride</td>
<td>Stride along the Y-axis.</td>
</tr>
<tr>
<td>ydilation</td>
<td>Dilation along the Y-axis.</td>
</tr>
<tr>
<td>zcenter</td>
<td>Z-coordinate of the kernel center ((\sim 0\text{U}) means 'centered').</td>
</tr>
<tr>
<td>zstart</td>
<td>Starting Z-coordinate of the instance image.</td>
</tr>
<tr>
<td>zend</td>
<td>Ending Z-coordinate of the instance image.</td>
</tr>
<tr>
<td>zstride</td>
<td>Stride along the Z-axis.</td>
</tr>
<tr>
<td>zdilation</td>
<td>Dilation along the Z-axis.</td>
</tr>
</tbody>
</table>

Note

- The convolution of the image instance \(*this\) by the kernel \(\text{kernel}\) is defined to be:
  \[
  \text{res}(x,y,z) = \sum_{i,j,k} (\!*this!*(x - (i - c_x), y - (j - c_y), z - (k - c_z)) \!\! \times \!\! \text{kernel}(i,j,k).
  \]

8.1.4.348 \texttt{cumulate()} [1/2]

\texttt{CImg<& cumulate (}
\texttt{  const char axis = 0 )}

Cumulate image values, optionally along specified axis.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axis</td>
<td>Cumulation axis. Set it to 0 to cumulate all values globally without taking axes into account.</td>
</tr>
</tbody>
</table>

8.1.4.349 \texttt{cumulate()} [2/2]

\texttt{CImg<& cumulate (}
\texttt{  const char *const axes )}

Cumulate image values, along specified axes.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axes</td>
<td>Cumulation axes, as a C-string.</td>
</tr>
</tbody>
</table>
Note

axes may contain multiple characters, e.g. "xyz"

8.1.4.350 erode() [1/3]

CImg<T>& erode {
    const CImg< t > & kernel,
    const bool boundary_conditions = true,
    const bool is_real = false
}

Erode image by a structuring element.

Parameters

<table>
<thead>
<tr>
<th>kernel</th>
<th>Structuring element.</th>
</tr>
</thead>
<tbody>
<tr>
<td>boundary_conditions</td>
<td>Boundary conditions.</td>
</tr>
<tr>
<td>is_real</td>
<td>Do the erosion in real (a.k.a 'non-flat') mode (true) rather than binary mode (false).</td>
</tr>
</tbody>
</table>

8.1.4.351 erode() [2/3]

CImg<T>& erode {
    const unsigned int sx,
    const unsigned int sy,
    const unsigned int sz = 1
}

Erode image by a rectangular structuring element of specified size.

Parameters

<table>
<thead>
<tr>
<th>sx</th>
<th>Width of the structuring element.</th>
</tr>
</thead>
<tbody>
<tr>
<td>sy</td>
<td>Height of the structuring element.</td>
</tr>
<tr>
<td>sz</td>
<td>Depth of the structuring element.</td>
</tr>
</tbody>
</table>

8.1.4.352 erode() [3/3]

CImg<T>& erode {
    const unsigned int s
}

Erode the image by a square structuring element of specified size.
Parameters

| s  | Size of the structuring element. |

8.1.4.353 dilate() [1/3]

\[ CImg<T>& \text{dilate}(\text{const } CImg<T>& \text{kernel,}\]
\[ \text{const bool boundary_conditions = true,}\]
\[ \text{const bool is_real = false})\]

Dilate image by a structuring element.

| kernel   | Structuring element. |
| boundary_conditions | Boundary conditions. |
| is_real   | Do the dilation in real (a.k.a 'non-flat') mode (true) rather than binary mode (false). |

8.1.4.354 dilate() [2/3]

\[ CImg<T>& \text{dilate}(\text{const unsigned int } sx,\]
\[ \text{const unsigned int } sy,\]
\[ \text{const unsigned int } sz = 1)\]

Dilate image by a rectangular structuring element of specified size.

| sx   | Width of the structuring element. |
| sy   | Height of the structuring element. |
| sz   | Depth of the structuring element. |

8.1.4.355 dilate() [3/3]

\[ CImg<T>& \text{dilate}(\text{const unsigned int } s)\]

Dilate image by a square structuring element of specified size.
Parameters

$s$ Size of the structuring element.

8.1.4.356 watershed()

```cpp
CImg<typename>& watershed(
    const CImg<typename>& priority,
    const bool is_high_connectivity = false)
```

Compute watershed transform.

Parameters

<table>
<thead>
<tr>
<th>priority</th>
<th>Priority map.</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_high_connectivity</td>
<td>Boolean that choose between 4(false)- or 8(true)-connectivity in 2D case, and between 6(false)- or 26(true)-connectivity in 3D case.</td>
</tr>
</tbody>
</table>

Note

Non-zero values of the instance instance are propagated to zero-valued ones according to specified the priority map.

8.1.4.357 deriche()

```cpp
CImg<typename>& deriche(
    const float sigma,
    const unsigned int order = 0,
    const char axis = 'x',
    const bool boundary_conditions = true)
```

Apply recursive Deriche filter.

Parameters

<table>
<thead>
<tr>
<th>sigma</th>
<th>Standard deviation of the filter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>order</td>
<td>Order of the filter. Can be { 0=smooth-filter</td>
</tr>
<tr>
<td>axis</td>
<td>Axis along which the filter is computed. Can be { 'x'</td>
</tr>
<tr>
<td>boundary_conditions</td>
<td>Boundary conditions. Can be { 0=dirichlet</td>
</tr>
</tbody>
</table>
8.1.4.358 vanvliet()

```cpp
CImg<T>& vanvliet(
    const float sigma,
    const unsigned int order,
    const char axis = 'x',
    const bool boundary_conditions = true)
```

Van Vliet recursive Gaussian filter.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sigma</td>
<td>Standard deviation of the Gaussian filter</td>
</tr>
<tr>
<td>order</td>
<td>The order of the filter 0,1,2,3</td>
</tr>
<tr>
<td>axis</td>
<td>Axis along which the filter is computed. Can be { 'x'</td>
</tr>
<tr>
<td>boundary_conditions</td>
<td>Boundary conditions. Can be { 0=dirichlet</td>
</tr>
</tbody>
</table>

**Note**

dirichlet boundary condition has a strange behavior


(this is an improvement over Young-Van Vliet, Sig. Proc. 44, 1995)


8.1.4.359 blur() [1/2]

```cpp
CImg<T>& blur(
    const float sigma_x,
    const float sigma_y,
    const float sigma_z,
    const bool boundary_conditions = true,
    const bool is_gaussian = false)
```

Blur image.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sigma_x</td>
<td>Standard deviation of the blur, along the X-axis.</td>
</tr>
<tr>
<td>sigma_y</td>
<td>Standard deviation of the blur, along the Y-axis.</td>
</tr>
<tr>
<td>sigma_z</td>
<td>Standard deviation of the blur, along the Z-axis.</td>
</tr>
<tr>
<td>boundary_conditions</td>
<td>Boundary conditions. Can be { false=dirichlet</td>
</tr>
<tr>
<td>is_gaussian</td>
<td>Tells if the blur uses a gaussian (true) or quasi-gaussian (false) kernel.</td>
</tr>
</tbody>
</table>
Note

- The blur is computed as a 0-order Deriche filter. This is not a gaussian blur.
- This is a recursive algorithm, not depending on the values of the standard deviations.

See also

deriche(), vanvliet().

8.1.4.360 blur() [2/2]

CImg\langle T\rangle & \text{blur (}
\hspace{1em} \text{const float } \text{sigma,}
\hspace{1em} \text{const bool } \text{boundary_conditions } = \text{true,}
\hspace{1em} \text{const bool } \text{is_gaussian } = \text{false })

Blur image isotropically.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sigma</td>
<td>Standard deviation of the blur.</td>
</tr>
<tr>
<td>boundary_conditions</td>
<td>Boundary conditions. Can be { 0=dirichlet</td>
</tr>
<tr>
<td>is_gaussian</td>
<td>Use a gaussian kernel (VanVliet) is set, a pseudo-gaussian (Deriche) otherwise.</td>
</tr>
</tbody>
</table>

See also

deriche(), vanvliet().

8.1.4.361 blur_anisotropic() [1/2]

CImg\langle T\rangle & \text{blur_anisotropic (}
\hspace{1em} \text{const CImg\langle t \rangle & } \text{G,}
\hspace{1em} \text{const float } \text{amplitude } = 60,
\hspace{1em} \text{const float } \text{dl } = 0.8f,
\hspace{1em} \text{const float } \text{ds } = 30,
\hspace{1em} \text{const float } \text{gauss_prec } = 2,
\hspace{1em} \text{const unsigned int } \text{interpolation_type } = 0,
\hspace{1em} \text{const bool } \text{is_fast_approx } = 1 )

Blur image anisotropically, directed by a field of diffusion tensors.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>Field of square roots of diffusion tensors/vectors used to drive the smoothing.</td>
</tr>
<tr>
<td>amplitude</td>
<td>Amplitude of the smoothing.</td>
</tr>
<tr>
<td>dl</td>
<td>Spatial discretization.</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>da</td>
<td>Angular discretization.</td>
</tr>
<tr>
<td>gauss_prec</td>
<td>Precision of the diffusion process.</td>
</tr>
<tr>
<td>interpolation_type</td>
<td>Interpolation scheme. Can be { 0=nearest-neighbor</td>
</tr>
<tr>
<td>is_fast_approx</td>
<td>Tells if a fast approximation of the gaussian function is used or not.</td>
</tr>
</tbody>
</table>

8.1.4.362 blur_anisotropic() [2/2]

```cpp
CImg< T >& blur_anisotropic(
    const float amplitude,
    const float sharpness = 0.7f,
    const float anisotropy = 0.6f,
    const float alpha = 0.6f,
    const float sigma = 1.1f,
    const float dl = 0.8f,
    const float da = 30,
    const float gauss_prec = 2,
    const unsigned int interpolation_type = 0,
    const bool is_fast_approx = true
)
```

Blur image anisotropically, in an edge-preserving way.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amplitude</td>
<td>Amplitude of the smoothing.</td>
</tr>
<tr>
<td>sharpness</td>
<td>Sharpness.</td>
</tr>
<tr>
<td>anisotropy</td>
<td>Anisotropy.</td>
</tr>
<tr>
<td>alpha</td>
<td>Standard deviation of the gradient blur.</td>
</tr>
<tr>
<td>sigma</td>
<td>Standard deviation of the structure tensor blur.</td>
</tr>
<tr>
<td>dl</td>
<td>Spatial discretization.</td>
</tr>
<tr>
<td>da</td>
<td>Angular discretization.</td>
</tr>
<tr>
<td>gauss_prec</td>
<td>Precision of the diffusion process.</td>
</tr>
<tr>
<td>interpolation_type</td>
<td>Interpolation scheme. Can be { 0=nearest-neighbor</td>
</tr>
<tr>
<td>is_fast_approx</td>
<td>Tells if a fast approximation of the gaussian function is used or not.</td>
</tr>
</tbody>
</table>

8.1.4.363 blur_bilateral() [1/2]

```cpp
CImg< T >& blur_bilateral(
    const CImg< T >& guide,
    const float sigma_x,
    const float sigma_y,
    const float sigma_z,
)
```

Generated by Doxygen
Blur image, with the joint bilateral filter.

**Parameters**

<table>
<thead>
<tr>
<th>guide</th>
<th>Image used to model the smoothing weights.</th>
</tr>
</thead>
<tbody>
<tr>
<td>sigma_x</td>
<td>Amount of blur along the X-axis.</td>
</tr>
<tr>
<td>sigma_y</td>
<td>Amount of blur along the Y-axis.</td>
</tr>
<tr>
<td>sigma_z</td>
<td>Amount of blur along the Z-axis.</td>
</tr>
<tr>
<td>sampling_x</td>
<td>Amount of downsampling along the X-axis used for the approximation. Defaults (0) to sigma_x.</td>
</tr>
<tr>
<td>sampling_y</td>
<td>Amount of downsampling along the Y-axis used for the approximation. Defaults (0) to sigma_y.</td>
</tr>
<tr>
<td>sampling_z</td>
<td>Amount of downsampling along the Z-axis used for the approximation. Defaults (0) to sigma_z.</td>
</tr>
<tr>
<td>sampling_r</td>
<td>Amount of downsampling along the value axis used for the approximation. Defaults (0) to sigma_r.</td>
</tr>
</tbody>
</table>

**Note**

This algorithm uses the optimisation technique proposed by S. Paris and F. Durand, in ECCV2006 (extended for 3D volumetric images). It is based on the reference implementation [http://people.csail.mit.edu/jiawen/software/bilateralFilter.m](http://people.csail.mit.edu/jiawen/software/bilateralFilter.m)

```cpp
CImg<T>& blur_bilateral (  
    const CImg<T>& guide,  
    const float sigma\_s,  
    const float sigma\_r,  
    const float sampling\_s = 0,  
    const float sampling\_r = 0 )
```

Blur image using the joint bilateral filter.

**Parameters**

<table>
<thead>
<tr>
<th>guide</th>
<th>Image used to model the smoothing weights.</th>
</tr>
</thead>
<tbody>
<tr>
<td>sigma_s</td>
<td>Amount of blur along the XYZ-axes.</td>
</tr>
<tr>
<td>sigma_r</td>
<td>Amount of blur along the value axis.</td>
</tr>
<tr>
<td>sampling_s</td>
<td>Amount of downsampling along the XYZ-axes used for the approximation. Defaults to sigma_s.</td>
</tr>
<tr>
<td>sampling_r</td>
<td>Amount of downsampling along the value axis used for the approximation. Defaults to sigma_r.</td>
</tr>
</tbody>
</table>
8.1.4.365  boxfilter()

\[ \text{CImg}\langle T \rangle \& \text{boxfilter} ( \right. \]
\[
\text{const float boxsize,} \\
\text{const int order,} \\
\text{const char axis = 'x',} \\
\text{const bool boundary_conditions = true,} \\
\text{const unsigned int nb_iter = 1} \)
\]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boxsize</td>
<td>Size of the box window (can be subpixel)</td>
</tr>
<tr>
<td>order</td>
<td>the order of the filter 0,1 or 2.</td>
</tr>
<tr>
<td>axis</td>
<td>Axis along which the filter is computed. Can be { 'x'</td>
</tr>
<tr>
<td>boundary_conditions</td>
<td>Boundary conditions. Can be { 0=dirichlet</td>
</tr>
<tr>
<td>nb_iter</td>
<td>Number of filter iterations.</td>
</tr>
</tbody>
</table>

8.1.4.366  blur_box()[/2]

\[ \text{CImg}\langle T \rangle \& \text{blur_box} ( \right. \]
\[
\text{const float boxsize}_x, \\
\text{const float boxsize}_y, \\
\text{const float boxsize}_z, \\
\text{const bool boundary_conditions = true,} \\
\text{const unsigned int nb_iter = 1} \)
\]

Blur image with a box filter.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boxsize_x</td>
<td>Size of the box window, along the X-axis (can be subpixel).</td>
</tr>
<tr>
<td>boxsize_y</td>
<td>Size of the box window, along the Y-axis (can be subpixel).</td>
</tr>
<tr>
<td>boxsize_z</td>
<td>Size of the box window, along the Z-axis (can be subpixel).</td>
</tr>
<tr>
<td>boundary_conditions</td>
<td>Boundary conditions. Can be { false=dirichlet</td>
</tr>
<tr>
<td>nb_iter</td>
<td>Number of filter iterations.</td>
</tr>
</tbody>
</table>

Note

- This is a recursive algorithm, not depending on the values of the box kernel size.

See also

- blur().
8.1.4.367 blur_box()

CImg<T>& blur_box (  
    const float boxsize,  
    const bool boundary_conditions = true )

Blur image with a box filter.

Parameters

| boxsize   | Size of the box window (can be subpixel). |
| boundary_conditions | Boundary conditions. Can be \{ 0=dirichlet | 1=neumann \}.a |

See also
deriche(), vanvliet().

8.1.4.368 blur_guided()

CImg<T>& blur_guided (  
    const CImg< t > & guide,  
    const float radius,  
    const float regularization )

Blur image, with the image guided filter.

Parameters

| guide        | Image used to guide the smoothing process. |
| radius       | Spatial radius. If negative, it is expressed as a percentage of the largest image size. |
| regularization | Regularization parameter. If negative, it is expressed as a percentage of the guide value range. |

Note

This method implements the filtering algorithm described in: He, Kaiming; Sun, Jian; Tang, Xiaoou, "Guided Image Filtering," Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol.35, no.6, pp.1397.1409, June 2013

8.1.4.369 blur_patch()

CImg<T>& blur_patch (  
    const CImg< t > & guide,  
    const float sigma_s,  
    const float sigma_r,  
    const unsigned int patch_size = 3,  

Generated by Doxygen
const unsigned int lookup_size = 4,
const float smoothness = 0,
const bool is_fast_approx = true
)

Blur image using patch-based space.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>guide</td>
<td>Image used to model the smoothing weights.</td>
</tr>
<tr>
<td>sigma_s</td>
<td>Amount of blur along the XYZ-axes.</td>
</tr>
<tr>
<td>sigma_r</td>
<td>Amount of blur along the value axis.</td>
</tr>
<tr>
<td>patch_size</td>
<td>Size of the patches.</td>
</tr>
<tr>
<td>lookup_size</td>
<td>Size of the window to search similar patches.</td>
</tr>
<tr>
<td>smoothness</td>
<td>Smoothness for the patch comparison.</td>
</tr>
<tr>
<td>is_fast_approx</td>
<td>Tells if a fast approximation of the gaussian function is used or not.</td>
</tr>
</tbody>
</table>

8.1.4.370 blur_median()

CImg<T>4 blur_median (  
    const unsigned int n,
    const float threshold = 0  )

Blur image with the median filter.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Size of the median filter.</td>
</tr>
<tr>
<td>threshold</td>
<td>Threshold used to discard pixels too far from the current pixel value in the median computation.</td>
</tr>
</tbody>
</table>

8.1.4.371 sharpen()

CImg<T>4 sharpen (  
    const float amplitude,
    const bool sharpen_type = false,
    const float edge = 1,
    const float alpha = 0,
    const float sigma = 0  )

Sharpen image.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amplitude</td>
<td>Sharpening amplitude</td>
</tr>
<tr>
<td>sharpen_type</td>
<td>Select sharpening method. Can be { false=inverse diffusion</td>
</tr>
<tr>
<td>edge</td>
<td>Edge threshold (shock filters only).</td>
</tr>
<tr>
<td>alpha</td>
<td>Gradient smoothness (shock filters only).</td>
</tr>
<tr>
<td>sigma</td>
<td>Tensor smoothness (shock filters only).</td>
</tr>
</tbody>
</table>
8.1.4.372  get_gradient()

CImgList<Tfloat> get_gradient (  
    const char *const axes = 0,  
    const int scheme = 0 ) const

Return image gradient.

Parameters

<table>
<thead>
<tr>
<th>axes</th>
<th>Axes considered for the gradient computation, as a C-string (e.g &quot;xy&quot;).</th>
</tr>
</thead>
<tbody>
<tr>
<td>scheme</td>
<td>Numerical scheme used for the gradient computation:</td>
</tr>
<tr>
<td></td>
<td>• -1 = Backward finite differences</td>
</tr>
<tr>
<td></td>
<td>• 0 = Centered finite differences (default)</td>
</tr>
<tr>
<td></td>
<td>• 1 = Forward finite differences</td>
</tr>
<tr>
<td></td>
<td>• 2 = Using Sobel kernels</td>
</tr>
<tr>
<td></td>
<td>• 3 = Using rotation invariant kernels</td>
</tr>
<tr>
<td></td>
<td>• 4 = Using Deriche recursive filter.</td>
</tr>
<tr>
<td></td>
<td>• 5 = Using Van Vliet recursive filter.</td>
</tr>
</tbody>
</table>

8.1.4.373  get_hessian()

CImgList<Tfloat> get_hessian (  
    const char *const axes = 0 ) const

Return image hessian.

Parameters

| axes | Axes considered for the hessian computation, as a C-string (e.g "xy"). |

8.1.4.374  structure_tensors()

CImg<T> & structure_tensors (  
    const bool is_fwbw_scheme = false )

Compute the structure tensor field of an image.
Parameters

| is_fwbw_scheme | scheme. Can be [false=centered | true=forward-backward]

8.1.4.375 diffusion_tensors()

```cpp
cImg< T >& diffusion_tensors (  
    const float sharpness = 0.7f,  
    const float anisotropy = 0.6f,  
    const float alpha = 0.6f,  
    const float sigma = 1.1f,  
    const bool is_sqrt = false  
)
```

Compute field of diffusion tensors for edge-preserving smoothing.

Parameters

| sharpness | Sharpness |
| anisotropy | Anisotropy |
| alpha | Standard deviation of the gradient blur. |
| sigma | Standard deviation of the structure tensor blur. |
| is_sqrt | Tells if the square root of the tensor field is computed instead. |

8.1.4.376 displacement()

```cpp
cImg< T >& displacement (  
    const CImg< T >& source,  
    const float smoothness = 0.1f,  
    const float precision = 5.f,  
    const unsigned int nb_scales = 0,  
    const unsigned int iteration_max = 10000,  
    const bool is_backward = false,  
    const CImg< floatT >& guide = CImg< floatT >::const_empty()  
)
```

Estimate displacement field between two images.

Parameters

| source | Reference image. |
| smoothness | Smoothness of estimated displacement field. |
| precision | Precision required for algorithm convergence. |
| nb_scales | Number of scales used to estimate the displacement field. |
| iteration_max | Maximum number of iterations allowed for one scale. |
| is_backward | If false, match I2(X + U(X)) = I1(X), else match I2(X) = I1(X - U(X)). |
| guide | Image used as the initial correspondence estimate for the algorithm. 'guide' may have a last channel with boolean values (0=false | other=true) that tells for each pixel if its correspondence vector is constrained to its initial value (constraint mask). |
8.1.4.377  matchpatch()

```cpp
CImg<T>& matchpatch {
    const CImg<T> & patch_image,
    const unsigned int patch_width,
    const unsigned int patch_height,
    const unsigned int patch_depth,
    const unsigned int nb_iterations,
    const unsigned int nb_randoms,
    const float occ_penalization,
    const CImg<tl> & guide,
    CImg<t2> & matching_score )
```

Compute correspondence map between two images, using a patch-matching algorithm.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>patch_image</td>
<td>The image containing the reference patches to match with the instance image.</td>
</tr>
<tr>
<td>patch_width</td>
<td>Width of the patch used for matching.</td>
</tr>
<tr>
<td>patch_height</td>
<td>Height of the patch used for matching.</td>
</tr>
<tr>
<td>patch_depth</td>
<td>Depth of the patch used for matching.</td>
</tr>
<tr>
<td>nb_iterations</td>
<td>Number of patch-match iterations.</td>
</tr>
<tr>
<td>nb_randoms</td>
<td>Number of randomization attempts (per pixel).</td>
</tr>
<tr>
<td>occ_penalization</td>
<td>Penalization factor in score related patch occurrences.</td>
</tr>
<tr>
<td>guide</td>
<td>Image used as the initial correspondence estimate for the algorithm. 'guide' may have a last channel with boolean values (0=false</td>
</tr>
<tr>
<td>matching_score</td>
<td>Returned as the image of matching scores.</td>
</tr>
</tbody>
</table>

8.1.4.378  distance() [1/2]

```cpp
CImg<T>& distance {
    const T & value,
    const unsigned int metric = 2 }
```

Compute Euclidean distance function to a specified value.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Reference value.</td>
</tr>
<tr>
<td>metric</td>
<td>Type of metric. Can be { 0=Chebyshev</td>
</tr>
</tbody>
</table>
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.

8.1.4.379  distance()

CImg<T>& distance (  
    const T & value,  
    const CImg< t > & metric_mask )

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

Parameters

<table>
<thead>
<tr>
<th>value</th>
<th>Reference value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>metric_mask</td>
<td>Metric mask.</td>
</tr>
</tbody>
</table>

Note

The algorithm code has been initially proposed by A. Meijster, and modified by D. Tschumperlé.

8.1.4.380  distance_dijkstra()

CImg<T>& distance_dijkstra (  
    const T & value,  
    const CImg< t > & metric,  
    const bool is_high_connectivity,  
    CImg< to > & return_path )

Compute distance to a specified value, according to a custom metric (use dijkstra algorithm).

Parameters

<table>
<thead>
<tr>
<th>value</th>
<th>Reference value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>metric</td>
<td>Field of distance potentials.</td>
</tr>
<tr>
<td>is_high_connectivity</td>
<td>Tells if the algorithm uses low or high connectivity.</td>
</tr>
<tr>
<td>return_path</td>
<td>An image containing the nodes of the minimal path.</td>
</tr>
</tbody>
</table>
8.1.4.381  distance_eikonal() [1/2]

`CImg<T>& distance_eikonal`

- `const T & value,`
- `const CImg< t > & metric` 

Compute distance map to one source point, according to a custom metric (use fast marching algorithm).

**Parameters**

<table>
<thead>
<tr>
<th>value</th>
<th>Reference value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>metric</td>
<td>Field of distance potentials.</td>
</tr>
</tbody>
</table>

8.1.4.382  distance_eikonal() [2/2]

`CImg<T>& distance_eikonal`

- `const unsigned int nb_iterations,`
- `const float band_size = 0,`
- `const float time_step = 0.5f` 

Compute distance function to 0-valued isophotes, using the Eikonal PDE.

**Parameters**

<table>
<thead>
<tr>
<th>nb_iterations</th>
<th>Number of PDE iterations.</th>
</tr>
</thead>
<tbody>
<tr>
<td>band_size</td>
<td>Size of the narrow band.</td>
</tr>
<tr>
<td>time_step</td>
<td>Time step of the PDE iterations.</td>
</tr>
</tbody>
</table>

8.1.4.383  haar() [1/2]

`CImg<T>& haar`

- `const char axis,`
- `const bool invert = false,`
- `const unsigned int nb_scales = 1` 

Compute Haar multiscale wavelet transform.

**Parameters**

<table>
<thead>
<tr>
<th>axis</th>
<th>Axis considered for the transform.</th>
</tr>
</thead>
<tbody>
<tr>
<td>invert</td>
<td>Set inverse of direct transform.</td>
</tr>
<tr>
<td>nb_scales</td>
<td>Number of scales used for the transform.</td>
</tr>
</tbody>
</table>
8.1.4.384  haar() [2/2]

CImg<T> & haar (  
  const bool invert = false,  
  const unsigned int nb_scales = 1 )

Compute Haar multiscale wavelet transform [overloading].

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invert</td>
<td>Set inverse of direct transform.</td>
</tr>
<tr>
<td>nb_scales</td>
<td>Number of scales used for the transform.</td>
</tr>
</tbody>
</table>

8.1.4.385  get_FFT()

CImgList<Tfloat> get_FFT (  
  const char axis,  
  const bool is_inverse = false ) const

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axis</td>
<td>Axis along which the FFT is computed.</td>
</tr>
<tr>
<td>is_inverse</td>
<td>Tells if the forward (false) or inverse (true) FFT is computed.</td>
</tr>
</tbody>
</table>

8.1.4.386  FFT() [1/2]

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

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in, out</td>
<td>real</td>
</tr>
<tr>
<td>in, out</td>
<td>imag</td>
</tr>
<tr>
<td>axis</td>
<td>Axis along which the FFT is computed.</td>
</tr>
<tr>
<td>is_inverse</td>
<td>Tells if the forward (false) or inverse (true) FFT is computed.</td>
</tr>
</tbody>
</table>
8.1.4.387  FFT()  [2/2]

```cpp
static void FFT (
    CImg< T > & real,
    CImg< T > & imag,
    const bool is_inverse = false,
    const unsigned int nb_threads = 0 ) [static]
```

Compute n-D Fast Fourier Transform.

**Parameters**

<table>
<thead>
<tr>
<th>in, out</th>
<th>real</th>
<th>Real part of the pixel values.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in, out</td>
<td>imag</td>
<td>Imaginary part of the pixel values.</td>
</tr>
<tr>
<td></td>
<td>is_inverse</td>
<td>Tells if the forward (false) or inverse (true) FFT is computed.</td>
</tr>
<tr>
<td></td>
<td>nb_threads</td>
<td>Number of parallel threads used for the computation. Use 0 to set this to the number of available cpus.</td>
</tr>
</tbody>
</table>

8.1.4.388  shift_object3d()  [1/2]

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

Shift 3D object's vertices.

**Parameters**

| tx | X-coordinate of the 3D displacement vector. |
| ty | Y-coordinate of the 3D displacement vector. |
| tz | Z-coordinate of the 3D displacement vector. |

8.1.4.389  shift_object3d()  [2/2]

```cpp
CImg< T > & shift_object3d ( )
```

Shift 3D object's vertices, so that it becomes centered.

**Note**

The object center is computed as its barycenter.
8.1.4.390 resize_object3d()

```cpp
CImg<T> & resize_object3d(
    const float sx,
    const float sy = -100,
    const float sz = -100 )
```

Resize 3D object.

Parameters

<table>
<thead>
<tr>
<th>sx</th>
<th>Width of the 3D object's bounding box.</th>
</tr>
</thead>
<tbody>
<tr>
<td>sy</td>
<td>Height of the 3D object's bounding box.</td>
</tr>
<tr>
<td>sz</td>
<td>Depth of the 3D object's bounding box.</td>
</tr>
</tbody>
</table>

8.1.4.391 append_object3d()

```cpp
CImg<T> & append_object3d (
    CImgList<tf> & primitives,
    const CImg<tp> & obj_vertices,
    const CImgList<tff> & obj_primitives )
```

Merge two 3D objects together.

Parameters

<table>
<thead>
<tr>
<th>in,out</th>
<th>primitives</th>
<th>Primitives data of the current 3D object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj_vertices</td>
<td>Vertices data of the additional 3D object.</td>
<td></td>
</tr>
<tr>
<td>obj_primitives</td>
<td>Primitives data of the additional 3D object.</td>
<td></td>
</tr>
</tbody>
</table>

8.1.4.392 texturize_object3d()

```cpp
const CImg<T> & texturize_object3d (
    CImgList<tp> & primitives,
    CImgList<tc> & colors,
    const CImg<tt> & texture,
    const CImg<tx> & coords = CImg<tx>::const_empty() ) const
```

Texturize primitives of a 3D object.

Parameters

<table>
<thead>
<tr>
<th>in,out</th>
<th>primitives</th>
<th>Primitives data of the 3D object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out</td>
<td>colors</td>
<td>Colors data of the 3D object.</td>
</tr>
<tr>
<td></td>
<td>texture</td>
<td>Texture image to map to 3D object.</td>
</tr>
<tr>
<td></td>
<td>coords</td>
<td>Texture-mapping coordinates.</td>
</tr>
</tbody>
</table>
8.1.4.393 get_elevation3d()

```cpp
CImg<floatT> get_elevation3d (
    CImgList< tf > & primitives,
    CImgList< tc > & colors,
    const CImg< te > & elevation ) const
```

Generate a 3D elevation of the image instance.

**Parameters**

<table>
<thead>
<tr>
<th>out</th>
<th>primitives</th>
<th>The returned list of the 3D object primitives (template type tf should be at least unsigned int).</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>colors</td>
<td>The returned list of the 3D object colors.</td>
</tr>
<tr>
<td></td>
<td>elevation</td>
<td>The input elevation map.</td>
</tr>
</tbody>
</table>

**Returns**

The N vertices \((x_i,y_i,z_i)\) of the 3D object as a Nx3 CImg<float> image \((0 \leq i \leq N - 1)\).

**Example**

```cpp
const CImg<float> img("reference.jpg");
CImgList<unsigned int> faces3d;
CImgList<unsigned char> colors3d;
const CImg<float> points3d = img.get_elevation3d(faces3d,colors3d,img.get_norm()*0.2);
CImg<unsigned char>().display_object3d("Elevation3d",points3d,faces3d,colors3d);
```

8.1.4.394 get_projections3d()

```cpp
CImg<floatT> get_projections3d (  
    CImgList< tf > & primitives,
    CImgList< tc > & colors,
    const unsigned int x0,
    const unsigned int y0,
    const unsigned int z0,
    const bool normalize_colors = false ) const
```

Generate the 3D projection planes of the image instance.

**Parameters**

<table>
<thead>
<tr>
<th>out</th>
<th>primitives</th>
<th>Primitives data of the returned 3D object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>colors</td>
<td>Colors data of the returned 3D object.</td>
</tr>
<tr>
<td></td>
<td>x0</td>
<td>X-coordinate of the projection point.</td>
</tr>
<tr>
<td></td>
<td>y0</td>
<td>Y-coordinate of the projection point.</td>
</tr>
<tr>
<td></td>
<td>z0</td>
<td>Z-coordinate of the projection point.</td>
</tr>
<tr>
<td></td>
<td>normalize_colors</td>
<td>Tells if the created textures have normalized colors.</td>
</tr>
</tbody>
</table>

Generated by Doxygen
8.1.4.395  get_isoline3d()

```cpp
CImg<floatT> get_isoline3d ( 
    CImgList< tf > & primitives,
    const float  isovalue,
    const int    size_x = -100,
    const int    size_y = -100 ) const
```

Generate a isoline of the image instance as a 3D object.

**Parameters**

<table>
<thead>
<tr>
<th>out</th>
<th>primitives</th>
<th>The returned list of the 3D object primitives (template type tf should be at least unsigned int).</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>isovalue</td>
<td>The returned list of the 3D object colors.</td>
</tr>
<tr>
<td></td>
<td>size_x</td>
<td>The number of subdivisions along the X-axis.</td>
</tr>
<tr>
<td></td>
<td>size_y</td>
<td>The number of subdivisions along the Y-axis.</td>
</tr>
</tbody>
</table>

**Returns**

The N vertices (xi,yi,zi) of the 3D object as a Nx3 CImg<float> image (0<i<N - 1).

**Example**

```cpp
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);
```

8.1.4.396  get_isosurface3d()

```cpp
CImg<floatT> get_isosurface3d ( 
    CImgList< tf > & primitives,
    const float  isovalue,
    const int    size_x = -100,
    const int    size_y = -100,
    const int    size_z = -100 ) const
```

Generate an isosurface of the image instance as a 3D object.

**Parameters**

<table>
<thead>
<tr>
<th>out</th>
<th>primitives</th>
<th>The returned list of the 3D object primitives (template type tf should be at least unsigned int).</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>isovalue</td>
<td>The returned list of the 3D object colors.</td>
</tr>
<tr>
<td></td>
<td>size_x</td>
<td>Number of subdivisions along the X-axis.</td>
</tr>
<tr>
<td></td>
<td>size_y</td>
<td>Number of subdivisions along the Y-axis.</td>
</tr>
<tr>
<td></td>
<td>size_z</td>
<td>Number of subdivisions along the Z-axis.</td>
</tr>
</tbody>
</table>
Returns

The N vertices (xi, yi, zi) of the 3D object as a Nx3 CImg<float> image (0 ≤ i ≤ N - 1).

Example

```cpp
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);,
```

8.1.4.397 elevation3d()

```cpp
static CImg<floatT> elevation3d {
    CImgList< tf > & primitives,
    const tfunc & func,
    const float x0,
    const float y0,
    const float x1,
    const float y1,
    const int size_x = 256,
    const int size_y = 256 } [static]
```

Compute 3D elevation of a function as a 3D object.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>primitives</th>
<th>Primitives data of the resulting 3D object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>func</td>
<td>Elevation function. Is of type float (*func)(const float x, const float y).</td>
<td></td>
</tr>
<tr>
<td>x0</td>
<td>X-coordinate of the starting point.</td>
<td></td>
</tr>
<tr>
<td>y0</td>
<td>Y-coordinate of the starting point.</td>
<td></td>
</tr>
<tr>
<td>x1</td>
<td>X-coordinate of the ending point.</td>
<td></td>
</tr>
<tr>
<td>y1</td>
<td>Y-coordinate of the ending point.</td>
<td></td>
</tr>
<tr>
<td>size_x</td>
<td>Resolution of the function along the X-axis.</td>
<td></td>
</tr>
<tr>
<td>size_y</td>
<td>Resolution of the function along the Y-axis.</td>
<td></td>
</tr>
</tbody>
</table>

8.1.4.398 isoline3d()

```cpp
static CImg<floatT> isoline3d {
    CImgList< tf > & primitives,
    const tfunc & func,
    const float isovalue,
    const float x0,
    const float y0,
    const float x1,
    const float y1,
    const int size_x = 256,
    const int size_y = 256 } [static]
```

Compute 0-isolines of a function, as a 3D object.
Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>primitives</th>
<th>Primitives data of the resulting 3D object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>func</td>
<td>Elevation function. Is of type <code>float (*)(float x, float y)</code>.</td>
<td></td>
</tr>
<tr>
<td>isovalue</td>
<td>Isovalue to extract from function.</td>
<td></td>
</tr>
<tr>
<td>x0</td>
<td>X-coordinate of the starting point.</td>
<td></td>
</tr>
<tr>
<td>y0</td>
<td>Y-coordinate of the starting point.</td>
<td></td>
</tr>
<tr>
<td>x1</td>
<td>X-coordinate of the ending point.</td>
<td></td>
</tr>
<tr>
<td>y1</td>
<td>Y-coordinate of the ending point.</td>
<td></td>
</tr>
<tr>
<td>size_x</td>
<td>Resolution of the function along the X-axis.</td>
<td></td>
</tr>
<tr>
<td>size_y</td>
<td>Resolution of the function along the Y-axis.</td>
<td></td>
</tr>
</tbody>
</table>

Note

Use the marching squares algorithm for extracting the isolines.

8.1.4.399 isosurface3d()

```cpp
static CImg< floatT > isosurface3d (  
    CImgList< tf > & primitives,  
    const tfunc & func,  
    const float isovalue,  
    const float x0,  
    const float y0,  
    const float z0,  
    const float x1,  
    const float y1,  
    const float z1,  
    const int size_x = 32,  
    const int size_y = 32,  
    const int size_z = 32 ) [static]
```

Compute isosurface of a function, as a 3D object.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>primitives</th>
<th>Primitives data of the resulting 3D object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>func</td>
<td>Implicit function. Is of type <code>float (*)(float x, float y, float z)</code>.</td>
<td></td>
</tr>
<tr>
<td>isovalue</td>
<td>Isovalue to extract.</td>
<td></td>
</tr>
<tr>
<td>x0</td>
<td>X-coordinate of the starting point.</td>
<td></td>
</tr>
<tr>
<td>y0</td>
<td>Y-coordinate of the starting point.</td>
<td></td>
</tr>
<tr>
<td>z0</td>
<td>Z-coordinate of the starting point.</td>
<td></td>
</tr>
<tr>
<td>x1</td>
<td>X-coordinate of the ending point.</td>
<td></td>
</tr>
<tr>
<td>y1</td>
<td>Y-coordinate of the ending point.</td>
<td></td>
</tr>
<tr>
<td>z1</td>
<td>Z-coordinate of the ending point.</td>
<td></td>
</tr>
<tr>
<td>size_x</td>
<td>Resolution of the elevation function along the X-axis.</td>
<td></td>
</tr>
<tr>
<td>size_y</td>
<td>Resolution of the elevation function along the Y-axis.</td>
<td></td>
</tr>
<tr>
<td>size_z</td>
<td>Resolution of the elevation function along the Z-axis.</td>
<td></td>
</tr>
</tbody>
</table>
Use the marching cubes algorithm for extracting the isosurface.

8.1.4.400 box3d()

```cpp
static CImg<
  floatT
>
box3d ( CImgList<
  tf
>&
primitives,
const float
size_x = 200,
const float
size_y = 100,
const float
size_z = 100
) [static]
```

Generate a 3D box object.

**Parameters**

<table>
<thead>
<tr>
<th>out</th>
<th>primitives</th>
<th>The returned list of the 3D object primitives (template type tf should be at least unsigned int).</th>
</tr>
</thead>
<tbody>
<tr>
<td>size_x</td>
<td>The width of the box (dimension along the X-axis).</td>
<td></td>
</tr>
<tr>
<td>size_y</td>
<td>The height of the box (dimension along the Y-axis).</td>
<td></td>
</tr>
<tr>
<td>size_z</td>
<td>The depth of the box (dimension along the Z-axis).</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

The N vertices (xi,yi,zi) of the 3D object as a Nx3 CImg<float> image (0≤i≤N - 1).

**Example**

```cpp
CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::box3d(faces3d,10,20,30);
CImg<unsigned char>().display_object3d("Box3d",points3d,faces3d);
```

8.1.4.401 cone3d()

```cpp
static CImg<
  floatT
>
cone3d ( CImgList<
  tf
>&
primitives,
const float
radius = 50,
const float
size_z = 100,
const unsigned int
subdivisions = 24
) [static]
```

Generate a 3D cone.

**Parameters**

<table>
<thead>
<tr>
<th>out</th>
<th>primitives</th>
<th>The returned list of the 3D object primitives (template type tf should be at least unsigned int).</th>
</tr>
</thead>
<tbody>
<tr>
<td>radius</td>
<td>The radius of the cone basis.</td>
<td></td>
</tr>
<tr>
<td>size_z</td>
<td>The cone's height.</td>
<td></td>
</tr>
<tr>
<td>subdivisions</td>
<td>The number of basis angular subdivisions.</td>
<td></td>
</tr>
</tbody>
</table>
8.1 CImg< T > Struct Template Reference

Returns

The N vertices \((x_i, y_i, z_i)\) of the 3D object as a \(N\times3\) CImg\(<float>\) image \((0 \leq i < N - 1)\).

Example

```cpp
CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::cone3d(faces3d, 50);
CImg<unsigned char>().display_object3d("Cone3d", points3d, faces3d);
```

8.1.4.402 cylinder3d()

static CImg< floatT > cylinder3d (  
CImgList< tf > & primitives,
const float radius = 50,
const float size_z = 100,
const unsigned int subdivisions = 24 ) [static]

Generate a 3D cylinder.

Parameters

<table>
<thead>
<tr>
<th>out primitives</th>
<th>The returned list of the 3D object primitives (template type (tf) should be at least (unsigned int)).</th>
</tr>
</thead>
<tbody>
<tr>
<td>radius</td>
<td>The radius of the cylinder basis.</td>
</tr>
<tr>
<td>size_z</td>
<td>The cylinder's height.</td>
</tr>
<tr>
<td>subdivisions</td>
<td>The number of basis angular subdivisions.</td>
</tr>
</tbody>
</table>

Returns

The N vertices \((x_i, y_i, z_i)\) of the 3D object as a \(N\times3\) CImg\(<float>\) image \((0 \leq i < N - 1)\).

Example

```cpp
CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::cylinder3d(faces3d, 50);
CImg<unsigned char>().display_object3d("Cylinder3d", points3d, faces3d);
```

8.1.4.403 torus3d()

static CImg< floatT > torus3d (  
CImgList< tf > & primitives,
const float radius1 = 100,
const float radius2 = 30,
const unsigned int subdivisions1 = 24,
const unsigned int subdivisions2 = 12 ) [static]

Generate a 3D torus.
Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>primitives</th>
<th>The returned list of the 3D object primitives (template type (tf) should be at least (unsigned\ \text{int})).</th>
</tr>
</thead>
<tbody>
<tr>
<td>radius1</td>
<td>The large radius.</td>
<td></td>
</tr>
<tr>
<td>radius2</td>
<td>The small radius.</td>
<td></td>
</tr>
<tr>
<td>subdivisions1</td>
<td>The number of angular subdivisions for the large radius.</td>
<td></td>
</tr>
<tr>
<td>subdivisions2</td>
<td>The number of angular subdivisions for the small radius.</td>
<td></td>
</tr>
</tbody>
</table>

Returns

The \(N\) vertices \((x_i, y_i, z_i)\) of the 3D object as a \(N\times3\) \(CImg<float>\) image \((0 \leq i \leq N - 1)\).

Example

```cpp
CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::torus3d(faces3d, 20, 4);
CImg<unsigned char>().display_object3d("Torus3d", points3d, faces3d);
```

8.1.4.404 plane3d()

```cpp
static CImg<floatT> plane3d ( 
    CImgList<tf> & primitives, 
    const float size_x = 100, 
    const float size_y = 100, 
    const unsigned int subdivisions_x = 10, 
    const unsigned int subdivisions_y = 10 ) [static]
```

Generate a 3D XY-plane.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>primitives</th>
<th>The returned list of the 3D object primitives (template type (tf) should be at least (unsigned\ \text{int})).</th>
</tr>
</thead>
<tbody>
<tr>
<td>size_x</td>
<td>The width of the plane (dimension along the X-axis).</td>
<td></td>
</tr>
<tr>
<td>size_y</td>
<td>The height of the plane (dimensions along the Y-axis).</td>
<td></td>
</tr>
<tr>
<td>subdivisions_x</td>
<td>The number of planar subdivisions along the X-axis.</td>
<td></td>
</tr>
<tr>
<td>subdivisions_y</td>
<td>The number of planar subdivisions along the Y-axis.</td>
<td></td>
</tr>
</tbody>
</table>

Returns

The \(N\) vertices \((x_i, y_i, z_i)\) of the 3D object as a \(N\times3\) \(CImg<float>\) image \((0 \leq i \leq N - 1)\).

Example

```cpp
CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::plane3d(faces3d, 100, 50);
CImg<unsigned char>().display_object3d("Plane3d", points3d, faces3d);
```
8.1.4.405 sphere3d()

static CImg<floatT> sphere3d (  
    CImgList< tf > & primitives,  
    const float radius = 50,  
    const unsigned int subdivisions = 3 ) [static]

Generate a 3D sphere.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>primitives</th>
<th>The returned list of the 3D object primitives (template type tf should be at least unsigned int).</th>
</tr>
</thead>
<tbody>
<tr>
<td>radius</td>
<td>The radius of the sphere (dimension along the X-axis).</td>
<td></td>
</tr>
<tr>
<td>subdivisions</td>
<td>The number of recursive subdivisions from an initial icosahedron.</td>
<td></td>
</tr>
</tbody>
</table>

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);

8.1.4.406 ellipsoid3d()

static CImg<floatT> ellipsoid3d (  
    CImgList< tf > & primitives,  
    const CImg< t > & tensor,  
    const unsigned int subdivisions = 3 ) [static]

Generate a 3D ellipsoid.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>primitives</th>
<th>The returned list of the 3D object primitives (template type tf should be at least unsigned int).</th>
</tr>
</thead>
<tbody>
<tr>
<td>tensor</td>
<td>The tensor which gives the shape and size of the ellipsoid.</td>
<td></td>
</tr>
<tr>
<td>subdivisions</td>
<td>The number of recursive subdivisions from an initial stretched icosahedron.</td>
<td></td>
</tr>
</tbody>
</table>

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);
8.1.4.407  object3dtoCImg3d()

\[ \text{CImg} < T > & \text{object3dtoCImg3d} \{
\text{const CImgList} < \text{tp} > & \text{primitives},
\text{const CImgList} < \text{tc} > & \text{colors},
\text{const to} & \text{opacities},
\text{const bool full_check} = \text{true} \}
\]

Convert 3D object into a CImg3d representation.

**Parameters**

| primitives | Primitives data of the 3D object. |
| colors     | Colors data of the 3D object.     |
| opacities  | Opacities data of the 3D object.  |
| full_check | Tells if full checking of the 3D object must be performed. |

8.1.4.408  CImg3dtoobject3d()

\[ \text{CImg} < T > & \text{CImg3dtoobject3d} \{
\text{CImgList} < \text{tp} > & \text{primitives},
\text{CImgList} < \text{tc} > & \text{colors},
\text{CImgList} < \text{to} > & \text{opacities},
\text{const bool full_check} = \text{true} \}
\]

Convert CImg3d representation into a 3D object.

**Parameters**

| out primitives | Primitives data of the 3D object. |
| out colors     | Colors data of the 3D object.     |
| out opacities  | Opacities data of the 3D object.  |
| full_check     | Tells if full checking of the 3D object must be performed. |

8.1.4.409  draw_point()[1/2]

\[ \text{CImg} < T > & \text{draw_point} \{
\text{const int x0,}
\text{const int y0,}
\text{const int z0,}
\text{const tc *const color,}
\text{const float opacity} = \text{1} \}
\]

Draw a 3D point.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>X-coordinate of the point.</td>
</tr>
<tr>
<td>y0</td>
<td>Y-coordinate of the point.</td>
</tr>
<tr>
<td>z0</td>
<td>Z-coordinate of the point.</td>
</tr>
<tr>
<td>color</td>
<td>Pointer to <code>spectrum()</code> consecutive values, defining the drawing color.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

Note

- To set pixel values without clipping needs, you should use the faster `CImg::operator()()` function.

Example:

```cpp
CImg<unsigned char> img(100,100,1,3,0);
const unsigned char color[3] = { 255,128,64 };
img.draw_point(50,50,color);
```

8.1.4.410 **draw_point()** [2/2]

```cpp
CImg<T>& draw_point (const CImg<T>& points, const tc*color, const float opacity = 1)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>points</td>
<td>Image of vertices coordinates.</td>
</tr>
<tr>
<td>color</td>
<td>Pointer to <code>spectrum()</code> consecutive values, defining the drawing color.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

8.1.4.411 **draw_line()** [1/6]

```cpp
CImg<T>& draw_line (int x0, int y0, int x1, int y1, const tc*const color, const float opacity = 1, const unsigned int pattern = ~0U, const bool init_hatch = true)
```

Draw a 2D line.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>X-coordinate of the starting line point.</td>
</tr>
<tr>
<td>y0</td>
<td>Y-coordinate of the starting line point.</td>
</tr>
<tr>
<td>x1</td>
<td>X-coordinate of the ending line point.</td>
</tr>
<tr>
<td>y1</td>
<td>Y-coordinate of the ending line point.</td>
</tr>
<tr>
<td>color</td>
<td>Pointer to <code>spectrum()</code> consecutive values of type <code>T</code>, defining the drawing color.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
<tr>
<td>pattern</td>
<td>An integer whose bits describe the line pattern.</td>
</tr>
<tr>
<td>init_hatch</td>
<td>Tells if a reinitialization of the hash state must be done.</td>
</tr>
</tbody>
</table>

Note

- Line routine uses Bresenham's algorithm.
- Set `init_hatch = false` to draw consecutive hatched segments without breaking the line pattern.

Example:

```cpp
CImg<unsigned char> img(100,100,1,3,0);
const unsigned char color[] = { 255,128,64 };
img.draw_line(40,40,80,70,color);
```

8.1.4.412 draw_line() [2/6]

```cpp
CImg<T>& draw_line ( 
    CImg< tz > & zbuffer,
    int x0,
    int y0,
    const float z0,
    int x1,
    int y1,
    const float z1,
    const tc *const color,
    const float opacity = 1,
    const unsigned int pattern = ~0U,
    const bool init_hatch = true )
```

Draw a 2D line, with z-buffering.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>zbuffer</td>
<td>Zbuffer image.</td>
</tr>
<tr>
<td>x0</td>
<td>X-coordinate of the starting point.</td>
</tr>
<tr>
<td>y0</td>
<td>Y-coordinate of the starting point.</td>
</tr>
<tr>
<td>z0</td>
<td>Z-coordinate of the starting point</td>
</tr>
<tr>
<td>x1</td>
<td>X-coordinate of the ending point.</td>
</tr>
<tr>
<td>y1</td>
<td>Y-coordinate of the ending point.</td>
</tr>
<tr>
<td>z1</td>
<td>Z-coordinate of the ending point.</td>
</tr>
<tr>
<td>color</td>
<td>Pointer to <code>spectrum()</code> consecutive values of type <code>T</code>, defining the drawing color.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
<tr>
<td>pattern</td>
<td>An integer whose bits describe the line pattern.</td>
</tr>
<tr>
<td>init_hatch</td>
<td>Tells if a reinitialization of the hash state must be done.</td>
</tr>
</tbody>
</table>
8.1.4.413 draw_line() [3/6]

CImg<T>& draw_line {
    int x0,
    int y0,
    int x1,
    int y1,
    const CImg<tc>& texture,
    int tx0,
    int ty0,
    int tx1,
    int ty1,
    const float opacity = 1,
    const unsigned int pattern = ~0U,
    const bool init_hatch = true
}

Draw a textured 2D line.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>X-coordinate of the starting line point.</td>
</tr>
<tr>
<td>y0</td>
<td>Y-coordinate of the starting line point.</td>
</tr>
<tr>
<td>x1</td>
<td>X-coordinate of the ending line point.</td>
</tr>
<tr>
<td>y1</td>
<td>Y-coordinate of the ending line point.</td>
</tr>
<tr>
<td>texture</td>
<td>Texture image defining the pixel colors.</td>
</tr>
<tr>
<td>tx0</td>
<td>X-coordinate of the starting texture point.</td>
</tr>
<tr>
<td>ty0</td>
<td>Y-coordinate of the starting texture point.</td>
</tr>
<tr>
<td>tx1</td>
<td>X-coordinate of the ending texture point.</td>
</tr>
<tr>
<td>ty1</td>
<td>Y-coordinate of the ending texture point.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
<tr>
<td>pattern</td>
<td>An integer whose bits describe the line pattern.</td>
</tr>
<tr>
<td>init_hatch</td>
<td>Tells if the hash variable must be reinitialized.</td>
</tr>
</tbody>
</table>

Note

- Line routine uses the well known Bresenham's algorithm.

Example:

```cpp
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);
```

8.1.4.414 draw_line() [4/6]

CImg<T>& draw_line {
    int x0,
```
int y0,
const float z0,
int x1,
int y1,
const float z1,
cost CImg< tc > & texture,
const int tx0,
const int ty0,
const int tx1,
const int ty1,
const float opacity = 1,
const unsigned int pattern = ~0U,
const bool init_hatch = true )

Draw a textured 2D line, with perspective correction.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>X-coordinate of the starting point.</td>
</tr>
<tr>
<td>y0</td>
<td>Y-coordinate of the starting point.</td>
</tr>
<tr>
<td>z0</td>
<td>Z-coordinate of the starting point.</td>
</tr>
<tr>
<td>x1</td>
<td>X-coordinate of the ending point.</td>
</tr>
<tr>
<td>y1</td>
<td>Y-coordinate of the ending point.</td>
</tr>
<tr>
<td>z1</td>
<td>Z-coordinate of the ending point.</td>
</tr>
<tr>
<td>texture</td>
<td>Texture image defining the pixel colors.</td>
</tr>
<tr>
<td>tx0</td>
<td>X-coordinate of the starting texture point.</td>
</tr>
<tr>
<td>ty0</td>
<td>Y-coordinate of the starting texture point.</td>
</tr>
<tr>
<td>tx1</td>
<td>X-coordinate of the ending texture point.</td>
</tr>
<tr>
<td>ty1</td>
<td>Y-coordinate of the ending texture point.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
<tr>
<td>pattern</td>
<td>An integer whose bits describe the line pattern.</td>
</tr>
<tr>
<td>init_hatch</td>
<td>Tells if the hash variable must be reinitialized.</td>
</tr>
</tbody>
</table>

8.1.4.415 draw_line() [5/6]

CImg<T>& draw_line (  
    CImg< tz > & zbuffer,
    int x0,
    int y0,
    const float z0,
    int x1,
    int y1,
    const float z1,
    const CImg< tc > & texture,
    const int tx0,
    const int ty0,
    const int tx1,
    const int ty1,
    const float opacity = 1,
    const unsigned int pattern = ~0U,
    const bool init_hatch = true )

Generated by Doxygen
Draw a textured 2D line, with perspective correction and z-buffering.
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>zbuffer</code></td>
<td>Z-buffer image.</td>
</tr>
<tr>
<td><code>x0</code></td>
<td>X-coordinate of the starting point.</td>
</tr>
<tr>
<td><code>y0</code></td>
<td>Y-coordinate of the starting point.</td>
</tr>
<tr>
<td><code>z0</code></td>
<td>Z-coordinate of the starting point.</td>
</tr>
<tr>
<td><code>x1</code></td>
<td>X-coordinate of the ending point.</td>
</tr>
<tr>
<td><code>y1</code></td>
<td>Y-coordinate of the ending point.</td>
</tr>
<tr>
<td><code>z1</code></td>
<td>Z-coordinate of the ending point.</td>
</tr>
<tr>
<td><code>texture</code></td>
<td>Texture image defining the pixel colors.</td>
</tr>
<tr>
<td><code>tx0</code></td>
<td>X-coordinate of the starting texture point.</td>
</tr>
<tr>
<td><code>ty0</code></td>
<td>Y-coordinate of the starting texture point.</td>
</tr>
<tr>
<td><code>tx1</code></td>
<td>X-coordinate of the ending texture point.</td>
</tr>
<tr>
<td><code>ty1</code></td>
<td>Y-coordinate of the ending texture point.</td>
</tr>
<tr>
<td><code>opacity</code></td>
<td>Drawing opacity.</td>
</tr>
<tr>
<td><code>pattern</code></td>
<td>An integer whose bits describe the line pattern.</td>
</tr>
<tr>
<td><code>init_hatch</code></td>
<td>Tells if the hash variable must be reinitialized.</td>
</tr>
</tbody>
</table>

#### 8.1.4.416 draw_line()

```cpp
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 lines.

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>points</code></td>
<td>Coordinates of vertices, stored as a list of vectors.</td>
</tr>
<tr>
<td><code>color</code></td>
<td>Pointer to spectrum() consecutive values of type T, defining the drawing color.</td>
</tr>
<tr>
<td><code>opacity</code></td>
<td>Drawing opacity.</td>
</tr>
<tr>
<td><code>pattern</code></td>
<td>An integer whose bits describe the line pattern.</td>
</tr>
<tr>
<td><code>init_hatch</code></td>
<td>If set to true, init hatch motif.</td>
</tr>
</tbody>
</table>

#### Note

- This function uses several call to the single CImg::draw_line() procedure, depending on the vectors size in `points`.

#### 8.1.4.417 draw_arrow()

```cpp
CImg<
  T> & draw_arrow(
    const int x0,
```

Generated by Doxygen
const int y0,
const int x1,
const int y1,
const tc const color,
const float opacity = 1,
const float angle = 30,
const float length = -10,
const unsigned int pattern = ~0U)

Draw a 2D arrow.

Parameters

<table>
<thead>
<tr>
<th>x0</th>
<th>X-coordinate of the starting arrow point (tail).</th>
</tr>
</thead>
<tbody>
<tr>
<td>y0</td>
<td>Y-coordinate of the starting arrow point (tail).</td>
</tr>
<tr>
<td>x1</td>
<td>X-coordinate of the ending arrow point (head).</td>
</tr>
<tr>
<td>y1</td>
<td>Y-coordinate of the ending arrow point (head).</td>
</tr>
<tr>
<td>color</td>
<td>Pointer to \texttt{spectrum()} consecutive values of type T, defining the drawing color.</td>
</tr>
<tr>
<td>angle</td>
<td>Aperture angle of the arrow head.</td>
</tr>
<tr>
<td>length</td>
<td>Length of the arrow head. If negative, describes a percentage of the arrow length.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
<tr>
<td>pattern</td>
<td>An integer whose bits describe the line pattern.</td>
</tr>
</tbody>
</table>

8.1.4.418 \texttt{draw_spline()} [1/4]

\texttt{CImg}<const T&> \texttt{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 2D spline.

Parameters

<table>
<thead>
<tr>
<th>x0</th>
<th>X-coordinate of the starting curve point</th>
</tr>
</thead>
<tbody>
<tr>
<td>y0</td>
<td>Y-coordinate of the starting curve point</td>
</tr>
<tr>
<td>u0</td>
<td>X-coordinate of the starting velocity</td>
</tr>
<tr>
<td>v0</td>
<td>Y-coordinate of the starting velocity</td>
</tr>
<tr>
<td>x1</td>
<td>X-coordinate of the ending curve point</td>
</tr>
<tr>
<td>y1</td>
<td>Y-coordinate of the ending curve point</td>
</tr>
</tbody>
</table>
## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>X-coordinate of the ending velocity</td>
</tr>
<tr>
<td>$v_1$</td>
<td>Y-coordinate of the ending velocity</td>
</tr>
<tr>
<td><code>color</code></td>
<td>Pointer to <code>spectrum()</code> consecutive values of type $T$, defining the drawing color.</td>
</tr>
<tr>
<td><code>precision</code></td>
<td>Curve drawing precision.</td>
</tr>
<tr>
<td><code>opacity</code></td>
<td>Drawing opacity.</td>
</tr>
<tr>
<td><code>pattern</code></td>
<td>An integer whose bits describe the line pattern.</td>
</tr>
<tr>
<td><code>init_hatch</code></td>
<td>If <code>true</code>, init hatch motif.</td>
</tr>
</tbody>
</table>

### Note

- The curve is a 2D cubic Bezier spline, from the set of specified starting/ending points and corresponding velocity vectors.
- The spline is drawn as a sequence of connected segments. The `precision` parameter sets the average number of pixels in each drawn segment.
- A cubic Bezier curve is sometimes defined by a set of 4 points \{ $(x_0, y_0), (x_a, y_a), (x_b, y_b), (x_1, y_1)$ \} where $(x_0, y_0)$ is the starting point, $(x_1, y_1)$ is the ending point and $(x_a, y_a), (x_b, y_b)$ are two control points. The starting and ending velocities $(u_0, v_0)$ and $(u_1, v_1)$ can be deduced easily from the control points as \[ u_0 = (x_a - x_0), v_0 = (y_a - y_0), u_1 = (x_1 - x_b) \text{ and } v_1 = (y_1 - y_b). \]

### Example:

```cpp
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);
```

---

### 8.1.4.419 draw_spline() [2/4]

```cpp
CImg<T>& draw_spline(
    const int x0,
    const int y0,
    const float u0,
    const float v0,
    const int x1,
    const int y1,
    const float u1,
    const float v1,
    const CImg< t > & texture,
    const int tx0,
    const int ty0,
    const int tx1,
    const int ty1,
    const float opacity = 1,
    const float precision = 4,
    const unsigned int pattern = ~0U,
    const bool init_hatch = true
)
```

Draw a textured 2D spline.
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>X-coordinate of the starting curve point</td>
</tr>
<tr>
<td>y0</td>
<td>Y-coordinate of the starting curve point</td>
</tr>
<tr>
<td>u0</td>
<td>X-coordinate of the starting velocity</td>
</tr>
<tr>
<td>v0</td>
<td>Y-coordinate of the starting velocity</td>
</tr>
<tr>
<td>x1</td>
<td>X-coordinate of the ending curve point</td>
</tr>
<tr>
<td>y1</td>
<td>Y-coordinate of the ending curve point</td>
</tr>
<tr>
<td>u1</td>
<td>X-coordinate of the ending velocity</td>
</tr>
<tr>
<td>v1</td>
<td>Y-coordinate of the ending velocity</td>
</tr>
<tr>
<td>texture</td>
<td>Texture image defining line pixel colors.</td>
</tr>
<tr>
<td>tx0</td>
<td>X-coordinate of the starting texture point.</td>
</tr>
<tr>
<td>ty0</td>
<td>Y-coordinate of the starting texture point.</td>
</tr>
<tr>
<td>tx1</td>
<td>X-coordinate of the ending texture point.</td>
</tr>
<tr>
<td>ty1</td>
<td>Y-coordinate of the ending texture point.</td>
</tr>
<tr>
<td>precision</td>
<td>Curve drawing precision.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
<tr>
<td>pattern</td>
<td>An integer whose bits describe the line pattern.</td>
</tr>
<tr>
<td>init_hatch</td>
<td>If true, reinit hatch motif.</td>
</tr>
</tbody>
</table>

### 8.1.4.420 draw_spline() [3/4]

```cpp
CImg<T>& draw_spline ( const CImg<tp>& points, const CImg<tt>& tangents, const tc* const color, const float opacity = 1, const bool is_closed_set = false, const float precision = 4, const unsigned int pattern = ~0U, const bool init_hatch = true )
```

Draw a set of consecutive splines.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>points</td>
<td>Vertices data.</td>
</tr>
<tr>
<td>tangents</td>
<td>Tangents data.</td>
</tr>
<tr>
<td>color</td>
<td>Pointer to spectrum() consecutive values of type T, defining the drawing color.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
<tr>
<td>is_closed_set</td>
<td>Tells if the drawn spline set is closed.</td>
</tr>
<tr>
<td>precision</td>
<td>Precision of the drawing.</td>
</tr>
<tr>
<td>pattern</td>
<td>An integer whose bits describe the line pattern.</td>
</tr>
<tr>
<td>init_hatch</td>
<td>If true, init hatch motif.</td>
</tr>
</tbody>
</table>
8.1.4.421 draw_spline() [4/4]

CImg<T>& draw_spline(
    const CImg<T> & points,
    const tc *const color,
    const float opacity = 1,
    const bool is_closed_set = false,
    const float precision = 4,
    const unsigned int pattern = ~0U,
    const bool init_hatch = true)

Draw a set of consecutive splines [overloading].
Similar to previous function, with the point tangents automatically estimated from the given points set.

8.1.4.422 draw_triangle() [1/9]

CImg<T>& draw_triangle(
    const int x0,
    const int y0,
    const int x1,
    const int y1,
    const int x2,
    const int y2,
    const tc *const color,
    const float opacity = 1)

Draw a filled 2D triangle.

Parameters

| x0   | X-coordinate of the first vertex. |
| y0   | Y-coordinate of the first vertex. |
| x1   | X-coordinate of the second vertex. |
| y1   | Y-coordinate of the second vertex. |
| x2   | X-coordinate of the third vertex. |
| y2   | Y-coordinate of the third vertex. |
| color| Pointer to spectrum() consecutive values of type T, defining the drawing color. |
| opacity | Drawing opacity. |

8.1.4.423 draw_triangle() [2/9]

CImg<T>& draw_triangle(
    const int x0,
    const int y0,
    const int x1,
    const int y1,
    const int x2,
    const int y2,
const tc *const color,
const float opacity,
const unsigned int pattern)

Draw a outlined 2D triangle.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>X-coordinate of the first vertex.</td>
</tr>
<tr>
<td>y0</td>
<td>Y-coordinate of the first vertex.</td>
</tr>
<tr>
<td>x1</td>
<td>X-coordinate of the second vertex.</td>
</tr>
<tr>
<td>y1</td>
<td>Y-coordinate of the second vertex.</td>
</tr>
<tr>
<td>x2</td>
<td>X-coordinate of the third vertex.</td>
</tr>
<tr>
<td>y2</td>
<td>Y-coordinate of the third vertex.</td>
</tr>
<tr>
<td>color</td>
<td>Pointer to spectrum() consecutive values of type T, defining the drawing color.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
<tr>
<td>pattern</td>
<td>An integer whose bits describe the outline pattern.</td>
</tr>
</tbody>
</table>

8.1.4.424 draw_triangle() [3/9]

CImg<T>& draw_triangle(
     CImg<tz>& zbuffer,
     int x0,
     int y0,
     const float z0,
     int x1,
     int y1,
     const float z1,
     int x2,
     int y2,
     const float z2,
     const tc *const color,
     const float opacity = 1,
     const float brightness = 1)

Draw a filled 2D triangle, with z-buffering.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>zbuffer</td>
<td>Z-buffer image.</td>
</tr>
<tr>
<td>x0</td>
<td>X-coordinate of the first vertex.</td>
</tr>
<tr>
<td>y0</td>
<td>Y-coordinate of the first vertex.</td>
</tr>
<tr>
<td>z0</td>
<td>Z-coordinate of the first vertex.</td>
</tr>
<tr>
<td>x1</td>
<td>X-coordinate of the second vertex.</td>
</tr>
<tr>
<td>y1</td>
<td>Y-coordinate of the second vertex.</td>
</tr>
<tr>
<td>z1</td>
<td>Z-coordinate of the second vertex.</td>
</tr>
<tr>
<td>x2</td>
<td>X-coordinate of the third vertex.</td>
</tr>
<tr>
<td>y2</td>
<td>Y-coordinate of the third vertex.</td>
</tr>
<tr>
<td>z2</td>
<td>Z-coordinate of the third vertex.</td>
</tr>
<tr>
<td>color</td>
<td>Pointer to spectrum() consecutive values of type T, defining the drawing color.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
<tr>
<td>brightness</td>
<td>Brightness factor.</td>
</tr>
</tbody>
</table>
8.1.4.425 draw_triangle() [4/9]

```
CImg<T>& draw_triangle (
    int x0,
    int y0,
    int x1,
    int y1,
    int x2,
    int y2,
    const tc *const color,
    float bs0,
    float bs1,
    float bs2,
    const float opacity = 1 )
```

Draw a Gouraud-shaded 2D triangle.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x0</code></td>
<td>X-coordinate of the first vertex in the image instance.</td>
</tr>
<tr>
<td><code>y0</code></td>
<td>Y-coordinate of the first vertex in the image instance.</td>
</tr>
<tr>
<td><code>x1</code></td>
<td>X-coordinate of the second vertex in the image instance.</td>
</tr>
<tr>
<td><code>y1</code></td>
<td>Y-coordinate of the second vertex in the image instance.</td>
</tr>
<tr>
<td><code>x2</code></td>
<td>X-coordinate of the third vertex in the image instance.</td>
</tr>
<tr>
<td><code>y2</code></td>
<td>Y-coordinate of the third vertex in the image instance.</td>
</tr>
<tr>
<td><code>color</code></td>
<td>Pointer to <code>spectrum()</code> consecutive values, defining the drawing color.</td>
</tr>
<tr>
<td><code>bs0</code></td>
<td>Brightness factor of the first vertex (in [0,2]).</td>
</tr>
<tr>
<td><code>bs1</code></td>
<td>Brightness factor of the second vertex (in [0,2]).</td>
</tr>
<tr>
<td><code>bs2</code></td>
<td>Brightness factor of the third vertex (in [0,2]).</td>
</tr>
<tr>
<td><code>opacity</code></td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

8.1.4.426 draw_triangle() [5/9]

```
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 color-interpolated 2D triangle.
8.1 CImg<T> Struct Template Reference

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>X-coordinate of the first vertex in the image instance.</td>
</tr>
<tr>
<td>y0</td>
<td>Y-coordinate of the first vertex in the image instance.</td>
</tr>
<tr>
<td>x1</td>
<td>X-coordinate of the second vertex in the image instance.</td>
</tr>
<tr>
<td>y1</td>
<td>Y-coordinate of the second vertex in the image instance.</td>
</tr>
<tr>
<td>x2</td>
<td>X-coordinate of the third vertex in the image instance.</td>
</tr>
<tr>
<td>y2</td>
<td>Y-coordinate of the third vertex in the image instance.</td>
</tr>
<tr>
<td>color1</td>
<td>Pointer to <code>spectrum()</code> consecutive values of type <code>T</code>, defining the color of the first vertex.</td>
</tr>
<tr>
<td>color2</td>
<td>Pointer to <code>spectrum()</code> consecutive values of type <code>T</code>, defining the color of the second vertex.</td>
</tr>
<tr>
<td>color3</td>
<td>Pointer to <code>spectrum()</code> consecutive values of type <code>T</code>, defining the color of the third vertex.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

### 8.1.4.427 draw_triangle() [6/9]

```cpp
CImg<T>& draw_triangle (
    int x0,
    int y0,
    int x1,
    int y1,
    int x2,
    int y2,
    const CImg<tc>& texture,
    int tx0,
    int ty0,
    int tx1,
    int ty1,
    int tx2,
    int ty2,
    const float opacity = 1,
    const float brightness = 1)
```

Draw a textured 2D triangle.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>X-coordinate of the first vertex in the image instance.</td>
</tr>
<tr>
<td>y0</td>
<td>Y-coordinate of the first vertex in the image instance.</td>
</tr>
<tr>
<td>x1</td>
<td>X-coordinate of the second vertex in the image instance.</td>
</tr>
<tr>
<td>y1</td>
<td>Y-coordinate of the second vertex in the image instance.</td>
</tr>
<tr>
<td>x2</td>
<td>X-coordinate of the third vertex in the image instance.</td>
</tr>
<tr>
<td>y2</td>
<td>Y-coordinate of the third vertex in the image instance.</td>
</tr>
<tr>
<td>texture</td>
<td>Texture image used to fill the triangle.</td>
</tr>
<tr>
<td>tx0</td>
<td>X-coordinate of the first vertex in the texture image.</td>
</tr>
<tr>
<td>ty0</td>
<td>Y-coordinate of the first vertex in the texture image.</td>
</tr>
<tr>
<td>tx1</td>
<td>X-coordinate of the second vertex in the texture image.</td>
</tr>
<tr>
<td>ty1</td>
<td>Y-coordinate of the second vertex in the texture image.</td>
</tr>
<tr>
<td>tx2</td>
<td>X-coordinate of the third vertex in the texture image.</td>
</tr>
<tr>
<td>ty2</td>
<td>Y-coordinate of the third vertex in the texture image.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
<tr>
<td>brightness</td>
<td>Brightness factor of the drawing (in [0,2]).</td>
</tr>
</tbody>
</table>
8.1.4.428  draw_triangle() [7/9]

CImg\<T\>\& draw_triangle (  
    int x0,  
    int y0,  
    int x1,  
    int y1,  
    int x2,  
    int y2,  
    const tc *const color,  
    const CImg< tl > \& light,  
    int lx0,  
    int ly0,  
    int lx1,  
    int ly1,  
    int lx2,  
    int ly2,  
    const float opacity = 1 )

Draw a Phong-shaded 2D triangle.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>X-coordinate of the first vertex in the image instance.</td>
</tr>
<tr>
<td>y0</td>
<td>Y-coordinate of the first vertex in the image instance.</td>
</tr>
<tr>
<td>x1</td>
<td>X-coordinate of the second vertex in the image instance.</td>
</tr>
<tr>
<td>y1</td>
<td>Y-coordinate of the second vertex in the image instance.</td>
</tr>
<tr>
<td>x2</td>
<td>X-coordinate of the third vertex in the image instance.</td>
</tr>
<tr>
<td>y2</td>
<td>Y-coordinate of the third vertex in the image instance.</td>
</tr>
<tr>
<td>color</td>
<td>Pointer to spectrum() consecutive values, defining the drawing color.</td>
</tr>
<tr>
<td>light</td>
<td>Light image.</td>
</tr>
<tr>
<td>lx0</td>
<td>X-coordinate of the first vertex in the light image.</td>
</tr>
<tr>
<td>ly0</td>
<td>Y-coordinate of the first vertex in the light image.</td>
</tr>
<tr>
<td>lx1</td>
<td>X-coordinate of the second vertex in the light image.</td>
</tr>
<tr>
<td>ly1</td>
<td>Y-coordinate of the second vertex in the light image.</td>
</tr>
<tr>
<td>lx2</td>
<td>X-coordinate of the third vertex in the light image.</td>
</tr>
<tr>
<td>ly2</td>
<td>Y-coordinate of the third vertex in the light image.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

8.1.4.429  draw_triangle() [8/9]

CImg\<T\>\& draw_triangle (  
    int x0,  
    int y0,  
    int x1,  
    int y1,  

Draw a textured Gouraud-shaded 2D triangle.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>X-coordinate of the first vertex in the image instance.</td>
</tr>
<tr>
<td>y0</td>
<td>Y-coordinate of the first vertex in the image instance.</td>
</tr>
<tr>
<td>x1</td>
<td>X-coordinate of the second vertex in the image instance.</td>
</tr>
<tr>
<td>y1</td>
<td>Y-coordinate of the second vertex in the image instance.</td>
</tr>
<tr>
<td>x2</td>
<td>X-coordinate of the third vertex in the image instance.</td>
</tr>
<tr>
<td>y2</td>
<td>Y-coordinate of the third vertex in the image instance.</td>
</tr>
<tr>
<td>texture</td>
<td>Texture image used to fill the triangle.</td>
</tr>
<tr>
<td>tx0</td>
<td>X-coordinate of the first vertex in the texture image.</td>
</tr>
<tr>
<td>ty0</td>
<td>Y-coordinate of the first vertex in the texture image.</td>
</tr>
<tr>
<td>tx1</td>
<td>X-coordinate of the second vertex in the texture image.</td>
</tr>
<tr>
<td>ty1</td>
<td>Y-coordinate of the second vertex in the texture image.</td>
</tr>
<tr>
<td>tx2</td>
<td>X-coordinate of the third vertex in the texture image.</td>
</tr>
<tr>
<td>ty2</td>
<td>Y-coordinate of the third vertex in the texture image.</td>
</tr>
<tr>
<td>bs0</td>
<td>Brightness factor of the first vertex.</td>
</tr>
<tr>
<td>bs1</td>
<td>Brightness factor of the second vertex.</td>
</tr>
<tr>
<td>bs2</td>
<td>Brightness factor of the third vertex.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>
```cpp
int tx2,
int ty2,
const CImg< tl > & light,
int lx0,
int ly0,
int lx1,
int ly1,
int lx2,
int ly2,
const float opacity = 1 )
```

Draw a textured Phong-shaded 2D triangle.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>X-coordinate of the first vertex in the image instance.</td>
</tr>
<tr>
<td>y0</td>
<td>Y-coordinate of the first vertex in the image instance.</td>
</tr>
<tr>
<td>x1</td>
<td>X-coordinate of the second vertex in the image instance.</td>
</tr>
<tr>
<td>y1</td>
<td>Y-coordinate of the second vertex in the image instance.</td>
</tr>
<tr>
<td>x2</td>
<td>X-coordinate of the third vertex in the image instance.</td>
</tr>
<tr>
<td>y2</td>
<td>Y-coordinate of the third vertex in the image instance.</td>
</tr>
<tr>
<td>texture</td>
<td>Texture image used to fill the triangle.</td>
</tr>
<tr>
<td>tx0</td>
<td>X-coordinate of the first vertex in the texture image.</td>
</tr>
<tr>
<td>ty0</td>
<td>Y-coordinate of the first vertex in the texture image.</td>
</tr>
<tr>
<td>tx1</td>
<td>X-coordinate of the second vertex in the texture image.</td>
</tr>
<tr>
<td>ty1</td>
<td>Y-coordinate of the second vertex in the texture image.</td>
</tr>
<tr>
<td>tx2</td>
<td>X-coordinate of the third vertex in the texture image.</td>
</tr>
<tr>
<td>ty2</td>
<td>Y-coordinate of the third vertex in the texture image.</td>
</tr>
<tr>
<td>light</td>
<td>Light image.</td>
</tr>
<tr>
<td>lx0</td>
<td>X-coordinate of the first vertex in the light image.</td>
</tr>
<tr>
<td>ly0</td>
<td>Y-coordinate of the first vertex in the light image.</td>
</tr>
<tr>
<td>lx1</td>
<td>X-coordinate of the second vertex in the light image.</td>
</tr>
<tr>
<td>ly1</td>
<td>Y-coordinate of the second vertex in the light image.</td>
</tr>
<tr>
<td>lx2</td>
<td>X-coordinate of the third vertex in the light image.</td>
</tr>
<tr>
<td>ly2</td>
<td>Y-coordinate of the third vertex in the light image.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

---

8.1.4.431 draw_rectangle() [1/3]

```cpp
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,  
```
Draw a filled 4D rectangle.

Parameters

- \(x_0\) X-coordinate of the upper-left rectangle corner.
- \(y_0\) Y-coordinate of the upper-left rectangle corner.
- \(z_0\) Z-coordinate of the upper-left rectangle corner.
- \(c_0\) C-coordinate of the upper-left rectangle corner.
- \(x_1\) X-coordinate of the lower-right rectangle corner.
- \(y_1\) Y-coordinate of the lower-right rectangle corner.
- \(z_1\) Z-coordinate of the lower-right rectangle corner.
- \(c_1\) C-coordinate of the lower-right rectangle corner.
- \(v\) Scalar value used to fill the rectangle area.
- \(\text{opacity}\) Drawing opacity.

8.1.4.432  \textbf{\texttt{draw_rectangle}} [2/3]

\begin{verbatim}
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 )
\end{verbatim}

Draw a filled 3D rectangle.

Parameters

- \(x_0\) X-coordinate of the upper-left rectangle corner.
- \(y_0\) Y-coordinate of the upper-left rectangle corner.
- \(z_0\) Z-coordinate of the upper-left rectangle corner.
- \(x_1\) X-coordinate of the lower-right rectangle corner.
- \(y_1\) Y-coordinate of the lower-right rectangle corner.
- \(z_1\) Z-coordinate of the lower-right rectangle corner.
- \(\text{color}\) Pointer to \texttt{spectrum()} consecutive values of type \(T\), defining the drawing color.
- \(\text{opacity}\) Drawing opacity.

8.1.4.433  \textbf{\texttt{draw_rectangle}} [3/3]

\begin{verbatim}
CImg<T>& draw_rectangle (  
    const int x0,  
    const int y0,  
    const int z0,  
    const int x1,  
    const int y1,  
    const int z1,  
    const color *const color,  
    const float opacity = 1 )
\end{verbatim}
```cpp
const int x0,
const int y0,
const int x1,
const int y1,
const tc *const color,
const float opacity = 1
)
```

Draw a filled 2D rectangle.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>X-coordinate of the upper-left rectangle corner.</td>
</tr>
<tr>
<td>y0</td>
<td>Y-coordinate of the upper-left rectangle corner.</td>
</tr>
<tr>
<td>x1</td>
<td>X-coordinate of the lower-right rectangle corner.</td>
</tr>
<tr>
<td>y1</td>
<td>Y-coordinate of the lower-right rectangle corner.</td>
</tr>
<tr>
<td>color</td>
<td>Pointer to <code>spectrum()</code> consecutive values of type <code>T</code>, defining the drawing color.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

8.1.434 draw_polygon()  

```cpp
CImg< T > & draw_polygon(
    const CImg< tp > & points,
    const tc *const color,
    const float opacity = 1
)
```

Draw a filled 2D polygon.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>points</td>
<td>Set of polygon vertices.</td>
</tr>
<tr>
<td>color</td>
<td>Pointer to <code>spectrum()</code> consecutive values of type <code>T</code>, defining the drawing color.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

8.1.435 draw_ellipse() [1/4]

```cpp
CImg< T > & draw_ellipse(
    const int x0,
    const int y0,
    const float r1,
    const float r2,
    const float angle,
    const tc *const color,
    const float opacity = 1
)
```

Draw a filled 2D ellipse.
8.1 CImg< T > Struct Template Reference

Parameters

<table>
<thead>
<tr>
<th>x0</th>
<th>X-coordinate of the ellipse center.</th>
</tr>
</thead>
<tbody>
<tr>
<td>y0</td>
<td>Y-coordinate of the ellipse center.</td>
</tr>
<tr>
<td>r1</td>
<td>First radius of the ellipse.</td>
</tr>
<tr>
<td>r2</td>
<td>Second radius of the ellipse.</td>
</tr>
<tr>
<td>angle</td>
<td>Angle of the first radius.</td>
</tr>
<tr>
<td>color</td>
<td>Pointer to spectrum() consecutive values, defining the drawing color.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

8.1.4.436 draw_ellipse() [2/4]

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 2D ellipse [overloading].

Parameters

<table>
<thead>
<tr>
<th>x0</th>
<th>X-coordinate of the ellipse center.</th>
</tr>
</thead>
<tbody>
<tr>
<td>y0</td>
<td>Y-coordinate of the ellipse center.</td>
</tr>
<tr>
<td>tensor</td>
<td>Diffusion tensor describing the ellipse.</td>
</tr>
<tr>
<td>color</td>
<td>Pointer to spectrum() consecutive values, defining the drawing color.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

8.1.4.437 draw_ellipse() [3/4]

CImg< T >& draw_ellipse (  
  const int x0,  
  const int y0,  
  const float r1,  
  const float r2,  
  const float angle,  
  const tc *const color,  
  const float opacity,  
  const unsigned int pattern )  

Draw an outlined 2D ellipse.

Parameters

<table>
<thead>
<tr>
<th>x0</th>
<th>X-coordinate of the ellipse center.</th>
</tr>
</thead>
</table>

Generated by Doxygen
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_0$</td>
<td>Y-coordinate of the ellipse center.</td>
</tr>
<tr>
<td>$r_1$</td>
<td>First radius of the ellipse.</td>
</tr>
<tr>
<td>$r_2$</td>
<td>Second radius of the ellipse.</td>
</tr>
<tr>
<td>$\text{angle}$</td>
<td>Angle of the first radius.</td>
</tr>
<tr>
<td>$\text{color}$</td>
<td>Pointer to <code>spectrum()</code> consecutive values, defining the drawing color.</td>
</tr>
<tr>
<td>$\text{opacity}$</td>
<td>Drawing opacity.</td>
</tr>
<tr>
<td>$\text{pattern}$</td>
<td>An integer whose bits describe the outline pattern.</td>
</tr>
</tbody>
</table>

#### 8.1.4.438 `draw_ellipse()` [4/4]

```cpp
CImg<typename T>& draw_ellipse(
    const int $x_0$,
    const int $y_0$,
    const CImg<typename T>& tensor,
    const tc*const color,
    const float $\text{opacity}$,
    const unsigned int $\text{pattern}$
)
```

Draw an outlined 2D ellipse [overloading].

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_0$</td>
<td>X-coordinate of the ellipse center.</td>
</tr>
<tr>
<td>$y_0$</td>
<td>Y-coordinate of the ellipse center.</td>
</tr>
<tr>
<td>$\text{tensor}$</td>
<td>Diffusion tensor describing the ellipse.</td>
</tr>
<tr>
<td>$\text{color}$</td>
<td>Pointer to <code>spectrum()</code> consecutive values, defining the drawing color.</td>
</tr>
<tr>
<td>$\text{opacity}$</td>
<td>Drawing opacity.</td>
</tr>
<tr>
<td>$\text{pattern}$</td>
<td>An integer whose bits describe the outline pattern.</td>
</tr>
</tbody>
</table>

#### 8.1.4.439 `draw_circle()` [1/2]

```cpp
CImg<typename T>& draw_circle(
    const int $x_0$,
    const int $y_0$,
    int $\text{radius}$,
    const tc*const color,
    const float $\text{opacity} = 1$
)
```

Draw a filled 2D circle.

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_0$</td>
<td>X-coordinate of the circle center.</td>
</tr>
<tr>
<td>$y_0$</td>
<td>Y-coordinate of the circle center.</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>radius</th>
<th>Circle radius.</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>Pointer to <code>spectrum()</code> consecutive values, defining the drawing color.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

Note

- Circle version of the Bresenham’s algorithm is used.

8.1.4.440 draw_circle() [2/2]

```cpp
CImg<T>& draw_circle (  
    const int x0,  
    const int y0,  
    int radius,  
    const tc *const color,  
    const float opacity,  
    const unsigned int pattern )
```

Draw an outlined 2D circle.

Parameters

<table>
<thead>
<tr>
<th>x0</th>
<th>X-coordinate of the circle center.</th>
</tr>
</thead>
<tbody>
<tr>
<td>y0</td>
<td>Y-coordinate of the circle center.</td>
</tr>
<tr>
<td>radius</td>
<td>Circle radius.</td>
</tr>
<tr>
<td>color</td>
<td>Pointer to <code>spectrum()</code> consecutive values, defining the drawing color.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
<tr>
<td>pattern</td>
<td>An integer whose bits describe the outline pattern.</td>
</tr>
</tbody>
</table>

8.1.4.441 draw_image() [1/2]

```cpp
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. |

Generated by Doxygen
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>X-coordinate of the sprite position.</td>
</tr>
<tr>
<td>y0</td>
<td>Y-coordinate of the sprite position.</td>
</tr>
<tr>
<td>z0</td>
<td>Z-coordinate of the sprite position.</td>
</tr>
<tr>
<td>c0</td>
<td>C-coordinate of the sprite position.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

#### 8.1.4.442 draw_image() [2/2]

```cpp
CImg<
>

draw_image {
    const int x0,
    const int y0,
    const int z0,
    const int c0,
    const CImg<
          > & sprite,
    const CImg<
          > & mask,
    const float opacity = 1,
    const float mask_max_value = 1
}
```

Draw a masked image.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sprite</td>
<td>Sprite image.</td>
</tr>
<tr>
<td>mask</td>
<td>Mask image.</td>
</tr>
<tr>
<td>x0</td>
<td>X-coordinate of the sprite position in the image instance.</td>
</tr>
<tr>
<td>y0</td>
<td>Y-coordinate of the sprite position in the image instance.</td>
</tr>
<tr>
<td>z0</td>
<td>Z-coordinate of the sprite position in the image instance.</td>
</tr>
<tr>
<td>c0</td>
<td>C-coordinate of the sprite position in the image instance.</td>
</tr>
<tr>
<td>mask_max_value</td>
<td>Maximum pixel value of the mask image mask.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

#### Note

- Pixel values of mask set the opacity of the corresponding pixels in sprite.
- Dimensions along x, y and z of sprite and mask must be the same.

#### 8.1.4.443 draw_text() [1/4]

```cpp
CImg<
>
draw_text {
    const int x0,
    const int y0,
    const char * const text,
```
8.1 CImg<T> Struct Template Reference

```c
const tc1 *const foreground_color,
const tc2 *const background_color,
const float opacity,
const CImgList< t > & font,
... )
```

Draw a text string.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>X-coordinate of the text in the image instance.</td>
</tr>
<tr>
<td>y0</td>
<td>Y-coordinate of the text in the image instance.</td>
</tr>
<tr>
<td>text</td>
<td>Format of the text ('printf'-style format string).</td>
</tr>
<tr>
<td>foreground_color</td>
<td>Pointer to <code>spectrum()</code> consecutive values, defining the foreground drawing color.</td>
</tr>
<tr>
<td>background_color</td>
<td>Pointer to <code>spectrum()</code> consecutive values, defining the background drawing color.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
<tr>
<td>font</td>
<td>Font used for drawing text.</td>
</tr>
</tbody>
</table>

8.1.4.444 `draw_text()` [2/4]

```c
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,  
... )
```

Draw a text string **[overloading]**.

**Note**

A transparent background is used for the text.

8.1.4.445 `draw_text()` [3/4]

```c
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,  
... )
```

Draw a text string **[overloading]**.

**Note**

A transparent foreground is used for the text.
8.1.4.446  draw_text() [4/4]

CImg&lt;T&gt; &amp; 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 string [overloading].

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>X-coordinate of the text in the image instance.</td>
</tr>
<tr>
<td>y0</td>
<td>Y-coordinate of the text in the image instance.</td>
</tr>
<tr>
<td>text</td>
<td>Format of the text ('printf'-style format string).</td>
</tr>
<tr>
<td>foreground_color</td>
<td>Array of spectrum() values of type T, defining the foreground color (0 means 'transparent').</td>
</tr>
<tr>
<td>background_color</td>
<td>Array of spectrum() values of type T, defining the background color (0 means 'transparent').</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
<tr>
<td>font_height</td>
<td>Height of the text font (exact match for 13,23,53,103, interpolated otherwise).</td>
</tr>
</tbody>
</table>

8.1.4.447  draw_quiver() [1/2]

CImg&lt;T&gt; &amp; draw_quiver (
    const CImg&lt; t1 &gt; &amp; flow,
    const t2 *const color,
    const float opacity = 1,
    const unsigned int sampling = 25,
    const float factor = -20,
    const bool is_arrow = true,
    const unsigned int pattern = ~0U )

Draw a 2D vector field.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flow</td>
<td>Image of 2D vectors used as input data.</td>
</tr>
<tr>
<td>color</td>
<td>Image of spectrum()-D vectors corresponding to the color of each arrow.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
<tr>
<td>sampling</td>
<td>Length (in pixels) between each arrow.</td>
</tr>
<tr>
<td>factor</td>
<td>Length factor of each arrow (if &lt;0, computed as a percentage of the maximum length).</td>
</tr>
<tr>
<td>is_arrow</td>
<td>Tells if arrows must be drawn, instead of oriented segments.</td>
</tr>
<tr>
<td>pattern</td>
<td>Used pattern to draw lines.</td>
</tr>
</tbody>
</table>
8.1.4.448 draw_quiver() [2/2]

`CImg<T>& draw_quiver ( `const CImg< t1 > & flow, `const CImg< t2 > & color, `const float opacity = 1, `const unsigned int sampling = 25, `const float factor = -20, `const bool is_arrow = true, `const unsigned int pattern = ~0U )`

Draw a 2D vector field, using a field of colors.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flow</td>
<td>Image of 2D vectors used as input data.</td>
</tr>
<tr>
<td>color</td>
<td>Image of <code>spectrum()</code>-D vectors corresponding to the color of each arrow.</td>
</tr>
<tr>
<td>opacity</td>
<td>Opacity of the drawing.</td>
</tr>
<tr>
<td>sampling</td>
<td>Length (in pixels) between each arrow.</td>
</tr>
<tr>
<td>factor</td>
<td>Length factor of each arrow (if &lt;0, computed as a percentage of the maximum length).</td>
</tr>
<tr>
<td>is_arrow</td>
<td>Tells if arrows must be drawn, instead of oriented segments.</td>
</tr>
<tr>
<td>pattern</td>
<td>Used pattern to draw lines.</td>
</tr>
</tbody>
</table>

Note

Clipping is supported.

8.1.4.449 draw_axis() [1/2]

`CImg<T>& draw_axis ( `const CImg< t > & values_x, `const int y, `const tc *const color, `const float opacity = 1, `const unsigned int pattern = ~0U, `const unsigned int font_height = 13, `const bool allow_zero = true, `const float round_x = 0 )`

Draw a labeled horizontal axis.

Note

Clipping is supported.
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>values_x</code></td>
<td>Values along the horizontal axis.</td>
</tr>
<tr>
<td><code>y</code></td>
<td>Y-coordinate of the horizontal axis in the image instance.</td>
</tr>
<tr>
<td><code>color</code></td>
<td>Pointer to <code>spectrum()</code> consecutive values, defining the drawing color.</td>
</tr>
<tr>
<td><code>opacity</code></td>
<td>Drawing opacity.</td>
</tr>
<tr>
<td><code>pattern</code></td>
<td>Drawing pattern.</td>
</tr>
<tr>
<td><code>font_height</code></td>
<td>Height of the labels (exact match for 13,23,53,103, interpolated otherwise).</td>
</tr>
<tr>
<td><code>allow_zero</code></td>
<td>Enable/disable the drawing of label '0' if found.</td>
</tr>
</tbody>
</table>

8.1.4.450  **draw_axis()** [2/2]

```cpp
CImg<T>& draw_axis(
    const int x,
    const CImg< t > & values_y,
    const tc *const color,
    const float opacity = 1,
    const unsigned int pattern = ~0U,
    const unsigned int font_height = 13,
    const bool allow_zero = true,
    const float round_y = 0 )
```

Draw a labeled vertical axis.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x</code></td>
<td>X-coordinate of the vertical axis in the image instance.</td>
</tr>
<tr>
<td><code>values_y</code></td>
<td>Values along the Y-axis.</td>
</tr>
<tr>
<td><code>color</code></td>
<td>Pointer to <code>spectrum()</code> consecutive values, defining the drawing color.</td>
</tr>
<tr>
<td><code>opacity</code></td>
<td>Drawing opacity.</td>
</tr>
<tr>
<td><code>pattern</code></td>
<td>Drawing pattern.</td>
</tr>
<tr>
<td><code>font_height</code></td>
<td>Height of the labels (exact match for 13,23,53,103, interpolated otherwise).</td>
</tr>
<tr>
<td><code>allow_zero</code></td>
<td>Enable/disable the drawing of label '0' if found.</td>
</tr>
</tbody>
</table>

8.1.4.451  **draw_axes()**

```cpp
CImg<T>& draw_axes(
    const CImg< tx > & values_x,
    const CImg< ty > & values_y,
    const tc *const color,
    const float opacity = 1,
    const unsigned int pattern_x = ~0U,
    const unsigned int pattern_y = ~0U,
    const unsigned int font_height = 13,
    const bool allow_zero = true,
)```
const float round_x = 0,
const float round_y = 0
}

Draw labeled horizontal and vertical axes.

Parameters

| values_x | Values along the X-axis. |
| values_y | Values along the Y-axis. |
| color     | Pointer to spectrum() consecutive values, defining the drawing color. |
| opacity   | Drawing opacity. |
| pattern_x | Drawing pattern for the X-axis. |
| pattern_y | Drawing pattern for the Y-axis. |
| font_height | Height of the labels (exact match for 13,23,53,103, interpolated otherwise). |
| allow_zero | Enable/disable the drawing of label '0' if found. |

8.1.4.452 draw_grid()

CImg<T>& draw_grid {
    const CImg<tx>& values_x,
    const CImg<ty>& values_y,
    const tc*const color,
    const float opacity = 1,
    const unsigned int pattern_x = ~0U,
    const unsigned int pattern_y = ~0U
}

Draw 2D grid.

Parameters

| values_x | X-coordinates of the vertical lines. |
| values_y | Y-coordinates of the horizontal lines. |
| color     | Pointer to spectrum() consecutive values, defining the drawing color. |
| opacity   | Drawing opacity. |
| pattern_x | Drawing pattern for vertical lines. |
| pattern_y | Drawing pattern for horizontal lines. |

8.1.4.453 draw_graph()

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 1D graph.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Image containing the graph values ( I = f(x) ).</td>
</tr>
<tr>
<td>color</td>
<td>Pointer to \texttt{spectrum()} consecutive values, defining the drawing color.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
<tr>
<td>plot_type</td>
<td>Define the type of the plot:</td>
</tr>
<tr>
<td></td>
<td>• 0 = No plot.</td>
</tr>
<tr>
<td></td>
<td>• 1 = Plot using segments.</td>
</tr>
<tr>
<td></td>
<td>• 2 = Plot using cubic splines.</td>
</tr>
<tr>
<td></td>
<td>• 3 = Plot with bars.</td>
</tr>
<tr>
<td>vertex_type</td>
<td>Define the type of points:</td>
</tr>
<tr>
<td></td>
<td>• 0 = No points.</td>
</tr>
<tr>
<td></td>
<td>• 1 = Point.</td>
</tr>
<tr>
<td></td>
<td>• 2 = Straight cross.</td>
</tr>
<tr>
<td></td>
<td>• 3 = Diagonal cross.</td>
</tr>
<tr>
<td></td>
<td>• 4 = Filled circle.</td>
</tr>
<tr>
<td></td>
<td>• 5 = Outlined circle.</td>
</tr>
<tr>
<td></td>
<td>• 6 = Square.</td>
</tr>
<tr>
<td></td>
<td>• 7 = Diamond.</td>
</tr>
<tr>
<td>ymin</td>
<td>Lower bound of the y-range.</td>
</tr>
<tr>
<td>ymax</td>
<td>Upper bound of the y-range.</td>
</tr>
<tr>
<td>pattern</td>
<td>Drawing pattern.</td>
</tr>
</tbody>
</table>

Note

- if \( ymin == ymax == 0 \), the y-range is computed automatically from the input samples.

8.1.4.454\hspace{1em} draw_fill()\hspace{1em}

\texttt{CImg<T> & draw_fill (}
    \hspace{1em}const int x0,
    \hspace{1em}const int y0,
\texttt{)}
Draw filled 3D region with the flood fill algorithm.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>X-coordinate of the starting point of the region to fill.</td>
</tr>
<tr>
<td>y0</td>
<td>Y-coordinate of the starting point of the region to fill.</td>
</tr>
<tr>
<td>z0</td>
<td>Z-coordinate of the starting point of the region to fill.</td>
</tr>
<tr>
<td>color</td>
<td>Pointer to spectrum() consecutive values, defining the drawing color.</td>
</tr>
<tr>
<td>out region</td>
<td>Image that will contain the mask of the filled region mask, as an output.</td>
</tr>
<tr>
<td>tolerance</td>
<td>Tolerance concerning neighborhood values.</td>
</tr>
<tr>
<td>opacity</td>
<td>Opacity of the drawing.</td>
</tr>
<tr>
<td>is_high_connectivity</td>
<td>Tells if 8-connexity must be used.</td>
</tr>
</tbody>
</table>

Returns

region is initialized with the binary mask of the filled region.

8.1.4.455 draw_plasma()

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

Draw a random plasma texture.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha</td>
<td>Alpha-parameter.</td>
</tr>
<tr>
<td>beta</td>
<td>Beta-parameter.</td>
</tr>
<tr>
<td>scale</td>
<td>Scale-parameter.</td>
</tr>
</tbody>
</table>

Note

Use the mid-point algorithm to render.

8.1.4.456 draw_mandelbrot()

CImg<T>& draw_mandelbrot ( 
    const int x0,
    const int y0,
    const int z0,
    const tc *const color,
    const float opacity,
    CImg<T>& region,
    const float tolerance = 0,
    const bool is_high_connectivity = false )
const int y0,
const int x1,
const int y1,
const CImg<tc>& colormap,
const float opacity = 1,
const double z0r = -2,
const double z0i = -2,
const double z1r = 2,
const double z1i = 2,
const unsigned int iteration_max = 255,
const bool is_normalized_iteration = false,
const bool is_julia_set = false,
const double param_r = 0,
const double param_i = 0)

Draw a quadratic Mandelbrot or Julia 2D fractal.

Parameters

|x0| X-coordinate of the upper-left pixel. |
|y0| Y-coordinate of the upper-left pixel. |
|x1| X-coordinate of the lower-right pixel. |
|y1| Y-coordinate of the lower-right pixel. |
|colormap| Colormap. |
|opacity| Drawing opacity. |
|z0r| Real part of the upper-left fractal vertex. |
|z0i| Imaginary part of the upper-left fractal vertex. |
|z1r| Real part of the lower-right fractal vertex. |
|z1i| Imaginary part of the lower-right fractal vertex. |
|iteration_max| Maximum number of iterations for each estimated point. |
|is_normalized_iteration| Tells if iterations are normalized. |
|is_julia_set| Tells if the Mandelbrot or Julia set is rendered. |
|param_r| Real part of the Julia set parameter. |
|param_i| Imaginary part of the Julia set parameter. |

Note
Fractal rendering is done by the Escape Time Algorithm.

8.1.4.457 draw_gaussian() [1/2]

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

Draw a 1D gaussian function.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( xc )</td>
<td>X-coordinate of the gaussian center.</td>
</tr>
<tr>
<td>( sigma )</td>
<td>Standard variation of the gaussian distribution.</td>
</tr>
<tr>
<td>( color )</td>
<td>Pointer to <code>spectrum()</code> consecutive values, defining the drawing color.</td>
</tr>
<tr>
<td>( opacity )</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

8.1.4.458 \texttt{draw\_gaussian()} [2/2]

\[
\begin{align*}
\text{CImg}<T> & \texttt{draw\_gaussian (} \\
& \quad \text{const float } xc, \\
& \quad \text{const float } yc, \\
& \quad \text{const CImg}<t> & \texttt{tensor}, \\
& \quad \text{const tc } & \texttt{color}, \\
& \quad \text{const float } opacity = 1 \\
\end{align*}
\]

Draw a 2D gaussian function.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( xc )</td>
<td>X-coordinate of the gaussian center.</td>
</tr>
<tr>
<td>( yc )</td>
<td>Y-coordinate of the gaussian center.</td>
</tr>
<tr>
<td>( tensor )</td>
<td>Covariance matrix (must be 2x2).</td>
</tr>
<tr>
<td>( color )</td>
<td>Pointer to <code>spectrum()</code> consecutive values, defining the drawing color.</td>
</tr>
<tr>
<td>( opacity )</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

8.1.4.459 \texttt{draw\_object3d()}

\[
\begin{align*}
\text{CImg}<T> & \texttt{draw\_object3d (} \\
& \quad \text{const float } x0, \\
& \quad \text{const float } y0, \\
& \quad \text{const float } z0, \\
& \quad \text{const CImg}<tp> & \texttt{vertices}, \\
& \quad \text{const CImgList}<tf> & \texttt{primitives}, \\
& \quad \text{const CImgList}<tc> & \texttt{colors}, \\
& \quad \text{const CImg}<to> & \texttt{opacities}, \\
& \quad \text{const unsigned int } render\_type = 4, \\
& \quad \text{const bool } is\_double\_sided = false, \\
& \quad \text{const float } focale = 700, \\
& \quad \text{const float } lightx = 0, \\
& \quad \text{const float } lighty = 0, \\
& \quad \text{const float } lightz = -5e8, \\
& \quad \text{const float } specular\_lightness = 0.2f, \\
& \quad \text{const float } specular\_shininess = 0.1f, \\
& \quad \text{const float } g\_opacity = 1 \\
\end{align*}
\]

Draw a 3D object.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_0 )</td>
<td>X-coordinate of the 3D object position</td>
</tr>
<tr>
<td>( y_0 )</td>
<td>Y-coordinate of the 3D object position</td>
</tr>
<tr>
<td>( z_0 )</td>
<td>Z-coordinate of the 3D object position</td>
</tr>
<tr>
<td>\textit{vertices}</td>
<td>Image Nx3 describing 3D point coordinates</td>
</tr>
<tr>
<td>\textit{primitives}</td>
<td>List of P primitives</td>
</tr>
<tr>
<td>\textit{colors}</td>
<td>List of P color (or textures)</td>
</tr>
<tr>
<td>\textit{opacities}</td>
<td>Image or list of P opacities</td>
</tr>
<tr>
<td>\textit{render_type}</td>
<td>d Render type (0=Points, 1=Lines, 2=Faces (no light), 3=Faces (flat), 4=Faces(Gouraud))</td>
</tr>
<tr>
<td>\textit{is_double_sided}</td>
<td>Tells if object faces have two sides or are oriented.</td>
</tr>
<tr>
<td>\textit{focale}</td>
<td>Length of the focale (0 for parallel projection)</td>
</tr>
<tr>
<td>\textit{lightx}</td>
<td>X-coordinate of the light</td>
</tr>
<tr>
<td>\textit{lighty}</td>
<td>Y-coordinate of the light</td>
</tr>
<tr>
<td>\textit{lightz}</td>
<td>Z-coordinate of the light</td>
</tr>
<tr>
<td>\textit{specular_lightness}</td>
<td>Amount of specular light.</td>
</tr>
<tr>
<td>\textit{specular_shininess}</td>
<td>Shininess of the object</td>
</tr>
<tr>
<td>\textit{g_opacity}</td>
<td>Global opacity of the object.</td>
</tr>
</tbody>
</table>

8.1.4.460 select()

\texttt{CImg<T>& select (}
\texttt{    CImgDisplay & disp,}
\texttt{    const unsigned int \textit{feature_type} = 2,}
\texttt{    unsigned int *const \textit{XYZ} = 0,}
\texttt{    const bool \textit{exit_on_anykey} = false,}
\texttt{    const bool \textit{is_deep_selection_default} = false )}

Launch simple interface to select a shape from an image.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{disp}</td>
<td>Display window to use.</td>
</tr>
<tr>
<td>\textit{feature_type}</td>
<td>Type of feature to select. Can be ( 0=point</td>
</tr>
<tr>
<td>\textit{XYZ}</td>
<td>Pointer to 3 values X,Y,Z which tells about the projection point coordinates, for volumetric images.</td>
</tr>
<tr>
<td>\textit{exit_on_anykey}</td>
<td>Exit function when any key is pressed.</td>
</tr>
</tbody>
</table>

8.1.4.461 load()

\texttt{CImg<T>& load (}
\texttt{    const char *const \textit{filename} )}

Load image from a file.
Parameters

| filename | Filename, as a C-string. |

Note
The extension of filename defines the file format. If no filename extension is provided, CImg::get_load() will try to load the file as a .cimg or .cimgz file.

8.1.4.462  load_ascii()

```cpp
CImg<T>& load_ascii (const char *const filename)
```

Load image from an Ascii file.

Parameters

| filename | Filename, as a C-string. |

8.1.4.463  load_dlm()

```cpp
CImg<T>& load_dlm (const char *const filename)
```

Load image from a DLM file.

Parameters

| filename | Filename, as a C-string. |

8.1.4.464  load_bmp()

```cpp
CImg<T>& load_bmp (const char *const filename)
```

Load image from a BMP file.

Parameters

| filename | Filename, as a C-string. |
8.1.4.465 load_jpeg()

```cpp
CImg<T>& load_jpeg (
    const char *const filename )
```

Load image from a JPEG file.

Parameters

| filename | Filename, as a C-string. |

8.1.4.466 load_magick()

```cpp
CImg<T>& load_magick (  
    const char *const filename )
```

Load image from a file, using Magick++ library.

Parameters

| filename | Filename, as a C-string. |

8.1.4.467 load_png()

```cpp
CImg<T>& load_png (  
    const char *const filename,  
    unsigned int *const bits_per_pixel = 0 )
```

Load image from a PNG file.

Parameters

<table>
<thead>
<tr>
<th>filename</th>
<th>Filename, as a C-string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>bits_per_pixel</td>
</tr>
</tbody>
</table>

8.1.4.468 load_pnm()
Load image from a PNM file.
### load_pfm()

```cpp
cImg<T>& load_pfm (const char *const filename)
```

Load image from a PFM file.

**Parameters**

| filename | Filename, as a C-string. |

### load_rgb()

```cpp
cImg<T>& load_rgb (const char *const filename, const unsigned int dimw, const unsigned int dimh = 1)
```

Load image from a RGB file.

**Parameters**

| filename | Filename, as a C-string. |
| dimw     | Width of the image buffer. |
| dimh     | Height of the image buffer. |

### load_rgba()

```cpp
cImg<T>& load_rgba (const char *const filename, const unsigned int dimw, const unsigned int dimh = 1)
```

Load image from a RGBA file.

**Parameters**

| filename | Filename, as a C-string. |
| dimw     | Width of the image buffer. |
| dimh     | Height of the image buffer. |
8.1.4.472 load_tiff()

\texttt{CImg< T> \& load_tiff (}
\begin{verbatim}
  const char *const filename, 
  const unsigned int first_frame = 0, 
  const unsigned int last_frame = ~0U, 
  const unsigned int step_frame = 1, 
  float *const voxel_size = 0, 
  CImg< charT > *const description = 0 )
\end{verbatim}

Load image from a TIFF file.

\textbf{Parameters}

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename, as a C-string.</td>
</tr>
<tr>
<td>first_frame</td>
<td>First frame to read (for multi-pages tiff).</td>
</tr>
<tr>
<td>last_frame</td>
<td>Last frame to read (for multi-pages tiff).</td>
</tr>
<tr>
<td>step_frame</td>
<td>Step value of frame reading.</td>
</tr>
<tr>
<td>out voxel_size</td>
<td>Voxel size, as stored in the filename.</td>
</tr>
<tr>
<td>out description</td>
<td>Description, as stored in the filename.</td>
</tr>
</tbody>
</table>

\textbf{Note}

- libtiff support is enabled by defining the precompilation directive \texttt{cimg\_use\_tif}.
- When libtiff is enabled, 2D and 3D (multipage) several channel per pixel are supported for \texttt{char, uchar, short, ushort, float and double} pixel types.
- If \texttt{cimg\_use\_tif} is not defined at compile time the function uses \texttt{CImg< T> \& load_other(const char\*)}.

8.1.4.473 load_minc2()

\texttt{CImg< T> \& load_minc2 (}
\begin{verbatim}
  const char *const filename )
\end{verbatim}

Load image from a MINC2 file.

\textbf{Parameters}

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename, as a C-string.</td>
</tr>
</tbody>
</table>

8.1.4.474 load_analyze()

\texttt{CImg< T> \& load_analyze (}
\begin{verbatim}
\end{verbatim}

Generated by Doxygen
Load image from an ANALYZE7.5/NIFTI file.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>filename</code></td>
<td>Filename, as a C-string.</td>
</tr>
<tr>
<td><code>voxel_size</code></td>
<td>Pointer to the three voxel sizes read from the file.</td>
</tr>
</tbody>
</table>

8.1.4.475  

```cpp
CImg<T>& load_cimg {
    const char *const filename,
    const char axis = 'z',
    const float align = 0
}
```

Load image from a .cimg[z] file.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>filename</code></td>
<td>Filename, as a C-string.</td>
</tr>
<tr>
<td><code>axis</code></td>
<td>Appending axis, if file contains multiple images. Can be `{ 'x'</td>
</tr>
<tr>
<td><code>align</code></td>
<td>Appending alignment.</td>
</tr>
</tbody>
</table>

8.1.4.476  

```cpp
CImg<T>& load_cimg {
    const char *const filename,
    const unsigned int n0,
    const unsigned int n1,
    const unsigned int x0,
    const unsigned int y0,
    const unsigned int z0,
    const unsigned int c0,
    const unsigned int x1,
    const unsigned int y1,
    const unsigned int z1,
    const unsigned int c1,
    const char axis = 'z',
    const float align = 0
}
```

Load sub-images of a .cimg file.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>filename</code></td>
<td>Filename, as a C-string.</td>
</tr>
</tbody>
</table>
8.1 CImg\textless{} T \textgreater{} Struct Template Reference

Parameters

\begin{tabular}{|l|l|}
\hline
\textit{n0} & Starting frame. \\
\hline
\textit{n1} & Ending frame (~0U for max). \\
\hline
\textit{x0} & X-coordinate of the starting sub-image vertex. \\
\hline
\textit{y0} & Y-coordinate of the starting sub-image vertex. \\
\hline
\textit{z0} & Z-coordinate of the starting sub-image vertex. \\
\hline
\textit{c0} & C-coordinate of the starting sub-image vertex. \\
\hline
\textit{x1} & X-coordinate of the ending sub-image vertex (~0U for max). \\
\hline
\textit{y1} & Y-coordinate of the ending sub-image vertex (~0U for max). \\
\hline
\textit{z1} & Z-coordinate of the ending sub-image vertex (~0U for max). \\
\hline
\textit{c1} & C-coordinate of the ending sub-image vertex (~0U for max). \\
\hline
\textit{axis} & Appending axis, if file contains multiple images. Can be \{ \textit{'x'} | \textit{'y'} | \textit{'z'} | \textit{'c'} \}. \\
\hline
\textit{align} & Appending alignment. \\
\hline
\end{tabular}

8.1.4.477 load_inr()

\begin{verbatim}
CImg<T>\& load_inr ( 
    const char \*const filename, 
    float \*const voxel_size = 0 )
\end{verbatim}

Load image from an INRIMAGE-4 file.

Parameters

\begin{tabular}{|l|l|}
\hline
\textit{filename} & Filename, as a C-string. \\
\hline
\textit{out voxel_size} & Pointer to the three voxel sizes read from the file. \\
\hline
\end{tabular}

8.1.4.478 load_exr()

\begin{verbatim}
CImg<T>\& load_exr ( 
    const char \*const filename )
\end{verbatim}

Load image from a EXR file.

Parameters

\begin{tabular}{|l|l|}
\hline
\textit{filename} & Filename, as a C-string. \\
\hline
\end{tabular}
8.1.4.479  load_pandore()

CImg<T>& load_pandore (  
    const char *const filename )

Load image from a PANDORE-5 file.

Parameters

| filename | Filename, as a C-string. |

8.1.4.480  load_parrec()

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

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

Parameters

| filename | Filename, as a C-string. |
| axis | Appending axis, if file contains multiple images. Can be { 'x' | 'y' | 'z' | 'c' }. |
| align | Appending alignment. |

8.1.4.481  load_raw()

CImg<T>& load_raw (  
    const char *const filename,  
    const unsigned int size_x = 0,  
    const unsigned int size_y = 1,  
    const unsigned int size_z = 1,  
    const unsigned int size_c = 1,  
    const bool is_multiplexed = false,  
    const bool invert_endianness = false,  
    const ulongT offset = 0 )

Load image from a raw binary file.

Parameters

| filename | Filename, as a C-string. |
| size_x | Width of the image buffer. |
| size_y | Height of the image buffer. |
| size_z | Depth of the image buffer. |
8.1.4.482  load_yuv()

CImg< T > & load_yuv ( const char *const filename,
        const unsigned int size_x,
        const unsigned int size_y = 1,
        const unsigned int chroma_subsampling = 444,
        const unsigned int first_frame = 0,
        const unsigned int last_frame = ~0U,
        const unsigned int step_frame = 1,
        const bool yuv2rgb = true,
        const char axis = 'z' )

Load image sequence from a YUV file.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename, as a C-string.</td>
</tr>
<tr>
<td>size_x</td>
<td>Width of the frames.</td>
</tr>
<tr>
<td>size_y</td>
<td>Height of the frames.</td>
</tr>
<tr>
<td>chroma_subsampling</td>
<td>Type of chroma subsampling. Can be { 420</td>
</tr>
<tr>
<td>first_frame</td>
<td>Index of the first frame to read.</td>
</tr>
<tr>
<td>last_frame</td>
<td>Index of the last frame to read.</td>
</tr>
<tr>
<td>step_frame</td>
<td>Step value for frame reading.</td>
</tr>
<tr>
<td>yuv2rgb</td>
<td>Tells if the YUV to RGB transform must be applied.</td>
</tr>
<tr>
<td>axis</td>
<td>Appending axis, if file contains multiple images. Can be { 'x'</td>
</tr>
</tbody>
</table>
8.1.4.484  load_video()

```cpp
taking CImg<T>

load_video ( const char *const filename,

        const unsigned int first_frame = 0,
        const unsigned int last_frame = ~0U,
        const unsigned int step_frame = 1,
        const char axis = 'z',
        const float align = 0 )
```

Load image sequence from a video file, using OpenCV library.

8.1.4.485  load_ffmpeg_external()

```cpp
load_ffmpeg_external ( const char *const filename,

        const char axis = 'z',
        const float align = 0 )
```

Load image sequence using FFMPEG's external tool 'ffmpeg'.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename, as a C-string.</td>
</tr>
<tr>
<td>axis</td>
<td>Alignment axis.</td>
</tr>
<tr>
<td>align</td>
<td>Appending alignment.</td>
</tr>
<tr>
<td>first_frame</td>
<td>Index of the first frame to read.</td>
</tr>
<tr>
<td>last_frame</td>
<td>Index of the last frame to read.</td>
</tr>
<tr>
<td>step_frame</td>
<td>Step value for frame reading.</td>
</tr>
<tr>
<td>align</td>
<td>Appending alignment.</td>
</tr>
<tr>
<td>axis</td>
<td>Appending axis, if file contains multiple images. Can be {'x', 'y', 'z', 'c'}.</td>
</tr>
</tbody>
</table>
8.1.4.486 load_gif_external()

```cpp
CImg<T>& load_gif_external (
    const char *const filename,
    const char axis = 'z',
    const float align = 0
)
```

Load gif file, using Imagemagick or GraphicsMagick's external tools.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename, as a C-string.</td>
</tr>
<tr>
<td>axis</td>
<td>Appending axis, if file contains multiple images. Can be ( 'x'</td>
</tr>
<tr>
<td>align</td>
<td>Appending alignment.</td>
</tr>
</tbody>
</table>

8.1.4.487 load_graphicsmagick_external()

```cpp
CImg<T>& load_graphicsmagick_external (
    const char *const filename
)
```

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename, as a C-string.</td>
</tr>
</tbody>
</table>

8.1.4.488 load_gzip_external()

```cpp
CImg<T>& load_gzip_external (
    const char *const filename
)
```

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename, as a C-string.</td>
</tr>
</tbody>
</table>

8.1.4.489 load_imagemagick_external()

```cpp
CImg<T>& load_imagemagick_external (
    const char *const filename
)
```

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

| **filename** | Filename, as a C-string. |

8.1.4.490 load_medcon_external()

```cpp
CImg<T>& load_medcon_external ( 
    const char *const filename )
```

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

Parameters

| **filename** | Filename, as a C-string. |

8.1.4.491 load_dcraw_external()

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

Load image from a RAW Color Camera file, using external tool 'dcraw'.

Parameters

| **filename** | Filename, as a C-string. |

8.1.4.492 load_camera()

```cpp
CImg<T>& load_camera ( 
    const unsigned int camera_index = 0, 
    const unsigned int capture_width = 0, 
    const unsigned int capture_height = 0, 
    const unsigned int skip_frames = 0, 
    const bool release_camera = true )
```

Load image from a camera stream, using OpenCV.

Parameters

| **index** | Index of the camera to capture images from (from 0 to 63). |
| **capture_width** | Width of the desired image ('0' stands for default value). |
| **capture_height** | Height of the desired image ('0' stands for default value). |
| **skip_frames** | Number of frames to skip before the capture. |
| **release_camera** | Tells if the camera resource must be released at the end of the method. |

Generated by Doxygen
8.1.4.493 load_other()

```cpp
CImg<T>& load_other ( const char *const filename )
```

Load image using various non-native ways.

**Parameters**

- `filename`: Filename, as a C-string.

8.1.4.494 print()

```cpp
const CImg<T>& print ( const char *const title = 0, const bool display_stats = true ) const
```

Display information about the image data.

**Parameters**

- `title`: Name for the considered image.
- `display_stats`: Tells to compute and display image statistics.

8.1.4.495 display() [1/3]

```cpp
const CImg<T>& display ( CImgDisplay & disp ) const
```

Display image into a CImgDisplay window.

**Parameters**

- `disp`: Display window.

8.1.4.496 display() [2/3]

```cpp
const CImg<T>& display ( CImgDisplay & disp,
```
const bool display_info, 
unsigned int *const XYZ = 0, 
const bool exit_on_anykey = false) const

Display image into a CImgDisplay window, in an interactive way.

Parameters

<table>
<thead>
<tr>
<th>disp</th>
<th>Display window.</th>
</tr>
</thead>
<tbody>
<tr>
<td>display_info</td>
<td>Tells if image information are displayed on the standard output.</td>
</tr>
<tr>
<td>in,out XYZ</td>
<td>Contains the XYZ coordinates at start / exit of the function.</td>
</tr>
<tr>
<td>exit_on_anykey</td>
<td>Exit function when any key is pressed.</td>
</tr>
</tbody>
</table>

8.1.4.497 display() [3/3]

const CImg<T>& display ( 
    const char *const title = 0, 
    const bool display_info = true, 
    unsigned int *const XYZ = 0, 
    const bool exit_on_anykey = false) const

Display image into an interactive window.

Parameters

<table>
<thead>
<tr>
<th>title</th>
<th>Window title</th>
</tr>
</thead>
<tbody>
<tr>
<td>display_info</td>
<td>Tells if image information are displayed on the standard output.</td>
</tr>
<tr>
<td>in,out XYZ</td>
<td>Contains the XYZ coordinates at start / exit of the function.</td>
</tr>
<tr>
<td>exit_on_anykey</td>
<td>Exit function when any key is pressed.</td>
</tr>
</tbody>
</table>

8.1.4.498 display_object3d()

const CImg<T>& display_object3d ( 
    CImgDisplay & disp, 
    const CImg< tp > & vertices, 
    const CImgList< tf > & primitives, 
    const CImgList< tc > & colors, 
    const to & opacities, 
    const bool centering = true, 
    const int render_static = 4, 
    const int render_motion = 1, 
    const bool is_double_sided = true, 
    const float focale = 700, 
    const float light_x = 0, 
    const float light_y = 0, 
    const float light_z = -5e8f, 
    const float specular_lightness = 0.2f, 
    const float specular_intensity = 0.2f, 
    const float specular_lighting = 0.2f, 
    const float ambient_lighting = 0.2f, 
    const float ambient_lightness = 0.2f, 
    const int antialiasing = 1, 
    const int antialiasing2 = 1, 
    const float max_resolution = 100, 
    const float min_resolution = 10, 
    const float global_focal = 700, 
    const float global_light_x = 0, 
    const float global_light_y = 0, 
    const float global_light_z = -5e8f, 
    const float global_specular_lightness = 0.2f, 
    const float global_specular_intensity = 0.2f, 
    const float global_specular_lighting = 0.2f, 
    const float global_ambient_lighting = 0.2f, 
    const float global_ambient_lightness = 0.2f, 
    const int global_antialiasing = 1, 
    const int global_antialiasing2 = 1, 
    const float global_max_resolution = 100, 
    const float global_min_resolution = 10, 
    const int background = 0, 
    const int background_color = 0, 
    const int background_opacity = 0, 
    const int background_alpha = 0, 
    const int background_ambient_lighting = 0, 
    const int background_lighting = 0, 
    const int background_combination = 0, 
    const int background_opacities = 0, 
    const int background_colors = 0, 
    const int background_antialiasing = 0, 
    const int background_antialiasing2 = 0, 
    const float background_focal = 700, 
    const float background_light_x = 0, 
    const float background_light_y = 0, 
    const float background_light_z = -5e8f, 
    const float background_specular_lightness = 0.2f, 
    const float background_specular_intensity = 0.2f, 
    const float background_specular_lighting = 0.2f, 
    const float background_ambient_lighting = 0.2f, 
    const float background_ambient_lightness = 0.2f, 
    const int background_antialiasing = 0, 
    const int background_antialiasing2 = 0, 
    const float background_max_resolution = 100, 
    const float background_min_resolution = 10, 
    const int background_antialiasing = 0, 
    const int background_antialiasing2 = 0, 
    const float background_max_resolution = 100, 
    const float background_min_resolution = 10, 
    const int background_antialiasing = 0, 
    const int background_antialiasing2 = 0, 
    const float background_max_resolution = 100, 
    const float background_min_resolution = 10, 
    const int background_antialiasing = 0, 
    const int background_antialiasing2 = 0, 
    const float background_max_resolution = 100, 
    const float background_min_resolution = 10, 
    const int background_antialiasing = 0, 
    const int background_antialiasing2 = 0, 
    const float background_max_resolution = 100, 
    const float background_min_resolution = 10, 
    const int background_antialiasing = 0, 
    const int background_antialiasing2 = 0, 
    const float background_max_resolution = 100, 
    const float background_min_resolution = 10, 
    const int background_antialiasing = 0, 
    const int background_antialiasing2 = 0, 
    const float background_max_resolution = 100, 
    const float background_min_resolution = 10, 
    const int background_antialiasing = 0, 
    const int background_antialiasing2 = 0, 
    const float background_max_resolution = 100, 
    const float background_min_resolution = 10, 
    const int background_antialiasing = 0, 
    const int background_antialiasing2 = 0, 
    const float background_max_resolution = 100, 
    const float background_min_resolution = 10, 
    const int background_antialiasing = 0, 
    const int background_antialiasing2 = 0, 
    const float background_max_resolution = 100, 
    const float background_min_resolution = 10, 
    const int background_antiali
const float specular_shininess = 0.1f,
const bool display_axes = true,
float *const pose_matrix = 0,
const bool exit_on_anykey = false) const

Display object 3D in an interactive window.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>disp</td>
<td>Display window.</td>
</tr>
<tr>
<td>vertices</td>
<td>Vertices data of the 3D object.</td>
</tr>
<tr>
<td>primitives</td>
<td>Primitives data of the 3D object.</td>
</tr>
<tr>
<td>colors</td>
<td>Colors data of the 3D object.</td>
</tr>
<tr>
<td>opacities</td>
<td>Opacities data of the 3D object.</td>
</tr>
<tr>
<td>centering</td>
<td>Tells if the 3D object must be centered for the display.</td>
</tr>
<tr>
<td>render_static</td>
<td>Rendering mode.</td>
</tr>
<tr>
<td>render_motion</td>
<td>Rendering mode, when the 3D object is moved.</td>
</tr>
<tr>
<td>is_double_sided</td>
<td>Tells if the object primitives are double-sided.</td>
</tr>
<tr>
<td>focale</td>
<td>Focale</td>
</tr>
<tr>
<td>light_x</td>
<td>X-coordinate of the light source.</td>
</tr>
<tr>
<td>light_y</td>
<td>Y-coordinate of the light source.</td>
</tr>
<tr>
<td>light_z</td>
<td>Z-coordinate of the light source.</td>
</tr>
<tr>
<td>specular_lightness</td>
<td>Amount of specular light.</td>
</tr>
<tr>
<td>specular_shininess</td>
<td>Shininess of the object material.</td>
</tr>
<tr>
<td>display_axes</td>
<td>Tells if the 3D axes are displayed.</td>
</tr>
<tr>
<td>pose_matrix</td>
<td>Pointer to 12 values, defining a 3D pose (as a 4x3 matrix).</td>
</tr>
<tr>
<td>exit_on_anykey</td>
<td>Exit function when any key is pressed.</td>
</tr>
</tbody>
</table>

8.1.4.499  display_graph()

const CImg< T > & display_graph (  
    CImgDisplay & disp,  
    const unsigned int plot_type = 1,  
    const unsigned int vertex_type = 1,  
    const char *const labelx = 0,  
    const double xmin = 0,  
    const double xmax = 0,  
    const char *const labely = 0,  
    const double ymin = 0,  
    const double ymax = 0,  
    const bool exit_on_anykey = false ) const

Display 1D graph in an interactive window.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>disp</td>
<td>Display window.</td>
</tr>
<tr>
<td>plot_type</td>
<td>Plot type. Can be { 0=points</td>
</tr>
<tr>
<td>vertex_type</td>
<td>Vertex type.</td>
</tr>
</tbody>
</table>
Parameters

| labelx | Title for the horizontal axis, as a C-string. |
| xmin   | Minimum value along the X-axis. |
| xmax   | Maximum value along the X-axis. |
| labely | Title for the vertical axis, as a C-string. |
| ymin   | Minimum value along the X-axis. |
| ymax   | Maximum value along the X-axis. |
| exit_on_anykey | Exit function when any key is pressed. |

8.1.4.500  save()

```cpp
const CImg<T>& save(
    const char *const filename,
    const int number = -1,
    const unsigned int digits = 6 ) const
```

Save image as a file.

Parameters

| filename | Filename, as a C-string. |
| number   | When positive, represents an index added to the filename. Otherwise, no number is added. |
| digits   | Number of digits used for adding the number to the filename. |

Note

- The used file format is defined by the file extension in the `filename`.
- Parameter `number` can be used to add a 6-digit number to the filename before saving.

8.1.4.501  save_ascii()

```cpp
const CImg<T>& save_ascii(
    const char *const filename ) const
```

Save image as an Ascii file.

Parameters

| filename | Filename, as a C-string. |
### 8.1.4.502 save_cpp()

```cpp
const CImg\<T\>& save_cpp ( const char *const filename ) const
```

Save image as a `.cpp` source file.

#### Parameters

<table>
<thead>
<tr>
<th><code>filename</code></th>
<th>Filename, as a C-string.</th>
</tr>
</thead>
</table>

---

### 8.1.4.503 save_dlm()

```cpp
const CImg\<T\>& save_dlm ( const char *const filename ) const
```

Save image as a DLM file.

#### Parameters

<table>
<thead>
<tr>
<th><code>filename</code></th>
<th>Filename, as a C-string.</th>
</tr>
</thead>
</table>

---

### 8.1.4.504 save_bmp()

```cpp
const CImg\<T\>& save_bmp ( const char *const filename ) const
```

Save image as a BMP file.

#### Parameters

<table>
<thead>
<tr>
<th><code>filename</code></th>
<th>Filename, as a C-string.</th>
</tr>
</thead>
</table>

---

### 8.1.4.505 save_jpeg()

```cpp
const CImg\<T\>& save_jpeg ( const char *const filename, const unsigned int quality = 100 ) const
```

Save image as a JPEG file.
Parameters

<table>
<thead>
<tr>
<th>filename</th>
<th>Filename, as a C-string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>quality</td>
<td>Image quality (in %)</td>
</tr>
</tbody>
</table>

8.1.4.506 save_magick()

const CImg<T>& save_magick (  
  const char *const filename,  
  const unsigned int bytes_per_pixel = 0 ) const

Save image, using built-in ImageMagick++ library.

Parameters

<table>
<thead>
<tr>
<th>filename</th>
<th>Filename, as a C-string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>bytes_per_pixel</td>
<td>Force the number of bytes per pixel for the saving, when possible.</td>
</tr>
</tbody>
</table>

8.1.4.507 save_png()

const CImg<T>& save_png (  
  const char *const filename,  
  const unsigned int bytes_per_pixel = 0 ) const

Save image as a PNG file.

Parameters

<table>
<thead>
<tr>
<th>filename</th>
<th>Filename, as a C-string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>bytes_per_pixel</td>
<td>Force the number of bytes per pixels for the saving, when possible.</td>
</tr>
</tbody>
</table>

8.1.4.508 save_pnm()

const CImg<T>& save_pnm (  
  const char *const filename,  
  const unsigned int bytes_per_pixel = 0 ) const

Save image as a PNM file.

Parameters

<table>
<thead>
<tr>
<th>filename</th>
<th>Filename, as a C-string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>bytes_per_pixel</td>
<td>Force the number of bytes per pixels for the saving, when possible.</td>
</tr>
</tbody>
</table>
8.1.4.509  save_pnk()

const CImg<T>& save_pnk ( const char *const filename ) const

Save image as a PNK file.

Parameters

| filename | Filename, as a C-string. |

---

8.1.4.510  save_pfm()

const CImg<T>& save_pfm ( const char *const filename ) const

Save image as a PFM file.

Parameters

| filename | Filename, as a C-string. |

---

8.1.4.511  save_rgb()

const CImg<T>& save_rgb ( const char *const filename ) const

Save image as a RGB file.

Parameters

| filename | Filename, as a C-string. |

---

8.1.4.512  save_rgba()

const CImg<T>& save_rgba ( const char *const filename ) const

Save image as a RGBA file.
8.1.4.513  save_tiff()

const CImg<T>& save_tiff {
    const char *const filename,
    const unsigned int compression_type = 0,
    const float *const voxel_size = 0,
    const char * const description = 0,
    const bool use_bigtiff = true ) const

Save image as a TIFF file.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename, as a C-string.</td>
</tr>
<tr>
<td>compression_type</td>
<td>Type of data compression. Can be { 0=None</td>
</tr>
<tr>
<td>voxel_size</td>
<td>Voxel size, to be stored in the filename.</td>
</tr>
<tr>
<td>description</td>
<td>Description, to be stored in the filename.</td>
</tr>
<tr>
<td>use_bigtiff</td>
<td>Allow to save big tiff files (&gt;4Gb).</td>
</tr>
</tbody>
</table>

Note

• 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 types.
• If cimg_use_tif is not defined at compile time the function uses CImg<T>&save_other(const char*).

8.1.4.514  save_minc2()

const CImg<T>& save_minc2 {
    const char *const filename,
    const char *const imitate_file = 0 ) const

Save image as a MINC2 file.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename, as a C-string.</td>
</tr>
<tr>
<td>imitate_file</td>
<td>If non-zero, reference filename, as a C-string, to borrow header from.</td>
</tr>
</tbody>
</table>
8.1.4.515  save_analyze()

```cpp
const CImg<T>& save_analyze (  
    const char *const filename,  
    const float *const voxel_size = 0 ) const
```

Save image as an ANALYZE7.5 or NIFTI file.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename, as a C-string.</td>
</tr>
<tr>
<td>voxel_size</td>
<td>Pointer to 3 consecutive values that tell about the voxel sizes along the X,Y and Z dimensions.</td>
</tr>
</tbody>
</table>

8.1.4.516  save_cimg() [1/2]

```cpp
const CImg<T>& save_cimg (  
    const char *const filename,  
    const bool is_compressed = false ) const
```

Save image as a .cimg file.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename, as a C-string.</td>
</tr>
<tr>
<td>is_compressed</td>
<td>Tells if the file contains compressed image data.</td>
</tr>
</tbody>
</table>

8.1.4.517  save_cimg() [2/2]

```cpp
const CImg<T>& save_cimg (  
    const char *const filename,  
    const unsigned int n0,  
    const unsigned int x0,  
    const unsigned int y0,  
    const unsigned int z0,  
    const unsigned int c0 ) const
```

Save image as a sub-image into an existing .cimg file.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename, as a C-string.</td>
</tr>
<tr>
<td>n0</td>
<td>Index of the image inside the file.</td>
</tr>
<tr>
<td>x0</td>
<td>X-coordinate of the sub-image location.</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>y0</td>
<td>Y-coordinate of the sub-image location.</td>
</tr>
<tr>
<td>z0</td>
<td>Z-coordinate of the sub-image location.</td>
</tr>
<tr>
<td>c0</td>
<td>C-coordinate of the sub-image location.</td>
</tr>
</tbody>
</table>

8.1.4.518  save_empty_cimg() [1/2]

```cpp
static void save_empty_cimg (  
    const char *const filename,  
    const unsigned int dx,  
    const unsigned int dy = 1,  
    const unsigned int dz = 1,  
    const unsigned int dc = 1 ) [static]
```

Save blank image as a .cimg file.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename, as a C-string.</td>
</tr>
<tr>
<td>dx</td>
<td>Width of the image.</td>
</tr>
<tr>
<td>dy</td>
<td>Height of the image.</td>
</tr>
<tr>
<td>dz</td>
<td>Depth of the image.</td>
</tr>
<tr>
<td>dc</td>
<td>Number of channels of the image.</td>
</tr>
</tbody>
</table>

Note

- All pixel values of the saved image are set to 0.
- Use this method to save large images without having to instantiate and allocate them.

8.1.4.519  save_empty_cimg() [2/2]

```cpp
static void save_empty_cimg (  
    std::FILE *const file,  
    const unsigned int dx,  
    const unsigned int dy = 1,  
    const unsigned int dz = 1,  
    const unsigned int dc = 1 ) [static]
```

Save blank image as a .cimg file [overloading].

Same as `save_empty_cimg(const char *,unsigned int,unsigned int,unsigned int,unsigned int)` with a file stream argument instead of a filename string.
8.1.4.520  save_inr()

const CImg<T>& save_inr (const char *const filename, const float *const voxel_size = 0) const

Save image as an INRIMAGE-4 file.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename, as a C-string.</td>
</tr>
<tr>
<td>voxel_size</td>
<td>Pointer to 3 values specifying the voxel sizes along the X,Y and Z dimensions.</td>
</tr>
</tbody>
</table>

8.1.4.521  save_exr()

const CImg<T>& save_exr (const char *const filename) const

Save image as an OpenEXR file.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename, as a C-string.</td>
</tr>
</tbody>
</table>

Note

The OpenEXR file format is described [here](#).

8.1.4.522  save_pandore() [1/2]

const CImg<T>& save_pandore (const char *const filename, const unsigned int colorspace = 0) const

Save image as a Pandore-5 file.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename, as a C-string.</td>
</tr>
<tr>
<td>colorspace</td>
<td>Colorspace data field in output file (see Pandore file specifications for more information).</td>
</tr>
</tbody>
</table>
8.1.4.523  save_pandore()  [2/2]

const CImg<T>& save_pandore (  
    std::FILE *const file,  
    const unsigned int colorspace = 0 ) const

Save image as a Pandore-5 file [overloading].

Same as save_pandore(const char * unsiged int) const with a file stream argument instead of a filename string.

8.1.4.524  save_raw()  [1/2]

const CImg<T>& save_raw (  
    const char *const filename,  
    const bool is_multiplexed = false ) const

Save image as a raw data file.

Parameters

<table>
<thead>
<tr>
<th>filename</th>
<th>Filename, as a C-string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_multiplexed</td>
<td>Tells if the image channels are stored in a multiplexed way (true) or not (false).</td>
</tr>
</tbody>
</table>

Note

The .raw format does not store the image dimensions in the output file, so you have to keep track of them somewhere to be able to read the file correctly afterwards.

8.1.4.525  save_raw()  [2/2]

const CImg<T>& save_raw (  
    std::FILE *const file,  
    const bool is_multiplexed = false ) const

Save image as a raw data file [overloading].

Same as save_raw(const char *,bool) const with a file stream argument instead of a filename string.

8.1.4.526  save_yuv()  [1/2]

const CImg<T>& save_yuv (  
    const char *const filename,  
    const unsigned int chroma_subsampling = 444,  
    const bool is_rgb = true ) const

Save image as a .yuv video file.
8.1 CImg< T > Struct Template Reference  339

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename, as a C-string.</td>
</tr>
<tr>
<td>chroma_subsampling</td>
<td>Type of chroma subsampling. Can be { 420</td>
</tr>
<tr>
<td>is_rgb</td>
<td>Tells if pixel values of the instance image are RGB-coded (true) or YUV-coded (false).</td>
</tr>
</tbody>
</table>

Note

Each slice of the instance image is considered to be a single frame of the output video file.

8.1.4.527 save_yuv() [2/2]

```cpp
const CImg< T >& save_yuv(
    std::FILE *const file,
    const unsigned int chroma_subsampling = 444,
    const bool is_rgb = true ) const
```

Save image as a .yuv video file [overloading].

Same as save_yuv(const char *,const unsigned int,const bool) const with a file stream argument instead of a filename string.

8.1.4.528 save_off() [1/2]

```cpp
const CImg< T >& save_off(
    const CImgList< tf > & primitives,
    const CImgList< tc > & colors,
    const char *const filename ) const
```

Save 3D object as an Object File Format (.off) file.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename, as a C-string.</td>
</tr>
<tr>
<td>primitives</td>
<td>List of 3D object primitives.</td>
</tr>
<tr>
<td>colors</td>
<td>List of 3D object colors.</td>
</tr>
</tbody>
</table>

Note

- Instance image contains the vertices data of the 3D object.
- Textured, transparent or sphere-shaped primitives cannot be managed by the .off file format. Such primitives will be lost or simplified during file saving.
- The .off file format is described here.
8.1.4.529 **save_off()** [2/2]

```cpp
const CImg<T>& save_off(
    const CImgList<tf>& primitives,
    const CImgList<tc>& colors,
    std::FILE *const file ) const
```

Save 3D object as an Object File Format (.off) file [overloading].

Same as `save_off(const CImgList<tf>&, const CImgList<tc>&, const char* const file) const` with a file stream argument instead of a filename string.

8.1.4.530 **save_video()**

```cpp
const CImg<T>& save_video(
    const char* const filename,
    const unsigned int fps = 25,
    const char* const codec = 0,
    const bool keep_open = false ) const
```

Save volumetric image as a video, using the OpenCV library.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>filename</strong></td>
<td>Filename to write data to.</td>
</tr>
<tr>
<td><strong>fps</strong></td>
<td>Number of frames per second.</td>
</tr>
<tr>
<td><strong>codec</strong></td>
<td>Type of compression (See <a href="http://www.fourcc.org/codecs.php">http://www.fourcc.org/codecs.php</a> to see available codecs).</td>
</tr>
<tr>
<td><strong>keep_open</strong></td>
<td>Tells if the video writer associated to the specified filename must be kept open or not (to allow frames to be added in the same file afterwards).</td>
</tr>
</tbody>
</table>

8.1.4.531 **save_ffmpeg_external()**

```cpp
const CImg<T>& save_ffmpeg_external(
    const char* const filename,
    const unsigned int fps = 25,
    const char* const codec = 0,
    const unsigned int bitrate = 2048 ) const
```

Save volumetric image as a video, using ffmpeg external binary.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>filename</strong></td>
<td>Filename, as a C-string.</td>
</tr>
<tr>
<td><strong>fps</strong></td>
<td>Video framerate.</td>
</tr>
<tr>
<td><strong>codec</strong></td>
<td>Video codec, as a C-string.</td>
</tr>
<tr>
<td><strong>bitrate</strong></td>
<td>Video bitrate.</td>
</tr>
</tbody>
</table>
Note

- Each slice of the instance image is considered to be a single frame of the output video file.
- This method uses ffmpeg, an external executable binary provided by FFmpeg. It must be installed for the method to succeed.

8.1.4.532 save_gzip_external()

```cpp
const CImg<T>& save_gzip_external ( const char *const filename ) const
```

Save image using gzip external binary.

Parameters

| Filename | Filename, as a C-string. |

Note

This method uses gzip, an external executable binary provided by gzip. It must be installed for the method to succeed.

8.1.4.533 save_graphicsmagick_external()

```cpp
const CImg<T>& save_graphicsmagick_external ( const char *const filename, const unsigned int quality = 100 ) const
```

Save image using GraphicsMagick's external binary.

Parameters

| Filename | Filename, as a C-string. |
| Quality  | Image quality (expressed in percent), when the file format supports it. |

Note

This method uses gm, an external executable binary provided by GraphicsMagick. It must be installed for the method to succeed.

8.1.4.534 save_imagemagick_external()

```cpp
const CImg<T>& save_imagemagick_external ( const char *const filename, const unsigned int quality = 100 ) const
```
Save image using ImageMagick’s external binary.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename, as a C-string.</td>
</tr>
<tr>
<td>quality</td>
<td>Image quality (expressed in percent), when the file format supports it.</td>
</tr>
</tbody>
</table>

Note

This method uses `convert`, an external executable binary provided by ImageMagick. It must be installed for the method to succeed.

8.1.4.535 save_medcon_external()

```cpp
const CImg<T>& save_medcon_external ( const char *const filename ) const
```

Save image as a Dicom file.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename, as a C-string.</td>
</tr>
</tbody>
</table>

Note

This method uses `medcon`, an external executable binary provided by (X)Medcon. It must be installed for the method to succeed.

8.1.4.536 save_other()

```cpp
const CImg<T>& save_other ( const char *const filename, const unsigned int quality = 100 ) const
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename, as a C-string.</td>
</tr>
<tr>
<td>quality</td>
<td>Image quality (expressed in percent), when the file format supports it.</td>
</tr>
</tbody>
</table>

Note

- The filename extension tells about the desired file format.
- This method tries to save the instance image as a file, using external tools from ImageMagick or GraphicsMagick. At least one of these tool must be installed for the method to succeed.
• It is recommended to use the generic method save(const char∗, int) const instead, as it can handle some file formats natively.

8.1.4.537  get_serialize()

CImg<ucharT> get_serialize (  
    const bool is_compressed = false ) const

Serialize a CImg<T> instance into a raw CImg<unsigned char> buffer.

Parameters

| is_compressed  | tells if zlib compression must be used for serialization (this requires 'cimg_use_zlib' been enabled). |

8.2  CImgDisplay Struct Reference

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

Constructors / Destructor / Instance Management

• ~CImgDisplay ()
    Destructor.

• CImgDisplay ()
    Constructor 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)
    Construct 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)
  Construct 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)
  Construct a display from an image list.

• CImgDisplay (const CImgDisplay & disp)
  Construct 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)
    Construct a display with specified dimensions [in-place version].
• template<typename T>
  ClmgDisplay & assign (const Clmg< T >&img, const char *const title=0, const unsigned int normalization=3, const bool is_fullscreen=false, const bool is_closed=false)
  Construct a display from an image [in-place version].
• template<typename T>
  ClmgDisplay & assign (const ClmgList< T >&list, const char *const title=0, const unsigned int normalization=3, const bool is_fullscreen=false, const bool is_closed=false)
  Construct a display from an image list [in-place version].
• ClmgDisplay & assign (const ClmgDisplay &disp)
  Construct a display as a copy of another one [in-place version].
• template<typename T>
  static void screenshot (Clmg< T >&img)
  Take a screenshot.
• static ClmgDisplay & empty ()
  Return a reference to an empty display.
• static const ClmgDisplay & const_empty ()
  Return a reference to an empty display [const version].

Overloaded Operators

• template<typename t>
  ClmgDisplay & operator= (const Clmg< t >&img)
  Display image on associated window.
• template<typename t>
  ClmgDisplay & operator= (const ClmgList< t >&list)
  Display list of images on associated window.
• ClmgDisplay & operator= (const ClmgDisplay &disp)
  Construct a display as a copy of another one [in-place version].
• operator bool () const
  Return false if display is empty, true otherwise.

Instance Checking

• bool is_empty () const
  Return true if display is empty, false otherwise.
• bool is_closed () const
  Return true if display is closed (i.e. not visible on the screen), false otherwise.
• bool is_resized () const
  Return true if associated window has been resized on the screen, false otherwise.
• bool is_moved () const
  Return true if associated window has been moved on the screen, false otherwise.
• bool is_event () const
  Return true if any event has occurred on the associated window, false otherwise.
• bool is_fullscreen () const
  Return true if current display is in fullscreen mode, false otherwise.
• bool is_key () const
  Return true if any key is being pressed on the associated window, false otherwise.
• bool & is_key (const unsigned int keycode) const
  Return true if key specified by given keycode is being pressed on the associated window, false otherwise.
• bool & is_key (const char *const keycode)
Return true if key specified by given keycode is being pressed on the associated window, false otherwise.

- bool is_key_sequence (const unsigned int* const keycodes_sequence, const unsigned int length, const bool remove_sequence=false)

  Return true if specified key sequence has been typed on the associated window, false otherwise.

- bool is_keyESC () const

  Return true if the ESC key is being pressed on the associated window, false otherwise.

- 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
8.2 CImgDisplay Struct Reference

- 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.

- `CImgDisplay & move_inside_screen ()` 
  Display image on associated window.
- `CImgDisplay & display (const CImg< T > &img)` 
  Template parameter `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.
- `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.

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.

Generated by Doxygen
• 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 occurring on the current display.
• CImgDisplay & wait (const unsigned int milliseconds)
  Wait for a given number of milliseconds since the last call to wait().
• template<typename T>
  CImgDisplay & render (const CImg< T > &img)
  Render image into internal display buffer.
• CImgDisplay & paint ()
  Paint internal display buffer on associated window.
• template<typename T>
  const CImgDisplay & snapshot (CImg< T > &img) const
  Take a snapshot of the associated window content.
• static void wait (CImgDisplay &disp1)
  Wait for any event occurring on the display disp1.
• static void wait (CImgDisplay &disp1, CImgDisplay &disp2)
  Wait for any event occurring either on the display disp1 or disp2.
• static void wait (CImgDisplay &disp1, CImgDisplay &disp2, CImgDisplay &disp3)
  Wait for any event occurring either on the display disp1, disp2 or disp3.
• static void wait (CImgDisplay &disp1, CImgDisplay &disp2, CImgDisplay &disp3, CImgDisplay &disp4)
  Wait for any event occurring either on the display disp1, disp2, disp3 or disp4.
• static void wait (CImgDisplay &disp1, CImgDisplay &disp2, CImgDisplay &disp3, CImgDisplay &disp4, CImgDisplay &disp5)
  Wait for any event occurring either on the display disp1, disp2, disp3, disp4 or disp5.
8.2 CImgDisplay Struct Reference

- static void wait (CImgDisplay &disp1, CImgDisplay &disp2, CImgDisplay &disp3, CImgDisplay &disp4, CImgDisplay &disp5, CImgDisplay &disp6)
  
  *Wait for any event occurring either on the display disp1, disp2, disp3, disp4, disp5, disp6.*

- static void wait (CImgDisplay &disp1, CImgDisplay &disp2, CImgDisplay &disp3, CImgDisplay &disp4, CImgDisplay &disp5, CImgDisplay &disp6, CImgDisplay &disp7)
  
  *Wait for any event occurring either on the display disp1, disp2, disp3, disp4, disp5, disp6, disp7.*

- static void wait (CImgDisplay &disp1, CImgDisplay &disp2, CImgDisplay &disp3, CImgDisplay &disp4, CImgDisplay &disp5, CImgDisplay &disp6, CImgDisplay &disp7, CImgDisplay &disp8)
  
  *Wait for any event occurring either on the display disp1, disp2, disp3, disp4, disp5, disp6, disp7, disp8.*

- static void wait (CImgDisplay &disp1, CImgDisplay &disp2, CImgDisplay &disp3, CImgDisplay &disp4, CImgDisplay &disp5, CImgDisplay &disp6, CImgDisplay &disp7, CImgDisplay &disp8, CImgDisplay &disp9)
  
  *Wait for any event occurring either on the display disp1, disp2, disp3, disp4, disp5, disp6, disp7, disp8, disp9.*

- static void wait (CImgDisplay &disp1, CImgDisplay &disp2, CImgDisplay &disp3, CImgDisplay &disp4, CImgDisplay &disp5, CImgDisplay &disp6, CImgDisplay &disp7, CImgDisplay &disp8, CImgDisplay &disp9, CImgDisplay &disp10)
  
  *Wait for any event occurring either on the display disp1, disp2, disp3, disp4, disp5, disp6, disp7, disp8, disp9, disp10.*

- static void wait_all ()
  
  *Wait for any window event occurring in any opened CImgDisplay.*

- template<typename T>
  static void screenshot (const int x0, const int y0, const int x1, const int y1, CImg<T>& img)
  
  *Take a snapshot of the current screen content.*

### 8.2.1 Detailed Description

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

CImgDisplay 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 outputted 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

Generated by Doxygen
8.2.2.1 CImgDisplay()

\texttt{CImgDisplay()}  

Destructor.

\textbf{Note}  

If the associated window is visible on the screen, it is closed by the call to the destructor.

8.2.2.2 CImgDisplay() [1/5]

\texttt{CImgDisplay()}  

\textbf{Construct an empty display.}

\textbf{Note}  

Constructing an empty \texttt{CImgDisplay} instance does not make a window appearing on the screen, until display of valid data is performed.

\textbf{Example}  

\begin{verbatim}
    CImgDisplay disp;  // Does actually nothing
    ...
    disp.display(img);  // Construct new window and display image in it
\end{verbatim}

8.2.2.3 CImgDisplay() [2/5]

\texttt{CImgDisplay}  

\begin{verbatim}
    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
\end{verbatim}

\textbf{Construct a display with specified dimensions.}

\textbf{Parameters}  

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{width}</td>
<td>Window width.</td>
</tr>
<tr>
<td>\texttt{height}</td>
<td>Window height.</td>
</tr>
<tr>
<td>\texttt{title}</td>
<td>Window title.</td>
</tr>
<tr>
<td>\texttt{normalization}</td>
<td>Normalization type (0=none, 1=always, 2=once, 3=pixel type-dependent, see \texttt{normalization()}).</td>
</tr>
<tr>
<td>\texttt{is_fullscreen}</td>
<td>Tells if fullscreen mode is enabled.</td>
</tr>
<tr>
<td>\texttt{is_closed}</td>
<td>Tells if associated window is initially visible or not.</td>
</tr>
</tbody>
</table>
Note
A black background is initially displayed on the associated window.

8.2.2.4 CImgDisplay() [3/5]

CImgDisplay (const CImg< T >& img,
             const char *const title = 0,
             const unsigned int normalization = 3,
             const bool is_fullscreen = false,
             const bool is_closed = false) [explicit]

Construct a display from an image.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>img</td>
<td>Image used as a model to create the window.</td>
</tr>
<tr>
<td>title</td>
<td>Window title.</td>
</tr>
<tr>
<td>normalization</td>
<td>Normalization type (0=none, 1=always, 2=once, 3=pixel type-dependent, see normalization()).</td>
</tr>
<tr>
<td>is_fullscreen</td>
<td>Tells if fullscreen mode is enabled.</td>
</tr>
<tr>
<td>is_closed</td>
<td>Tells if associated window is initially visible or not.</td>
</tr>
</tbody>
</table>

Note
The pixels of the input image are initially displayed on the associated window.

8.2.2.5 CImgDisplay() [4/5]

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]

Construct a display from an image list.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>The images list to display.</td>
</tr>
<tr>
<td>title</td>
<td>Window title.</td>
</tr>
<tr>
<td>normalization</td>
<td>Normalization type (0=none, 1=always, 2=once, 3=pixel type-dependent, see normalization()).</td>
</tr>
<tr>
<td>is_fullscreen</td>
<td>Tells if fullscreen mode is enabled.</td>
</tr>
<tr>
<td>is_closed</td>
<td>Tells if associated window is initially visible or not.</td>
</tr>
</tbody>
</table>

Generated by Doxygen
Note

All images of the list, appended along the X-axis, are initially displayed on the associated window.

### 8.2.2.6 CImgDisplay()

\begin{verbatim}
CImgDisplay {
    const CImgDisplay & disp }
\end{verbatim}

Construct 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.

### 8.2.3 Member Function Documentation

#### 8.2.3.1 screenshot() [1/2]

\begin{verbatim}
static void screenshot {
    CImg< T > & img } [static]
\end{verbatim}

Take a screenshot.

**Parameters**

- **out** \textit{img} Output screenshot. Can be empty on input

#### 8.2.3.2 assign()

\begin{verbatim}
CImgDisplay& assign ( )
\end{verbatim}

Destructor - Empty constructor [in-place version].

**Note**

Replace the current instance by an empty display.
8.2.3.3 empty()

static CImgDisplay& empty () [static]

Return a reference to an empty display.

Note
Can be useful for writing function prototypes where one of the argument (of type CImgDisplay&) must have a
default value.

Example

```c
void foo(CImgDisplay& disp=CImgDisplay::empty());
```

8.2.3.4 operator=() [1/3]

CImgDisplay& operator= (const CImg< t >& img )

Display image on associated window.

Note

disp = img is equivalent to disp.display(img).

8.2.3.5 operator=() [2/3]

CImgDisplay& operator= (const CImgList< t >& list )

Display list of images on associated window.

Note

disp = list is equivalent to disp.display(list).

8.2.3.6 operator=() [3/3]

CImgDisplay& operator= (const CImgDisplay & disp )

Construct a display as a copy of another one [in-place version].

Note

Equivalent to assign(const CImgDisplay&).
8.2.3.7 operator bool()

operator bool ( ) const

Return \textit{false} if display is empty, \textit{true} otherwise.

\textbf{Note}

\begin{verbatim}
if (disp) { ... } is equivalent to if (!disp.is_empty()) { ... }.
\end{verbatim}

8.2.3.8 is_closed()

bool is_closed ( ) const

Return \textit{true} if display is closed (i.e. not visible on the screen), \textit{false} otherwise.

\textbf{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 \texttt{show()}.

8.2.3.9 is_key() [1/3]

bool is_key ( ) const

Return \textit{true} if any key is being pressed on the associated window, \textit{false} otherwise.

\textbf{Note}

The methods below do the same only for specific keys.

8.2.3.10 is_key() [2/3]

bool is_key ( const unsigned int keycode ) const

Return \textit{true} if key specified by given keycode is being pressed on the associated window, \textit{false} otherwise.

\textbf{Parameters}

- \texttt{keycode} Keycode to test.
8.2 CImgDisplay Struct Reference

Note

Keycode constants are defined in the cimg namespace and are architecture-dependent. Use them to ensure your code stay portable (see \texttt{cimg::keyESC}).

Example

\begin{verbatim}
CImgDisplay disp(400,400);
while (!disp.is_closed()) {
    if (disp.key(cimg::keyTAB)) { ... } // Equivalent to 'if (disp.is_keyTAB())'
    disp.wait();
}
\end{verbatim}

8.2.3.11 is_key() [3/3]

\begin{verbatim}
bool is_key {
    const char *const keycode
}
\end{verbatim}

\textbf{Return} true if key specified by given keycode is being pressed on the associated window, false otherwise.

\textbf{Parameters}

| \textbf{keycode} | C-string containing the keycode label of the key to test. |

\textbf{Note}

Use it when the key you want to test can be dynamically set by the user.

Example

\begin{verbatim}
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();
}
\end{verbatim}

8.2.3.12 is_key_sequence()

\begin{verbatim}
bool is_key_sequence {
    const unsigned int *const keycodes_sequence,
    const unsigned int length,
    const bool remove_sequence = false
}
\end{verbatim}

\textbf{Return} true if specified key sequence has been typed on the associated window, false otherwise.

\textbf{Parameters}

| \textbf{keycodes_sequence} | Buffer of keycodes to test. |
| \textbf{length} | Number of keys in the keycodes_sequence buffer. |
| \textbf{remove_sequence} | Tells if the key sequence must be removed from the key history, if found. |

Generated by Doxygen
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

```cpp
CImgDisplay disp(400,400);
const unsigned int key_seq[] = { cimg::keyCTRLLEFT, cimg::keyD };
while (!disp.is_closed()) {
    if (disp.is_key_sequence(key_seq,2)) { ... } // Test for the 'CTRL+D' keyboard event
    disp.wait();
}
```

8.2.3.13 is_keyESC()

```cpp
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).

8.2.3.14 width()

```cpp
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.

8.2.3.15 height()

```cpp
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.
### 8.2.3.16 normalization()

```c
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.

### 8.2.3.17 title()

```c
const char* title() const
```

Return title of the associated window as a C-string.

**Note**

Window title may be not visible, depending on the used window manager or if the current display is in fullscreen mode.

### 8.2.3.18 window_width()

```c
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.
8.2.3.19  window_height()

int window_height ( ) const

Return height of the associated window.

Note
The height of the display (i.e. the height of the pixel data buffer associated to the CImgDisplay instance) may be different from the actual height of the associated window.

8.2.3.20  window_x()

int window_x ( ) const

Return X-coordinate of the associated window.

Note
The returned coordinate corresponds to the location of the upper-left corner of the associated window.

8.2.3.21  window_y()

int window_y ( ) const

Return Y-coordinate of the associated window.

Note
The returned coordinate corresponds to the location of the upper-left corner of the associated window.

8.2.3.22  mouse_x()

int mouse_x ( ) const

Return X-coordinate of the mouse pointer.

Note
• If the mouse pointer is outside window area, −1 is returned.
• Otherwise, the returned value is in the range [0,width()-1].
8.2.3.23  mouse_y()

int mouse_y ( ) const

Return Y-coordinate of the mouse pointer.

Note

• If the mouse pointer is outside window area, −1 is returned.
• Otherwise, the returned value is in the range [0, height()-1].

8.2.3.24  button()

unsigned int button ( ) const

Return current state of the mouse buttons.

Note

Three mouse buttons can be managed. If one button is pressed, its corresponding bit in the returned value is set:

• bit 0 (value 0x1): State of the left mouse button.
• bit 1 (value 0x2): State of the right mouse button.
• bit 2 (value 0x4): State of the middle mouse button.

Several bits can be activated if more than one button are pressed at the same time.

Example

CImgDisplay disp(400,400);
while (!disp.is_closed()) {
    if (disp.button()&1) { // Left button clicked
        ...
    }
    if (disp.button()&2) { // Right button clicked
        ...
    }
    if (disp.button()&4) { // Middle button clicked
        ...
    }
    disp.wait();
}
8.2.3.25 wheel()

```c
int wheel ( ) const
```

Return current state of the mouse wheel.

**Note**

- The returned value can be positive or negative depending on whether the mouse wheel has been scrolled forward or backward.
- Scrolling the wheel forward add 1 to the wheel value.
- Scrolling the wheel backward subtract 1 to the wheel value.
- The returned value cumulates the number of forward or backward scrolls since the creation of the display, or since the last reset of the wheel value (using set_wheel()). It is strongly recommended to quickly reset the wheel counter when an action has been performed regarding the current wheel value. Otherwise, the returned wheel value may be for instance 0 despite the fact that many scrolls have been done (as many in forward as in backward directions).

**Example**

```c
CImgDisplay disp(400,400);
while (!disp.is_closed()) {
    if (disp.wheel()) {
        int counter = disp.wheel(); // Read the state of the mouse wheel
        ... // Do what you want with 'counter'
        disp.set_wheel(); // Reset the wheel value to 0
    }
    disp.wait();
}
```

8.2.3.26 key()

```c
unsigned int key ( const unsigned int pos = 0 ) const
```

Return one entry from the pressed keys history.

**Parameters**

- **pos**: Index to read from the pressed keys history (index 0 corresponds to latest entry).

**Returns**

- Keycode of a pressed key or 0 for a released key.

**Note**

- Each CImgDisplay stores a history of the pressed keys in a buffer of size 128. When a new key is pressed, its keycode is stored in the pressed keys history. When a key is released, 0 is put instead. This means that up to the 64 last pressed keys may be read from the pressed keys history. When a new value is stored, the pressed keys history is shifted so that the latest entry is always stored at position 0.
• Keycode constants are defined in the cimg namespace and are architecture-dependent. Use them to ensure your code stay portable (see cimg::keyESC).

8.2.3.27 released_key()

unsigned int released_key (  
    const unsigned int pos = 0 ) const

Return one entry from the released keys history.

Parameters

| pos | Index to read from the released keys history (index 0 corresponds to latest entry). |

Returns

Keycode of a released key or 0 for a pressed key.

Note

• Each CImgDisplay stores a history of the released keys in a buffer of size 128. When a new key is released, its keycode is stored in the pressed keys history. When a key is pressed, 0 is put instead. This means that up to the 64 last released keys may be read from the released keys history. When a new value is stored, the released keys history is shifted so that the latest entry is always stored at position 0.

• Keycode constants are defined in the cimg namespace and are architecture-dependent. Use them to ensure your code stay portable (see cimg::keyESC).

8.2.3.28 keycode()

static unsigned int keycode (  
    const char ∗const keycode ) [static]

Return keycode corresponding to the specified string.

Note

Keycode constants are defined in the cimg namespace and are architecture-dependent. Use them to ensure your code stay portable (see cimg::keyESC).

Example

    const unsigned int keyTAB = CImgDisplay::keycode("TAB"); // Return cimg::keyTAB
8.2.3.29 frames_per_second()

float frames_per_second ( )

Return the current refresh rate, in frames per second.

Note

Returns a significant value when the current instance is used to display successive frames. It measures the delay between successive calls to frames_per_second().

8.2.3.30 display() [1/2]

CImgDisplay& display ( 
    const CImg< T > & img )

Display image on associated window.

Parameters

| *img* | Input image to display. |

Note

This method returns immediately.

8.2.3.31 display() [2/2]

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). |
8.2.3.32 show()

CImgDisplay show()

Show (closed) associated window on the screen.

Note
• Force the associated window of a display to be visible on the screen, even if it has been closed before.
• Using show() on a visible display does nothing.

8.2.3.33 close()

CImgDisplay close()

Close (visible) associated window and make it disappear from the screen.

Note
• A closed display only means the associated window is not visible anymore. This does not mean the display has been destroyed. Use show() to make the associated window reappear.
• Using close() on a closed display does nothing.

8.2.3.34 move()

CImgDisplay move(const int pos_x, const int pos_y)

Move associated window to a new location.

Parameters

<table>
<thead>
<tr>
<th>pos→_x</th>
<th>X-coordinate of the new window location.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos→_y</td>
<td>Y-coordinate of the new window location.</td>
</tr>
</tbody>
</table>

Generated by Doxygen
Depending on the window manager behavior, this method may not succeed (no exceptions are thrown nevertheless).

**8.2.3.35 resize() [1/4]**

```cpp
CImgDisplay& resize (
    const bool force_redraw = true )
```

Resize display to the size of the associated window.

**Parameters**

<table>
<thead>
<tr>
<th>force_redraw</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tells if the previous window content must be updated and refreshed as well.</td>
</tr>
</tbody>
</table>

**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.

**8.2.3.36 resize() [2/4]**

```cpp
CImgDisplay& resize ( 
    const int width,    
    const int height,   
    const bool force_redraw = true )
```

Resize display to the specified size.

**Parameters**

<table>
<thead>
<tr>
<th>width</th>
<th>height</th>
<th>force_redraw</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requested display width.</td>
<td>Requested display height.</td>
<td>Tells if the previous window content must be updated and refreshed as well.</td>
</tr>
</tbody>
</table>

**Note**

The associated window is also resized to specified dimensions.
8.2.3.37  resize() [3/4]

CImgDisplay& resize (  
    const CImg<T> & img,  
    const bool force_redraw = true  
)

Resize display to the size of an input image.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>img</strong></td>
<td>Input image to take size from.</td>
</tr>
<tr>
<td><strong>force_redraw</strong></td>
<td>Tells if the previous window content must be resized and updated as well.</td>
</tr>
</tbody>
</table>

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.

8.2.3.38  resize() [4/4]

CImgDisplay& resize (  
    const CImgDisplay & disp,  
    const bool force_redraw = true  
)

Resize display to the size of another CImgDisplay instance.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>disp</strong></td>
<td>Input display to take size from.</td>
</tr>
<tr>
<td><strong>force_redraw</strong></td>
<td>Tells if the previous window content must be resized and updated as well.</td>
</tr>
</tbody>
</table>

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.

8.2.3.39  set_normalization()

CImgDisplay& set_normalization (  
    const unsigned int normalization  
)

Set normalization type.
Parameters

| normalization | New normalization mode. |

8.2.3.40 set_title()

```cpp
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.

8.2.3.41 set_fullscreen()

```cpp
cImgDisplay set_fullscreen {
    const bool is_fullscreen,
    const bool force_redraw = true }
```

Enable or disable fullscreen mode.

Parameters

| is_fullscreen | Tells is the fullscreen mode must be activated or not. |
| force_redraw  | Tells 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).
8.2.3.42 toggle_fullscreen()

`CImgDisplay& toggle_fullscreen (const bool force_redraw = true)`

Toggle fullscreen mode.

**Parameters**

- `force_redraw` Tells if the previous window content must be displayed as well.

**Note**

Enable fullscreen mode if it was not enabled, and disable it otherwise.

8.2.3.43 show_mouse()

`CImgDisplay& show_mouse ( )`

Show mouse pointer.

**Note**

Depending on the window manager behavior, this method may not succeed (no exceptions are thrown nevertheless).

8.2.3.44 hide_mouse()

`CImgDisplay& hide_mouse ( )`

Hide mouse pointer.

**Note**

Depending on the window manager behavior, this method may not succeed (no exceptions are thrown nevertheless).
8.2.3.45  set_mouse()

CImgDisplay& set_mouse (const int pos_x, const int pos_y)

Move mouse pointer to a specified location.

Note
Depending on the window manager behavior, this method may not succeed (no exceptions are thrown nevertheless).

8.2.3.46  set_button() [1/2]

CImgDisplay set_button ( )

Simulate a mouse button release event.

Note
All mouse buttons are considered released at the same time.

8.2.3.47  set_button() [2/2]

CImgDisplay set_button (const unsigned int button, const bool is_pressed = true)

Simulate a mouse button press or release event.

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>button</td>
<td>Buttons event code, where each button is associated to a single bit.</td>
</tr>
<tr>
<td>is_pressed</td>
<td>Tells if the mouse button is considered as pressed or released.</td>
</tr>
</tbody>
</table>

8.2.3.48  set_wheel() [1/2]

CImgDisplay set_wheel ( )

Flush all mouse wheel events.
8.2 CImgDisplay Struct Reference

Note
Make \texttt{wheel()} to return 0, if called afterwards.

8.2.3.49 set_wheel [2/2]

\begin{verbatim}
CImgDisplay::set_wheel {
    const int amplitude
}
\end{verbatim}

Simulate a wheel event.

Parameters

\begin{center}
\begin{tabular}{|l|p{0.5\textwidth}|}
\hline
\textit{amplitude} & Amplitude of the wheel scrolling to simulate. \\
\hline
\end{tabular}
\end{center}

Note
Make \texttt{wheel()} to return amplitude, if called afterwards.

8.2.3.50 set_key [1/2]

\begin{verbatim}
CImgDisplay::set_key()
\end{verbatim}

Flush all key events.

Note
Make \texttt{key()} to return 0, if called afterwards.

8.2.3.51 set_key [2/2]

\begin{verbatim}
CImgDisplay::set_key {
    const unsigned int keycode,
    const bool is_pressed = true
}
\end{verbatim}

Simulate a keyboard press/release event.

Parameters

\begin{center}
\begin{tabular}{|l|p{0.5\textwidth}|}
\hline
\textit{keycode} & Keycode of the associated key. \\
\hline
\textit{is_pressed} & Tells if the key is considered as pressed or released. \\
\hline
\end{tabular}
\end{center}
Note

Keycode constants are defined in the cimg namespace and are architecture-dependent. Use them to ensure your code stay portable (see cimg::keyESC).

8.2.3.52 flush()

CImgDisplay flush();

Flush all display events.

Note

Remove all passed events from the current display.

8.2.3.53 wait()

CImgDisplay wait(const unsigned int milliseconds);

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

Parameters

| milliseconds | Number of milliseconds to wait for. |

Note

Similar to cimg::wait().

8.2.3.54 render()

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.

### 8.2.3.55 paint()

```cpp
CImgDisplay& paint ()
```

Paint internal display buffer on associated window.

Note

- Update the content of the associated window with the internal display buffer, e.g. after a `render()` call.
- Should not be used for common `CImgDisplay` uses, since `display()` is more useful.

### 8.2.3.56 screenshot() [2/2]

```cpp
static void screenshot (
    const int x0,
    const int y0,
    const int x1,
    const int y1,
    CImg< T >& img ) [static]
```

Take a snapshot of the current screen content.

**Parameters**

<table>
<thead>
<tr>
<th>x0</th>
<th>X-coordinate of the upper left corner.</th>
</tr>
</thead>
<tbody>
<tr>
<td>y0</td>
<td>Y-coordinate of the upper left corner.</td>
</tr>
<tr>
<td>x1</td>
<td>X-coordinate of the lower right corner.</td>
</tr>
<tr>
<td>y1</td>
<td>Y-coordinate of the lower right corner.</td>
</tr>
<tr>
<td>out</td>
<td><code>img</code></td>
</tr>
</tbody>
</table>

### 8.2.3.57 snapshot()

```cpp
const CImgDisplay& snapshot ( CImg< T > & img ) const
```

Take a snapshot of the associated window content.
Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>img</th>
</tr>
</thead>
</table>

Output snapshot. Can be empty on input.

8.3 CImgException Struct Reference

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

Inherits `exception`.

Inherited by `CImgArgumentException`, `CImgDisplayException`, `CImgInstanceException`, `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` (except `CImgAbortException`). `CImgException` is never thrown itself. Derived classes that specify the type of error are thrown instead. These classes can be:

- **CImgAbortException**: Thrown when a computationally-intensive function is aborted by an external signal. This is the only non-derived exception class.

- **CImgArgumentException**: Thrown when one argument of a called `CImg` function is invalid. This is probably one of the most 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 `CImg` 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 occurred when trying to load or save image files. This happens when trying to read files that do not exist or with invalid formats. For instance, the following example throws a `CImgIOException`:

  ```
  ```

Generated by Doxygen
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:

```cpp
#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 CImg
    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.

- **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

- \(~\text{CImgList}()\)
  Destructor.

- \text{CImgList}()\)
  Default constructor.

- \text{CImgList}(\text{const unsigned int } n)\)
  Construct list containing empty images.

- \text{CImgList}(\text{const unsigned int } n, \text{const unsigned int } width, \text{const unsigned int } height=1, \text{const unsigned int } depth=1, \text{const unsigned int } spectrum=1)\)
  Construct list containing images of specified size.

- \text{CImgList}(\text{const unsigned int } n, \text{const unsigned int } width, \text{const unsigned int } height, \text{const unsigned int } depth, \text{const unsigned int } spectrum, \text{const T &val})\)
  Construct list containing images of specified size, and initialize pixel values.

- \text{CImgList}(\text{const unsigned int } n, \text{const unsigned int } width, \text{const unsigned int } height, \text{const unsigned int } depth, \text{const unsigned int } spectrum, \text{const double } val0, \text{const double } val1,...)\)
  Construct list containing images of specified size, and initialize pixel values from a sequence of doubles.

- \text{CImgList}(\text{const unsigned int } n, \text{const unsigned int } width, \text{const unsigned int } height, \text{const unsigned int } depth, \text{const unsigned int } spectrum, \text{const T } &\text{val}0, \text{const T } &\text{val}1,...)\)
  Construct list containing images of specified size, and initialize pixel values from a sequence of integers.

- \text{CImgList}(\text{const unsigned int } n, \text{const unsigned int } width, \text{const unsigned int } height, \text{const unsigned int } depth, \text{const unsigned int } spectrum, \text{const CImg } t &\text{img}, \text{const bool } is\text{_shared}=false)\)
  Construct list containing copies of an input image.

- \text{CImgList}(\text{const CImg } t &\text{img}, \text{const bool } is\text{_shared}=false)\)
  Construct list from one image.

- \text{CImgList}(\text{const CImg } t &\text{img}1, \text{const CImg } t &\text{img}2, \text{const bool } is\text{_shared}=false)\)
  Construct list from two images.

- \text{CImgList}(\text{const CImg } t &\text{img}1, \text{const CImg } t &\text{img}2, \text{const CImg } t &\text{img}3, \text{const CImg } t &\text{img}4, \text{const bool } is\text{_shared}=false)\)
  Construct list from three images.

- \text{CImgList}(\text{const CImg } t &\text{img}1, \text{const CImg } t &\text{img}2, \text{const CImg } t &\text{img}3, \text{const CImg } t &\text{img}4, \text{const CImg } t &\text{img}5, \text{const bool } is\text{_shared}=false)\)
  Construct list from four images.

- \text{CImgList}(\text{const CImg } t &\text{img}1, \text{const CImg } t &\text{img}2, \text{const CImg } t &\text{img}3, \text{const CImg } t &\text{img}4, \text{const CImg } t &\text{img}5, \text{const CImg } t &\text{img}6, \text{const bool } is\text{_shared}=false)\)
  Construct list from five images.

- \text{CImgList}(\text{const CImg } t &\text{img}1, \text{const CImg } t &\text{img}2, \text{const CImg } t &\text{img}3, \text{const CImg } t &\text{img}4, \text{const CImg } t &\text{img}5, \text{const CImg } t &\text{img}6, \text{const CImg } t &\text{img}7, \text{const bool } is\text{_shared}=false)\)
  Construct list from six images.

- \text{CImgList}(\text{const CImg } t &\text{img}1, \text{const CImg } t &\text{img}2, \text{const CImg } t &\text{img}3, \text{const CImg } t &\text{img}4, \text{const CImg } t &\text{img}5, \text{const CImg } t &\text{img}6, \text{const CImg } t &\text{img}7, \text{const CImg } t &\text{img}8, \text{const bool } is\text{_shared}=false)\)
  Construct list from seven images.
Construct list from eight images.

- `template<typename t >
  
  CImgList (const CImgList< t >&list)

  Construct list copy.

- `CImgList (const CImgList< T >&list)

  Construct list copy [specialization].

- `template<typename t >
  
  CImgList (const CImgList< t >&list, const bool is_shared)

  Construct list copy, and force the shared state of the list elements.

- `CImgList (const char *const filename)

  Construct list by reading the content of a file.

- `CImgList (const CImgDisplay &disp)

  Construct 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 ()

  Destructor [in-place version].

- `CImgList< T > & clear ()

  Destructor [in-place version].

- `CImgList< T > & assign (const unsigned int n)

  Construct list containing 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)

  Construct list containing images of specified size [in-place version].

- `CImgList< T > & assign (const unsigned int n, const unsigned int width, const unsigned int height, const unsigned int depth, const unsigned int spectrum, const T &val)

  Construct list containing images of specified size, and initialize pixel values [in-place version].

- `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 &val0, const int val1,...)

  Construct list with images of 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,...)

  Construct list with images of 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)

  Construct list containing copies of an input image [in-place version].

- `template<typename t >
  
  CImgList< T > & assign (const CImg< t >&img, const bool is_shared=false)

  Construct list from one image [in-place version].

- `template<typename t1 , typename t2 >
  
  CImgList< T > & assign (const CImg< t1 >&img1, const CImg< t2 >&img2, const bool is_shared=false)

  Construct list from two images [in-place version].

- `template<typename t1 , typename t2 , typename t3 >
  
  CImgList< T > & assign (const CImg< t1 >&img1, const CImg< t2 >&img2, const CImg< t3 >&img3, const bool is_shared=false)

  Construct list from three images [in-place version].

- `template<typename t1 , typename t2 , typename t3 , typename t4 >
  
  CImgList< T > & assign (const CImg< t1 >&img1, const CImg< t2 >&img2, const CImg< t3 >&img3, const CImg< t4 >&img4, const bool is_shared=false)

  Construct list from four images [in-place version].
• \texttt{ClmgList< T & operator() (const unsigned int pos, const unsigned int x, const unsigned int y=0, const unsigned int z=0, const bool is\_shared=false)}

Construct list from five images \texttt{[\texttt{in\_place\_version}].}

• \texttt{ClmgList< T & operator() (const unsigned int pos, const unsigned int x, const unsigned int y=0, const unsigned int z=0, const CImg< T & operator() (const unsigned int pos, const unsigned int x, const unsigned int y=0, const unsigned int z=0, const bool is\_shared=false)}

Construct list from six images \texttt{[\texttt{in\_place\_version}].}

• \texttt{ClmgList< T & operator() (const unsigned int pos, const unsigned int x, const unsigned int y=0, const unsigned int z=0, const CImgList< T & operator() (const unsigned int pos, const unsigned int x, const unsigned int y=0, const unsigned int z=0, const bool is\_shared=false)}

Construct list from seven images \texttt{[\texttt{in\_place\_version}].}

• \texttt{ClmgList< T & operator() (const unsigned int pos, const unsigned int x, const unsigned int y=0, const unsigned int z=0, const CImgList< T & operator() (const unsigned int pos, const unsigned int x, const unsigned int y=0, const unsigned int z=0, const bool is\_shared=false)}

Construct list from eight images \texttt{[\texttt{in\_place\_version}].}

• \texttt{ClmgList< T & operator() (const unsigned int pos)}

Construct list as a copy of an existing list and force the shared state of the list elements \texttt{[\texttt{in\_place\_version}].}

• \texttt{ClmgList< T & operator() (const unsigned int pos)}

Construct list as a copy of an existing list and force shared state of elements \texttt{[\texttt{in\_place\_version}] [\texttt{specialization}].}

• \texttt{ClmgList< T & operator() (const char *const filename)}

Construct list by reading the content of a file \texttt{[\texttt{in\_place\_version}].}

• \texttt{ClmgList< T & operator() (const CImgDisplay &disp)}

Construct list from the content of a display window \texttt{[\texttt{in\_place\_version}].}

• \texttt{ClmgList< T & operator() (ClmgList< T & list)}

Transfer the content of the list instance to another list.

• \texttt{ClmgList< T & operator() (ClmgList< T & list, const unsigned int pos)}

Transfer the content of the list instance at a specified position in another list.

• \texttt{ClmgList< T & operator() (ClmgList< T & list)}

Swap all fields between two list instances.

• \texttt{static CImgList< T & operator() (const CImg & img1, const CImg & img2, const bool is\_shared=false)}

Return a reference to one image element of the list.

• \texttt{static CImgList< T & operator() (const CImg & img1, const CImg & img2, const bool is\_shared=false)}

Return a reference to one pixel value of one image of the list.

• \texttt{static CImgList< T & operator() (const CImg & img1, const CImg & img2, const bool is\_shared=false)}

Return a reference to one pixel value of one image of the list \texttt{[\texttt{const\_version}].}

### Overloaded Operators

• \texttt{Clmg< T & operator() (const unsigned int pos)}

\texttt{Return a reference to one image element of the list.}

• \texttt{const Clmg< T & operator() (const unsigned int pos)}

\texttt{Return a reference to one image of the list.}

• \texttt{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)}

\texttt{Return a reference to one pixel value of one image of the list.}

• \texttt{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)}

\texttt{Return a reference to one pixel value of one image of the list \texttt{[\texttt{const\_version}].}}
• `operator CImg<T> *()`
  Return pointer to the first image of the list.

• `operator const CImg<T> *()` const
  Return pointer to the first image of the list [const version].

• `template<typename t>
  CImgList<T> & operator= (const CImg<T> &img)`
  Construct list from one image [in-place version].

• `template<
t>
  CImgList<T> & operator= (const CImgList<T> &list)`
  Construct list from another list.

• `CImgList<T> & operator= (const CImgList<T> &list)` const
  Construct list from another list [specialization].

• `CImgList<T> & operator= (const char *const filename)`
  Construct list by reading the content of a file [in-place version].

• `CImgList<T> & operator= (const CImgDisplay &disp)`
  Construct list from the content of a display window [in-place version].

• `CImgList<T> & operator+ () const`
  Return a non-shared copy of a list.

• `template<
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<
t>
  CImgList<T> & operator, (const CImg<T> &img) const`
  Return a copy of the list instance, where image img has been inserted at the end [const version].

• `template<
t>
  CImgList<T> & operator, (const CImgList<T> &list)`
  Return a copy of the list instance, where all elements of input list list have been inserted at the end.

• `template<
t>
  CImgList<T> & operator, (const CImgList<T> &list) const`
  Return a copy of the list instance, where all elements of input list list have been inserted at the end [const version].

• `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& 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 Neumann boundary conditions.

- 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 Neumann boundary conditions [const version].

- T& atNXY(const int pos, const int x, const int y, const int z, const int c, const T& out_value)
  Access to pixel value with Dirichlet boundary conditions for the 3 coordinates (pos, x, y, z).

- T atNXY(const int pos, const int x, const int y, const int z, const int c, const T& out_value) const
  Access to pixel value with Dirichlet boundary conditions for the 3 coordinates (pos, x, y, z) [const version].

- T& atN(const int pos, const int x, const int y, const int z, const int c, const T& out_value)
  Access to pixel value with Dirichlet boundary conditions for the coordinate (pos).

- T atN(const int pos, const int x, const int y, const int z, const int c, const T& out_value) const
  Access to pixel value with Dirichlet boundary conditions for the coordinate (pos) [const version].
8.4 ClmgList＜T＞ Struct Template Reference

- T & atN (const int pos, const int x=0, const int y=0, const int z=0, const int c=0)
  
  *Return pixel value with Neumann boundary conditions for the coordinate (pos).*

- T atN (const int pos, const int x=0, const int y=0, const int z=0, const int c=0) const
  
  *Return pixel value with Neumann boundary conditions for the coordinate (pos) [const version].*

- static const char * pixel_type ()
  
  *Return the type of image pixel values as a C string.*

### Instance Checking

- bool is_empty () const
  
  *Return true if list is empty.*

- bool is_sameN (const unsigned int size_n) const
  
  *Test if number of image elements is equal to specified value.*

  template<typename t >
  bool is_sameN (const ClmgList＜t＞ &list) const

  *Test if number of image elements is equal between two images lists.*

- template<typename t >
  bool is_sameXY (const Clmg＜t＞ &img) const

- template<typename t >
  bool is_sameXY (const ClmgList＜t＞ &list) const

- template<typename t >
  bool is_sameNXY (const unsigned int n, const Clmg＜t＞ &img) const

- template<typename t >
  bool is_sameNXY (const ClmgList＜t＞ &list) const

- template<typename t >
  bool is_sameXZ (const Clmg＜t＞ &img) const

- template<typename t >
  bool is_sameXZ (const ClmgList＜t＞ &list) const

- template<typename t >
  bool is_sameNXZ (const unsigned int n, const Clmg＜t＞ &img) const

- template<typename t >
  bool is_sameNXZ (const ClmgList＜t＞ &list) const

- template<typename t >
  bool is_sameXC (const Clmg＜t＞ &img) const

- template<typename t >
  bool is_sameXC (const ClmgList＜t＞ &list) const

- template<typename t >
  bool is_sameNXC (const unsigned int n, const Clmg＜t＞ &img) const

- template<typename t >
  bool is_sameNXC (const ClmgList＜t＞ &list) const

- template<typename t >
  bool is_sameYZ (const Clmg＜t＞ &img) const

- template<typename t >
  bool is_sameYZ (const ClmgList＜t＞ &list) const

- template<typename t >
  bool is_sameNYZ (const unsigned int n, const Clmg＜t＞ &img) const

- template<typename t >
  bool is_sameNYZ (const ClmgList＜t＞ &list) const

- template<typename t >
  bool is_sameYC (const Clmg＜t＞ &img) const

- template<typename t >
  bool is_sameYC (const ClmgList＜t＞ &list) const

- template<typename t >
  bool is_sameNYC (const unsigned int n, const Clmg＜t＞ &img) const

- template<typename t >
  bool is_sameNYC (const ClmgList＜t＞ &list) const

Generated by Doxygen
• template<typename t >
  bool is_sameNYC (const ClmList< t > &list) const
• template<typename t >
  bool is_sameXYZ (const Clm< t > &img) const
• template<typename t >
  bool is_sameXYZ (const ClmList< t > &list) const
• template<typename t >
  bool is_sameNXYZ (const unsigned int n, const Clm< t > &img) const
• template<typename t >
  bool is_sameNXYZ (const ClmList< t > &list) const
• template<typename t >
  bool is_sameXYC (const Clm< t > &img) const
• template<typename t >
  bool is_sameXYC (const ClmList< t > &list) const
• template<typename t >
  bool is_sameNXYC (const unsigned int n, const Clm< t > &img) const
• template<typename t >
  bool is_sameNXYC (const ClmList< 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
8.4 CImgList< T > Struct Template Reference

- 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
  
  Test if dimensions of each image of the list match specified arguments.

- bool is_sameNXYZC(const unsigned int n, const unsigned int dx, const unsigned int dy, const unsigned int dz, const unsigned int dc) const
  
  Test if list dimensions match specified arguments.

- bool containsNXYZC(const int n, const int x=0, const int y=0, const int z=0, const int c=0) const
  
  Test if list contains one particular pixel location.

- bool containsN(const int n) const
  
  Test if list contains image with specified index.

  template<typename t >
  bool contains(const T &pixel, t &n, t &x, t &y, t &z, t &c) const
  
  Test if one image of the list contains the specified referenced value.

  template<typename t >
  bool contains(const T &pixel, t &n, t &x, t &y, t &z) const
  
  Test if one of the image list contains the specified referenced value.

  template<typename t >
  bool contains(const T &pixel, t &n, t &x) const
  
  Test if one of the image list contains the specified referenced value.

  template<typename t >
  bool contains(const T &pixel, t &n) const
  
  Test if one of the image list contains the specified referenced value.

  template<typename t >
  bool contains(const Clmg< T > &img, t &n) const
  
  Test if the list contains the image 'img'.

- bool contains(const Clmg< T > &img) const
  
  Test if the list contains the image img.

Mathematical Functions

- T & min ()
  
  Return a reference to the minimum pixel value of the instance list.

- const T & min () const
  
  Return a reference to the minimum pixel value of the instance list [const version].

- T & max ()
  
  Return a reference to the maximum pixel value of the instance list.
• const T & max () const
  
  Return a reference to the maximum pixel value of the instance list [const version].

• template<typename t >
  T & min_max (t &max_val)

  Return a reference to the minimum pixel value of the instance list and return the maximum value as well.

• template<typename t >
  const T & min_max (t &max_val) const

  Return a reference to the minimum pixel value of the instance list and return the maximum value as well [const version].

• template<typename t >
  T & max_min (t &min_val)

  Return a reference to the 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> & get_insert (const CImg<T> &img, const unsigned int pos=~0U) const

  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 bool is_shared=false) const

  Insert n empty images into the current image list, at position pos [new-instance version].

• template<typename t >
  CImgList<T> & insert (const unsigned int n, const unsigned int pos=~0U)

  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 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> 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 sublist which is the appending of all images of the current CImgList instance.*

• CImgList<T> & split (const char axis, const int nb=-1)
  
  *Return a list where each image has been split along the specified axis.*

• CImgList<T> get_split (const char axis, const int nb=-1) const
  
  *Return a list where each image has been split along the specified axis [new-instance version].*

• 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 sublist which is the appending of all images of the current CImgList instance.*

• CImgList<T> & split (const char axis, const int nb=-1)
  
  *Return a list where each image has been split along the specified axis.*

• CImgList<T> get_split (const char axis, const int nb=-1) const
  
  *Return a list where each image has been split along the specified axis [new-instance version].*
Data Input

- `CImg< intT > get_select (CImgDisplay &disp, const bool feature_type=true, const char axis='x', const float align=0, const bool exit_on_anykey=false) const
  
  Display a simple interactive interface to select images or sublists.

- `CImg< intT > get_select (const char∗ const title, const bool feature_type=true, const char axis='x', const float align=0, const bool exit_on_anykey=false) const
  
  Display a simple interactive interface to select images or sublists.

- `CImgList< T > & load (const char∗ const filename)
  
  Load a list from a file.

- `CImgList< T > & load_cimg (const char∗ const filename)
  
  Load a list from a .cimg file.

- `CImgList< T > & load_cimg (std::FILE∗ const file)
  
  Load a list from a .cimg file.

- `CImgList< T > & load_cimg (const char∗ const filename, const unsigned int n0, const unsigned int n1, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0, const unsigned int x1, const unsigned int y1, const unsigned int z1, const unsigned int c1)
  
  Load a sublist list from a (non compressed) .cimg file.

- `CImgList< T > & load_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 [overloading].

- `CImgList< T > & load_parrec (const char∗ const filename)
  
  Load a list from a PAR/REC (Philips) file.

- `CImgList< T > & load_yuv (const char∗ const filename, const unsigned int size_x, const unsigned int size_y, const unsigned int chroma_subsampling=444, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool yuv2rgb=true)
  
  Load a list from a YUV image sequence file.

- `CImgList< T > & load_yuv (std::FILE∗ const file, const unsigned int size_x, const unsigned int size_y, const unsigned int chroma_subsampling=444, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool yuv2rgb=true)
  
  Load a list from an image sequence YUV file [overloading].

- `CImgList< T > & load_video (const char∗ const filename, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1)
  
  Load an image from a video file, using OpenCV library.

- `CImgList< T > & load_ffmpeg_external (const char∗ const filename)
  
  Load an image from a video file using the external tool 'ffmpeg'.

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

- `CImgList< T > & load_gzip_external (const char∗ const filename)
  
  Load a gzipped list, using external tool 'gunzip'.

- `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, float∗ const voxel_size=0, CImg< charT >∗ const description=0)
  
  Load images from a 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].

Generated by Doxygen
• 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 chroma_subsampling=444, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool yuv2rgb=true)
  Load a list from a YUV image sequence file [new-instance version].

• static CImgList< T > get_load_yuv (std::FILE *const file, const unsigned int size_x, const unsigned int size_y=1, const unsigned int chroma_subsampling=444, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool yuv2rgb=true)
  Load a list from an image sequence YUV file [new-instance version].

• static CImgList< T > get_load_video (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1)
  Load an image from a video file, using OpenCV library [new-instance version].

• static 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_gif_external (const char *const filename)
  Load gif file, using ImageMagick or GraphicsMagick's external tools [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].

• 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, float *const voxel_size=0, CImg< charT > *const description=0)
  Load a multi-page TIFF file [new-instance version].

Data Output

• const CImgList< T > & print (const char *const title=0, const bool display_stats=true) const
  Print information about the list on the standard output.

• const CImgList< T > & display (CImgDisplay &disp, const char axis='x', const float align=0) const
  Display the current CImgList instance in an existing CImgDisplay window (by reference).

• const CImgList< T > & display (CImgDisplay &disp, const boolo display_info, const char axis='x', const float align=0, unsigned int *const XYZ=0, const bool exit_on_anykey=false) const
  Display the current CImgList instance in a new display window.

• const CImgList< T > & display (const char *const title=0, const bool display_info=true, const char axis='x', const float align=0, unsigned int *const XYZ=0, const boolo exit_on_anykey=false) const
  Display the current CImgList instance in a new display window.

• const CImgList< T > & save (const char *const filename, const int number=-1, const unsigned int digits=6) const
  Save list into a file.

• const CImgList< T > & save_gif_external (const char *const filename, const float fps=25, const unsigned int nb_loops=0)
  Save image sequence as a GIF animated file.

• const CImgList< T > & save_yuv (const char *const filename=0, const unsigned int chroma_subsampling=444, const boolo is_rgb=true) const
Save list as a YUV image sequence file.

- `const CImgList<T>& save_yuv` (std::FILE *file, const unsigned int chroma_subsampling=444, const bool is_rgb=true) const
  
  Save image sequence into a YUV file.

- `const CImgList<T>& save_cimg` (const char *const filename, const bool is_compressed=false) const
  
  Save list into a .cimg file.

- `const CImgList<T>& save_cimg` (std::FILE *file, const bool is_compressed=false) const
  
  Save 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 an existing .cimg file, at specified coordinates.

- `const CImgList<T>& save_tiff` (const char *const filename, const unsigned int compression_type=0, const float *const voxel_size=0, const char *const description=0, const bool use_bigtiff=true) const
  
  Save list as a TIFF file.

- `const CImgList<T>& save_gzip_external` (const char *const filename) const
  
  Save list as a gzipped file, using external tool 'gzip'.

- `const CImgList<T>& save_video` (const char *const filename, const unsigned int fps=25, const char *const codec=0, const bool keep_open=false) const
  
  Save image sequence, using the OpenCV library.

- `const CImgList<T>& save_ffmpeg_external` (const char *const filename, const unsigned int fps=25, const char *const codec=0, const unsigned int bitrate=2048) const
  
  Save image sequence, using the external tool 'ffmpeg'.

- `CImg<ucharT> get_serialize` (const bool is_compressed=false) const
  
  Serialize a CImgList<T> instance into a raw CImg<uchar> buffer.

- `static bool is_saveable` (const char *const filename)
  
  Tell if an image list can be saved as one single file.

- `static void save_empty_cimg` (const char *const filename, const unsigned int nb, const unsigned int dx, const unsigned int dy=1, const unsigned int dz=1, const unsigned int dc=1)
  
  Save empty (non-compressed) .cimg file with specified dimensions.

- `static void save_empty_cimg` (std::FILE *const file, const unsigned int nb, const unsigned int dx, const unsigned int dy=1, const unsigned int dz=1, const unsigned int dc=1)
  
  Save empty .cimg file with specified dimensions.

- `template<typename t>`
  
  static CImgList<T> get_unserialize (const CImg<t>&buffer)
  
  Unserialize a CImg<unsigned char> serialized buffer into a CImgList<T> list.

Others

- `CImgList<T>& FFT` (const char axis, const bool invert=false)
  
  Compute a 1D Fast Fourier Transform, along specified axis.

- `CImgList<Tfloat>& get_FFT` (const char axis, const bool invert=false) const
  
  Compute a 1-D Fast Fourier Transform, along specified axis [new-instance version].

- `CImgList<T>& FFT` (const bool invert=false)
  
  Compute n-D Fast Fourier Transform.

- `CImgList<Tfloat>& get_FFT` (const bool invert=false) const
  
  Compute n-D Fast Fourier Transform [new-instance version].

- `CImgList<T>& reverse_object3d` ()
  
  Reverse primitives orientations of a 3D object.
8.4 CImgList<T> > Struct Template Reference

- **CImgList<T> > get_reverse_object3d () const**
  Reverse primitives orientations of a 3D object [new-instance version].
- **static const CImgList<ucharT > & font (const unsigned int requested_height, const bool is_variable_width=true)**
  Return a CImg pre-defined font with requested height.

### 8.4.1 Detailed Description

```cpp
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 iterator

defined as a CImg<T>*.

```cpp
typedef CImg<T>* iterator
```

Simple iterator type, to loop through each image of a list.

**Note**
- The **CImgList<T>::iterator** type is defined as a CImg<T>*.
- You may use it like this:
  ```cpp
  CImgList<> list; // Assuming this image list is not empty
  for (CImgList<>::iterator it = list.begin(); it < list.end(); ++it) (*it).mirror('x');
  ```
- Using the loop macro **cimglist_for** is another (more concise) alternative:
  ```cpp
cimglist_for(list, l) list[l].mirror('x');
  ```

#### 8.4.2.2 const_iterator

defined to be a const CImg<T>*.

```cpp
typedef const CImg<T>* const_iterator
```

Simple const iterator type, to loop through each image of a **const** list instance.

**Note**
- The **CImgList<T>::const_iterator** type is defined to be a const CImg<T>*.
- Similar to **CImgList<T>::iterator**, but for constant list instances.
8.4.2.3 value_type

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()

~CImgList ( )

Destructor.
Destroy current list instance.

Note
- Any allocated buffer is deallocated.
- Destroying an empty list does nothing actually.

8.4.3.2 CImgList() [1/19]

CImgList ( )

Default constructor.
Construct a new empty list instance.

Note
- An empty list has no pixel data and its dimension width() is set to 0, as well as its image buffer pointer data().
- An empty list may be reassigned afterwards, with the family of the assign() methods. In all cases, the type of pixels stays T.

8.4.3.3 CImgList() [2/19]

CImgList ( const unsigned int n ) [explicit]

Construct list containing empty images.
Parameters

| $n$ | Number of empty images. |

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 needs for reallocation (that may occur when starting from an empty list and inserting several images in it).

8.4.3.4 CImgList() [3/19]

```cpp
CImgList(
  const unsigned int n,
  const unsigned int width,
  const unsigned int height = 1,
  const unsigned int depth = 1,
  const unsigned int spectrum = 1)
```

Construct list containing images of specified size.

Parameters

| $n$ | Number of images. |
| width | Width of images. |
| height | Height of images. |
| depth | Depth of images. |
| spectrum | Number of channels of images. |

Note

Pixel values are not initialized and may probably contain garbage.

8.4.3.5 CImgList() [4/19]

```cpp
CImgList(
  const unsigned int n,
  const unsigned int width,
  const unsigned int height,
  const unsigned int depth,
  const unsigned int spectrum,
  const T & val)
```

Construct list containing images of specified size, and initialize pixel values.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Number of images.</td>
</tr>
<tr>
<td>width</td>
<td>Width of images.</td>
</tr>
<tr>
<td>height</td>
<td>Height of images.</td>
</tr>
<tr>
<td>depth</td>
<td>Depth of images.</td>
</tr>
<tr>
<td>spectrum</td>
<td>Number of channels of images.</td>
</tr>
<tr>
<td>val</td>
<td>Initialization value for images pixels.</td>
</tr>
</tbody>
</table>

8.4.3.6  CImgList() [5/19]

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,
    ...)

Construct list containing images of specified size, and initialize pixel values from a sequence of integers.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Number of images.</td>
</tr>
<tr>
<td>width</td>
<td>Width of images.</td>
</tr>
<tr>
<td>height</td>
<td>Height of images.</td>
</tr>
<tr>
<td>depth</td>
<td>Depth of images.</td>
</tr>
<tr>
<td>spectrum</td>
<td>Number of channels of images.</td>
</tr>
<tr>
<td>val0</td>
<td>First value of the initializing integers sequence.</td>
</tr>
<tr>
<td>val1</td>
<td>Second value of the initializing integers sequence.</td>
</tr>
</tbody>
</table>

Warning

You must specify at least \texttt{width*height*depth*spectrum} values in your argument list, or you will probably segfault.

8.4.3.7  CImgList() [6/19]

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,
... )

Construct list containing images of specified size, and initialize pixel values from a sequence of doubles.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Number of images</td>
</tr>
<tr>
<td>width</td>
<td>Width of images</td>
</tr>
<tr>
<td>height</td>
<td>Height of images</td>
</tr>
<tr>
<td>depth</td>
<td>Depth of images</td>
</tr>
<tr>
<td>spectrum</td>
<td>Number of channels of images</td>
</tr>
<tr>
<td>val0</td>
<td>First value of the initializing doubles sequence</td>
</tr>
<tr>
<td>val1</td>
<td>Second value of the initializing doubles sequence</td>
</tr>
</tbody>
</table>

Warning

You must specify at least \(width \times height \times depth \times spectrum\) values in your argument list, or you will probably segfault.

8.4.3.8 CImgList() [7/19]

CImgList (  
    const unsigned int \(n\),
    const CImg< t > & img,
    const bool is_shared = false )

Construct list containing copies of an input image.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Number of images</td>
</tr>
<tr>
<td>img</td>
<td>Input image to copy in the constructed list.</td>
</tr>
<tr>
<td>is_shared</td>
<td>Tells if the elements of the list are shared or non-shared copies of img.</td>
</tr>
</tbody>
</table>

8.4.3.9 CImgList() [8/19]

CImgList (  
    const CImg< t > & img,
    const bool is_shared = false ) [explicit]

Construct list from one image.
Parameters

<table>
<thead>
<tr>
<th>img</th>
<th>Input image to copy in the constructed list.</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_shared</td>
<td>Tells if the element of the list is a shared or non-shared copy of img.</td>
</tr>
</tbody>
</table>

### 8.4.3.10 CImgList() [9/19]

```
CImgList (const CImg< t1 > & img1,
          const CImg< t2 > & img2,
          const bool is_shared = false )
```

Construct list from two images.

Parameters

<table>
<thead>
<tr>
<th>img1</th>
<th>First input image to copy in the constructed list.</th>
</tr>
</thead>
<tbody>
<tr>
<td>img2</td>
<td>Second input image to copy in the constructed list.</td>
</tr>
<tr>
<td>is_shared</td>
<td>Tells if the elements of the list are shared or non-shared copies of input images.</td>
</tr>
</tbody>
</table>

### 8.4.3.11 CImgList() [10/19]

```
CImgList (const CImg< t1 > & img1,
          const CImg< t2 > & img2,
          const CImg< t3 > & img3,
          const bool is_shared = false )
```

Construct list from three images.

Parameters

<table>
<thead>
<tr>
<th>img1</th>
<th>First input image to copy in the constructed list.</th>
</tr>
</thead>
<tbody>
<tr>
<td>img2</td>
<td>Second input image to copy in the constructed list.</td>
</tr>
<tr>
<td>img3</td>
<td>Third input image to copy in the constructed list.</td>
</tr>
<tr>
<td>is_shared</td>
<td>Tells if the elements of the list are shared or non-shared copies of input images.</td>
</tr>
</tbody>
</table>

### 8.4.3.12 CImgList() [11/19]

```
CImgList (const CImg< t1 > & img1,
```


\begin{verbatim}
const CImg< t2 > & img2,
const CImg< t3 > & img3,
const CImg< t4 > & img4,
const bool is_shared = false }

Construct list from four images.

Parameters

\begin{itemize}
\item \textit{img1} First input image to copy in the constructed list.
\item \textit{img2} Second input image to copy in the constructed list.
\item \textit{img3} Third input image to copy in the constructed list.
\item \textit{img4} Fourth input image to copy in the constructed list.
\item \textit{is_shared} Tells if the elements of the list are shared or non-shared copies of input images.
\end{itemize}

\end{verbatim}

\section*{8.4.3.13 \texttt{CImgList()} \[12/19\]}

\begin{verbatim}
CImgList (  
    const CImg< t1 > & img1,
    const CImg< t2 > & img2,
    const CImg< t3 > & img3,
    const CImg< t4 > & img4,
    const CImg< t5 > & img5,
    const bool is_shared = false }

Construct list from five images.

Parameters

\begin{itemize}
\item \textit{img1} First input image to copy in the constructed list.
\item \textit{img2} Second input image to copy in the constructed list.
\item \textit{img3} Third input image to copy in the constructed list.
\item \textit{img4} Fourth input image to copy in the constructed list.
\item \textit{img5} Fifth input image to copy in the constructed list.
\item \textit{is_shared} Tells if the elements of the list are shared or non-shared copies of input images.
\end{itemize}

\end{verbatim}

\section*{8.4.3.14 \texttt{CImgList()} \[13/19\]}

\begin{verbatim}
CImgList (  
    const CImg< t1 > & img1,
    const CImg< t2 > & img2,
    const CImg< t3 > & img3,
    const CImg< t4 > & img4,
    const CImg< t5 > & img5,
    const CImg< t6 > & img6,
    const bool is_shared = false }

Construct list from six images.

\end{verbatim}
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>img1</td>
<td>First input image to copy in the constructed list.</td>
</tr>
<tr>
<td>img2</td>
<td>Second input image to copy in the constructed list.</td>
</tr>
<tr>
<td>img3</td>
<td>Third input image to copy in the constructed list.</td>
</tr>
<tr>
<td>img4</td>
<td>Fourth input image to copy in the constructed list.</td>
</tr>
<tr>
<td>img5</td>
<td>Fifth input image to copy in the constructed list.</td>
</tr>
<tr>
<td>img6</td>
<td>Sixth input image to copy in the constructed list.</td>
</tr>
<tr>
<td>is_shared</td>
<td>Tells if the elements of the list are shared or non-shared copies of input images.</td>
</tr>
</tbody>
</table>

#### 8.4.3.15 CImgList()

```cpp
CImgList (const CImg< t1 > & img1,
          const CImg< t2 > & img2,
          const CImg< t3 > & img3,
          const CImg< t4 > & img4,
          const CImg< t5 > & img5,
          const CImg< t6 > & img6,
          const CImg< t7 > & img7,
          const bool is_shared = false )
```

Construct list from seven images.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>img1</td>
<td>First input image to copy in the constructed list.</td>
</tr>
<tr>
<td>img2</td>
<td>Second input image to copy in the constructed list.</td>
</tr>
<tr>
<td>img3</td>
<td>Third input image to copy in the constructed list.</td>
</tr>
<tr>
<td>img4</td>
<td>Fourth input image to copy in the constructed list.</td>
</tr>
<tr>
<td>img5</td>
<td>Fifth input image to copy in the constructed list.</td>
</tr>
<tr>
<td>img6</td>
<td>Sixth input image to copy in the constructed list.</td>
</tr>
<tr>
<td>img7</td>
<td>Seventh input image to copy in the constructed list.</td>
</tr>
<tr>
<td>is_shared</td>
<td>Tells if the elements of the list are shared or non-shared copies of input images.</td>
</tr>
</tbody>
</table>

#### 8.4.3.16 CImgList()

```cpp
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,
```

Generated by Doxygen
const CImg< t8 > & img8,
const bool is_shared = false }

Construct list from eight images.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>img1</td>
<td>First input image to copy in the constructed list.</td>
</tr>
<tr>
<td>img2</td>
<td>Second input image to copy in the constructed list.</td>
</tr>
<tr>
<td>img3</td>
<td>Third input image to copy in the constructed list.</td>
</tr>
<tr>
<td>img4</td>
<td>Fourth input image to copy in the constructed list.</td>
</tr>
<tr>
<td>img5</td>
<td>Fifth input image to copy in the constructed list.</td>
</tr>
<tr>
<td>img6</td>
<td>Sixth input image to copy in the constructed list.</td>
</tr>
<tr>
<td>img7</td>
<td>Seventh input image to copy in the constructed list.</td>
</tr>
<tr>
<td>img8</td>
<td>Eighth input image to copy in the constructed list.</td>
</tr>
<tr>
<td>is_shared</td>
<td>Tells if the elements of the list are shared or non-shared copies of input images.</td>
</tr>
</tbody>
</table>

8.4.3.17 CImgList() [16/19]

CImgList ( const CImgList< t > & list )

Construct list copy.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>Input list to copy.</td>
</tr>
</tbody>
</table>

Note

The shared state of each element of the constructed list is kept the same as in list.

8.4.3.18 CImgList() [17/19]

CImgList ( const CImgList< t > & list,
            const bool is_shared )

Construct list copy, and force the shared state of the list elements.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>Input list to copy.</td>
</tr>
<tr>
<td>is_shared</td>
<td>Tells if the elements of the list are shared or non-shared copies of input images.</td>
</tr>
</tbody>
</table>
8.4.3.19 **CImgList()**

```c++
CImgList ( const char *const filename ) [explicit]
```

Construct list by reading the content of a file.

**Parameters**

- **filename** | Filename, as a C-string.

8.4.3.20 **CImgList()**

```c++
CImgList ( const CImgDisplay & disp ) [explicit]
```

Construct list from the content of a display window.

**Parameters**

- **disp** | Display window to get content from.

**Note**

Constructed list contains a single image only.

8.4.4 **Member Function Documentation**

8.4.4.1 **get_shared()**

```c++
CImgList<T> get_shared ( )
```

Return a list with elements being shared copies of images in the list instance.

**Note**

`list2 = list1.get_shared()` is equivalent to `list2.assign(list1,true)`. 
8.4.2 assign() [1/18]

CImgList\LT\T & assign( )

Destructor [in-place version].

See also

CImgList().

8.4.3 clear()

CImgList\LT\T & clear( )

Destructor [in-place version].

Equivalent to assign().

Note

Only here for compatibility with STL naming conventions.

8.4.4 assign() [2/18]

CImgList\LT\T & assign( const unsigned int n )

Construct list containing empty images [in-place version].

See also

CImgList(unsigned int).

8.4.5 assign() [3/18]

CImgList\LT\T & assign( const unsigned int n,
const unsigned int width,
const unsigned int height = 1,
const unsigned int depth = 1,
const unsigned int spectrum = 1 )

Construct list containing images of specified size [in-place version].

See also

CImgList(unsigned int, unsigned int, unsigned int, unsigned int, unsigned int).
8.4.4.6 assign() [4/18]

`CImgList<T>& assign (const unsigned int n, const unsigned int width, const unsigned int height, const unsigned int depth, const unsigned int spectrum, const T & val );`

Construct list containing images of specified size, and initialize pixel values [in-place version].

See also

`CImgList(unsigned int, unsigned int, unsigned int, unsigned int, unsigned int, const T).`

8.4.4.7 assign() [5/18]

`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, ... );`

Construct list with images of specified size, and initialize pixel values from a sequence of integers [in-place version].

See also

`CImgList(unsigned int, unsigned int, unsigned int, unsigned int, unsigned int, const int, const int, ...).`

8.4.4.8 assign() [6/18]

`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, ... );`

Construct list with images of specified size, and initialize pixel values from a sequence of doubles [in-place version].

See also

`CImgList(unsigned int, unsigned int, unsigned int, unsigned int, unsigned int, const double, const double, ...).`
8.4.4.9 assign() [7/18]

```cpp
CImgList<T>& assign {
    const unsigned int n,
    const CImg< t > & img,
    const bool is_shared = false }
```

Construct list containing copies of an input image [in-place version].

See also

CImgList(unsigned int, const CImg<t>&, bool).

8.4.4.10 assign() [8/18]

```cpp
CImgList<T>& assign {
    const CImg< t > & img,
    const bool is_shared = false }
```

Construct list from one image [in-place version].

See also

CImgList(const CImg<t>&, bool).

8.4.4.11 assign() [9/18]

```cpp
CImgList<T>& assign {
    const CImg< t1 > & img1,
    const CImg< t2 > & img2,
    const bool is_shared = false }
```

Construct list from two images [in-place version].

See also

CImgList(const CImg<t>&, const CImg<t>&, bool).
8.4.4.12 assign() [10/18]

```cpp
CImgList<T>& assign {
    const CImg<T> & img1,
    const CImg<T> & img2,
    const CImg<T> & img3,
    const bool is_shared = false
}
```

Construct list from three images [in-place version].

See also

CImgList(const CImg<T>&, const CImg<T>&, const CImg<T>&, bool).

8.4.4.13 assign() [11/18]

```cpp
CImgList<T>& assign {
    const CImg<T> & img1,
    const CImg<T> & img2,
    const CImg<T> & img3,
    const CImg<T> & img4,
    const bool is_shared = false
}
```

Construct list from four images [in-place version].

See also

CImgList(const CImg<T>&, const CImg<T>&, const CImg<T>&, const CImg<T>&, bool).

8.4.4.14 assign() [12/18]

```cpp
CImgList<T>& assign {
    const CImg<T> & img1,
    const CImg<T> & img2,
    const CImg<T> & img3,
    const CImg<T> & img4,
    const CImg<T> & img5,
    const bool is_shared = false
}
```

Construct list from five images [in-place version].

See also

CImgList(const CImg<T>&, const CImg<T>&, const CImg<T>&, const CImg<T>&, const CImg<T>&, bool).
8.4.4.15 assign() [13/18]

\begin{verbatim}
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 }
\end{verbatim}

Construct list from six images [in-place version].

See also

\begin{verbatim}
CImgList(const CImg<t> &, const CImg<t> &, const CImg<t> &, const CImg<t> &, const CImg<t> &, const CImg<t> &, bool).
\end{verbatim}

8.4.4.16 assign() [14/18]

\begin{verbatim}
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 }
\end{verbatim}

Construct list from seven images [in-place version].

See also

\begin{verbatim}
CImgList(const CImg<t> &, const CImg<t> &, const CImg<t> &, const CImg<t> &, const CImg<t> &, const CImg<t> &, const CImg<t> &, bool).
\end{verbatim}

8.4.4.17 assign() [15/18]

\begin{verbatim}
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 }
\end{verbatim}

Construct list from eight images [in-place version].

See also

\begin{verbatim}
CImgList(const CImg<t> &, const CImg<t> &, const CImg<t> &, const CImg<t> &, const CImg<t> &, const CImg<t> &, const CImg<t> &, bool).
\end{verbatim}
8.4.4.18 assign() [16/18]

```cpp
CImgList<T>& assign {
    const CImgList<T> & list,
    const bool is_shared = false
}
```

Construct list as a copy of an existing list and force the shared state of the list elements [in-place version].

See also

`CImgList(const CImgList<T>&, bool is_shared)`.

8.4.4.19 assign() [17/18]

```cpp
CImgList<T>& assign {
    const char *const filename
}
```

Construct list by reading the content of a file [in-place version].

See also

`CImgList(const char *const)`.

8.4.4.20 assign() [18/18]

```cpp
CImgList<T>& assign {
    const CImgDisplay & disp
}
```

Construct list from the content of a display window [in-place version].

See also

`CImgList(const CImgDisplay&)`.

8.4.4.21 move_to() [1/2]

```cpp
CImgList<T>& move_to {
    CImgList<T> & list
}
```

Transfer the content of the list instance to another list.
Parameters

| list | Destination list. |

Note

When returning, the current list instance is empty and the initial content of list is destroyed.

8.4.4.22 move_to() [2/2]

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. |
| pos | Index of the insertion in the list. |

Note

When returning, the list instance is empty and the initial content of list is preserved (only images indexes may be modified).

8.4.4.23 swap()

CImgList< T >& swap ( CImgList< T >& list )

Swap all fields between two list instances.

Parameters

| list | List to swap fields with. |

Note

Can be used to exchange the content of two lists in a fast way.
8.4.4.24 empty()

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.

```cpp
void f(const CImgList<char>& list=CImgList<char>::empty());
```

8.4.4.25 operator()() [1/3]

CImg<T>& operator() {
    const unsigned int pos
}

Return a reference to one image element of the list.

Parameters

| pos  | Index of the image element. |

8.4.4.26 operator()() [2/3]

const CImg<T>& operator() {
    const unsigned int pos
    const

Return a reference to one image of the list.

Parameters

| pos  | Index of the image element. |

8.4.4.27 operator()() [3/3]

T& operator() {
    const unsigned int pos,
    const unsigned int x,
    const unsigned int y = 0,
    const unsigned int z = 0,
    const unsigned int c = 0
}

Return a reference to one pixel value of one image of the list.
### Parameters

<table>
<thead>
<tr>
<th>pos</th>
<th>Index of the image element.</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>X-coordinate of the pixel value.</td>
</tr>
<tr>
<td>y</td>
<td>Y-coordinate of the pixel value.</td>
</tr>
<tr>
<td>z</td>
<td>Z-coordinate of the pixel value.</td>
</tr>
<tr>
<td>c</td>
<td>C-coordinate of the pixel value.</td>
</tr>
</tbody>
</table>

### Note

`list(n,x,y,z,c)` is equivalent to `list[n](x,y,z,c)`.

---

#### 8.4.4.28 operator CImg\(<\ T\ >\ *()\)

```cpp
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 operator\(=()\) [1/4]

```cpp
CImgList< T >& operator= (const CImg< T >& img)
```

Construct list from one image **[in-place version]**.

**Parameters**

| img | Input image to copy in the constructed list. |

**Note**

`list = img;` is equivalent to `list.assign(img);`.

---

#### 8.4.4.30 operator\(=()\) [2/4]

```cpp
CImgList< T >& operator= (const CImgList< T >& list)
```

Construct list from another list.
Parameters

| list | Input list to copy. |

Note

\[
\text{list1} = \text{list2} \text{ is equivalent to } \text{list1.assign(list2)};.
\]

8.4.4.31 operator=() [3/4]

\[
\text{CImgList}\langle T\rangle\& \text{ operator}= (\text{const char }\ast\text{const filename })
\]

Construct list by reading the content of a file [in-place version].

See also

CImgList(const char *const).

8.4.4.32 operator=() [4/4]

\[
\text{CImgList}\langle T\rangle\& \text{ operator}= (\text{const CImgDisplay }\&\text{ disp })
\]

Construct list from the content of a display window [in-place version].

See also

CImgList(const CImgDisplay&).

8.4.4.33 operator+()

\[
\text{CImgList}\langle T\rangle \text{ operator+ ( ) const}
\]

Return a non-shared copy of a list.

Note

\[
+\text{list is equivalent to CImgList}\langle T\rangle(\text{list},\text{false}). \text{ It forces the copy to have non-shared elements.}
\]

8.4.4.34 operator,() [1/2]

\[
\text{CImgList}\langle T\rangle\& \text{ operator, ( \text{const CImg}\langle t \rangle \&\text{ img })}
\]

Return a copy of the list instance, where image img has been inserted at the end.
Parameters

\[ \text{Parameters} \]

| 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:

\[
\text{img1, img2, img3, img4).display("My four images");}
\]

8.4.4.35 \texttt{operator,() [2/2]}

\texttt{CImgList<T>& operator, (}

\texttt{  const CImgList< t > & list) }

Return a copy of the list instance, where all elements of input list \texttt{list} have been inserted at the end.

Parameters

| list | List inserted at the end of the instance copy. |

8.4.4.36 \texttt{operator>()}

\texttt{CImg<T> operator> (}

\texttt{  const char axis ) const}

Return image corresponding to the appending of all images of the instance list along specified axis.

Parameters

| axis | Appending axis. Can be { 'x' | 'y' | 'z' | 'c' }. |

Note

\texttt{list>'x' is equivalent to list.get_append('x').}

8.4.4.37 \texttt{operator<()}

\texttt{CImgList<T> operator< (}

\texttt{  const char axis ) const}

Return list corresponding to the splitting of all images of the instance list along specified axis.

---

Generated by Doxygen
Parameters

| axis | Axis used for image splitting. |

Note

list< 'x' > is equivalent to list.get_split('x').

8.4.4.38 pixel_type()

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.

8.4.4.39 width()

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.

8.4.4.40 size()

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.
8.4.4.41 \texttt{data()} [1/2]

\begin{verbatim}
CImg<T>* data()
\end{verbatim}

Return pointer to the first image of the list.

\textbf{Note}

Images in a list are stored as a buffer of \texttt{CImg<T>}. 

8.4.4.42 \texttt{data()} [2/2]

\begin{verbatim}
CImg<T>* data(
    const unsigned int pos)
\end{verbatim}

Return pointer to the pos-th image of the list.

\textbf{Parameters}

\begin{itemize}
    \item \texttt{pos} \quad Index of the image element to access.
\end{itemize}

\textbf{Note}

\texttt{list.data(n)}; is equivalent to \texttt{list.data + n};.

8.4.4.43 \texttt{at()}

\begin{verbatim}
CImg<T>& at(
    const int pos)
\end{verbatim}

Return pos-th image of the list.

\textbf{Parameters}

\begin{itemize}
    \item \texttt{pos} \quad Index of the image element to access.
\end{itemize}

8.4.4.44 \texttt{atXYZC()} [1/2]

\begin{verbatim}
T& atXYZC(
    const int pos,
    const int x,
\end{verbatim}

Generated by Doxygen
\begin{verbatim}
const int y,
const int x,
const int c,
const T & out_value
\)
\end{verbatim}

Access to pixel value with Dirichlet boundary conditions.

Parameters

<table>
<thead>
<tr>
<th>pos</th>
<th>Index of the image element to access.</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>X-coordinate of the pixel value.</td>
</tr>
<tr>
<td>y</td>
<td>Y-coordinate of the pixel value.</td>
</tr>
<tr>
<td>z</td>
<td>Z-coordinate of the pixel value.</td>
</tr>
<tr>
<td>c</td>
<td>C-coordinate of the pixel value.</td>
</tr>
<tr>
<td>out_value</td>
<td>Default value returned if offset is outside image bounds.</td>
</tr>
</tbody>
</table>

Note

\texttt{list.atNXYZC(p,x,y,z,c);} \textbf{is equivalent to} \texttt{list[p].atXYZC(x,y,z,c);}.

8.4.4.45 atNXYZC() [2/2]

\begin{verbatim}
T& atNXYZC (  
    const int pos,
    const int x,
    const int y,
    const int z,
    const int c )
\end{verbatim}

Access to pixel value with Neumann boundary conditions.

Parameters

<table>
<thead>
<tr>
<th>pos</th>
<th>Index of the image element to access.</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>X-coordinate of the pixel value.</td>
</tr>
<tr>
<td>y</td>
<td>Y-coordinate of the pixel value.</td>
</tr>
<tr>
<td>z</td>
<td>Z-coordinate of the pixel value.</td>
</tr>
<tr>
<td>c</td>
<td>C-coordinate of the pixel value.</td>
</tr>
</tbody>
</table>

Note

\texttt{list.atNXYZC(p,x,y,z,c);} \textbf{is equivalent to} \texttt{list[p].atXYZC(x,y,z,c);}.

8.4.4.46 atXYZ() [1/2]

\begin{verbatim}
T& atXYZ (  
    const int pos,
\end{verbatim}
const int x,
const int y,
const int z,
const int c,
const T & out_value }

Access pixel value with Dirichlet boundary conditions for the 3 coordinates (pos, x, y, z).

Parameters

<table>
<thead>
<tr>
<th>pos</th>
<th>Index of the image element to access.</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>X-coordinate of the pixel value.</td>
</tr>
<tr>
<td>y</td>
<td>Y-coordinate of the pixel value.</td>
</tr>
<tr>
<td>z</td>
<td>Z-coordinate of the pixel value.</td>
</tr>
<tr>
<td>c</td>
<td>C-coordinate of the pixel value.</td>
</tr>
<tr>
<td>out_value</td>
<td>Default value returned if offset is outside image bounds.</td>
</tr>
</tbody>
</table>

Note

list.atNXYZ(p, x, y, z, c); is equivalent to list[p].atXYZ(x, y, z, c);

8.4.4.47 atNXYZ() [2/2]

T& atNXYZ (  
    const int pos,
    const int x,
    const int y,
    const int z,
    const int c = 0 )

Access to pixel value with Neumann boundary conditions for the 4 coordinates (pos, x, y, z).

Parameters

<table>
<thead>
<tr>
<th>pos</th>
<th>Index of the image element to access.</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>X-coordinate of the pixel value.</td>
</tr>
<tr>
<td>y</td>
<td>Y-coordinate of the pixel value.</td>
</tr>
<tr>
<td>z</td>
<td>Z-coordinate of the pixel value.</td>
</tr>
<tr>
<td>c</td>
<td>C-coordinate of the pixel value.</td>
</tr>
</tbody>
</table>

Note

list.atNXYZ(p, x, y, z, c); is equivalent to list[p].atXYZ(x, y, z, c);

Generated by Doxygen
8.4.4.48 atNXY() [1/2]

T& atNXY (  
    const int pos,  
    const int x,  
    const int y,  
    const int z,  
    const int c,  
    const T & out_value  
)

Access to pixel value with Dirichlet boundary conditions for the 3 coordinates (pos, x,y).

Parameters

<table>
<thead>
<tr>
<th>pos</th>
<th>Index of the image element to access.</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>X-coordinate of the pixel value.</td>
</tr>
<tr>
<td>y</td>
<td>Y-coordinate of the pixel value.</td>
</tr>
<tr>
<td>z</td>
<td>Z-coordinate of the pixel value.</td>
</tr>
<tr>
<td>c</td>
<td>C-coordinate of the pixel value.</td>
</tr>
<tr>
<td>out_value</td>
<td>Default value returned if offset is outside image bounds.</td>
</tr>
</tbody>
</table>

Note

\[ \text{list.atNXYZ(p,x,y,z,c)}; \text{ is equivalent to list[p].atXYZ(x,y,z,c);} . \]

8.4.4.49 atNXY() [2/2]

T& atNXY (  
    const int pos,  
    const int x,  
    const int y,  
    const int z = 0,  
    const int c = 0  
)

Access to pixel value with Neumann boundary conditions for the 3 coordinates (pos, x,y).

Parameters

<table>
<thead>
<tr>
<th>pos</th>
<th>Index of the image element to access.</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>X-coordinate of the pixel value.</td>
</tr>
<tr>
<td>y</td>
<td>Y-coordinate of the pixel value.</td>
</tr>
<tr>
<td>z</td>
<td>Z-coordinate of the pixel value.</td>
</tr>
<tr>
<td>c</td>
<td>C-coordinate of the pixel value.</td>
</tr>
</tbody>
</table>

Note

\[ \text{list.atNXYZ(p,x,y,z,c)}; \text{ is equivalent to list[p].atXYZ(x,y,z,c);} . \]
8.4.4.50 atNX() [1/2]

```cpp
T& atNX (  
    const int pos,  
    const int x,  
    const int y,  
    const int z,  
    const int c,  
    const T & out_value )
```

Access to pixel value with Dirichlet boundary conditions for the 2 coordinates \((pos, x)\).

**Parameters**

<table>
<thead>
<tr>
<th>pos</th>
<th>Index of the image element to access.</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>X-coordinate of the pixel value.</td>
</tr>
<tr>
<td>y</td>
<td>Y-coordinate of the pixel value.</td>
</tr>
<tr>
<td>z</td>
<td>Z-coordinate of the pixel value.</td>
</tr>
<tr>
<td>c</td>
<td>C-coordinate of the pixel value.</td>
</tr>
<tr>
<td>out_value</td>
<td>Default value returned if offset is outside image bounds.</td>
</tr>
</tbody>
</table>

**Note**

\[ \text{list.atNXYZ(p,x,y,z,c);} \text{ is equivalent to list[p].atXYZ(x,y,z,c);} \]

8.4.4.51 atNX() [2/2]

```cpp
T& atNX (  
    const int pos,  
    const int x,  
    const int y = 0,  
    const int z = 0,  
    const int c = 0 )
```

Access to pixel value with Neumann boundary conditions for the 2 coordinates \((pos, x)\).

**Parameters**

<table>
<thead>
<tr>
<th>pos</th>
<th>Index of the image element to access.</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>X-coordinate of the pixel value.</td>
</tr>
<tr>
<td>y</td>
<td>Y-coordinate of the pixel value.</td>
</tr>
<tr>
<td>z</td>
<td>Z-coordinate of the pixel value.</td>
</tr>
<tr>
<td>c</td>
<td>C-coordinate of the pixel value.</td>
</tr>
</tbody>
</table>

**Note**

\[ \text{list.atNXYZ(p,x,y,z,c);} \text{ is equivalent to list[p].atXYZ(x,y,z,c);} \]

Generated by Doxygen
8.4.4.52 atN() [1/2]

```cpp
T& atN {
    const int pos,
    const int x,
    const int y,
    const int z,
    const int c,
    const T & out_value }
```

Access to pixel value with Dirichlet boundary conditions for the coordinate (pos).

### Parameters

<table>
<thead>
<tr>
<th>pos</th>
<th>Index of the image element to access.</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>X-coordinate of the pixel value.</td>
</tr>
<tr>
<td>y</td>
<td>Y-coordinate of the pixel value.</td>
</tr>
<tr>
<td>z</td>
<td>Z-coordinate of the pixel value.</td>
</tr>
<tr>
<td>c</td>
<td>C-coordinate of the pixel value.</td>
</tr>
<tr>
<td>out_value</td>
<td>Default value returned if offset is outside image bounds.</td>
</tr>
</tbody>
</table>

**Note**

`list.atNXYZ(p,x,y,z,c);` is equivalent to `list[p].atXYZ(x,y,z,c);`.

8.4.4.53 atN() [2/2]

```cpp
T& atN {
    const int pos,
    const int x = 0,
    const int y = 0,
    const int z = 0,
    const int c = 0 }
```

Return pixel value with Neumann boundary conditions for the coordinate (pos).

### Parameters

<table>
<thead>
<tr>
<th>pos</th>
<th>Index of the image element to access.</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>X-coordinate of the pixel value.</td>
</tr>
<tr>
<td>y</td>
<td>Y-coordinate of the pixel value.</td>
</tr>
<tr>
<td>z</td>
<td>Z-coordinate of the pixel value.</td>
</tr>
<tr>
<td>c</td>
<td>C-coordinate of the pixel value.</td>
</tr>
</tbody>
</table>

**Note**

`list.atNXYZ(p,x,y,z,c);` is equivalent to `list[p].atXYZ(x,y,z,c);`.
8.4.4.54  is_sameN() [1/2]

bool is_sameN (  
    const unsigned int size_n ) const

Test if number of image elements is equal to specified value.

Parameters

| size← _n | Number of image elements to test. |

8.4.4.55  is_sameN() [2/2]

bool is_sameN (  
    const CImgList< t > & list ) const

Test if number of image elements is equal between two images lists.

Parameters

| list | Input list to compare with. |

8.4.4.56  is_sameXYZC()

bool is_sameXYZC (  
    const unsigned int dx,  
    const unsigned int dy,  
    const unsigned int dz,  
    const unsigned int dc ) const

Test if dimensions of each image of the list match specified arguments.

Parameters

| dx   | Checked image width. |
| dy   | Checked image height. |
| dz   | Checked image depth.  |
| dc   | Checked image spectrum. |
const unsigned int n,
const unsigned int dx,
const unsigned int dy,
const unsigned int dz,
const unsigned int dc ) const

Test if list dimensions match specified arguments.

Parameters

<table>
<thead>
<tr>
<th>n</th>
<th>Number of images in the list.</th>
</tr>
</thead>
<tbody>
<tr>
<td>dx</td>
<td>Checked image width.</td>
</tr>
<tr>
<td>dy</td>
<td>Checked image height.</td>
</tr>
<tr>
<td>dz</td>
<td>Checked image depth.</td>
</tr>
<tr>
<td>dc</td>
<td>Checked image spectrum.</td>
</tr>
</tbody>
</table>

8.4.4.58 containsNXYZC()

bool containsNXYZC (  
    const int n,  
    const int x = 0,  
    const int y = 0,  
    const int z = 0,  
    const int c = 0 ) const

Test if list contains one particular pixel location.

Parameters

<table>
<thead>
<tr>
<th>n</th>
<th>Index of the image whom checked pixel value belong to.</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>X-coordinate of the checked pixel value.</td>
</tr>
<tr>
<td>y</td>
<td>Y-coordinate of the checked pixel value.</td>
</tr>
<tr>
<td>z</td>
<td>Z-coordinate of the checked pixel value.</td>
</tr>
<tr>
<td>c</td>
<td>C-coordinate of the checked pixel value.</td>
</tr>
</tbody>
</table>

8.4.4.59 containsN()

bool containsN (  
    const int n ) const

Test if list contains image with specified index.

Parameters

<table>
<thead>
<tr>
<th>n</th>
<th>Index of the checked image.</th>
</tr>
</thead>
</table>
8.4.4.60 contains() [1/8]

```cpp
bool contains (  
    const T & pixel,  
    t & n,  
    t & x,  
    t & y,  
    t & z,  
    t & c ) const
```

Test if one image of the list contains the specified referenced value.

**Parameters**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pixel</strong></td>
<td>Reference to pixel value to test.</td>
<td></td>
</tr>
<tr>
<td><strong>out n</strong></td>
<td>Index of image containing the pixel value, if test succeeds.</td>
<td></td>
</tr>
<tr>
<td><strong>out x</strong></td>
<td>X-coordinate of the pixel value, if test succeeds.</td>
<td></td>
</tr>
<tr>
<td><strong>out y</strong></td>
<td>Y-coordinate of the pixel value, if test succeeds.</td>
<td></td>
</tr>
<tr>
<td><strong>out z</strong></td>
<td>Z-coordinate of the pixel value, if test succeeds.</td>
<td></td>
</tr>
<tr>
<td><strong>out c</strong></td>
<td>C-coordinate of the pixel value, if test succeeds.</td>
<td></td>
</tr>
</tbody>
</table>

**Note**

If true, set coordinates (n,x,y,z,c).

8.4.4.61 contains() [2/8]

```cpp
bool contains (  
    const T & pixel,  
    t & n,  
    t & x,  
    t & y,  
    t & z ) const
```

Test if one of the image list contains the specified referenced value.

**Parameters**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pixel</strong></td>
<td>Reference to pixel value to test.</td>
<td></td>
</tr>
<tr>
<td><strong>out n</strong></td>
<td>Index of image containing the pixel value, if test succeeds.</td>
<td></td>
</tr>
<tr>
<td><strong>out x</strong></td>
<td>X-coordinate of the pixel value, if test succeeds.</td>
<td></td>
</tr>
<tr>
<td><strong>out y</strong></td>
<td>Y-coordinate of the pixel value, if test succeeds.</td>
<td></td>
</tr>
<tr>
<td><strong>out z</strong></td>
<td>Z-coordinate of the pixel value, if test succeeds.</td>
<td></td>
</tr>
</tbody>
</table>

Generated by Doxygen
Note

If true, set coordinates \((n,x,y,z)\).

### 8.4.4.62 contains() [3/8]

```cpp
bool contains (  
    const T & pixel,  
    t & n,  
    t & x,  
    t & y ) const
```

Test if one of the image list contains the specified referenced value.

**Parameters**

<table>
<thead>
<tr>
<th></th>
<th>pixel</th>
<th>Reference to pixel value to test.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>n</td>
<td>Index of image containing the pixel value, if test succeeds.</td>
</tr>
<tr>
<td>out</td>
<td>x</td>
<td>X-coordinate of the pixel value, if test succeeds.</td>
</tr>
<tr>
<td>out</td>
<td>y</td>
<td>Y-coordinate of the pixel value, if test succeeds.</td>
</tr>
</tbody>
</table>

Note

If true, set coordinates \((n,x,y)\).

### 8.4.4.63 contains() [4/8]

```cpp
bool contains (  
    const T & pixel,  
    t & n,  
    t & x ) const
```

Test if one of the image list contains the specified referenced value.

**Parameters**

<table>
<thead>
<tr>
<th></th>
<th>pixel</th>
<th>Reference to pixel value to test.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>n</td>
<td>Index of image containing the pixel value, if test succeeds.</td>
</tr>
<tr>
<td>out</td>
<td>x</td>
<td>X-coordinate of the pixel value, if test succeeds.</td>
</tr>
</tbody>
</table>

Note

If true, set coordinates \((n,x)\).
8.4.4.64 contains() [5/8]

```cpp
bool contains ( const T & pixel,
               t & n ) const
```

Test if one of the image list contains the specified referenced value.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>n</th>
<th>Reference to pixel value to test.</th>
</tr>
</thead>
</table>

Note

If true, set coordinates (n).

8.4.4.65 contains() [6/8]

```cpp
bool contains ( const T & pixel ) const
```

Test if one of the image list contains the specified referenced value.

Parameters

<table>
<thead>
<tr>
<th>pixel</th>
<th>Reference to pixel value to test.</th>
</tr>
</thead>
</table>

8.4.4.66 contains() [7/8]

```cpp
bool contains ( const CImg< T > & img,
               t & n ) const
```

Test if the list contains the image 'img'.

Parameters

<table>
<thead>
<tr>
<th>img</th>
<th>Reference to image to test.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>n</td>
</tr>
</tbody>
</table>

Note

If true, returns the position (n) of the image in the list.
8.4.4.67 contains() [8/8]

```cpp
bool contains (const CImg<T> & img) const
```

Test if the list contains the image img.

Parameters

- `img` Reference to image to test.

8.4.4.68 min_max() [1/2]

```cpp
T& min_max (t & max_val)
```

Return a reference to the minimum pixel value of the instance list and return the maximum value as well.

Parameters

- `out` `max_val` Value of the maximum value found.

8.4.4.69 min_max() [2/2]

```cpp
const T& min_max (t & max_val) const
```

Return a reference to the minimum pixel value of the instance list and return the maximum value as well [const version].

Parameters

- `out` `max_val` Value of the maximum value found.

8.4.4.70 max_min()

```cpp
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. |

8.4.4.71  

```cpp
CImgList<T>& insert {
    const CImg& 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 | Tells if the inserted image is a shared copy of `img` or not. |

8.4.4.72  

```cpp
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. |

8.4.4.73  

```cpp
CImgList<T>& insert {
    const unsigned int n,
    const CImg& 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 copies to insert. |
| pos | Index of the insertion. |
# Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Number of image copies to insert.</td>
</tr>
<tr>
<td>img</td>
<td>Image to insert by copy.</td>
</tr>
<tr>
<td>pos</td>
<td>Index of the insertion.</td>
</tr>
<tr>
<td>is_shared</td>
<td>Tells if inserted images are shared copies of img or not.</td>
</tr>
</tbody>
</table>

## 8.4.4.74 insert() [4/5]

```cpp
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>Image list to insert.</td>
</tr>
<tr>
<td>pos</td>
<td>Index of the insertion.</td>
</tr>
<tr>
<td>is_shared</td>
<td>Tells if inserted images are shared copies of images of list or not.</td>
</tr>
</tbody>
</table>

## 8.4.4.75 insert() [5/5]

```cpp
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Number of list copies to insert.</td>
</tr>
<tr>
<td>list</td>
<td>Image list to insert.</td>
</tr>
<tr>
<td>pos</td>
<td>Index of the insertion.</td>
</tr>
<tr>
<td>is_shared</td>
<td>Tells if inserted images are shared copies of images of list or not.</td>
</tr>
</tbody>
</table>

## 8.4.4.76 remove() [1/2]

```cpp
CImgList<T>& remove {
```
8.4 CImgList<T> Struct Template Reference

Remove all images between from indexes.

Parameters

<table>
<thead>
<tr>
<th>pos1</th>
<th>Starting index of the removal.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos2</td>
<td>Ending index of the removal.</td>
</tr>
</tbody>
</table>

8.4.4.77 remove() [2/2]

CImgList<T>& remove {
    const unsigned int pos
}

Remove image at index pos from the image list.

Parameters

| pos  | Index of the image to remove. |

8.4.4.78 images()

CImgList<T>& images {
    const unsigned int pos0,
    const unsigned int pos1
}

Return a sublist.

Parameters

<table>
<thead>
<tr>
<th>pos0</th>
<th>Starting index of the sublist.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos1</td>
<td>Ending index of the sublist.</td>
</tr>
</tbody>
</table>

8.4.4.79 get_shared_images()

CImgList<T> get_shared_images {
    const unsigned int pos0,
    const unsigned int pos1
}

Return a shared sublist.
Parameters

<table>
<thead>
<tr>
<th>pos0</th>
<th>Starting index of the sublist.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos1</td>
<td>Ending index of the sublist.</td>
</tr>
</tbody>
</table>

8.4.4.80  get_append()

\[
\text{CImg<T> get_append (} \\
\quad \text{const char axis,} \\
\quad \text{const float align = } 0 \text{ } \text{) const}
\]

Return a single image which is the appending of all images of the current \text{CImgList} instance.

Parameters

<table>
<thead>
<tr>
<th>axis</th>
<th>Appending axis. Can be {'x', 'y', 'z', 'c'}.</th>
</tr>
</thead>
<tbody>
<tr>
<td>align</td>
<td>Appending alignment.</td>
</tr>
</tbody>
</table>

8.4.4.81  split()

\[
\text{CImgList<T> & split (} \\
\quad \text{const char axis,} \\
\quad \text{const int nb = } -1 \text{ })
\]

Return a list where each image has been split along the specified axis.

Parameters

<table>
<thead>
<tr>
<th>axis</th>
<th>Axis to split images along.</th>
</tr>
</thead>
<tbody>
<tr>
<td>nb</td>
<td>Number of split parts for each image.</td>
</tr>
</tbody>
</table>

8.4.4.82  push_back() [1/2]

\[
\text{CImgList<T> & push_back (} \\
\quad \text{const CImg<T> & img})
\]

Insert image at the end of the list.

Parameters

| img | Image to insert. |
8.4.4.83  push_front() [1/2]

```cpp
CImgList<T>& push_front (  
    const CImg<T> & img )
```

Insert image at the front of the list.

Parameters

| img | Image to insert. |

8.4.4.84  push_back() [2/2]

```cpp
CImgList<T>& push_back (  
    const CImgList<T> & list )
```

Insert list at the end of the current list.

Parameters

| list | List to insert. |

8.4.4.85  push_front() [2/2]

```cpp
CImgList<T>& push_front (  
    const CImgList<T> & list )
```

Insert list at the front of the current list.

Parameters

| list | List to insert. |

8.4.4.86  erase()

```cpp
CImgList<T>& erase (  
    const iterator iter )
```

Remove image pointed by iterator.
Parameters

| iter | Iterator pointing to the image to remove. |

8.4.4.87 get_select() [1/2]

```cpp
CImg<dT> get_select (  
    CImgDisplay & disp,  
    const bool feature_type = true,  
    const char axis = 'x',  
    const float align = 0,  
    const bool exit_on_anykey = false ) const
```

Display a simple interactive interface to select images or sublists.

Parameters

| 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. |
| exit_on_anykey | Exit function when any key is pressed. |

Returns

A one-column vector containing the selected image indexes.

8.4.4.88 get_select() [2/2]

```cpp
CImg<dT> get_select (  
    const char *const title,  
    const bool feature_type = true,  
    const char axis = 'x',  
    const float align = 0,  
    const bool exit_on_anykey = false ) const
```

Display a simple interactive interface to select images or sublists.

Parameters

| 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. |
| exit_on_anykey | Exit function when any key is pressed. |
Returns

A one-column vector containing the selected image indexes.

8.4.4.89 load()

\texttt{CImgList<T>\& load (const char \textasciitilde const filename )}

Load a list from a file.

Parameters

\begin{tabular}{|l|l|}
\hline
\textit{filename} & Filename to read data from. \\
\hline
\end{tabular}

8.4.4.90 load_cimg() [1/3]

\texttt{CImgList<T>\& load_cimg (const char \textasciitilde const filename )}

Load a list from a .cimg file.

Parameters

\begin{tabular}{|l|l|}
\hline
\textit{filename} & Filename to read data from. \\
\hline
\end{tabular}

8.4.4.91 load_cimg() [2/3]

\texttt{CImgList<T>\& load_cimg (std::FILE \textasciitilde const file )}

Load a list from a .cimg file.

Parameters

\begin{tabular}{|l|l|}
\hline
\textit{file} & File to read data from. \\
\hline
\end{tabular}

8.4.4.92 load_cimg() [3/3]

\texttt{CImgList<T>\& load_cimg (}
Load a sublist list from a (non compressed) .cimg file.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename to read data from.</td>
</tr>
<tr>
<td>n0</td>
<td>Starting index of images to read (~0U for max).</td>
</tr>
<tr>
<td>n1</td>
<td>Ending index of images to read (~0U for max).</td>
</tr>
<tr>
<td>x0</td>
<td>Starting X-coordinates of image regions to read.</td>
</tr>
<tr>
<td>y0</td>
<td>Starting Y-coordinates of image regions to read.</td>
</tr>
<tr>
<td>z0</td>
<td>Starting Z-coordinates of image regions to read.</td>
</tr>
<tr>
<td>c0</td>
<td>Starting C-coordinates of image regions to read.</td>
</tr>
<tr>
<td>x1</td>
<td>Ending X-coordinates of image regions to read (~0U for max).</td>
</tr>
<tr>
<td>y1</td>
<td>Ending Y-coordinates of image regions to read (~0U for max).</td>
</tr>
<tr>
<td>z1</td>
<td>Ending Z-coordinates of image regions to read (~0U for max).</td>
</tr>
<tr>
<td>c1</td>
<td>Ending C-coordinates of image regions to read (~0U for max).</td>
</tr>
</tbody>
</table>

8.4.4.93 load_parrec()

CImgList<T>& load_parrec (  
    const char *const filename  )

Load a list from a PAR/REC (Philips) file.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename to read data from.</td>
</tr>
</tbody>
</table>

8.4.4.94 load_yuv()

CImgList<T>& load_yuv (  
    const char *const filename,  
    const unsigned int size_x,  
    const unsigned int size_y,  
    const unsigned int chroma_subsampling = 444,
Load a list from a YUV image sequence file.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>filename</code></td>
<td>Filename to read data from.</td>
</tr>
<tr>
<td><code>size_x</code></td>
<td>Width of the images.</td>
</tr>
<tr>
<td><code>size_y</code></td>
<td>Height of the images.</td>
</tr>
<tr>
<td><code>chroma_subsampling</code></td>
<td>Type of chroma subsampling. Can be `{ 420</td>
</tr>
<tr>
<td><code>first_frame</code></td>
<td>Index of first image frame to read.</td>
</tr>
<tr>
<td><code>last_frame</code></td>
<td>Index of last image frame to read.</td>
</tr>
<tr>
<td><code>step_frame</code></td>
<td>Step applied between each frame.</td>
</tr>
<tr>
<td><code>yuv2rgb</code></td>
<td>Apply YUV to RGB transformation during reading.</td>
</tr>
</tbody>
</table>

8.4.4.95  `load_video()`

```
CImgList<T>& load_video (   
    const char *const filename, 
    const unsigned int first_frame = 0, 
    const unsigned int last_frame = ~0U, 
    const unsigned int step_frame = 1 )
```

Load an image from a video file, using OpenCV library.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>filename</code></td>
<td>Filename, as a C-string.</td>
</tr>
<tr>
<td><code>first_frame</code></td>
<td>Index of the first frame to read.</td>
</tr>
<tr>
<td><code>last_frame</code></td>
<td>Index of the last frame to read (can be higher than the actual number of frames, e.g. <code>~0U</code>).</td>
</tr>
<tr>
<td><code>step_frame</code></td>
<td>Step value for frame reading.</td>
</tr>
</tbody>
</table>

Note

If `step_frame==0`, the current video stream is forced to be released (without any frames read).

8.4.4.96  `load_ffmpeg_external()`

```
CImgList<T>& load_ffmpeg_external (   
    const char *const filename )
```

Load an image from a video file using the external tool 'ffmpeg'.

Generated by Doxygen
**Parameters**

| filename | Filename to read data from. |

### 8.4.4.97 load_gif_external()

```cpp
CImgList<T>& load_gif_external (const char *const filename)
```

Load gif file, using ImageMagick or GraphicsMagick's external tools.

**Parameters**

| filename | Filename to read data from. |

### 8.4.4.98 load_gzip_external()

```cpp
CImgList<T>& load_gzip_external (const char *const filename)
```

Load a gzipped list, using external tool 'gunzip'.

**Parameters**

| filename | Filename to read data from. |

### 8.4.4.99 load_tiff()

```cpp
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,
float *const voxel_size = 0,
CImg< charT > *const description = 0)
```

Load images from a TIFF file.

**Parameters**

| filename | Filename to read data from. |
| first_frame | Index of first image frame to read. |
Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>last_frame</td>
<td>Index of last image frame to read.</td>
</tr>
<tr>
<td>step_frame</td>
<td>Step applied between each frame.</td>
</tr>
<tr>
<td>out voxel_size</td>
<td>Voxel size, as stored in the filename.</td>
</tr>
<tr>
<td>out description</td>
<td>Description, as stored in the filename.</td>
</tr>
</tbody>
</table>

8.4.4.100 print()

```cpp
const CImgList<T>& print (
    const char *const title = 0,
    const bool display_stats = true ) const
```

Print information about the list on the standard output.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>Label set to the information displayed.</td>
</tr>
<tr>
<td>display_stats</td>
<td>Tells if image statistics must be computed and displayed.</td>
</tr>
</tbody>
</table>

8.4.4.101 display() [1/3]

```cpp
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

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>disp</td>
<td>Reference to an existing CImgDisplay instance, where the current image list will be displayed.</td>
</tr>
<tr>
<td>axis</td>
<td>Appending axis. Can be ( 'x'</td>
</tr>
<tr>
<td>align</td>
<td>Appending alignment.</td>
</tr>
</tbody>
</table>

Note

This function displays the list images of the current CImgList instance into an existing CImgDisplay window. Images of the list are appended in a single temporary image for visualization purposes. The function returns immediately.
8.4.4.102  display() [2/3]

const CImgList<T>& display {
    CImgDisplay & disp,
    const bool display_info,
    const char axis = 'x',
    const float align = 0,
    unsigned int *const XYZ = 0,
    const bool exit_on_anykey = false } const

Display the current CImgList instance in a new display window.

Parameters

<table>
<thead>
<tr>
<th>disp</th>
<th>Display window.</th>
</tr>
</thead>
<tbody>
<tr>
<td>display_info</td>
<td>Tells if image information are displayed on the standard output.</td>
</tr>
<tr>
<td>axis</td>
<td>Alignment axis for images viewing.</td>
</tr>
<tr>
<td>align</td>
<td>Appending alignment.</td>
</tr>
<tr>
<td>in,out XYZ</td>
<td>Contains the XYZ coordinates at start / exit of the function.</td>
</tr>
<tr>
<td>exit_on_anykey</td>
<td>Exit function when any key is pressed.</td>
</tr>
</tbody>
</table>

Note

This function opens a new window with a specific title and displays the list images of the current CImgList instance into it. Images of the list are appended in a single temporary image for visualization purposes. The function returns when a key is pressed or the display window is closed by the user.

8.4.4.103  display() [3/3]

const CImgList<T>& display {
    const char *const title = 0,
    const bool display_info = true,
    const char axis = 'x',
    const float align = 0,
    unsigned int *const XYZ = 0,
    const bool exit_on_anykey = false } const

Display the current CImgList instance in a new display window.

Parameters

<table>
<thead>
<tr>
<th>title</th>
<th>Title of the opening display window.</th>
</tr>
</thead>
<tbody>
<tr>
<td>display_info</td>
<td>Tells if list information must be written on standard output.</td>
</tr>
<tr>
<td>axis</td>
<td>Appending axis. Can be { 'x'</td>
</tr>
<tr>
<td>align</td>
<td>Appending alignment.</td>
</tr>
<tr>
<td>in,out XYZ</td>
<td>Contains the XYZ coordinates at start / exit of the function.</td>
</tr>
<tr>
<td>exit_on_anykey</td>
<td>Exit function when any key is pressed.</td>
</tr>
</tbody>
</table>
8.4.4.104 save()

\begin{verbatim}
const CImgList\lt T\gt \& save (  
    const char *const filename,  
    const int number = -1,  
    const unsigned int digits = 6 ) const
\end{verbatim}

Save list into a file.

Parameters

<table>
<thead>
<tr>
<th>filename</th>
<th>Filename to write data to.</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>When positive, represents an index added to the filename. Otherwise, no number is added.</td>
</tr>
<tr>
<td>digits</td>
<td>Number of digits used for adding the number to the filename.</td>
</tr>
</tbody>
</table>

8.4.4.105 is_saveable()

\begin{verbatim}
static bool is_saveable (  
    const char *const filename ) [static]
\end{verbatim}

Tell if an image list can be saved as one single file.

Parameters

| filename | Filename, as a C-string. |

Returns

true if the file format supports multiple images, false otherwise.

8.4.4.106 save_gif_external()

\begin{verbatim}
const CImgList\lt T\gt \& save_gif_external (  
    const char *const filename,  
    const float fps = 25,  
    const unsigned int nb_loops = 0 )
\end{verbatim}

Save image sequence as a GIF animated file.

Parameters

| filename | Filename to write data to. |
| fps      | Number of desired frames per second. |
| nb_loops | Number of loops (0 for infinite looping). |
8.4.4.107  save_yuv() [1/2]

const CImgList<T>& save_yuv (  
    const char *const filename = 0,  
    const unsigned int chroma_subsampling = 444,  
    const bool is_rgb = true ) const

Save list as a YUV image sequence file.

Parameters

<table>
<thead>
<tr>
<th>filename</th>
<th>Filename to write data to.</th>
</tr>
</thead>
<tbody>
<tr>
<td>chroma_subsampling</td>
<td>Type of chroma subsampling. Can be { 420</td>
</tr>
<tr>
<td>is_rgb</td>
<td>Tells if the RGB to YUV conversion must be done for saving.</td>
</tr>
</tbody>
</table>

8.4.4.108  save_yuv() [2/2]

const CImgList<T>& save_yuv (  
    std::FILE *const file,  
    const unsigned int chroma_subsampling = 444,  
    const bool is_rgb = true ) const

Save image sequence into a YUV file.

Parameters

<table>
<thead>
<tr>
<th>file</th>
<th>File to write data to.</th>
</tr>
</thead>
<tbody>
<tr>
<td>chroma_subsampling</td>
<td>Type of chroma subsampling. Can be { 420</td>
</tr>
<tr>
<td>is_rgb</td>
<td>Tells if the RGB to YUV conversion must be done for saving.</td>
</tr>
</tbody>
</table>

8.4.4.109  save_cimg() [1/4]

const CImgList<T>& save_cimg (  
    const char *const filename,  
    const bool is_compressed = false ) const

Save list into a .cimg file.

Parameters

<table>
<thead>
<tr>
<th>filename</th>
<th>Filename to write data to.</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_compressed</td>
<td>Tells if data compression must be enabled.</td>
</tr>
</tbody>
</table>
8.4.4.110  save_cimg() [2/4]

const CImgList<T>& save_cimg (  
    std::FILE * file,  
    const bool is_compressed = false ) const

Save list into a .cimg file.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>File to write data to.</td>
</tr>
<tr>
<td>is_compressed</td>
<td>Tells if data compression must be enabled.</td>
</tr>
</tbody>
</table>

8.4.4.111  save_cimg() [3/4]

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename to write data to.</td>
</tr>
<tr>
<td>n0</td>
<td>Starting index of images to write.</td>
</tr>
<tr>
<td>x0</td>
<td>Starting X-coordinates of image regions to write.</td>
</tr>
<tr>
<td>y0</td>
<td>Starting Y-coordinates of image regions to write.</td>
</tr>
<tr>
<td>z0</td>
<td>Starting Z-coordinates of image regions to write.</td>
</tr>
<tr>
<td>c0</td>
<td>Starting C-coordinates of image regions to write.</td>
</tr>
</tbody>
</table>

8.4.4.112  save_cimg() [4/4]

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 an existing .cimg file, at specified coordinates.
8.4.113 save_empty_cimg() [1/2]

```c
static void save_empty_cimg (  
    const char *const filename,  
    const unsigned int nb,  
    const unsigned int dx,  
    const unsigned int dy = 1,  
    const unsigned int dz = 1,  
    const unsigned int dc = 1 ) [static]
```

Save empty (non-compressed) .cimg file with specified dimensions.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename to write data to.</td>
</tr>
<tr>
<td>nb</td>
<td>Number of images to write.</td>
</tr>
<tr>
<td>dx</td>
<td>Width of images in the written file.</td>
</tr>
<tr>
<td>dy</td>
<td>Height of images in the written file.</td>
</tr>
<tr>
<td>dz</td>
<td>Depth of images in the written file.</td>
</tr>
<tr>
<td>dc</td>
<td>Spectrum of images in the written file.</td>
</tr>
</tbody>
</table>

8.4.114 save_empty_cimg() [2/2]

```c
static void save_empty_cimg (  
    std::FILE *const file,  
    const unsigned int nb,  
    const unsigned int dx,  
    const unsigned int dy = 1,  
    const unsigned int dz = 1,  
    const unsigned int dc = 1 ) [static]
```

Save empty .cimg file with specified dimensions.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>File to write data to.</td>
</tr>
<tr>
<td>nb</td>
<td>Number of images to write.</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dx</td>
<td>Width of images in the written file.</td>
</tr>
<tr>
<td>dy</td>
<td>Height of images in the written file.</td>
</tr>
<tr>
<td>dz</td>
<td>Depth of images in the written file.</td>
</tr>
<tr>
<td>dc</td>
<td>Spectrum of images in the written file.</td>
</tr>
</tbody>
</table>

8.4.4.115  `save_tiff()`

```cpp
const CImgList<T>& save_tiff (  
    const char *const filename,  
    const unsigned int compression_type = 0,  
    const float *const voxel_size = 0,  
    const char *const description = 0,  
    const bool use_bigtiff = true ) const
```

Save list as a TIFF file.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename to write data to.</td>
</tr>
<tr>
<td>compression_type</td>
<td>Compression mode used to write data.</td>
</tr>
<tr>
<td>voxel_size</td>
<td>Voxel size, to be stored in the filename.</td>
</tr>
<tr>
<td>description</td>
<td>Description, to be stored in the filename.</td>
</tr>
<tr>
<td>use_bigtiff</td>
<td>Allow to save big tiff files (&gt;4Gb).</td>
</tr>
</tbody>
</table>

8.4.4.116  `save_gzip_external()`

```cpp
const CImgList<T>& save_gzip_external (  
    const char *const filename ) const
```

Save list as a gzipped file, using external tool 'gzip'.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename to write data to.</td>
</tr>
</tbody>
</table>

8.4.4.117  `save_video()`

```cpp
const CImgList<T>& save_video (  
    const char *const filename,  
```
const unsigned int fps = 25,
const char * codec = 0,
const bool keep_open = false ) const

Save image sequence, using the OpenCV library.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename to write data to.</td>
</tr>
<tr>
<td>fps</td>
<td>Number of frames per second.</td>
</tr>
<tr>
<td>codec</td>
<td>Type of compression (See <a href="http://www.fourcc.org/codecs.php">http://www.fourcc.org/codecs.php</a> to see available codecs).</td>
</tr>
<tr>
<td>keep_open</td>
<td>Tells if the video writer associated to the specified filename must be kept open or not (to allow frames to be added in the same file afterwards).</td>
</tr>
</tbody>
</table>

8.4.4.118 save_ffmpeg_external()

const CImgList<T> & save_ffmpeg_external (  
    const char *const filename,
    const unsigned int fps = 25,
    const char * const codec = 0,
    const unsigned int bitrate = 2048 ) const

Save image sequence, using the external tool 'ffmpeg'.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Filename to write data to.</td>
</tr>
<tr>
<td>fps</td>
<td>Number of frames per second.</td>
</tr>
<tr>
<td>codec</td>
<td>Type of compression.</td>
</tr>
<tr>
<td>bitrate</td>
<td>Output bitrate</td>
</tr>
</tbody>
</table>

8.4.4.119 get_serialize()

CImg<ucharT> get_serialize (  
    const bool is_compressed = false ) const

Serialize a CImgList<T> instance into a raw CImg<ucharT> buffer.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_compressed</td>
<td>tells if zlib compression must be used for serialization (this requires 'cimg_use_zlib' been enabled).</td>
</tr>
</tbody>
</table>
8.4.4.120  font()

static const CImgList<ucharT>& font {
    const unsigned int requested_height,
    const bool is_variable_width = true } [static]

Return a CImg pre-defined font with requested height.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>font_height</td>
<td>Height of the desired font (exact match for 13,23,53,103).</td>
</tr>
<tr>
<td>is_variable_width</td>
<td>Decide if the font has a variable (true) or fixed (false) width.</td>
</tr>
</tbody>
</table>

8.4.4.121  FFT()[1/2]

CImgList<T>& FFT (  
    const char axis,  
    const bool invert = false )

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axis</td>
<td>Axis along which the Fourier transform is computed.</td>
</tr>
<tr>
<td>invert</td>
<td>Tells if the direct (false) or inverse transform (true) is computed.</td>
</tr>
</tbody>
</table>

8.4.4.122  FFT()[2/2]

CImgList<T>& FFT (  
    const bool invert = false )

Compute n-D Fast Fourier Transform.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invert</td>
<td>Tells if the direct (false) or inverse transform (true) is computed.</td>
</tr>
</tbody>
</table>
Index

∼CImg
  cimg_library::CImg, 124
∼CImgDisplay
  cimg_library::CImgDisplay, 349
∼CImgList
  cimg_library::CImgList, 388
abs
  cimg_library::CImg, 193
acos
  cimg_library::CImg, 196
acosh
  cimg_library::CImg, 198
append
  cimg_library::CImg, 252
append_object3d
  cimg_library::CImg, 273
asin
  cimg_library::CImg, 196
asinh
  cimg_library::CImg, 198
assign
  cimg_library::CImg, 133–135
  cimg_library::CImgList, 396–402
at
  cimg_library::CImg, 166, 167
  cimg_library::CImgList, 409
atNXYZC
  cimg_library::CImgList, 409, 410
atNXYZ
  cimg_library::CImgList, 410, 411
atNXY
  cimg_library::CImgList, 411, 412
atNX
  cimg_library::CImgList, 412, 413
atXYZC
  cimg_library::CImg, 171
atXYZ
  cimg_library::CImg, 170
atXY
  cimg_library::CImg, 170
atan
  cimg_library::CImg, 197
atan2
  cimg_library::CImg, 197
atanh
  cimg_library::CImg, 198
atN
  cimg_library::CImgList, 413, 414
atof
  cimg_library::cimg, 49
atX
  cimg_library::CImg, 167, 169
autocrop
  cimg_library::CImg, 245
back
  cimg_library::CImg, 166
begin
  cimg_library::CImg, 165
blur
  cimg_library::CImg, 259, 260
  cimg_library::CImg, 260, 261
blur_anisotropic
  cimg_library::CImg, 260, 261
blur_bilateral
  cimg_library::CImg, 261, 262
blur_box
  cimg_library::CImg, 263
blur_guided
  cimg_library::CImg, 264
blur_median
  cimg_library::CImg, 265
blur_patch
  cimg_library::CImg, 264
box3d
  cimg_library::CImg, 278
boxfilter
  cimg_library::CImg, 263
button
  cimg_library::CImgList, 359
CImg
  cimg_library::CImg, 124–132
CImg Library Overview, 11
Clg< T >, 65
Clg3Dtoobject3d
  cimg_library::CImg, 282
ClgDisplay, 343
  cimg_library::CImgDisplay, 350–352
ClgException, 372
ClgList
  cimg_library::ClgList, 388–396
ClgList< T >, 373
cimg_library, 33
cimg_library::Cimg
  ∼CImg, 124
  abs, 193
  acos, 196
  acosh, 198
INDEX 443

get_isoline3d, 275
get_isosurface3d, 275
get_matrix_at, 209
get_projections2d, 244
get_projections3d, 274
get_rows, 247
get_serialize, 343
get_shared_channel, 251
get_shared_channels, 251
get_shared_points, 249
get_shared_row, 250
get_shared_rows, 250
get_shared_slice, 251
get_shared_slices, 250
get_slice, 247
get_slices, 247
get_split, 252
generate_symmetric_eigen, 216
generate_tensor_at, 210
generate_vector_at, 209
HSV_LUT256, 234
haar, 270
histogram, 231
hot_LUT256, 234
identity_matrix, 212, 222
index, 232
invert, 214
is_clmg3d, 190
is_empty, 182
is_inf, 182
is_nan, 182
is_object3d, 190
is_overlapped, 189
is_sameXYZC, 187
is_sameXYC, 186
is_sameXYZ, 185
is_sameXZC, 186
is_sameXC, 184
is_sameXY, 183
is_sameXZ, 183
is_sameYZC, 186, 187
is_sameYC, 184, 185
is_sameY, 184
is_sameZC, 185
is_shared, 182
isoline3d, 276
isosurface3d, 277
iterator, 123
jet_LUT256, 235
kth_smallest, 205
label, 233
linear_atXYZC, 175
linear_atXYZ, 174
linear_atXY, 174
linear_atX, 171, 173
lines_LUT256, 234
load, 314
load_analyze, 319
load_asci, 315
load_bmp, 315
load_camera, 326
load_clmg, 320
load_dcrav_external, 326
load_dim, 315
load_exr, 321
load_ffmpeg_external, 324
load_gif_external, 324
load_graphicsmagick_external, 325
load_gzip_external, 325
load_imagemagick_external, 325
load_inr, 321
load_jpeg, 316
load_magick, 316
load_medcon_external, 326
load_minc2, 319
load_off, 323
load_other, 327
load_pandore, 321
load_parrec, 322
load_pfm, 318
load_png, 316
load_pnm, 316
load_raw, 322
load_rgb, 318
load_rgba, 318
load_tiff, 319
load_video, 324
load_yuv, 323
log, 192
log10, 193
log2, 192
MSE, 206
magnitude, 208
map, 232
matchpatch, 268
matrix, 220, 221
max, 202, 204
max_min, 204
min, 201, 202
min_max, 204
mirror, 239
move_to, 136
mul, 198
normalize, 228
norm, 228
object3dtoCImg3d, 282
offset, 164
operator &, 150, 151
operator &==, 150
operator T, 139
operator==, 157, 158
operator<, 161
operator<<, 154, 155
operator<<=, 154

Generated by Doxygen
| vector, 212, 219, 220 |
| warp, 243 |
| watershed, 258 |
| width, 161 |
| XYZtoRGB, 236 |
| cimg_library::CImgDisplay =~CImgDisplay, 349 |
| assign, 352 |
| button, 359 |
| CImgDisplay, 350–352 |
| close, 363 |
| display, 362 |
| empty, 352 |
| flush, 370 |
| frames_per_second, 361 |
| height, 356 |
| hide_mouse, 367 |
| is_closed, 354 |
| is_key, 354, 355 |
| is_key_sequence, 355 |
| is_keyESC, 356 |
| key, 360 |
| keycode, 361 |
| mouse_x, 358 |
| mouse_y, 358 |
| move, 363 |
| normalization, 356 |
| operator bool, 353 |
| operator=, 353 |
| paint, 371 |
| released_key, 361 |
| render, 370 |
| resize, 364, 365 |
| screenshot, 352, 371 |
| set_button, 368 |
| set_fullscreen, 366 |
| set_key, 369 |
| set_mouse, 367 |
| set_normalization, 365 |
| set_title, 366 |
| set_wheel, 368, 369 |
| show, 363 |
| show_mouse, 367 |
| snapshot, 371 |
| title, 357 |
| toggle_fullscreen, 367 |
| wait, 370 |
| wheel, 359 |
| width, 356 |
| window_height, 357 |
| window_width, 357 |
| window_x, 358 |
| window_y, 358 |
| cimg_library::CImgList =~CImgList, 388 |
| assign, 396–402 |
| at, 409 |
| atNXYZC, 409, 410 |
| atNXY, 411, 412 |
| atNX, 412, 413 |
| atN, 413, 414 |
| CImgList, 388–396 |
| close, 397 |
| const_iterator, 387 |
| contains, 417–420 |
| containsNXYZC, 416 |
| containsN, 416 |
| data, 408, 409 |
| display, 431, 432 |
| empty, 403 |
| erase, 425 |
| FFT, 440 |
| font, 439 |
| get_append, 424 |
| get_select, 426 |
| get_serialize, 439 |
| get_shared, 396 |
| get_shared_images, 423 |
| images, 423 |
| insert, 421, 422 |
| is_sameNXYZC, 415 |
| is_sameXYZC, 415 |
| is_sameN, 414, 415 |
| is_saveable, 433 |
| iterator, 387 |
| load, 427 |
| load_cimg, 427 |
| load_ffmpeg_external, 429 |
| load_gif_external, 430 |
| load_gzip_external, 430 |
| load_parrec, 428 |
| load_tiff, 430 |
| load_video, 429 |
| load_yuv, 428 |
| max_min, 420 |
| min_max, 420 |
| move_to, 402, 403 |
| operator CImg< T > *, 405 |
| operator<, 407 |
| operator>, 407 |
| operator(), 404 |
| operator+, 406 |
| operator-, 406, 407 |
| operator=, 405, 406 |
| pixel_type, 408 |
| print, 431 |
| push_back, 424, 425 |
| push_front, 425 |
| remove, 422, 423 |
| save, 432 |
| save_cimg, 434, 435 |
| save_ffmpeg_external, 439 |
| save_gif_external, 433 |
| save_gzip_external, 438 |
INDEX

...
gzip_path
  cimg_library::cimg, 58
HSV_LUT256
  cimg_library::CImg, 234
haar
  cimg_library::CImg, 270
height
  cimg_library::CImg, 162
cimg_library::CImgDisplay, 356
hide_mouse
  cimg_library::CImgDisplay, 367
histogram
  cimg_library::CImg, 231
hot_LUT256
  cimg_library::CImg, 234
How pixel data are stored with CImg., 29
How to use CImg library with Visual C++ 2005 Express Edition ?, 19
identity_matrix
  cimg_library::CImg, 212, 222
imagemagick_path
  cimg_library::cimg, 56
images
  cimg_library::CImgList, 423
index
  cimg_library::CImg, 232
info
  cimg_library::cimg, 43
insert
  cimg_library::CImgList, 421, 422
invert
  cimg_library::CImg, 214
invert_endianness
  cimg_library::cimg, 46
is_CImg3d
  cimg_library::CImg, 190
is_closed
  cimg_library::CImgDisplay, 354
is_directory
  cimg_library::cimg, 53
is_empty
  cimg_library::CImg, 182
is_file
  cimg_library::cimg, 53
is_inf
  cimg_library::CImg, 182
is_key
  cimg_library::CImgDisplay, 354, 355
is_key_sequence
  cimg_library::CImgDisplay, 355
is_keyESC
  cimg_library::CImgDisplay, 356
is_nan
  cimg_library::CImg, 182
is_object3d
  cimg_library::CImg, 190
is_overlapped
cimg_library::CImg, 171, 173
load
  cimg_library::CImg, 314
  cimg_library::CImgList, 427
load_analyze
  cimg_library::CImg, 319
load_ascii
  cimg_library::CImg, 315
load_bmp
  cimg_library::CImg, 315
load_camera
  cimg_library::CImg, 326
load_cimg
  cimg_library::CImg, 320
  cimg_library::CImgList, 427
load_dcraw_external
  cimg_library::CImg, 326
load_dim
  cimg_library::CImg, 315
load_exr
  cimg_library::CImg, 321
load_ffmpeg_external
  cimg_library::CImg, 324
  cimg_library::CImgList, 429
load_gif_external
  cimg_library::CImg, 324
  cimg_library::CImgList, 430
load_graphicsmagick_external
  cimg_library::CImg, 325
load_gzip_external
  cimg_library::CImg, 325
  cimg_library::CImgList, 430
load_imagemagick_external
  cimg_library::CImg, 325
load_inr
  cimg_library::CImg, 321
load_jpeg
  cimg_library::CImg, 316
load_magick
  cimg_library::CImg, 316
load_medcon_external
  cimg_library::CImg, 326
load_minc2
  cimg_library::CImg, 319
load_network
  cimg_library::Cimg, 62
load_off
  cimg_library::CImg, 323
load_pandore
  cimg_library::CImg, 321
load_parrec
  cimg_library::CImg, 322
  cimg_library::CImgList, 428
load_pfm
load_png
  cimg_library::CImg, 318
load_pnm
  cimg_library::CImg, 316
load_raw
  cimg_library::CImg, 316
load_rgb
  cimg_library::CImg, 318
load_rgb_a
  cimg_library::CImg, 318
load_tiff
  cimg_library::CImg, 319
  cimg_library::CImgList, 430
load_video
  cimg_library::CImg, 324
  cimg_library::CImgList, 429
load_yuv
  cimg_library::CImg, 323
  cimg_library::CImgList, 428
log
  cimg_library::CImg, 192
log10
  cimg_library::CImg, 193
log2
  cimg_library::CImg, 192
MSE
  cimg_library::CImg, 206
magnitude
  cimg_library::CImg, 208
map
  cimg_library::CImg, 232
matchpatch
  cimg_library::CImg, 268
matrix
  cimg_library::CImg, 220, 221
max
  cimg_library::CImg, 201, 202
max_min
  cimg_library::CImg, 204
  cimg_library::CImgList, 420
medcon_path
  cimg_library::cimg, 57
min
  cimg_library::CImg, 201, 202
min_max
  cimg_library::CImg, 204
  cimg_library::CImgList, 420
minmod
  cimg_library::cimg, 48
mirror
  cimg_library::CImg, 239
mod
  cimg_library::cimg, 48
mouse_x
  cimg_library::CImgDisplay, 358
mouse_y
  cimg_library::CImgDisplay, 358
move
cimg_library::CImgDisplay, 363
move_to
cimg_library::CImg, 136
  cimg_library::CImgList, 402, 403
mul
cimg_library::CImg, 198
noise
cimg_library::CImg, 227
norm
cimg_library::CImg, 228
normalization
  cimg_library::CImgDisplay, 356
normalize
cimg_library::CImg, 228
object3dtoCImg3d
cimg_library::CImg, 282
offset
cimg_library::CImg, 164
openmp_mode
cimg_library::cimg, 43
operator &
cimg_library::CImg, 150, 151
operator &=
cimg_library::CImg, 150
operator bool
  cimg_library::CImgDisplay, 353
operator CImg< T >
  cimg_library::CImgList, 405
operator T *
cimg_library::CImg, 139
operator!=
cimg_library::CImg, 157, 158
operator<
cimg_library::CImg, 161
  cimg_library::CImgList, 407
operator<<
cimg_library::CImg, 154, 155
operator<<=
cimg_library::CImg, 154
operator>
  cimg_library::CImgList, 407
operator>>
cimg_library::CImg, 155, 156
operator>>>=
cimg_library::CImg, 155
operator*
  cimg_library::CImg, 147
operator==
cimg_library::CImg, 146
operator~
cimg_library::CImg, 156
operator^=
cimg_library::CImg, 153
operator^()
cimg_library::CImg, 152, 153
operator()
operator++
cimg_library::CImg, 142, 143
operator++=
cimg_library::CImg, 141, 142
operator,
cimg_library::CImg, 158, 160
  cimg_library::CImgList, 406, 407
operator--
cimg_library::CImg, 145, 146
operator--=
cimg_library::CImg, 145
operator/=
cimg_library::CImg, 144
operator/=
cimg_library::CImg, 147, 148
operator==
cimg_library::CImg, 139–141
  cimg_library::CImgDisplay, 353
  cimg_library::CImgList, 405, 406
operator===
cimg_library::CImg, 156, 157
operator%
cimg_library::CImg, 149, 150
operator%=
cimg_library::CImg, 149
operator |
cimg_library::CImg, 152
operator |=
cimg_library::CImg, 151
output
cimg_library::cimg, 42
PSNR
cimg_library::CImg, 206
paint
cimg_library::CImgDisplay, 371
permute_axes
cimg_library::CImg, 240
pixel_type
  cimg_library::CImg, 161
  cimg_library::CImgList, 408
plane3d
cimg_library::CImg, 280
pow
cimg_library::CImg, 199, 200
print
cimg_library::CImg, 327
  cimg_library::CImgList, 431
push_back
cimg_library::CImgList, 424, 425
push_front
cimg_library::CImgList, 425

Generated by Doxygen
<table>
<thead>
<tr>
<th>Function</th>
<th>Library/Module</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>quantize</td>
<td>cimg_library::CImg</td>
<td>230</td>
</tr>
<tr>
<td>RGBtoXYZ</td>
<td>cimg_library::CImg</td>
<td>236</td>
</tr>
<tr>
<td>rand</td>
<td>cimg_library::CImg</td>
<td>226</td>
</tr>
<tr>
<td>released_key</td>
<td>cimg_library::CImgDisplay</td>
<td>361</td>
</tr>
<tr>
<td>remove</td>
<td>cimg_library::CImgList</td>
<td>422, 423</td>
</tr>
<tr>
<td>render</td>
<td>cimg_library::CImgDisplay</td>
<td>370</td>
</tr>
<tr>
<td>resize</td>
<td>cimg_library::CImg</td>
<td>236–238</td>
</tr>
<tr>
<td>resize_doubleXY</td>
<td>cimg_library::CImg</td>
<td>238</td>
</tr>
<tr>
<td>resize_object3d</td>
<td>cimg_library::CImg</td>
<td>272</td>
</tr>
<tr>
<td>resize_tripleXY</td>
<td>cimg_library::CImg</td>
<td>238</td>
</tr>
<tr>
<td>Retrieving Command Line Arguments,</td>
<td></td>
<td>31</td>
</tr>
<tr>
<td>rol</td>
<td>cimg_library::CImg</td>
<td>200</td>
</tr>
<tr>
<td>ror</td>
<td>cimg_library::CImg</td>
<td>200, 201</td>
</tr>
<tr>
<td>rotate</td>
<td>cimg_library::CImg</td>
<td>240, 241, 243</td>
</tr>
<tr>
<td>rotation_matrix</td>
<td>cimg_library::CImg</td>
<td>223</td>
</tr>
<tr>
<td>round</td>
<td>cimg_library::CImg</td>
<td>227</td>
</tr>
<tr>
<td>row</td>
<td>cimg_library::CImg</td>
<td>49</td>
</tr>
<tr>
<td>SVD</td>
<td>cimg_library::CImg</td>
<td>217</td>
</tr>
<tr>
<td>save</td>
<td>cimg_library::CImg</td>
<td>330</td>
</tr>
<tr>
<td>save_analyze</td>
<td>cimg_library::CImg</td>
<td>335</td>
</tr>
<tr>
<td>save_ascii</td>
<td>cimg_library::CImg</td>
<td>330</td>
</tr>
<tr>
<td>save_bmp</td>
<td>cimg_library::CImg</td>
<td>331</td>
</tr>
<tr>
<td>save_cimg</td>
<td>cimg_library::CImg</td>
<td>335</td>
</tr>
<tr>
<td>save_cpp</td>
<td>cimg_library::CImg</td>
<td>330</td>
</tr>
<tr>
<td>save_dlm</td>
<td>cimg_library::CImg</td>
<td>331</td>
</tr>
<tr>
<td>save_empty_cimg</td>
<td>cimg_library::CImg</td>
<td>336</td>
</tr>
<tr>
<td>save_exr</td>
<td>cimg_library::CImgList</td>
<td>437</td>
</tr>
<tr>
<td>save_ffmpeg_external</td>
<td>cimg_library::CImg</td>
<td>340</td>
</tr>
<tr>
<td>save_gif_external</td>
<td>cimg_library::CImgList</td>
<td>439</td>
</tr>
<tr>
<td>save_graphicsmagick_external</td>
<td>cimg_library::CImg</td>
<td>341</td>
</tr>
<tr>
<td>save_gzip_external</td>
<td>cimg_library::CImgList</td>
<td>438</td>
</tr>
<tr>
<td>save Słowęski_external</td>
<td>cimg_library::CImg</td>
<td>341</td>
</tr>
<tr>
<td>save_imageMagick</td>
<td>cimg_library::CImg</td>
<td>341</td>
</tr>
<tr>
<td>save инр</td>
<td>cimg_library::CImg</td>
<td>341</td>
</tr>
<tr>
<td>save_jpeg</td>
<td>cimg_library::CImg</td>
<td>331</td>
</tr>
<tr>
<td>save_magick</td>
<td>cimg_library::CImg</td>
<td>332</td>
</tr>
<tr>
<td>save_medcon_external</td>
<td>cimg_library::CImg</td>
<td>342</td>
</tr>
<tr>
<td>save_minc2</td>
<td>cimg_library::CImg</td>
<td>334</td>
</tr>
<tr>
<td>save_off</td>
<td>cimg_library::CImg</td>
<td>339</td>
</tr>
<tr>
<td>save_other</td>
<td>cimg_library::CImg</td>
<td>342</td>
</tr>
<tr>
<td>save_pandore</td>
<td>cimg_library::CImg</td>
<td>337</td>
</tr>
<tr>
<td>save_pfm</td>
<td>cimg_library::CImg</td>
<td>333</td>
</tr>
<tr>
<td>save_png</td>
<td>cimg_library::CImg</td>
<td>332</td>
</tr>
<tr>
<td>save_pmk</td>
<td>cimg_library::CImg</td>
<td>333</td>
</tr>
<tr>
<td>save_pnm</td>
<td>cimg_library::CImg</td>
<td>332</td>
</tr>
<tr>
<td>save_raw</td>
<td>cimg_library::CImg</td>
<td>338</td>
</tr>
<tr>
<td>save_rgb</td>
<td>cimg_library::CImg</td>
<td>333</td>
</tr>
<tr>
<td>save_rgba</td>
<td>cimg_library::CImg</td>
<td>333</td>
</tr>
<tr>
<td>save_tiff</td>
<td>cimg_library::CImg</td>
<td>334</td>
</tr>
<tr>
<td>save_video</td>
<td>cimg_library::CImgList</td>
<td>438</td>
</tr>
<tr>
<td>screenshot</td>
<td>cimg_library::CImgDisplay</td>
<td>352, 371</td>
</tr>
<tr>
<td>select</td>
<td>cimg_library::CImg</td>
<td>314</td>
</tr>
<tr>
<td>sequence</td>
<td>cimg_library::CImg</td>
<td>213, 222</td>
</tr>
</tbody>
</table>
set_button
cimg_library::CImgDisplay, 368

set_fullscreen
cimg_library::CImgDisplay, 366

set_key
cimg_library::CImgDisplay, 369

set_linear_atXYZ
cimg_library::CImg, 181

set_linear_atXY
cimg_library::CImg, 181

set_linear_atX
cimg_library::CImg, 180

set_matrix_at
cimg_library::CImg, 210

set_mouse
cimg_library::CImgDisplay, 367

set_normalization
cimg_library::CImgDisplay, 365

set_tensor_at
cimg_library::CImg, 212

set_title
cimg_library::CImgDisplay, 366

set_vector_at
cimg_library::CImg, 210

set_wheel
cimg_library::CImgDisplay, 368, 369

Setting Environment Variables, 18

sharpen
cimg_library::CImg, 265

shift
cimg_library::CImg, 239

shift_object3d
cimg_library::CImg, 272

show
cimg_library::CImgDisplay, 363

show_mouse
cimg_library::CImgDisplay, 367

sign
cimg_library::CImg, 193

sin
cimg_library::CImg, 194

sinc
cimg_library::CImg, 194

sinh
cimg_library::CImg, 195

size
cimg_library::CImg, 163
cimg_library::CImgList, 408

sleep
cimg_library::cimg, 47

snapshot
cimg_library::CImgDisplay, 371

solve
cimg_library::CImg, 214

solve_tridiagonal
cimg_library::CImg, 214

sort
cimg_library::CImg, 216

spectrum
cimg_library::CImg, 162

sphere3d
cimg_library::CImg, 280

split
cimg_library::CImgList, 424

split_filename
cimg_library::cimg, 60

sqr
cimg_library::CImg, 191

sqrt
cimg_library::CImg, 191

strcasemcmp
cimg_library::CImg, 50

streamline
cimg_library::CImg, 248

stellipsize
cimg_library::cimg, 50, 51

string
cimg_library::CImg, 219

strncasemcmp
cimg_library::cimg, 49

strpare
cimg_library::cimg, 51

structure_tensors
cimg_library::CImg, 266

strainescape
cimg_library::cimg, 52

strwindows_reserved
cimg_library::cimg, 52

swap
cimg_library::CImg, 137
cimg_library::CImgList, 403

symmetric_eigen
cimg_library::CImg, 215

system
cimg_library::cimg, 45

tan
cimg_library::CImg, 195

tanh
cimg_library::CImg, 196

temporary_path
cimg_library::cimg, 56

tensor
cimg_library::CImg, 222

texturize_object3d
cimg_library::CImg, 273

threshold
cimg_library::CImg, 273

tic
cimg_library::cimg, 47

time
cimg_library::cimg, 47

title
cimg_library::CImgDisplay, 357

toc
cimg_library::CImgDisplay, 47

toggle_fullscreen
torus3d

cimg_library::CImgDisplay, 367
cimg_library::CImg, 279
transpose

cimg_library::CImg, 213

Tutorial : Getting Started., 20

unroll

cimg_library::CImg, 240

Using Display Windows., 28
Using Image Loops., 22

value_string

cimg_library::CImg, 181
value_type

cimg_library::CImg, 123
cimg_library::CImgList, 387

vanvliet

cimg_library::CImg, 258

variance

cimg_library::CImg, 205
variance_mean

cimg_library::CImg, 205
variance_noise

cimg_library::CImg, 206

vector

cimg_library::CImg, 212, 219, 220

wait

cimg_library::CImgDisplay, 370
cimg_library::cimg, 47

warn

cimg_library::cimg, 44

warp

cimg_library::CImg, 243

watershed

cimg_library::CImg, 258

wget_path

cimg_library::cimg, 59

wheel

cimg_library::CImgDisplay, 359

width

cimg_library::CImg, 161
cimg_library::CImgDisplay, 356
cimg_library::CImgList, 408

window_height

cimg_library::CImgDisplay, 357

window_width

cimg_library::CImgDisplay, 357

window_x

cimg_library::CImgDisplay, 358

window_y

cimg_library::CImgDisplay, 358

XYZtoRGB

cimg_library::CImg, 236