<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>
# Namespace Documentation

## 7.1 cimg_library Namespace Reference

### 7.1.1 Detailed Description

## 7.2 cimg_library::cimg Namespace Reference

### 7.2.1 Detailed Description

### 7.2.2 Function Documentation

#### 7.2.2.1 output()

#### 7.2.2.2 info()

#### 7.2.2.3 exception_mode() [1/2]

#### 7.2.2.4 exception_mode() [2/2]

#### 7.2.2.5 openmp_mode()

#### 7.2.2.6 eval()

#### 7.2.2.7 warn()

#### 7.2.2.8 system()

#### 7.2.2.9 endianness()

#### 7.2.2.10 invert_endianness() [1/2]

#### 7.2.2.11 invert_endianness() [2/2]

#### 7.2.2.12 time()

#### 7.2.2.13 tic()

#### 7.2.2.14 toc()

#### 7.2.2.15 sleep()

#### 7.2.2.16 wait()

#### 7.2.2.17 mod()

#### 7.2.2.18 minmod()

#### 7.2.2.19 round()

#### 7.2.2.20 atof()

#### 7.2.2.21 strncasecmp()

#### 7.2.2.22 strcasecmp()

#### 7.2.2.23 strellipsize() [1/2]

#### 7.2.2.24 strellipsize() [2/2]
7.2.2.25 strpare() ......................................................... 53
7.2.2.26 strwindows_reserved() .................................. 53
7.2.2.27 strunescape() .............................................. 53
7.2.2.28 fopen() ....................................................... 54
7.2.2.29 fclose() ....................................................... 54
7.2.2.30 is_directory() .................................................. 55
7.2.2.31 is_file() ....................................................... 55
7.2.2.32 fsize() ......................................................... 55
7.2.2.33 fdate() [1/2] ................................................. 55
7.2.2.34 fdate() [2/2] ................................................. 57
7.2.2.35 date() [1/2] .................................................... 57
7.2.2.36 date() [2/2] .................................................... 58
7.2.2.37 temporary_path() ............................................ 58
7.2.2.38 imagemagick_path() ....................................... 58
7.2.2.39 graphicsmagick_path() .................................... 59
7.2.2.40 medcon_path() .............................................. 59
7.2.2.41 ffmpeg_path() ............................................... 60
7.2.2.42 gzip_path() .................................................. 60
7.2.2.43 gunzip_path() ............................................... 61
7.2.2.44 dcraw_path() ................................................. 61
7.2.2.45 wget_path() ................................................ 61
7.2.2.46 curl_path() ................................................... 62
7.2.2.47 split_filename() ........................................... 62
7.2.2.48 fread() ....................................................... 62
7.2.2.49 fwrite() ....................................................... 63
7.2.2.50 fempty() ....................................................... 63
7.2.2.51 ftype() ......................................................... 64
7.2.2.52 load_network() .............................................. 64
7.2.2.53 files() ......................................................... 65
7.2.2.54 dialog() ....................................................... 65
8 Class Documentation

8.1 CImg< T > Struct Template Reference ........................................... 67

8.1.1 Detailed Description ................................................................. 123

8.1.2 Member Typedef Documentation .................................................. 125

8.1.2.1 iterator ................................................................................. 125

8.1.2.2 const_iterator ................................................................. 126

8.1.2.3 value_type ............................................................................. 126

8.1.3 Constructor & Destructor Documentation ....................................... 126

8.1.3.1 ~CImg() ............................................................................. 126

8.1.3.2 CImg() [1/13] ...................................................................... 127

8.1.3.3 CImg() [2/13] ................................................................. 127

8.1.3.4 CImg() [3/13] ................................................................. 128

8.1.3.5 CImg() [4/13] ................................................................. 128

8.1.3.6 CImg() [5/13] ................................................................. 129

8.1.3.7 CImg() [6/13] ................................................................. 131

8.1.3.8 CImg() [7/13] ................................................................. 132

8.1.3.9 CImg() [8/13] ................................................................. 133

8.1.3.10 CImg() [9/13] ................................................................. 134

8.1.3.11 CImg() [10/13] ................................................................. 134

8.1.3.12 CImg() [11/13] ................................................................. 135

8.1.3.13 CImg() [12/13] ................................................................. 135

8.1.3.14 CImg() [13/13] ................................................................. 136

8.1.4 Member Function Documentation ............................................... 136

8.1.4.1 assign() [1/13] ................................................................. 136

8.1.4.2 assign() [2/13] ................................................................. 137

8.1.4.3 assign() [3/13] ................................................................. 137

8.1.4.4 assign() [4/13] ................................................................. 137

8.1.4.5 assign() [5/13] ................................................................. 137

8.1.4.6 assign() [6/13] ................................................................. 138

8.1.4.7 assign() [7/13] ................................................................. 138
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1.4.38</td>
<td>operator--() [1/2]</td>
</tr>
<tr>
<td>8.1.4.39</td>
<td>operator--() [2/2]</td>
</tr>
<tr>
<td>8.1.4.40</td>
<td>operator() [1/4]</td>
</tr>
<tr>
<td>8.1.4.41</td>
<td>operator() [2/4]</td>
</tr>
<tr>
<td>8.1.4.42</td>
<td>operator() [3/4]</td>
</tr>
<tr>
<td>8.1.4.43</td>
<td>operator() [4/4]</td>
</tr>
<tr>
<td>8.1.4.44</td>
<td>operator*() [1/3]</td>
</tr>
<tr>
<td>8.1.4.45</td>
<td>operator*() [2/3]</td>
</tr>
<tr>
<td>8.1.4.46</td>
<td>operator*() [3/3]</td>
</tr>
<tr>
<td>8.1.4.47</td>
<td>operator*() [1/3]</td>
</tr>
<tr>
<td>8.1.4.48</td>
<td>operator*() [2/3]</td>
</tr>
<tr>
<td>8.1.4.49</td>
<td>operator*() [3/3]</td>
</tr>
<tr>
<td>8.1.4.50</td>
<td>operator/=() [1/3]</td>
</tr>
<tr>
<td>8.1.4.51</td>
<td>operator/=() [2/3]</td>
</tr>
<tr>
<td>8.1.4.52</td>
<td>operator/=() [3/3]</td>
</tr>
<tr>
<td>8.1.4.53</td>
<td>operator/() [1/3]</td>
</tr>
<tr>
<td>8.1.4.54</td>
<td>operator/() [2/3]</td>
</tr>
<tr>
<td>8.1.4.55</td>
<td>operator/() [3/3]</td>
</tr>
<tr>
<td>8.1.4.56</td>
<td>operator%() [1/3]</td>
</tr>
<tr>
<td>8.1.4.57</td>
<td>operator%() [2/3]</td>
</tr>
<tr>
<td>8.1.4.58</td>
<td>operator%() [3/3]</td>
</tr>
<tr>
<td>8.1.4.59</td>
<td>operator%() [1/3]</td>
</tr>
<tr>
<td>8.1.4.60</td>
<td>operator%() [2/3]</td>
</tr>
<tr>
<td>8.1.4.61</td>
<td>operator%() [3/3]</td>
</tr>
<tr>
<td>8.1.4.62</td>
<td>operator &amp;=() [1/3]</td>
</tr>
<tr>
<td>8.1.4.63</td>
<td>operator &amp;=() [2/3]</td>
</tr>
<tr>
<td>8.1.4.64</td>
<td>operator &amp;=() [3/3]</td>
</tr>
<tr>
<td>8.1.4.65</td>
<td>operator &amp;() [1/3]</td>
</tr>
<tr>
<td>8.1.4.66</td>
<td>operator &amp;() [2/3]</td>
</tr>
<tr>
<td>8.1.4.67</td>
<td>operator &amp;() [3/3]</td>
</tr>
</tbody>
</table>
8.1.4.68  operator** | =()  [1/3]          ........................................ 155
8.1.4.69  operator** | =()  [2/3]          ........................................ 155
8.1.4.70  operator** | =()  [3/3]          ........................................ 155
8.1.4.71  operator** | ()  [1/3]          ........................................ 155
8.1.4.72  operator** | ()  [2/3]          ........................................ 156
8.1.4.73  operator** | ()  [3/3]          ........................................ 156
8.1.4.74  operator^ =()  [1/3]          ........................................ 156
8.1.4.75  operator^ =()  [2/3]          ........................................ 156
8.1.4.76  operator^ =()  [3/3]          ........................................ 156
8.1.4.77  operator^ ()  [1/3]          ........................................ 156
8.1.4.78  operator^ ()  [2/3]          ........................................ 156
8.1.4.79  operator^ ()  [3/3]          ........................................ 156
8.1.4.80  operator<< =()  [1/3]          ........................................ 157
8.1.4.81  operator<< =()  [2/3]          ........................................ 158
8.1.4.82  operator<< =()  [3/3]          ........................................ 158
8.1.4.83  operator<< ()  [1/3]          ........................................ 158
8.1.4.84  operator<< ()  [2/3]          ........................................ 158
8.1.4.85  operator<< ()  [3/3]          ........................................ 158
8.1.4.86  operator>> =()  [1/3]          ........................................ 159
8.1.4.87  operator>> =()  [2/3]          ........................................ 159
8.1.4.88  operator>> =()  [3/3]          ........................................ 159
8.1.4.89  operator>> ()  [1/3]          ........................................ 159
8.1.4.90  operator>> ()  [2/3]          ........................................ 159
8.1.4.91  operator>> ()  [3/3]          ........................................ 160
8.1.4.92  operator~()          ........................................ 160
8.1.4.93  operator|=()  [1/3]          ........................................ 160
8.1.4.94  operator|=()  [2/3]          ........................................ 160
8.1.4.95  operator|=()  [3/3]          ........................................ 160
8.1.4.96  operator!=()  [1/3]          ........................................ 161
8.1.4.97  operator!=()  [2/3]          ........................................ 161
8.1.4.158 is_sameXZ() [2/2] ........................................ 185
8.1.4.159 is_sameXC() [1/2] ........................................ 185
8.1.4.160 is_sameXC() [2/2] ........................................ 185
8.1.4.161 is_sameYZ() [1/2] ........................................ 185
8.1.4.162 is_sameYZ() [2/2] ........................................ 185
8.1.4.163 is_sameYC() [1/2] ........................................ 186
8.1.4.164 is_sameYC() [2/2] ........................................ 186
8.1.4.165 is_sameZC() [1/2] ........................................ 186
8.1.4.166 is_sameZC() [2/2] ........................................ 186
8.1.4.167 is_sameXYZ() [1/2] ...................................... 186
8.1.4.168 is_sameXYZ() [2/2] ...................................... 187
8.1.4.169 is_sameXYC() [1/2] ...................................... 187
8.1.4.170 is_sameXYC() [2/2] ...................................... 187
8.1.4.171 is_sameXZC() [1/2] ...................................... 187
8.1.4.172 is_sameXZC() [2/2] ...................................... 187
8.1.4.173 is_sameYZC() [1/2] ...................................... 188
8.1.4.174 is_sameYZC() [2/2] ...................................... 188
8.1.4.175 is_sameXYZC() [1/2] ..................................... 188
8.1.4.176 is_sameXYZC() [2/2] ..................................... 188
8.1.4.177 containsXYZC() ........................................... 188
8.1.4.178 contains() [1/5] .......................................... 189
8.1.4.179 contains() [2/5] .......................................... 190
8.1.4.180 contains() [3/5] .......................................... 190
8.1.4.181 contains() [4/5] .......................................... 190
8.1.4.182 contains() [5/5] .......................................... 190
8.1.4.183 is_overlapped() .......................................... 190
8.1.4.184 is_object3d() ............................................. 191
8.1.4.185 is_CImg3d() .............................................. 192
8.1.4.186 sqr() .......................................................... 192
8.1.4.187 sqrt() .......................................................... 193
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>exp()</code></td>
<td>193</td>
</tr>
<tr>
<td><code>log()</code></td>
<td>193</td>
</tr>
<tr>
<td><code>log2()</code></td>
<td>194</td>
</tr>
<tr>
<td><code>log10()</code></td>
<td>194</td>
</tr>
<tr>
<td><code>abs()</code></td>
<td>194</td>
</tr>
<tr>
<td><code>sign()</code></td>
<td>195</td>
</tr>
<tr>
<td><code>cos()</code></td>
<td>195</td>
</tr>
<tr>
<td><code>sin()</code></td>
<td>195</td>
</tr>
<tr>
<td><code>sinc()</code></td>
<td>196</td>
</tr>
<tr>
<td><code>tan()</code></td>
<td>196</td>
</tr>
<tr>
<td><code>cosh()</code></td>
<td>196</td>
</tr>
<tr>
<td><code>sinh()</code></td>
<td>197</td>
</tr>
<tr>
<td><code>tanh()</code></td>
<td>197</td>
</tr>
<tr>
<td><code>acos()</code></td>
<td>197</td>
</tr>
<tr>
<td><code>asin()</code></td>
<td>198</td>
</tr>
<tr>
<td><code>atan()</code></td>
<td>198</td>
</tr>
<tr>
<td><code>atan2()</code></td>
<td>198</td>
</tr>
<tr>
<td><code>acosh()</code></td>
<td>199</td>
</tr>
<tr>
<td><code>asinh()</code></td>
<td>199</td>
</tr>
<tr>
<td><code>atanh()</code></td>
<td>199</td>
</tr>
<tr>
<td><code>mul()</code></td>
<td>199</td>
</tr>
<tr>
<td><code>div()</code></td>
<td>200</td>
</tr>
<tr>
<td><code>pow()</code></td>
<td>200</td>
</tr>
<tr>
<td><code>rol()</code></td>
<td>201</td>
</tr>
<tr>
<td><code>ror()</code></td>
<td>202</td>
</tr>
</tbody>
</table>
8.1.4.248 set_matrix_at() ........................................ 215
8.1.4.249 set_tensor_at() ......................................... 217
8.1.4.250 vector() [1/5] ........................................ 217
8.1.4.251 diagonal() ................................................ 217
8.1.4.252 identity_matrix() [1/2] ............................... 218
8.1.4.253 sequence() [1/2] ....................................... 218
8.1.4.254 transpose() ............................................. 218
8.1.4.255 cross() .................................................. 218
8.1.4.256 invert() ................................................. 219
8.1.4.257 solve() .................................................. 219
8.1.4.258 solve_tridiagonal() .................................. 219
8.1.4.259 eigen() .................................................. 220
8.1.4.260 get_eigen() .............................................. 220
8.1.4.261 symmetric_eigen() .................................... 220
8.1.4.262 get_symmetric_eigen() ............................... 221
8.1.4.263 sort() [1/2] ............................................. 221
8.1.4.264 sort() [2/2] ............................................. 221
8.1.4.265 SVD() .................................................... 222
8.1.4.266 get_SVD() ............................................... 222
8.1.4.267 dijkstra() [1/2] ....................................... 223
8.1.4.268 dijkstra() [2/2] ....................................... 223
8.1.4.269 string() .................................................. 224
8.1.4.270 vector() [2/5] ......................................... 224
8.1.4.271 vector() [3/5] ......................................... 224
8.1.4.272 vector() [4/5] ......................................... 225
8.1.4.273 vector() [5/5] ......................................... 225
8.1.4.274 matrix() [1/3] ......................................... 225
8.1.4.275 matrix() [2/3] ......................................... 226
8.1.4.276 matrix() [3/3] ......................................... 226
8.1.4.277 tensor() .................................................. 227
8.1.4.278 identity_matrix() [2/2] .............................................. 227
8.1.4.279 sequence() [2/2] .................................................. 227
8.1.4.280 rotation_matrix() .................................................. 228
8.1.4.281 fill() [1/4] .......................................................... 228
8.1.4.282 fill() [2/4] .......................................................... 228
8.1.4.284 fill() [4/4] .......................................................... 229
8.1.4.285 fillX() .............................................................. 229
8.1.4.286 fillY() .............................................................. 230
8.1.4.287 fillZ() .............................................................. 230
8.1.4.288 fillC() .............................................................. 231
8.1.4.289 discard() ........................................................... 231
8.1.4.290 rand() .............................................................. 231
8.1.4.291 round() ............................................................. 232
8.1.4.292 noise() .............................................................. 232
8.1.4.293 normalize() [1/2] .................................................. 233
8.1.4.294 normalize() [2/2] .................................................. 233
8.1.4.295 norm() ............................................................... 234
8.1.4.296 cut() ................................................................. 234
8.1.4.297 quantize() .......................................................... 234
8.1.4.298 threshold() ........................................................ 235
8.1.4.299 histogram() ......................................................... 235
8.1.4.300 equalize() .......................................................... 236
8.1.4.301 index() ............................................................. 236
8.1.4.302 map() ............................................................... 237
8.1.4.303 label() [1/2] ......................................................... 237
8.1.4.304 label() [2/2] ......................................................... 238
8.1.4.305 default_LUT256() ................................................ 238
8.1.4.306 HSV_LUT256() ................................................... 238
8.1.4.307 lines_LUT256() .................................................... 239
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_slices()</td>
<td>250</td>
</tr>
<tr>
<td>get_channel()</td>
<td>250</td>
</tr>
<tr>
<td>get_channels()</td>
<td>250</td>
</tr>
<tr>
<td>streamline()</td>
<td>251</td>
</tr>
<tr>
<td>get_shared_points()</td>
<td>252</td>
</tr>
<tr>
<td>get_shared_rows()</td>
<td>252</td>
</tr>
<tr>
<td>get_shared_row()</td>
<td>252</td>
</tr>
<tr>
<td>get_shared_slices()</td>
<td>253</td>
</tr>
<tr>
<td>get_shared_slice()</td>
<td>253</td>
</tr>
<tr>
<td>get_shared_channels()</td>
<td>253</td>
</tr>
<tr>
<td>get_shared_channel()</td>
<td>254</td>
</tr>
<tr>
<td>get_split()</td>
<td>254</td>
</tr>
<tr>
<td>append()</td>
<td>255</td>
</tr>
<tr>
<td>correlate()</td>
<td>255</td>
</tr>
<tr>
<td>convolve()</td>
<td>256</td>
</tr>
<tr>
<td>cumulate()</td>
<td>257</td>
</tr>
<tr>
<td>erode()</td>
<td>259</td>
</tr>
<tr>
<td>dilate()</td>
<td>260</td>
</tr>
<tr>
<td>watershed()</td>
<td>261</td>
</tr>
<tr>
<td>deriche()</td>
<td>261</td>
</tr>
<tr>
<td>vanvliet()</td>
<td>262</td>
</tr>
<tr>
<td>blur()</td>
<td>262</td>
</tr>
<tr>
<td>blur_anisotropic()</td>
<td>263</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------</td>
<td>------</td>
</tr>
<tr>
<td>8.1.4.368 blur_anisotropic()</td>
<td>264</td>
</tr>
<tr>
<td>8.1.4.369 blur_bilateral()</td>
<td>264</td>
</tr>
<tr>
<td>8.1.4.370 blur_bilateral()</td>
<td>265</td>
</tr>
<tr>
<td>8.1.4.371 boxfilter()</td>
<td>266</td>
</tr>
<tr>
<td>8.1.4.372 blur_box()</td>
<td>266</td>
</tr>
<tr>
<td>8.1.4.373 blur_box()</td>
<td>267</td>
</tr>
<tr>
<td>8.1.4.374 blur_guided()</td>
<td>267</td>
</tr>
<tr>
<td>8.1.4.375 blur_patch()</td>
<td>267</td>
</tr>
<tr>
<td>8.1.4.376 blur_median()</td>
<td>268</td>
</tr>
<tr>
<td>8.1.4.377 sharpen()</td>
<td>268</td>
</tr>
<tr>
<td>8.1.4.378 get_gradient()</td>
<td>269</td>
</tr>
<tr>
<td>8.1.4.379 get_hessian()</td>
<td>269</td>
</tr>
<tr>
<td>8.1.4.380 structure_tensors()</td>
<td>269</td>
</tr>
<tr>
<td>8.1.4.381 diffusion_tensors()</td>
<td>270</td>
</tr>
<tr>
<td>8.1.4.382 displacement()</td>
<td>270</td>
</tr>
<tr>
<td>8.1.4.383 matchpatch()</td>
<td>271</td>
</tr>
<tr>
<td>8.1.4.384 distance()</td>
<td>271</td>
</tr>
<tr>
<td>8.1.4.385 distance()</td>
<td>272</td>
</tr>
<tr>
<td>8.1.4.386 distance_dijkstra()</td>
<td>272</td>
</tr>
<tr>
<td>8.1.4.387 distance_eikonal()</td>
<td>273</td>
</tr>
<tr>
<td>8.1.4.388 distance_eikonal()</td>
<td>273</td>
</tr>
<tr>
<td>8.1.4.389 haar()</td>
<td>273</td>
</tr>
<tr>
<td>8.1.4.390 haar()</td>
<td>274</td>
</tr>
<tr>
<td>8.1.4.391 get_FFT()</td>
<td>274</td>
</tr>
<tr>
<td>8.1.4.392 FFT()</td>
<td>274</td>
</tr>
<tr>
<td>8.1.4.393 FFT()</td>
<td>275</td>
</tr>
<tr>
<td>8.1.4.394 shift_object3d()</td>
<td>275</td>
</tr>
<tr>
<td>8.1.4.395 shift_object3d()</td>
<td>275</td>
</tr>
<tr>
<td>8.1.4.396 resize_object3d()</td>
<td>276</td>
</tr>
<tr>
<td>8.1.4.397 append_object3d()</td>
<td>276</td>
</tr>
</tbody>
</table>
8.1.4.398 texturize_object3d() .......................................................... 276
8.1.4.399 get_elevation3d() .............................................................. 277
8.1.4.400 get_projections3d() ........................................................... 277
8.1.4.401 get_isoline3d() ................................................................. 278
8.1.4.402 get_isosurface3d() ............................................................. 278
8.1.4.403 elevation3d() ................................................................. 279
8.1.4.404 isoline3d() ................................................................. 279
8.1.4.405 isosurface3d() ................................................................. 280
8.1.4.406 box3d() ........................................................................ 281
8.1.4.407 cone3d() ................................................................ ...... 281
8.1.4.408 cylinder3d() ................................................................. 282
8.1.4.409 torus3d() ...................................................................... 282
8.1.4.410 plane3d() ...................................................................... 283
8.1.4.411 sphere3d() ...................................................................... 284
8.1.4.412 ellipsoid3d() ................................................................... 284
8.1.4.413 object3dtoCImg3d() ............................................................. 285
8.1.4.414 CImg3dtoobject3d() ............................................................ 285
8.1.4.415 draw_point() [1/2] .............................................................. 285
8.1.4.416 draw_point() [2/2] .............................................................. 286
8.1.4.417 draw_line() [1/6] ............................................................... 286
8.1.4.418 draw_line() [2/6] ............................................................... 287
8.1.4.419 draw_line() [3/6] ............................................................... 288
8.1.4.420 draw_line() [4/6] ............................................................... 288
8.1.4.421 draw_line() [5/6] ............................................................... 289
8.1.4.422 draw_line() [6/6] ............................................................... 291
8.1.4.423 draw_arrow() ................................................................. 291
8.1.4.424 draw_spline() [1/4] ........................................................... 292
8.1.4.425 draw_spline() [2/4] ........................................................... 293
8.1.4.426 draw_spline() [3/4] ........................................................... 294
| 8.1.4.458 | draw_grid() | 312 |
| 8.1.4.459 | draw_graph() | 312 |
| 8.1.4.460 | draw_fill() | 313 |
| 8.1.4.461 | draw_plasma() | 314 |
| 8.1.4.462 | draw_mandelbrot() | 314 |
| 8.1.4.463 | draw_gaussian() [1/2] | 315 |
| 8.1.4.464 | draw_gaussian() [2/2] | 316 |
| 8.1.4.465 | draw_object3d() | 316 |
| 8.1.4.466 | select() | 317 |
| 8.1.4.467 | load() | 317 |
| 8.1.4.468 | load_ascii() | 318 |
| 8.1.4.469 | load_dlm() | 318 |
| 8.1.4.470 | load_bmp() | 318 |
| 8.1.4.471 | load_jpeg() | 319 |
| 8.1.4.472 | load_magick() | 319 |
| 8.1.4.473 | load_png() | 319 |
| 8.1.4.474 | load_pnm() | 319 |
| 8.1.4.475 | load_pfm() | 321 |
| 8.1.4.476 | load_rgb() | 321 |
| 8.1.4.477 | load_rgba() | 321 |
| 8.1.4.478 | load_tiff() | 322 |
| 8.1.4.479 | load_minc2() | 322 |
| 8.1.4.480 | load_analyze() | 322 |
| 8.1.4.481 | load_cimg() [1/2] | 323 |
| 8.1.4.482 | load_cimg() [2/2] | 323 |
| 8.1.4.483 | load_inr() | 324 |
| 8.1.4.484 | load_exr() | 324 |
| 8.1.4.485 | load_pandore() | 325 |
| 8.1.4.486 | load_parrec() | 325 |
| 8.1.4.487 | load_raw() | 325 |
8.1.4.488 load_yuv() .................................................. 326
8.1.4.489 load_off() .................................................. 326
8.1.4.490 load_video() ................................................. 327
8.1.4.491 load_ffmpeg_external() ................................. 327
8.1.4.492 load_gif_external() ...................................... 328
8.1.4.493 load_graphicsmagick_external() ................... 328
8.1.4.494 load_gzip_external() .................................... 328
8.1.4.495 load_imagemagick_external() ......................... 328
8.1.4.496 load_medcon_external() ............................... 329
8.1.4.497 load_dcraw_external() .................................. 329
8.1.4.498 load_camera() ............................................. 329
8.1.4.499 load_other() ............................................... 330
8.1.4.500 print() ..................................................... 330
8.1.4.501 display() [1/3] ............................................ 330
8.1.4.502 display() [2/3] ............................................ 330
8.1.4.503 display() [3/3] ............................................ 331
8.1.4.504 display_object3d() ..................................... 331
8.1.4.505 display_graph() .......................................... 332
8.1.4.506 save() .................................................... 333
8.1.4.507 save_ascii() .............................................. 333
8.1.4.508 save_cpp() ............................................... 334
8.1.4.509 save_dlm() ............................................... 334
8.1.4.510 save_bmp() ............................................... 334
8.1.4.511 save_jpeg() .............................................. 334
8.1.4.512 save_magick() .......................................... 335
8.1.4.513 save_png() ............................................... 335
8.1.4.514 save_pnm() ............................................... 335
8.1.4.515 save_pnk() ............................................... 336
8.1.4.516 save_pfm() ............................................... 336
8.1.4.517 save_rgb() ............................................... 336
8.1.4.518 save_rgba() ................................................. 336
8.1.4.519 save_tiff() .................................................. 337
8.1.4.520 save_minc2() .................................................. 337
8.1.4.521 save_analyze() .................................................. 338
8.1.4.522 save_cimg() [1/2] ........................................... 338
8.1.4.523 save_cimg() [2/2] ........................................... 338
8.1.4.524 save_empty_cimg() [1/2] ................................... 339
8.1.4.525 save_empty_cimg() [2/2] ................................... 339
8.1.4.526 save_inr() .................................................... 340
8.1.4.527 save_exr() .................................................... 340
8.1.4.528 save_pandore() [1/2] ...................................... 340
8.1.4.529 save_pandore() [2/2] ...................................... 341
8.1.4.530 save_raw() [1/2] ............................................. 341
8.1.4.531 save_raw() [2/2] ............................................. 341
8.1.4.532 save_yuv() [1/2] ............................................ 341
8.1.4.533 save_yuv() [2/2] ............................................ 342
8.1.4.534 save_off() [1/2] ............................................. 342
8.1.4.535 save_off() [2/2] ............................................. 343
8.1.4.536 save_video() ................................................ 343
8.1.4.537 save_ffmpeg_external() .................................. 343
8.1.4.538 save_gzip_external() ..................................... 344
8.1.4.539 save_graphicsmagick_external() ....................... 344
8.1.4.540 save_imagemagick_external() ............................ 344
8.1.4.541 save_medcon_external() ................................ 345
8.1.4.542 save_other() ............................................... 345
8.1.4.543 get_serialize() ............................................ 346

8.2 CImgDisplay Struct Reference .................................. 346
  8.2.1 Detailed Description ........................................... 352
  8.2.2 Constructor & Destructor Documentation .................. 352
    8.2.2.1 ~CImgDisplay() .......................................... 353
8.2.2.2 CImgDisplay() ........................................ 353
8.2.2.3 CImgDisplay() ........................................ 353
8.2.2.4 CImgDisplay() ........................................ 354
8.2.2.5 CImgDisplay() ........................................ 354
8.2.2.6 CImgDisplay() ........................................ 355

8.2.3 Member Function Documentation .................................. 355
8.2.3.1 screenshot() .......................................... 355
8.2.3.2 assign() ........................................... 355
8.2.3.3 empty() ............................................. 356
8.2.3.4 operator=() ......................................... 356
8.2.3.5 operator=() ......................................... 356
8.2.3.6 operator=() ......................................... 356
8.2.3.7 operator bool() ...................................... 357
8.2.3.8 is_closed() ........................................... 357
8.2.3.9 is_key() ............................................. 357
8.2.3.10 is_key() ............................................ 357
8.2.3.11 is_key() ............................................ 358
8.2.3.12 is_key_sequence() .................................. 358
8.2.3.13 is_keyESC() ......................................... 359
8.2.3.14 width() ............................................. 359
8.2.3.15 height() ............................................. 359
8.2.3.16 normalization() ..................................... 360
8.2.3.17 title() ............................................. 360
8.2.3.18 window_width() .................................... 360
8.2.3.19 window_height() ................................... 361
8.2.3.20 window_x() ......................................... 361
8.2.3.21 window_y() ......................................... 361
8.2.3.22 mouse_x() .......................................... 361
8.2.3.23 mouse_y() .......................................... 362
8.2.3.24 button() ............................................ 362
| 8.2.3.25 | wheel() | 363 |
| 8.2.3.26 | key() | 363 |
| 8.2.3.27 | released_key() | 364 |
| 8.2.3.28 | keycode() | 364 |
| 8.2.3.29 | frames_per_second() | 365 |
| 8.2.3.30 | display() [1/2] | 365 |
| 8.2.3.31 | display() [2/2] | 365 |
| 8.2.3.32 | show() | 366 |
| 8.2.3.33 | close() | 366 |
| 8.2.3.34 | move() | 366 |
| 8.2.3.35 | resize() [1/4] | 367 |
| 8.2.3.36 | resize() [2/4] | 367 |
| 8.2.3.37 | resize() [3/4] | 368 |
| 8.2.3.38 | resize() [4/4] | 368 |
| 8.2.3.39 | set_normalization() | 368 |
| 8.2.3.40 | set_title() | 369 |
| 8.2.3.41 | set_fullscreen() | 369 |
| 8.2.3.42 | toggle_fullscreen() | 370 |
| 8.2.3.43 | show_mouse() | 370 |
| 8.2.3.44 | hide_mouse() | 370 |
| 8.2.3.45 | set_mouse() | 371 |
| 8.2.3.46 | set_button() [1/2] | 371 |
| 8.2.3.47 | set_button() [2/2] | 371 |
| 8.2.3.48 | set_wheel() [1/2] | 371 |
| 8.2.3.49 | set_wheel() [2/2] | 372 |
| 8.2.3.50 | set_key() [1/2] | 372 |
| 8.2.3.51 | set_key() [2/2] | 372 |
| 8.2.3.52 | flush() | 373 |
| 8.2.3.53 | wait() | 373 |
| 8.2.3.54 | render() | 373 |
8.2.3.55  paint() ........................................ 374
8.2.3.56  screenshot() [2/2] .............................. 374
8.2.3.57  snapshot() ..................................... 374

8.3  CImgException Struct Reference ........................ 375
  8.3.1  Detailed Description .................................. 375

8.4  CImgList<T> Struct Template Reference ............. 376
  8.4.1  Detailed Description .................................. 390
  8.4.2  Member Typedef Documentation ....................... 390
    8.4.2.1  iterator ...................................... 390
    8.4.2.2  const_iterator ................................ 390
    8.4.2.3  value_type ................................... 391
  8.4.3  Constructor & Destructor Documentation ............ 391
    8.4.3.1  ~CImgList() .................................. 391
    8.4.3.2  CImgList() [1/19] .............................. 391
    8.4.3.3  CImgList() [2/19] .............................. 391
    8.4.3.4  CImgList() [3/19] .............................. 392
    8.4.3.5  CImgList() [4/19] .............................. 392
    8.4.3.6  CImgList() [5/19] .............................. 393
    8.4.3.7  CImgList() [6/19] .............................. 393
    8.4.3.8  CImgList() [7/19] .............................. 394
    8.4.3.9  CImgList() [8/19] .............................. 394
    8.4.3.10 CImgList() [9/19] ............................. 395
    8.4.3.11 CImgList() [10/19] ........................... 395
    8.4.3.12 CImgList() [11/19] ........................... 395
    8.4.3.13 CImgList() [12/19] ........................... 396
    8.4.3.14 CImgList() [13/19] ........................... 396
    8.4.3.15 CImgList() [14/19] ........................... 397
    8.4.3.16 CImgList() [15/19] ........................... 397
    8.4.3.17 CImgList() [16/19] ........................... 398
    8.4.3.18 CImgList() [17/19] ........................... 398
8.4.3.19 CImgList() [18/19] ........................................ 399
8.4.3.20 CImgList() [19/19] ........................................ 399

8.4.4 Member Function Documentation .................................. 399

8.4.4.1 get_shared() .................................................. 399
8.4.4.2 assign() [1/18] ............................................... 400
8.4.4.3 clear() ......................................................... 400
8.4.4.4 assign() [2/18] ............................................... 400
8.4.4.5 assign() [3/18] ............................................... 400
8.4.4.6 assign() [4/18] ............................................... 401
8.4.4.7 assign() [5/18] ............................................... 401
8.4.4.8 assign() [6/18] ............................................... 401
8.4.4.9 assign() [7/18] ............................................... 402
8.4.4.10 assign() [8/18] .............................................. 402
8.4.4.11 assign() [9/18] .............................................. 402
8.4.4.12 assign() [10/18] ............................................. 403
8.4.4.13 assign() [11/18] ............................................. 403
8.4.4.14 assign() [12/18] ............................................. 403
8.4.4.15 assign() [13/18] ............................................. 404
8.4.4.16 assign() [14/18] ............................................. 404
8.4.4.17 assign() [15/18] ............................................. 404
8.4.4.18 assign() [16/18] ............................................. 405
8.4.4.19 assign() [17/18] ............................................. 405
8.4.4.20 assign() [18/18] ............................................. 405
8.4.4.21 move_to() [1/2] ............................................. 405
8.4.4.22 move_to() [2/2] ............................................. 406
8.4.4.23 swap() ......................................................... 406
8.4.4.24 empty() ....................................................... 407
8.4.4.25 operator()() [1/3] ........................................... 407
8.4.4.26 operator()() [2/3] ........................................... 407
8.4.4.27 operator()() [3/3] ........................................... 407
8.4.4.28 operator Clmg<T> *( ) ........................................ 408
8.4.4.29 operator=() [1/4] ........................................ 408
8.4.4.30 operator=() [2/4] ........................................ 408
8.4.4.31 operator=() [3/4] ........................................ 409
8.4.4.32 operator=() [4/4] ........................................ 409
8.4.4.33 operator+() ................................................. 409
8.4.4.34 operator,() [1/2] ............................................ 409
8.4.4.35 operator,() [2/2] ............................................ 410
8.4.4.36 operator>() .................................................. 410
8.4.4.37 operator<() .................................................. 410
8.4.4.38 pixel_type() .................................................. 411
8.4.4.39 width() ....................................................... 411
8.4.4.40 size() ......................................................... 411
8.4.4.41 data() [1/2] ................................................... 412
8.4.4.42 data() [2/2] ................................................... 412
8.4.4.43 at() ............................................................ 412
8.4.4.44 atNXYZC() [1/2] ............................................. 412
8.4.4.45 atNXYZC() [2/2] ............................................. 413
8.4.4.46 atNXYZ() [1/2] ................................................. 413
8.4.4.47 atNXYZ() [2/2] ................................................. 414
8.4.4.48 atNXY() [1/2] .................................................. 415
8.4.4.49 atNXY() [2/2] .................................................. 415
8.4.4.50 atNX() [1/2] ................................................... 416
8.4.4.51 atNX() [2/2] ................................................... 416
8.4.4.52 atN() [1/2] .................................................... 417
8.4.4.53 atN() [2/2] .................................................... 417
8.4.4.54 is_sameN() [1/2] ............................................. 418
8.4.4.55 is_sameN() [2/2] ............................................. 418
8.4.4.56 is_sameXYZC() .............................................. 418
8.4.4.57 is_sameNXYZC() ............................................ 418
8.4.4.58 containsNXYZC() .................................................. 419
8.4.4.59 containsN() ......................................................... 419
8.4.4.60 contains() [1/8] .................................................... 420
8.4.4.61 contains() [2/8] .................................................... 420
8.4.4.62 contains() [3/8] .................................................... 421
8.4.4.63 contains() [4/8] .................................................... 421
8.4.4.64 contains() [5/8] .................................................... 422
8.4.4.65 contains() [6/8] .................................................... 422
8.4.4.66 contains() [7/8] .................................................... 422
8.4.4.67 contains() [8/8] .................................................... 423
8.4.4.68 min_max() [1/2] ..................................................... 423
8.4.4.69 min_max() [2/2] ..................................................... 423
8.4.4.70 max_min() .......................................................... 423
8.4.4.71 insert() [1/5] ......................................................... 424
8.4.4.72 insert() [2/5] ......................................................... 424
8.4.4.73 insert() [3/5] ......................................................... 424
8.4.4.74 insert() [4/5] ......................................................... 425
8.4.4.75 insert() [5/5] ......................................................... 425
8.4.4.76 remove() [1/2] ....................................................... 425
8.4.4.77 remove() [2/2] ....................................................... 426
8.4.4.78 images() ........................................................... 426
8.4.4.79 get_shared_images() .............................................. 426
8.4.4.80 get_append() ....................................................... 427
8.4.4.81 split() .............................................................. 427
8.4.4.82 push_back() [1/2] .................................................. 427
8.4.4.83 push_front() [1/2] .................................................. 428
8.4.4.84 push_back() [2/2] .................................................. 428
8.4.4.85 push_front() [2/2] .................................................. 428
8.4.4.86 erase() ............................................................. 428
8.4.4.87 get_select() [1/2] .................................................. 429
| 8.4.4.88 | get_select() | 2/2 | 429 |
| 8.4.4.89 | load() | | 430 |
| 8.4.4.90 | load_cimg() | 1/3 | 430 |
| 8.4.4.91 | load_cimg() | 2/3 | 430 |
| 8.4.4.92 | load_cimg() | 3/3 | 430 |
| 8.4.4.93 | load_parrec() | | 431 |
| 8.4.4.94 | load_yuv() | | 431 |
| 8.4.4.95 | load_video() | | 432 |
| 8.4.4.96 | load_ffmpeg_external() | | 432 |
| 8.4.4.97 | load_gif_external() | | 433 |
| 8.4.4.98 | load_gzip_external() | | 433 |
| 8.4.4.99 | load_tiff() | | 433 |
| 8.4.4.100 | print() | | 434 |
| 8.4.4.101 | display() | 1/3 | 434 |
| 8.4.4.102 | display() | 2/3 | 435 |
| 8.4.4.103 | display() | 3/3 | 435 |
| 8.4.4.104 | save() | | 436 |
| 8.4.4.105 | is_saveable() | | 436 |
| 8.4.4.106 | save_gif_external() | | 436 |
| 8.4.4.107 | save_yuv() | 1/2 | 437 |
| 8.4.4.108 | save_yuv() | 2/2 | 437 |
| 8.4.4.109 | save_cimg() | 1/4 | 437 |
| 8.4.4.110 | save_cimg() | 2/4 | 438 |
| 8.4.4.111 | save_cimg() | 3/4 | 438 |
| 8.4.4.112 | save_cimg() | 4/4 | 438 |
| 8.4.4.113 | save_empty_cimg() | 1/2 | 440 |
| 8.4.4.114 | save_empty_cimg() | 2/2 | 440 |
| 8.4.4.115 | save_tiff() | | 441 |
| 8.4.4.116 | save_gzip_external() | | 441 |
| 8.4.4.117 | save_video() | | 441 |
| 8.4.4.118 | save_ffmpeg_external() | | 442 |
| 8.4.4.119 | get_serialize() | | 442 |
| 8.4.4.120 | font() | | 443 |
| 8.4.4.121 | FFT() | 1/2 | 443 |
| 8.4.4.122 | FFT() | 2/2 | 443 |
Chapter 1

Main Page

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

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 ........................................ 33
- `cimg_library::cimg`
  Contains low-level functions and variables of the CImg Library ................................ 34
Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

CImg< T > ................................................................. 67
CImg< charT > .......................................................... 67
CImg< doubleT > ...................................................... 67
CImg< floatT > .......................................................... 67
CImg< intT > ............................................................ 67
CImg< uintT > .......................................................... 67
CImg< ulongT > ........................................................ 67
CImgDisplay ............................................................ 346
CImgException ......................................................... 375
CImgList< T > .......................................................... 376
CImgList< boolT > ..................................................... 376
CImgList< charT > ..................................................... 376
CImgList< doubleT > ................................................ 376
CImgList< ulongT > ................................................... 376
Chapter 5

Class Index

5.1 Class List

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

\texttt{CImg<T>}
\quad Class representing an image (up to 4 dimensions wide), each pixel being of type \( T \) \ldots 67

\texttt{CImgDisplay}
\quad Allow the creation of windows, display images on them and manage user events (keyboard, mouse and windows events) \ldots 346

\texttt{CImgException}
\quad Instances of \texttt{CImgException} are thrown when errors are encountered in a \texttt{CImg} function call 375

\texttt{CImgList<T>}
\quad Represent a list of images \texttt{CImg<T>} \ldots 376
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> img(640,400,1,3); // Define a 640x400 color image with 8 bits per color component.
    img.fill(0); // Set pixel values to 0 (color : black)
    unsigned char purple[] = { 255,0,255 }; // Define a purple color
    img.draw_text(100,100,"Hello World",purple); // Draw a purple "Hello world" at coordinates (100,100).
    img.display("My first CImg code"); // Display the image in a display window.
    return 0;
}
```

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

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

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

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

6.1.3 How to compile ?

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

- **Intel ICL compiler**: Use the following command to compile a CImg-based program with ICL:

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

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

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

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

  ```
  g++ -o hello_world.exe hello_world.cpp -O2 -L/usr/X11R6/lib -lm -lpthread -lX11
  ```

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

  ```
  g++ -o hello_world.exe hello_world.cpp -O2 -lm -lpthread -R/usr/X11R6/lib -lrt -lnsl -lsocket
  ```

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

  ```
  g++ -o hello_world.exe hello_world.cpp -O2 -lm -lpthread -L/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.
6.2.3 2. C++ related questions

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

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

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

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

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

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

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

- First, the library is based on template datatypes (images with generic pixel type), meaning that the programmer is free to decide what type of image he instantiates in his code. Even if there are roughly a limited number of fully supported types (basically, the "atomic" types of C++ : unsigned char, int, float, ...), this is not imaginable to pre-compile the library classes and functions for all possible atomic datatypes, since many functions and methods can have two or three arguments having different template parameters. This really means a huge number of possible combinations. The size of the object binary file generated to cover all possible cases would be just colossal. Is the STL library a pre-compiled one? No, CImg neither. CImg is not using a classical .cpp and .h mechanism, just like the STL. Architectures of C++ template-based libraries are somewhat special in this sense. This is a proven technical fact.

- Second, why CImg does not have several header files, just like the STL does (one for each class for instance)? This would be possible of course. There are only 4 classes in CImg, the two most important being CImg<T> and CImgList<T> representing respectively an image and a collection of images. But contrary to the STL library, these two CImg classes are strongly inter-dependent. All CImg algorithms are actually not defined as separate functions acting on containers (as the STL does with his header <algorithm>), but are directly methods of the image and image collection classes. This inter-dependence practically means that you will undoubtly need these two main classes at the same time if you are using CImg. If they were defined in separate header files, you would be forced to include both of them. What is the gain then? No gain. Concerning the two other classes : You can disable the third most important class CImgDisplay of the CImg library, by setting the compilation macro cimg_display to 0, avoiding thus to compile this class if you don't use display capabilities of CImg in your code. But to be honest, this is a quite small class and doing this doesn't save much compilation time. The last and fourth class is CImgException, which is only few lines long and is obviously required in almost all methods of CImg. Including this one is mandatory. As a consequence, having a single header file instead of several ones is just a way for you to avoid including all of them, without any consequences on compilation time. This is both good technical and practical reasons to do like this.
• Third, having a single header file has plenty of advantages: Simplicity for the user, and for the developers (maintenance is in fact easier). Look at the CImg.h file, it looks like a mess at a first glance, but it is in fact very well organized and structured. Finding pieces of code in CImg functions or methods is particularly easy and fast. Also, how about the fact that library installation problems just disappear? Just bring CImg.h with you, put it in your source directory, and the library is ready to go!

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

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:

```cpp
#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.

- **cimglister_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. Replace the line AdditionalDependencies="kernel32.lib" by AdditionalDependencies="kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib"
- Restart Visual C++
- Import CImg library in your main file
6.5 Tutorial: Getting Started.

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

using namespace cimg_library;

Use the library namespace to ease the declarations afterward.

int main() {

Definition of the main function.

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

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

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

Definition of three different colors as array of unsigned char. This will be used to draw plots with different colors.
```
image.blur(2.5);

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

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

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

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

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

```
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);
}
```

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

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

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

```
This line illustrates the pipeline property of most of the CImg class functions. The first function `fill(0)` simply sets all pixel values with 0 (i.e. clear the image visu). The interesting thing is that it returns a reference to visu and then, can be pipelined with the function `draw_graph()` which draws a plot in the image visu. The plot data are given by another image (the first argument of `draw_graph()`). In this case, the given image is the red-component of the line y of the original image, retrieved by the function `get_crop()` which returns a sub-image of the image image. Remember that images coordinates are 4D \((x,y,z,c)\) and for color images, the R,G,B channels are respectively given by \(v=0\), \(v=1\) and \(v=2\).

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

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

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

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

```
...till the end
I don't think you need more explanations!
```

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

The CImg Library provides different macros that define useful iterative loops over an image. Basically, it can be used to replace one or several \texttt{for(\ldots)} 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 \texttt{cimg\_library::CImg} image. Two macros are defined for this purpose:

- \textbf{cimg\_for(img,ptr,T)}: This macro loops over the pixel data buffer of the image \texttt{img}, using a pointer \texttt{T* ptr}, starting from the beginning of the buffer (first pixel) till the end of the buffer (last pixel).
  
  - \texttt{img} must be a (non empty) \texttt{cimg\_library::CImg} image of pixels \texttt{T}.
  
  - \texttt{ptr} is a pointer of type \texttt{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:

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

- \textbf{cimg\_rof(img,ptr,T)}: This macro does the same as \textbf{cimg\_for()} but from the end to the beginning of the pixel buffer.

- \textbf{cimg\_foroff(img,off)}: This macro loops over the pixel data buffer of the image \texttt{img}, using an offset \texttt{off}, starting from the beginning of the buffer (first pixel, \texttt{off=0}) till the end of the buffer (last pixel value, \texttt{off = img.size()-1}).

  - \texttt{img} must be a (non empty) \texttt{cimg\_library::CImg\<T\>} image of pixels \texttt{T}.
  
  - \texttt{off} is an inner-loop variable, only defined inside the scope of the loop.

Here is an example of use:

\begin{verbatim}
CImg<float> img(320,200);
cimg_foroff(img,off) { img[off]=0; } // Equivalent to 'img.fill(0);'
\end{verbatim}
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: `for (int x = 0; x < img.width(); ++x).
- `cimg_forY(img, y)` equivalent to: `for (int y = 0; y < img.height(); ++y).
- `cimg_forZ(img, z)` equivalent to: `for (int z = 0; z < img.depth(); ++z).
- `cimg_forC(img, c)` equivalent to: `for (int c = 0; c < img.spectrum(); ++c).

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

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

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

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

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

```cpp
CImg<unsigned char> img(256, 256, 1, 3); // Define a 256x256 color image
for (int x = 0; x < img.width(); ++x) { 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<uchar> 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:

- `cimg_for2x2(img,x,y,z,c,I,T)` : Loop along the (x,y)-axes using a centered 2x2 neighborhood.
- `cimg_for3x3(img,x,y,z,c,I,T)` : Loop along the (x,y)-axes using a centered 3x3 neighborhood.
- `cimg_for4x4(img,x,y,z,c,I,T)` : Loop along the (x,y)-axes using a centered 4x4 neighborhood.
- `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 `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 Defining neighborhoods).

6.6.4.2 Neighborhood-based loops for 3D images

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

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

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

6.6.4.3 Defining neighborhoods

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

For instance, a 3x3 neighborhood can be defined either as a 'float[9]' or a 'CImg<float>(3,3)' variable.
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, Inn, Ipc, Icc, Inc, Ipn, Icn, Inn` which correspond to each pixel value of a 3x3 neighborhood. Variable indices are p, c or n, and stand respectively for 'previous', 'current' and 'next'. First indice denotes the x-axis, second indice denotes the y-axis. Then, the names of the variables are directly related to the position of the corresponding pixels in the neighborhood. For 3D neighborhoods, a third indice denotes the z-axis. Then, inside a neighborhood loop, you will have the following equivalence:

- `Ipp = img(x-1,y-1)`
- `Icn = img(x,y+1)`
- `Inp = img(x+1,y-1)`
- `Inpc = img(x+1,y-1,z)`
- `Ippn = img(x-1,y-1,z+1)`
- and so on...

For bigger neighborhoods, such as 4x4 or 5x5 neighborhoods, two 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-Ipc); // 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<uchar> N(5,5); // Define a 5x5 neighborhood as a 5x5 image.
cimg_forC(src,k) // Standard loop on color channels
cimg_for5x5(src,x,y,0,k,N,float) // 5x5 neighborhood loop.
  dest(x,y,k) = N.sum()/(5*5); // Averaging pixels to filter the color image.
cImgList<unsigned char> visu(src,dest); visu.display("Original + Filtered"); // Display both original and filtered image.
```

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

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

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

- 0: No pixel normalization is performed when displaying an image. This is the fastest process, but you must be sure your displayed image have pixel values inside the range [0,255].
- 1: Pixel value normalization is done for each new image display. Image pixels are not modified themselves, only displayed pixels are normalized.
- 2: Pixel value normalization is done for the first image display, then the normalization parameters are kept and used for all the next image displays.
First, Clmg\(<\text{T}\>\) 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 Clmg, if you have only 64kb of RAM).

A Clmg\(<\text{T}\>\) is basically a 4th-dimensional array (width, height, depth, dim), and its pixel data are stored linearly in a single memory buffer of general size (width\(\times\)height\(\times\)depth\(\times\)dim). Nothing more, nothing less. The address of this memory buffer can be retrieved by the function Clmg\(<\text{T}\>\)::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 \(\text{const } T^*\) if your image is \(\text{const}\). 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\(\times\)height\(\times\)depth\(\times\)dim\(\times\)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 dim=3 and depth=1, will be stored in memory as:

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

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

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

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 \(\text{img(x,y,k)}\) instead of \(\text{img(x,y,0,k)}\) to access the kth channel of the \((x,y)\) pixel. More generally, if you have one dimension that is 1 in your image, you can just skip it in the call to the operator(). Similarly, values of a column vector stored as an image with width=depth=spectrum=1 can be accessed by \(\text{img(y)}\) instead of \(\text{img(0,y)}\). This is very convenient.

Another cool thing is that it allows you to work easily with 'shared' images. A shared image is a Clmg\(<\text{T}\>\) 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).
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.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>usage</td>
<td>A string describing the program goal and usage.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the option to be retrieved from the command line.</td>
</tr>
<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 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 `cimg_usage()` and `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", "Output image file"),
  const double sigma = cimg_option("-s", 1.0, "Standard variation of the gaussian smoothing"),
  const int nlevels = cimg_option("-n", 16, "Number of quantification levels"),
  const bool hidden = (char *)0,   
    "Output image file" ),
  return 0;
}

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

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

test : Retrieve command line arguments (Oct 16 2004, 12:34:26)
   
   -i  = foo.jpg : Input image file
   -o  = 0 : Output image file
   -s  = 1 : Standard variation of the gaussian smoothing
   -n  = 20 : Number of quantification levels

You found me !
```

Warning

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

```cpp
const double sigma = cimg_option("-val", 5.0, "A floating point value");
```

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

6.10.5 How to learn more about command line options?

You should take a look at the examples `examples/gmic.cpp` provided in the CImg Library package. This is a command line based image converter which intensively uses the `cimg_option()` and `cimg_usage()` macros to retrieve command line parameters.
Chapter 7

Namespace Documentation

7.1 cimg_library Namespace Reference

Contains all classes and functions of the CImg library.

Namespaces

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

Classes

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

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

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

- `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.
7.2 cimg_library::cimg Namespace Reference

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

Functions

- `std::FILE * output (std::FILE *file)`
  Get/set default output stream for the CImg library messages.
- `void info ()`
  Print information about CImg environment variables.
- `template<typename T >
  void unused (const T &,...)`
  Avoid warning messages due to unused parameters. 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)`
- `template<typename T >
  T & temporary (const T &)`
  Return a reference to a temporary variable of type T.
- `template<typename T >
  void swap (T &a, T &b)`
  Exchange values of variables a and b.
- `template<typename T1 , typename T2 >
  void swap (T1 &a1, T1 &b1, T2 &a2, T2 &b2)`
  Exchange values of variables (a1,a2) and (b1,b2).
- `template<typename T1 , typename T2 , typename T3 >
  void swap (T1 &a1, T1 &b1, T2 &a2, T2 &b2, T3 &a3, T3 &b3)`
  Exchange values of variables (a1,a2,a3) and (b1,b2,b3).
- `template<typename T1 , typename T2 , typename T3 , typename T4 >
  void swap (T1 &a1, T1 &b1, T2 &a2, T2 &b2, T3 &a3, T3 &b3, T4 &a4, T4 &b4)`
  Exchange values of variables (a1,a2,a3,a4) and (b1,b2,b3,b4).
- `template<typename T1 , typename T2 , typename T3 , typename T4 , typename T5 >
  void swap (T1 &a1, T1 &b1, T2 &a2, T2 &b2, T3 &a3, T3 &b3, T4 &a4, T4 &b4, T5 &a5, T5 &b5)`
  Exchange values of variables (a1,a2,...,a5) and (b1,b2,...,b5).
- `template<typename T1 , typename T2 , typename T3 , typename T4 , typename T5 , typename T6 >
  void swap (T1 &a1, T1 &b1, T2 &a2, T2 &b2, T3 &a3, T3 &b3, T4 &a4, T4 &b4, T5 &a5, T5 &b5, T6 &a6, T6 &b6)`
Exchange values of variables \((a_1, a_2, \ldots, a_6)\) and \((b_1, b_2, \ldots, b_6)\).

- 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 \((a_1, a_2, \ldots, a_7)\) and \((b_1, b_2, \ldots, b_7)\).

- 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 \((a_1, a_2, \ldots, a_8)\) and \((b_1, b_2, \ldots, b_8)\).

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

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

  Reverse endianness of all elements in a memory buffer.

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

  Reverse endianness of a single variable.

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

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

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

- void sleep (const unsigned int milliseconds)
  Sleep for a given numbers of milliseconds.

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

- template<typename T, 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 arccosine 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 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 \texttt{cbrt} (const T &x)
  \textit{Return cubic root of a value.}
• template<typename t >
  t \texttt{min} (const t &a, const t &b, const t &c)
  \textit{Return the minimum between three values.}
• template<typename t >
  t \texttt{min} (const t &a, const t &b, const t &c, const t &d)
  \textit{Return the minimum between four values.}
• template<typename t >
  t \texttt{minabs} (const t &a, const t &b)
  \textit{Return the minabs between two values.}
• template<typename t >
  t \texttt{max} (const t &a, const t &b, const t &c)
  \textit{Return the maximum between three values.}
• template<typename t >
  t \texttt{max} (const t &a, const t &b, const t &c, const t &d)
  \textit{Return the maximum between four values.}
• template<typename t >
  t \texttt{maxabs} (const t &a, const t &b)
  \textit{Return the maxabs between two values.}
• template<typename T >
  T \texttt{sign} (const T &x)
  \textit{Return the sign of a value.}
• template<typename T >
  unsigned long long \texttt{nearest\_pow2} (const T &x)
  \textit{Return the nearest power of 2 higher than given value.}
• template<typename T >
  T \texttt{mod} (const T &x, const T &m)
  \textit{Return the modulo of a value.}
• template<typename T >
  T \texttt{minmod} (const T &a, const T &b)
  \textit{Return the min-mod of two values.}
• template<typename T >
  T \texttt{round} (const T &x, const double y, const int rounding\_type=0)
  \textit{Return rounded value.}
• template<typename T >
  T \texttt{hypot} (const T x, const T y)
  \textit{Return }sqrt(x^2 + y^2)\textit{.}
• double \texttt{factorial} (const int n)
  \textit{Return the factorial of n.}
• double \texttt{permutations} (const int k, const int n, const bool with\_order)
  \textit{Return the number of permutations of k objects in a set of n objects.}
• double \texttt{fibonacci} (const int n)
  \textit{Calculate fibonacci number.}
• long \texttt{gcd} (long a, long b)
  \textit{Calculate greatest common divisor.}
• char \texttt{lowercase} (const char x)
  \textit{Convert character to lower case.}
• void \texttt{lowercase} (char *const str)
  \textit{Convert C-string to lower case.}
• char \texttt{uppercase} (const char x)
7.2 cimg_library::cimg Namespace Reference

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

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

- double atof (const char *const str)
  Read value in a C-string.

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

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

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

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

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

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

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

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

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

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

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

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

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

- 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 fdate (const char *const path, unsigned int attr)
  Get last write time of a given file or directory (single-attribute version).

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

- template<typename T>
  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<char>` 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, 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.

Generated by Doxygen
Variables

- const unsigned int keyESC = 1U
  Keycode for the ESC key (architecture-dependent)
- const unsigned int keyF1 = 2U
  Keycode for the F1 key (architecture-dependent)
- const unsigned int keyF2 = 3U
  Keycode for the F2 key (architecture-dependent)
- const unsigned int keyF3 = 4U
  Keycode for the F3 key (architecture-dependent)
- const unsigned int keyF4 = 5U
  Keycode for the F4 key (architecture-dependent)
- const unsigned int keyF5 = 6U
  Keycode for the F5 key (architecture-dependent)
- const unsigned int keyF6 = 7U
  Keycode for the F6 key (architecture-dependent)
- const unsigned int keyF7 = 8U
  Keycode for the F7 key (architecture-dependent)
- const unsigned int keyF8 = 9U
  Keycode for the F8 key (architecture-dependent)
- const unsigned int keyF9 = 10U
  Keycode for the F9 key (architecture-dependent)
- const unsigned int keyF10 = 11U
  Keycode for the F10 key (architecture-dependent)
- const unsigned int keyF11 = 12U
  Keycode for the F11 key (architecture-dependent)
- const unsigned int keyF12 = 13U
  Keycode for the F12 key (architecture-dependent)
- const unsigned int keyPAUSE = 14U
  Keycode for the PAUSE key (architecture-dependent)
- const unsigned int key1 = 15U
  Keycode for the 1 key (architecture-dependent)
- const unsigned int key2 = 16U
  Keycode for the 2 key (architecture-dependent)
- const unsigned int key3 = 17U
  Keycode for the 3 key (architecture-dependent)
- const unsigned int key4 = 18U
  Keycode for the 4 key (architecture-dependent)
- const unsigned int key5 = 19U
  Keycode for the 5 key (architecture-dependent)
- const unsigned int key6 = 20U
  Keycode for the 6 key (architecture-dependent)
- const unsigned int key7 = 21U
  Keycode for the 7 key (architecture-dependent)
- const unsigned int key8 = 22U
  Keycode for the 8 key (architecture-dependent)
- const unsigned int key9 = 23U
  Keycode for the 9 key (architecture-dependent)
- const unsigned int key0 = 24U
  Keycode for the 0 key (architecture-dependent)
- const unsigned int keyBACKSPACE = 25U
Namespace Documentation

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 \texttt{keyK} = 51U  
  \textit{Keycode for the} K \textit{key (architecture-dependent)}
• const unsigned int \texttt{keyL} = 52U  
  \textit{Keycode for the} L \textit{key (architecture-dependent)}
• const unsigned int \texttt{keyENTER} = 53U  
  \textit{Keycode for the} \texttt{ENTER} \textit{key (architecture-dependent)}
• const unsigned int \texttt{keySHIFTLEFT} = 54U  
  \textit{Keycode for the} \texttt{SHIFTLEFT} \textit{key (architecture-dependent)}
• const unsigned int \texttt{keyZ} = 55U  
  \textit{Keycode for the} Z \textit{key (architecture-dependent)}
• const unsigned int \texttt{keyX} = 56U  
  \textit{Keycode for the} X \textit{key (architecture-dependent)}
• const unsigned int \texttt{keyC} = 57U  
  \textit{Keycode for the} C \textit{key (architecture-dependent)}
• const unsigned int \texttt{keyV} = 58U  
  \textit{Keycode for the} V \textit{key (architecture-dependent)}
• const unsigned int \texttt{keyB} = 59U  
  \textit{Keycode for the} B \textit{key (architecture-dependent)}
• const unsigned int \texttt{keyN} = 60U  
  \textit{Keycode for the} N \textit{key (architecture-dependent)}
• const unsigned int \texttt{keyM} = 61U  
  \textit{Keycode for the} M \textit{key (architecture-dependent)}
• const unsigned int \texttt{keySHIFTRIGHT} = 62U  
  \textit{Keycode for the} \texttt{SHIFTRIGHT} \textit{key (architecture-dependent)}
• const unsigned int \texttt{keyARROWUP} = 63U  
  \textit{Keycode for the} \texttt{ARROWUP} \textit{key (architecture-dependent)}
• const unsigned int \texttt{keyCTRLLEFT} = 64U  
  \textit{Keycode for the} \texttt{CTRLLEFT} \textit{key (architecture-dependent)}
• const unsigned int \texttt{keyAPPLEFT} = 65U  
  \textit{Keycode for the} \texttt{APPLEFT} \textit{key (architecture-dependent)}
• const unsigned int \texttt{keyALT} = 66U  
  \textit{Keycode for the} ALT \textit{key (architecture-dependent)}
• const unsigned int \texttt{keySPACE} = 67U  
  \textit{Keycode for the} SPACE \textit{key (architecture-dependent)}
• const unsigned int \texttt{keyALTGR} = 68U  
  \textit{Keycode for the} ALTGR \textit{key (architecture-dependent)}
• const unsigned int \texttt{keyAPPRIGHT} = 69U  
  \textit{Keycode for the} APPRIGHT \textit{key (architecture-dependent)}
• const unsigned int \texttt{keyMENU} = 70U  
  \textit{Keycode for the} MENU \textit{key (architecture-dependent)}
• const unsigned int \texttt{keyCTRLRIGHT} = 71U  
  \textit{Keycode for the} CTRLRIGHT \textit{key (architecture-dependent)}
• const unsigned int \texttt{keyARROWLEFT} = 72U  
  \textit{Keycode for the} ARROWLEFT \textit{key (architecture-dependent)}
• const unsigned int \texttt{keyARROWDOWN} = 73U  
  \textit{Keycode for the} ARROWDOWN \textit{key (architecture-dependent)}
• const unsigned int \texttt{keyARROWRIGHT} = 74U  
  \textit{Keycode for the} ARROWRIGHT \textit{key (architecture-dependent)}
• const unsigned int \texttt{keyPAD0} = 75U  
  \textit{Keycode for the} PAD0 \textit{key (architecture-dependent)}
• const unsigned int \texttt{keyPAD1} = 76U  
  \textit{Keycode for the} PAD1 \textit{key (architecture-dependent)}
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()

```cpp
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]

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

7.2.2.4 exception_mode() [2/2]

```cpp
unsigned int& cimg_library::cimg::exception_mode ()
```
Return current CImg exception mode.

Note

By default, return the value of configuration macro \texttt{cimg_verbosity}

### 7.2.2.5 openmp_mode()

```cpp
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()

```cpp
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

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

7.2.2.7 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

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

instead of

```cpp
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 )

Generated by Doxygen
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 window on Windows-based systems.

7.2.2.9 endianness()

```cpp
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]

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

Reverse endianness of all elements in a memory buffer.

Parameters

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

7.2.2.11 invert_endianness() [2/2]

```cpp
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 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 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 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.
7.2 cimg_library::cimg Namespace Reference

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
unsigned int cimg_library::cimg::wait ( const unsigned int milliseconds )
```

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

Parameters

| milliseconds | Number of milliseconds to wait for. |

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

Note
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

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

- \( \text{minmod}(a,b) = \min(a,b) \), if \( a \) and \( b \) have the same sign.
- \( \text{minmod}(a,b) = 0 \), if \( a \) and \( b \) have different signs.

7.2.2.19 round()

```cpp
T cimg_library::cimg::round ( const T & x, const double y, const int rounding_type = 0 )
```

Return rounded value.

Parameters

<table>
<thead>
<tr>
<th>Parameter</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>rounding_type</td>
<td>Type of rounding operation (0 = nearest, -1 = backward, 1 = forward).</td>
</tr>
</tbody>
</table>

Returns

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

7.2.2.20 atof()

```cpp
double cimg_library::cimg::atof ( const char *const str )
```

Read value in a C-string.
Parameters

\( \text{str} \)  
C-string containing the float value to read.

Returns

Read value.

Note

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

7.2.2.21 \texttt{strncasecmp()}

\[
\text{int cimg\_library::cimg::strncasecmp (const char *const str1, const char *const str2, const int l )}
\]

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

Parameters

\( \text{str1} \)  
C-string.

\( \text{str2} \)  
C-string.

\( \text{l} \)  
Number of characters to compare.

Returns

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

Note

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

7.2.2.22 \texttt{strcasecmp()}

\[
\text{int cimg\_library::cimg::strcasecmp (const char *const str1, const char *const str2 )}
\]

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]

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

Ellipsize a string.

Parameters

<table>
<thead>
<tr>
<th>str</th>
<th>C-string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>Max number of characters.</td>
</tr>
<tr>
<td>is_ending</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]

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

Ellipsize a string.

Parameters

<table>
<thead>
<tr>
<th>str</th>
<th>C-string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>output C-string.</td>
</tr>
<tr>
<td>l</td>
<td>Max number of characters.</td>
</tr>
<tr>
<td>is_ending</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 )
```

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

**Parameters**

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

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

7.2.2.27  strunescape()

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

Replace escape sequences in C-strings by character values.
Parameters

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

### 7.2.2.28 fopen()

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

Open a file.

Parameters

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

Returns

Opened file.

Note

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()

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()

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()

long 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]

int cimg_library::cimg::fdate ( const char *const path,
Get last write time of a given file or directory (multiple-attributes version).
### Parameters

<table>
<thead>
<tr>
<th><code>path</code></th>
<th>Specified path to get attributes from.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in, out attr</code></td>
<td>Type of requested time attributes. Can be { 0=year</td>
</tr>
<tr>
<td><code>nb_attr</code></td>
<td>Number of attributes to read/write.</td>
</tr>
</tbody>
</table>

### Returns

Latest read attribute.

### 7.2.2.34 fdate() [2/2]

```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><code>path</code></th>
<th>Specified path to get attributes from.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>attr</code></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 date() [1/2]

```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 date() [2/2]

```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

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

Returns

Path where temporary files can be saved.

7.2.2.38 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).
7.2 cimg_library::cimg Namespace Reference

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>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 `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>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 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>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 `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>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 `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>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 `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.

7.2.2.44 dcraw_path()

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

Get/set path to the `dcraw` 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 `dcraw` binary.

7.2.2.45 wget_path()

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

Get/set path to the `wget` 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 `wget` binary.

### 7.2.2.46 curl_path()

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

Get/set path to the `curl` binary.

<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 `curl` binary.

### 7.2.2.47 split_filename()

```cpp
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()

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

Read data from file.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ptr</code></td>
<td>Pointer to memory buffer that will contain the binary data read from file.</td>
</tr>
<tr>
<td><code>nmemb</code></td>
<td>Number of elements to read.</td>
</tr>
<tr>
<td><code>stream</code></td>
<td>File to read data from.</td>
</tr>
</tbody>
</table>
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()

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

Write data to file.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ptr</code></td>
<td>Pointer to memory buffer containing the binary data to write on file.</td>
</tr>
<tr>
<td><code>nmemb</code></td>
<td>Number of elements to write.</td>
</tr>
<tr>
<td><code>stream</code></td>
<td>File to write data on.</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()

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

Create an empty file.

Parameters

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

```c
const char *ftype ( 
   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><code>file</code></td>
<td>Input file (can be 0 if filename is set).</td>
</tr>
<tr>
<td><code>filename</code></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, 
   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><code>url</code></td>
<td>URL of the filename, as a C-string.</td>
</tr>
<tr>
<td><code>filename_local</code></td>
<td>C-string containing the path to a local copy of filename.</td>
</tr>
<tr>
<td><code>timeout</code></td>
<td>Maximum time (in seconds) authorized for downloading the file from the URL.</td>
</tr>
<tr>
<td><code>try_fallback</code></td>
<td>When using libcurl, tells using system calls as fallbacks in case of libcurl failure.</td>
</tr>
<tr>
<td><code>referer</code></td>
<td>Referrer 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 cimg_library::cimg Namespace Reference

7.2.2.53 files()

```cpp
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><code>path</code></td>
<td>Path to the directory. Set to 0 for current directory.</td>
</tr>
<tr>
<td><code>is_pattern</code></td>
<td>Tell if specified path has a matching pattern in it.</td>
</tr>
<tr>
<td><code>mode</code></td>
<td>Output type, can be primary ( 0=files only</td>
</tr>
<tr>
<td><code>include_path</code></td>
<td>Tell if <code>path</code> 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><code>title</code></td>
<td>Title of the dialog window.</td>
</tr>
<tr>
<td><code>msg</code></td>
<td>Main message displayed inside the dialog window.</td>
</tr>
<tr>
<td><code>button1_label</code></td>
<td>Label of the 1st button.</td>
</tr>
<tr>
<td><code>button2_label</code></td>
<td>Label of the 2nd button (0 to hide button).</td>
</tr>
<tr>
<td><code>button3_label</code></td>
<td>Label of the 3rd button (0 to hide button).</td>
</tr>
<tr>
<td><code>button4_label</code></td>
<td>Label of the 4th button (0 to hide button).</td>
</tr>
<tr>
<td><code>button5_label</code></td>
<td>Label of the 5th button (0 to hide button).</td>
</tr>
<tr>
<td><code>button6_label</code></td>
<td>Label of the 6th button (0 to hide button).</td>
</tr>
<tr>
<td><code>logo</code></td>
<td>Image logo displayed at the left of the main message.</td>
</tr>
<tr>
<td><code>is_centered</code></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.
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

- \(~\text{CImg}()\)
  
  Destroy image.

- \text{CImg}()
  
  Construct empty image.

- \text{CImg}(\text{const unsigned int size}_x, \text{const unsigned int size}_y=1, \text{const unsigned int size}_z=1, \text{const unsigned int size}_c=1)
  
  Construct image with specified size.

- \text{CImg}(\text{const unsigned int size}_x, \text{const unsigned int size}_y, \text{const unsigned int size}_z, \text{const unsigned int size}_c, \text{const T &value})
  
  Construct image with specified size and initialize pixel values.

- \text{CImg}(\text{const unsigned int size}_x, \text{const unsigned int size}_y, \text{const unsigned int size}_z, \text{const unsigned int size}_c, \text{const int value}_0, \text{const int value}_1,...)
  
  Construct image with specified size and initialize pixel values from a sequence of integers.

- \text{CImg}(\text{const unsigned int size}_x, \text{const unsigned int size}_y, \text{const unsigned int size}_z, \text{const unsigned int size}_c, \text{const double value}_0, \text{const double value}_1,...)
  
  Construct image with specified size and initialize pixel values from a sequence of doubles.

- \text{CImg}(\text{const unsigned int size}_x, \text{const unsigned int size}_y, \text{const unsigned int size}_z, \text{const unsigned int size}_c, \text{const char *const values}, \text{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.

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

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

• template<typename t>
  CImg (const CImg<t> &img, const char *const dimensions)
  
  Construct image with dimensions borrowed from another image.

• template<typename t>
  CImg (const CImg<t> &img, const char *const dimensions, const T &value)
  
  Construct image with dimensions borrowed from another image and initialize pixel values.

• CImg (const CImgDisplay &disp)
  
  Construct image from a display window.

• CImg<T> & assign ()
  
  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].
8.1 CImg<T> Struct Template Reference

- **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 with specified size and initialize pixel values from a memory buffer [overloading].

- **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 with specified size and initialize pixel values from a memory buffer [overloading].

- 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 char* const axes_order)

  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].

- template<typename t>
  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].

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

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

- **CImg<T>** & clear ()

  Construct empty image [in-place version].

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

  Transfer content of an image instance into another one.

- **CImg<T>** & move_to (CImg<T> &img)

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

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

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

- **CImg<T>** & swap (CImg<T> &img)

  Swap fields of two image instances.

- static **CImg<T>** & empty ()

  Return a reference to an empty image.

- 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].
- Climg<T> & operator= (const T &value)
  
  Assign a value to all image pixels.
- Climg<T> & operator= (const char *const expression)
  
  Assign pixels values from a specified expression.
- template<typename t>
  
  Climg<T> & operator= (const Climg<T> &img)
  
  Copy an image into the current image instance.
- Climg<T> & operator= (const Climg<T> &img)
  
  Copy an image into the current image instance [specialization].
- Climg<T> & operator= (const ClimgDisplay &disp)
  
  Copy the content of a display window to the current image instance.
- template<typename t>
  
  Climg<T> & operator++ ()
  
  In-place increment operator (prefix).
- Climg<T> & operator++ (int)
  
  In-place increment operator (postfix).
- Climg<T> & operator+ () const
  
  Return a non-shared copy of the image instance.
- template<typename t>
  
  Climg<typename cimg::superset<T, t>::type> operator+ (const t value) const
  
  Addition operator.
- Climg<T>float & operator+ (const char *const expression) const
  
  Addition operator.
- template<typename t>
  
  Climg<typename cimg::superset<T, t>::type> operator+ (const Climg<T> &img) const
  
  Addition operator.
- template<typename t>
  
  Climg<T> & operator+= (const t value)
  
  In-place addition operator.
- Climg<T> & operator+= (const char *const expression)
  
  In-place addition operator.
- template<typename t>
  
  Climg<typename cimg::superset<T, t>::type> operator+= (const Climg<T> &img)
  
  In-place addition operator.
In-place subtraction operator.

- **Clmgb T > & operator= (const char *const expression)**
  In-place subtraction operator.
- **template<typename t >
  Clmgb T > & operator= (const Clmgb t > &img)**
  In-place subtraction operator.
- **Clmgb T > & operator-- ()**
  In-place decrement operator (prefix).
- **Clmgb T > operator-- (int)**
  In-place decrement operator (postfix).
- **Clmgb T > operator- () const**
  Replace each pixel by its opposite value.
- **template<typename t >
  Clmgb typename cimg::superset < T, t >::type
  operator- (const t value) const**
  Subtraction operator.
- **Clmgb Tfloat > operator- (const char *const expression) const**
  Subtraction operator.
- **template<typename t >
  Clmgb typename cimg::superset < T, t >::type
  operator- (const Clmgb t > &img) const**
  Subtraction operator.
- **template<typename t >
  Clmgb < t > & operator*= (const t value)**
  In-place multiplication operator.
- **Clmgb T > & operator*= (const char *const expression) const**
  In-place multiplication operator.
- **template<typename t >
  Clmgb < t > & operator*= (const Clmgb t > &img)**
  In-place multiplication operator.
- **template<typename t >
  Clmgb typename cimg::superset < T, t >::type
  operator/ (const t value) const**
  Division operator.
- **Clmgb Tfloat > operator/ (const char *const expression) const**
  Division operator.
- **template<typename t >
  Clmgb typename cimg::superset < T, t >::type
  operator/ (const Clmgb t > &img) const**
  Division operator.
- **Clmgb T > & operator/= (const t value)**
  In-place division operator.
- **Clmgb < t > & operator/= (const char *const expression)**
  In-place division operator.
- **template<typename t >
  Clmgb < t > & operator/= (const Clmgb t > &img)**
  In-place division operator.
• template<typename t >
  Clmg<T> & operator %= (const t value)
  In-place modulo operator.
• Clmg<T> & operator %= (const char *const expression)
  In-place modulo operator.
• template<typename t >
  Clmg<T> & operator %= (const Clmg<T> &img)
  In-place modulo operator.
• Clmg<Tfloat> operator% (const char *const expression) const
  Modulo operator.
• Clmg<Tfloat> operator% (const Clmg<Tfloat> &img) const
  Modulo operator.
• template<typename t >
  Clmg<T> & operator &= (const t value)
  In-place bitwise AND operator.
• Clmg<T> & operator &= (const char *const expression)
  In-place bitwise AND operator.
• template<typename t >
  Clmg<T> & operator &= (const Clmg<T> &img)
  In-place bitwise AND operator.
• Clmg<T> & operator & (const char *const expression) const
  Bitwise AND operator.
• Clmg<T> & operator & (const Clmg<T> &img) const
  Bitwise AND operator.
• template<typename t >
  Clmg<T> & operator |=(const t value)
  In-place bitwise OR operator.
• Clmg<T> & operator |=(const char *const expression)
  In-place bitwise OR operator.
• template<typename t >
  Clmg<T> & operator |=(const Clmg<T> &img)
  In-place bitwise OR operator.
• Clmg<T> & operator | (const t value) const
  Bitwise OR operator.
• Clmg<T> & operator | (const char *const expression) const
  Bitwise OR operator.
• template<typename t >
  Clmg<T> & operator ^= (const t value)
  In-place bitwise XOR operator.
• Clmg<T> & operator ^= (const char *const expression)
  In-place bitwise XOR operator.
In-place bitwise XOR operator.

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

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

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

- template<typename t >
  
  CImg< T > & operator<< (const t value) const
  
  Bitwise left shift operator.

- template<typename t >
  
  CImg< T > & operator<< (const char *const expression) const
  
  Bitwise left shift operator.

- template<typename t >
  
  CImg< T > & operator<< (const CImg< t > &img) const
  
  Bitwise left shift operator.

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

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

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

- template<typename t >
  
  CImg< T > & operator>> (const t value) const
  
  Bitwise right shift operator.

- template<typename t >
  
  CImg< T > & operator>> (const char *const expression) const
  
  Bitwise right shift operator.

- template<typename t >
  
  CImg< T > & operator>> (const CImg< t > &img) const
  
  Bitwise right shift operator.

- CImg< T > operator~ () const
  
  Bitwise inversion operator.

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

- bool operator== (const char *const expression) const
  
  Test if all pixel values of an image follow a specified expression.

- template<typename t >
  
  bool operator== (const CImg< t > &img) const
Test if two images have the same size and values.

- template<typename t >
  bool operator!= (const t value) const
  Test if pixels of an image are all different from a value.
  - bool operator!= (const char ∗const expression) const
    Test if all pixel values of an image are different from a specified expression.

- template<typename t >
  bool operator!= (const Clmg< t > &img) const
  Test if two images have different sizes or values.

- template<typename t >
  ClmgList< typename cimg::superset< T, t >::type > operator, (const Clmg< t > &img) const
  Construct an image list from two images.

- template<typename t >
  ClmgList< typename cimg::superset< T, t >::type > operator, (const ClmgList< 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.
8.1 CImg < T > Struct Template Reference

- 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=0, 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=0, 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=0, const int z=0, const int c=0)
  Access to a pixel value, using Neumann boundary conditions for the X,Y and Z-coordinates.
- const T & atXYZ (const int x, const int y=0, const int z=0, const int c=0) const
  Access to a pixel value, using Neumann boundary conditions for the X,Y and Z-coordinates [const version].
- T & atXYZC (const int x, const int y, const int z, const int c, const T & out_value)
  Access to a pixel value, using Dirichlet boundary conditions.
- T atXYZC (const int x, const int y, const int z, const int c, const T & out_value) const
  Access to a pixel value, using Dirichlet boundary conditions [const version].
- T & atXYZC (const int x, const int y=0, const int z=0, const int c=0)
  Access to a pixel value, using Neumann boundary conditions.
- const T & atXYZC (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 [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 Neumann boundary conditions for the X-coordinate.
Return pixel value, using linear interpolation and periodic boundary conditions for the X-coordinate.

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

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`
  - Return pixel value, using cubic interpolation and Dirichlet boundary conditions for the X-coordinate.
- `T cubic_atX_c (const float fx, const int y, const int z, const int c, const T &out_value) const`
  - Return clamped pixel value, using cubic interpolation and Dirichlet boundary conditions for the X-coordinate.
- `T cubic_atX (const float fx, const int y=0, const int z=0, const int c=0) const`
  - Return pixel value, using cubic interpolation and Neumann boundary conditions for the X-coordinate.
- `T cubic_atX_c (const float fx, const int y, const int z, const int c) const`
  - Return clamped pixel value, using cubic interpolation and Neumann boundary conditions for the X-coordinate.
- `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 periodic boundary conditions for the X-coordinate.
- `T cubic_atX_pc (const float fx, const int y, const int z, const int c) const`
- `Tfloat cubic_atXY (const float fx, const float fy, const int z, const int c, const T &out_value) const`
  - Return pixel value, using cubic interpolation and Dirichlet boundary conditions for the X and Y-coordinates.
- `T cubic_atXY_c (const float fx, const float fy, const int z, const int c, const T &out_value) const`
  - Return clamped pixel value, using cubic interpolation and Dirichlet boundary conditions for the X and Y-coordinates.
- `T float cubic_atXY (const float fx, const float fy, const int z=0, const int c=0) const`
  - Return pixel value, using cubic interpolation and Neumann boundary conditions for the X and Y-coordinates.
- `T cubic_atXY_c (const float fx, const float fy, const int z, const int c) const`
  - Return clamped pixel value, using cubic interpolation and Neumann boundary conditions for the X and Y-coordinates.
- `T float 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_atXYZ_pc**(const float fx, const float fy, const float fz, const int c) const
  
  `Set pixel value, using linear interpolation for the X-coordinates.`

• **Clmg<T> & set_linear_atX**(const T &value, const float fx, const int y=0, const int z=0, const int c=0, const bool is_added=false)
  
  `Set pixel value, using linear interpolation for the X and Y-coordinates.`

• **Clmg<T> & set_linear_atXYZ**(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, Y and Z-coordinates.`

• **Clmg<charT> > value_string**(const char separator='\', const unsigned int max_size=0, const char * const format=0) const
  
  `Return a C-string containing a list of all values of the image instance.`

• static const char * pixel_type() const
  
  `Return the type of image pixel values as a C string.`

**Instance Checking**

• bool is_shared() const
  
  `Test shared state of the pixel buffer.`

• bool is_empty() const
  
  `Test if image instance is empty.`

• bool is_inf() const
  
  `Test if image instance contains a 'inf' value.`

• bool is_nan() const
  
  `Test if image instance contains a NaN value.`

• bool is_sameX(const unsigned int size_x) const
  
  `Test if image width is equal to specified value.`

• template<typename t>
  
  `bool is_sameX(const CImg<t> &img) const`

  `Test if image width is equal to specified value.`

• bool is_sameY(const unsigned int size_y) const
  
  `Test if image height is equal to specified value.`

• template<typename t>
  
  `bool is_sameY(const CImg<t> &img) const`

  `Test if image height is equal to specified value.`

• bool is_sameZ(const unsigned int size_z) const
  
  `Test if image depth is equal to specified value.`

• template<typename t>
  
  `bool is_sameZ(const CImg<t> &img) const`

  `Test if image depth is equal to specified value.`

• bool is_sameC(const unsigned int size_c) const
  
  `Test if image spectrum is equal to specified value.`

• template<typename t>
  
  `bool is_sameC(const CImg<t> &img) const`

  `Test if image spectrum is equal to specified value.`

• bool is_sameXY(const unsigned int size_x, const unsigned int size_y) const
Test if image width and height are equal to specified values.

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

Test if image width and height are the same as that of another image.

- bool is_sameXY (const CImgDisplay &disp) const

Test if image width and height are the same as that of an existing display window.

- bool is_sameXZ (const unsigned int size_x, const unsigned int size_z) const

Test if image width and depth are equal to specified values.

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

Test if image width and depth are the same as that of another image.

- bool is_sameXC (const unsigned int size_x, const unsigned int size_c) const

Test if image width and spectrum are equal to specified values.

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

Test if image width and spectrum are the same as that of another image.

- bool is_sameYZ (const unsigned int size_y, const unsigned int size_z) const

Test if image height and depth are equal to specified values.

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

Test if image height and depth are the same as that of another image.

- bool is_sameYC (const unsigned int size_y, const unsigned int size_c) const

Test if image height and spectrum are equal to specified values.

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

Test if image height and spectrum are the same as that of another image.

- bool is_sameZC (const unsigned int size_z, const unsigned int size_c) const

Test if image depth and spectrum are equal to specified values.

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

Test if image depth and spectrum are the same as that of another image.

- bool is_sameXYZ (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z) const

Test if image width, height and depth are equal to specified values.

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

Test if image width, height and depth are the same as that of another image.

- bool is_sameXYC (const unsigned int size_x, const unsigned int size_y, const unsigned int size_c) const

Test if image width, height and spectrum are equal to specified values.

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

Test if image width, height and spectrum are the same as that of another image.

- bool is_sameXZC (const unsigned int size_x, const unsigned int size_z, const unsigned int size_c) const

Test if image width, depth and spectrum are equal to specified values.

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

Test if image width, depth and spectrum are the same as that of another image.

- bool is_sameYZC (const unsigned int size_y, const unsigned int size_z, const unsigned int size_c) const

Test if image height, depth and spectrum are equal to specified values.

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

Test if image height, depth and spectrum are the same as that of another image.
8.1 CImg< T > Struct Template Reference

- `bool is_sameXYZC (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c) const`
  
  Test if image width, height, depth and spectrum are equal to specified values.

- `template<typename t>
  bool is_sameXYZC (const CImg< t >&img) const`
  
  Test if image width, height, depth and spectrum are the same as that of another image.

- `bool containsXYZC (const int x, const int y=0, const int z=0, const int c=0) const`
  
  Test if specified coordinates are inside image bounds.

- `template<typename t>
  bool contains (const T &pixel, t &x, t &y, t &z, t &c) const`
  
  Test if pixel value is inside image bounds and get its X, Y, Z and C-coordinates.

- `template<typename t>
  bool contains (const T &pixel, t &x, t &y, t &z) const`
  
  Test if pixel value is inside image bounds and get its X, Y and Z-coordinates.

- `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<
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

- `CImg< T > & sq()`
  
  Compute the square value of each pixel value.

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

- `CImg< T > & sqrt ()`
  
  Compute the exponential of each pixel value.

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

- `CImg< T > & log ()`
  
  Compute the base-2 logarithm of each pixel value.

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

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

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

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

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

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

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

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

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

• `CImg<Tfloat> get_acosh () const
• `CImg<T> & asinh ()
Compute the hyperbolic arcsine of each pixel value.

• `CImg<Tfloat> get_asinh () const
• `CImg<T> & atanh ()
Compute the hyperbolic arctangent of each pixel value.

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

• `CImg<Tfloat> get_acosh () const
• `CImg<T> & asinh ()
Compute the arcsine of each pixel value.

• `CImg<Tfloat> get_asinh () const
• `CImg<T> & atan ()
Compute the arctangent of each pixel value.

• `CImg<Tfloat> get_atan () const
• `CImg<T> & atan2 (const CImg<T> &img)
Compute the arctangent2 of each pixel value.

• `CImg<Tfloat> get_atan2 (const CImg<T> &img) const
Compute the arctangent2 of each pixel value [new-instance version].

• `CImg<T> & tanh ()
Compute the hyperbolic arctangent of each pixel value.

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

• `CImg<Tfloat> get_acosh () const
• `CImg<T> & asinh ()
Compute the hyperbolic arcsine of each pixel value.

• `CImg<Tfloat> get_asinh () const
• `CImg<T> & atanh ()
Compute the hyperbolic arctangent of each pixel value.
• template<typename t >
  CImg<T> & mul (const CImg<T> &img)
  In-place pointwise multiplication.
• template<typename t >
  CImg< typename cimg::superset<T, t>::type >
  get_mul (const CImg<T> &img) const
  In-place pointwise multiplication [new-instance version].
• template<typename t >
  CImg<T> & div (const CImg<T> &img)
  In-place pointwise division.
• template<typename t >
  CImg< typename cimg::superset<T, t>::type >
  get_div (const CImg<T> &img) const
  In-place pointwise division [new-instance version].
• CImg<T> & pow (const double p)
  Raise each pixel value to a specified power.
• CImg<Tfloat> get_pow (const double p) const
  Raise each pixel value to a specified power [new-instance version].
• CImg<T> & pow (const char *const expression)
  Raise each pixel value to a power, specified from an expression.
• CImg<Tfloat> get_pow (const char *const expression) const
  Raise each pixel value to a power, specified from an expression [new-instance version].
• template<typename t >
  CImg<T> & pow (const CImg<T> &img)
  Raise each pixel value to a power, pointwisely specified from another image.
• template<typename t >
  CImg<Tfloat> get_pow (const CImg<T> &img) const
  Raise each pixel value to a power, pointwisely specified from another image [new-instance version].
• CImg<T> & rol (const unsigned int n=1)
  Compute the bitwise left rotation of each pixel value.
• CImg<T> get_rol (const unsigned int n=1) const
  Compute the bitwise left rotation of each pixel value [new-instance version].
• CImg<T> & rol (const char *const expression)
  Compute the bitwise left rotation of each pixel value.
• CImg<T> get_rol (const char *const expression) const
  Compute the bitwise left rotation of each pixel value [new-instance version].
• template<typename t >
  CImg<T> & rol (const CImg<T> &img)
  Compute the bitwise left rotation of each pixel value.
• template<typename t >
  CImg<T> get_rol (const CImg<T> &img) const
  Compute the bitwise left rotation of each pixel value [new-instance version].
• CImg<T> & ror (const unsigned int n=1)
  Compute the bitwise right rotation of each pixel value.
• CImg<T> get_ror (const unsigned int n=1) const
  Compute the bitwise right rotation of each pixel value [new-instance version].
• CImg<T> & ror (const char *const expression)
  Compute the bitwise right rotation of each pixel value.
• CImg<T> get_ror (const char *const expression) const
  Compute the bitwise right rotation of each pixel value [new-instance version].
• template<typename t >
  CImg<T> & ror (const CImg<T> &img)
  Compute the bitwise right rotation of each pixel value.
• template<typename t >
  CImg<T> get_ror (const CImg<T> &img) const
  Compute the bitwise right rotation of each pixel value [new-instance version].
• template<typename t>
  CImg<T> get_ror (const CImg<t> &img) const
  
  Compute the bitwise right rotation of each pixel value [new-instance version].

• CImg<T> & min (const T &value)
  Pointwise min operator between instance image and a value.

• CImg<T> get_min (const T &value) const
  Pointwise min operator between instance image and a value [new-instance version].

• template<typename t>
  CImg<T> & min (const CImg<t> &img)
  Pointwise min operator between two images.

• template<typename t>
  CImg<typename cimg::superset<T, t>::type> get_min (const CImg<t> &img) const
  Pointwise min operator between two images [new-instance version].

• CImg<T> & max (const T &value)
  Pointwise max operator between instance image and a value.

• CImg<T> get_max (const T &value) const
  Pointwise max operator between instance image and a value [new-instance version].

• template<typename t>
  CImg<T> & max (const CImg<t> &img)
  Pointwise max operator between two images.

• template<typename t>
  CImg<typename cimg::superset<T, t>::type> get_max (const CImg<t> &img) const
  Pointwise max operator between two images [new-instance version].

• CImg<T> & minabs (const T &value)
  Pointwise minabs operator between instance image and a value.

• CImg<T> get_minabs (const T &value) const
  Pointwise minabs operator between instance image and a value [new-instance version].

• template<typename t>
  CImg<T> & minabs (const CImg<t> &img)
  Pointwise minabs operator between two images.

• template<typename t>
  CImg<typename cimg::superset<T, t>::type> get_minabs (const CImg<t> &img) const
  Pointwise minabs operator between two images [new-instance version].

• CImg<T> & maxabs (const T &value)
  Pointwise maxabs operator between instance image and a value.

• CImg<T> get_maxabs (const T &value) const
  Pointwise maxabs operator between instance image and a value [new-instance version].

• template<typename t>
  CImg<T> & maxabs (const CImg<t> &img)
  Pointwise maxabs operator between two images.
8.1 Clmg< T > Struct Template Reference

• template< typename t >
  Clmg< typename cimg::superset< T, t >::type > get_maxabs (const Clmg< t >& img) const
  Pointwise maxabs operator between two images [new-instance version].

• Clmg< T > & maxabs (const char *const expression)
  Pointwise maxabs operator between an image and an expression.

• Clmg< Tfloat > get_maxabs (const char *const expression) const
  Pointwise maxabs operator between an image and an expression [new-instance version].

• T & min ()
  Return a reference to the minimum pixel value.

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

• T & max ()
  Return a reference to the maximum pixel value.

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

• template< typename t >
  T & min_max (t &max_val)
  Return a reference to the minimum pixel value as well as the maximum pixel value.

• template< typename t >
  const T & min_max (t &max_val) const
  Return a reference to the minimum pixel value as well as the maximum pixel value [const version].

• template< typename t >
  T & max_min (t &min_val)
  Return a reference to the maximum pixel value as well as the minimum pixel value.

• template< typename t >
  const T & max_min (t &min_val) const
  Return a reference to the maximum pixel value as well as the minimum pixel value [const version].

• T kth_smallest (const ulongT k) const
  Return the kth smallest pixel value.

• T median () const
  Return the median pixel value.

• 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 Clmg< t >& img) const
  Compute the MSE (Mean-Squared Error) between two images.

• template< typename t >
  double PSNR (const Clmg< 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 ClmgList< T > *const list_inputs=0, ClmgList< 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 ClImgList<T> *const list_inputs=0, ClImgList<T> *const list_outputs=0) const

  Evaluate math formula [const version].

• template<typename t>
  void **eval** (ClImg<T> &output, const char *const expression, const double x=0, const double y=0, const double z=0, const double c=0, const ClImgList<T> *const list_inputs=0, ClImgList<T> *const list_outputs=0)

  Evaluate math formula.

• template<typename t>
  void **eval** (ClImg<T> &output, const char *const expression, const double x=0, const double y=0, const double z=0, const double c=0, const ClImgList<T> *const list_inputs=0, ClImgList<T> *const list_outputs=0) const

  Evaluate math formula [const version].

• template<typename t>
  ClImg< double T > **eval** (const char *const expression, const ClImg< t > &xyzc, const ClImgList<T> *const list_inputs=0, ClImgList<T> *const list_outputs=0)

  Evaluate math formula on a set of variables.

• template<typename t>
  ClImg< double T > **eval** (const char *const expression, const ClImg< t > &xyzc, const ClImgList<T> *const list_inputs=0, ClImgList<T> *const list_outputs=0) const

  Evaluate math formula on a set of variables [const version].

• ClImg<T> & get_stats (const unsigned int variance_method=1) const

  Compute statistics vector from the pixel values.

• ClImg<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 ClImg< t > &img) const

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

• ClImg<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.

• ClImg<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.

• ClImg<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>
  ClImg<T> > & set_vector_at (const ClImg< 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>
  ClImg<T> > & set_matrix_at (const ClImg< 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 Clmg< T > Struct Template Reference

- template<typename t>
  Clmg< T > & set_tensor_at (const Clmg< t > &ten, const unsigned int x=0, const unsigned int y=0, const unsigned int z=0)
  Set tensor-valued pixel at specified position.

- Clmg< T > & vector ()
  Unroll pixel values along axis y.

- Clmg< T > & get_vector () const
  Unroll pixel values along axis y [new-instance version].

- Clmg< T > & matrix ()
  Resize image to become a scalar square matrix.

- Clmg< T > & get_matrix () const
  Resize image to become a scalar square matrix [new-instance version].

- Clmg< T > & tensor ()
  Resize image to become a symmetric tensor.

- Clmg< T > & get_tensor () const
  Resize image to become a symmetric tensor [new-instance version].

- Clmg< T > & diagonal ()
  Resize image to become a diagonal matrix.

- Clmg< T > & get_diagonal () const
  Resize image to become a diagonal matrix [new-instance version].

- Clmg< T > & identity_matrix ()
  Replace the image by an identity matrix.

- Clmg< T > & get_identity_matrix () const
  Replace the image by an identity matrix [new-instance version].

- Clmg< T > & sequence (const T &a0, const T &a1)
  Fill image with a linear sequence of values.

- Clmg< T > & get_sequence (const T &a0, const T &a1) const
  Fill image with a linear sequence of values [new-instance version].

- Clmg< T > & transpose ()
  Transpose the image, viewed as a matrix.

- Clmg< T > & get_transpose () const
  Transpose the image, viewed as a matrix [new-instance version].

- template<typename t>
  Clmg< T > & cross (const Clmg< t > &img)
  Compute the cross product between two 1x3 images, viewed as 3D vectors.

- template<typename t>
  Clmg< typename cimg::superset< T, t >::type > get_cross (const Clmg< t > &img) const
  Compute the cross product between two 1x3 images, viewed as 3D vectors [new-instance version].

- Clmg< T > & invert (const bool use_LU=true)
  Invert the instance image, viewed as a matrix.

- Clmg< Tfloat > & get_invert (const bool use_LU=true) const
  Invert the instance image, viewed as a matrix [new-instance version].

- Clmg< T > & pseudoinvert ()
  Compute the Moore-Penrose pseudo-inverse of the instance image, viewed as a matrix.

- Clmg< Tfloat > & get_pseudoinvert () const
  Compute the Moore-Penrose pseudo-inverse of the instance image, viewed as a matrix [new-instance version].

- template<typename t>
  Clmg< T > & solve (const Clmg< t > &A)
  Solve a system of linear equations.

- template<typename t>
  Clmg< typename cimg::superset2< T, t, float >::type > get_solve (const Clmg< t > &A) const
Solve a system of linear equations [new-instance version].

- template<typename t>
  
  Clm< T > & solve_tridiagonal (const Clm< t > &A)
  
  Solve a tridiagonal system of linear equations.

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

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

- CImgList< Tfloat >
  
  get_eigen () const
  
  Compute eigenvalues and eigenvectors of the instance image, viewed as a matrix.

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

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

- template<typename t>
  
  CImg< T >
  
  & sort (CImg< t > &permutations, const bool is_increasing=true)
  
  Sort pixel values and get sorting permutations.

- template<typename t>
  
  CImg< T > & get_sort (CImg< t > &permutations, const bool is_increasing=true) const
  
  Sort pixel values and get sorting permutations [new-instance version].

- CImg< T > &
  
  get_sort (const bool is_increasing=true, const char axis=0)
  
  Sort pixel values [new-instance version].

- template<typename t>
  
  const CImg< T >
  
  & SVD (CImg< t > &U, CImg< t > &S, CImg< t > &V, const bool sorting=true, const unsigned int max_iteration=40, const float lambda=0) const
  
  Compute the SVD of the instance image, viewed as a general matrix.

- 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.

- template<typename t, typename tf>
  
  static CImg< T >
  
  dijkstra (const tf &distance, const unsigned int nb_nodes, const unsigned int starting_node, CImg< t >&previous_node)
  
  Compute minimal path in a graph, using the Dijkstra algorithm.

- template<typename t, typename tf>
  
  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)
  
  Compute minimal path in a graph, using the Dijkstra algorithm [new-instance version].

- 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.

- CImg< T > &
  
  dijkstra (const unsigned int starting_node, const unsigned int ending_node=−U)
  
  Return minimal path in a graph, using the Dijkstra algorithm.

- CImg< Tfloat >
  
  get_dijkstra (const unsigned int starting_node, const unsigned int ending_node=−U) const
  
  Return minimal path in a graph, using the Dijkstra algorithm [new-instance version].

- template<typename tf, typename t>
  
  static Clm< T >
  
  dijkstra (const tf &distance, const unsigned int nb_nodes, const unsigned int starting_node, const unsigned int ending_node, Clm< t > &previous_node)
  
  Compute minimal path in a graph, using the Dijkstra algorithm.

- template<typename tf, typename t>
  
  static Clm< T >
  
  dijkstra (const tf &distance, const unsigned int nb_nodes, const unsigned int starting_node, const unsigned int ending_node=−U)
Return minimal path in a graph, using the Dijkstra algorithm.

- static CImg< T > string (const char* str, const bool is_last_zero=true, const bool is_shared=false)
  
  Return an image containing the character codes of specified string.

- static CImg< T > vector (const T &a0)
  
  Return a 1x1 image containing specified value.

- static CImg< T > vector (const T &a0, const T &a1)
  
  Return a 1x2 image containing specified values.

- static CImg< T > vector (const T &a0, const T &a1, const T &a2)
  
  Return a 1x3 image containing specified values.

- static CImg< T > vector (const T &a0, const T &a1, const T &a2, const T &a3)
  
  Return a 1x4 image containing specified values.

- static CImg< T > vector (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4)
  
  Return a 1x5 image containing specified values.

- static CImg< 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.

- static CImg< T > vector (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6)
  
  Return a 1x7 image containing specified values.

- static CImg< T > vector (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7)
  
  Return a 1x8 image containing specified values.

- static CImg< T > vector (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7, const T &a8)
  
  Return a 1x9 image containing specified values.

- static CImg< T > vector (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7, const T &a8, const T &a9)
  
  Return a 1x10 image containing specified values.

- static CImg< T > vector (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7, const T &a8, const T &a9, const T &a10)
  
  Return a 1x11 image containing specified values.

- static CImg< T > vector (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7, const T &a8, const T &a9, const T &a10, const T &a11)
  
  Return a 1x12 image containing specified values.

- static CImg< T > vector (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7, const T &a8, const T &a9, const T &a10, const T &a11, const T &a12, const T &a13)
  
  Return a 1x13 image containing specified values.

- static CImg< T > vector (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7, const T &a8, const T &a9, const T &a10, const T &a11, const T &a12, const T &a13, const T &a14)
  
  Return a 1x14 image containing specified values.

- static CImg< T > vector (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7, const T &a8, const T &a9, const T &a10, const T &a11, const T &a12, const T &a13, const T &a14, const T &a15)
  
  Return a 1x15 image containing specified values.

- static CImg< T > vector (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7, const T &a8, const T &a9, const T &a10, const T &a11, const T &a12, const T &a13, const T &a14)
  
  Return a 1x16 image containing specified values.

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

- static CImg< T > matrix (const T &a0, const T &a1)
  
  Return a 2x2 matrix containing specified coefficients.
• static CImg< T > matrix (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7, const T &a8)
  
  *Return a 3x3 matrix containing specified coefficients.*

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

  *Return a 4x4 matrix containing specified coefficients.*

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

  *Return a 5x5 matrix containing specified coefficients.*

• static CImg< T > tensor (const T &a0)
  
  *Return a 1x1 symmetric matrix containing specified coefficients.*

• static CImg< T > tensor (const T &a0, const T &a1, const T &a2)

  *Return a 2x2 symmetric matrix tensor containing specified coefficients.*

• static CImg< T > tensor (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5)

  *Return a 3x3 symmetric matrix containing specified coefficients.*

• static CImg< T > diagonal (const T &a0)

  *Return a 1x1 diagonal matrix containing specified coefficients.*

• static CImg< T > diagonal (const T &a0, const T &a1)

  *Return a 2x2 diagonal matrix containing specified coefficients.*

• static CImg< T > diagonal (const T &a0, const T &a1, const T &a2)

  *Return a 3x3 diagonal matrix containing specified coefficients.*

• static CImg< T > diagonal (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5)

  *Return a 4x4 diagonal matrix containing specified coefficients.*

• static CImg< T > diagonal (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4)

  *Return a 5x5 diagonal matrix 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 [overloading].*

• 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 [overloading].

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

• `CImg< T > & fill (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6, const T &val7, const T &val8, const T &val9, const T &val10, const T &val11, const T &val12, const T &val13)`
  Fill sequentially all pixel values with specified values [new-instance version].
Fill pixel values along the C-axis at a specified pixel position.

- `CImg< T > & fillC (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 [overloading].

- template<typename t >
  `CImg< T > & discard (const CImg< t > &values, const char axis=0)`
  Discard specified sequence of values in the image buffer, along a specific axis.

- template<typename t >
  `CImg< T > get_discard (const CImg< t > &values, const char axis=0)`
  Discard neighboring duplicates in the image buffer, along the specified axis.

- `CImg< T > get_discard (const char axis=0)`
  Discard neighboring duplicates in the image buffer, along the specified axis [new-instance version].

- `CImg< T > & invert_endianness ()`
  Invert endianness of all pixel values.

- `CImg< T > get_invert_endianness () const`
  Invert endianness of all pixel values [new-instance version].

- `CImg< T > & rand (const T &val_min, const T &val_max)`
  Fill image with random values in specified range.

- `CImg< T > get_rand (const T &val_min, const T &val_max) const`
  Fill image with random values in specified range [new-instance version].

- `CImg< T > & round (const double y=1, const int rounding_type=0)`
  Round pixel values.

- `CImg< T > get_round (const double y=1, const unsigned int rounding_type=0) const`
  Round pixel values [new-instance version].

- `CImg< T > & noise (const double sigma, const unsigned int noise_type=0)`
  Add random noise to pixel values.

- `CImg< T > get_noise (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, const float constant_case_ratio=0)`
  Linearly normalize pixel values.

- `CImg< Tfloat > get_normalize (const T &min_value, const T &max_value, const float ratio_if_constant_image=0) const`
  Linearly normalize pixel values [new-instance version].

- `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, with respect to their L2-norm [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< T >` > `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< T >` > `get_histogram` (const unsigned int `nb_levels`) const  

  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 `nblevels`, const `T` &`val_min`, const `T` &`val_max`) const  

  Equalize histogram of pixel values [new-instance version].

• `CImg< T >` > `get_equalize` (const unsigned int `nblevels`) const  

  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< typename 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.

• `template<typename t>`

  `CImg< T >` > `get_map` (const `CImg< t >` &`colormap`, const unsigned int `boundary_conditions`=0) const  

  Map predefined colormap on the scalar (indexed) image instance [new-instance version].

• `CImg< T >` `& label` (const bool is_high_connectivity=false, const `Tfloat` tolerance=0)  

  Label connected components.

• `CImg< T >` > `get_label` (const bool is_high_connectivity=false, const `Tfloat` tolerance=0) const  

  Label connected components [new-instance version].

• `template<typename t>`

  `CImg< T >` `& label` (const `CImg< t >` &`connectivity_mask`, const `Tfloat` tolerance=0)  

  Label connected components [overloading].

• `template<typename t>`

  `CImg< T >` > `get_label` (const `CImg< t >` &`connectivity_mask`, const `Tfloat` tolerance=0) const  

  Label connected components [new-instance version].

---

**Color Base Management**

• `CImg< T >` `& sRGBtoRGB` ()

  Convert pixel values from sRGB to RGB color spaces.

• `CImg< Tfloat >` `get_sRGBtoRGB` () const  

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

• `CImg< T >` `& RGBtosRGB` ()  

  Generated by Doxygen
Convert pixel values from RGB to sRGB color spaces.

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

- \texttt{CImg< Tfloat > get_RGBtoHSI () const}
  Convert pixel values from RGB to HSI color spaces.

- \texttt{CImg< Tfloat > get_RGBtoHSL () const}
  Convert pixel values from RGB to HSL color spaces.

- \texttt{CImg< Tfloat > get_RGBtoHSV () const}
  Convert pixel values from RGB to HSV color spaces.

- \texttt{CImg< Tfloat > get_RGBtoYCbCr () const}
  Convert pixel values from RGB to YCbCr color spaces.

- \texttt{CImg< Tfloat > get_RGBtoCMY () const}
  Convert pixel values from RGB to CMY color spaces.

Convert pixel values from RGB to HSI color spaces.

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

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

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

Convert pixel values from HSI to RGB color spaces.

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

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

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

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

Convert pixel values from HSI to HSV color spaces.

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

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

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

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

Convert pixel values from HSI to YCbCr color spaces.

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

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

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

Convert pixel values from HSI to YUV color spaces.

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

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

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

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

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

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

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

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

- \texttt{CImg< T > & CMYtoRGB ()}
  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<T> & CMYKtoCMY ()
  Convert pixel values from CMYK to CMY color spaces.

• CImg<Tfloat> & get_CMYKtoCMY () const
  Convert pixel values from CMYK to CMY 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<Tuchar> & 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].

• CImg<T> & xyYtoXYZ ()
  Convert pixel values from xyY to XYZ color spaces.

• CImg<Tfloat> & get_xyYtoXYZ () const
  Convert pixel values from xyY to XYZ color spaces [new-instance version].

• CImg<T> & RGBtoCMYK ()
  Convert pixel values from RGB to CMYK color spaces.

• CImg<Tfloat> & get_RGBtoCMYK () const
  Convert pixel values from RGB to CMYK color spaces [new-instance version].
Convert pixel values from RGB to CMYK color spaces [new-instance version].

- \texttt{CImg\textless T\textgreater \& CMYKtoRGB ()}
  Convert pixel values from CMYK to RGB color spaces.

- \texttt{CImg\textless Tuchar\textgreater \& get\_CMYKtoRGB () const}
  Convert pixel values from CMYK to RGB color spaces [new-instance version].

- static const \texttt{CImg\textless Tuchar\textgreater \& default\_LUT256 ()}
  Return colormap "default", containing 256 colors entries in RGB.

- static const \texttt{CImg\textless Tuchar\textgreater \& HSV\_LUT256 ()}
  Return colormap "HSV", containing 256 colors entries in RGB.

- static const \texttt{CImg\textless Tuchar\textgreater \& lines\_LUT256 ()}
  Return colormap "lines", containing 256 colors entries in RGB.

- static const \texttt{CImg\textless Tuchar\textgreater \& hot\_LUT256 ()}
  Return colormap "hot", containing 256 colors entries in RGB.

- static const \texttt{CImg\textless Tuchar\textgreater \& cool\_LUT256 ()}
  Return colormap "cool", containing 256 colors entries in RGB.

- static const \texttt{CImg\textless Tuchar\textgreater \& jet\_LUT256 ()}
  Return colormap "jet", containing 256 colors entries in RGB.

- static const \texttt{CImg\textless Tuchar\textgreater \& flag\_LUT256 ()}
  Return colormap "flag", containing 256 colors entries in RGB.

- static const \texttt{CImg\textless Tuchar\textgreater \& cube\_LUT256 ()}
  Return colormap "cube", containing 256 colors entries in RGB.

Geometric / Spatial Manipulation

- \texttt{CImg\textless T\textgreater \& 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.

- \texttt{CImg\textless T\textgreater \& 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 >
  \texttt{CImg\textless T\textgreater \& resize (const CImg\textless t\textgreater &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 >
  \texttt{CImg\textless T\textgreater \& get\_resize (const CImg\textless t\textgreater &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].

- \texttt{CImg\textless T\textgreater \& 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.

- \texttt{CImg\textless T\textgreater \& 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].

- \texttt{CImg\textless T\textgreater \& resize\_halfXY ()}
  Resize image to half-size along XY axes, using an optimized filter.
• CImg< T > get_resize_halfXY () const
  Resize image to half-size along XY axes, using an optimized filter [new-instance version].

• CImg< T > & resize_doubleXY ()
  Resize image to double-size, using the Scale2X algorithm.

• CImg< T > get_resize_doubleXY () const
  Resize image to double-size, using the Scale2X algorithm [new-instance version].

• CImg< T > & resize_tripleXY ()
  Resize image to triple-size, using the Scale3X algorithm.

• CImg< T > get_resize_tripleXY () const
  Resize image to triple-size, using the Scale3X algorithm [new-instance version].

• CImg< T > & mirror (const char axis)
  Mirror image content along specified axis.

• CImg< T > get_mirror (const char axis) const
  Mirror image content along specified axis [new-instance version].

• CImg< T > & mirror (const char *const axes)
  Mirror image content along specified axes.

• CImg< T > get_mirror (const char *const axes) const
  Mirror image content along specified axes [new-instance version].

• CImg< T > & shift (const int delta_x, const int delta_y=0, const int delta_z=0, const int delta_c=0, const unsigned int boundary_conditions=0)
  Shift image content.

• CImg< T > & get_shift (const int delta_x, const int delta_y=0, const int delta_z=0, const int delta_c=0, const unsigned int boundary_conditions=0) const
  Shift image content [new-instance version].

• CImg< T > & permute_axes (const char *const axes_order)
  Permute axes order.

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

• CImg< T > & unroll (const char axis)
  Unroll pixel values along specified axis.

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

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

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

• CImg< T > & rotate (const float angle, const float cx, const float cy, const unsigned int interpolation, const unsigned int boundary_conditions=0)
  Rotate image with arbitrary angle, around a center point.

• CImg< T > & get_rotate (const float angle, const float cx, const float cy, const unsigned int interpolation, const unsigned int boundary_conditions=0) const
  Rotate image with arbitrary angle, around a center point [new-instance version].

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

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

• CImg< T > rotate (const float u, const float v, const float w, const float angle, const unsigned int interpolation=1, const unsigned int boundary_conditions=0)
Rotate volumetric image with arbitrary angle and axis, around a center point.

- \texttt{CImg<T> get_rotate} (const float \texttt{u}, const float \texttt{v}, const float \texttt{w}, const float \texttt{angle}, const float \texttt{cx}, const float \texttt{cy}, const float \texttt{cz}, const unsigned \texttt{int interpolation}=1, const unsigned \texttt{int boundary_conditions}=0) const

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

- template<typename \texttt{t}>
  \texttt{CImg<T> \& warp} (const \texttt{CImg<t> \& p_warp}, const unsigned \texttt{int mode}=0, const unsigned \texttt{int interpolation}=1, const unsigned \texttt{int boundary_conditions}=0) const

  Warp image content by a warping field.

- template<typename \texttt{t}>
  \texttt{CImg<T> \& get_warp} (const \texttt{CImg<t> \& p_warp}, const unsigned \texttt{int mode}=0, const unsigned \texttt{int interpolation}=1, const unsigned \texttt{int boundary_conditions}=0) const

  Warp image content by a warping field \textit{[new-instance version]}

- \texttt{CImg<T> \& get_projections2d} (const unsigned \texttt{int x0}, const unsigned \texttt{int y0}, const unsigned \texttt{int z0}) const

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

- \texttt{CImg<T> \& projections2d} (const unsigned \texttt{int x0}, const unsigned \texttt{int y0}, const unsigned \texttt{int z0})

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

- \texttt{CImg<T> \& get_crop} (const int \texttt{x0}, const int \texttt{y0}, const int \texttt{z0}, const int \texttt{x1}, const int \texttt{y1}, const int \texttt{z1}, const int \texttt{c0}, const int \texttt{c1}, const unsigned \texttt{int boundary_conditions}=0) const

  Crop image region.

- \texttt{CImg<T> \& get_crop} (const int \texttt{x0}, const int \texttt{y0}, const int \texttt{z0}, const int \texttt{c0}, const int \texttt{x1}, const int \texttt{y1}, const int \texttt{z1}, const int \texttt{c1}, const unsigned \texttt{int boundary_conditions}=0) const

  Crop image region \textit{[new-instance version]}.

- \texttt{CImg<T> \& crop} (const int \texttt{x0}, const int \texttt{y0}, const int \texttt{z0}, const int \texttt{x1}, const int \texttt{y1}, const int \texttt{z1}, const int \texttt{c0}, const int \texttt{c1}, const unsigned \texttt{int boundary_conditions}=0) const

  Crop image region \textit{[overloading]}.

- \texttt{CImg<T> \& crop} (const int \texttt{x0}, const int \texttt{y0}, const int \texttt{x1}, const int \texttt{y1}, const unsigned \texttt{int boundary_conditions}=0) const

  Crop image region \textit{[new-instance version]}.

- \texttt{CImg<T> \& get_crop} (const int \texttt{x0}, const int \texttt{y0}, const int \texttt{z0}, const int \texttt{x1}, const int \texttt{y1}, const int \texttt{z1}, const unsigned \texttt{int boundary_conditions}=0) const

  Crop image region \textit{[overloading]}.

- \texttt{CImg<T> \& get_crop} (const int \texttt{x0}, const int \texttt{y0}, const int \texttt{x1}, const int \texttt{y1}, const unsigned \texttt{int boundary_conditions}=0) const

  Crop image region \textit{[new-instance version]}.

- \texttt{CImg<T> \& autocrop} (const \texttt{T \&value}, const char* \texttt{const axes}="czyx")

  Autocrop region, regarding the specified background value.

- \texttt{CImg<T> \& get_autocrop} (const \texttt{T \&value}, const char* \texttt{const axes}="czyx")

  Autocrop region, regarding the specified background value \textit{[new-instance version]}.

- \texttt{CImg<T> \& get_autocrop} (const \texttt{T \&value}, \texttt{const char* \texttt{const color}}=0, \texttt{const char* \texttt{const axes}}="zyx")

  Autocrop region, regarding the specified background color.

- \texttt{CImg<T> \& get_autocrop} (const \texttt{T \&value}, \texttt{const char* \texttt{const color}}=0, \texttt{const char* \texttt{const axes}}="zyx")

  Autocrop region, regarding the specified background color \textit{[new-instance version]}.

- \texttt{CImg<T> \& get_column} (const int \texttt{x0})

  Return specified image column.

- \texttt{CImg<T> \& \& column} (const int \texttt{x0})

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

• `CImg<T>` `get_row` (const int y0) const
  
  Return specified image row.

• `CImg<T>` & `row` (const int y0)
  
  Return specified image row [in-place version].

• `CImg<T>` `get_rows` (const int y0, const int y1) const
  
  Return specified range of image rows.

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

• `CImg<T>` `get_slice` (const int z0) const
  
  Return specified image slice.

• `CImg<T>` & `slice` (const int z0)
  
  Return specified image slice [in-place version].

• `CImg<T>` `get_slices` (const int z0, const int z1) const
  
  Return specified range of image slices.

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

• `CImg<T>` `get_channel` (const int c0) const
  
  Return specified image channel.

• `CImg<T>` & `channel` (const int c0)
  
  Return specified image channel [in-place version].

• `CImg<T>` `get_channels` (const int c0, const int c1) const
  
  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<floatT>` `get_streamline` (const float x, const float y, const float z, const float L=256, const float dl=0.1f, const unsigned int interpolation_type=2, const bool is_backward_tracking=false, const bool is←oriented_only=false) const
  
  Return streamline 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)
  
  Return a shared-memory image referencing a range of pixels of the image instance.

• `const CImg<T>` `get_shared_points` (const unsigned int x0, const unsigned int x1, const unsigned int y0=0, const unsigned int z0=0, const unsigned int c0=0) const
  
  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)
  
  Return a shared-memory image referencing a range of rows of the image instance.

• `const 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)
  
  Return a shared-memory image referencing one row of the image instance.

• `const 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)
  
  Return a shared memory image referencing a range of slices of the image instance.

• `const 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].
• \texttt{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 \texttt{CImg< T > get\_shared\_slice} (const unsigned int z0, const unsigned int c0=0) const
  
  Return a shared-memory image referencing one slice of the image instance \textit{[const version]}.
• \texttt{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 \texttt{CImg< T > get\_shared\_channels} (const unsigned int c0, const unsigned int c1) const
  
  Return a shared-memory image referencing a range of channels of the image instance \textit{[const version]}.
• \texttt{CImg< T > get\_shared\_channel} (const unsigned int c0)
  
  Return a shared-memory image referencing one channel of the image instance.
• const \texttt{CImg< T > get\_shared\_channel} (const unsigned int c0) const
  
  Return a shared-memory image referencing one channel of the image instance \textit{[const version]}.
• \texttt{CImg< T > get\_shared\_}}
  
  Return a shared-memory version of the image instance.
• const \texttt{CImg< T > get\_shared\_}} (const unsigned int c0)
  
  Return a shared-memory version of the image instance \textit{[const version]}.
• \texttt{CImgList< T > get\_split} (const char axis, const int nb=-1) const
  
  Split image into a list along specified axis.
• template<typename t >
  
  \texttt{CImgList< T > get\_split} (const \texttt{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.
• template<typename t >
  
  \texttt{CImg< T > & append} (const \texttt{CImg< t >} &img, const char axis='x', const float align=0)
  
  Append two images along specified axis.
• \texttt{template<typename t >}
  
  \texttt{CImg< T > & append} (const \texttt{CImg< T >} &img, const char axis='x', const float align=0)
  
  Append two images along specified axis \textit{[specialization]}.
• \texttt{template<typename t >}
  
  \texttt{CImg< typename cimg::superset< T, t >::type} (const \texttt{CImg< T >} &img, const char axis='x', const float align=0) const
  
  Append two images along specified axis \textit{[const version]}.
• \texttt{CImg< T > & append} (const \texttt{CImg< T >} &img, const char axis='x', const float align=0) const
  
  Append two images along specified axis \textit{[specialization]}.
• template<typename tfunc >
  
  static \texttt{CImg< floatT > streamline} (const tfunc &func, const float x, const float y, const float z, const float L=256, const float d=0.1f, const unsigned int interpolation\_type=2, const bool is\_backward\_tracking=false, const bool is\_oriented\_only=false, const float x0=0, const float y0=0, const float z0=0, const float x1=0, const float y1=0, const float z1=0)
  
  Return stream line of a 3D vector field.
• static \texttt{CImg< floatT > streamline} (const char *const expression, const float x, const float y, const float z, const float L=256, const float d=0.1f, const unsigned int interpolation\_type=2, const bool is\_backward\_tracking=true, const bool is\_oriented\_only=false, const float x0=0, const float y0=0, const float z0=0, const float x1=0, const float y1=0, const float z1=0)
  
  Return stream line of a 3D vector field \textit{[overloading]}.

Filtering / Transforms

• template<typename t >
  
  \texttt{CImg< T > & correlate} (const \texttt{CImg< 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.

- \texttt{template<typename t> \ CImg< t > \ get_correlate(const \ CImg< t > &kernel, const unsigned int boundary_conditions=true, const bool is_normalized=false)}

Convolve image by a kernel.

- \texttt{template<typename t> \ CImg< t > \ & convolve(const \ CImg< t > &kernel, const unsigned int boundary_conditions=true, 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)}

- \texttt{CImg< t > \ & cumulate(const \ char \ axis=0)}

- \texttt{CImg< t > \ & get_cumulate(const \ char \ axis=0)}

- \texttt{CImg< t > \ & erode(const \ CImg< t > &kernel, const bool boundary_conditions=true, const bool \ is_normalized=false)}

Erode image by a structuring element.

- \texttt{template<typename t> \ CImg< t > \ get_erode(const \ CImg< t > &kernel, const bool boundary_conditions=true, const bool \ is_normalized=false)}

- \texttt{CImg< t > \ & erode(const \ unsigned \ int \ sx, const \ unsigned \ int \ sy, const \ unsigned \ int \ sz=1)}

Dilate image by a structuring element.

- \texttt{template<typename t> \ CImg< t > \ & dilate(const \ CImg< t > &kernel, const bool boundary_conditions=true, const bool \ is_normalized=false)}
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< Tfloat > 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].

- `CImg< T > & vanvliet` (const float sigma, const unsigned int order, const char axis='x', const bool boundary_conditions=true)
  Van Vliet recursive Gaussian filter.

- `CImg< Tfloat > get_vanvliet` (const float sigma, const unsigned int order, const char axis='x', const bool boundary_conditions=true) const
  Blur image using Van Vliet recursive Gaussian filter. [new-instance version].

- `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.

- `CImg< Tfloat > 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 [new-instance version].

- `CImg< T > & blur` (const float sigma, const bool boundary_conditions=true, const bool is_gaussian=false)
  Blur image isotropically.

- `CImg< Tfloat > get_blur` (const float sigma, const bool boundary_conditions=true, const bool is_gaussian=false) const
  Blur image isotropically. [new-instance version].

- template<typename t >
  `CImg< T > & blur_anisotropic` (const `CImg< t > &G`, const float amplitude=60, const float dl=0.8f, const float da=30, const float gauss_prec=2, const unsigned int interpolation_type=0, const bool is_fast_approx=1)
  Blur image anisotropically, directed by a field of diffusion tensors.

- template<typename t >
  `CImg< Tfloat > get_blur_anisotropic` (const `CImg< t > &G`, const float amplitude=60, const float dl=0.8f, const float da=30, const float gauss_prec=2, const unsigned int interpolation_type=0, const bool is_fast_approx=true) const
  Blur image anisotropically, directed by a field of diffusion tensors [new-instance version].

- `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 unsign int interpolation_type=0, const bool is_fast_approx=true)
  Blur image anisotropically, in an edge-preserving way.

- `CImg< Tfloat > get_blur_anisotropic` (const float amplitude, const float sharpness=0.7f, const float anisotropy=0.6f, const float alpha=0.6f, const float sigma=1.1f, const float dl=0.8f, const float da=30, const float gauss_prec=2, const unsign int interpolation_type=0, const bool is_fast_approx=true) const
  Blur image anisotropically, in an edge-preserving way.
Blur image anisotropically, in an edge-preserving way [new-instance version].

- template<typename t >
  CImg< T > & blur_bilateral (const CImg< t > &guide, const float sigma_x, const float sigma_y, const float sigma_z, const float sigma_r, const float sampling_x, const float sampling_y, const float sampling_z, const float sampling_r)

  Blur image, with the joint bilateral filter.

- template<typename t >
  CImg< Tfloat > get_blur_bilateral (const CImg< t > &guide, const float sigma_x, const float sigma_y, const float sigma_z, const float sigma_r, const float sampling_x, const float sampling_y, const float sampling_z, const float sampling_r)

  Blur image, with the joint bilateral filter [new-instance version].

- template<typename t >
  CImg< T > & blur_bilateral (const CImg< t > &guide, const float sigma_s, const float sigma_r, const float sampling_s=0, const float sampling_r=0)

  Blur image using the joint bilateral filter.

- template<typename t >
  CImg< Tfloat > get_blur_bilateral (const 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 [new-instance version].

- CImg< T > & boxfilter (const float boxsize, const int order, const char axis='x', const bool boundary_conditions=true, const unsigned int nb_iter=1)

- 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

- CImg< T > & 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.

- CImg< Tfloat > get_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) const

  Blur image with a box filter [new-instance version].

- CImg< T > & 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< T > & blur_patch (const CImg< t > &guide, const float sigma_s, const float sigma_r, const unsigned int patch_size=3, const unsigned int lookup_size=4, const float smoothness=0, const bool is_fast_approx=true)

  Blur image using patch-based space.

- 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< T > & 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].
• \texttt{Clmg< T > get_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) const

  Blur image using patch-based space \textit{[simplification]} \textit{[new-instance version]}.

• \texttt{Clmg< T > & blur_median} (const unsigned int n, const float threshold=0)

  Blur image with the median filter.

• \texttt{Clmg< T > get_blur_median} (const unsigned int n, const float threshold=0) const

  Blur image with the median filter \textit{[new-instance version]}.

• \texttt{Clmg< T > & sharpen} (const float amplitude, const bool sharpen_type=false, const float edge=1, const float alpha=0, const float sigma=0)

  Sharpen image.

• \texttt{Clmg< T > get_sharpen} (const float amplitude, const bool sharpen_type=false, const float edge=1, const float alpha=0, const float sigma=0) const

  Sharpen image \textit{[new-instance version]}.

• \texttt{ClmgList< Tfloat > get_gradient} (const char *const axes=0, const int scheme=0) const

  Return image gradient.

• \texttt{ClmgList< Tfloat > get_hessian} (const char *const axes=0) const

  Return image hessian.

• \texttt{Clmg< T > & laplacian} ()

  Compute image Laplacian.

• \texttt{Clmg< Tfloat > get_laplacian} () const

  Compute image Laplacian \textit{[new-instance version]}.

• \texttt{Clmg< T > & structure_tensors} (const bool is_fwbw_scheme=false)

  Compute the structure tensor field of an image.

• \texttt{Clmg< Tfloat > get_structure_tensors} (const bool is_fwbw_scheme=false) const

  Compute the structure tensor field of an image \textit{[new-instance version]}.

• \texttt{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.

• \texttt{Clmg< Tfloat > get_diffusion_tensors} (const float sharpness=0.7f, const float anisotropy=0.6f, const float alpha=0.6f, const float sigma=1.1f, const bool is_sqrt=false) const

  Compute field of diffusion tensors for edge-preserving smoothing \textit{[new-instance version]}.

• \texttt{Clmg< T > & displacement} (const \texttt{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 \texttt{Clmg< floatT > &guide}=	exttt{Clmg< floatT >::const_empty()})

  Estimate displacement field between two images.

• \texttt{Clmg< floatT > get_displacement} (const \texttt{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 \texttt{Clmg< floatT > &guide}=	exttt{Clmg< floatT >::const_empty()}) const

  Estimate displacement field between two images \textit{[new-instance version]}.

• template<typename t1 , typename t2 >

  \texttt{Clmg< T > & matchpatch} (const \texttt{Clmg< T > &patch_image}, const unsigned int patch_width, const unsigned int patch_height, const unsigned int patch_depth, const unsigned int nb_iterations, const unsigned int nb_randoms, const float patch_penalization, const \texttt{Clmg< t1 > &guide}=	exttt{Clmg< t1 >::const_empty()})

  Compute correspondence map between two images, using a patch-matching algorithm.

• template<typename t1 , typename t2 >

  \texttt{Clmg< intT > get_matchpatch} (const \texttt{Clmg< T > &patch_image}, const unsigned int patch_width, const unsigned int patch_height, const unsigned int patch_depth, const unsigned int nb_iterations, const unsigned int nb_randoms=5, const float patch_penalization=0, const \texttt{Clmg< t1 > &guide}=	exttt{Clmg< t1 >::const_empty()})

  Compute correspondence map between two images, using the patch-match algorithm \textit{[new-instance version]}.
Compute correspondence map between two images, using the patch-match algorithm [overloading].

- template<typename T>
  
  CImg<typename T> get_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=5, const unsigned int nb_randoms=5, const float patch_penalization=0, const CImg<T>& guide=CImg<T>::const_empty()) const

  Compute correspondence map between two images, using the patch-match algorithm [overloading].

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

  Compute Euclidean distance function to a specified value.

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

  Compute distance to a specified value [new-instance version].

- template<typename T>
  
  CImg<typename T> distance(const T& value, const CImg<typename T>& metric_mask)

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

- template<typename T, typename to>
  
  CImg<typename cimg::superset<T, long>::type> get_distance_dijkstra(const T& value, const CImg<typename T>& metric, const bool is_high_connectivity=false, 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, typename to>
  
  CImg<T> distance_dijkstra(const T& value, const CImg<typename 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].

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

  Compute Haar multiscale wavelet transform.

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

  Compute Haar multiscale wavelet transform [new-instance version].

- CImg<T>& haar(const bool invert=false, const unsigned int nb_scales=1)

  Compute Haar multiscale wavelet transform [overloading].
• Clmg<T> float > get_haar (const bool invert=false, const unsigned int nb_scales=1) const
  Compute Haar multiscale wavelet transform [new-instance version].

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

• ClmgList<Tfloat> get_FFT (const bool is_inverse=false) const
  Compute n-D Fast Fourier Transform.

• static void FFT (Clmg<T>& real, Clmg<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 (Clmg<T>& real, Clmg<T>& imag, const bool is_inverse=false, const unsigned int nb_threads=0)
  Compute n-D Fast Fourier Transform.

3D Objects Management

• Clmg<T> & shift_object3d (const float tx, const float ty=0, const float tz=0)
  Shift 3D object's vertices.

• Clmg<T> float > get_shift_object3d (const float tx, const float ty=0, const float tz=0) const
  Shift 3D object's vertices [new-instance version].

• Clmg<T> & shift_object3d ()
  Shift 3D object's vertices, so that it becomes centered.

• Clmg<T> float > get_shift_object3d () const
  Shift 3D object's vertices, so that it becomes centered [new-instance version].

• Clmg<T> float > & resize_object3d (const float sx, const float sy=-100, const float sz=-100)
  Resize 3D object.

• Clmg<T> float > get_resize_object3d (const float sx, const float sy=-100, const float sz=-100) const
  Resize 3D object [new-instance version].

• Clmg<T> float > & resize_object3d ()
  Resize 3D object to unit size.

• Clmg<T> float > get_resize_object3d () const
  Resize 3D object to unit size [new-instance version].

• template<typename tf , typename tp , typename tff>
  Clmg<T> & append_object3d (ClmgList<tf>& primitives, const Clmg<tp>& obj_vertices, const ClmgList<tff>& obj_primitives)
  Merge two 3D objects 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>& coords=Clmg<tx>::const_empty()) const
  Texturize primitives of a 3D object.

• template<typename tf , typename tc , typename te>
  Clmg<floatT> get_elevation3d (ClmgList<tf>& primitives, const float isovalue, const int size_x=-100, const int size_y=-100) const
  Generate a 3D elevation of the image instance.

• template<typename tf , typename tc>
  Clmg<floatT> get_projections3d (ClmgList<tf>& primitives, ClmgList<tc>& colors, const unsigned int x0, const unsigned int y0, const unsigned int z0, const bool normalize_colors=false) const
  Generate the 3D projection planes of the image instance.

• template<typename tf>
  Clmg<floatT> get_isoline3d (ClmgList<tf>& primitives, const float isovalue, const int size_x=100, const int size_y=100) const
  Generate an isoline of the image instance as a 3D object.
• \texttt{template<typename \textit{tp} > \texttt{CImg< floatT > get_isosurface3d (CImgList< \textit{tf} > \&primitives, const float isovalue, const int size_x=-100, const int size_y=-100, const int size_z=-100) const}

\textit{Generate an isosurface of the image instance as a 3D object.}

• \texttt{template<typename \textit{tp} , typename \textit{tc} , typename \textit{to} > \texttt{CImg< T > \& object3dtoCImg3d (const CImgList< \textit{tp} > \&primitives, const CImgList< \textit{tc} > \&colors, const \textit{to} &opacities, const bool full_check=true)}

\textit{Convert 3D object into a CImg3d representation.}

• \texttt{template<typename \textit{tp} , typename \textit{tc} , typename \textit{to} > \texttt{CImg< T > & object3dtoCImg3d (const CImgList< \textit{tp} > \&primitives, const CImgList< \textit{tc} > \&colors, const bool full_check=true)}

\textit{Convert 3D object into a CImg3d representation [overloading].}

• \texttt{template<typename \textit{tp} > \texttt{CImg< T > \& object3dtoCImg3d (const CImgList< \textit{tp} > \&primitives, const bool full_check=true) const}

\textit{Convert 3D object into a CImg3d representation [overloading].}

• \texttt{CImg< T > \& object3dtoCImg3d (const bool full_check=true) const}

\textit{Convert 3D object into a CImg3d representation [overloading].}

• \texttt{template<typename \textit{tp} , typename \textit{tc} , typename \textit{to} > \texttt{CImg< floatT > get_object3dtoCImg3d (const CImgList< \textit{tp} > \&primitives, const bool full_check=true) const}

\textit{Convert 3D object into a CImg3d representation [new-instance version].}

• \texttt{template<typename \textit{tp} , typename \textit{tc} , typename \textit{to} > \texttt{CImg< floatT > get_object3dtoCImg3d (const CImgList< \textit{tp} > \&primitives, const CImgList< \textit{tc} > \&colors, const bool full_check=true) const}

\textit{Convert 3D object into a CImg3d representation [overloading].}

• \texttt{template<typename \textit{tp} > \texttt{CImg< floatT > get_object3dtoCImg3d (const CImgList< \textit{tp} > \&primitives, const bool full_check=true) const}

\textit{Convert 3D object into a CImg3d representation [overloading].}

• \texttt{CImg< floatT > get_object3dtoCImg3d (const bool full_check=true) const}

\textit{Convert 3D object into a CImg3d representation [overloading].}

• \texttt{template<typename \textit{tp} , typename \textit{tc} , typename \textit{to} > \texttt{CImg< floatT > get_isosurface3d (CImgList< \textit{tf} > \&primitives, const float x0, const float y0, const float x1, const float y1, const int size_x=-256, const int size_y=-256)}

\textit{Compute 3D elevation of a function as a 3D object.}

• \texttt{template<typename \textit{tf} , typename \textit{tfunc} > static CImg< floatT > elevation3d (CImgList< \textit{tf} > \&primitives, const \textit{tfunc} &func, const float x0, const float y0, const float x1, const float y1, const int size_x=-256, const int size_y=-256)}

\textit{Compute 3D elevation of a function, as a 3D object [overloading].}

• \texttt{template<typename \textit{tf} , typename \textit{tfunc} > static CImg< floatT > isoline3d (CImgList< \textit{tf} > \&primitives, 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)}

\textit{Compute 0-isolines of a function, as a 3D object.}

• \texttt{template<typename \textit{tf} > static CImg< floatT > isoline3d (CImgList< \textit{tf} > \&primitives, 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)}

\textit{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)
  
  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 >
  static CImg<floatT> sphere3d (CImgList<tf>& primitives, const float radius=50, const unsigned int subdivisions=3)
  
  Generate a 3D sphere.

• 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)

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

  Draw a textured 2D line.

• template<typename tz , typename tc >
  CImg<T>& draw_line (int x0, int y0, const float z0, int x1, int y1, const float z1, const CImg<tc>&texture,
  const int tx0, const int ty0, const int tx1, const int ty1, const float opacity=1, const unsigned int pattern=~0U,
  const bool init_hatch=true)

  Draw a textured 2D line, with perspective correction.

• template<typename tz , typename tc >
  CImg<T>& draw_line (int x0, int y0, const float z0, int x1, int y1, const float z1, const CImg<tc>&texture,
  const int tx0, const int ty0, const int tx1, const int ty1, const float opacity=1, const unsigned int pattern=~0U,
  const bool init_hatch=true)

  Draw a textured 2D line, with perspective correction and z-buffering.

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

  Draw a set of consecutive 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 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.

• template<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 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 [overloading].

• template<typename tc >
  CImg<T>& draw_triangle (const int x0, const int y0, const int x1, const int y1, const int x2, const int y2,
  const tc *const color, const float opacity=1)

  Draw a filled 2D triangle.

• template<typename tc >
  CImg<T>& draw_triangle (const int x0, const int y0, const int x1, const int y1, const int x2, const int y2,
  const tc *const color, const float opacity=1, const unsigned int pattern)

  Draw a outlined 2D triangle.
• template<typename tz, typename tc>
  Clmg<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>
  Clmg<T> & draw_triangle (int x0, int y0, int x1, int y1, int x2, int y2, const tc *const color, const float opacity=1)

  Draw a Gouraud-shaded 2D triangle.

• template<typename tz, typename tc>
  Clmg<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 brightness=1)

  Draw a Gouraud-shaded 2D triangle, with z-buffering [overloading].

• template<typename tc1, typename tc2, typename tc3>
  Clmg<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.

• template<typename tc>
  Clmg<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.

• template<typename tc>
  Clmg<T> & draw_triangle (int x0, int y0, const float z0, int x1, int y1, const float z1, int x2, int y2, const float z2, const 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 2D textured triangle, with perspective correction.

• template<typename tz, typename tc>
  Clmg<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 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 2D textured triangle, with perspective correction and z-buffering.

• template<typename tc, typename tl>
  Clmg<T> & draw_triangle (int x0, int y0, int x1, int y1, int x2, int y2, const 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.

• template<typename tz, typename tc, typename tl>
  Clmg<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 CImg<tc>& texture, 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, with z-buffering.

• template<typename tc>
  Clmg<T> & draw_triangle (int x0, int y0, int x1, int y1, int x2, int y2, const CImg<tc>& texture, int tx0, int ty0, int tx1, int ty1, int tx2, int ty2, float bs0, float bs1, float bs2, const float opacity=1)

  Draw a textured Gouraud-shaded 2D triangle.

• template<typename tc>
  Clmg<T> & draw_triangle (int x0, int y0, int x1, int y1, int x2, int y2, const CImg<tc>& texture, int tx0, int ty0, int tx1, int ty1, int tx2, int ty2, float bs0, float bs1, float bs2, const float brightness=1)

  Draw a textured Gouraud-shaded 2D triangle, with perspective correction [overloading].

• template<typename tz, typename tc>
  Clmg<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 CImg<tc>& texture, int tx0, int ty0, int tx1, int ty1, int tx2, int ty2, float bs0, float bs1, float bs2, const float brightness=1)

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

• template<typename tc , typename tl >
  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 Clmg< tl >&light, int lx0, int ly0, int lx1, int ly1, int lx2, int ly2, const float opacity=1)
  Draw a textured Phong-shaded 2D triangle, with perspective correction.

• template<typename tz , typename tc , typename tl >
  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 CImg< tc >&texture, int tx0, int ty0, int tx1, int ty1, int tx2, int ty2, const Clmg< tl >&light, int lx0, int ly0, int lx1, int ly1, int lx2, int ly2, const float opacity=1)
  Draw a textured Phong-shaded 2D triangle, with perspective correction and z-buffering.

• CImg< T >& draw_rectangle (const int x0, const int y0, const int z0, const int c0, const int x1, const int y1, const int z1, const int c1, const T val, const float opacity=1)
  Draw a filled 4D rectangle.

• template<typename tc >
  CImg< T >& draw_rectangle (const int x0, const int y0, const int z0, const int x1, const int y1, const int z1, const tc* const color, const float opacity=1)
  Draw a filled 3D rectangle.

• template<typename tc >
  CImg< T >& draw_rectangle (const int x0, const int y0, const int x1, const int y1, const tc* const color, const float opacity, const unsigned int pattern)
  Draw a 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, int radius, 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=1)
  Draw a filled 2D ellipse [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, const unsigned int pattern)
  Draw an outlined 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 an outlined 2D ellipse [overloading].

• template<typename tc >
  CImg< T >& draw_ellipse (const int x0, const int y0, const CImg< t >&tensor, const tc* const color, const float opacity, const unsigned int pattern)
  Draw an outlined 2D ellipse [overloading].

• template<typename tc >
  CImg< T >& draw_circle (const int x0, const int y0, int radius, const tc* const color, const float opacity=1)
Draw a filled 2D circle.
• template<typename tc>
  Clmgt & draw_circle (const int x0, const int y0, int radius, const tc <const color, const float opacity, const unsigned int pattern)

Draw an outlined 2D circle.
• template<typename t>
  Clmgt & draw_image (const int x0, const int y0, const int z0, const int c0, const Clmgt <t> &sprite, const float opacity=1)

Draw an image.
• Clmgt <t> & draw_image (const int x0, const int y0, const int z0, const Clmgt <t> &sprite, const float opacity=1)

Draw an image [specialization].
• template<typename t>
  Clmgt <t> & draw_image (const int x0, const int y0, const Clmgt <t> &sprite, const float opacity=1)

Draw an image [overloading].
• template<typename t>
  Clmgt <t> & draw_image (const int x0, const Clmgt <t> &sprite, const float opacity=1)

Draw an image [overloading].
• template<typename t>
  Clmgt <t> & draw_image (const int x0, const CImg <t> &sprite, const float opacity=1)

Draw an image.
• template<typename ti , typename tm>
  Clmgt <ti > & draw_image (const int x0, const int y0, const int z0, const int c0, const Clmgt <ti > &sprite, const Clmgt <tm> &mask, const float opacity=1, const float mask_max_value=1)

Draw a masked image.
• Clmgt <ti > & draw_image (const int x0, const int y0, const int z0, const Clmgt <ti > &sprite, const Clmgt <tm> &mask, const float opacity=1, const float mask_max_value=1)

Draw a masked image [overloading].
• template<typename ti , typename tm>
  Clmgt <ti > & draw_image (const int x0, const CImg <ti > &sprite, const Clmgt <tm> &mask, const float opacity=1, const float mask_max_value=1)

Draw an image [overloading].
• template<typename ti , typename tm>
  Clmgt <ti > & draw_image (const CImg <ti > &sprite, const CImg <tm> &mask, const float opacity=1, const float mask_max_value=1)

Draw an image.
• template<typename tc1 , typename tc2 , typename t>
  Clmgt <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 ClmgtList <t> &font,...)

Draw a text string.
• template<typename tc , typename t>
  Clmgt <t> & draw_text (const int x0, const int y0, const char *const text, const tc *const foreground_color, const int, const float opacity, const ClmgtList <t> &font,...)

Draw a text string [overloading].
• template<typename tc , typename t1 >
  Clmg< T > & draw_text (const int x0, const int y0, const char *const text, const int, const tc *const background_color, const float opacity, const ClmgList< t > &font,...)
  
  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 tc1 *const foreground_color, const tc2 *const background_color, const float opacity=1, const unsigned int font_height=13,...)
  
  Draw a text string [overloading].
• template<typename tc >
  Clmg< T > & draw_text (const int x0, const int y0, const char *const text, const int, const bool is_arrow=true, const unsigned int pattern=round_x=0)
  
  Draw a text string [overloading].
• template<typename tc >
  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 t1 , typename t2 >
  Clmg< T > & draw_quiver (const CImg< t1 > &flow, const t2 *const color, const float color, const float float opacity=1, const unsigned int pattern=round_x=0, const unsigned int sampling=25, const float factor=-20, const bool is_arrow=true, const unsigned int font_height=13,...)
  
  Draw a 2D vector field.
• template<typename t1 , typename t2 >
  Clmg< T > & draw_quiver (const CImg< t1 > &flow, const Clmg< t2 > &color, const float float opacity=1, const unsigned int pattern=round_x=0, const unsigned int sampling=25, const float factor=-20, const bool is_arrow=true, const unsigned int font_height=13,...)
  
  Draw a 2D vector field, using a field of colors.
• template<typename t , typename tc >
  Clmg< T > & draw_axis (const int x, const CImg< t > &values_x, const tc *const color, const float opacity=1, const unsigned int pattern=round_x=0)
  
  Draw a labeled horizontal axis.
• template<typename t , typename tc >
  Clmg< T > & draw_axis (const int x, const Clmg< t > &values_x, const tc *const color, const float opacity=1, const unsigned int pattern=round_x=0, const unsigned int font_height=13, const bool allow_zero=true, const float round_x=0)
  
  Draw a labeled horizontal axis.
• template<typename tx , typename ty , typename tc >
  Clmg< T > & draw_axes (const Clmg< tx > &values_x, const Clmg< ty > &values_y, const tc *const color, const float opacity=1, const unsigned int pattern=round_x=0, const unsigned int pattern_y=0, 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 >
  Clmg< T > & draw_axes (const float x0, const float x1, const float y0, const float y1, const tc *const color, const float opacity=1, const unsigned int pattern=round_x=0, const unsigned int pattern_y=0, const unsigned int font_height=13, const int subdivisionx=-60, const int subdivisiony=-60, const float precisionx=0, const float precisiony=0, const unsigned int pattern_x=0, const unsigned int pattern_y=0, const unsigned int font_height=13)
  
  Draw labeled horizontal and vertical axes [overloading].
• template<typename tx , typename ty , typename tc >
  Clmg< T > & draw_grid (const Clmg< tx > &values_x, const Clmg< ty > &values_y, const tc *const color, const float float opacity=1, const unsigned int pattern=0U, const unsigned int pattern_y=0U)
  
  Draw 2D grid.
• template<typename tc >
  Clmg< 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 float opacity=1, const unsigned int pattern=round_x=0U, const unsigned int pattern_y=0U)
  
  Draw 2D grid [simplification].
• template<typename t , typename tc >
  Clmg< T > & draw_graph (const Clmg< t > &data, const tc ∗const color, const float opacity=1, const unsigned int plot_type=1, const int vertex_type=1, const double ymin=0, const double ymax=0, const unsigned int pattern=~0U)

  Draw 1D graph.

• template<typename tc , typename t >
  Clmg< T > & draw_fill (const int x0, const int y0, const int z0, const tc ∗const color, const float opacity, const Clmg< t > &region, const float tolerance=0, const bool is_high_connectivity=false)

  Draw filled 3D region with the flood fill algorithm.

• template<typename tc >
  Clmg< 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].

• 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 >
  Clmg< T > & draw_mandelbrot (const int x0, const int y0, const int x1, const int y1, 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.

• 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 tc ∗const color, const float opacity=1)

  Draw a 1D gaussian function.

• template<typename t , typename tc >
  Clmg< T > & draw_gaussian (const float xc, const float yc, const CImg< t > &tensor, const tc ∗const color, const float opacity=1)

  Draw a 2D gaussian function.

• template<typename tc >
  Clmg< T > & draw_gaussian (const float xc, const float yc, const float r1, const float r2, const float ru, const float rv, const tc ∗const 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 CImg< t > &tensor, const tc ∗const color, const float opacity=1)

  Draw a 3D gaussian function [overloading].

• template<typename tc >
  Clmg< T > & draw_gaussian (const float xc, const float yc, const float zc, const float sigma, const tc ∗const color, const float opacity=1)

  Draw a 3D gaussian function [overloading].
• template<typename tp, typename tf, typename tc, typename to>
  CImg< T > & draw_object3d (const float x0, const float y0, const float z0, const CImg< tp >&vertices,
  const CImgList< tf >&primitives, const CImgList< tc >&colors, const CImg< to >&opacities, const unsigned int render_type=4,
  const bool is_double_sided=false, const float 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.

• template<typename tp, typename tf, typename tc, typename to, typename tz>
  CImg< T > & draw_object3d (const float x0, const float y0, const float z0, const CImg< tp >&vertices,
  const CImgList< tf >&primitives, const CImgList< tc >&colors, const CImg< to >&opacities, const unsigned int render_type=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, const float specular_shininess, const float g_opacity, CImg< tz >&zbuffer)
  
  Draw a 3D object [simplification].

• template<typename tp, typename tf, typename tc, typename to, typename tz>
  CImg< T > & draw_object3d (const float x0, const float y0, const float z0, const CImg< tp >&vertices,
  const CImgList< tf >&primitives, const CImgList< tc >&colors, const CImg< to >&opacities, const unsigned int render_type=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 to>
  CImg< T > & draw_object3d (const float x0, const float y0, const float z0, const CImg< tp >&vertices,
  const CImgList< tf >&primitives, const CImgList< tc >&colors, const CImg< to >&opacities, const unsigned int render_type=4,
  const bool is_double_sided, 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].

• template<typename tp, typename tf, typename tc, typename to>
  CImg< T > & draw_object3d (const float x0, const float y0, const float z0, const CImg< tp >&vertices,
  const CImgList< tf >&primitives, const CImgList< tc >&colors, const CImg< to >&opacities, const unsigned int render_type=4,
  const bool is_double_sided, 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].

• template<typename tp, typename tf, typename tc, typename to, typename tz>
  CImg< T > & draw_object3d (const float x0, const float y0, const float z0, const CImg< tp >&vertices,
  const CImgList< tf >&primitives, const CImgList< tc >&colors, const CImg< to >&opacities, const unsigned int render_type=4,
  const bool is_double_sided, 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].

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< intT > & get_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) const
  
  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 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].
8.1 CImg<T> Struct Template Reference

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

- \texttt{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.

- \texttt{CImg<T> & load (const char *const filename)}
  
  Load image from a file.

- \texttt{CImg<T> & load_ascii (const char *const filename)}
  
  Load image from an ascii file.[overloading].

- \texttt{CImg<T> & load_dlm (const char *const filename)}
  
  Load image from a DLM file.[overloading].

- \texttt{CImg<T> & load_bmp (const char *const filename)}
  
  Load image from a BMP file.[overloading].

- \texttt{CImg<T> & load_jpeg (const char *const filename)}
  
  Load image from a JPEG file.[overloading].

- \texttt{CImg<T> & load_magick (const char *const filename)}
  
  Load image from a file, using Magick++ library.

- \texttt{CImg<T> & load_png (const char *const filename, unsigned int *const bits_per_pixel=0)}
  
  Load image from a PNG file.[overloading].

- \texttt{CImg<T> & load_pnm (const char *const filename)}
  
  Load image from a PNM file.[overloading].

- \texttt{CImg<T> & load_pfm (const char *const filename)}
  
  Load image from a PFM file.[overloading].

- \texttt{CImg<T> & load_rgb (const char *const filename, const unsigned int dimw, const unsigned int dimh=1)}
  
  Load image from a RGB file.[overloading].

- \texttt{CImg<T> & load_rgba (const char *const filename, const unsigned int dimw, const unsigned int dimh=1)}
  
  Load image from a RGBA file.[overloading].

- \texttt{CImg<T> & load_tiff (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=\sim 0U, const unsigned int step_frame=1, float *const voxel_size=0, CImg<charT> *const description=0)}
  
  Load image from a TIFF file.

- \texttt{CImg<T> & load_minc2 (const char *const filename)}
  
  Load image from a MINC2 file.


Generated by Doxygen
- \texttt{CImg< T > \& load\_analyze\ (const char \*const filename, float \*const voxel\_size=0)}
  Load image from an ANALYZE7.5/NIFTI file.

- \texttt{CImg< T > \& load\_analyze\ (std::FILE \*const file, float \*const voxel\_size=0)}
  Load image from an ANALYZE7.5/NIFTI file [overloading].

- \texttt{CImg< T > \& load\_cimg\ (const char \*const filename, const char axis='z', const float align=0)}
  Load image from a .cimg[z] file.

- \texttt{CImg< T > \& load\_cimg\ (std::FILE \*const file, const char axis='z', const float align=0)}
  Load image from a .cimg[z] file [overloading].

- \texttt{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.

- \texttt{CImg< T > \& load\_cimg\ (std::FILE \*const file, const unsigned int n0, const unsigned int n1, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0, const unsigned int x1, const unsigned int y1, const unsigned int z1, const unsigned int c1, const char axis='z', const float align=0)}
  Load sub-images of a .cimg file [overloading].

- \texttt{CImg< T > \& load\_inr\ (const char \*const filename, float \*const voxel\_size=0)}
  Load image from an INRIMAGE-4 file.

- \texttt{CImg< T > \& load\_inr\ (std::FILE \*const file, float \*const voxel\_size=0)}
  Load image from an INRIMAGE-4 file [overloading].

- \texttt{CImg< T > \& load\_exr\ (const char \*const filename)}
  Load image from an EXR file.

- \texttt{CImg< T > \& load\_pandore\ (const char \*const filename)}
  Load image from a PANDORE-5 file.

- \texttt{CImg< T > \& load\_pandore\ (std::FILE \*const 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 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.

- \texttt{CImg< T > \& load\_raw\ (std::FILE \*const file, const unsigned int size\_x=0, const unsigned int size\_y=1, const unsigned int size\_z=1, const unsigned int size\_c=1, const bool is\_multiplexed=false, const bool invert\_endianness=false, const ulongT offset=0)}
  Load image from a raw binary file [overloading].

- \texttt{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.

- \texttt{CImg< T > \& load\_yuv\ (std::FILE \*const file, const unsigned int size\_x, const unsigned int size\_y=1, const unsigned int chroma\_subsampling=444, const unsigned int first\_frame=0, const unsigned int last\_frame=\~0U, const unsigned int step\_frame=1, const bool yuv2rgb=true, const char axis='z')} 
  Load image sequence from a YUV file [overloading].

- \texttt{template<typename tf, typename tc> CImg< T > \& load\_off\ (CImgList< tf > \&primitives, CImgList< tc > \&colors, const char \*const filename)}
  Load 3D object from a .OFF file.

- \texttt{template<typename tf, typename tc> CImg< T > \& load\_off\ (CImgList< tf > \&primitives, CImgList< tc > \&colors, std::FILE \*const file)}
  Load 3D object from a .OFF file [overloading].

- \texttt{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.
• \texttt{CImg< T > & load_ffmpeg_external} (const char *const filename, const char axis='z', const float align=0)
  Load image sequence using FFMPEG's external tool 'ffmpeg'.
• \texttt{CImg< T > & load_gif_external} (const char *const filename, const char axis='z', const float align=0)
  Load gif file, using Imagemagick or GraphicsMagicks's external tools.
• \texttt{CImg< T > & load_graphicsmagick_external} (const char *const filename)
  Load image using GraphicsMagick's external tool 'gm'.
• \texttt{CImg< T > & load_gzip_external} (const char *const filename)
  Load gzipped image file, using external tool 'gunzip'.
• \texttt{CImg< T > & load_imagemagick_external} (const char *const filename)
  Load image using ImageMagick's external tool 'convert'.
• \texttt{CImg< T > & load_medcon_external} (const char *const filename)
  Load image from a DICOM file, using XMedcon's external tool 'medcon'.
• \texttt{CImg< T > & load_dcraw_external} (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.
• static \texttt{CImg< T > get_load} (const char *const filename)
  Load image from a file [new-instance version].
• static \texttt{CImg< T > get_load_ascii} (const char *const filename)
  Load image from an ascii file [in-place version].
• static \texttt{CImg< T > get_load_ascii} (std::FILE *const file)
  Load image from an ascii file [new-instance version].
• static \texttt{CImg< T > get_load_dlm} (const char *const filename)
  Load image from a DLM file [new-instance version].
• static \texttt{CImg< T > get_load_dlm} (std::FILE *const file)
  Load image from a DLM file [new-instance version].
• static \texttt{CImg< T > get_load_bmp} (const char *const filename)
  Load image from a BMP file [new-instance version].
• static \texttt{CImg< T > get_load_bmp} (std::FILE *const file)
  Load image from a BMP file [new-instance version].
• static \texttt{CImg< T > get_load_jpeg} (const char *const filename)
  Load image from a JPEG file [new-instance version].
• static \texttt{CImg< T > get_load_jpeg} (std::FILE *const file)
  Load image from a JPEG file [new-instance version].
• static \texttt{CImg< T > get_load_magick} (const char *const filename)
  Load image from a file, using Magick++ library [new-instance version].
• static \texttt{CImg< T > get_load_png} (const char *const filename, unsigned int *const bits_per_pixel=0)
  Load image from a PNG file [new-instance version].
• static \texttt{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 \texttt{CImg< T > get_load_pnm} (const char *const filename)
  Load image from a PNM file [new-instance version].
• static \texttt{CImg< T > get_load_pnm} (std::FILE *const file)
  Load image from a PNM file [new-instance version].
• static \texttt{CImg< T > get_load_pfm} (const char *const filename)
  Load image from a PFM file [new-instance version].
• static \texttt{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)
  Load image from a RGB file [new-instance version].
• static CImg< T > get_load_rgb (std::FILE *const file, const unsigned int size_x=0, const unsigned int size_y=0, 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)
  Load image from a RGBA file [new-instance version].
• static CImg< T > get_load_rgba (std::FILE *const file, const unsigned int size_x=0, const unsigned int size_y=0, 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 file, const unsigned int offset=0)
  Load image from an INRIMAGE-4 file [new-instance version].
• static CImg< T > get_load_pandore (const char *const file, const unsigned int dimw, const unsigned int dimh=1)
  Load image from a PANDORE-5 file [new-instance version].
• static CImg< T > get_load_parrec (const char *const file, const unsigned int offset=0)
  Load image from a PANDORE-5 file [new-instance version].
• static CImg< T > get_load_raw (std::FILE *const file, const unsigned int offset=0)
  Load image from a raw binary file [new-instance version].
8.1 CImg<T> Struct Template Reference

Load image from a raw binary file [new-instance version].
- static CImg<T> get_load_yuv (const char *const filename, const unsigned int size_x, const unsigned int size_y=1, const unsigned int chroma_subsampling=444, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool yuv2rgb=true, const char axis='z')

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

Load image sequence from a YUV file [new-instance version].
- template<typename tf, typename tc>
  static 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 char axis='z', const float align=0)

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 image using GraphicsMagick's external tool 'gm' [new-instance version].
- static CImg<T> get_load_gzip_external (const char *const filename)

Load gzipped image file, using external tool 'gunzip' [new-instance version].
- static CImg<T> get_load_imagemagick_external (const char *const filename)

Load image using ImageMagick's external tool 'convert' [new-instance version].
- static CImg<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 CImg<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 CImg<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 capture_step_frames=0, const bool release_camera=true)

Load image from a camera stream, using OpenCV [new-instance version].
- static CImg<T> get_load_other (const char *const filename)

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

Data Output

- const CImg<T> & print (const char *const title=0, const bool display_stats=true) const
  Display information about the image data.
- const CImg<T> & display (CImgDisplay &disp) const
  Display image into a CImgDisplay window.
- const CImg<T> & display (CImgDisplay &disp, const bool display_info, unsigned int *const XYZ=0, const bool exit_on_anykey=false) const
  Display image into a CImgDisplay window, in an interactive way.
- const CImg<T> & display (const char *const title=0, const bool display_info=true, unsigned int *const XYZ=0, const bool exit_on_anykey=false) const
  Display image into a CImgDisplay window, in an interactive way.
Display image into an interactive window.

* template<typename tp , typename tf , typename tc , typename to >
  const Clmg<T> & display_object3d (ClmgDisplay &disp, const Clmg<T> &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 , typename to >
  const Clmg<T> & display_object3d (const char *const title, const Clmg<T> &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 [simplification].

* template<typename tp , typename tf , typename tc >
  const Clmg<T> & display_object3d (ClmgDisplay &disp, const Clmg<T> &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 >
  const Clmg<T> & display_object3d (ClmgDisplay &disp, const Clmg<T> &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 >
  const Clmg<T> & display_object3d (ClmgDisplay &disp, const Clmg<T> &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 >
  const Clmg<T> & display_object3d (const Clmg<T> &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 >
  const Clmg<T> & display_object3d (const char *const title, const Clmg<T> &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].
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].

• const Clmg<T> & display_graph (ClmgDisplay &disp, const unsigned int plot_type=1, const unsigned int
vertex_type=1, const char *const labelx=0, const double xmin=0, const double xmax=0, const char *const
labely=0, const double ymin=0, const double ymax=0, const bool bool_exit_on_anykey=false) const

Display 1D graph in an interactive window.

• const Clmg<T> & display_graph (const char *const title=0, const unsigned int plot_type=1, const unsigned
int vertex_type=1, const char *const labelx=0, const double xmin=0, const double xmax=0, const char *const
labely=0, const double ymin=0, const double ymax=0, const bool bool_exit_on_anykey=false) const

Display 1D graph in an interactive window [overloading].

• const Clmg<T> & save (const char *const filename, const int number=-1, const unsigned int digits=6) const
Save image as a file.

• const Clmg<T> & save_ascii (const char *const filename) const
Save image as an ascii file.

• const Clmg<T> & save_ascii (std::FILE *const file) const
Save image as an Ascii file [overloading].

• const Clmg<T> & save_cpp (const char *const filename) const
Save image as a .cpp source file.

• const Clmg<T> & save_cpp (std::FILE *const file) const
Save image as a .cpp source file [overloading].

• const Clmg<T> & save_dlm (const char *const filename) const
Save image as a DLM file.

• const Clmg<T> & save_dlm (std::FILE *const file) const
Save image as a DLM file [overloading].

• const Clmg<T> & save_bmp (const char *const filename) const
Save image as a BMP file.

• const Clmg<T> & save_bmp (std::FILE *const file) const
Save image as a BMP file [overloading].

• const Clmg<T> & save_jpeg (const char *const filename, const unsigned int quality=100) const
Save image as a JPEG file.

• const Clmg<T> & save_jpeg (std::FILE *const file, const unsigned int quality=100) const
Save image as a JPEG file [overloading].

• const Clmg<T> & save_magick (const char *const filename, const unsigned int bytes_per_pixel=0) const
Save image, using built-in ImageMagick++ library.

• const Clmg<T> & save_png (const char *const filename, const unsigned int bytes_per_pixel=0) const
Save image as a PNG file.

• const Clmg<T> & save_png (std::FILE *const file, const unsigned int bytes_per_pixel=0) const
Save image as a PNG file [overloading].

• const Clmg<T> & save_pnm (const char *const filename, const unsigned int bytes_per_pixel=0) const
Save image as a PNM file.

• const Clmg<T> & save_pnm (std::FILE *const file, const unsigned int bytes_per_pixel=0) const
Save image as a PNM file [overloading].

• const Clmg<T> & save_pnk (const char *const filename) const
Save image as a PNK file.

• const Clmg<T> & save_pnk (std::FILE *const file) const
Save image as a PNK file [overloading].

• const Clmg<T> & save_pfm (const char *const filename) const
Save image as a PFM file.
Save image as a PFM file [overloading].

- const Clmg< T > & save_rgb (const char *const filename) const
  
  Save image as a RGB file.
- const Clmg< T > & save_rgb (std::FILE *const file) const
  
  Save image as a RGB file [overloading].
- const Clmg< T > & save_rgba (const char *const filename) const
  
  Save image as a RGBA file.
- const Clmg< T > & save_rgba (std::FILE *const file) const
  
  Save image as a RGBA file [overloading].

Save image as a TIFF file.

- const Clmg< T > & save_tiff (const char *const filename, const unsigned int compression_type=0, const float *const voxel_size=0, const char *const description=0, const bool use_bigtiff=true) const
  
  Save image as a TIFF file.
- const Clmg< T > & save_minc2 (const char *const filename, const char *const imitate_file=0) const
  
  Save image as a MINC2 file.
- const Clmg< T > & save_analyze (const char *const filename, const float *const voxel_size=0) const
  
  Save image as an ANALYZE7.5 or NIFTI file.
- const Clmg< T > & save_cimg (const char *const filename, const bool is_compressed=false) const
  
  Save image as a .cimg file.
- const Clmg< T > & save_cimg (std::FILE *const file, const bool is_compressed=false) const
  
  Save image as a .cimg file [overloading].
- const Clmg< T > & save_cimg (const char *const filename, const unsigned int n0, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0) const
  
  Save image as a sub-image into an existing .cimg file.
- const Clmg< 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 Clmg< T > & save_inr (const char *const filename, const float *const voxel_size=0) const
  
  Save image as an INRIMAGE-4 file.
- const Clmg< T > & save_inr (std::FILE *const file, const float *const voxel_size=0) const
  
  Save image as an INRIMAGE-4 file [overloading].
- const Clmg< T > & save_exr (const char *const filename) const
  
  Save image as an OpenEXR file.
- const Clmg< T > & save_pandore (const char *const filename, const unsigned int colorspace=0) const
  
  Save image as a Pandore-5 file.
- const Clmg< T > & save_pandore (std::FILE *const file, const unsigned int colorspace=0) const
  
  Save image as a Pandore-5 file [overloading].
- const Clmg< T > & save_raw (const char *const filename, const bool is_multiplexed=false) const
  
  Save image as a raw data file.
- const Clmg< T > & save_raw (std::FILE *const file, const bool is_multiplexed=false) const
  
  Save image as a raw data file [overloading].
- const Clmg< 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 Clmg< 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 Clmg< T > & save_off (const ClmgList< tf > &primitives, const ClmgList< tc > &colors, const char *const filename) const
  
  Save 3D object as an Object File Format (.off) file.
const Clmg<T> & save_off (const ClmgList<tf> &primitives, const ClmgList<tc> &colors, std::FILE *const file) const

Save 3D object as an Object File Format (.off) file [overloading].

const Clmg<T> & save_video (const char *const filename, const unsigned int fps=25, const char *codec=0, const bool keep_open=false) const

Save volumetric image as a video, using the OpenCV library.

const Clmg<T> & save_ffmpeg_external (const char *const filename, const unsigned int fps=25, const char *codec=0, const unsigned int bitrate=2048) const

Save volumetric image as a video, using ffmpeg external binary.

const Clmg<T> & save_gzip_external (const char *const filename) const

Save image using gzip external binary.

const Clmg<T> & save_graphicsmagick_external (const char *const filename, const unsigned int quality=100) const

Save image using GraphicsMagick’s external binary.

const Clmg<T> & save_imagemagick_external (const char *const filename, const unsigned int quality=100) const

Save image using ImageMagick’s external binary.

const Clmg<T> & save_medcon_external (const char *const filename) const

Save image as a Dicom file.

const Clmg<T> & save_other (const char *const filename, const unsigned int quality=100) const

Clmg<ucharT> get_serialize (const bool is_compressed=false) const

Serialize a Clmg<T> instance into a raw Clmg<unsigned char> 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)

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

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

This is the main class of the Clmg Library. It declares and constructs an image, allows access to its pixel values, and is able to perform various image operations.

Image representation

A Clmg image is defined as an instance of the container Clmg<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 ClmgList<T> rather than simple images Clmg<T>.

Thus, the Clmg<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 Clmg<T> are designed to handle this maximum case of (3+1) dimensions.
Concerning the pixel value type \( T \): fully supported template types are the basic C++ types: unsigned char, char, short, unsigned int, int, unsigned long, long, float, double, ....

Typically, fast image display can be done using CImg\(<\text{unsigned char}>\) images, while complex image processing algorithms may be rather coded using CImg\(<\text{float}>\) or CImg\(<\text{double}>\) 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\(<\text{char}>\) img; declares an empty image.
  - CImg\(<\text{unsigned char}>\) img(128,128); declares a 128x128 greyscale image with unsigned char pixel values.
  - CImg\(<\text{double}>\) img(3,3); declares a 3x3 matrix with double coefficients.
  - CImg\(<\text{unsigned char}>\) img(256,256,1,3); declares a 256x256x1x3 (color) image (colors are stored as an image with three channels).
  - CImg\(<\text{double}>\) 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
// Set all pixels to '0', with a CImg iterator.
for (CImg<float>::iterator it = img.begin(), it!=img.end(); ++it) *it = 0;
img.fill(0); // Do the same with a built-in method
```
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

    const CImg<float> img("reference.jpg"); // Load image from file
    float sum = 0;
    // Compute sum of all pixel values, with a CImg iterator.
    for (CImg<float>::iterator it = img.begin(), it<img.end(); ++it) sum+=*it;
    const float sum2 = img.sum(); // Do the same with a built-in method

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 CImg<T>::value_type type of a CImg<T> is defined to be a T.
• CImg<T>::value_type is actually not used in CImg methods. It has been mainly defined for compatibility with STL naming conventions.

8.1.3  Constructor & Destructor Documentation

8.1.3.1  ~CImg()

~CImg ()

Destroy image.

Note

• The pixel buffer data() is deallocated if necessary, e.g. for non-empty and non-shared image instances.
• Destroying an empty or shared image does nothing actually.

Warning

• When destroying a non-shared image, make sure that you will not operate on a remaining shared image that shares its buffer with the destroyed instance, in order to avoid further invalid memory access (to a deallocated buffer).
8.1.3.2 CImg()

Construct empty image.

Note

- An empty image has no pixel data and all of its dimensions width(), height(), depth(), spectrum() are set to 0, as well as its pixel buffer pointer data().
- An empty image may be re-assigned afterwards, e.g. with the family of assign(unsigned int,unsigned int,unsigned int,unsigned int) methods, or by operator=(const CImg<T>&). In all cases, the type of pixels stays T.
- An empty image is never shared.

Example

```cpp
CImg<float> img1, img2; // Construct two empty images
img1.assign(256,256,1,3); // Re-assign 'img1' to be a 256x256x1x3 (color) image
img2 = img1.get_rand(0,255); // Re-assign 'img2' to be a random-valued version of 'img1'
img2.assign(); // Re-assign 'img2' to be an empty image again
```

8.1.3.3 CImg()

Construct image with specified size.

Parameters

| `size_x` | Image width(). |
| `size_y` | Image height(). |
| `size_z` | Image depth(). |
| `size_c` | Image spectrum() (number of channels). |

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

```cpp
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]

```cpp
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]

```cpp
CImg {
    const unsigned int size_x,
```
Construct image with specified size and initialize pixel values from a sequence of integers.

Construct a new image instance of size \( \text{size}_x \times \text{size}_y \times \text{size}_z \times \text{size}_c \), with pixels of type \( T \), and initialize pixel values from the specified sequence of integers \( \text{value}_0, \text{value}_1, \ldots \).

**Parameters**

| \( \text{size}_\text{x} \) | Image width(). |
| \( \text{size}_\text{y} \) | Image height(). |
| \( \text{size}_\text{z} \) | Image depth(). |
| \( \text{size}_\text{c} \) | Image spectrum() (number of channels). |
| \( \text{value}_0 \) | First value of the initialization sequence (must be an \textit{integer}). |
| \( \text{value}_1 \) | Second value of the initialization sequence (must be an \textit{integer}). |
| ... | ... |

**Note**

- Similar to \( \text{CImg}(\text{unsigned int}, \text{unsigned int}, \text{unsigned int}, \text{unsigned int}) \), but it also fills the pixel buffer with a sequence of specified integer values.

**Warning**

- You must specify exactly \( \text{size}_x \times \text{size}_y \times \text{size}_z \times \text{size}_c \) integers in the initialization sequence. Otherwise, the constructor may crash or fill your image pixels with garbage.

**Example**

```cpp
const CImg<float> img(2,2,1,3, // Construct a 2x2 color (RGB) image
0,255,0,255, // Set the 4 values for the red component
0,0,255,255, // Set the 4 values for the green component
64,64,64,64); // Set the 4 values for the blue component
img.resize(150,150).display();
```

**8.1.3.6 CImg**

```cpp
CImg(
    const unsigned int \text{size}_x,
    const unsigned int \text{size}_y,
    const unsigned int \text{size}_z,
    const unsigned int \text{size}_c,
    const double \text{value}_0,
    ...
)
```
Construct image with specified size and initialize pixel values from a sequence of doubles.

Construct a new image instance of size size_x x size_y x size_z x size_c, with pixels of type T, and initialize pixel values from the specified sequence of doubles value0,value1,...
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>size_x</code></td>
<td>Image width().</td>
</tr>
<tr>
<td><code>size_y</code></td>
<td>Image height().</td>
</tr>
<tr>
<td><code>size_z</code></td>
<td>Image depth().</td>
</tr>
<tr>
<td><code>size_c</code></td>
<td>Image spectrum() (number of channels).</td>
</tr>
<tr>
<td><code>value0</code></td>
<td>First value of the initialization sequence (must be a double).</td>
</tr>
<tr>
<td><code>value1</code></td>
<td>Second value of the initialization sequence (must be a double).</td>
</tr>
</tbody>
</table>

Note

- Similar to `CImg(unsigned int,unsigned int,unsigned int,unsigned int,int,int,...)`, but takes a sequence of double values instead of integers.

Warning

- You must specify exactly $dx \times dy \times dz \times dc$ 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()` [6/13]

`CImg`

```cpp
    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><code>size_x</code></td>
<td>Image width().</td>
</tr>
<tr>
<td><code>size_y</code></td>
<td>Image height().</td>
</tr>
<tr>
<td><code>size_z</code></td>
<td>Image depth().</td>
</tr>
<tr>
<td><code>size_c</code></td>
<td>Image spectrum() (number of channels).</td>
</tr>
<tr>
<td><code>values</code></td>
<td>Value string describing the way pixel values are set.</td>
</tr>
<tr>
<td><code>repeat_values</code></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) * 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 x size_y x size_z x size_c`, with pixels of type `T`, and initializes pixel values from the specified `t* memory buffer`.

Parameters

<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]

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() [9/13]

```cpp
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

| `img` | Input image to copy. |
| `is_shared` | Tells about the shared state of the constructed copy. |

Note

- Similar to `CImg(const CImg< t >&)`), except that it allows to decide the shared state of the constructed image, which does not depend anymore on the shared state of the input image `img`:
8.1 CImg T Struct Template Reference

- 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]

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>img</th>
<th>Input image from which dimensions are borrowed.</th>
</tr>
</thead>
<tbody>
<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() [12/13]

CImg (const CImg< t >& img, const char *const dimensions, const T & value)

Construct image with dimensions borrowed from another image and initialize pixel values.

Construct a new image instance with pixels of type T, and size get from the dimensions of an existing CImg< t > instance, and set all pixel values to specified value.
Parameters

<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<typename T>&,const char*)`, but it also fills the pixel buffer with the specified `value`.

8.1.3.14 **CImg()**

```cpp
CImg (const CImgDisplay & disp) [explicit]
```

Construct image from a display window.

Construct a new image instance with pixels of type `T`, as a snapshot of an existing `CImgDisplay` instance.

Parameters

<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).
- The image pixels are read as 8-bits RGB values.

8.1.4 Member Function Documentation

8.1.4.1 **assign()**

```cpp
CImg<typename 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]

\[ \text{CImg<}T\text{>}& \text{assign} ( \]
\[ \text{const unsigned int } size_x, \]
\[ \text{const unsigned int } size_y, \]
\[ \text{const unsigned int } size_z, \]
\[ \text{const unsigned int } size_c, \]
\[ \text{const char } ∗\text{const values,} \]
\[ \text{const bool repeat_values} ) \]

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]

\[ \text{CImg<}T\text{>}& \text{assign} ( \]
\[ \text{const t } ∗\text{const values,} \]
\[ \text{const unsigned int } size_x, \]
\[ \text{const unsigned int } size_y \text{=} 1, \]
\[ \text{const unsigned int } size_z \text{=} 1, \]
\[ \text{const unsigned int } size_c \text{=} 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]

\[ \text{CImg<}T\text{>}& \text{assign} ( \]
\[ \text{const char } ∗\text{const filename } \]

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]

\[ \text{CImg<}T\text{>}& \text{assign} ( \]
\[ \text{const CImg<}t\text{>}& \text{img} ) \]

Construct image copy [in-place version].

In-place version of the constructor CImg(const CImg< t >&).

8.1.4.10 **assign()** [10/13]

\[ \text{CImg<}T\text{>}& \text{assign} ( \]
\[ \text{const CImg<}t\text{>}& \text{img,} \]
\[ \text{const bool is_shared} ) \]

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<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<T>& move_to ( 
    CImg<T> & img )

Transfer content of an image instance into another one.
Transfer the dimensions and the pixel buffer content of an image instance into another one, and replace instance by an empty image. It avoids the copy of the pixel buffer when possible.
Parameters

* `img` - Destination image.

Note

- Pixel types `T` and `t` of source and destination images can be different, though the process is designed to be instantaneous when `T` and `t` are the same.

Example

```cpp
CImg<float> src(256,256,1,3,0); // Construct a 256x256x1x3 (color) image filled with value '0'
src.move_to(dest); // Now, 'src' is empty and 'dest' is the 256x256x1x3 image
```

8.1.4.16 move_to() [2/2]

```cpp
CImgList<t>& move_to ( 
    CImgList<t> & list, 
    const unsigned int pos = ~0U )
```

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

Transfer the dimensions and the pixel buffer content of an image instance into a newly inserted image at position `pos` in specified `CImgList<t>` instance.

Parameters

<table>
<thead>
<tr>
<th>list</th>
<th>Destination list.</th>
</tr>
</thead>
<tbody>
<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.
8.1 CImg< T > Struct Template Reference

Parameters

| img | Image to swap fields with. |

Note

- It can be used to interchange the content of two images in a very fast way. Can be convenient when dealing with algorithms requiring two swapping buffers.

Example

```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()

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 const, whether the image instance is const or not. This is the standard method to get/set pixel values in CImg< T > images.

Parameters

| x   | X-coordinate of the pixel value. |
| y   | Y-coordinate of the pixel value. |
| z   | Z-coordinate of the pixel value. |
| c   | C-coordinate of the pixel value. |
Note

- Range of pixel coordinates start from \((0,0,0,0)\) to \((\text{width}() - 1, \text{height}() - 1, \text{depth}() - 1, \text{spectrum}() - 1)\).

- Due to the particular arrangement of the pixel buffers defined in CImg, you can omit one coordinate if the corresponding dimension is equal to 1. For instance, pixels of a 2D image (\(\text{depth}()\) equal to 1) can be accessed by \(\text{img}(x,y,c)\) instead of \(\text{img}(x,y,0,c)\).

Warning

- There is no boundary checking done in this operator, to make it as fast as possible. You must take care of out-of-bounds access by yourself, if necessary. For debugging purposes, you may want to define macro \('\text{cimg}_\text{verbosity}'\)\(\geq 3\) to enable additional boundary checking operations in this operator. In that case, warning messages will be printed on the error output when accessing out-of-bounds pixels.

Example

```cpp
CImg<float> img(100,100,1,3,0); // Construct a 100x100x1x3 (color) image with pixels set to '0'

const float valR = img(10,10,0,0), // Read red value at coordinates (10,10)
               valG = img(10,10,0,1), // Read green value at coordinates (10,10)
               valB = img(10,10,2), // Read blue value at coordinates (10,10) (Z-coordinate can be omitted)
               avg = (valR + valG + valB)/3; // Compute average pixel value

img(10,10,0) = img(10,10,1) = img(10,10,2) = avg; // Replace the color pixel (10,10) by the average grey value
```

8.1.4.20 operator()()[2/2]

\[T& \; \operatorname{operator}()\]

\(T& \; \operatorname{operator}()\) (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

| \(x\) | X-coordinate of the pixel value. |
| \(y\) | Y-coordinate of the pixel value. |
| \(z\) | Z-coordinate of the pixel value. |
| \(c\) | C-coordinate of the pixel value. |
| \(wh\) | Precomputed offset, must be equal to \(\text{width}() \times \text{height}()\). |
| \(whd\) | Precomputed offset, must be equal to \(\text{width}() \times \text{height}() \times \text{depth}()\). |

Note

- Similar to (but faster than) \(\operatorname{operator}()\). 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
    if (!img2) { // Test succeeds, 'img2' is an empty image
        std::printf("'img1' is not empty, 'img2' is empty.");
    }
}
```

- It also allows to use brackets to access pixel values, without need for a CImg<T>::operator[](), e.g.

```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 \texttt{operator=} [2/4]

\begin{verbatim}
CImg\langle T \rangle & \texttt{operator=} ( \\
    \text{const char} * \text{const} \text{expression} )
\end{verbatim}

Assign pixels values from a specified expression.

Initialize all pixel values from the specified string \texttt{expression}.

**Parameters**

| expression | Value string describing the way pixel values are set. |

**Note**

- String parameter \texttt{expression} may describe different things:
  - If \texttt{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 \texttt{expression} and the image size is not modified.
  - If \texttt{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**

\begin{verbatim}
CImg<float> img1(100,100), img2(img1), img3(img1); // Declare 3 scalar images 100x100 with uninitialized values
img1 = "0,50,100,150,200,250,200,150,100,50"; // Set pixel values of 'img1' from a value sequence
img2 = "10*((x*y)\%25)"; // Set pixel values of 'img2' from a formula
img3 = "reference.jpg"; // Set pixel values of 'img3' from a file (image size is modified)
(img1, img2, img3).display();
\end{verbatim}

### 8.1.4.24 \texttt{operator=} [3/4]

\begin{verbatim}
CImg\langle T \rangle & \texttt{operator=} ( \\
    \text{const CImg<} T \text{> & img})
\end{verbatim}

Copy an image into the current image instance.

Similar to the in-place copy constructor \texttt{assign(const CImg<\texttt{t}>&)}. 

### 8.1.4.25 \texttt{operator=} [4/4]

\begin{verbatim}
CImg\langle T \rangle & \texttt{operator=} ( \\
    \text{const CImgDisplay & disp})
\end{verbatim}

Copy the content of a display window to the current image instance.

Similar to \texttt{assign(const CImgDisplay&)}. 

### 8.1.4.26 \texttt{operator+=} [1/3]

\begin{verbatim}
CImg\langle T \rangle & \texttt{operator+=} ( \\
    \text{const \texttt{t} value})
\end{verbatim}

In-place addition operator.

Add specified \texttt{value} to all pixels of an image instance.
Parameters

<table>
<thead>
<tr>
<th>value</th>
<th>Value to add</th>
</tr>
</thead>
</table>

Note

- Resulting pixel values are casted to fit the pixel type $T$. For instance, adding 0.2 to a `CImg<\text{char}>` is possible but does nothing indeed.

- Overflow values are treated as with standard C++ numeric types. For instance,

  ```
  CImg<\text{unsigned char}> \text{img}(100,100,1,1,255); // Construct a 100x100 image with pixel values '255'
  \text{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<\text{unsigned char}> \text{img1}("reference.jpg"); // Load a 8-bits RGB image (values in [0,255])
CImg<\text{float}> \text{img2}(\text{img1}); // Construct a float-valued copy of 'img1'
\text{img2}+=100; // Add '100' to pixel values -> goes out of [0,255] but no problems with floats
\text{img2}.cut(0,255); // Cut values in [0,255] to fit the 'unsigned char' constraint
\text{img1} = \text{img2}; // Rewrite safe result in 'unsigned char' version 'img1'
const CImg<\text{unsigned char}> \text{img3} = (\text{img1} + 100).cut(0,255); // Do the same in a more simple and elegant way
(\text{img1},\text{img2},\text{img3}).\text{display}();
```

8.1.4.27 `operator+=()`  [2/3]

```
CImg<T>& \text{operator}+= (const char ∗\text{expression})
```

In-place addition operator.

Add values to image pixels, according to the specified string `expression`.

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>Value string describing the way pixel values are added</th>
</tr>
</thead>
</table>

Note

- Similar to `operator=(const char+)`, except that it adds values to the pixels of the current image instance, instead of assigning them.

8.1.4.28 `operator+=()`  [3/3]

```
CImg<T>& \text{operator}+= (const CImg<\text{t}> &\text{img})
```

In-place addition operator.

Add values to image pixels, according to the values of the input image `img`.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>img</code></td>
<td>Input image to add.</td>
</tr>
</tbody>
</table>

Note

- The size of the image instance is never modified.
- It is not mandatory that input image `img` has the same size as the image instance. If less values are available in `img`, then the values are added periodically. For instance, adding one WxH scalar image (`spectrum()` equal to 1) to one WxH color image (`spectrum()` equal to 3) means each color channel will be incremented with the same values at the same locations.

Example

```cpp
CImg<float> img1("reference.jpg"); // Load a RGB color image (img1.spectrum()==3)
// Construct a scalar shading (img2.spectrum()==1).
const CImg<float> img2(img1.width(),img1.height(),1,1,"255*(x/w)^2");
img1+=img2; // Add shading to each channel of 'img1'
img1.cut(0,255); // Prevent [0,255] overflow
(img2,img1).display();
```

8.1.4.29 `operator++()` [1/2]

```cpp
CImg<T>& operator++ ()
```

In-place increment operator (prefix).

Add 1 to all image pixels, and return a reference to the current incremented image instance.

Note

- Writing `++img` is equivalent to `img+=1`.

8.1.4.30 `operator++()` [2/2]

```cpp
CImg<T> operator++ (int)
```

In-place increment operator (postfix).

Add 1 to all image pixels, and return a new copy of the initial (pre-incremented) image instance.

Note

- Use the prefixed version `operator++()` if you don’t need a copy of the initial (pre-incremented) image instance, since a useless image copy may be expensive in terms of memory usage.
8.1.4.31  

\texttt{CImg<T> \ operator+ () const}

Return a non-shared copy of the image instance.

\textbf{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 \texttt{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 \(+\text{img}\) is equivalent to \texttt{CImg<T>\( (\text{img, false})\).}

8.1.4.32  

\texttt{CImg<\texttt{typename cimg::superset<T,t>::type}> \ operator+ ( const t value ) const}

Addition 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 \( T \), if necessary.

8.1.4.33  

\texttt{CImg<Tfloat> \ operator+ ( const char *const expression ) const}

Addition 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 \( T \), if necessary.

8.1.4.34  

\texttt{CImg<\texttt{typename cimg::superset<T,t>::type}> \ operator+ ( const CImg<t> & img ) const}

Addition operator.

Similar to \texttt{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.
8.1.4.35 operator-() [1/3]

CImg<T>& operator-= (  
    const t value  )

In-place subtraction operator.

Similar to operator+=(const t), except that it performs a subtraction instead of an addition.

8.1.4.36 operator-() [2/3]

CImg<T>& operator-= (  
    const char *const expression  )

In-place subtraction operator.

Similar to operator+=(const char*), except that it performs a subtraction instead of an addition.

8.1.4.37 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]

\texttt{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 \texttt{unsigned char} opposite of 1 is 255.

Example

```cpp
const CImg<unsigned char> img1("reference.jpg"), // Load an RGB color image
    img2 = -img1; // Compute its opposite (in 'unsigned char')
(img1, img2).display();
```

8.1.4.41 operator-() [2/4]

\texttt{CImg<typename cimg::superset<T,t>::type> operator-(
    const t value) const}

Subtraction 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.

8.1.4.42 operator-() [3/4]

\texttt{CImg<T float> operator-(
    const char *const expression) const}

Subtraction 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.

8.1.4.43 operator-() [4/4]

\texttt{CImg<typename cimg::superset<T,t>::type> operator-(
    const CImg<T> &img) const}

Subtraction operator.

Similar to \texttt{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 \texttt{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**

| `img` | Second operand of the matrix multiplication. |

**Note**

- It does not compute a pointwise multiplication between two images. For this purpose, use `mul(const CImg<T>&)` instead.
- The size of the image instance can be modified by this operator.

**Example**

```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]

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]

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]

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 operator/() [2/3]

CImg<T>& operator/= (  
    const char *const expression )

In-place division operator.

Similar to operator+=(const char*), except that it performs a division instead of an addition.

8.1.4.52 operator/() [3/3]

CImg<T>& operator/= (  
    const CImg< t > & img )

In-place division operator.

Replace the image instance by the (right) matrix division between the image instance and the specified matrix img.

Parameters

| img | Second operand of the matrix division. |

Generated by Doxygen
Note

- It does not compute a pointwise division between two images. For this purpose, use div(const CImg<typename Cimg::superset<T,t>::type>&) instead.
- It returns the matrix operation A*inverse(img).
- The size of the image instance can be modified by this operator.

8.1.4.53 operator(/) [1/3]

CImg<typename Cimg::superset<T,t>::type> operator/ (const t value) const

Division operator.

Similar to operator/=(const t), except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type T, if necessary.

8.1.4.54 operator(/) [2/3]

CImg<Tfloat> operator/ (const char *const expression) const

Division operator.

Similar to operator/=(const char*), except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type T, if necessary.

8.1.4.55 operator(/) [3/3]

CImg<typename Cimg::superset<T,t>::type> operator/ (const CImg<T> &img) const

Division operator.

Similar to operator/=(const CImg<T> &), except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type T, if necessary.

8.1.4.56 operator%=(1/3)

CImg<T> & operator%=( const t value )

In-place modulo operator.

Similar to operator+=(const t), except that it performs a modulo operation instead of an addition.
8.1.4.57  `operator%=(const char∗)`  

```cpp
CImg<T>& operator%=( const char∗ const expression )
```

In-place modulo operator.

Similar to `operator+=(const char+)`, except that it performs a modulo operation instead of an addition.

8.1.4.58  `operator%=(const CImg<T>&)`  

```cpp
CImg<T>& operator%=( const CImg<T>& img )
```

In-place modulo operator.

Similar to `operator+=(const CImg<T>&)`, except that it performs a modulo operation instead of an addition.

8.1.4.59  `operator%=(const t)`  

```cpp
CImg<typename cimg::superset<T,t>::type> operator% ( const t value ) const
```

Modulo operator.

Similar to `operator%=(const t)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type `T`, if necessary.

8.1.4.60  `operator%=(const char∗)`  

```cpp
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%=(const CImg<T>&)`  

```cpp
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 \texttt{operator \&() \[3/3\]}

\begin{verbatim}
CImg<T> operator&( 
    const CImg<T> & img ) const

Bitwise AND operator.
Similar to \texttt{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 \( T \).
\end{verbatim}

8.1.4.68 \texttt{operator\|=\[1/3\]}

\begin{verbatim}
CImg<T>& operator|=( 
    const t value )

In-place bitwise OR operator.
Similar to \texttt{operator\|=(const t)}, except that it performs a bitwise OR operation instead of an addition.
\end{verbatim}

8.1.4.69 \texttt{operator\|=\[2/3\]}

\begin{verbatim}
CImg<T>& operator|=( 
    const char *const expression )

In-place bitwise OR operator.
Similar to \texttt{operator\|=(const char*)}, except that it performs a bitwise OR operation instead of an addition.
\end{verbatim}

8.1.4.70 \texttt{operator\|=\[3/3\]}

\begin{verbatim}
CImg<T>& operator|=( 
    const CImg<T>& img )

In-place bitwise OR operator.
Similar to \texttt{operator\|=(const CImg<T>&)}, except that it performs a bitwise OR operation instead of an addition.
\end{verbatim}

8.1.4.71 \texttt{operator\|() \[1/3\]}

\begin{verbatim}
CImg<T> operator| ( 
    const t value ) const

Bitwise OR 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 \( T \).
\end{verbatim}
8.1.4.72  \texttt{operator\textasciitilde|}() [2/3]

\texttt{CImg\textlt{\textlt{T}> operator\textasciitilde| (}
\texttt{\textlt{\textlt{\textlt{const char \textasciitilde*const expression}} const}}

Bitwise OR operator.

Similar to \texttt{operator\textasciitilde|=(const char\textasciitilde*)}, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is \textit{T}.

8.1.4.73  \texttt{operator\textasciitilde|}() [3/3]

\texttt{CImg\textlt{\textlt{T}> operator\textasciitilde| (}
\texttt{\textlt{\textlt{\textlt{const CImg\textlt{\textlt{t> & img}} const}}}}

Bitwise OR operator.

Similar to \texttt{operator\textasciitilde|=(const CImg\textlt{\textlt{t> &)\textasciitilde}}, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is \textit{T}.

8.1.4.74  \texttt{operator\textasciitilde^=}() [1/3]

\texttt{CImg\textlt{\textlt{T}> & operator\textasciitilde^= (}
\texttt{\textlt{\textlt{const t value}}}}

In-place bitwise XOR operator.

Similar to \texttt{operator\textasciitilde^+=(const t)}, except that it performs a bitwise XOR operation instead of an addition.

\textbf{Warning}

- It does not compute the power of pixel values. For this purpose, use \texttt{pow(const t)} instead.

8.1.4.75  \texttt{operator\textasciitilde^=}() [2/3]

\texttt{CImg\textlt{\textlt{T}> & operator\textasciitilde^= (}
\texttt{\textlt{\textlt{const char \textasciitilde*const expression}}}}

In-place bitwise XOR operator.

Similar to \texttt{operator\textasciitilde^+=(const char\textasciitilde*)}, except that it performs a bitwise XOR operation instead of an addition.

\textbf{Warning}

- It does not compute the power of pixel values. For this purpose, use \texttt{pow(const char\textasciitilde*)} instead.
8.1.4.76 operator^() [3/3]

\texttt{CImg<\textbackslash T>\& operator^\{ const CImg<\textbackslash T>\& img \}}

In-place bitwise XOR operator.

Similar to \texttt{operator+=(const CImg<\textbackslash T>\&)} , except that it performs a bitwise XOR operation instead of an addition.

**Warning**
- It does \emph{not} compute the power of pixel values. For this purpose, use \texttt{pow(const CImg<\textbackslash T>\&)} instead.

8.1.4.77 operator^() [1/3]

\texttt{CImg<\textbackslash T> operator^\{ const \textbackslash t value \} const}

Bitwise XOR operator.

Similar to \texttt{operator^\{const \textbackslash 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<\textbackslash 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}.

8.1.4.79 operator^() [3/3]

\texttt{CImg<\textbackslash T> operator^\{ const CImg<\textbackslash t>\& img \} const}

Bitwise XOR operator.

Similar to \texttt{operator^\{const CImg<\textbackslash t>\&\}}, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is \texttt{T}.

8.1.4.80 operator<<() [1/3]

\texttt{CImg<\textbackslash T>& operator<<\{ const \textbackslash t value \}}

In-place bitwise left shift operator.

Similar to \texttt{operator+=(const \textbackslash t)}, except that it performs a bitwise left shift instead of an addition.
8.1.4.81  operator<<() [2/3]

CImg<T>& operator<< (  
    const char *const expression )

In-place bitwise left shift operator.

Similar to operator+=(const char*), except that it performs a bitwise left shift instead of an addition.

8.1.4.82  operator<<() [3/3]

CImg<T>& operator<< (  
    const CImg<T>& img )

In-place bitwise left shift operator.

Similar to operator+=(const CImg<T>&), except that it performs a bitwise left shift instead of an addition.

8.1.4.83  operator<<( ) [1/3]

CImg<T> operator<<(  
    const t value ) const

Bitwise left shift operator.

Similar to operator<<(const t), except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is T.

8.1.4.84  operator<<( ) [2/3]

CImg<T> operator<<(  
    const char *const expression ) const

Bitwise left shift operator.

Similar to operator<<(const char*), except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is T.

8.1.4.85  operator<<( ) [3/3]

CImg<T> operator<<(  
    const CImg<T>& img ) const

Bitwise left shift operator.

Similar to operator<<(const CImg<T>&), except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is T.
8.1.4.86 operator\[>\!\!\!=()\] [1/3]

\begin{verbatim}
CImg<T>& operator>>= ( 
    const t value 
)
\end{verbatim}

In-place bitwise right shift operator.

Similar to operator+=(const t), except that it performs a bitwise right shift instead of an addition.

8.1.4.87 operator\[>\!\!\!=()\] [2/3]

\begin{verbatim}
CImg<T>& operator>>= ( 
    const char *const expression
)
\end{verbatim}

In-place bitwise right shift operator.

Similar to operator+=(const char*), except that it performs a bitwise right shift instead of an addition.

8.1.4.88 operator\[>\!\!\!=()\] [3/3]

\begin{verbatim}
CImg<T>& operator>>= ( 
    const CImg< t >& img
)
\end{verbatim}

In-place bitwise right shift operator.

Similar to operator+=(const CImg<t>&), except that it performs a bitwise right shift instead of an addition.

8.1.4.89 operator\[>\!\!\!()\] [1/3]

\begin{verbatim}
CImg<T> operator>> ( 
    const t value ) const
\end{verbatim}

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]

\begin{verbatim}
CImg<T> operator>>( 
    const char *const expression ) const
\end{verbatim}

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 \textbf{operator$\gg$()} [3/3]

\begin{verbatim}
CImg\langle T \rangle \textbf{operator$\gg$}( \\
    \textbf{const CImg}\langle T \rangle & \textbf{img}) \textbf{const}
\end{verbatim}

Bitwise right shift operator.

Similar to \texttt{operator$\gg=$(const CImg\langle T \rangle &)}, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is \texttt{T}.

8.1.4.92 \textbf{operator$\neg$()}

\begin{verbatim}
CImg\langle T \rangle \textbf{operator$\neg$}() \textbf{const}
\end{verbatim}

Bitwise inversion operator.

Similar to \texttt{operator-()}, except that it compute the bitwise inverse instead of the opposite value.

8.1.4.93 \textbf{operator==()} [1/3]

\begin{verbatim}
bool \textbf{operator==}( \\
    \textbf{const t value}) \textbf{const}
\end{verbatim}

Test if all pixels of an image have the same value.

Return \texttt{true} is all pixels of the image instance are equal to the specified \texttt{value}.

Parameters

\begin{tabular}{|l|l|}
\hline
\texttt{value} & Reference value to compare with. \\
\hline
\end{tabular}

8.1.4.94 \textbf{operator==(expression)} [2/3]

\begin{verbatim}
bool \textbf{operator==}( \\
    \textbf{const char *const expression}) \textbf{const}
\end{verbatim}

Test if all pixel values of an image follow a specified expression.

Return \texttt{true} is all pixels of the image instance are equal to the specified \texttt{expression}.

Parameters

\begin{tabular}{|l|l|}
\hline
\texttt{expression} & Value string describing the way pixel values are compared. \\
\hline
\end{tabular}
8.1.4.95  operator==()  [3/3]

bool operator== (  
    const CImg< t > & img ) const 

Test if two images have the same size and values.

Return true if the image instance and the input image img have the same pixel values, even if the dimensions of the two images do not match. It returns false otherwise.

Parameters

    img | Input image to compare with.

Note

- The pixel buffer pointers data() of the two compared images do not have to be the same for operator==() to return true. Only the dimensions and the pixel values matter. Thus, the comparison can be true even for different pixel types T and t.

Example

    const CImg<float> img1(1,3,1,1, 0,1,2); // Construct a 1x3 vector [0;1;2] (with 'float' pixel values)
    const CImg<char> img2(1,3,1,1, 0,1,2); // Construct a 1x3 vector [0;1;2] (with 'char' pixel values)
    if (img1==img2) {  // Test succeeds, image dimensions and values are the same
        std::printf("'img1' and 'img2' have same dimensions and values.\n");
    }

8.1.4.96  operator"!=()  [1/3]

bool operator!= (  
    const t value ) const 

Test if pixels of an image are all different from a value.

Return true is all pixels of the image instance are different than the specified value.

Parameters

    value | Reference value to compare with.

8.1.4.97  operator"!=()  [2/3]

bool operator!= (  
    const char *const expression ) const 

Test if all pixel values of an image are different from a specified expression.

Return true is all pixels of the image instance are different to the specified expression.
Parameters

| expression | Value string describing the way pixel values are compared. |

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 `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
class CImgList

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<typename cimg::superset<T,t>::type** operator, (const CImgList<T> & list) const

Construct an image list from image instance and an input image list.

**Return**

A new list of images (CImgList instance) containing exactly list.size() + 1 elements:

- A copy of the image instance, at position [0].
- A copy of the specified image list list, from positions [1] to [list.size()].

**Parameters**

| list  | Input image list that will be appended to the image instance. |

**Note**

- Similar to `operator,(const CImg<T>&)` const, except that it takes an image list as an argument.

### 8.1.4.101 operator($<$)()

**CImgList<T>** operator< (const char axis) const

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.
### Parameters

| axis | Splitting axis (can be 'x','y','z' or 'c') |

#### Note
- Similar to `get_split(char,int) const`, with default second argument.

#### Example

```cpp
class CImg<unsigned char> img("reference.jpg"); // Load a RGB color image
class CImgList<unsigned char> list = (img<'c'); // Get a list of its three R,G,B channels
(img,list).display();
```

### 8.1.4.102 pixel_type()

```cpp
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.

### 8.1.4.103 width()

```cpp
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`. 

---

Generated by Doxygen
8.1.4.104 height()

```cpp
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()

```cpp
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()

```cpp
int spectrum() const
```

Return the number of image channels.

Return the number of image channels, i.e. the image dimension along the C-axis.

**Note**

- The `spectrum()` of an empty image is equal to 0.
- `spectrum()` is typically equal to 1 when considering scalar-valued images, to 3 for RGB-coded color images, and to 4 for RGBA-coded color images (with alpha-channel). The number of channels of an image instance is not limited. The meaning of the pixel values is not linked up to the number of channels (e.g. a 4-channel image may indifferently stands for a RGBA or CMYK color image).
- `spectrum()` returns an `int`, although the image spectrum is internally stored as an `unsigned int`. Using an `int` is safer and prevents arithmetic traps possibly encountered when doing calculations involving `unsigned int` variables. Access to the initial `unsigned int` variable is possible (though not recommended) by `(*this)._spectrum`.

Generated by Doxygen
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>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>X-coordinate of the pixel value.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>Y-coordinate of the pixel value.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>Z-coordinate of the pixel value.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>C-coordinate of the pixel value.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note

- Writing \texttt{img.data(x, y, z, c)} is equivalent to \texttt{& (img(x, y, z, c))}. Thus, this method has the same properties as \texttt{operator(unsigned int, unsigned int, unsigned int, unsigned int)}.

8.1.4.110 \texttt{offset()}

longT offset {
    const int x,
    const int y = 0,
    const int z = 0,
    const int c = 0)
}

Return the offset to a located pixel value, with respect to the beginning of the pixel buffer.

Parameters

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>X-coordinate of the pixel value.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>Y-coordinate of the pixel value.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>Z-coordinate of the pixel value.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>C-coordinate of the pixel value.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note

- Writing \texttt{img.data(x, y, z, c)} is equivalent to \texttt{& (img(x, y, z, c)) - img.data()}. Thus, this method has the same properties as \texttt{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 \texttt{begin()}

iterator begin ()

Return a \texttt{CImg&lt;T&gt;::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]

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 )
```

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()

```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**

- **x**: X-coordinate of the pixel value.
- **y**: Y-coordinate of the pixel value.
- **z**: Z-coordinate of the pixel value.
- **c**: C-coordinate of the pixel value.

**Note**

- Similar to `at(int,int,int,int,const T)`, except that an out-of-bounds access returns the value of the nearest pixel in the image instance, regarding the specified X-coordinate.
- Due to the additional boundary checking operation, this method is slower than `operator()()`. Use it when you are not sure about the validity of the specified pixel coordinates.
- If you know your image instance is not empty, you may rather use the slightly faster method `at(int,int,int,int).`

**Warning**

- There is no boundary checking performed for the Y,Z and C-coordinates, so they must be inside image bounds.

### 8.1.4.119 atXY()

```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,const T)`, except that boundary checking is performed both on X and Y-coordinates.
8.1.4.120  atXY() [2/2]

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), 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]

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), 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] 

```cpp
T& atXYZC (  
    const int x,  
    const int y,  
    const int z,  
    const int c,  
    const T & out_value )
```

Access to a pixel value, using Dirichlet boundary conditions.

Similar to `atX(int,int,int,int,const T)`, except that boundary checking is performed on all X,Y,Z and C-coordinates.

8.1.4.124  `atXYZC()` [2/2] 

```cpp
T& atXYZC (  
    const int x,  
    const int y,  
    const int z,  
    const int c  
)
```

Access to a pixel value, using Neumann boundary conditions.

Similar to `atX(int,int,int,int)`, except that boundary checking is performed on all X,Y,Z and C-coordinates.

Note

- If you know your image instance is **not** empty, you may rather use the slightly faster method `C(int,int,int,int)`.

8.1.4.125  `linear_atX()` [1/2] 

```cpp
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 `atX(int,int,int,const T)`, except that the returned pixel value is approximated by a linear interpolation along the X-axis, if corresponding coordinates are not integers.
- The type of the returned pixel value is extended to `float`, if the pixel type `T` is not float-valued.

Warning

- There is no boundary checking performed for the Y,Z and C-coordinates, so they must be inside image bounds.

8.1.4.126 linear_atX() [2/2]

```cpp
tfloat linear_atX (const float fx, const int y = 0, const int z = 0, const int c = 0) const
```

Return pixel value, using linear interpolation and Neumann boundary conditions for the X-coordinate.

Return a linearly-interpolated pixel value of the image instance located at `(fx,y,z,c)`, or the value of the nearest pixel location in the image instance in case of out-of-bounds access along the X-axis.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fx</code></td>
<td>X-coordinate of the pixel value (float-valued).</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>
<tr>
<td><code>c</code></td>
<td>C-coordinate of the pixel value.</td>
</tr>
</tbody>
</table>

Note

- Similar to `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 not empty, you may rather use the slightly faster method `_linear←_atX(float,int,int)`. 

Warning

- There is no boundary checking performed for the Y,Z and C-coordinates, so they must be inside image bounds.
8.1.4.127 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,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 int z = 0,  
    const int c = 0 ) const
```

Return pixel value, using linear interpolation and Neumann boundary conditions for the X and Y-coordinates.

Similar to linear_atX(float,int,int,int) const, except that the linear interpolation and the boundary checking are achieved both for X and Y-coordinates.

Note

- If you know your image instance is not empty, you may rather use the slightly faster method \_linear←\_atXY(float,float,int,int).

8.1.4.129 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,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 linear_atXYZ() [2/2]

Tfloat linear_atXYZ (  
    const float fx,  
    const float fy = 0,  
    const float fz = 0,  
    const int c = 0 ) const

Return pixel value, using linear interpolation and Neumann boundary conditions for the X,Y and Z-coordinates.

Similar to linear_atX(float,int,int,int) const, except that the linear interpolation and the boundary checking are achieved both for X,Y and Z-coordinates.

Note

- If you know your image instance is not empty, you may rather use the slightly faster method _linear←_atXYZC(float,float,float,int).

8.1.4.131 linear_atXYZC() [1/2]

Tfloat linear_atXYZC (  
    const float fx,  
    const float fy,  
    const float fz,  
    const float fc,  
    const T & out_value ) const

Return pixel value, using linear interpolation and Dirichlet boundary conditions for all X,Y,Z,C-coordinates.

Similar to linear_atX(float,int,int,int,const T) const, except that the linear interpolation and the boundary checking are achieved for all X,Y,Z and C-coordinates.

8.1.4.132 linear_atXYZC() [2/2]

Tfloat linear_atXYZC (  
    const float fx,  
    const float fy = 0,  
    const float fz = 0,  
    const float fc = 0 ) const

Return pixel value, using linear interpolation and Neumann boundary conditions for all X,Y,Z and C-coordinates.

Similar to linear_atX(float,int,int,int) const, except that the linear interpolation and the boundary checking are achieved for all X,Y,Z and C-coordinates.

Note

- If you know your image instance is not empty, you may rather use the slightly faster method _linear←_atXYZC(float,float,float,float).
8.1.4.133 cubic_atX() [1/2]

T float cubic_atX (const float fx, const int y, const int z, const int c, const T & out_value) const

Return pixel value, using cubic interpolation and Dirichlet boundary conditions for the X-coordinate.

Return a cubicly-interpolated pixel value of the image instance located at \((fx, y, z, c)\), or a specified default value in case of out-of-bounds access along the X-axis. 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>
<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 linear_atX(float,int,int,int,const T) const, except that the returned pixel value is approximated by a cubic interpolation along the X-axis.
- The type of the returned pixel value is extended to float, if the pixel type \(T\) is not float-valued.

Warning

- There is no boundary checking performed for the Y,Z and C-coordinates, so they must be inside image bounds.

8.1.4.134 cubic_atX_c() [1/2]

T cubic_atX_c (const float fx, const int y, const int z, const int c, const T & out_value) const

Return clamped pixel value, using cubic interpolation and Dirichlet boundary conditions for the X-coordinate.

Similar to cubic_atX(float,int,int,int,const T) const, except that the return value is clamped to stay in the min/max range of the datatype \(T\).

Generated by Doxygen
8.1.4.135  cubic_atX() [2/2]

Tfloat cubic_atX(
    const float fx,
    const int y = 0,
    const int z = 0,
    const int c = 0 ) const

Return pixel value, using cubic interpolation and Neumann boundary conditions for the X-coordinate.

Return a cubicly-interpolated pixel value of the image instance located at (fx,y,z,c), or the value of the nearest pixel location in the image instance in case of out-of-bounds access along the X-axis. 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]

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_atXY(float,float,int,int,const T) const, except that the return value is clamped to stay in the min/max range of the datatype T.

8.1.4.139 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,const T) 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}. 

Generated by Doxygen
8.1.4.140 cubic_atXY_c() [2/2]

```cpp
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]

```cpp
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]

```cpp
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]

```cpp
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)`. 

---

Generated by Doxygen
8.1.4.144 cubic_atXYZ_c() [2/2]

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>value</th>
<th>Pixel value to set.</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>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()

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()

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()

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>separator</th>
<th>A char character which specifies the separator between values in the returned C-string.</th>
</tr>
</thead>
<tbody>
<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 text 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.
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 CImg< t > & img ) const  
```

Test if image width and height are the same as that of another image.

Test if `is_sameX(const CImg< t >&) const` and `is_sameY(const CImg< t >&) const` are both verified.

**8.1.4.156 is_sameXY() [3/3]**

```cpp
bool is_sameXY (  
                   const CImgDisplay & disp ) const  
```

Test if image width and height are the same as that of an existing display window.

Test if `is_sameX(const CImgDisplay&) const` and `is_sameY(const CImgDisplay&) const` are both verified.

**8.1.4.157 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.
### 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.

### 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.

### 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.

### 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.

### 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.
8.1.4.163 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.
8.1.4.168  is_sameXYZ() [2/2]

bool is_sameXYZ (  
    const CImg< t >& img ) const  

Test if image width, height and depth are the same as that of another image.

Test if is_sameXY(const CImg< t >& ) const and is_sameZ(const CImg< t >& ) const are both verified.

8.1.4.169  is_sameXYC() [1/2]

bool is_sameXYC (  
    const unsigned int size_x,  
    const unsigned int size_y,  
    const unsigned int size_c ) const  

Test if image width, height and spectrum are equal to specified values.

Test if is_sameXY(unsigned int,unsigned int) const and is_sameC(unsigned int) const are both verified.

8.1.4.170  is_sameXYC() [2/2]

bool is_sameXYC (  
    const CImg< t >& img ) const  

Test if image width, height and spectrum are the same as that of another image.

Test if is_sameXY(const CImg< t >& ) const and is_sameC(const CImg< t >& ) const are both verified.

8.1.4.171  is_sameXZC() [1/2]

bool is_sameXZC (  
    const unsigned int size_x,  
    const unsigned int size_z,  
    const unsigned int size_c ) const  

Test if image width, depth and spectrum are equal to specified values.

Test if is_sameXZ(unsigned int,unsigned int) const and is_sameC(unsigned int) const are both verified.

8.1.4.172  is_sameXZC() [2/2]

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_sameYZ(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 >& img) const and is_sameC(const CImg< t >& img) 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 >& img) const and is_sameC(const CImg< t >& img) 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.
8.1 CImg< T > Struct Template Reference

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<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>
<tr>
<td><code>c</code></td>
<td>C-coordinate of the pixel value.</td>
</tr>
</tbody>
</table>

Note

- Return `true` only if all these conditions are verified:
  - The image instance is *not* empty.
  - `0<=x<=width() - 1`.
  - `0<=y<=height() - 1`.
  - `0<=z<=depth() - 1`.
  - `0<=c<=spectrum() - 1`.

8.1.4.178 `contains()` [1/5]

```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>
</tr>
</thead>
<tbody>
<tr>
<td><code>pixel</code></td>
<td>Reference to pixel value to test.</td>
</tr>
<tr>
<td><code>out x</code></td>
<td>X-coordinate of the pixel value, if test succeeds.</td>
</tr>
<tr>
<td><code>out y</code></td>
<td>Y-coordinate of the pixel value, if test succeeds.</td>
</tr>
<tr>
<td><code>out z</code></td>
<td>Z-coordinate of the pixel value, if test succeeds.</td>
</tr>
<tr>
<td><code>out c</code></td>
<td>C-coordinate of the pixel value, if test succeeds.</td>
</tr>
</tbody>
</table>

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&,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&,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&,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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>img</td>
<td>Input image to compare with.</td>
</tr>
</tbody>
</table>

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>out 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><code>full_check</code></td>
<td>Tells if full checking of the instance must be performed.</td>
</tr>
<tr>
<td><code>error_message</code></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 $\sqrt{I_{(x,y,z,c)}}$.

**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.

**Example**

```cpp
const CImg<float> img("reference.jpg");
{img, img.get_sqrt().normalize(0,255)}.display();
```

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 $e^{I_{(x,y,z,c)}}$.

**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.

8.1.4.189 \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 $\log_e{(I_{(x,y,z,c)})}$.

**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.
8.1.4.190 log2()

CImg<T>& log2 ( )

Compute the base-2 logarithm of each pixel value.

Replace each pixel value \( I_{(x,y,z,c)} \) of the image instance by its base-2 logarithm \( \log_2(I_{(x,y,z,c)}) \).

Note

- The [in-place version] of this method statically casts the computed values to the pixel type T.
- The [new-instance version] returns a CImg<float> image, if the pixel type T is not float-valued.

8.1.4.191 log10()

CImg<T>& log10 ( )

Compute the base-10 logarithm of each pixel value.

Replace each pixel value \( I_{(x,y,z,c)} \) of the image instance by its base-10 logarithm \( \log_{10}(I_{(x,y,z,c)}) \).

Note

- The [in-place version] of this method statically casts the computed values to the pixel type T.
- The [new-instance version] returns a CImg<float> image, if the pixel type T is not float-valued.

8.1.4.192 abs()

CImg<T>& abs ( )

Compute the absolute value of each pixel value.

Replace each pixel value \( I_{(x,y,z,c)} \) of the image instance by its absolute value \( |I_{(x,y,z,c)}| \).

Note

- The [in-place version] of this method statically casts the computed values to the pixel type T.
- The [new-instance version] returns a CImg<float> image, if the pixel type T is not float-valued.
8.1.4.193  sign()

\texttt{CImg<\textgreater T\textless> sign ( )}

Compute the sign of each pixel value.

Replace each pixel value $I(x,y,z,c)$ of the image instance by its sign $\text{sign}(I(x,y,z,c))$.

\textbf{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 \texttt{CImg<float>} image, if the pixel type $T$ is \textit{not} float-valued.

8.1.4.194  cos()

\texttt{CImg<\textgreater T\textless> cos ( )}

Compute the cosine of each pixel value.

Replace each pixel value $I(x,y,z,c)$ of the image instance by its cosine $\cos(I(x,y,z,c))$.

\textbf{Note}

- Pixel values are regarded as being in \textit{radian}.
- 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.

8.1.4.195  sin()

\texttt{CImg<\textgreater T\textless> sin ( )}

Compute the sine of each pixel value.

Replace each pixel value $I(x,y,z,c)$ of the image instance by its sine $\sin(I(x,y,z,c))$.

\textbf{Note}

- Pixel values are regarded as being in \textit{radian}.
- 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.
8.1.4.196 sinc()  

\texttt{CImg<T>\& sinc ( ) }

Compute the sinc of each pixel value.

Replace each pixel value \( I(x,y,z,c) \) of the image instance by its sinc \( \text{sinc}(I(x,y,z,c)) \).

\textbf{Note}  
- Pixel values are regarded as being exin radian.
- 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 \textit{not} float-valued.

8.1.4.197 tan()  

\texttt{CImg<T>\& tan ( ) }

Compute the tangent of each pixel value.

Replace each pixel value \( I(x,y,z,c) \) of the image instance by its tangent \( \tan(I(x,y,z,c)) \).

\textbf{Note}  
- Pixel values are regarded as being exin radian.
- 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 \textit{not} float-valued.

8.1.4.198 cosh()  

\texttt{CImg<T>\& cosh ( ) }

Compute the hyperbolic cosine of each pixel value.

Replace each pixel value \( I(x,y,z,c) \) of the image instance by its hyperbolic cosine \( \cosh(I(x,y,z,c)) \).

\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 \textit{not} float-valued.
8.1.4.199 sinh()

\texttt{CImg<\textless T\textgreater 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 [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.

8.1.4.200 tanh()

\texttt{CImg<\textless T\textgreater 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 [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.

8.1.4.201 acos()

\texttt{CImg<\textless T\textgreater 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 \( \arccos(I_{(x,y,z,c)}) \).

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.
8.1.4.202  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 \( \text{asin}(I(x,y,z,c)) \).

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 \textit{not} float-valued.

8.1.4.203  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 \( \text{atan}(I(x,y,z,c)) \).

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 \textit{not} float-valued.

8.1.4.204  atan2()

\texttt{CImg<T>& atan2 ( )}

\begin{verbatim}
c const CImg< t > & img
\end{verbatim}

Compute the arctangent2 of each pixel value.

Replace each pixel value \( I(x,y,z,c) \) of the image instance by its arctangent2 \( \text{atan2}(I(x,y,z,c)) \).

Parameters

\begin{table}
\begin{tabular}{|l|}
\hline
\textbf{img} & Image whose pixel values specify the second argument of the \texttt{atan2()} function. \\
\hline
\end{tabular}
\end{table}

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 \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()

\texttt{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 \( \text{acosh}(I_{(x,y,z,c)}) \).

\begin{itemize}
  \item The [\textbf{in-place version}] of this method statically casts the computed values to the pixel type \( T \).
  \item The [\textbf{new-instance version}] returns a \texttt{CImg<float>} image, if the pixel type \( T \) is \textit{not} float-valued.
\end{itemize}

8.1.4.206 asinh()

\texttt{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 \( \text{asinh}(I_{(x,y,z,c)}) \).

\begin{itemize}
  \item The [\textbf{in-place version}] of this method statically casts the computed values to the pixel type \( T \).
  \item The [\textbf{new-instance version}] returns a \texttt{CImg<float>} image, if the pixel type \( T \) is \textit{not} float-valued.
\end{itemize}

8.1.4.207 atanh()

\texttt{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 \( \text{atanh}(I_{(x,y,z,c)}) \).

\begin{itemize}
  \item The [\textbf{in-place version}] of this method statically casts the computed values to the pixel type \( T \).
  \item The [\textbf{new-instance version}] returns a \texttt{CImg<float>} image, if the pixel type \( T \) is \textit{not} float-valued.
\end{itemize}

8.1.4.208 mul()

\texttt{CImg<T>& mul (const CImg \(< T > & \ img \) )}

In-place pointwise multiplication.

Compute the pointwise multiplication between the image instance and the specified input image \texttt{img}.
Parameters

| img  | Input image, as the second operand of the multiplication. |

Note

- Similar to `operator+=(const CImg<T>&)`, except that it performs a pointwise multiplication instead of an addition.
- It does not perform a matrix multiplication. For this purpose, use `operator*=(const CImg<T>&)` instead.

Example

```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()

```cpp
CImg<T>& div (const CImg<T>& img)
```

In-place pointwise division.

Similar to `mul(const CImg<T>&)`, except that it performs a pointwise division instead of a multiplication.

8.1.4.210 pow() [1/3]

```cpp
CImg<T>& pow (const double p)
```

Raise each pixel value to a specified power.

Replace each pixel value $I(x,y,z,c)$ of the image instance by its power $I^p(x,y,z,c)$.

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
```
8.1.4.211 pow() [2/3]

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]

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]

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]

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]

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 ror() [1/3]

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

Compute the bitwise right rotation of each pixel value.
Similar to operator \texttt{>>}=(unsigned int), except that it performs a right rotation instead of a right shift.

8.1.4.217 ror() [2/3]

CImg<
T
>& ror (  
    const char *const expression  
)

Compute the bitwise right rotation of each pixel value.
Similar to operator \texttt{>>}=(const char*), except that it performs a right rotation instead of a right shift.

8.1.4.218 ror() [3/3]

CImg<
T
>& ror (  
    const CImg<
t>& img  
)

Compute the bitwise right rotation of each pixel value.
Similar to operator \texttt{>>}=(const CImg<
t>&), except that it performs a right rotation instead of a right shift.

8.1.4.219 min() [1/3]

CImg<
T
>& min (  
    const T & value  
)

Pointwise min operator between instance image and a value.

Parameters

| val | Value used as the reference argument of the min operator. |

Note

Replace each pixel value \(I_{(x,y,z,c)}\) of the image instance by \(\text{min}(I_{(x,y,z,c)}, \text{val})\).

8.1.4.220 min() [2/3]

CImg<
T
>& min (  
    const CImg<
t>& img  
)

Pointwise min operator between two images.
Parameters

\[ \text{img} \] | 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]

\text{CImg}<T> & \text{min} ( \\
& \text{const char *const expression} )

Pointwise min 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 \( \min(I_{(x,y,z,c)}, \text{expr}_{(x,y,z,c)}) \).

8.1.4.222 max() [1/3]

\text{CImg}<T> & \text{max} ( \\
& \text{const T & value} )

Pointwise max operator between instance image and a value.

Parameters

\[ \text{val} \] | 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]

\text{CImg}<T> & \text{max} ( \\
& \text{const CImg< t > & img} )
Pointwise max operator between two images.
Parameters

\[ img \] 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 \texttt{max() [3/3]}

\texttt{CImg} < \texttt{T} \texttt{ max (}
  \texttt{ 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 \texttt{minabs()} [1/3]

\texttt{CImg} < \texttt{T} \texttt{ minabs (}
  \texttt{ const T & value )}

Pointwise minabs operator between instance image and a value.

Parameters

\[ \text{val} \] Value used as the reference argument of the minabs operator.

Note

Replace each pixel value \( I_{(x,y,z,c)} \) of the image instance by \( \minabs(I_{(x,y,z,c)}, \text{val}) \).

8.1.4.226 \texttt{minabs()} [2/3]

\texttt{CImg} < \texttt{T} \texttt{ minabs (}
  \texttt{ const CImg< t > & img )}

Generated by Doxygen
Pointwise minabs operator between two images.
8.1 Clmg< T > Struct Template Reference

Parameters

| img | Image used as the reference argument of the minabs operator. |

Note

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by $\minabs(I_{(x,y,z,c)}), \text{img}_{(x,y,z,c)}$.

8.1.4.227 minabs() [3/3]

\texttt{Clmg<T>} & \minabs ( \\
  \texttt{const char *const expression})

Pointwise minabs 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 $\minabs(I_{(x,y,z,c)}), \text{expr}_{(x,y,z,c)}$.

8.1.4.228 maxabs() [1/3]

\texttt{Clmg<T>} & \maxabs ( \\
  \texttt{const T & value})

Pointwise maxabs operator between instance image and a value.

Parameters

| val | Value used as the reference argument of the maxabs operator. |

Note

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by $\maxabs(I_{(x,y,z,c)}, \text{val})$.

8.1.4.229 maxabs() [2/3]

\texttt{Clmg<T>} & \maxabs ( \\
  \texttt{const Clmg< t > & img})
Pointwise maxabs operator between two images.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>img</td>
<td>Image used as the reference argument of the maxabs operator.</td>
</tr>
</tbody>
</table>

Note

Replace each pixel value \( I_{(x,y,z,c)} \) of the image instance by \( \text{maxabs}(I_{(x,y,z,c)}, \text{img}_{(x,y,z,c)}) \).

8.1.4.230 maxabs() [3/3]

\texttt{CImg<\textgreater T\textless> & maxabs (const char*const expression)}

Pointwise maxabs operator between an image and an expression.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression</td>
<td>Math formula as a C-string.</td>
</tr>
</tbody>
</table>

Note

Replace each pixel value \( I_{(x,y,z,c)} \) of the image instance by \( \text{maxabs}(I_{(x,y,z,c)}, \text{expr}_{(x,y,z,c)}) \).

8.1.4.231 min_max()  

\texttt{T& min_max (t & max_val)}

Return a reference to the minimum pixel value as well as the maximum pixel value.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>max_val</td>
</tr>
</tbody>
</table>

8.1.4.232 max_min()  

\texttt{T& max_min (t & min_val)}

Return a reference to the maximum pixel value as well as the minimum pixel value.
Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>min_val</th>
<th>Minimum pixel value.</th>
</tr>
</thead>
</table>

8.1.4.233  kth_smallest()

```cpp
t kth_smallest ( const ulongT k ) const
```

Return the kth smallest pixel value.

Parameters

<table>
<thead>
<tr>
<th>k</th>
<th>Rank of the search smallest element.</th>
</tr>
</thead>
</table>

8.1.4.234  variance()

```cpp
double variance ( const unsigned int variance_method = 1 ) 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</td>
</tr>
<tr>
<td></td>
<td>[1 / N \sum_{k=1}^{N} (x_k - \bar{x})^2 = 1 / N \left( \sum_{k=1}^{N} x_k^2 - \left( \sum_{k=1}^{N} x_k \right)^2 / N \right) ] with ( \bar{x} = 1 / N \sum_{k=1}^{N} x_k ).</td>
</tr>
<tr>
<td>1</td>
<td>Best unbiased estimator, computed as [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>

8.1.4.235  variance_mean()

```cpp
double variance_mean ( const unsigned int variance_method, t & mean ) const
```

Return the variance as well as the average of the pixel values.
### Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variance_method</td>
<td>Method used to estimate the variance (see variance(const unsigned int) const).</td>
</tr>
<tr>
<td>out mean</td>
<td>Average pixel value.</td>
</tr>
</tbody>
</table>

#### 8.1.4.236 `variance_noise()`

```cpp
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.

#### 8.1.4.237 `MSE()`

```cpp
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.

#### 8.1.4.238 `PSNR()`

```cpp
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>

8.1.4.239 `eval()` [1/3]

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>

8.1.4.240 `eval()` [2/3]

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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression</td>
<td>Math formula, as a C-string.</td>
</tr>
<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 list_outputs</td>
<td>A pointer to a list of output images attached to the specified math formula.</td>
</tr>
</tbody>
</table>

#### 8.1.4.241 eval()

```cpp
CImg< double 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.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression</td>
<td>Math formula, as a C-string.</td>
</tr>
<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 list_outputs</td>
<td>A pointer to a list of output images attached to the specified math formula.</td>
</tr>
</tbody>
</table>

#### 8.1.4.242 magnitude()

```cpp
double magnitude (  
    const int magnitude_type = 2 ) const
```

Compute norm of the image, viewed as a matrix.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>magnitude_type</td>
<td>Norm type. Can be:</td>
</tr>
<tr>
<td></td>
<td>• -1: Linf-norm</td>
</tr>
<tr>
<td></td>
<td>• 0: L0-norm</td>
</tr>
<tr>
<td></td>
<td>• 1: L1-norm</td>
</tr>
<tr>
<td></td>
<td>• 2: L2-norm</td>
</tr>
</tbody>
</table>
8.1.4.243  dot()

```cpp
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.

8.1.4.244  get_vector_at()

```cpp
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.

8.1.4.245  get_matrix_at()

```cpp
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.

### 8.1.4.246 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.

### 8.1.4.247 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.

### 8.1.4.248 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 )
```

Generated by Doxygen
Set (square) matrix-valued pixel at specified position.
Parameters

<table>
<thead>
<tr>
<th>mat</th>
<th>Matrix to put on the instance image.</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>
</tbody>
</table>

8.1.4.249 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>ten</th>
<th>Tensor to put on the instance image.</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>
</tbody>
</table>

8.1.4.250 vector() [1/5]

```cpp
CImg<T>& vector()
```

Unroll pixel values along axis y.

Note

Equivalent to

```cpp
unroll('y');
```

8.1.4.251 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.
8.1.4.252 identity_matrix() [1/2]

CImg< T > & identity_matrix ( )

Replace the image by an identity matrix.

Note
If the instance image is not square, it is resized to a square matrix using its maximum dimension as a reference.

8.1.4.253 sequence() [1/2]

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

Fill image with a linear sequence of values.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</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>

8.1.4.254 transpose()

CImg< T > & transpose ( )

Transpose the image, viewed as a matrix.

Note
Equivalent to

```cpp
cpermute_axes("yxzc");
```

8.1.4.255 cross()

CImg< T > & cross ( const CImg< T > & img )

Compute the cross product between two 1x3 images, viewed as 3D vectors.
Parameters

| img | Image used as the second argument of the cross product. |

Note

The first argument of the cross product is `*this`.

8.1.4.256 invert()

```cpp
CImg<T> & invert (
    const bool use_LU = true )
```

Invert the instance image, viewed as a matrix.

Parameters

| use_LU | Choose the inverting algorithm. Can be:
|--------|-----------------------------------------|
| • true: LU-based matrix inversion.
| • false: SVD-based matrix inversion. |

8.1.4.257 solve()

```cpp
CImg<T> & solve ( 
    const CImg< t > & A )
```

Solve a system of linear equations.

Parameters

| A | Matrix of the linear system. |

Note

Solve $AX = B$ where $B = *this$.

8.1.4.258 solve_tridiagonal()

```cpp
CImg<T> & solve_tridiagonal ( 
    const CImg< t > & A )
```

Solve a tridiagonal system of linear equations.
Parameters

A

Coefficients of the tridiagonal system. A is a tridiagonal matrix \( A = [ b_0, c_0, 0, \ldots; a_1, b_1, c_1, 0, \ldots; \ldots; 0, a_N, b_N ] \), stored as a 3 columns matrix

Note

Solve \( AX = B \) where \( B = \text{this} \), using the Thomas algorithm.

8.1.4.259 eigen()

```cpp
const CImg<

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.

8.1.4.260 get_eigen()

```

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.

8.1.4.261 symmetric_eigen()

```cpp
const CImg<

Compute eigenvalues and eigenvectors of the instance image, viewed as a symmetric matrix.
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>

8.1.4.262 `get_symmetric_eigen()`

```cpp
CImgList<Tfloat> get_symmetric_eigen() const
```

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

Returns

A list of two images `[val; vec]`, whose meaning are similar as in `symmetric_eigen(CImg<T>&, CImg<T>&) const`.

8.1.4.263 `sort()` [1/2]

```cpp
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>

8.1.4.264 `sort()` [2/2]

```cpp
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>

8.1.4.265 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 \(*this=U*S*V'*\) where \(U\) and \(V\) are orthogonal matrices and \(S\) is a diagonal matrix. \(V'\) denotes the matrix transpose of \(V\).

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>First matrix of the SVD product.</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>Coefficients of the second (diagonal) matrix of the SVD product. These coefficients are stored as a vector.</td>
</tr>
<tr>
<td>V</td>
<td>Third matrix of the SVD product.</td>
</tr>
<tr>
<td>sorting</td>
<td>Tells if the diagonal coefficients are sorted (in decreasing order).</td>
</tr>
<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);
```

8.1.4.266 get_SVD()

```cpp
CImgList<Tfloat> get_SVD {
    const bool sorting = true,
```

Generated by Doxygen
const unsigned int max_iteration = 40,
const float lambda = 0.0) 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 \texttt{SVD(CImg<\texttt{t}>\&,CImg<\texttt{t}>\&.CImg<\texttt{t}>\&.bool,unsigned int,float) const}.

\subsection{dijkstra() [1/2]}

\begin{verbatim}
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]
\end{verbatim}

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>distance</td>
<td>An object having operator((i, j)) which returns distance between two nodes ((i,j)).</td>
</tr>
<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 (~0\U) 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.

\subsection{dijkstra() [2/2]}

\begin{verbatim}
CImg<T>& dijkstra (    
    const unsigned int starting_node,
    const unsigned int ending_node,
    CImg<T> & previous_node )
\end{verbatim}

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>starting_node</td>
<td>Index of the starting node.</td>
</tr>
<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.

8.1.4.269  string()

```
static CImg<T> string (  
    const char *const str,  
    const bool is_last_zero = true,  
    const bool is_shared = false ) [static]
```

Return an image containing the character codes of specified string.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>input C-string to encode as an image.</td>
</tr>
<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>

8.1.4.270  vector() [2/5]

```
static CImg<T> vector (  
    const T & a0 ) [static]
```

Return a 1x1 image containing specified value.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td>First vector value.</td>
</tr>
</tbody>
</table>

8.1.4.271  vector() [3/5]

```
static CImg<T> vector (  
    const T & a0,  
    const T & a1 ) [static]
```

Return a 1x2 image containing specified values.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_0$</td>
<td>First vector value.</td>
</tr>
<tr>
<td>$a_1$</td>
<td>Second vector value.</td>
</tr>
</tbody>
</table>

8.1.4.272   vector() [4/5]

static CImg$<$T$>$ vector (const T & $a_0$, const T & $a_1$, const T & $a_2$) [static]

Return a 1x3 image containing specified values.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_0$</td>
<td>First vector value.</td>
</tr>
<tr>
<td>$a_1$</td>
<td>Second vector value.</td>
</tr>
<tr>
<td>$a_2$</td>
<td>Third vector value.</td>
</tr>
</tbody>
</table>

8.1.4.273   vector() [5/5]

static CImg$<$T$>$ vector (const T & $a_0$, const T & $a_1$, const T & $a_2$, const T & $a_3$) [static]

Return a 1x4 image containing specified values.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_0$</td>
<td>First vector value.</td>
</tr>
<tr>
<td>$a_1$</td>
<td>Second vector value.</td>
</tr>
<tr>
<td>$a_2$</td>
<td>Third vector value.</td>
</tr>
<tr>
<td>$a_3$</td>
<td>Fourth vector value.</td>
</tr>
</tbody>
</table>

8.1.4.274   matrix() [1/3]

static CImg$<$T$>$ matrix (const T & $a_0$) [static]

Return a 1x1 matrix containing specified coefficients.

Generated by Doxygen
Parameters

\begin{align*}
\text{a0} & \quad \text{First matrix value.} \\
\end{align*}

Note

Equivalent to \text{vector(const T &)}. 

\textbf{8.1.4.275 matrix() [2/3]}

static \text{CImg\textless T\textgreater} matrix ( 
\quad \text{const T & a0,} \\
\quad \text{const T & a1,} \\
\quad \text{const T & a2,} \\
\quad \text{const T & a3} \} \quad \text{[static]}

Return a 2x2 matrix containing specified coefficients.

Parameters

\begin{align*}
\text{a0} & \quad \text{First matrix value.} \\
\text{a1} & \quad \text{Second matrix value.} \\
\text{a2} & \quad \text{Third matrix value.} \\
\text{a3} & \quad \text{Fourth matrix value.} \\
\end{align*}

\textbf{8.1.4.276 matrix() [3/3]}

static \text{CImg\textless T\textgreater} matrix ( 
\quad \text{const T & a0,} \\
\quad \text{const T & a1,} \\
\quad \text{const T & a2,} \\
\quad \text{const T & a3,} \\
\quad \text{const T & a4,} \\
\quad \text{const T & a5,} \\
\quad \text{const T & a6,} \\
\quad \text{const T & a7,} \\
\quad \text{const T & a8} \} \quad \text{[static]}

Return a 3x3 matrix containing specified coefficients.

Parameters

\begin{align*}
\text{a0} & \quad \text{First matrix value.} \\
\text{a1} & \quad \text{Second matrix value.} \\
\text{a2} & \quad \text{Third matrix value.} \\
\text{a3} & \quad \text{Fourth matrix value.} \\
\text{a4} & \quad \text{Fifth matrix value.} \\
\end{align*}
Parameters

<table>
<thead>
<tr>
<th>a5</th>
<th>Sixth matrix value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>a6</td>
<td>Seventh matrix value.</td>
</tr>
<tr>
<td>a7</td>
<td>Eighth matrix value.</td>
</tr>
<tr>
<td>a8</td>
<td>Ninth matrix value.</td>
</tr>
</tbody>
</table>

8.1.4.277  tensor()

static CImg<T> tensor (  
    const T & a0  ) [static]

Return a 1x1 symmetric matrix containing specified coefficients.

Parameters

| a0  | First matrix value. |

Note

Equivalent to vector(const T&).

8.1.4.278  identity_matrix() [2/2]

static CImg<T> identity_matrix (  
    const unsigned int N  ) [static]

Return a NxN identity matrix.

Parameters

| N   | Dimension of the matrix. |

8.1.4.279  sequence() [2/2]

static CImg<T> sequence (  
    const unsigned int N,
    const T & a0,
    const T & a1  ) [static]

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

<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>

#### 8.1.4.280 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

<table>
<thead>
<tr>
<th>x</th>
<th>X-coordinate of the rotation axis, or first quaternion coordinate.</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>Y-coordinate of the rotation axis, or second quaternion coordinate.</td>
</tr>
<tr>
<td>z</td>
<td>Z-coordinate of the rotation axis, or third quaternion coordinate.</td>
</tr>
<tr>
<td>w</td>
<td>Angle of the rotation axis (in degree), or fourth quaternion coordinate.</td>
</tr>
<tr>
<td>is_quaternion</td>
<td>Tell is the four arguments denotes a set { axis + angle } or a quaternion (x,y,z,w).</td>
</tr>
</tbody>
</table>

#### 8.1.4.281 fill() [1/4]

```cpp
CImg<T>& fill ( const T & val )
```

Fill all pixel values with specified value.

#### Parameters

| val | Fill value. |

#### 8.1.4.282 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>

8.1.4.283 fill() [3/4]

```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

<table>
<thead>
<tr>
<th>expression</th>
<th>C-string describing a math formula, or a sequence of values.</th>
</tr>
</thead>
<tbody>
<tr>
<td>repeat_values</td>
<td>In case a list of values is provided, tells if this list must be repeated for the filling.</td>
</tr>
<tr>
<td>allow_formula</td>
<td>Tells that mathematical formulas are authorized for the filling.</td>
</tr>
<tr>
<td>list_inputs</td>
<td>In case of a mathematical expression, attach a list of images to the specified expression.</td>
</tr>
<tr>
<td>list_outputs</td>
<td>In case of a math expression, list of images attached to the specified expression.</td>
</tr>
</tbody>
</table>

8.1.4.284 fill() [4/4]

```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. |

8.1.4.285 fillX()

```cpp
CImg<T>& fillX(
    const unsigned int y,
```

Generated by Doxygen
const unsigned int z,
const unsigned int c,
const int a0,
...
)

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</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>

8.1.4.286 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</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>

8.1.4.287 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</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>
8.1.4.288  fillC()

\[
\text{CImg}<T>& \text{ fillC}(
    \text{const unsigned int } x,
    \text{const unsigned int } y,
    \text{const unsigned int } z,
    \text{const int } a0,
    \text{...})
\]

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x)</td>
<td>X-coordinate of the filled channel.</td>
</tr>
<tr>
<td>(y)</td>
<td>Y-coordinate of the filled channel.</td>
</tr>
<tr>
<td>(z)</td>
<td>Z-coordinate of the filled channel.</td>
</tr>
<tr>
<td>(a0)</td>
<td>First filling value.</td>
</tr>
</tbody>
</table>

8.1.4.289  discard()

\[
\text{CImg}<T>& \text{ discard}(
    \text{const CImg}<t>& \text{ values},
    \text{const char } axis = 0)
\]

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(values)</td>
<td>Sequence of values to discard.</td>
</tr>
<tr>
<td>(axis)</td>
<td>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.</td>
</tr>
</tbody>
</table>

Note

Discarded values will change the image geometry, so the resulting image is returned as a one-column vector.

8.1.4.290  rand()

\[
\text{CImg}<T>& \text{ rand}(
    \text{const T & } \text{ val_min},
    \text{const T & } \text{ val_max})
\]

Fill image with random values in specified range.
Parameters

| val_min | Minimal authorized random value. |
| val_max | Maximal authorized random value. |

Note

Random variables are uniformly distributed in [val_min,val_max].

8.1.4.291 round()

```cpp
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>-1: Backward.</td>
<td></td>
</tr>
<tr>
<td>0: Nearest.</td>
<td></td>
</tr>
<tr>
<td>1: Forward.</td>
<td></td>
</tr>
</tbody>
</table>

8.1.4.292 noise()

```cpp
cImg<T>& noise {
    const double sigma,
    const unsigned int noise_type = 0
}
```

Add random noise to pixel values.

Parameters

| sigma | Amplitude of the random additive noise. If sigma<0, it stands for a percentage of the global value range. |
| noise_type | Type of additive noise (can be 0=gaussian, 1=uniform, 2=Salt and Pepper, 3=Poisson or 4=Rician). |
Returns

A reference to the modified image instance.

Note

- For Poisson noise \( \text{noise}_\text{type}=3 \), parameter \( \text{sigma} \) is ignored, as Poisson noise only depends on the image value itself.
- Function \( \text{CImg}<T>::\text{get_noise()} \) is also defined. It returns a non-shared modified copy of the image instance.

Example

```cpp
const CImg<float> img("reference.jpg"), res = img.get_noise(40);
{img,res.normalize(0,255)}.display();
```

8.1.4.293 normalize() [1/2]

\( \text{CImg}<T>\& \text{normalize} ( \text{const T & min_value, const T & max_value, const float constant_case_ratio = 0} ) \)

Linearly normalize pixel values.

Parameters

<table>
<thead>
<tr>
<th>\text{min_value}</th>
<th>Minimum desired value of the resulting image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{max_value}</td>
<td>Maximum desired value of the resulting image.</td>
</tr>
<tr>
<td>\text{constant_case_ratio}</td>
<td>In case of instance image having a constant value, tell what ratio of ( [\text{min_value},\text{max_value}] ) is used to fill the normalized image (=0 for \text{min_value}, =1 for \text{max_value}, =0.5 for ( \text{min_value} + \text{max_value} )/2).</td>
</tr>
</tbody>
</table>

Example

```cpp
const CImg<float> img("reference.jpg"), res = img.get_normalize(160,220);
{img,res}.display();
```

8.1.4.294 normalize() [2/2]

\( \text{CImg}<T>\& \text{normalize} ( ) \)

Normalize multi-valued pixels of the image instance, with respect to their L2-norm.

Example

```cpp
const CImg<float> img("reference.jpg"), res = img.get_normalize();
{img,res.normalize(0,255)}.display();
```
8.1.4.295 norm()

```
CImg<T>& norm {
    const int norm_type = 2
}
```

Compute Lp-norm of each multi-valued pixel of the image instance.

**Parameters**

<table>
<thead>
<tr>
<th>norm_type</th>
<th>Type of computed vector norm (can be -1=Linf, or greater or equal than 0).</th>
</tr>
</thead>
</table>

**Example**

```
const CImg<float> img("reference.jpg");
res = img.get_norm();
{img, res}.normalize(0, 255).display();
```

8.1.4.296 cut()

```
CImg<T>& cut {
    const T & min_value,
    const T & max_value
}
```

Cut pixel values in specified range.

**Parameters**

<table>
<thead>
<tr>
<th>min_value</th>
<th>Minimum desired value of the resulting image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_value</td>
<td>Maximum desired value of the resulting image.</td>
</tr>
</tbody>
</table>

**Example**

```
const CImg<float> img("reference.jpg");
res = img.get_cut(160, 220);
{img, res}.display();
```

8.1.4.297 quantize()

```
CImg<T>& quantize {
    const unsigned int nb_levels,
    const bool keep_range = true
}
```

Uniformly quantize pixel values.

**Parameters**

<table>
<thead>
<tr>
<th>nb_levels</th>
<th>Number of quantization levels.</th>
</tr>
</thead>
<tbody>
<tr>
<td>keep_range</td>
<td>Tells if resulting values keep the same range as the original ones.</td>
</tr>
</tbody>
</table>
Example

```cpp
const CImg<float> img("reference.jpg");
const CImg<float> res = img.get_quantize(4);
(img, res).display();
```

### 8.1.4.298 threshold()

```cpp
const 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();
```

### 8.1.4.299 histogram()

```cpp
const 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
class CImgDocumentation
{
private:
    const CImg<
    ...


8.1.4.300  equalize()

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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nb_levels</td>
<td>Number of histogram levels used for the equalization.</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>

Example

```cpp
class CImgDocumentation
{
private:
    const CImg<
    ...


8.1.4.301  index()

CImg<T>& index(
    const CImg<
    ...

Index multi-valued pixels regarding to a specified colormap.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>colormap</td>
<td>Multi-valued colormap used as the basis for multi-valued pixel indexing.</td>
</tr>
<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>

Example

```cpp
class CImgDocumentation
{
private:
    const CImg<
    ...

```
8.1 CImg< T > Struct Template Reference

Note

- `img.index(colormap, dithering, 1)` is equivalent to `img.index(colormap, dithering, 0).map(colormap)`.

Example

```cpp
class CImg< T >
{
    public:
        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>colormap</th>
<th>Multi-valued colormap used for mapping the indexes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>boundary_conditions</td>
<td>Boundary conditions. Can be { 0=dirichlet</td>
</tr>
</tbody>
</table>

Example

```cpp
const CImg<float> img("reference.jpg"), colormap(3,1,1,3, 0,128,255, 0,128,255, 0,128,255);
const CImg<float> res = img.get_index(colormap,1,true);
(img,res).display();
```

8.1.4.302 map()

```
8.1.4.303 label() [1/2]

CImg< T >& label (const bool is_high_connectivity = false, const Tfloat tolerance = 0);

Label connected components.

Parameters

<table>
<thead>
<tr>
<th>is_high_connectivity</th>
<th>Boolean that choose between 4(false)- or 8(true)-connectivity in 2D case, and between 6(false)- or 26(true)-connectivity in 3D case.</th>
</tr>
</thead>
<tbody>
<tr>
<td>tolerance</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.

8.1.4.304 label()

\[2/2\]

\begin{verbatim}
CImg\langle T\rangle & label {
    const CImg\langle T \rangle & connectivity_mask,
    const Tfloat tolerance = 0
}
\end{verbatim}

Label connected components [overloading].

Parameters

<table>
<thead>
<tr>
<th>connectivity_mask</th>
<th>Mask of the neighboring pixels.</th>
</tr>
</thead>
<tbody>
<tr>
<td>tolerance</td>
<td>Tolerance used to determine if two neighboring pixels belong to the same region.</td>
</tr>
</tbody>
</table>

8.1.4.305 default_LUT256()

\begin{verbatim}
static const CImg\langle Tuchar\rangle & default_LUT256 ( ) [static]
\end{verbatim}

Return colormap "default", containing 256 colors entries in RGB.

Returns

The following 256x1x1x3 colormap is returned:

8.1.4.306 HSV_LUT256()

\begin{verbatim}
static const CImg\langle Tuchar\rangle & HSV_LUT256 ( ) [static]
\end{verbatim}

Return colormap "HSV", containing 256 colors entries in RGB.

Returns

The following 256x1x1x3 colormap is returned:
8.1.4.307 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:

8.1.4.308 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:

8.1.4.309 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:

8.1.4.310 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:
8.1.4.311  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:

8.1.4.312  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:

8.1.4.313  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). |

8.1.4.314  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). |
8.1.4.315 resize() [1/3]

```cpp
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</td>
</tr>
<tr>
<td></td>
<td>boundary_conditions.</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 interpolation_type=0).</td>
</tr>
<tr>
<td><code>centering_y</code></td>
<td>Set centering type (only if interpolation_type=0).</td>
</tr>
<tr>
<td><code>centering_z</code></td>
<td>Set centering type (only if interpolation_type=0).</td>
</tr>
<tr>
<td><code>centering_c</code></td>
<td>Set centering type (only if interpolation_type=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).
8.1.4.316 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 \text{interpolation_type}=0).</td>
</tr>
<tr>
<td>centering_y</td>
<td>Set centering type (only if \text{interpolation_type}=0).</td>
</tr>
<tr>
<td>centering_z</td>
<td>Set centering type (only if \text{interpolation_type}=0).</td>
</tr>
<tr>
<td>centering_c</td>
<td>Set centering type (only if \text{interpolation_type}=0).</td>
</tr>
</tbody>
</table>

8.1.4.317 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 \text{interpolation_type}=0).</td>
</tr>
<tr>
<td>centering_y</td>
<td>Set centering type (only if \text{interpolation_type}=0).</td>
</tr>
<tr>
<td>centering_z</td>
<td>Set centering type (only if \text{interpolation_type}=0).</td>
</tr>
<tr>
<td>centering_c</td>
<td>Set centering type (only if \text{interpolation_type}=0).</td>
</tr>
</tbody>
</table>
8.1.4.318 resize_doubleXY()

\texttt{CImg\textless T\textgreater \& resize\_doubleXY()}  

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

\textbf{Note}
Use anisotropic upscaling algorithm \textit{described here}.

8.1.4.319 resize_tripleXY()

\texttt{CImg\textless T\textgreater \& resize\_tripleXY()}  

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

\textbf{Note}
Use anisotropic upscaling algorithm \textit{described here}.

8.1.4.320 mirror() [1/2]

\texttt{CImg\textless T\textgreater \& mirror(const char \texttt{axis})}  

Mirror image content along specified axis.

\textbf{Parameters}

\begin{itemize}
  \item \texttt{axis} \ Mirror axis
\end{itemize}

8.1.4.321 mirror() [2/2]

\texttt{CImg\textless T\textgreater \& mirror(const char *const \texttt{axes})}  

Mirror image content along specified axes.

\textbf{Parameters}

\begin{itemize}
  \item \texttt{axes} \ Mirror axes, as a C-string
\end{itemize}
Note

*axes* may contain multiple characters, e.g. "xyz"

### 8.1.4.322 shift()

```cpp
CImg<
```

Shift image content.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>delta_x</code></td>
<td>Amount of displacement along the X-axis.</td>
</tr>
<tr>
<td><code>delta_y</code></td>
<td>Amount of displacement along the Y-axis.</td>
</tr>
<tr>
<td><code>delta_z</code></td>
<td>Amount of displacement along the Z-axis.</td>
</tr>
<tr>
<td><code>delta_c</code></td>
<td>Amount of displacement along the C-axis.</td>
</tr>
</tbody>
</table>
| `boundary_conditions` | Boundary conditions. Can be { 0=dirichlet | 1=neumann | 2=periodic | 3=mirror }.

### 8.1.4.323 permute_axes()

```cpp
CImg<
```

Permute axes order.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>axes_order</code></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>

### 8.1.4.324 unroll()

```cpp
CImg<
```

Unroll pixel values along specified axis.
Parameters

| axis | Unroll axis (can be 'x', 'y', 'z' or 'c'). |

8.1.4.325 rotate() [1/4]

```c
CImg<T>& 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.

8.1.4.326 rotate() [2/4]

```c
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. Can be { 0=nearest</td>
</tr>
<tr>
<td>boundary_conditions</td>
<td>Boundary conditions. Can be { 0=dirichlet</td>
</tr>
</tbody>
</table>
CImg<

\[
\text{\texttt{rotate}}(u, v, w, \text{\texttt{angle}}, \text{\texttt{interpolation}}, \text{\texttt{boundary_conditions}})
\]

Rotate volumetric image with arbitrary angle and axis.

**Parameters**

| \(u\) | X-coordinate of the 3D rotation axis. |
| \(v\) | Y-coordinate of the 3D rotation axis. |
| \(w\) | Z-coordinate of the 3D rotation axis. |
| \texttt{angle} | Rotation angle, in degrees. |
| \texttt{interpolation} | Type of interpolation. Can be \{ 0=nearest | 1=linear | 2=cubic \}. |
| \texttt{boundary_conditions} | Boundary conditions. Can be \{ 0=dirichlet | 1=neumann | 2=periodic | 3=mirror \}. |

**Note**

Most of the time, size of the image is modified.

---

**Parameters**

| \(u\) | X-coordinate of the 3D rotation axis. |
| \(v\) | Y-coordinate of the 3D rotation axis. |
| \(w\) | Z-coordinate of the 3D rotation axis. |
| \texttt{angle} | Rotation angle, in degrees. |
| \texttt{interpolation} | Type of interpolation. Can be \{ 0=nearest | 1=linear | 2=cubic \}. |
| \texttt{boundary_conditions} | Boundary conditions. Can be \{ 0=dirichlet | 1=neumann | 2=periodic | 3=mirror \}. |
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</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>cz</td>
<td>Z-coordinate of the rotation center.</td>
</tr>
<tr>
<td>interpolation</td>
<td>Type of interpolation. Can be { 0=nearest | 1=linear | 2=cubic | 3=mirror }.</td>
</tr>
<tr>
<td>boundary_conditions</td>
<td>Boundary conditions. Can be { 0=dirichlet | 1=neumann | 2=periodic }.</td>
</tr>
</tbody>
</table>

Note

Most of the time, size of the image is modified.

8.1.4.329 warp()

```cpp
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 | 1=backward-relative | 2=forward-absolute | 3=foward-relative }.</td>
</tr>
<tr>
<td>interpolation</td>
<td>Can be { 0=nearest | 1=linear | 2=cubic }.</td>
</tr>
<tr>
<td>boundary_conditions</td>
<td>Boundary conditions { 0=dirichlet | 1=neumann | 2=periodic | 3=mirror }.</td>
</tr>
</tbody>
</table>

8.1.4.330 get_projections2d()

```cpp
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>

Generated by Doxygen
8.1.4.331  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.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x0</code></td>
<td>X-coordinate of the upper-left crop rectangle corner.</td>
</tr>
<tr>
<td><code>y0</code></td>
<td>Y-coordinate of the upper-left crop rectangle corner.</td>
</tr>
<tr>
<td><code>z0</code></td>
<td>Z-coordinate of the upper-left crop rectangle corner.</td>
</tr>
<tr>
<td><code>c0</code></td>
<td>C-coordinate of the upper-left crop rectangle corner.</td>
</tr>
<tr>
<td><code>x1</code></td>
<td>X-coordinate of the lower-right crop rectangle corner.</td>
</tr>
<tr>
<td><code>y1</code></td>
<td>Y-coordinate of the lower-right crop rectangle corner.</td>
</tr>
<tr>
<td><code>z1</code></td>
<td>Z-coordinate of the lower-right crop rectangle corner.</td>
</tr>
<tr>
<td><code>c1</code></td>
<td>C-coordinate of the lower-right crop rectangle corner.</td>
</tr>
<tr>
<td><code>boundary_conditions</code></td>
<td>Can be `{ 0=dirichlet</td>
</tr>
</tbody>
</table>

8.1.4.332  autocrop()

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

Autocrop image region, regarding the specified background color.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>color</code></td>
<td>Color used for the crop. If 0, color is guessed.</td>
</tr>
<tr>
<td><code>axes</code></td>
<td>Axes used for the crop.</td>
</tr>
</tbody>
</table>

8.1.4.333  get_column()

`CImg<T> get_column ( 
  const int x0 ) const`
Return specified image column.

Parameters

\[ x0 \] Image column.

---

### 8.1.4.334 \texttt{columns()}

\texttt{CImg< T >} \texttt{columns (}
\hspace{1em} \texttt{const int x0,}
\hspace{1em} \texttt{const int x1)}

Return specified range of image columns.

Parameters

\[ x0 \] Starting image column.
\[ x1 \] Ending image column.

---

### 8.1.4.335 \texttt{row()}

\texttt{CImg< T >} \texttt{row (}
\hspace{1em} \texttt{const int y0 )}

Return specified image row [in-place version].

Parameters

\[ y0 \] Image row.

---

### 8.1.4.336 \texttt{get_rows()}

\texttt{CImg< T >} \texttt{get_rows (}
\hspace{1em} \texttt{const int y0,}
\hspace{1em} \texttt{const int y1 ) const}

Return specified range of image rows.

Parameters

\[ y0 \] Starting image row.
\[ y1 \] Ending image row.
8.1.4.337  get_slice()

`CImg<T> get_slice (const int z0) const`

Return specified image slice.

Parameters

- `z0` Image slice.

8.1.4.338  get_slices()

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

Return specified range of image slices.

Parameters

- `z0` Starting image slice.
- `z1` Ending image slice.

8.1.4.339  get_channel()

`CImg<T> get_channel (const int c0) const`

Return specified image channel.

Parameters

- `c0` Image channel.

8.1.4.340  get_channels()

`CImg<T> get_channels (const int c0, const int c1) const`
8.1 CImg< T > Struct Template Reference

Return specified range of image channels.

**Parameters**

| c0 | Starting image channel. |
| c1 | Ending image channel. |

8.1.4.341 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 stream line of a 3D vector field.

**Parameters**

| func | Vector field function. |
| x    | X-coordinate of the starting point of the streamline. |
| y    | Y-coordinate of the starting point of the streamline. |
| z    | Z-coordinate of the starting point of the streamline. |
| L    | Streamline length. |
| dl   | Streamline length increment. |
| interpolation_type | Type of interpolation. Can be \{ 0=nearest int | 1=linear | 2=2nd-order RK | 3=4th-order RK. \}. |
| is_backward_tracking | Tells if the streamline is estimated forward or backward. |
| is_oriented_only | Tells if the direction of the vectors must be ignored. |
| x0   | X-coordinate of the first bounding-box vertex. |
| y0   | Y-coordinate of the first bounding-box vertex. |
| z0   | Z-coordinate of the first bounding-box vertex. |
| x1   | X-coordinate of the second bounding-box vertex. |
| y1   | Y-coordinate of the second bounding-box vertex. |
| z1   | Z-coordinate of the second bounding-box vertex. |
8.1.4.342  get_shared_points()

```cpp
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.
- `x1` X-coordinate of the ending pixel.
- `y0` Y-coordinate.
- `z0` Z-coordinate.
- `c0` C-coordinate.

8.1.4.343  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**

- `y0` Y-coordinate of the starting row.
- `y1` Y-coordinate of the ending row.
- `z0` Z-coordinate.
- `c0` C-coordinate.

8.1.4.344  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.
8.1.4.345  get_shared_slices()

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

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

Parameters

<table>
<thead>
<tr>
<th></th>
<th>Z-coordinate of the starting slice.</th>
</tr>
</thead>
<tbody>
<tr>
<td>z0</td>
<td></td>
</tr>
<tr>
<td>z1</td>
<td>Z-coordinate of the ending slice.</td>
</tr>
<tr>
<td>c0</td>
<td>C-coordinate.</td>
</tr>
</tbody>
</table>

8.1.4.346  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></th>
<th>Z-coordinate.</th>
</tr>
</thead>
<tbody>
<tr>
<td>z0</td>
<td></td>
</tr>
<tr>
<td>c0</td>
<td>C-coordinate.</td>
</tr>
</tbody>
</table>

8.1.4.347  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

| c0 | C-coordinate of the starting channel. |
| c1 | C-coordinate of the ending channel. |

8.1.4.348  

`get_shared_channel()`

```cpp
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. |

8.1.4.349  

`get_split()` [1/2]

```cpp
CImgList< T > get_split ( const char axis, const int nb = -1 )
```

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.

8.1.4.350  

`get_split()` [2/2]

```cpp
CImgList< T > get_split ( const CImg< t >& values, const char axis = 0, const bool keep_values = true )
```

Split image into a list of sub-images, according to a specified splitting value sequence and optionally axis.
Parameters

<table>
<thead>
<tr>
<th>values</th>
<th>Splitting value sequence.</th>
</tr>
</thead>
<tbody>
<tr>
<td>axis</td>
<td>Axis along which the splitting is performed. Can be '0' to ignore axis.</td>
</tr>
<tr>
<td>keep_values</td>
<td>Tells if the splitting sequence must be kept in the split blocs.</td>
</tr>
</tbody>
</table>

### 8.1.4.351 append()

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

Append two images along specified axis.

Parameters

<table>
<thead>
<tr>
<th>img</th>
<th>Image to append with instance image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>axis</td>
<td>Appending axis. Can be ( 'x'</td>
</tr>
<tr>
<td>align</td>
<td>Append alignment in [0,1].</td>
</tr>
</tbody>
</table>

### 8.1.4.352 correlate()

CImg< T > & correlate (  
    const CImg< t > & kernel,  
    const unsigned int boundary_conditions = 1,  
    const bool is_normalized = false,  
    const unsigned int channel_mode = 1,  
    const 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>kernel</td>
<td>= the correlation kernel.</td>
</tr>
</tbody>
</table>
| boundary_conditions | Boundary condition. Can be { 0=dirichlet | 1=neumann | 2=periodic | 3=mirror }.
| is_normalized    | = enable local normalization.                                              |
| channel          | mode Channel processing mode. Can be { 0=sum inputs | 1=one-for-one | 2=expand }       |
| xcenter          | X-coordinate of the kernel center (~0U means 'centered').                  |
| xstart           | Starting X-coordinate of the instance image.                               |
| xend             | Ending X-coordinate of the instance image.                                 |
| xstride          | Stride along the X-axis.                                                   |
| xdilation        | Dilation along the X-axis.                                                 |
| ycenter          | Y-coordinate of the kernel center (~0U means 'centered').                  |
| ystart           | Starting Y-coordinate of the instance image.                               |
| yend             | Ending Y-coordinate of the instance image.                                 |
| ystride          | Stride along the Y-axis.                                                   |
| ydilation        | Dilation along the Y-axis.                                                 |
| zcenter          | Z-coordinate of the kernel center (~0U means 'centered').                  |
| zstart           | Starting Z-coordinate of the instance image.                               |
| zend             | Ending Z-coordinate of the instance image.                                 |
| zstride          | Stride along the Z-axis.                                                   |
| zdilation        | Dilation along the Z-axis.                                                 |

Note

- The correlation of the image instance *this by the kernel kernel is defined to be: \( \text{res}(x,y,z) = \sum_{i,j,k} (*this)(x + (i - c_x), y + (j - c_y), z + (k - c_z)) \times \text{kernel}(i,j,k) \).

8.1.4.353 convolve()

```cpp
CImg<T>& convolve(
    const CImg<T>& kernel,
    const unsigned int boundary_conditions = 1,
    const bool is_normalized = false,
    const unsigned int channel_mode = 1,
    const 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
)
```

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 (~0U 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>
<tr>
<td>xdilation</td>
<td>Dilation along the X-axis.</td>
</tr>
<tr>
<td>ycenter</td>
<td>Y-coordinate of the kernel center (~0U 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 (~0U 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 kernel is defined to be:
  \[ \text{res}(x,y,z) = \sum_{i,j,k} (\ast\text{this})(x - (i - c_x),y - (j - c_y),z - (k - c_z)) \ast\text{kernel}(i,j,k). \]

8.1.4.354 cumulate() [1/2]

CImg<T>& cumulate (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>

8.1.4.355 cumulate() [2/2]

CImg<T>& cumulate (const char *const axes )
Cumulate image values, along specified axes.
Parameters

| axes       | Cumulation axes, as a C-string. |

Note

axes may contain multiple characters, e.g. "xyz"

8.1.4.356 erode() [1/3]

```cpp
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>

8.1.4.357 erode() [2/3]

```cpp
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>

8.1.4.358 erode() [3/3]

```cpp
CImg<T> & erode {
    const unsigned int s
}
```

Generated by Doxygen
Erode the image by a square structuring element of specified size.

Parameters

- **s**: Size of the structuring element.

8.1.4.359  dilate() [1/3]

```cpp
CImg<
```

Dilate image by a structuring element.

Parameters

- **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).

8.1.4.360  dilate() [2/3]

```cpp
CImg<
```

Dilate image by a rectangular structuring element of specified size.

Parameters

- **sx**: Width of the structuring element.
- **sy**: Height of the structuring element.
- **sz**: Depth of the structuring element.

8.1.4.361  dilate() [3/3]

```cpp
CImg<
```

Dilate image by a square structuring element of specified size.
Parameters

\( s \) | Size of the structuring element.

---

8.1.4.362 watershed()

\[
\text{CImg<}T\text{>\& watershed (}
\text{const CImg<} T \text{> \& priority,}
\text{const bool is_high_connectivity = false )}
\]

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.

---

8.1.4.363 deriche()

\[
\text{CImg<}T\text{>\& deriche (}
\text{const float sigma,}
\text{const unsigned int order = 0,}
\text{const char axis = 'x',}
\text{const bool boundary_conditions = true )}
\]

Apply recursive Deriche filter.

Parameters

| sigma | Standard deviation of the filter. |
| order | Order of the filter. Can be \{ 0=smooth-filter | 1=1st-derivative | 2=2nd-derivative \}. |
| axis | Axis along which the filter is computed. Can be \{ 'x' | 'y' | 'z' | 'c' \}. |
| boundary_conditions | Boundary conditions. Can be \{ 0=dirichlet | 1=neumann \}. |
8.1.4.364  vanvliet()

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>Parameter</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', 'y', 'z', 'c'}.</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)


8.1.4.365  blur() [1/2]

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>Parameter</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()`.

### 8.1.4.366 blur() [2/2]

```cpp
CImg<T> & blur (
    const float sigma,
    const bool boundary_conditions = true,
    const bool is_gaussian = false
)
```

Blur image isotropically.

**Parameters**

<table>
<thead>
<tr>
<th><strong>sigma</strong></th>
<th>Standard deviation of the blur.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boundary_conditions</strong></td>
<td>Boundary conditions. Can be `{ 0=dirichlet</td>
</tr>
<tr>
<td><strong>is_gaussian</strong></td>
<td>Use a gaussian kernel (VanVliet) is set, a pseudo-gaussian (Deriche) otherwise.</td>
</tr>
</tbody>
</table>

See also

`deriche()`, `vanvliet()`.

### 8.1.4.367 blur_anisotropic() [1/2]

```cpp
CImg<T> & blur_anisotropic (
    const CImg< & G,  
    const float amplitude = 60,  
    const float dl = 0.8f,  
    const float ds = 30,  
    const float gauss_prec = 2,  
    const unsigned int interpolation_type = 0,  
    const bool is_fast_approx = 1
)
```

Blur image anisotropically, directed by a field of diffusion tensors.

**Parameters**

<table>
<thead>
<tr>
<th><strong>G</strong></th>
<th>Field of square roots of diffusion tensors/vectors used to drive the smoothing.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>amplitude</strong></td>
<td>Amplitude of the smoothing.</td>
</tr>
<tr>
<td><strong>dl</strong></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\text{_prec})</td>
<td>Precision of the diffusion process.</td>
</tr>
<tr>
<td>(interpolation_type)</td>
<td>Interpolation scheme. Can be (0=\text{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>

### 8.1.4.368 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\text{_prec})</td>
<td>Precision of the diffusion process.</td>
</tr>
<tr>
<td>(interpolation_type)</td>
<td>Interpolation scheme. Can be (0=\text{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>

### 8.1.4.369 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
const float sigma_r,
const float sampling_x,
const float sampling_y,
const float sampling_z,
const float sampling_r
)

Blur image, with the joint bilateral filter.

Parameters

<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>sigma_r</td>
<td>Amount of blur along the value 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 ECCV'2006 (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)

8.1.4.370 blur_bilateral() [2/2]

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>
### boxfilter()

```cpp
CImg<T>& boxfilter(
    const float boxsize,
    const int order,
    const char axis = 'x',
    const bool boundary_conditions = true,
    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', 'y', 'z', 'c'}.</td>
</tr>
<tr>
<td>boundary_conditions</td>
<td>Boundary conditions. Can be {0=dirichlet, 1=neumann}.</td>
</tr>
<tr>
<td>nb_iter</td>
<td>Number of filter iterations.</td>
</tr>
</tbody>
</table>

### blur_box() [1/2]

```cpp
CImg<T>& 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.

**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, true=neumann}.</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()`
8.1.4.373 blur_box()

CImg<T>& blur_box (  
    const float boxsize,  
    const bool boundary_conditions = true )

Blur image with a box filter.

Parameters

<table>
<thead>
<tr>
<th>boxsize</th>
<th>Size of the box window (can be subpixel).</th>
</tr>
</thead>
<tbody>
<tr>
<td>boundary_conditions</td>
<td>Boundary conditions. Can be { 0=dirichlet</td>
</tr>
</tbody>
</table>

See also
deriche(), vanvliet().

8.1.4.374 blur_guided()

CImg<T>& blur_guided (  
    const CImg<t>& guide,  
    const float radius,  
    const float regularization )

Blur image, with the image guided filter.

Parameters

<table>
<thead>
<tr>
<th>guide</th>
<th>Image used to guide the smoothing process.</th>
</tr>
</thead>
<tbody>
<tr>
<td>radius</td>
<td>Spatial radius. If negative, it is expressed as a percentage of the largest image size.</td>
</tr>
<tr>
<td>regularization</td>
<td>Regularization parameter. If negative, it is expressed as a percentage of the guide value range.</td>
</tr>
</tbody>
</table>

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

8.1.4.375 blur_patch()

CImg<T>& blur_patch (  
    const CImg<t>& guide,  
    const float sigma_s,  
    const float sigma_r,  
    const unsigned int patch_size = 3,
    const bool boundary_conditions = true )

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>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>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>

8.1.4.376 blur_median()

`CImg<T>@ blur_median ( 
    const unsigned int n,
    const float threshold = 0 )`

Blur image with the median filter.

**Parameters**

<table>
<thead>
<tr>
<th>n</th>
<th>Size of the median filter.</th>
</tr>
</thead>
<tbody>
<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>

8.1.4.377 sharpen()

`CImg<T>@ sharpen ( 
    const float amplitude,
    const bool sharpen_type = false,
    const float edge = 1,
    const float alpha = 0,
    const float sigma = 0 )`

Sharpen image.

**Parameters**

<table>
<thead>
<tr>
<th>amplitude</th>
<th>Sharpening amplitude</th>
</tr>
</thead>
<tbody>
<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>
8.1.4.378  get_gradient()

\texttt{CImgList<Tfloat> get\_gradient (}
  \texttt{const char *const axes = 0,}
  \texttt{const int scheme = 0 ) const}

Return image gradient.

Parameters

\begin{tabular}{|c|l|}
\hline
\textit{axes} & Axes considered for the gradient computation, as a \texttt{C-string (e.g "xy")}. \\
\textit{scheme} & \texttt{= Numerical scheme used for the gradient computation:} \\
& \begin{itemize}
  \item \texttt{-1 = Backward finite differences}
  \item \texttt{0 = Centered finite differences (default)}
  \item \texttt{1 = Forward finite differences}
  \item \texttt{2 = Using Sobel kernels}
  \item \texttt{3 = Using rotation invariant kernels}
  \item \texttt{4 = Using Deriche recursive filter.}
  \item \texttt{5 = Using Van Vliet recursive filter.}
\end{itemize} \\
\end{tabular}

8.1.4.379  get_hessian()

\texttt{CImgList<Tfloat> get\_hessian (}
  \texttt{const char *const axes = 0 ) const}

Return image hessian.

Parameters

\begin{tabular}{|c|l|}
\hline
\textit{axes} & Axes considered for the hessian computation, as a \texttt{C-string (e.g "xy")}. \\
\end{tabular}

8.1.4.380  structure_tensors()

\texttt{CImg<T> & structure\_tensors (}
  \texttt{const bool is\_fwbw\_scheme = false )}

Compute the structure tensor field of an image.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>is_fwbw_scheme</code></td>
<td>Scheme. Can be `{ false=centered</td>
</tr>
</tbody>
</table>

### 8.1.4.381 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sharpness</code></td>
<td>Sharpness</td>
</tr>
<tr>
<td><code>anisotropy</code></td>
<td>Anisotropy</td>
</tr>
<tr>
<td><code>alpha</code></td>
<td>Standard deviation of the gradient blur.</td>
</tr>
<tr>
<td><code>sigma</code></td>
<td>Standard deviation of the structure tensor blur.</td>
</tr>
<tr>
<td><code>is_sqrt</code></td>
<td>Tells if the square root of the tensor field is computed instead.</td>
</tr>
</tbody>
</table>

### 8.1.4.382 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<float,T>& guide = CImg<float,T>::const_empty()
}
```

Estimate displacement field between two images.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>source</code></td>
<td>Reference image.</td>
</tr>
<tr>
<td><code>smoothness</code></td>
<td>Smoothness of estimated displacement field.</td>
</tr>
<tr>
<td><code>precision</code></td>
<td>Precision required for algorithm convergence.</td>
</tr>
<tr>
<td><code>nb_scales</code></td>
<td>Number of scales used to estimate the displacement field.</td>
</tr>
<tr>
<td><code>iteration_max</code></td>
<td>Maximum number of iterations allowed for one scale.</td>
</tr>
<tr>
<td><code>is_backward</code></td>
<td>If false, match I2(X + U(X)) = I1(X), else match I2(X) = I1(X - U(X)).</td>
</tr>
<tr>
<td><code>guide</code></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>
</tbody>
</table>
8.1.4.383  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 patch_penalization,
    const CImg<t1> & 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>patch_penalization</td>
<td>Penalization factor in score related patch occurrences. If negative, also tells that identity result is not avoided.</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>out matching_score</td>
<td>Returned as the image of matching scores.</td>
</tr>
</tbody>
</table>

8.1.4.384  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.

### 8.1.4.385 distance()

```cpp
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.

### 8.1.4.386 distance_dijkstra()

```cpp
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>out return_path</td>
<td>An image containing the nodes of the minimal path.</td>
</tr>
</tbody>
</table>
8.1.4.387  distance_eikonal() [1/2]

```cpp
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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Reference value.</td>
</tr>
<tr>
<td>metric</td>
<td>Field of distance potentials.</td>
</tr>
</tbody>
</table>

8.1.4.388  distance_eikonal() [2/2]

```cpp
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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nb_iterations</td>
<td>Number of PDE iterations.</td>
</tr>
<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>

8.1.4.389  haar() [1/2]

```cpp
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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axis</td>
<td>Axis considered for the transform.</td>
</tr>
<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>
8.1.4.390 haar() [2/2]

```cpp
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><code>invert</code></td>
<td>Set inverse of direct transform.</td>
</tr>
<tr>
<td><code>nb_scales</code></td>
<td>Number of scales used for the transform.</td>
</tr>
</tbody>
</table>

8.1.4.391 get_FFT()

```cpp
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><code>axis</code></td>
<td>Axis along which the FFT is computed.</td>
</tr>
<tr>
<td><code>is_inverse</code></td>
<td>Tells if the forward (false) or inverse (true) FFT is computed.</td>
</tr>
</tbody>
</table>

8.1.4.392 FFT() [1/2]

```cpp
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><code>real</code></td>
<td>Real part of the pixel values.</td>
</tr>
<tr>
<td><code>imag</code></td>
<td>Imaginary part of the pixel values.</td>
</tr>
<tr>
<td><code>axis</code></td>
<td>Axis along which the FFT is computed.</td>
</tr>
<tr>
<td><code>is_inverse</code></td>
<td>Tells if the forward (false) or inverse (true) FFT is computed.</td>
</tr>
</tbody>
</table>
8.1.4.393 FFT() [2/2]

```c
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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in,out</code></td>
<td><code>real</code></td>
</tr>
<tr>
<td><code>in,out</code></td>
<td><code>imag</code></td>
</tr>
<tr>
<td></td>
<td><code>is_inverse</code></td>
</tr>
<tr>
<td></td>
<td><code>nb_threads</code></td>
</tr>
</tbody>
</table>

8.1.4.394 shift_object3d() [1/2]

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

Shift 3D object's vertices.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tx</code></td>
<td>X-coordinate of the 3D displacement vector.</td>
</tr>
<tr>
<td><code>ty</code></td>
<td>Y-coordinate of the 3D displacement vector.</td>
</tr>
<tr>
<td><code>tz</code></td>
<td>Z-coordinate of the 3D displacement vector.</td>
</tr>
</tbody>
</table>

8.1.4.395 shift_object3d() [2/2]

```c
CImg< T >& shift_object3d() 
```

Shift 3D object's vertices, so that it becomes centered.

**Note**

The object center is computed as its barycenter.
8.1.4.396 resize_object3d()

```cpp
CImg<T>& resize_object3d (  
    const float sx,  
    const float sy = -100,  
    const float sz = -100  
)
```

Resize 3D object.

Parameters

| sx | Width of the 3D object's bounding box. |
| sy | Height of the 3D object's bounding box. |
| sz | Depth of the 3D object's bounding box. |

8.1.4.397 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 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>
</tr>
<tr>
<td>obj_primitives</td>
<td>Primitives data of the additional 3D object.</td>
</tr>
</tbody>
</table>

8.1.4.398 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 primitives</th>
<th>Primitives data of the 3D object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out colors</td>
<td>Colors data of the 3D object.</td>
</tr>
<tr>
<td>texture</td>
<td>Texture image to map to 3D object.</td>
</tr>
<tr>
<td>coords</td>
<td>Texture-mapping coordinates.</td>
</tr>
</tbody>
</table>
8.1.4.399 get_elevation3d()

\texttt{CImg\langle floatT\rangle\ get\_elevation3d (~}\begin{verbatim}
CImgList\langle tf \rangle \& primitives,
CImgList\langle tc \rangle \& colors,
const CImg\langle te \rangle \& elevation) const
\end{verbatim}\texttt{)}

Generate a 3D elevation of the image instance.

\textbf{Parameters}

\begin{tabular}{|c|c|}
\hline
\texttt{out primitives} & The returned list of the 3D object primitives (template type \texttt{tf} should be at least \texttt{unsigned int}). \\
\texttt{out colors} & The returned list of the 3D object colors. \\
\texttt{elevation} & The input elevation map. \\
\hline
\end{tabular}

\textbf{Returns}

The \(N\) vertices \((x_i,y_i,z_i)\) of the 3D object as a \(N\times3\) \texttt{CImg\langle float\rangle} image \((0 <= i <= N - 1)\).

\textbf{Example}

\begin{verbatim}
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);
\end{verbatim}

8.1.4.400 get_projections3d()

\texttt{CImg\langle floatT\rangle\ get\_projections3d (~}\begin{verbatim}
CImgList\langle tf \rangle \& primitives,
CImgList\langle tc \rangle \& colors,
const unsigned int x0,
const unsigned int y0,
const unsigned int z0,
const bool normalize_colors = false) const
\end{verbatim}\texttt{)}

Generate the 3D projection planes of the image instance.

\textbf{Parameters}

\begin{tabular}{|c|c|}
\hline
\texttt{out primitives} & Primitives data of the returned 3D object. \\
\texttt{out colors} & Colors data of the returned 3D object. \\
x0 & X-coordinate of the projection point. \\
y0 & Y-coordinate of the projection point. \\
z0 & Z-coordinate of the projection point. \\
\texttt{normalize_colors} & Tells if the created textures have normalized colors. \\
\hline
\end{tabular}
8.1.4.401  get_isoline3d()

\[
\text{CImg<}\text{floatT} \geq \text{get_isoline3d} ( \\
\hspace{1em} \text{CImgList<} \text{tf} > & \text{primitives}, \\
\hspace{2em} \text{const float isovalue}, \\
\hspace{3em} \text{const int size_x = -100,} \\
\hspace{4em} \text{const int size_y = -100} ) \text{ const}
\]

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 \text{tf} should be at least \text{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 \((x_i, y_i, z_i)\) of the 3D object as a \( N \times 3 \) \text{CImg<}\text{float} \text{ image} \((0 \leq i \leq N - 1)\).

Example

\[
\begin{align*}
\text{const CImg<}\text{float> img("reference.jpg");} \\
\text{CImgList<}\text{unsigned int} > \text{faces3d;} \\
\text{const CImg<}\text{float> points3d = img.get_isoline3d(faces3d,100);} \\
\text{CImg<}\text{unsigned char} >().\text{display_object3d("Isoline3d",points3d,faces3d,colors3d);} \\
\end{align*}
\]

8.1.4.402  get_isosurface3d()

\[
\text{CImg<}\text{floatT} \geq \text{get_isosurface3d} ( \\
\hspace{1em} \text{CImgList<} \text{tf} > & \text{primitives}, \\
\hspace{2em} \text{const float isovalue}, \\
\hspace{3em} \text{const int size_x = -100,} \\
\hspace{4em} \text{const int size_y = -100,} \\
\hspace{5em} \text{const int size_z = -100} ) \text{ const}
\]

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 \text{tf} should be at least \text{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 \((x_i, y_i, z_i)\) of the 3D object as a \(N \times 3\) \(\text{CImg}<\text{float}>\) image (0\(\leq i \leq N - 1\)).

Example

```cpp
const \text{CImg}<\text{float}> \text{img} = \text{CImg}<\text{unsigned char}>("reference.jpg").\text{resize}(-100,-100,20);
\text{CImgList<\text{unsigned int}> faces3d;}
const \text{CImg}<\text{float}> \text{points3d} = \text{img}.\text{get_isosurface3d}(\text{faces3d},100);
\text{CImg<\text{unsigned char}>().display_object3d("Isosurface3d",\text{points3d},\text{faces3d},\text{colors3d});}
```

8.1.4.403 elevation3d()

```cpp
\text{static \text{CImg}<\text{float}> elevation3d (}
\text{\text{CImgList<\text{tf}> & primitives,}}
\text{const \text{tfunc} & \text{func,}}
\text{const \text{float} x0,}
\text{const \text{float} y0,}
\text{const \text{float} x1,}
\text{const \text{float} y1,}
\text{const \text{int} size_x = 256,}
\text{const \text{int} size_y = 256 \} \text{[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>

8.1.4.404 isoline3d()

```cpp
\text{static \text{CImg}<\text{float}> isoline3d (}
\text{\text{CImgList<\text{tf}> & primitives,}}
\text{const \text{tfunc} & \text{func,}}
\text{const \text{float} isovalue,}
\text{const \text{float} x0,}
\text{const \text{float} y0,}
\text{const \text{float} x1,}
\text{const \text{float} y1,}
\text{const \text{int} size_x = 256,}
\text{const \text{int} size_y = 256 \} \text{[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 float (*func)(const float x, const float y).</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.

### 8.1.4.405 isosurface3d()

```cpp
static CImg<floatT> isosurface3d ( 
    CImgList< tf > & primitives, 
    const tf & 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 float (*func)(const float x, const float y, const float z).</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>
Note

Use the marching cubes algorithm for extracting the isosurface.

8.1.4.406 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></td>
<td>The width of the box (dimension along the X-axis).</td>
</tr>
<tr>
<td>size_y</td>
<td></td>
<td>The height of the box (dimension along the Y-axis).</td>
</tr>
<tr>
<td>size_z</td>
<td></td>
<td>The depth of the box (dimension 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
CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::box3d(faces3d,10,20,30);
CImg<unsigned char>().display_object3d("Box3d",points3d,faces3d);
```

8.1.4.407 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></td>
<td>The radius of the cone basis.</td>
</tr>
<tr>
<td>size_z</td>
<td></td>
<td>The cone's height.</td>
</tr>
<tr>
<td>subdivisions</td>
<td></td>
<td>The number of basis angular subdivisions.</td>
</tr>
</tbody>
</table>

Generated by Doxygen
Returns

The N vertices \((x_i, y_i, z_i)\) of the 3D object as a \(N\times3\) \(\text{CImg}<\text{float}>\) image \((0 \leq i \leq N - 1)\).

Example

```cpp
CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::cone3d(faces3d, 50);
CImg<unsigned char>().display_object3d("Cone3d", points3d, faces3d);
```

8.1.4.408  cylinder3d()

```cpp
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</th>
<th>primitives</th>
<th>The returned list of the 3D object primitives (template type (tf) should be at least \text{unsigned int}).</th>
</tr>
</thead>
<tbody>
<tr>
<td>radius</td>
<td>The radius of the cylinder basis.</td>
<td></td>
</tr>
<tr>
<td>size_z</td>
<td>The cylinder's height.</td>
<td></td>
</tr>
<tr>
<td>subdivisions</td>
<td>The number of basis angular subdivisions.</td>
<td></td>
</tr>
</tbody>
</table>

Returns

The N vertices \((x_i, y_i, z_i)\) of the 3D object as a \(N\times3\) \(\text{CImg}<\text{float}>\) image \((0 \leq i \leq N - 1)\).

Example

```cpp
CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::cylinder3d(faces3d, 50);
CImg<unsigned char>().display_object3d("Cylinder3d", points3d, faces3d);
```

8.1.4.409  torus3d()

```cpp
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 int).</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>radius1</td>
<td>The large radius.</td>
</tr>
<tr>
<td></td>
<td>radius2</td>
<td>The small radius.</td>
</tr>
<tr>
<td></td>
<td>subdivisions1</td>
<td>The number of angular subdivisions for the large radius.</td>
</tr>
<tr>
<td></td>
<td>subdivisions2</td>
<td>The number of angular subdivisions for the small radius.</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>::torus3d(faces3d, 20, 4);
CImg<unsigned char>().display_object3d("Torus3d", points3d, faces3d);
```

8.1.4.410 plane3d()

```cpp
static CImg< float > 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 int).</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>size_x</td>
<td>The width of the plane (dimension along the X-axis).</td>
</tr>
<tr>
<td></td>
<td>size_y</td>
<td>The height of the plane (dimensions along the Y-axis).</td>
</tr>
<tr>
<td></td>
<td>subdivisions_x</td>
<td>The number of planar subdivisions along the X-axis.</td>
</tr>
<tr>
<td></td>
<td>subdivisions_y</td>
<td>The number of planar 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
CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::plane3d(faces3d, 100, 50);
CImg<unsigned char>().display_object3d("Plane3d", points3d, faces3d);
```
8.1.4.111 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></td>
<td>radius</td>
<td>The radius of the sphere (dimension along the X-axis).</td>
</tr>
<tr>
<td></td>
<td>subdivisions</td>
<td>The number of recursive subdivisions from an initial icosahedron.</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>::sphere3d(faces3d,100,4);  
CImg<unsigned char>().display_object3d("Sphere3d",points3d,faces3d);
```

8.1.4.112 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></td>
<td>tensor</td>
<td>The tensor which gives the shape and size of the ellipsoid.</td>
</tr>
<tr>
<td></td>
<td>subdivisions</td>
<td>The number of recursive subdivisions from an initial stretched icosahedron.</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> tensor = CImg<float>::diagonal(10,7,3);  
points3d = CImg<float>::ellipsoid3d(faces3d,tensor,4);  
CImg<unsigned char>().display_object3d("Ellipsoid3d",points3d,faces3d);
```
8.1.4.413  object3dtoCImg3d()

\begin{verbatim}
CImg<T>& object3dtoCImg3d {
    const CImgList<tp> & primitives,
    const CImgList<tc> & colors,
    const to & opacities,
    const bool full_check = true
}
\end{verbatim}

Convert 3D object into a CImg3d representation.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{primitives}</td>
<td>Primitives data of the 3D object.</td>
</tr>
<tr>
<td>\textit{colors}</td>
<td>Colors data of the 3D object.</td>
</tr>
<tr>
<td>\textit{opacities}</td>
<td>Opacities data of the 3D object.</td>
</tr>
<tr>
<td>\textit{full_check}</td>
<td>Tells if full checking of the 3D object must be performed.</td>
</tr>
</tbody>
</table>

8.1.4.414  CImg3dtoobject3d()

\begin{verbatim}
CImg<T>& CImg3dtoobject3d {
    CImgList<tp> & primitives,
    CImgList<tc> & colors,
    CImgList<to> & opacities,
    const bool full_check = true
}
\end{verbatim}

Convert CImg3d representation into a 3D object.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{primitives}</td>
<td>Primitives data of the 3D object.</td>
</tr>
<tr>
<td>\textit{colors}</td>
<td>Colors data of the 3D object.</td>
</tr>
<tr>
<td>\textit{opacities}</td>
<td>Opacities data of the 3D object.</td>
</tr>
<tr>
<td>\textit{full_check}</td>
<td>Tells if full checking of the 3D object must be performed.</td>
</tr>
</tbody>
</table>

8.1.4.415  draw_point()

\begin{verbatim}
CImg<T>& draw_point {
    const int x0,
    const int y0,
    const int z0,
    const tc *const color,
    const float opacity = 1
}
\end{verbatim}

Draw a 3D point.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_0$</td>
<td>X-coordinate of the point.</td>
</tr>
<tr>
<td>$y_0$</td>
<td>Y-coordinate of the point.</td>
</tr>
<tr>
<td>$z_0$</td>
<td>Z-coordinate of the point.</td>
</tr>
<tr>
<td>$\text{color}$</td>
<td>Pointer to $\text{spectrum()}$ consecutive values, defining the drawing color.</td>
</tr>
<tr>
<td>$\text{opacity}$</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

Note

- To set pixel values without clipping needs, you should use the faster $\text{CImg::operator()}$ function.

Example:

```cpp
CImg<unsigned char> img(100,100,1,3,0);
const unsigned char color[] = { 255,128,64 };
img.draw_point(50,50,color);
```

8.1.4.416 draw_point() [2/2]

```cpp
CImg<T> & draw_point {
    const CImg<T> & points,
    const tc *const color,
    const float opacity = 1
}
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{points}$</td>
<td>Image of vertices coordinates.</td>
</tr>
<tr>
<td>$\text{color}$</td>
<td>Pointer to $\text{spectrum()}$ consecutive values, defining the drawing color.</td>
</tr>
<tr>
<td>$\text{opacity}$</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

8.1.4.417 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);
```

8.1.4.418 `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>
**draw_line()**

```cpp
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);
```
Draw a textured 2D line, with perspective correction.

<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><code>texture</code></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><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>

8.1.421 `draw_line()` [5/6]

```cpp
cImg< T > & draw_line (  
    CImg< tz > & zbuffer,  
    int \( x0 \),  
    int \( y0 \),  
    const float \( z0 \),  
    int \( x1 \),  
    int \( y1 \),  
    const float \( z1 \),  
    const CImg< tc > & texture,  
    const int \( tx0 \),  
    const int \( ty0 \),  
    const int \( tx1 \),  
    const int \( ty1 \),  
    const float `opacity` = 1,  
    const unsigned int `pattern` = ~0U,  
    const bool `init_hatch` = true )
```
Draw a textured 2D line, with perspective correction and z-buffering.
8.1 CImg< T > Struct Template Reference

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 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>

8.1.4.422 draw_line() [6/6]

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>points</td>
<td>Coordinates of vertices, stored as a list of vectors.</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 line pattern.</td>
</tr>
<tr>
<td>init_hatch</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.

8.1.4.423 draw_arrow()

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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>X-coordinate of the starting arrow point (tail).</td>
</tr>
<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 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>

8.1.4.424 draw_spline()

CImg<T> & draw_spline(
    const int x0,
    const int y0,
    const float u0,
    const float v0,
    const int x1,
    const int y1,
    const float u1,
    const float v1,
    const tc *const color,
    const float opacity = 1,
    const float precision = 0.25,
    const unsigned int pattern = ~0U,
    const bool init_hatch = true)

Draw a 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>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>color</td>
<td>Pointer to <code>spectrum()</code> consecutive values of type T, defining the drawing color.</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, 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 \( u_0 = (x_a - x_0), v_0 = (y_a - y_0), u_1 = (x_1 - x_b) \) 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);
```

8.1.4.425 `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 ul, 
    const float vl, 
    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><code>x0</code></td>
<td>X-coordinate of the starting curve point</td>
</tr>
<tr>
<td><code>y0</code></td>
<td>Y-coordinate of the starting curve point</td>
</tr>
<tr>
<td><code>u0</code></td>
<td>X-coordinate of the starting velocity</td>
</tr>
<tr>
<td><code>v0</code></td>
<td>Y-coordinate of the starting velocity</td>
</tr>
<tr>
<td><code>x1</code></td>
<td>X-coordinate of the ending curve point</td>
</tr>
<tr>
<td><code>y1</code></td>
<td>Y-coordinate of the ending curve point</td>
</tr>
<tr>
<td><code>u1</code></td>
<td>X-coordinate of the ending velocity</td>
</tr>
<tr>
<td><code>v1</code></td>
<td>Y-coordinate of the ending velocity</td>
</tr>
<tr>
<td><code>texture</code></td>
<td>Texture image defining line 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>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 true, reinit hatch motif.</td>
</tr>
</tbody>
</table>

### 8.1.4.426 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><code>points</code></td>
<td>Vertices data.</td>
</tr>
<tr>
<td><code>tangents</code></td>
<td>Tangents data.</td>
</tr>
<tr>
<td><code>color</code></td>
<td>Pointer to <code>spectrum()</code> consecutive values of type <code>T</code>, defining the drawing color.</td>
</tr>
<tr>
<td><code>opacity</code></td>
<td>Drawing opacity.</td>
</tr>
<tr>
<td><code>is_closed_set</code></td>
<td>Tells if the drawn spline set is closed.</td>
</tr>
<tr>
<td><code>precision</code></td>
<td>Precision of the drawing.</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 true, init hatch motif.</td>
</tr>
</tbody>
</table>
8.1.4.427 draw_spline() [4/4]

\[
\text{CImg<T>& draw_spline (}
\begin{align*}
&\text{const CImg<T>& points,} \\
&\text{const tc*const color,} \\
&\text{const float opacity = 1,} \\
&\text{const bool is_closed_set = \text{false,}} \\
&\text{const float precision = 4,} \\
&\text{const unsigned int pattern = \text{~0U,}} \\
&\text{const bool init_hatch = \text{true})}
\end{align*}
\]

Draw a set of consecutive splines [overloading].

Similar to previous function, with the point tangents automatically estimated from the given points set.

8.1.4.428 draw_triangle() [1/9]

\[
\text{CImg<T>& draw_triangle (}
\begin{align*}
&\text{const int x0,} \\
&\text{const int y0,} \\
&\text{const int x1,} \\
&\text{const int y1,} \\
&\text{const int x2,} \\
&\text{const int y2,} \\
&\text{const tc*const color,} \\
&\text{const float opacity = 1})
\end{align*}
\]

Draw a filled 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 \texttt{spectrum()} consecutive values of type (T), defining the drawing color.</td>
</tr>
<tr>
<td>(opacity)</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

8.1.4.429 draw_triangle() [2/9]

\[
\text{CImg<T>& draw_triangle (}
\begin{align*}
&\text{const int x0,} \\
&\text{const int y0,} \\
&\text{const int x1,} \\
&\text{const int y1,} \\
&\text{const int x2,} \\
&\text{const int y2,}
\end{align*}
\]
```
const tc *const color,
const float opacity,
const unsigned int pattern )
```

Draw a outlined 2D triangle.

**Parameters**

<table>
<thead>
<tr>
<th>x0</th>
<th>X-coordinate of the first vertex.</th>
</tr>
</thead>
<tbody>
<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 <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 outline pattern.</td>
</tr>
</tbody>
</table>

CImg\langle T \rangle$\&$ draw\_triangle ( 
    CImg\langle tz \rangle$\&$ 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>zbuffer</th>
<th>Z-buffer image.</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>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 <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>brightness</td>
<td>Brightness factor.</td>
</tr>
</tbody>
</table>
8.1.4.431  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>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>bs0</td>
<td>Brightness factor of the first vertex (in [0,2]).</td>
</tr>
<tr>
<td>bs1</td>
<td>Brightness factor of the second vertex (in [0,2]).</td>
</tr>
<tr>
<td>bs2</td>
<td>Brightness factor of the third vertex (in [0,2]).</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

8.1.4.432  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.
Parameters

<table>
<thead>
<tr>
<th>Variable</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>

8.1.4.433 `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>Variable</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>
8.1.4.434 draw_triangle() [7/9]

\begin{verbatim}
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 )
\end{verbatim}

Draw a Phong-shaded 2D triangle.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{x0}</td>
<td>X-coordinate of the first vertex in the image instance.</td>
</tr>
<tr>
<td>\textit{y0}</td>
<td>Y-coordinate of the first vertex in the image instance.</td>
</tr>
<tr>
<td>\textit{x1}</td>
<td>X-coordinate of the second vertex in the image instance.</td>
</tr>
<tr>
<td>\textit{y1}</td>
<td>Y-coordinate of the second vertex in the image instance.</td>
</tr>
<tr>
<td>\textit{x2}</td>
<td>X-coordinate of the third vertex in the image instance.</td>
</tr>
<tr>
<td>\textit{y2}</td>
<td>Y-coordinate of the third vertex in the image instance.</td>
</tr>
<tr>
<td>\textit{color}</td>
<td>Pointer to \textit{spectrum()} consecutive values, defining the drawing color.</td>
</tr>
<tr>
<td>\textit{light}</td>
<td>Light image.</td>
</tr>
<tr>
<td>\textit{lx0}</td>
<td>X-coordinate of the first vertex in the light image.</td>
</tr>
<tr>
<td>\textit{ly0}</td>
<td>Y-coordinate of the first vertex in the light image.</td>
</tr>
<tr>
<td>\textit{lx1}</td>
<td>X-coordinate of the second vertex in the light image.</td>
</tr>
<tr>
<td>\textit{ly1}</td>
<td>Y-coordinate of the second vertex in the light image.</td>
</tr>
<tr>
<td>\textit{lx2}</td>
<td>X-coordinate of the third vertex in the light image.</td>
</tr>
<tr>
<td>\textit{ly2}</td>
<td>Y-coordinate of the third vertex in the light image.</td>
</tr>
<tr>
<td>\textit{opacity}</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

8.1.4.435 draw_triangle() [8/9]

\begin{verbatim}
CImg<T> & draw_triangle ( int x0, int y0, int x1, int y1, |
\end{verbatim}
int x0,  
int y0,  
const CImg< tc > & texture,  
int tx0,  
int ty0,  
int tx1,  
int ty1,  
int tx2,  
int ty2,  
float bs0,  
float bs1,  
float bs2,  
const float opacity = 1  
)

Draw a textured Gouraud-shaded 2D triangle.

### Parameters

<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>
Draw a textured Phong-shaded 2D triangle.

Parameters

| x0   | X-coordinate of the first vertex in the image instance. |
| y0   | Y-coordinate of the first vertex in the image instance. |
| x1   | X-coordinate of the second vertex in the image instance. |
| y1   | Y-coordinate of the second vertex in the image instance. |
| x2   | X-coordinate of the third vertex in the image instance. |
| y2   | Y-coordinate of the third vertex in the image instance. |
| texture | Texture image used to fill the triangle. |
| tx0  | X-coordinate of the first vertex in the texture image. |
| ty0  | Y-coordinate of the first vertex in the texture image. |
| tx1  | X-coordinate of the second vertex in the texture image. |
| ty1  | Y-coordinate of the second vertex in the texture image. |
| tx2  | X-coordinate of the third vertex in the texture image. |
| ty2  | Y-coordinate of the third vertex in the texture image. |
| light | Light image. |
| lx0  | X-coordinate of the first vertex in the light image. |
| ly0  | Y-coordinate of the first vertex in the light image. |
| lx1  | X-coordinate of the second vertex in the light image. |
| ly1  | Y-coordinate of the second vertex in the light image. |
| lx2  | X-coordinate of the third vertex in the light image. |
| ly2  | Y-coordinate of the third vertex in the light image. |
| opacity | Drawing opacity. |

8.1.4.437 `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,  
```
const T \texttt{val},
const float \texttt{opacity} = 1 \}

Draw a filled 4D rectangle.

**Parameters**

<table>
<thead>
<tr>
<th>\texttt{x0}</th>
<th>X-coordinate of the upper-left rectangle corner.</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{y0}</td>
<td>Y-coordinate of the upper-left rectangle corner.</td>
</tr>
<tr>
<td>\texttt{z0}</td>
<td>Z-coordinate of the upper-left rectangle corner.</td>
</tr>
<tr>
<td>\texttt{c0}</td>
<td>C-coordinate of the upper-left rectangle corner.</td>
</tr>
<tr>
<td>\texttt{x1}</td>
<td>X-coordinate of the lower-right rectangle corner.</td>
</tr>
<tr>
<td>\texttt{y1}</td>
<td>Y-coordinate of the lower-right rectangle corner.</td>
</tr>
<tr>
<td>\texttt{z1}</td>
<td>Z-coordinate of the lower-right rectangle corner.</td>
</tr>
<tr>
<td>\texttt{c1}</td>
<td>C-coordinate of the lower-right rectangle corner.</td>
</tr>
<tr>
<td>\texttt{val}</td>
<td>Scalar value used to fill the rectangle area.</td>
</tr>
<tr>
<td>\texttt{opacity}</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

8.1.4.438 **draw_rectangle()** [2/3]

\begin{verbatim}
CImg<T>& draw_rectangle (  
    const int \texttt{x0},  
    const int \texttt{y0},  
    const int \texttt{z0},  
    const int \texttt{x1},  
    const int \texttt{y1},  
    const int \texttt{z1},  
    \texttt{const tc *const color},  
    const float \texttt{opacity} = 1 \}
\end{verbatim}

Draw a filled 3D rectangle.

**Parameters**

<table>
<thead>
<tr>
<th>\texttt{x0}</th>
<th>X-coordinate of the upper-left rectangle corner.</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{y0}</td>
<td>Y-coordinate of the upper-left rectangle corner.</td>
</tr>
<tr>
<td>\texttt{z0}</td>
<td>Z-coordinate of the upper-left rectangle corner.</td>
</tr>
<tr>
<td>\texttt{x1}</td>
<td>X-coordinate of the lower-right rectangle corner.</td>
</tr>
<tr>
<td>\texttt{y1}</td>
<td>Y-coordinate of the lower-right rectangle corner.</td>
</tr>
<tr>
<td>\texttt{z1}</td>
<td>Z-coordinate of the lower-right rectangle corner.</td>
</tr>
<tr>
<td>\texttt{color}</td>
<td>Pointer to \texttt{spectrum()} consecutive values of type \texttt{T}, defining the drawing color.</td>
</tr>
<tr>
<td>\texttt{opacity}</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

8.1.4.439 **draw_rectangle()** [3/3]

\begin{verbatim}
CImg<T>& draw_rectangle (  
\end{verbatim}
8.1 CImg< T > Struct Template Reference

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 T, defining the drawing color.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

8.1.4.40 draw_polygon()

```
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 T, defining the drawing color.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

8.1.4.41 draw_ellipse()

```
CImg<T>& draw_ellipse ( 
  const int x0,
  const int y0,
  const float r1,
  const float r2,
  const float angle,
  const tc *const color,
  const float opacity = 1 )
```

Draw a filled 2D ellipse.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>X-coordinate of the ellipse center.</td>
</tr>
<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 <code>spectrum()</code> consecutive values, defining the drawing color.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

8.1.4.442  `draw_ellipse()` [2/4]

```cpp
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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>X-coordinate of the ellipse center.</td>
</tr>
<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 <code>spectrum()</code> consecutive values, defining the drawing color.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

8.1.4.443  `draw_ellipse()` [3/4]

```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,  
  const unsigned int pattern )
```

Draw an outlined 2D ellipse.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>X-coordinate of the ellipse center.</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>y0</code></td>
<td>Y-coordinate of the ellipse center.</td>
</tr>
<tr>
<td><code>r1</code></td>
<td>First radius of the ellipse.</td>
</tr>
<tr>
<td><code>r2</code></td>
<td>Second radius of the ellipse.</td>
</tr>
<tr>
<td><code>angle</code></td>
<td>Angle of the first radius.</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>An integer whose bits describe the outline pattern.</td>
</tr>
</tbody>
</table>

8.1.4.444 draw_ellipse() [4/4]

```cpp
CImg<T>& draw_ellipse(
    const int x0,
    const int y0,
    const CImg<T> & tensor,
    const tc * const color,
    const float opacity,
    const unsigned int pattern)
```

Draw an outlined 2D ellipse [overloading].

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x0</code></td>
<td>X-coordinate of the ellipse center.</td>
</tr>
<tr>
<td><code>y0</code></td>
<td>Y-coordinate of the ellipse center.</td>
</tr>
<tr>
<td><code>tensor</code></td>
<td>Diffusion tensor describing the ellipse.</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>An integer whose bits describe the outline pattern.</td>
</tr>
</tbody>
</table>

8.1.4.445 draw_circle() [1/2]

```cpp
CImg<T>& draw_circle(
    const int x0,
    const int y0,
    int radius,
    const tc *const color,
    const float opacity = 1)
```

Draw a filled 2D circle.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x0</code></td>
<td>X-coordinate of the circle center.</td>
</tr>
<tr>
<td><code>y0</code></td>
<td>Y-coordinate of the circle center.</td>
</tr>
</tbody>
</table>
Parameters

| radius | Circle radius. |
| color | Pointer to `spectrum()` consecutive values, defining the drawing color. |
| opacity | Drawing opacity. |

Note

- Circle version of the Bresenham’s algorithm is used.

8.1.4.446 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

| x0 | X-coordinate of the circle center. |
| y0 | Y-coordinate of the circle center. |
| radius | Circle radius. |
| color | Pointer to `spectrum()` consecutive values, defining the drawing color. |
| opacity | Drawing opacity. |
| pattern | An integer whose bits describe the outline pattern. |

8.1.4.447 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. |
Parameter

<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>

8.1.4.448 draw_image() [2/2]

```cpp
CImg<T>& draw_image {
    const int x0,
    const int y0,
    const int z0,
    const int c0,
    const CImg<ti> & sprite,
    const CImg<tm> & mask,
    const float opacity = 1,
    const float mask_max_value = 1
}
```

Draw a masked image.

Parameter

<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.

8.1.4.449 draw_text() [1/4]

```cpp
CImg<T>& draw_text {
    const int x0,
    const int y0,
    const char * const text,
```
const tc1 *const foreground_color,
const tc2 *const background_color,
const float opacity,
const CImgList< t > & font,
... )

Draw a text 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 spectrum() consecutive values, defining the foreground drawing color.</td>
</tr>
<tr>
<td>background_color</td>
<td>Pointer to spectrum() 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>

8.1.4.450 draw_text() [2/4]

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.

8.1.4.451 draw_text() [3/4]

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.
8.1.4.452 draw_text() [4/4]

CImg<T>& draw_text (  
    const int x0,  
    const int y0,  
    const char *const text,  
    const tc1 *const foreground_color,  
    const tc2 *const background_color,  
    const float opacity = 1,  
    const unsigned int font_height = 13,  
    ...  )

Draw a text 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 mean</td>
</tr>
<tr>
<td>background_color</td>
<td>Array of spectrum() values of type T, defining the background color (0 mean</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>

8.1.4.453 draw_quiver() [1/2]

CImg<T>& draw_quiver (  
    const CImg< t1 > & flow,  
    const t2 *const color,  
    const float opacity = 1,  
    const unsigned int sampling = 25,  
    const float factor = -20,  
    const bool is_arrow = true,  
    const unsigned int pattern = ~0U )

Draw a 2D vector field.

Parameters

<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</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>
Class Documentation

Note

Clipping is supported.

8.1.4.454 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 spectrum()-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.

8.1.4.455 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.
Parameters

<table>
<thead>
<tr>
<th>values_x</th>
<th>Values along the horizontal axis.</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>Y-coordinate of the horizontal axis in the image instance.</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>Drawing pattern.</td>
</tr>
<tr>
<td>font_height</td>
<td>Height of the labels (exact match for 13,23,53,103, interpolated otherwise).</td>
</tr>
<tr>
<td>allow_zero</td>
<td>Enable/disable the drawing of label '0' if found.</td>
</tr>
</tbody>
</table>

8.1.4.456 `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>x</th>
<th>X-coordinate of the vertical axis in the image instance.</th>
</tr>
</thead>
<tbody>
<tr>
<td>values_y</td>
<td>Values along the Y-axis.</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>Drawing pattern.</td>
</tr>
<tr>
<td>font_height</td>
<td>Height of the labels (exact match for 13,23,53,103, interpolated otherwise).</td>
</tr>
<tr>
<td>allow_zero</td>
<td>Enable/disable the drawing of label '0' if found.</td>
</tr>
</tbody>
</table>

8.1.4.457 `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

<table>
<thead>
<tr>
<th>values_x</th>
<th>Values along the X-axis.</th>
</tr>
</thead>
<tbody>
<tr>
<td>values_y</td>
<td>Values along the Y-axis.</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>
<tr>
<td>pattern_x</td>
<td>Drawing pattern for the X-axis.</td>
</tr>
<tr>
<td>pattern_y</td>
<td>Drawing pattern for the Y-axis.</td>
</tr>
<tr>
<td>font_height</td>
<td>Height of the labels (exact match for 13,23,53,103, interpolated otherwise).</td>
</tr>
<tr>
<td>allow_zero</td>
<td>Enable/disable the drawing of label '0' if found.</td>
</tr>
</tbody>
</table>

8.1.4.458 draw_grid()

```cpp
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

<table>
<thead>
<tr>
<th>values←_x</th>
<th>X-coordinates of the vertical lines.</th>
</tr>
</thead>
<tbody>
<tr>
<td>values←_y</td>
<td>Y-coordinates of the horizontal lines.</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>
<tr>
<td>pattern←_x</td>
<td>Drawing pattern for vertical lines.</td>
</tr>
<tr>
<td>pattern←_y</td>
<td>Drawing pattern for horizontal lines.</td>
</tr>
</tbody>
</table>

8.1.4.459 draw_graph()

```cpp
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 <code>spectrum()</code> 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.

8.1.4.460 draw_fill()

```cpp
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.

Parameters

<table>
<thead>
<tr>
<th>Parameter</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.

8.1.4.461 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>Parameter</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.

8.1.4.462 draw_mandelbrot()

CImg<T> & draw_mandelbrot ( const int x0,
const int y0,
const int x1,
const int y1,
const CImg< tc > & colormap,
const float opacity = 1,
const double z0r = -2,
const double z0i = -2,
const double z1r = 2,
const double z1i = 2,
const unsigned int iteration_max = 255,
const bool is_normalized_iteration = false,
const bool is_julia_set = false,
const double param_r = 0,
const double param_i = 0)

Draw a quadratic Mandelbrot or Julia 2D fractal.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>X-coordinate of the upper-left pixel.</td>
</tr>
<tr>
<td>y0</td>
<td>Y-coordinate of the upper-left pixel.</td>
</tr>
<tr>
<td>x1</td>
<td>X-coordinate of the lower-right pixel.</td>
</tr>
<tr>
<td>y1</td>
<td>Y-coordinate of the lower-right pixel.</td>
</tr>
<tr>
<td>colormap</td>
<td>Colormap.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
<tr>
<td>z0r</td>
<td>Real part of the upper-left fractal vertex.</td>
</tr>
<tr>
<td>z0i</td>
<td>Imaginary part of the upper-left fractal vertex.</td>
</tr>
<tr>
<td>z1r</td>
<td>Real part of the lower-right fractal vertex.</td>
</tr>
<tr>
<td>z1i</td>
<td>Imaginary part of the lower-right fractal vertex.</td>
</tr>
<tr>
<td>iteration_max</td>
<td>Maximum number of iterations for each estimated point.</td>
</tr>
<tr>
<td>is_normalized_iteration</td>
<td>Tells if iterations are normalized.</td>
</tr>
<tr>
<td>is_julia_set</td>
<td>Tells if the Mandelbrot or Julia set is rendered.</td>
</tr>
<tr>
<td>param_r</td>
<td>Real part of the Julia set parameter.</td>
</tr>
<tr>
<td>param_i</td>
<td>Imaginary part of the Julia set parameter.</td>
</tr>
</tbody>
</table>

**Note**

Fractal rendering is done by the Escape Time Algorithm.

---

8.1.4.463 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 spectrum() consecutive values, defining the drawing color.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

8.1.4.464 draw_gaussian() [2/2]

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

Draw a 2D gaussian function.

Parameters

<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 spectrum() consecutive values, defining the drawing color.</td>
</tr>
<tr>
<td>opacity</td>
<td>Drawing opacity.</td>
</tr>
</tbody>
</table>

8.1.4.465 draw_object3d()

CImg<T>& draw_object3d (  
    const float x0,  
    const float y0,  
    const float z0,  
    const CImg< tp >& vertices,  
    const CImgList< tf >& primitives,  
    const CImgList< tc >& colors,  
    const CImg< to >& opacities,  
    const unsigned int render_type = 4,  
    const bool is_double_sided = false,  
    const float 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.
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>vertices</td>
<td>Image $N \times 3$ describing 3D point coordinates</td>
</tr>
<tr>
<td>primitives</td>
<td>List of $P$ primitives</td>
</tr>
<tr>
<td>colors</td>
<td>List of $P$ color (or textures)</td>
</tr>
<tr>
<td>opacities</td>
<td>Image or list of $P$ opacities</td>
</tr>
<tr>
<td>render_type</td>
<td>Render type ($0$=Points, $1$=Lines, $2$=Faces (no light), $3$=Faces (flat), $4$=Faces (Gouraud))</td>
</tr>
<tr>
<td>is_double_sided</td>
<td>Tells if object faces have two sides or are oriented.</td>
</tr>
<tr>
<td>focale</td>
<td>Length of the focale ($0$ for parallel projection)</td>
</tr>
<tr>
<td>lightx</td>
<td>X-coordinate of the light</td>
</tr>
<tr>
<td>lighty</td>
<td>Y-coordinate of the light</td>
</tr>
<tr>
<td>lightz</td>
<td>Z-coordinate of the light</td>
</tr>
<tr>
<td>specular_lightness</td>
<td>Amount of specular light.</td>
</tr>
<tr>
<td>specular_shininess</td>
<td>Shininess of the object.</td>
</tr>
<tr>
<td>$g$_opacity</td>
<td>Global opacity of the object.</td>
</tr>
</tbody>
</table>

8.1.4.466 select()

```cpp
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.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>disp</td>
<td>Display window to use.</td>
</tr>
<tr>
<td>feature_type</td>
<td>Type of feature to select. Can be { 0=point</td>
</tr>
<tr>
<td>XYZ</td>
<td>Pointer to 3 values X,Y,Z which tells about the projection point coordinates, for volumetric images.</td>
</tr>
<tr>
<td>exit_on_anykey</td>
<td>Exit function when any key is pressed.</td>
</tr>
</tbody>
</table>

8.1.4.467 load()

```cpp
CImg<T>& load (  
    const char *const filename )
```

Load image from a 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>

Note

The extension of `filename` defines the file format. If no filename extension is provided, `CImg<T>::get←load()` will try to load the file as a .cimg or .cimgz file.

8.1.4.468 load_ascii()

```
CImg<T>& load_ascii (const char *const filename )
```

Load image from an ascii 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>

8.1.4.469 load_dlm()

```
CImg<T>& load_dlm (const char *const filename )
```

Load image from a DLM 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>

8.1.4.470 load_bmp()

```
CImg<T>& load_bmp (const char *const filename )
```

Load image from a BMP 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>
8.1.4.471  load_jpeg()

CImg<T>& load_jpeg (const char *const filename)

Load image from a JPEG file.

Parameters

| filename | Filename, as a C-string. |

8.1.4.472  load_magick()

CImg<T>& load_magick (const char *const filename)

Load image from a file, using Magick++ library.

Parameters

| filename | Filename, as a C-string. |

8.1.4.473  load_png()

CImg<T>& load_png (const char *const filename, unsigned int *const bits_per_pixel = 0)

Load image from a PNG file.

Parameters

| filename | Filename, as a C-string. |
| out      | bits_per_pixel  | Number of bits per pixels used to store pixel values in the image file. |

8.1.4.474  load_pnm()

CImg<T>& load_pnm (const char *const filename)

Generated by Doxygen
Load image from a PNM file.
8.1.4.475  load_pfm()

\texttt{CImg}\langle T\rangle \& \text{load}\_pfm (  
\quad \text{const char } \ast \text{const filename} 
)

Load image from a PFM file.

Parameters

| filename | Filename, as a C-string. |

8.1.4.476  load_rgb()

\texttt{CImg}\langle T\rangle \& \text{load}\_rgb (  
\quad \text{const char } \ast \text{const filename},  
\quad \text{const unsigned int dimw},  
\quad \text{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. |

8.1.4.477  load_rgba()

\texttt{CImg}\langle T\rangle \& \text{load}\_rgba (  
\quad \text{const char } \ast \text{const filename},  
\quad \text{const unsigned int dimw},  
\quad \text{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. |
8.1.4.478 load_tiff()

\begin{verbatim}
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 )
\end{verbatim}

Load image from 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>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>voxel_size</td>
<td>Voxel size, as stored in the filename.</td>
</tr>
<tr>
<td>description</td>
<td>Description, as stored in the filename.</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>& load_other(const char*)`.

8.1.4.479 load_minc2()

\begin{verbatim}
CImg<T>& load_minc2 ( 
    const char *const filename )
\end{verbatim}

Load image from 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>
</tbody>
</table>

8.1.4.480 load_analyze()

\begin{verbatim}
CImg<T>& load_analyze ( 
\end{verbatim}

Generated by Doxygen
const char *const filename,
float *const voxel_size = 0 )

Load image from an ANALYZE7.5/NIFTI file.

Parameters

<table>
<thead>
<tr>
<th>filename</th>
<th>File name, as a C-string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>voxel_size</td>
<td>Pointer to the three voxel sizes read from the file.</td>
</tr>
</tbody>
</table>

8.1.4.481 load_cimg() [1/2]

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>filename</th>
<th>File name, as a C-string.</th>
</tr>
</thead>
<tbody>
<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>

8.1.4.482 load_cimg() [2/2]

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

| filename | File name, as a C-string. |
Parameters

<table>
<thead>
<tr>
<th>n0</th>
<th>Starting frame.</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1</td>
<td>Ending frame (~0U for max).</td>
</tr>
<tr>
<td>x0</td>
<td>X-coordinate of the starting sub-image vertex.</td>
</tr>
<tr>
<td>y0</td>
<td>Y-coordinate of the starting sub-image vertex.</td>
</tr>
<tr>
<td>z0</td>
<td>Z-coordinate of the starting sub-image vertex.</td>
</tr>
<tr>
<td>c0</td>
<td>C-coordinate of the starting sub-image vertex.</td>
</tr>
<tr>
<td>x1</td>
<td>X-coordinate of the ending sub-image vertex (~0U for max).</td>
</tr>
<tr>
<td>y1</td>
<td>Y-coordinate of the ending sub-image vertex (~0U for max).</td>
</tr>
<tr>
<td>z1</td>
<td>Z-coordinate of the ending sub-image vertex (~0U for max).</td>
</tr>
<tr>
<td>c1</td>
<td>C-coordinate of the ending sub-image vertex (~0U for max).</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>

8.1.4.483 load_inr()

CImg<T>& load_inr (  
    const char *const filename,  
    float *const voxel_size = 0 )

Load image from an INRIMAGE-4 file.

Parameters

<table>
<thead>
<tr>
<th>filename</th>
<th>Filename, as a C-string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>voxel_size</td>
</tr>
<tr>
<td></td>
<td>Pointer to the three voxel sizes read from the file.</td>
</tr>
</tbody>
</table>

8.1.4.484 load_exr()

CImg<T>& load_exr (  
    const char *const filename )

Load image from a EXR file.

Parameters

| filename | Filename, as a C-string. |
8.1.4.485 load_pandore()

\texttt{CImg\langle T\rangle \& load\_pandore (}
\texttt{const char \&const filename )}

Load image from a PANDORE-5 file.

\textbf{Parameters}

\begin{tabular}{|l|l|}
\hline
\textit{filename} & Filename, as a C-string. \\
\hline
\end{tabular}

8.1.4.486 load_parrec()

\texttt{CImg\langle T\rangle \& load\_parrec (}
\texttt{const char \&const filename,}
\texttt{const char axis = 'c',}
\texttt{const float align = 0 )}

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

\textbf{Parameters}

\begin{tabular}{|l|l|}
\hline
\textit{filename} & Filename, as a C-string. \\
\textit{axis} & Appending axis, if file contains multiple images. Can be \{ 'x' | 'y' | 'z' | 'c' \}. \\
\textit{align} & Appending alignment. \\
\hline
\end{tabular}

8.1.4.487 load_raw()

\texttt{CImg\langle T\rangle \& load\_raw (}
\texttt{const char \&const filename,}
\texttt{const unsigned int size_x = 0,}
\texttt{const unsigned int size_y = 1,}
\texttt{const unsigned int size_z = 1,}
\texttt{const unsigned int size_c = 1,}
\texttt{const bool is\_multiplexed = false,}
\texttt{const bool invert\_endianness = false,}
\texttt{const ulongT offset = 0 )}

Load image from a raw binary file.

\textbf{Parameters}

\begin{tabular}{|l|l|}
\hline
\textit{filename} & Filename, as a C-string. \\
\textit{size\_x} & Width of the image buffer. \\
\textit{size\_y} & Height of the image buffer. \\
\textit{size\_z} & Depth of the image buffer. \\
\hline
\end{tabular}
8.1.4.488 load_yuv()

```c
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>

8.1.4.489 load_off()

```c
CImg<T>& load_off(CImgList<tf>& primitives,
                  CImgList<tc>& colors,
                  const char *const filename )
```

Load 3D object from a .OFF file.
8.1.4.490  load_video()

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

Load image sequence from a video file, using OpenCV library.

Parameters

<table>
<thead>
<tr>
<th>filename</th>
<th>Filename, as a C-string.</th>
</tr>
</thead>
<tbody>
<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>axis</td>
<td>Alignment axis.</td>
</tr>
<tr>
<td>align</td>
<td>Appending alignment.</td>
</tr>
</tbody>
</table>

8.1.4.491  load_ffmpeg_external()

```cpp
CImg<T>& load_ffmpeg_external ( const char *const filename,
                               const char axis = 'z',
                               const float align = 0 )
```

Load image sequence using FFMPEG's external tool 'ffmpeg'.

Parameters

<table>
<thead>
<tr>
<th>filename</th>
<th>Filename, as a C-string.</th>
</tr>
</thead>
<tbody>
<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>
8.1.4.492 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.

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>Appending axis, if file contains multiple images. Can be ( 'x'</td>
</tr>
<tr>
<td>align</td>
<td>Appending alignment.</td>
</tr>
</tbody>
</table>

8.1.4.493 load_graphicsmagick_external() 

```cpp
CImg<T>& load_graphicsmagick_external(
    const char *const filename )
```

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

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>

8.1.4.494 load_gzip_external() 

```cpp
CImg<T>& load_gzip_external(
    const char *const filename )
```

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

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>

8.1.4.495 load_imagemagick_external() 

```cpp
CImg<T>& load_imagemagick_external(
    const char *const filename )
```

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

Generated by Doxygen
8.1.4.496  load_medcon_external()

\texttt{CImg\langle T\rangle \& load\_medcon\_external (}
\texttt{    const char \*const filename }
\texttt{)}

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

Parameters

| filename | Filename, as a C-string. |

8.1.4.497  load_dcraw_external()

\texttt{CImg\langle T\rangle \& load\_dcraw\_external (}
\texttt{    const char \*const filename }
\texttt{)}

Load image from a RAW Color Camera file, using external tool 'dcraw'.

Parameters

| filename | Filename, as a C-string. |

8.1.4.498  load_camera()

\texttt{CImg\langle T\rangle \& load\_camera (}
\texttt{    const unsigned int camera\_index = 0,}
\texttt{    const unsigned int capture\_width = 0,}
\texttt{    const unsigned int capture\_height = 0,}
\texttt{    const unsigned int skip\_frames = 0,}
\texttt{    const bool release\_camera = true )}

Load image from a camera stream, using OpenCV.

Parameters

<table>
<thead>
<tr>
<th>index</th>
<th>Index of the camera to capture images from (from 0 to 63).</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture_width</td>
<td>Width of the desired image ('0' stands for default value).</td>
</tr>
<tr>
<td>capture_height</td>
<td>Height of the desired image ('0' stands for default value).</td>
</tr>
<tr>
<td>skip_frames</td>
<td>Number of frames to skip before the capture.</td>
</tr>
<tr>
<td>release_camera</td>
<td>Tells if the camera resource must be released at the end of the method.</td>
</tr>
</tbody>
</table>
8.1.4.499 load_other()

```cpp
CImg<
T>
& load_other (const char *const filename )
```

Load image using various non-native ways.

Parameters

<table>
<thead>
<tr>
<th>filename</th>
<th>Filename, as a C-string.</th>
</tr>
</thead>
</table>

8.1.4.500 print()

```cpp
const CImg<
T>
& print (const char *const title = 0, const bool display_stats = true ) const
```

Display information about the image data.

Parameters

<table>
<thead>
<tr>
<th>title</th>
<th>Name for the considered image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>display_stats</td>
<td>Tells to compute and display image statistics.</td>
</tr>
</tbody>
</table>

8.1.4.501 display() [1/3]

```cpp
const CImg<
T>
& display (CImgDisplay & disp ) const
```

Display image into a CImgDisplay window.

Parameters

<table>
<thead>
<tr>
<th>disp</th>
<th>Display window.</th>
</tr>
</thead>
</table>

8.1.4.502 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>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>

8.1.4.503 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>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>

8.1.4.504 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,
Display object 3D in an interactive window.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>disp</strong></td>
<td>Display window.</td>
</tr>
<tr>
<td><strong>vertices</strong></td>
<td>Vertices data of the 3D object.</td>
</tr>
<tr>
<td><strong>primitives</strong></td>
<td>Primitives data of the 3D object.</td>
</tr>
<tr>
<td><strong>colors</strong></td>
<td>Colors data of the 3D object.</td>
</tr>
<tr>
<td><strong>opacities</strong></td>
<td>Opacities data of the 3D object.</td>
</tr>
<tr>
<td><strong>centering</strong></td>
<td>Tells if the 3D object must be centered for the display.</td>
</tr>
<tr>
<td><strong>render_static</strong></td>
<td>Rendering mode.</td>
</tr>
<tr>
<td><strong>render_motion</strong></td>
<td>Rendering mode, when the 3D object is moved.</td>
</tr>
<tr>
<td><strong>is_double_sided</strong></td>
<td>Tells if the object primitives are double-sided.</td>
</tr>
<tr>
<td><strong>focale</strong></td>
<td>Focale</td>
</tr>
<tr>
<td><strong>light_x</strong></td>
<td>X-coordinate of the light source.</td>
</tr>
<tr>
<td><strong>light_y</strong></td>
<td>Y-coordinate of the light source.</td>
</tr>
<tr>
<td><strong>light_z</strong></td>
<td>Z-coordinate of the light source.</td>
</tr>
<tr>
<td><strong>specular_lightness</strong></td>
<td>Amount of specular light.</td>
</tr>
<tr>
<td><strong>specular_shininess</strong></td>
<td>Shininess of the object material.</td>
</tr>
<tr>
<td><strong>display_axes</strong></td>
<td>Tells if the 3D axes are displayed.</td>
</tr>
<tr>
<td><strong>pose_matrix</strong></td>
<td>Pointer to 12 values, defining a 3D pose (as a 4x3 matrix).</td>
</tr>
<tr>
<td><strong>exit_on_anykey</strong></td>
<td>Exit function when any key is pressed.</td>
</tr>
</tbody>
</table>

8.1.4505 display_graph()

Display 1D graph in an interactive window.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>disp</strong></td>
<td>Display window.</td>
</tr>
<tr>
<td><strong>plot_type</strong></td>
<td>Plot type. Can be { 0=points</td>
</tr>
<tr>
<td><strong>vertex_type</strong></td>
<td>Vertex type.</td>
</tr>
</tbody>
</table>
8.1.4.506  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**

<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>number</code></td>
<td>When positive, represents an index added to the filename. Otherwise, no number is added.</td>
</tr>
<tr>
<td><code>digits</code></td>
<td>Number of digits used for adding the number to the filename.</td>
</tr>
</tbody>
</table>

**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.

8.1.4.507  save_ascii()

```cpp
const CImg< T >& save_ascii ( 
    const char *const filename ) const
```

Save image as an ascii 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>
8.1.4.508  save_cpp()

const CImg<T>& save_cpp (  
    const char *const filename ) const

Save image as a .cpp source file.

Parameters

| filename  | Filename, as a C-string. |

8.1.4.509  save_dlm()

const CImg<T>& save_dlm (  
    const char *const filename ) const

Save image as a DLM file.

Parameters

| filename  | Filename, as a C-string. |

8.1.4.510  save_bmp()

const CImg<T>& save_bmp (  
    const char *const filename ) const

Save image as a BMP file.

Parameters

| filename  | Filename, as a C-string. |

8.1.4.511  save_jpeg()

const CImg<T>& save_jpeg (  
    const char *const filename,  
    const unsigned int quality = 100 ) const

Save image as a JPEG file.
8.1 CImg< T > Struct Template Reference

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>

8.1.4.512 save_magick()

```cpp
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>

8.1.4.513 save_png()

```cpp
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>

8.1.4.514 save_pnm()

```cpp
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 byte per pixels for the saving.</td>
</tr>
</tbody>
</table>
8.1.4.515  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. |

8.1.4.516  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. |

8.1.4.517  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. |

8.1.4.518  save_rgba()

const CImg<T>& save_rgba (  
    const char *const filename ) const  

Save image as a RGBA file.
### 8.1.4.519 save_tiff()

```cpp
cconst 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.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>filename</code></td>
<td>Filename, as a C-string.</td>
</tr>
<tr>
<td><code>compression_type</code></td>
<td>Type of data compression. Can be `{ 0=None</td>
</tr>
<tr>
<td><code>voxel_size</code></td>
<td>Voxel size, to be stored in the filename.</td>
</tr>
<tr>
<td><code>description</code></td>
<td>Description, to be stored in the filename.</td>
</tr>
<tr>
<td><code>use_bigtiff</code></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_tiff` is not defined at compile time the function uses `CImg<T>&save_other(const char*).`

### 8.1.4.520 save_minc2()

```cpp
cconst CImg<T>& save_minc2 {
    const char *const filename,
    const char *const imitate_file = 0 } const
```

Save image as a MINC2 file.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>filename</code></td>
<td>Filename, as a C-string.</td>
</tr>
<tr>
<td><code>imitate_file</code></td>
<td>If non-zero, reference filename, as a C-string, to borrow header from.</td>
</tr>
</tbody>
</table>
8.1.4.521  save_analyze()

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>filename</th>
<th>Filename, as a C-string.</th>
</tr>
</thead>
<tbody>
<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>

8.1.4.522  save_cimg()

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>filename</th>
<th>Filename, as a C-string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_compressed</td>
<td>Tells if the file contains compressed image data.</td>
</tr>
</tbody>
</table>

8.1.4.523  save_cimg()

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>filename</th>
<th>Filename, as a C-string.</th>
</tr>
</thead>
<tbody>
<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>

8.1.4.524  save_empty_cimg() [1/2]

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.

8.1.4.525  save_empty_cimg() [2/2]

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.
8.1.4.526 save_inr()

const CImg<

const char const filename,
const float const voxel_size = 0 ) const

Save image as an INRIMAGE-4 file.

Parameters

<table>
<thead>
<tr>
<th>filename</th>
<th>Filename, as a C-string.</th>
</tr>
</thead>
<tbody>
<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>

8.1.4.527 save_exr()

const CImg<

const char const filename ) const

Save image as an OpenEXR file.

Parameters

| filename  | Filename, as a C-string. |

Note

The OpenEXR file format is described here.

8.1.4.528 save_pandore() [1/2]

const CImg<

const char const filename,
const unsigned int colorspace = 0 ) const

Save image as a Pandore-5 file.

Parameters

| filename  | Filename, as a C-string. |
| colorspace | Colorspace data field in output file (see Pandore file specifications for more information). |
8.1.4.529  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 *,unsigned int) const with a file stream argument instead of a filename string.

8.1.4.530  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.

8.1.4.531  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.

8.1.4.532  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.
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.

8.1.4.533 save_yuv() [2/2]

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.

8.1.4.534 save_off() [1/2]

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.
8.1.4.535 save_off() [2/2]

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 with a file stream argument instead of a filename string.

8.1.4.536 save_video()

const CImg<T>& save_video (  
    const char * const filename,  
    const unsigned int fps = 25,  
    const char * codec = 0,  
    const bool keep_open = false ) const

Save volumetric image as a video, using the OpenCV library.

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>

8.1.4.537 save_ffmpeg_external()

const CImg<T>& save_ffmpeg_external (  
    const char * const filename,  
    const unsigned int fps = 25,  
    const char * 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>filename</td>
<td>Filename, as a C-string.</td>
</tr>
<tr>
<td>fps</td>
<td>Video framerate.</td>
</tr>
<tr>
<td>codec</td>
<td>Video codec, as a C-string.</td>
</tr>
<tr>
<td>bitrate</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.

8.1.4.538 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.

8.1.4.539 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.

8.1.4.540 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>filename</th>
<th>Filename, as a C-string.</th>
</tr>
</thead>
<tbody>
<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.

8.1.4.541 save_medcon_external()

```cpp
cnst CImg<T>& save_medcon_external {
    const char *const filename } const
```

Save image as a Dicom file.

Parameters

| filename | Filename, as a C-string. |

Note

This method uses `medcon`, an external executable binary provided by (X)Medcon. It must be installed for the method to succeed.

8.1.4.542 save_other()

```cpp
cnst CImg<T>& save_other {
    const char *const filename,
    const unsigned int quality = 100 } const
```

Parameters

<table>
<thead>
<tr>
<th>filename</th>
<th>Filename, as a C-string.</th>
</tr>
</thead>
<tbody>
<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.

8.1.4.543 get_serialize()

```cpp
CImg<ucharT> get_serialize (   
    const bool is_compressed = false ) const
```

Serialize a CImg<T> instance into a raw CImg<unsigned char> 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>

8.2 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 ()  
  Construct 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].
8.2 CImgDisplay Struct Reference

- template<typename T>
  CImgDisplay & assign (const CImg< T > &img, const char *const title=0, const unsigned int normalization=3, const bool is_fullscreen=false, const bool is_closed=false)
  Construct a display from an image [in-place version].

- template<typename T>
  CImgDisplay & assign (const CImgList< T > &list, const char *const title=0, const unsigned int normalization=3, const bool is_fullscreen=false, const bool is_closed=false)
  Construct a display from an image list [in-place version].

- CImgDisplay & assign (const CImgDisplay &disp)
  Construct a display as a copy of another one [in-place version].

- template<typename T>
  static void screenshot (CImg< T > &img)
  Take a screenshot.

- static CImgDisplay & empty ()
  Return a reference to an empty display.

- static const CImgDisplay & const_empty ()
  Return a reference to an empty display [const version].

Overloaded Operators

- template<typename t>
  CImgDisplay & operator= (const CImg< t > &img)
  Display image on associated window.

- template<typename t>
  CImgDisplay & operator= (const CImgList< t > &list)
  Display list of images on associated window.

- CImgDisplay & operator= (const CImgDisplay &disp)
  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)

Generated by Doxygen
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
8.2 CImgDisplay Struct Reference

- bool is_keyJ () const
- bool is_keyK () const
- bool is_keyL () const
- bool is_keyENTER () const
- bool is_keySHIFTLEFT () const
- bool is_keyZ () const
- bool is_keyX () const
- bool is_keyC () const
- bool is_keyV () const
- bool is_keyB () const
- bool is_keyN () const
- bool is_keyM () const
- bool is_keySHIFTRIGHT () const
- bool is_keyARROWUP () const
- bool is_keyCTRLLEFT () const
- bool is_keyAPPLEFT () const
- bool is_keyALT () const
- bool is_keySPACE () const
- bool is_keyALTGR () const
- bool is_keyAPPRIGHT () const
- bool is_keyMENU () const
- bool is_keyCTRLRIGHT () const
- bool is_keyARROWLEFT () const
- bool is_keyARROWDOWN () const
- bool is_keyARROWRIGHT () const
- bool is_keyPAD0 () const
- bool is_keyPAD1 () const
- bool is_keyPAD2 () const
- bool is_keyPAD3 () const
- bool is_keyPAD4 () const
- bool is_keyPAD5 () const
- bool is_keyPAD6 () const
- bool is_keyPAD7 () const
- bool is_keyPAD8 () const
- bool is_keyPAD9 () const
- bool is_keyPADADD () const
- bool is_keyPADSUB () const
- bool is_keyPADMUL () const
- bool is_keyPADDIV () const

Instance Characteristics

- int width () const
  
  Return display width.
- int height () const
  
  Return display height.
- unsigned int normalization () const
  
  Return normalization type of the display.
- const char * title () const
  
  Return title of the associated window as a C-string.
- int window_width () const
  
  Return width of the associated window.
- int window_height () const
Return height of the associated window.

- int window_x () const
    Return X-coordinate of the associated window.
- int window_y () const
    Return Y-coordinate of the associated window.
- int mouse_x () const
    Return X-coordinate of the mouse pointer.
- int mouse_y () const
    Return Y-coordinate of the mouse pointer.
- unsigned int button () const
    Return current state of the mouse buttons.
- int wheel () const
    Return current state of the mouse wheel.
- unsigned int key (const unsigned int pos=0) const
    Return one entry from the pressed keys history.
- unsigned int released_key (const unsigned int pos=0) const
    Return one entry from the released keys history.
- float frames_per_second ()
    Return the current refresh rate, in frames per second.

CImgDisplay & move_inside_screen ()

- CImgDisplay & display (const CImg &img)
    Display image on associated window.
- CImgDisplay & display (const CImgList &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 &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.
- CImgDisplay & move ()
    Show (closed) associated window on the screen.
- CImgDisplay & close ()
    Close (visible) associated window and make it disappear from the screen.
- CImgDisplay & move (const int pos_x, const int pos_y)
    Move associated window to a new location.
- CImgDisplay & resize (const bool force_redraw=true)
    Resize display to the size of the associated window.
- CImgDisplay & resize (const int width, const int height, const bool force_redraw=true)
    Resize display to the specified size.
- template<typename T>
  CImgDisplay & resize (const CImg<T> &img, const bool force_redraw=true)
    Resize display to the size of an input image.
- CImgDisplay & resize (const CImgDisplay &disp, const bool force_redraw=true)
    Resize display to the size of another CImgDisplay instance.
• CImgDisplay & set_normalization (const unsigned int normalization)
  Set normalization type.
• CImgDisplay & set_title (const char *const format,...)
  Set title of the associated window.
• CImgDisplay & setFullscreen (const bool isFullscreen, 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.
• static void wait (CImgDisplay &disp1, CImgDisplay &disp2, CImgDisplay &disp3, CImgDisplay &disp4, CImgDisplay &disp5, CImgDisplay &disp6)
  Wait for any event occurring either on the display disp1, disp2, disp3, disp4, ... disp6.

• static void wait (CImgDisplay &disp1, CImgDisplay &disp2, CImgDisplay &disp3, CImgDisplay &disp4, CImgDisplay &disp5, CImgDisplay &disp6, CImgDisplay &disp7)
  Wait for any event occurring either on the display disp1, disp2, disp3, disp4, ... disp7.

• static void wait (CImgDisplay &disp1, CImgDisplay &disp2, CImgDisplay &disp3, CImgDisplay &disp4, CImgDisplay &disp5, CImgDisplay &disp6, CImgDisplay &disp7, CImgDisplay &disp8)
  Wait for any event occurring either on the display disp1, disp2, disp3, disp4, ... disp8.

• static void wait (CImgDisplay &disp1, CImgDisplay &disp2, CImgDisplay &disp3, CImgDisplay &disp4, CImgDisplay &disp5, CImgDisplay &disp6, CImgDisplay &disp7, CImgDisplay &disp8, CImgDisplay &disp9)
  Wait for any event occurring either on the display disp1, disp2, disp3, disp4, ... disp9.

• static void wait (CImgDisplay &disp1, CImgDisplay &disp2, CImgDisplay &disp3, CImgDisplay &disp4, CImgDisplay &disp5, CImgDisplay &disp6, CImgDisplay &disp7, CImgDisplay &disp8, CImgDisplay &disp9, CImgDisplay &disp10)
  Wait for any event occurring either on the display disp1, disp2, disp3, disp4, ... disp10.

• static void wait_all ()
  Wait for any window event occurring in any opened CImgDisplay.

• template<typename T>
  static void screenshot (const int x0, const int y0, const int x1, const int y1, CImg<T>& img)
  Take a snapshot of the current screen content.

8.2.1 Detailed Description

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

CImgDisplay 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
8.2 CImgDisplay Struct Reference

### 8.2.2.1 \~CImgDisplay()

\~CImgDisplay ( )

Destructor.

**Note**

If the associated window is visible on the screen, it is closed by the call to the destructor.

---

### 8.2.2.2 CImgDisplay() [1/5]

CImgDisplay ( )

Construct an empty display.

**Note**

Constructing an empty CImgDisplay instance does not make a window appearing on the screen, until display of valid data is performed.

**Example**

```cpp
CImgDisplay disp; // Does actually nothing
... disp.display(img); // Construct new window and display image in it
```

---

### 8.2.2.3 CImgDisplay() [2/5]

CImgDisplay ( const unsigned int width, const unsigned int height, const char ∗const title = 0, const unsigned int normalization = 3, const bool is_fullscreen = false, const bool is_closed = false )

Construct a display with specified dimensions.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>width</code></td>
<td>Window width.</td>
</tr>
<tr>
<td><code>height</code></td>
<td>Window height.</td>
</tr>
<tr>
<td><code>title</code></td>
<td>Window title.</td>
</tr>
<tr>
<td><code>normalization</code></td>
<td>Normalization type (0=none, 1=always, 2=once, 3=pixel type-dependent, see normalization()).</td>
</tr>
<tr>
<td><code>is_fullscreen</code></td>
<td>Tells if fullscreen mode is enabled.</td>
</tr>
<tr>
<td><code>is_closed</code></td>
<td>Tells if associated window is initially visible or not.</td>
</tr>
</tbody>
</table>

Generated by Doxygen
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 isfullscreen = 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 isfullscreen = 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>
Note

All images of the list, appended along the X-axis, are initially displayed on the associated window.

8.2.2.6 CImgDisplay()

CImgDisplay (const CImgDisplay & disp)

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()

static void screenshot (CImg< T > & img) [static]

Take a screenshot.

Parameters

out | img | Output screenshot. Can be empty on input

8.2.3.2 assign()

CImgDisplay& assign ()

Destructor - Empty constructor [in-place version].

Note

Replace the current instance by an empty display.
8.2.3.3 empty()

static CImgDisplay& empty() [static]

Return a reference to an empty display.

Note

Can be useful for writing function prototypes where one of the argument (of type CImgDisplay&) must have a
default value.

Example

void foo(CImgDisplay& disp=CImgDisplay::empty());

8.2.3.4 operator=() [1/3]

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 false if display is empty, true otherwise.

Note
   if (disp) { ... } is equivalent to if (!disp.is_empty()) { ... }.

8.2.3.8 is_closed()

bool is_closed ( ) const

Return true if display is closed (i.e. not visible on the screen), false otherwise.

Note
   • When a user physically closes the associated window, the display is set to closed.
   • A closed display is not destroyed. Its associated window can be show again on the screen using show().

8.2.3.9 is_key() [1/3]

bool is_key ( ) const

Return true if any key is being pressed on the associated window, false otherwise.

Note
   The 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 true if key specified by given keycode is being pressed on the associated window, false otherwise.

Parameters
  
<table>
<thead>
<tr>
<th>keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keycode</td>
<td>Keycode to test.</td>
</tr>
</tbody>
</table>
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);
while (!disp.is_closed()) {
    if (disp.key(cimg::keyTAB)) { ... } // Equivalent to 'if (disp.is_keyTAB())'
    disp.wait();
}
```

### 8.2.3.11 is_key()

```cpp
bool is_key {
    const char *const keycode}
```

**Return** `true` if key specified by given keycode is being pressed on the associated window, `false` otherwise.

**Parameters**

| `keycode`         | C-string containing the keycode label of the key to test. |

**Note**

Use it when the key you want to test can be dynamically set by the user.

Example

```cpp
CImgDisplay disp(400,400);
const char *const keycode = "TAB";
while (!disp.is_closed()) {
    if (disp.is_key(keycode)) { ... } // Equivalent to 'if (disp.is_keyTAB())'
    disp.wait();
}
```

### 8.2.3.12 is_key_sequence()

```cpp
bool is_key_sequence {
    const unsigned int *const keycodes_sequence,
    const unsigned int length,
    const bool remove_sequence = false}
```

**Return** `true` if specified key sequence has been typed on the associated window, `false` otherwise.

**Parameters**

<table>
<thead>
<tr>
<th><code>keycodes_sequence</code></th>
<th>Buffer of keycodes to test.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>length</code></td>
<td>Number of keys in the <code>keycodes_sequence</code> buffer.</td>
</tr>
<tr>
<td><code>remove_sequence</code></td>
<td>Tells if the key sequence must be removed from the key history, if found.</td>
</tr>
</tbody>
</table>
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 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()

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()

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()

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()

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()

const char* title ( ) const

Return title of the associated window as a C-string.

Note

Window title may be not visible, depending on the used window manager or if the current display is in fullscreen mode.

8.2.3.18 window_width()

int window_width ( ) const

Return width of the associated window.

Note

The width of the display (i.e. the width of the pixel data buffer associated to the CImgDisplay instance) may be different from the actual width of the associated window.
8.2.3.19 window_height()

```cpp
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()

```cpp
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()

```cpp
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()

```cpp
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()

```cpp
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 \([0,\text{height}()-1]\).

8.2.3.24  button()

```cpp
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**

```cpp
CImgDisplay disp(400,400);
while (!disp.is_closed()) {
    if (disp.button()&1) { // Left button clicked
        ...
    }
    if (disp.button()&2) { // Right button clicked
        ...
    }
    if (disp.button()&4) { // Middle button clicked
        ...
    }
    disp.wait();
}
```
8.2.3.25 wheel()

int wheel() const

Return current state of the mouse wheel.

**Note**
- The returned value can be positive or negative depending on whether the mouse wheel has been scrolled forward or backward.
- Scrolling the wheel forward add 1 to the wheel value.
- Scrolling the wheel backward subtract 1 to the wheel value.
- The returned value cumulates the number of forward of backward scrolls since the creation of the display, or since the last reset of the wheel value (using `set_wheel()`). It is strongly recommended to quickly reset the wheel counter when an action has been performed regarding the current wheel value. Otherwise, the returned wheel value may be for instance 0 despite the fact that many scrolls have been done (as many in forward as in backward directions).

**Example**
```cpp
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()

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()

```cpp
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.

8.2.3.28 keycode()

```cpp
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

```cpp
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). |
Note

This method returns immediately.

8.2.3.32  show()

CImgDisplay& show ( )

Show (closed) associated window on the screen.

Note

- Force the associated window of a display to be visible on the screen, even if it has been closed before.
- Using show() on a visible display does nothing.

8.2.3.33  close()

CImgDisplay& close ( )

Close (visible) associated window and make it disappear from the screen.

Note

- A closed display only means the associated window is not visible anymore. This does not mean the display has been destroyed. Use show() to make the associated window reappear.
- Using close() on a closed display does nothing.

8.2.3.34  move()

CImgDisplay& move ( 
    const int pos_x,
    const int pos_y )

Move associated window to a new location.

Parameters

<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>
8.2 CImgDisplay Struct Reference

Note

Depending on the window manager behavior, this method may not succeed (no exceptions are thrown Nevertheless).

8.2.3.35 resize() [1/4]

CImgDisplay& resize (const bool force_redraw = true)

Resize display to the size of the associated window.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>force_redraw</td>
<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]

CImgDisplay& resize (const int width, const int height, const bool force_redraw = true)

Resize display to the specified size.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>width</td>
<td>Requested display width.</td>
</tr>
<tr>
<td>height</td>
<td>Requested display height.</td>
</tr>
<tr>
<td>force_redraw</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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>img</td>
<td>Input image to take size from.</td>
</tr>
<tr>
<td>force_redraw</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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>disp</td>
<td>Input display to take size from.</td>
</tr>
<tr>
<td>force_redraw</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

```cpp
disp.set_title("%s",window_title);
```

instead of

```cpp
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 if 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()

\texttt{CImgDisplay} \texttt{set\_mouse} (  
  \texttt{const int pos\_x},  
  \texttt{const int pos\_y} )

Move mouse pointer to a specified location.

\textbf{Note}
Depending on the window manager behavior, this method may not succeed (no exceptions are thrown nevertheless).

8.2.3.46  set_button\([1/2]\)

\texttt{CImgDisplay} \texttt{set\_button} ( )

Simulate a mouse button release event.

\textbf{Note}
All mouse buttons are considered released at the same time.

8.2.3.47  set_button\([2/2]\)

\texttt{CImgDisplay} \texttt{set\_button} (  
  \texttt{const unsigned int button},  
  \texttt{const bool is\_pressed = true} )

Simulate a mouse button press or release event.

\textbf{Parameters}

<table>
<thead>
<tr>
<th>\textit{button}</th>
<th>Buttons event code, where each button is associated to a single bit.</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{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\([1/2]\)

\texttt{CImgDisplay} \texttt{set\_wheel} ( )

Flush all mouse wheel events.
8.2.3.49 set_wheel() [2/2]

CImgDisplay & set_wheel {
  const int amplitude
}

Simulate a wheel event.

**Parameters**

- **amplitude** | Amplitude of the wheel scrolling to simulate.

**Note**

Make `wheel()` to return `0`, if called afterwards.

8.2.3.50 set_key() [1/2]

CImgDisplay & set_key ( )

Flush all key events.

**Note**

Make `key()` to return `0`, if called afterwards.

8.2.3.51 set_key() [2/2]

CImgDisplay & set_key {
  const unsigned int keycode,
  const bool is_pressed = true
}

Simulate a keyboard press/release event.

**Parameters**

- **keycode** | Keycode of the associated key.
- **is_pressed** | Tells if the key is considered as pressed or released.
Note
Keycode constants are defined in the cimg namespace and are architecture-dependent. Use them to ensure your code stay portable (see cimg::keyESC).

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()

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]

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>Output screenshot. Can be empty on input</td>
</tr>
</tbody>
</table>

8.2.3.57  snapshot()

const CImgDisplay& snapshot (  
    CImg< T >& img ) const

Take a snapshot of the associated window content.
### 8.3 CImgException Struct Reference

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

Inherits exception.

Inherited by `CImgAbortException`, `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 errord 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`:

  ```cpp
  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 `CImgException`:

  ```cpp
  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`:

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out</code></td>
<td><code>img</code></td>
</tr>
</tbody>
</table>
```
const CImg<float> img("missing_file.jpg"); // Try to load a file that does not exist

• CImgWarningException: Thrown only if configuration macro cimg_strict_warnings is set, and when a CImg function has to display a warning message (see cimg::warn()).

It is not recommended to throw CImgException instances by yourself, since they are expected to be thrown only by CImg. When an error occurs in a library function call, CImg may display error messages on the screen or on the standard output, depending on the current CImg exception mode. The CImg exception mode can be get and set by functions cimg::exception_mode() and cimg::exception_mode(unsigned int).

Exceptions handling

In all cases, when an error occurs in CImg, an instance of the corresponding exception class is thrown. This may lead the program to break (this is the default behavior), but you can bypass this behavior by handling the exceptions by yourself, using a usual try { ... } catch () { ... } bloc, as in the following example:

```
#define "CImg.h"
using namespace cimg_library;
int main() {
  cimg::exception_mode(0); // Enable quiet exception mode
  try {
    ... // Here, do what you want to stress CImg
    ... // You succeeded: something went
    catch (CImgException& e) { // Display your custom error message
      std::fprintf(stderr,"CImg Library Error: %s",e.what());
      ... // 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

- `~CImgList()`
  
  Destructor.

- `CImgList()`
  
  Default constructor.

- `CImgList(const unsigned int n)`
  
  Construct list containing empty images.

- `CImgList(const unsigned int n, const unsigned int width, const unsigned int height=1, const unsigned int depth=1, const unsigned int spectrum=1)`
  
  Construct list containing images of specified size.

- `CImgList(const unsigned int n, const unsigned int width, const unsigned int height, const unsigned int depth, const unsigned int spectrum, const T &val)`
  
  Construct list containing images of specified size, and initialize pixel values.

- `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.

- `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.

- `template<typename t>
  CImgList(const unsigned int n, const CImg<t> &img, const bool is_shared=false)`
  
  Construct list containing copies of an input image.

- `template<typename t>
  CImgList(const Clmg<t> &img, const bool is_shared=false)`
  
  Construct list from one image.

- `template<typename t1 , typename t2 >
  CImgList(const Clmg<t1> &img1, const Clmg<t2> &img2, const bool is_shared=false)`
  
  Construct list from two images.

- `template<typename t1 , typename t2 , typename t3 >
  CImgList(const Clmg<t1> &img1, const Clmg<t2> &img2, const Clmg&t3> &img3, const bool is_shared=false)`
  
  Construct list from three images.

- `template<typename t1 , typename t2 , typename t3 , typename t4 >
  CImgList(const Clmg<t1> &img1, const Clmg&t2> &img2, const Clmg<t3> &img3, const Clmg<t4> &img4, const bool is_shared=false)`
  
  Construct list from four images.

- `template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 >
  CImgList(const Clmg<t1> &img1, const Clmg&t2> &img2, const Clmg<t3> &img3, const Clmg<t4> &img4, const Clmg&t5> &img5, const bool is_shared=false)`
  
  Construct list from five images.

- `template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 , typename t6 >
  CImgList(const Clmg<t1> &img1, const Clmg&t2> &img2, const Clmg&t3> &img3, const Clmg&t4> &img4, const Clmg&t5> &img5, const Clmg&t6> &img6, const bool is_shared=false)`
  
  Construct list from six images.

- `template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 , typename t6 , typename t7 >
  CImgList(const Clmg<t1> &img1, const Clmg&t2> &img2, const Clmg&t3> &img3, const Clmg&t4> &img4, const Clmg&t5> &img5, const Clmg&t6> &img6, const Clmg&t7> &img7, const bool is_shared=false)`
  
  Construct list from seven images.
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 &val, const T &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 unsigned int n, 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 CImg<t4> &img4, 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].
8.4 CImgList < T > Struct Template Reference

- template<typename T1, typename T2, typename T3, typename T4, typename T5>
  CImgList < T > & assign (const CImg < T1 > & img1, const CImg < T2 > & img2, const CImg < T3 > & img3,
  const CImg < T4 > & img4, const CImg < T5 > & img5, const bool is_shared=false)

  Construct list from five images [in-place version].

- template<typename T1, typename T2, typename T3, typename T4, typename T5, typename T6>
  CImgList < T > & assign (const CImg < T1 > & img1, const CImg < T2 > & img2, const CImg < T3 > & img3,
  const CImg < T4 > & img4, const CImg < T5 > & img5, const CImg < T6 > & img6, const bool is_shared=false)

  Construct list from six images [in-place version].

- template<typename T1, typename T2, typename T3, typename T4, typename T5, typename T6, typename T7>
  CImgList < T > & assign (const CImg < T1 > & img1, const CImg < T2 > & img2, const CImg < T3 > & img3,
  const CImg < T4 > & img4, const CImg < T5 > & img5, const CImg < T6 > & img6, const CImg < T7 > & img7,
  const bool is_shared=false)

  Construct list from seven images [in-place version].

- template<typename T1, typename T2, typename T3, typename T4, typename T5, typename T6, typename T7, typename T8>
  CImgList < T > & assign (const CImg < T1 > & img1, const CImg < T2 > & img2, const CImg < T3 > & img3,
  const CImg < T4 > & img4, const CImg < T5 > & img5, const CImg < T6 > & img6, const CImg < T7 > & img7,
  const CImg < T8 > & img8, const bool is_shared=false)

  Construct list from eight images [in-place version].

- template<typename T>
  CImgList < T > & assign (const CImgList < T > & list, const bool is_shared=false)

  Construct list as a copy of an existing list and force the shared state of the list elements [in-place version].

- template<typename T>
  CImgList < T > & assign (const CImgList < T > & list, const bool is_shared=false)

  Construct list as a copy of an existing list and force shared state of elements [in-place version] [specialization].

- template<typename T>
  CImgList < T > & assign (const char *const filename)

  Construct list by reading the content of a file [in-place version].

- template<typename T>
  CImgList < T > & assign (const CImgDisplay & disp)

  Construct list from the content of a display window [in-place version].

- template<typename T>
  CImgList < T > & assign (const CImgList < T > & list)

  Transfer the content of the list instance to another list.

- template<typename T>
  CImgList < T > & assign (const CImgList < T > & list, const unsigned int pos)

  Transfer the content of the list instance at a specified position in another list.

- template<typename T>
  CImgList < T > & swap (CImgList < T > & list)

  Swap all fields between two list instances.

- static CImgList < T > & empty ()

  Return a reference to an empty list.

- static const CImgList < T > & const_empty ()

  Return a reference to an empty list [const version].

### Overloaded Operators

- CImgList < T > & operator() (const unsigned int pos)

  Return a reference to one image element of the list.

- CImgList < T > & operator() (const unsigned int pos) const

  Return a reference to one image of the list.

- T & operator() (const unsigned int pos, const unsigned int x, const unsigned int y=0, const unsigned int z=0,
  const unsigned int c=0)

  Return a reference to one pixel value of one image of the list.

- const T & operator() (const unsigned int pos, const unsigned int x, const unsigned int y=0, const unsigned int z=0,
  const unsigned int c=0) const

  Return a reference to one pixel value of one image of the list [const version].
• operator CImg\< T \> * ()
  
  Return pointer to the first image of the list.

• operator const CImg\< T \> * () const
  
  Return pointer to the first image of the list [const version].

• template<typename t >
  CImgList\< T \> & operator= (const CImg\< t \> &img)
  
  Construct list from one image [in-place version].

• template<typename t >
  CImgList\< T \> & operator= (const CImgList\< T \> &list)
  
  Construct list from another list.

• CImgList\< T \> & operator= (const char *const filename)
  
  Construct list by reading the content of a file [in-place version].

• CImg\< T \> operator+ () const
  
  Return a non-shared copy of a list.

• template<typename t >
  CImgList\< T \> & operator, (const CImg\< t \> &img)
  
  Return a copy of the list instance, where image \texttt{img} has been inserted at the end.

• template<typename t >
  CImgList\< T \> & operator, (const CImg\< t \> &img) const
  
  Return a copy of the list instance, where image \texttt{img} has been inserted at the end [const version].

• template<typename t >
  CImgList\< T \> & operator, (const CImgList\< T \> &list)
  
  Return a copy of the list instance, where all elements of input list \texttt{list} have been inserted at the end.

• template<typename t >
  CImgList\< T \> & operator, (const CImgList\< T \> &list) const
  
  Return a copy of the list instance, where all elements of input \texttt{list} have been inserted at the end [const version].

• CImg\< T \> operator> (const char axis) const
  
  Return image corresponding to the appending of all images of the instance list along specified axis.

• CImgList\< T \> operator< (const char axis) const
  
  Return list corresponding to the splitting of all images of the instance list along specified axis.

Instance Characteristics

• int width () const
  
  Return the size of the list, i.e. the number of images contained in it.

• unsigned int size () const
  
  Return the size of the list, i.e. the number of images contained in it.

• CImg\< T \> * data ()
  
  Return pointer to the first image of the list.

• const CImg\< T \> * data () const
  
  Return pointer to the first image of the list [const version].

• CImg\< T \> * data (const unsigned int pos)
  
  Return pointer to the pos-th image of the list.

• const CImg\< T \> * data (const unsigned int l) const

• iterator begin ()
  
  Return iterator to the first image of the list.

• const_iterator begin () const
Return iterator to the first image of the list [const version].
- iterator end()
  Return iterator to one position after the last image of the list.
- const_iterator end() const
  Return iterator to one position after the last image of the list [const version].
- Clmg<T>& front()
  Return reference to the first image of the list.
- const Clmg<T>& front() const
  Return reference to the first image of the list [const version].
- Clmg<T>& back()
  Return a reference to the last image of the list.
- const Clmg<T>& back() const
  Return a reference to the last image of the list [const version].
- Clmg<T>& at(const int pos)
  Return pos-th image of the list.
- T& atNXYZC(const int pos, const int x, const int y, const int z, const int c, const T& out_value)
  Access to pixel value with Dirichlet boundary conditions.
- T atNXYZC(const int pos, const int x, const int y, const int z, const int c, const T& out_value) const
  Access to pixel value with Dirichlet boundary conditions [const version].
- T& atNXYZC(const int pos, const int x, const int y, const int z, const int c)
  Access to pixel value with Neumann boundary conditions.
- T atNXYZC(const int pos, const int x, const int y, const int z, const int c, const T& out_value) const
  Access to pixel value with Neumann boundary conditions [const version].
- T& atNXYZ(const int pos, const int x, const int y, const int z, const int c, const T& out_value)
  Access to pixel value with Dirichlet boundary conditions for the 3 coordinates (pos, x, y, z).
- T atNXYZ(const int pos, const int x, const int y, const int z, const int c, const T& out_value) const
  Access to pixel value with Dirichlet boundary conditions for the 3 coordinates (pos, x, y, z) [const version].
- T& atNXYZ(const int pos, const int x, const int y, const int z=0, const int c=0)
  Access to pixel value with Neumann boundary conditions for the 3 coordinates (pos, x, y).
- T atNXYZ(const int pos, const int x, const int y=0, const int z=0, const int c=0) const
  Access to pixel value with Neumann boundary conditions for the 3 coordinates (pos, x, y) [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].

Generated by Doxygen
- \texttt{T \& atN (const int pos, const int x=0, const int y=0, const int z=0, const int c=0)}
  \textit{Return pixel value with Neumann boundary conditions for the coordinate (pos).}
- \texttt{T atN (const int pos, const int x=0, const int y=0, const int z=0, const int c=0) const}
  \textit{Return pixel value with Neumann boundary conditions for the coordinate (pos) [const version].}
- \texttt{static const char \* pixel_type ()}
  \textit{Return the type of image pixel values as a C string.}

\textbf{Instance Checking}

- \texttt{bool is_empty () const}
  \textit{Return true if list is empty.}
- \texttt{bool is_sameN (const unsigned int size_n) const}
  \textit{Test if number of image elements is equal to specified value.}
- \texttt{template<typename t> bool is_sameXY (const CImgList< t > \&list) const}
  \textit{Test if number of image elements is equal between two images lists.}
- \texttt{template<typename t> bool is_sameXZ (const CImgList< t > \&list) const}
- \texttt{template<typename t> bool is_sameNXZ (const unsigned int n, const CImg\< t \> \&img) const}
- \texttt{template<typename t> bool is_sameYX (const CImgList< t > \&list) const}
- \texttt{template<typename t> bool is_sameNYX (const unsigned int n, const CImg\< t \> \&img) const}
- \texttt{template<typename t> bool is_sameNYZ (const unsigned int n, const CImg\< t \> \&img) const}
- \texttt{template<typename t> bool is_sameNYC (const unsigned int n, const CImg\< t \> \&img) const}
8.4 ClmgList< T > Struct Template Reference

- template<typename t>
  bool is_sameNYC (const ClmgList< t >&list) const
- template<typename t>
  bool is_sameXYZ (const Clmg< t >&img) const
- template<typename t>
  bool is_sameXYZ (const ClmgList< t >&list) const
- template<typename t>
  bool is_sameNXYZ (const unsigned int n, const Clmg< t >&img) const
- template<typename t>
  bool is_sameNXYZ (const ClmgList< t >&list) const
- template<typename t>
  bool is_sameXYC (const Clmg< t >&img) const
- template<typename t>
  bool is_sameXYC (const ClmgList< t >&list) const
- template<typename t>
  bool is_sameNXYC (const unsigned int n, const Clmg< t >&img) const
- template<typename t>
  bool is_sameNXYC (const ClmgList< t >&list) const
- template<typename t>
  bool is_sameYZC (const Clmg< t >&img) const
- template<typename t>
  bool is_sameYZC (const ClmgList< t >&list) const
- template<typename t>
  bool is_sameNYZC (const unsigned int n, const Clmg< t >&img) const
- template<typename t>
  bool is_sameNYZC (const ClmgList< t >&list) const
- template<typename t>
  bool is_sameXYZC (const Clmg< t >&img) const
- template<typename t>
  bool is_sameXYZC (const ClmgList< t >&list) const
- template<typename t>
  bool is_sameNXYZC (const unsigned int n, const Clmg< t >&img) const
- template<typename t>
  bool is_sameNXYZC (const ClmgList< t >&list) const
- bool is_sameX (const unsigned int val) const
- bool is_sameNX (const unsigned int n, const unsigned int val) const
- bool is_sameY (const unsigned int val) const
- bool is_sameNY (const unsigned int n, const unsigned int val) const
- bool is_sameZ (const unsigned int val) const
- bool is_sameNZ (const unsigned int n, const unsigned int val) const
- bool is_sameC (const unsigned int val) const
- bool is_sameNC (const unsigned int n, const unsigned int val) const
- bool is_sameXY (const unsigned int val1, const unsigned int val2) const
- bool is_sameNXY (const unsigned int n, const unsigned int val1, const unsigned int val2) const
- bool is_sameXZ (const unsigned int val1, const unsigned int val2) const
- bool is_sameNXZ (const unsigned int n, const unsigned int val1, const unsigned int val2) const
- bool is_sameXC (const unsigned int val1, const unsigned int val2) const
- bool is_sameNXC (const unsigned int n, const unsigned int val1, const unsigned int val2) const
- bool is_sameYZ (const unsigned int val1, const unsigned int val2) const
- bool is_sameNYZ (const unsigned int n, const unsigned int val1, const unsigned int val2) const
- bool is_sameYC (const unsigned int val1, const unsigned int val2) const
- bool is_sameNYC (const unsigned int n, const unsigned int val1, const unsigned int val2) const
- bool is_sameZC (const unsigned int val1, const unsigned int val2) const
- bool is_sameNZC (const unsigned int n, const unsigned int val1, const unsigned int val2) const
- bool is_sameXYZ (const unsigned int val1, const unsigned int val2, const unsigned int val3) const
• bool is_sameNXYZ (const unsigned int n, const unsigned int val1, const unsigned int val2, const unsigned int val3) const
• bool is_sameXYC (const unsigned int val1, const unsigned int val2, const unsigned int val3) const
• bool is_sameNXYC (const unsigned int n, const unsigned int val1, const unsigned int val2, const unsigned int val3) const
• bool is_sameXZC (const unsigned int val1, const unsigned int val2, const unsigned int val3) const
• bool is_sameNXZC (const unsigned int n, const unsigned int val1, const unsigned int val2, const unsigned int val3) const
• bool is_sameYZC (const unsigned int val1, const unsigned int val2, const unsigned int val3) const
• bool is_sameNYZC (const unsigned int n, const unsigned int val1, const unsigned int val2, const unsigned int val3) const
• bool is_sameXYZC (const unsigned int dx, const unsigned int dy, const unsigned int dz, const unsigned int dc) const

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 CImg<T> &img, t &n) const

Test if the list contains the image 'img'.

• template<typename t>
  bool contains (const CImg<T> &img) const

Test if the list contains the image img.

Mathematical Functions

• T & min ()

Return a reference to the minimum pixel value of the instance list.

• const T & min () const

Return a reference to the minimum pixel value of the instance list [const version].

• T & max ()

Return a reference to the maximum pixel value of the instance list.
• const T & max () const  
  Return a reference to the maximum pixel value of the instance list [const version].

• template<typename t>
  T & min_max (t &max_val)
  Return a reference to the minimum pixel value of the instance list and return the maximum value as well.

• template<typename t>
  const T & min_max (t &max_val) const
  Return a reference to the minimum pixel value of the instance list and return the maximum value as well [const version].

• template<typename t>
  T & max_min (t &min_val)
  Return a reference to the 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 unsigned int n, const unsigned int pos=∼0U)
  Insert `n` empty images into the current image list, at position `pos`.

• CImgList< T > get_insert (const unsigned int n, const unsigned int pos=∼0U) const
  Insert `n` empty images into the current image list, at position `pos` [new-instance version].

• template<typename t>
  CImgList< T > & insert (const unsigned int n, const CImg< t > &img, const unsigned int pos=∼0U, const bool is_shared=false)
  Insert `n` copies of the image `img` into the current image list, at position `pos`.

• template<typename t>
  CImgList< T > get_insert (const unsigned int n, const CImg< t > &img, const unsigned int pos=∼0U, const bool is_shared=false) const
  Insert `n` copies of the image `img` into the current image list, at position `pos` [new-instance version].

• template<typename t>
  CImgList< T > & insert (const CImgList< t > &list, const unsigned int pos=∼0U, const bool is_shared=false)
  Insert a copy of the image list `list` into the current image list, at 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, at position `pos` [new-instance version].

• template<typename t>
  CImgList< T > & insert (const unsigned int n, const CImgList< t > &list, const unsigned int pos=∼0U, const bool is_shared=false)
  Insert `n` copies of the list `list` at position `pos` of the current list.
• template<typename t >
  CImgList<T> get_insert (const unsigned int n, const CImgList<T> &list, const unsigned int pos=0U, const bool is_shared=false) const
  Insert n copies of the list list at position pos of the current list [new-instance version].

• CImgList<T> & remove (const unsigned int pos1, const unsigned int pos2)
  Remove all images between from indexes.

• CImgList<T> get_remove (const unsigned int pos1, const unsigned int pos2) const
  Remove all images between from indexes [new-instance version].

• CImgList<T> & remove (const unsigned int pos)
  Remove image at index pos from the image list.

• CImgList<T> get_remove (const unsigned int pos) const
  Remove image at index pos from the image list [new-instance version].

• CImgList<T> & remove ()
  Remove last image.

• CImgList<T> get_remove () const
  Remove last image [new-instance version].

• CImgList<T> & reverse ()
  Reverse list order.

• CImgList<T> get_reverse () const
  Reverse list order [new-instance version].

• CImgList<T> & images (const unsigned int pos0, const unsigned int pos1)
  Return a sublist.

• CImgList<T> get_images (const unsigned int pos0, const unsigned int pos1) const
  Return a sublist [new-instance version].

• CImgList<T> get_shared_images (const unsigned int pos0, const unsigned int pos1) const
  Return a shared sublist.

• const CImgList<T> get_shared_images (const unsigned int pos0, const unsigned int pos1) const
  Return a shared sublist [new-instance version].

• CImg<T> get_append (const char axis, const float align=0) const
  Return a single image which is the 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> & push_back (const CImg<T> &img)
  Insert image at the end of the list.

• template<typename t >
  CImgList<T> & push_front (const CImg<T> &img)
  Insert image at the front of the list.

• template<typename t >
  CImgList<T> & push_back (const CImgList<T> &list)
  Insert list at the end of the current list.

• template<typename t >
  CImgList<T> & push_front (const CImgList<T> &list)
  Insert list at the front of the current list.

• CImgList<T> & pop_back ()
  Remove last image.

• CImgList<T> & pop_front ()
  Remove first image.

• CImgList<T> & erase (const iterator iter)
  Remove image pointed by iterator.
Data Input

- \(\texttt{CImg< intT > get\_select (CImgDisplay &disp, const bool feature\_type=true, const char axis='x', const float align=0, const bool exit\_on\_anykey=false) const}\)
  
  Display a simple interactive interface to select images or sublists.

- \(\texttt{CImg< intT > get\_select (const char *const title, const bool feature\_type=true, const char axis='x', const float align=0, const bool exit\_on\_anykey=false) const}\)
  
  Display a simple interactive interface to select images or sublists.

- \(\texttt{CImgList<T> & load (const char *const filename)}\)
  
  Load a list from a file.

- \(\texttt{CImgList<T> & load\_cimg (const char *const filename)}\)
  
  Load a list from a .cimg file.

- \(\texttt{CImgList<T> & load\_cimg (std::FILE *const file)}\)
  
  Load a list from a .cimg file.

- \(\texttt{CImgList<T> & load\_cimg (const char *const filename, const unsigned int n0, const unsigned int n1, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0, const unsigned int x1, const unsigned int y1, const unsigned int z1, const unsigned int c1)}\)
  
  Load a sublist list from a (non compressed) .cimg file.

- \(\texttt{CImgList<T> & load\_cimg (std::FILE *const file, const unsigned int n0, const unsigned int n1, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0, const unsigned int x1, const unsigned int y1, const unsigned int z1, const unsigned int c1)}\)
  
  Load a sub-image list from a (non compressed) .cimg file [overloading].

- \(\texttt{CImgList<T> & load\_parrec (const char *const filename)}\)
  
  Load a list from a PAR/REC (Philips) file.

- \(\texttt{CImgList<T> & load\_yuv (const char *const filename, const unsigned int size\_x, const unsigned int size\_y, const unsigned int chroma\_subsampling=444, const unsigned int first\_frame=0, const unsigned int last\_frame=\sim0U, const unsigned int step\_frame=1, const bool yuv2rgb=true)}\)
  
  Load a list from a YUV image sequence file.

- \(\texttt{CImgList<T> & load\_yuv (std::FILE *const file, const unsigned int size\_x, const unsigned int size\_y, const unsigned int chroma\_subsampling=444, const unsigned int first\_frame=0, const unsigned int last\_frame=\sim0U, const unsigned int step\_frame=1, const bool yuv2rgb=true)}\)
  
  Load a list from an image sequence YUV file [overloading].

- \(\texttt{CImgList<T> & load\_video (const char *const filename, const unsigned int first\_frame=0, const unsigned int last\_frame=\sim0U, const unsigned int step\_frame=1)}\)
  
  Load an image from a video file, using OpenCV library.

- \(\texttt{CImgList<T> & load\_ffmpeg\_external (const char *const filename)}\)
  
  Load an image from a video file using the external tool 'ffmpeg'.

- \(\texttt{CImgList<T> & load\_gif\_external (const char *const filename)}\)
  
  Load gif file, using ImageMagick or GraphicsMagick’s external tools.

- \(\texttt{CImgList<T> & load\_gzip\_external (const char *const filename)}\)
  
  Load a gzipped list, using external tool ‘gunzip’.

- \(\texttt{CImgList<T> & load\_tiff (const char *const filename, const unsigned int first\_frame=0, const unsigned int last\_frame=\sim0U, const unsigned int step\_frame=1, float *const voxel\_size=0, CImg< charT > *const description=0)}\)
  
  Load images from a TIFF file.

- \(\texttt{static CImgList<T> & get\_load (const char *const filename)}\)
  
  Load a list from a file [new-instance version].

- \(\texttt{static CImgList<T> & get\_load\_cimg (const char *const filename)}\)
  
  Load a list from a .cimg file [new-instance version].

- \(\texttt{static CImgList<T> & get\_load\_cimg (std::FILE *const file)}\)
  
  Load a list from a .cimg file [new-instance version].
• static `CImgList<T>`\> get_load_cimg (const char *const filename, const unsigned int n0, const unsigned int n1, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0, const unsigned int x1, const unsigned int y1, const unsigned int z1, const unsigned int c1)
  Load a sublist list from a (non compressed) .cimg file [new-instance version].

• static `CImgList<T>`\> get_load_cimg (std::FILE *const file, const unsigned int n0, const unsigned int n1, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0, const unsigned int x1, const unsigned int y1, const unsigned int z1, const unsigned int c1)
  Load a sub-image list from a (non compressed) .cimg file [new-instance version].

• static `CImgList<T>`\> get_load_parrec (const char *const filename)
  Load a list from a PAR/REC (Philips) file [new-instance version].

• static `CImgList<T>`\> get_load_yuv (const char *const filename, const unsigned int size_x, const unsigned int size_y=1, const unsigned int 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< char T > *const description=0)
  Load a multi-page TIFF file [new-instance version].

Data Output

• const `CImgList<T>`\> & print (const char *const title=0, const bool display_stats=true) const
  Print information about the list on the standard output.

• const `CImgList<T>`\> & display (CImgDisplay &disp, const char axis='x', const float align=0) const
  Display the current `CImgList` instance in an existing CImgDisplay window (by reference).

• const `CImgList<T>`\> & display (CImgDisplay &disp, const bool display_info, const char axis='x', const float align=0, unsigned int *const XYZ=0, const bool exit_on_anykey=false) const
  Display the current `CImgList` instance in a new display window.

• const `CImgList<T>`\> & display (const char *const title=0, const bool display_info=true, const char axis='x', const float align=0, unsigned int *const XYZ=0, const bool exit_on_anykey=false) const
  Display the current `CImgList` instance in a new display window.

• const `CImgList<T>`\> & save (const char *const filename, const int number=-1, const unsigned int digits=6) const
  Save list into a file.

• const `CImgList<T>`\> & save_gif_external (const char *const filename=0, const unsigned int chroma_subsampling=444, const bool is_rgb=true) const
  Save image sequence as a GIF animated file.
Save list as a YUV image sequence file.
• const CImgList< T > & save_yuv (std::FILE *const file, const unsigned int chroma_subsampling=444, const bool is_rgb=true) const
  Save 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 * const file, const bool is_compressed=false) const
  Insert the image instance into an existing .cimg file, at specified coordinates.
• 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'.
• 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.
• 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

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
typedef CImg<T>* iterator

Simple iterator type, to loop through each image of a list.

Note
  • The CImgList<T>::iterator type is defined as a CImg<T>*.
  • You may use it like this:
    CImgList<> list; // Assuming this image list is not empty
    for (CImgList<>::iterator it = list.begin(); it<list.end(); ++it) (*it).mirror('x');
  • Using the loop macro cimglist_for is another (more concise) alternative:
    cimglist_for(list,l) list[l].mirror('x');

8.4.2.2 const_iterator
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>Number of empty images.</td>
</tr>
</tbody>
</table>

Note

Useful when you know in advance the number of images you want to manage, as it will allocate the right amount of memory for the list, without the need for reallocation (that may occur when starting from an empty list and inserting several images in it).

### 8.4.3.4 CImgList()

```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

<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>
</tbody>
</table>

Note

Pixel values are not initialized and may probably contain garbage.

### 8.4.3.5 CImgList()

```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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><em>n</em></td>
<td>Number of images.</td>
</tr>
<tr>
<td><em>width</em></td>
<td>Width of images.</td>
</tr>
<tr>
<td><em>height</em></td>
<td>Height of images.</td>
</tr>
<tr>
<td><em>depth</em></td>
<td>Depth of images.</td>
</tr>
<tr>
<td><em>spectrum</em></td>
<td>Number of channels of images.</td>
</tr>
<tr>
<td><em>val</em></td>
<td>Initialization value for images pixels.</td>
</tr>
</tbody>
</table>

### 8.4.3.6 CImgList()

```cpp
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**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><em>n</em></td>
<td>Number of images.</td>
</tr>
<tr>
<td><em>width</em></td>
<td>Width of images.</td>
</tr>
<tr>
<td><em>height</em></td>
<td>Height of images.</td>
</tr>
<tr>
<td><em>depth</em></td>
<td>Depth of images.</td>
</tr>
<tr>
<td><em>spectrum</em></td>
<td>Number of channels of images.</td>
</tr>
<tr>
<td><em>val0</em></td>
<td>First value of the initializing integers sequence.</td>
</tr>
<tr>
<td><em>val1</em></td>
<td>Second value of the initializing integers sequence.</td>
</tr>
</tbody>
</table>

**Warning**

You must specify at least \( \text{width} \times \text{height} \times \text{depth} \times \text{spectrum} \) values in your argument list, or you will probably segfault.

### 8.4.3.7 CImgList()

```cpp
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></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*height*depth*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></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.
8.4 CImgList< T > Struct Template Reference

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,
    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>
Construct list from four images.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>img1</code></td>
<td>First input image to copy in the constructed list.</td>
</tr>
<tr>
<td><code>img2</code></td>
<td>Second input image to copy in the constructed list.</td>
</tr>
<tr>
<td><code>img3</code></td>
<td>Third input image to copy in the constructed list.</td>
</tr>
<tr>
<td><code>img4</code></td>
<td>Fourth input image to copy in the constructed list.</td>
</tr>
<tr>
<td><code>is_shared</code></td>
<td>Tells if the elements of the list are shared or non-shared copies of input images.</td>
</tr>
</tbody>
</table>

8.4.3.13 `CImgList()` [12/19]

`CImgList` ()

```cpp
    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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>img1</code></td>
<td>First input image to copy in the constructed list.</td>
</tr>
<tr>
<td><code>img2</code></td>
<td>Second input image to copy in the constructed list.</td>
</tr>
<tr>
<td><code>img3</code></td>
<td>Third input image to copy in the constructed list.</td>
</tr>
<tr>
<td><code>img4</code></td>
<td>Fourth input image to copy in the constructed list.</td>
</tr>
<tr>
<td><code>img5</code></td>
<td>Fifth input image to copy in the constructed list.</td>
</tr>
<tr>
<td><code>is_shared</code></td>
<td>Tells if the elements of the list are shared or non-shared copies of input images.</td>
</tr>
</tbody>
</table>

8.4.3.14 `CImgList()` [13/19]

`CImgList` ()

```cpp
    const CImg< t1 > & img1,
    const CImg< t2 > & img2,
    const CImg< t3 > & img3,
    const CImg< t4 > & img4,
    const CImg< t5 > & img5,
    const CImg< t6 > & img6,
    const bool is_shared = false }
```

Construct list from six images.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>img1</em></td>
<td>First input image to copy in the constructed list.</td>
</tr>
<tr>
<td><em>img2</em></td>
<td>Second input image to copy in the constructed list.</td>
</tr>
<tr>
<td><em>img3</em></td>
<td>Third input image to copy in the constructed list.</td>
</tr>
<tr>
<td><em>img4</em></td>
<td>Fourth input image to copy in the constructed list.</td>
</tr>
<tr>
<td><em>img5</em></td>
<td>Fifth input image to copy in the constructed list.</td>
</tr>
<tr>
<td><em>img6</em></td>
<td>Sixth input image to copy in the constructed list.</td>
</tr>
<tr>
<td><em>is_shared</em></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()  [14/19]

```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><em>img1</em></td>
<td>First input image to copy in the constructed list.</td>
</tr>
<tr>
<td><em>img2</em></td>
<td>Second input image to copy in the constructed list.</td>
</tr>
<tr>
<td><em>img3</em></td>
<td>Third input image to copy in the constructed list.</td>
</tr>
<tr>
<td><em>img4</em></td>
<td>Fourth input image to copy in the constructed list.</td>
</tr>
<tr>
<td><em>img5</em></td>
<td>Fifth input image to copy in the constructed list.</td>
</tr>
<tr>
<td><em>img6</em></td>
<td>Sixth input image to copy in the constructed list.</td>
</tr>
<tr>
<td><em>img7</em></td>
<td>Seventh input image to copy in the constructed list.</td>
</tr>
<tr>
<td><em>is_shared</em></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()  [15/19]

```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,  
```
Construct list from eight images.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>img1</code></td>
<td>First input image to copy in the constructed list.</td>
</tr>
<tr>
<td><code>img2</code></td>
<td>Second input image to copy in the constructed list.</td>
</tr>
<tr>
<td><code>img3</code></td>
<td>Third input image to copy in the constructed list.</td>
</tr>
<tr>
<td><code>img4</code></td>
<td>Fourth input image to copy in the constructed list.</td>
</tr>
<tr>
<td><code>img5</code></td>
<td>Fifth input image to copy in the constructed list.</td>
</tr>
<tr>
<td><code>img6</code></td>
<td>Sixth input image to copy in the constructed list.</td>
</tr>
<tr>
<td><code>img7</code></td>
<td>Seventh input image to copy in the constructed list.</td>
</tr>
<tr>
<td><code>img8</code></td>
<td>Eighth input image to copy in the constructed list.</td>
</tr>
<tr>
<td><code>is_shared</code></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]

```cpp
CImgList (const CImgList< t > & list )
```

Construct list copy.

Parameters

- `list` | Input list to copy.

Note

The shared state of each element of the constructed list is kept the same as in list.

8.4.3.18 `CImgList()` [17/19]

```cpp
CImgList (const CImgList< t > & list,
          const bool is_shared )
```

Construct list copy, and force the shared state of the list elements.

Parameters

- `list` | Input list to copy. |
- `is_shared` | Tells if the elements of the list are shared or non-shared copies of input images. |
8.4.3.19  CImgList() [18/19]

\begin{verbatim}
CImgList ( 
    const char *const filename ) [explicit]
\end{verbatim}

Construct list by reading the content of a file.

Parameters

\begin{verbatim}
filename  Filename, as a C-string.
\end{verbatim}

8.4.3.20  CImgList() [19/19]

\begin{verbatim}
CImgList ( 
    const CImgDisplay & disp ) [explicit]
\end{verbatim}

Construct list from the content of a display window.

Parameters

\begin{verbatim}
disp  Display window to get content from.
\end{verbatim}

Note

Constructed list contains a single image only.

8.4.4  Member Function Documentation

8.4.4.1  get_shared()

\begin{verbatim}
CImgList<T> get_shared ( )
\end{verbatim}

Return a list with elements being shared copies of images in the list instance.

Note

\begin{verbatim}
list2 = list1.get_shared() is equivalent to list2.assign(list1,true).
\end{verbatim}
8.4.4.2  assign() [1/18]

CImgList<T>& assign ( )

Destructor [in-place version].

See also
   CImgList().

8.4.4.3  clear()

CImgList<T>& clear ( )

Destructor [in-place version].

Equivalent to assign().

Note
   Only here for compatibility with STL naming conventions.

8.4.4.4  assign() [2/18]

CImgList<T>& assign ( const unsigned int n )

Construct list containing empty images [in-place version].

See also
   CImgList(unsigned int).

8.4.4.5  assign() [3/18]

CImgList<T>& assign ( const unsigned int n,
   const unsigned int width,
   const unsigned int height = 1,
   const unsigned int depth = 1,
   const unsigned int spectrum = 1 )

Construct list containing images of specified size [in-place version].

See also
   CImgList(unsigned int, unsigned int, unsigned int, unsigned int, unsigned int).
8.4.4.6 assign() \[4/18\]

\begin{verbatim}
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 \textbf{[in-place version]}.

See also
CImgList(unsigned int, unsigned int, unsigned int, unsigned int, unsigned int, const T).
\end{verbatim}

8.4.4.7 assign() \[5/18\]

\begin{verbatim}
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 \textbf{[in-place version]}.

See also
CImgList(unsigned int, unsigned int, unsigned int, unsigned int, unsigned int, const int, const int, ...).
\end{verbatim}

8.4.4.8 assign() \[6/18\]

\begin{verbatim}
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 \textbf{[in-place version]}.

See also
CImgList(unsigned int, unsigned int, unsigned int, unsigned int, unsigned int, const double, const double, ...).
\end{verbatim}
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< t1 > & img1,
    const CImg< t2 > & img2,
    const CImg< t3 > & img3,
    const bool is_shared = false
}
```

Construct list from three images [in-place version].

See also

`CImgList(const CImg< t1 >&, const CImg< t2 >&, const CImg< t3 >&, bool)`.

8.4.4.13 `assign()` [11/18]

```cpp
cImgList<T>& assign {
    const CImg< t1 > & img1,
    const CImg< t2 > & img2,
    const CImg< t3 > & img3,
    const CImg< t4 > & img4,
    const bool is_shared = false
}
```

Construct list from four images [in-place version].

See also

`CImgList(const CImg< t1 >&, const CImg< t2 >&, const CImg< t3 >&, const CImg< t4 >&, bool)`.

8.4.4.14 `assign()` [12/18]

```cpp
cImgList<T>& assign {
    const CImg< t1 > & img1,
    const CImg< t2 > & img2,
    const CImg< t3 > & img3,
    const CImg< t4 > & img4,
    const CImg< t5 > & img5,
    const bool is_shared = false
}
```

Construct list from five images [in-place version].

See also

`CImgList(const CImg< t1 >&, const CImg< t2 >&, const CImg< t3 >&, const CImg< t4 >&, const CImg< t5 >& , bool)`.
8.4.4.15 assign() [13/18]

```
CImgList<T>& assign {
    const CImg< t1    > & img1,
    const CImg< t2    > & img2,
    const CImg< t3    > & img3,
    const CImg< t4    > & img4,
    const CImg< t5    > & img5,
    const CImg< t6    > & img6,
    const bool is_shared = false
}
```

Construct list from six images [in-place version].

See also

```
CImgList(const CImg< t1    >&, const CImg< t2    >&, const CImg< t3    >&,
         const CImg< t4    >&, const CImg< t5    >&,
         const CImg< t6    >&, bool).
```

8.4.4.16 assign() [14/18]

```
CImgList<T>& assign {
    const CImg< t1    > & img1,
    const CImg< t2    > & img2,
    const CImg< t3    > & img3,
    const CImg< t4    > & img4,
    const CImg< t5    > & img5,
    const CImg< t6    > & img6,
    const CImg< t7    > & img7,
    const bool is_shared = false
}
```

Construct list from seven images [in-place version].

See also

```
CImgList(const CImg< t1    >&, const CImg< t2    >&,
         const CImg< t3    >&, const CImg< t4    >&,
         const CImg< t5    >&, const CImg< t6    >&,
         const CImg< t7    >&, const CImg< t8    >&,
         const CImg< t9    >&, bool).
```

8.4.4.17 assign() [15/18]

```
CImgList<T>& assign {
    const CImg< t1    > & img1,
    const CImg< t2    > & img2,
    const CImg< t3    > & img3,
    const CImg< t4    > & img4,
    const CImg< t5    > & img5,
    const CImg< t6    > & img6,
    const CImg< t7    > & img7,
    const CImg< t8    > & img8,
    const bool is_shared = false
}
```

Construct list from eight images [in-place version].

See also

```
CImgList(const CImg< t1    >&, const CImg< t2    >&,
         const CImg< t3    >&, const CImg< t4    >&,
         const CImg< t5    >&, const CImg< t6    >&,
         const CImg< t7    >&, const CImg< t8    >&,
         const CImg< t9    >&, const CImg< t10   >&, bool).
```
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()

`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

\[
list(n, x, y, z, c) \text{ is equivalent to } list[n](x, y, z, c).
\]

8.4.4.28 \hspace{1mm} \textbf{operator CImg} \hspace{1mm} < \hspace{1mm} T \hspace{1mm} > \hspace{1mm} \ast ()

\textbf{operator CImg} \hspace{1mm} < \hspace{1mm} T \hspace{1mm} > \hspace{1mm} \ast ()

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 \hspace{1mm} \textbf{operator=} \hspace{1mm} [1/4]

\textbf{CImgList} < \hspace{1mm} T \hspace{1mm} > & \hspace{1mm} \textbf{operator=} \hspace{1mm} ( \hspace{1mm} const \hspace{1mm} \textbf{CImg} < \hspace{1mm} T \hspace{1mm} > & \hspace{1mm} \text{img} )

Construct list from one image \textbf{[in-place version]}.

Parameters

\begin{itemize}
  \item \textbf{img} \hspace{1mm} \text{Input image to copy in the constructed list.}
\end{itemize}

Note

\[
list = img; \text{ is equivalent to } list.\text{assign(img)};
\]

8.4.4.30 \hspace{1mm} \textbf{operator=} \hspace{1mm} [2/4]

\textbf{CImgList} < \hspace{1mm} T \hspace{1mm} > & \hspace{1mm} \textbf{operator=} \hspace{1mm} ( \hspace{1mm} const \hspace{1mm} \textbf{CImgList} < \hspace{1mm} T \hspace{1mm} > & \hspace{1mm} \text{list} )

Construct list from another list.
Parameters

| list | Input list to copy |

Note

\[
\text{list}_1 = \text{list}_2 \text{ is equivalent to } \text{list}_1.\text{assign} (\text{list}_2);. \]

8.4.4.31 operator=() [3/4]

```
CImgList\langle T \rangle & \text{operator= (}
    \text{const char } * \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]

```
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+()

```
CImgList\langle T \rangle \text{operator+ ( ) const}
```

Return a non-shared copy of a list.

Note

\[
+\text{list} \text{ is equivalent to } \text{CImgList}\langle T \rangle (\text{list}, \text{false}). \text{It forces the copy to have non-shared elements.} \]

8.4.4.34 operator,() [1/2]

```
CImgList\langle T \rangle & \text{operator, (}
    \text{const CImg}\langle t > & \text{img }
)
```

Return a copy of the list instance, where image img has been inserted at the end.
Parameters

| img | Image inserted at the end of the instance copy. |

Note

Define a convenient way to create temporary lists of images, as in the following code:

```
(img1, img2, img3, img4).display("My four images");
```

8.4.4.35 **operator,()** [2/2]

```
CImgList<T>& operator, (const CImgList<T>& list)
```

Return a copy of the list instance, where all elements of input list `list` have been inserted at the end.

Parameters

| list | List inserted at the end of the instance copy. |

8.4.4.36 **operator>()**

```
CImg<T> operator> (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

`list>'x'` is equivalent to `list.get_append('x')`.

8.4.4.37 **operator<()**

```
CImgList<T> operator< (const char axis) const
```

Return list corresponding to the splitting of all images of the instance list along specified axis.
Parameters

| axis       | Axis used for image splitting. |

Note

list<’x’> is equivalent to list.get_split(’x’).

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 data()[1/2]

CImg<T>* data()

Return pointer to the first image of the list.

Note

Images in a list are stored as a buffer of CImg<T>.

8.4.4.42 data()[2/2]

CImg<T>* data(const unsigned int pos)

Return pointer to the pos-th image of the list.

Parameters

| pos | Index of the image element to access. |

Note

list.data(n); is equivalent to list.data + n;.

8.4.4.43 at()

CImg<T>& at(const int pos)

Return pos-th image of the list.

Parameters

| pos | Index of the image element to access. |

8.4.4.44 atXYZC()[1/2]

T& atXYZC(const int pos, const int x,

Generated by Doxygen
const int y,
const int z,
const int c,
const T & out_value )

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

list.atNXYZC(p,x,y,z,c); is equivalent to list[p].atXYZC(x,y,z,c);

8.4.4.45 atNXYZC() [2/2]

T& atNXYZC (  
    const int pos,  
    const int x,  
    const int y,  
    const int z,  
    const int c )

Access to pixel value with Neumann boundary conditions.

Parameters

<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.atNXYZC(p,x,y,z,c); is equivalent to list[p].atXYZC(x,y,z,c);

8.4.4.46 atXYZ() [1/2]

T& atXYZ (  
    const int pos,  

Generated by Doxygen
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);.
### 8.4.4.48  \texttt{atNXY()} [1/2]

**T& \texttt{atNXY} (**
\begin{verbatim}
    const int pos,
    const int x,
    const int y,
    const int z,
    const int c,
    const T & out_value
\end{verbatim}
**)

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**

\texttt{list.atNXYZ(p,x,y,z,c); is equivalent to list[p].atXYZ(x,y,z,c);}.

### 8.4.4.49  \texttt{atNXY()} [2/2]

**T& \texttt{atNXY} (**
\begin{verbatim}
    const int pos,
    const int x,
    const int y,
    const int z = 0,
    const int c = 0
\end{verbatim}
**)

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**

\texttt{list.atNXYZ(p,x,y,z,c); 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**

list.atNXYZ(p,x,y,z,c); 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**

list.atNXYZ(p,x,y,z,c); is equivalent to list[p].atXYZ(x,y,z,c);.
8.4.4.52  atN() [1/2]

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

\[
\text{list.atNXYZ(p,x,y,z,c); is equivalent to list[p].atXYZ(x,y,z,c);}.
\]

8.4.4.53  atN() [2/2]

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

\[
\text{list.atNXYZ(p,x,y,z,c); is equivalent to list[p].atXYZ(x,y,z,c);}.
\]
8.4.4.54  is_sameN()  [1/2]

```cpp
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]

```cpp
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()  

```cpp
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. |

8.4.4.57  is_sameNXYZC()  

```cpp
bool is_sameNXYZC (  
```

Generated by Doxygen
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

| n   | Number of images in the list. |
| dx  | Checked image width.         |
| dy  | Checked image height.        |
| dz  | Checked image depth.         |
| dc  | Checked image spectrum.      |

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

| n   | Index of the image whom checked pixel value belong to. |
| x   | X-coordinate of the checked pixel value.             |
| y   | Y-coordinate of the checked pixel value.             |
| z   | Z-coordinate of the checked pixel value.             |
| c   | C-coordinate of the checked pixel value.             |

8.4.4.59 containsN()

bool containsN (
    const int n ) const

Test if list contains image with specified index.

Parameters

| n   | Index of the checked image. |

Generated by Doxygen
8.4.4.60  contains() [1/8]

```
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

<table>
<thead>
<tr>
<th></th>
<th>Reference to pixel value to test.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pixel</strong></td>
<td>Reference to pixel value to test.</td>
</tr>
<tr>
<td><strong>out n</strong></td>
<td>Index of image containing the pixel value, if test succeeds.</td>
</tr>
<tr>
<td><strong>out x</strong></td>
<td>X-coordinate of the pixel value, if test succeeds.</td>
</tr>
<tr>
<td><strong>out y</strong></td>
<td>Y-coordinate of the pixel value, if test succeeds.</td>
</tr>
<tr>
<td><strong>out z</strong></td>
<td>Z-coordinate of the pixel value, if test succeeds.</td>
</tr>
<tr>
<td><strong>out c</strong></td>
<td>C-coordinate of the pixel value, if test succeeds.</td>
</tr>
</tbody>
</table>

Note

If true, set coordinates (n,x,y,z,c).

8.4.4.61  contains() [2/8]

```
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

<table>
<thead>
<tr>
<th></th>
<th>Reference to pixel value to test.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pixel</strong></td>
<td>Reference to pixel value to test.</td>
</tr>
<tr>
<td><strong>out n</strong></td>
<td>Index of image containing the pixel value, if test succeeds.</td>
</tr>
<tr>
<td><strong>out x</strong></td>
<td>X-coordinate of the pixel value, if test succeeds.</td>
</tr>
<tr>
<td><strong>out y</strong></td>
<td>Y-coordinate of the pixel value, if test succeeds.</td>
</tr>
<tr>
<td><strong>out z</strong></td>
<td>Z-coordinate of the pixel value, if test succeeds.</td>
</tr>
</tbody>
</table>
Note
If true, set coordinates (n,x,y,z).

8.4.4.62 contains() [3/8]

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>pixel</th>
<th>Reference to pixel value to test.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out n</td>
<td>Index of image containing the pixel value, if test succeeds.</td>
</tr>
<tr>
<td>out x</td>
<td>X-coordinate of the pixel value, if test succeeds.</td>
</tr>
<tr>
<td>out 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]

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>pixel</th>
<th>Reference to pixel value to test.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out n</td>
<td>Index of image containing the pixel value, if test succeeds.</td>
</tr>
<tr>
<td>out 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></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>
</tbody>
</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**

| pixel | Reference to pixel value to test. |

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></th>
<th>img</th>
<th>Reference to image to test.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>n</td>
<td>Index of image in the list, if test succeeds.</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**

- `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**

- `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 insert() [1/5]

```cpp
CImgList<T>& insert {
    const CImg< t > & img,
    const unsigned int pos = ~0U,
    const bool is_shared = false }
```

Insert a copy of the image `img` into the current image list, at position `pos`.

Parameters

| img  | Image to insert a copy to the list. |
| pos  | Index of the insertion. |
| is_shared | Tells if the inserted image is a shared copy of `img` or not. |

8.4.4.72 insert() [2/5]

```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 insert() [3/5]

```cpp
CImgList<T>& insert {
    const unsigned int n,
    const CImg< t > & img,
    const unsigned int pos = ~0U,
    const bool is_shared = false }
```

Insert `n` copies of the image `img` into the current image list, at position `pos`. 
Parameters

<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()
```
Remove all images between from indexes.

Parameters

| pos1 | Starting index of the removal. |
| pos2 | Ending index of the removal. |

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

| pos0 | Starting index of the sublist. |
| pos1 | Ending index of the sublist. |

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()**

```cpp
CImg<T> get_append (  
    const char axis,  
    const float align = 0 ) const
```

Return a single image which is the appending of all images of the current `CImgList` instance.

Parameters

| axis | Appending axis. Can be { 'x' | 'y' | 'z' | 'c' }. |
|------|--------------------------------------------------|
| align | Appending alignment. |

8.4.4.81  **split()**

```cpp
CImgList<T>& split (  
    const char axis,  
    const int nb = -1 )
```

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]

```cpp
CImgList<T>& push_back (  
    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.
8.4 CImgList< T > Struct Template Reference

Parameters

| iter | Iterator pointing to the image to remove. |

8.4.4.87 get_select() [1/2]

CImg< intT > get_select (  
    CImgDisplay & disp,  
    const bool feature_type = true,  
    const char axis = 'x',  
    const float align = 0,  
    const bool exit_on_anykey = false ) const

Display a simple interactive interface to select images or sublists.

Parameters

| 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]

CImg< intT > get_select (  
    const char *const title,  
    const bool feature_type = true,  
    const char axis = 'x',  
    const float align = 0,  
    const bool exit_on_anykey = false ) const

Display a simple interactive interface to select images or sublists.

Parameters

| 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. |

Generated by Doxygen
Returns

A one-column vector containing the selected image indexes.

8.4.4.89  load()

```cpp
CImgList<T>& load ( const char *const filename )
```

Load a list from a file.

**Parameters**

| filename | Filename to read data from. |

8.4.4.90  load_cimg() [1/3]

```cpp
CImgList<T>& load_cimg ( const char *const filename )
```

Load a list from a .cimg file.

**Parameters**

| filename | Filename to read data from. |

8.4.4.91  load_cimg() [2/3]

```cpp
CImgList<T>& load_cimg ( std::FILE *const file )
```

Load a list from a .cimg file.

**Parameters**

| file     | File to read data from. |

8.4.4.92  load_cimg() [3/3]

```cpp
CImgList<T>& load_cimg ( 
```
8.4 CImgList\lt T \gt  Struct Template Reference

const char *const filename,
const unsigned int n0,
const unsigned int n1,
const unsigned int x0,
const unsigned int y0,
const unsigned int z0,
const unsigned int c0,
const unsigned int x1,
const unsigned int y1,
const unsigned int z1,
const unsigned int c1

Load a sublist list from a (non compressed) .cimg file.

Parameters

<table>
<thead>
<tr>
<th>filename</th>
<th>Filename to read data from.</th>
</tr>
</thead>
<tbody>
<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\lt T\gt\& load_parrec ( 
    const char *const filename )
```

Load a list from a PAR/REC (Philips) file.

Parameters

<table>
<thead>
<tr>
<th>filename</th>
<th>Filename to read data from.</th>
</tr>
</thead>
</table>

8.4.4.94 load_yuv()

```
CImgList\lt T\gt\& load_yuv ( 
    const char *const filename, 
    const unsigned int size_x, 
    const unsigned int size_y, 
    const unsigned int chroma_subsampling = 444,
```

Generated by Doxygen
const unsigned int first_frame = 0,
const unsigned int last_frame = ~0U,
const unsigned int step_frame = 1,
const bool yuv2rgb = true )

Load a list from a YUV image sequence file.

Parameters

<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>size_x</td>
<td>Width of the images.</td>
</tr>
<tr>
<td>size_y</td>
<td>Height of the images.</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 first image frame to read.</td>
</tr>
<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>yuv2rgb</td>
<td>Apply YUV to RGB transformation during reading.</td>
</tr>
</tbody>
</table>

8.4.4.95 load_video()

CImgList&lt;T&gt;& 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>filename</td>
<td>Filename, as a C-string.</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 (can be higher than the actual number of frames, e.g. '~0U').</td>
</tr>
<tr>
<td>step_frame</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&lt;T&gt;& load_ffmpeg_external (    const char *const filename )

Load an image from a video file using the external tool 'ffmpeg'.
Parameters

| filename | Filename to read data from. |

8.4.4.97  **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. |

Generated by Doxygen
Parameters

<table>
<thead>
<tr>
<th>last_frame</th>
<th>Index of last image frame to read.</th>
</tr>
</thead>
<tbody>
<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

| title | Label set to the information displayed. |
| display_stats | Tells if image statistics must be computed and displayed. |

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

| disp | Reference to an existing CImgDisplay instance, where the current image list will be displayed. |
| axis | Appending axis. Can be { 'x', 'y', 'z', 'c' }. |
| align | Appending alignment. |

Note

This function displays the list images of the current CImgList instance into an existing CImgDisplay window. Images of the list are appended in a single temporary image for visualization purposes. The function returns immediately.
8.4.4.102  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()

```cpp
const CImgList<T> & save (  
    const char *const filename,  
    const int number = -1,  
    const unsigned int digits = 6 ) const
```

Save list into a 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>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()

```cpp
static bool is_saveable (  
    const char *const filename ) [static]
```

Tell if an image list can be saved as one single 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>

**Returns**

true if the file format supports multiple images, false otherwise.

8.4.4.106 save_gif_external()

```cpp
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.

**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 desired frames per second.</td>
</tr>
<tr>
<td>nb_loops</td>
<td>Number of loops (0 for infinite looping).</td>
</tr>
</tbody>
</table>
8.4.4.107 save_yuv() [1/2]

const CImgList&lt;T&gt;& 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&lt;T&gt;& 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&lt;T&gt;& 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 (  
    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 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 (  
    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.
8.4.4.113 save_empty_cimg() [1/2]

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>

8.4.4.114 save_empty_cimg() [2/2]

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>
### 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 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,
```

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>
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<unsigned char> 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

| font_height | Height of the desired font (exact match for 13,23,53,103). |
| is_variable_width | Decide if the font has a variable (true) or fixed (false) width. |

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

| axis | Axis along which the Fourier transform is computed. |
| invert | Tells if the direct (false) or inverse transform (true) is computed. |

8.4.4.122  FFT(2/2)

CImgList<T>& FFT (const bool invert = false)

Compute n-D Fast Fourier Transform.

Parameters

| invert | Tells if the direct (false) or inverse transform (true) is computed. |
Index

~CImg
  cimg_library::CImg, 126
~CImgDisplay
  cimg_library::CImgDisplay, 352
~CImgList
  cimg_library::CImgList, 391

abs
  cimg_library::CImg, 194
acos
  cimg_library::CImg, 197
acosh
  cimg_library::CImg, 199
append
  cimg_library::CImg, 255
append_object3d
  cimg_library::CImg, 276
asin
  cimg_library::CImg, 197
asinh
  cimg_library::CImg, 199
assign
  cimg_library::CImg, 136–139
  cimg_library::CImgDisplay, 355
  cimg_library::CImgList, 399–405
at
  cimg_library::CImg, 169
  cimg_library::CImgList, 412
atNXYZC
  cimg_library::CImgList, 412, 413
atNXYZ
  cimg_library::CImgList, 413, 414
atNXY
  cimg_library::CImgList, 414, 415
atNX
  cimg_library::CImgList, 414, 415
atXYZC
  cimg_library::CImg, 172, 173
atXYZ
  cimg_library::CImg, 172
atXY
  cimg_library::CImg, 171
atan
  cimg_library::CImg, 198
atan2
  cimg_library::CImg, 198
atanh
  cimg_library::CImg, 199
atN
  cimg_library::CImgList, 416, 417
atof
  cimg_library::CImg, 50
atX
  cimg_library::CImg, 170
autocrop
  cimg_library::CImg, 248
back
  cimg_library::CImg, 168
begin
  cimg_library::CImg, 167
blur
  cimg_library::CImg, 262, 263
blur_anisotropic
  cimg_library::CImg, 263, 264
blur_bilateral
  cimg_library::CImg, 264, 265
blur_box
  cimg_library::CImg, 266
blur_guided
  cimg_library::CImg, 267
blur_median
  cimg_library::CImg, 268
blur_patch
  cimg_library::CImg, 267
box3d
  cimg_library::CImg, 281
boxfilter
  cimg_library::CImg, 266
button
  cimg_library::CImgDisplay, 362

Clmg
  cimg_library::Clmg, 126–129, 131–136
Clmg Library Overview, 11
Clmg< T >, 67
Clmg3dtoobject3d
  cimg_library::Clmg, 285
ClmgDisplay, 346
  cimg_library::ClmgDisplay, 353–355
ClmgException, 375
ClmgList
  cimg_library::ClmgList, 391–399
ClmgList< T >, 376
  cimg_library, 33
cimg_library::Clmg
  ~Clmg, 126
  abs, 194
  acos, 197
  acosh, 199
append, 255
append_object3d, 276
asin, 197
asinh, 199
assign, 136–139
at, 169
atXYZC, 172, 173
atXYZ, 172
atXY, 171
atan, 198
atan2, 198
atanh, 199
atX, 170
autocrop, 248
blur, 262, 263
blur_anisotropic, 263, 264
blur_bilateral, 264, 265
blur_box, 266
blur_guided, 267
blur_median, 268
blur_patch, 267
box3d, 281
boxfilter, 266
Clmg, 126–129, 131–136
Clmg3dtoobject3d, 285
clear, 139
columns, 249
cone3d, 281
const_iterator, 125
contains, 189, 190
containsXYZC, 188
convolve, 256
cool_LUT256, 239
correlate, 255
cos, 195
cosh, 196
crop, 248
cross, 218
cube_LUT256, 240
cubic_atX_c, 177, 178
cubic_atXY_c, 179
cubic_atXYZ_c, 180
cubic_atXYZ_p, 181
cubic_atXYZ, 180
cubic_atXY, 178, 179
cubic_atX, 176, 177
cumulate, 257
cut, 234
cylinder3d, 282
data, 166
default_LUT256, 238
depth, 165
deriche, 261
diagonal, 217
diffusion_tensors, 270
dijkstra, 223
dilate, 260
discard, 231
displacement, 270
display, 330, 331
display_graph, 332
display_object3d, 331
distance, 271, 272
distance_dijkstra, 272
distance_eikonal, 272, 273
div, 200
dot, 214
draw_arrow, 291
draw_axes, 311
draw_axis, 310, 311
draw_circle, 305, 306
draw_ellipse, 303–305
draw_fill, 313
draw_gaussian, 315, 316
draw_graph, 312
draw_grid, 312
draw_image, 306, 307
draw_line, 286–289, 291
draw_mandelbrot, 314
draw_object3d, 316
draw_plasma, 314
draw_point, 285, 286
draw_polygon, 303
draw_quiver, 309, 310
draw_rectangle, 301, 302
draw_spline, 292–294
draw_text, 307, 308
draw_triangle, 295–300
eigen, 220
elevation3d, 279
ellipsoid3d, 284
empty, 141
end, 168
equalize, 236
erode, 259
eval, 212, 213
exp, 193
FFT, 274
fill, 228, 229
fillC, 231
fillX, 229
fillY, 230
fillZ, 230
flag_LUT256, 239
front, 168
get_FFT, 274
get_SVD, 222
get_channel, 250
get_channels, 250
get_column, 248
get_eigen, 220
get_elevation3d, 277
get_gradient, 269
get_hessian, 269
get_isoline3d, 278
get_isosurface3d, 278
get_matrix_at, 214
get_projections2d, 247
get_projections3d, 277
get_rows, 249
get_serialize, 346
get_shared_channel, 254
get_shared_channels, 253
get_shared_row, 252
get_shared_rows, 252
get_shared_slice, 253
get_shared_slices, 253
get_slice, 250
get_slices, 250
get_split, 254
get_symmetric_eigen, 221
get_tensor_at, 215
get_vector_at, 214
HSV_LUT256, 238
haar, 273
height, 164
histogram, 235
hot_LUT256, 239
identity_matrix, 217, 227
index, 236
invert, 219
is_CImg3d, 191
is_empty, 183
is_inf, 183
is_nan, 183
is_object3d, 191
is_overlapped, 190
is_sameXYZC, 188
is_sameXYC, 187
is_sameXYZ, 186
is_sameXZC, 187
is_sameXC, 185
is_sameXY, 184
is_sameXZ, 184
is_sameYZC, 187, 188
is_sameYC, 185, 186
is_sameZC, 185
is_sameZC, 186
is_shared, 183
isoline3d, 279
isosurface3d, 280
iterator, 125
jet_LUT256, 239
kth_smallest, 210
label, 237, 238
linear_atXYZC, 176
linear_atXYZ, 175
linear_atXY, 174, 175
linear_atX, 173, 174
lines_LUT256, 238
load, 317
load_analyze, 322
load_ascii, 318
load_bmp, 318
load_camera, 329
load_cimg, 323
load_dcmread_external, 329
load_dim, 318
load_ext, 324
load_ffmpeg_external, 327
load_gif_external, 327
load_graphicmagick_external, 328
load_gzip_external, 328
load_imagemagic_external, 328
load_inr, 324
load_jpeg, 319
load_magick, 319
load_medcon_external, 329
load_minc2, 322
load_off, 326
load_other, 330
load_pandore, 324
load_parrec, 325
load_pfm, 321
load_png, 319
load_pnm, 319
load_raw, 325
load_rgb, 321
load_rgbia, 321
load_tiff, 322
load_video, 327
load_yuv, 326
log, 193
log10, 194
log2, 193
MSE, 211
magnitude, 213
map, 237
matchpatch, 271
matrix, 225, 226
max, 203, 205
max abs, 209
max abs, 207, 209
min, 202, 203
min max, 209
min abs, 205, 207
mirror, 243
move_to, 139, 140
mul, 199
noise, 232
norm, 233
normalize, 233
object3dtoCImg3d, 285
offset, 167
operator & , 154
operator &==, 153, 154
operator T*, 142
operator ! = , 161, 162
operator < , 163

Generated by Doxygen
operator<<, 158
operator<<=, 157, 158
operator>>, 159
operator>>=, 158, 159
operator*, 150, 151
operator*=, 149, 150
operator~, 160
operator\, 157
operator+=, 156
operator(), 141, 142
operator+, 146, 147
operator++, 146
operator+=, 144, 145
operator,, 162, 163
operator-, 148, 149
operator=-, 147, 148
operator/, 152
operator/=, 151
operator%=, 143, 144
operator*, 160
operator%, 153
operator%=(152, 153
operator|, 155, 156
operator|=, 155
PSNR, 211
permute_axes, 244
pixel_type, 164
plane3d, 283
pow, 200, 201
print, 330
quantize, 234
RGBtoXYZ, 240
rand, 231
resize, 241, 242
resize_doubleXY, 242
resize_object3d, 275
resize_tripleXY, 243
rotate, 201
ror, 201, 202
rotation_matrix, 228
round, 232
row, 249
SVD, 222
save, 333
save_analyze, 338
save_ascii, 333
save_bmp, 334
save_cimg, 338
save_cpp, 333
save_dim, 334
save_empty_cimg, 339
save_exr, 340
save_ffmpeg_external, 343
save_graphicsmagick_external, 344
save_gzip_external, 344
save_imagemagick_external, 344
save_inr, 339
save_jpeg, 334
save_magick, 335
save_medcon_external, 345
save_minc2, 337
save_off, 342
save_other, 345
save_pandore, 340
save_pfm, 336
save_png, 335
save_pmk, 336
save_pnm, 335
save_raw, 341
save_rgb, 336
save_rgba, 336
save_tiff, 337
save_video, 343
save_yuv, 341, 342
select, 317
sequence, 218, 227
set_linear_atXYZ, 182
set_linear_atXY, 182
set_linear_atX, 181
set_matrix_at, 215
set_tensor_at, 217
set_vector_at, 215
sharpen, 268
shift, 244
shift_object3d, 275
sign, 194
sin, 195
sinc, 195
sinh, 196
size, 165
sqr, 192
solve, 219
solve_tridiagonal, 219
sort, 221
tan, 196
spectrum, 165
torus3d, 282	sqr, 192
tanh, 197
tensorize, 269
tensorize_object3d, 276
threshold, 235
torus3d, 282
transpose, 218
unroll, 244
value_string, 182
value_type, 126
vanvliet, 261
variance, 210
INDEX 449

variance_mean, 210
variance_noise, 211
vector, 217, 224, 225
warp, 247
watershed, 261
width, 164
XYZtoRGB, 240
cimg_library::CImgDisplay
∼CImgDisplay, 352
assign, 355
button, 362
CImgDisplay, 353–355
close, 366
display, 365
empty, 355
flush, 373
frames_per_second, 364
height, 359
hide_mouse, 370
is_closed, 357
is_key, 357, 358
is_key_sequence, 358
is_keyESC, 359
key, 363
keycode, 364
mouse_x, 361
mouse_y, 361
move, 366
normalization, 359
operator bool, 356
operator=, 356
paint, 374
released_key, 364
render, 373
resize, 367, 368
snapshot, 374
show, 366
show_mouse, 370
set_button, 371
set_fullscreen, 369
set_key, 372
set_mouse, 370
set_normalization, 368
set_title, 369
set_wheel, 371, 372
show, 366
show_mouse, 370
snapshot, 374
title, 360
toggle_fullscreen, 370
wait, 373
wheel, 362
width, 359
window_height, 360
window_width, 360
window_x, 361
window_y, 361
cimg_library::CImgList
∼CImgList, 391
assign, 399–405

at, 412
atNXYZC, 412, 413
atNXYZ, 413, 414
atNXY, 414, 415
atNX, 415, 416
atN, 416, 417
CImgList, 391–399
clear, 400
const_iterator, 390
contains, 420–423
containsNXYZC, 419
containsN, 419
data, 411, 412
display, 434, 435
empty, 406
erase, 428
FFT, 443
font, 442
get_append, 427
gtNXYZC, 412, 413
gtNXYZ, 413, 414
gtNXY, 414, 415
gtNX, 415, 416
gtN, 416, 417
CImgList, 391–399
clear, 400
const_iterator, 390
contains, 420–423
containsNXYZC, 419
containsN, 419
data, 411, 412
display, 434, 435
empty, 406
erase, 428
FFT, 443
font, 442
get_append, 427
get_select, 429
get_serialize, 442
get_shared, 399
get_shared_images, 426
images, 426
insert, 424, 425
is_sameNXYZC, 418
is_sameXYZC, 418
is_sameN, 417, 418
is_saveable, 436
iterator, 390
load, 430
load_cimg, 430
load_ffmpeg_external, 432
load_gif_external, 433
load_gzip_external, 433
load_parrec, 431
load_tiff, 433
load_video, 432
load_yuv, 431
max_min, 423
min_max, 423
move_to, 405, 406
operator Clmg< T > *, 408
operator<, 410
operator>, 410
operator(), 407
operator+, 409
operator-, 409, 410
operator-, 408, 409
pixel_type, 411
print, 434
push_back, 427, 428
push_front, 428
remove, 425, 426
save, 435
save_cimg, 437, 438
save_empty_cimg, 440
save_ffmpeg_external, 442

Generated by Doxygen
cimg_library::Clmg, 193

FAQ : Frequently Asked Questions., 14
FFT
  cimg_library::Clmg, 274
  cimg_library::ClmgList, 443
fclose
  cimg_library::cimg, 54
fdate
  cimg_library::cimg, 55, 57
fempty
  cimg_library::cimg, 63
ffmpeg_path
  cimg_library::cimg, 60
files
  cimg_library::cimg, 64
Files IO in Clmg., 30
fill
  cimg_library::Clmg, 228, 229
fillC
  cimg_library::Clmg, 231
fillX
  cimg_library::Clmg, 229
fillY
  cimg_library::Clmg, 230
fillZ
  cimg_library::Clmg, 230
flag_LUT256
  cimg_library::Clmg, 239
flush
  cimg_library::ClmgDisplay, 373
font
  cimg_library::ClmgList, 442
fopen
  cimg_library::cimg, 54
frames_per_second
  cimg_library::ClmgDisplay, 364
fread
  cimg_library::cimg, 62
front
  cimg_library::Clmg, 168
fsize
  cimg_library::cimg, 55
ftype
  cimg_library::cimg, 63
fwrite
  cimg_library::cimg, 63
get_FFT
  cimg_library::Clmg, 274
get_SVD
  cimg_library::Clmg, 222
get_append
  cimg_library::ClmgList, 427
get_channel
  cimg_library::Clmg, 250
get_channels
  cimg_library::Clmg, 250
get_column
  cimg_library::Clmg, 248
generate
  cimg_library::Clmg, 220
generate_elevation3d
  cimg_library::Clmg, 277
generate_gradient
  cimg_library::Clmg, 269
generate_hessian
  cimg_library::Clmg, 269
generate_isoline3d
  cimg_library::Clmg, 278
generate_isosurface3d
  cimg_library::Clmg, 278
generate_matrix_at
  cimg_library::Clmg, 214
generate_projections2d
  cimg_library::Clmg, 247
generate_projections3d
  cimg_library::Clmg, 277
generate_rows
  cimg_library::Clmg, 249
generate_select
  cimg_library::ClmgList, 429
generate_serialize
  cimg_library::Clmg, 346
  cimg_library::ClmgList, 442
generate_shared
  cimg_library::ClmgList, 399
generate_shared_channel
  cimg_library::Clmg, 254
generate_shared_channels
  cimg_library::Clmg, 253
generate_shared_images
  cimg_library::ClmgList, 426
generate_shared_points
  cimg_library::Clmg, 251
generate_shared_row
  cimg_library::Clmg, 252
generate_shared_rows
  cimg_library::Clmg, 252
generate_shared_slice
  cimg_library::Clmg, 253
generate_shared_slices
  cimg_library::Clmg, 253
generate_slice
  cimg_library::Clmg, 250
generate_slices
  cimg_library::Clmg, 250
generate_split
  cimg_library::Clmg, 250
generate_symmetric_eigen
  cimg_library::Clmg, 221
generate_tenors_at
  cimg_library::Clmg, 215
generate_vector_at
  cimg_library::Clmg, 214
graphicsmagick_path
  cimg_library::cimg, 59
gunzip_path
  cimg_library::cimg, 60
gzip_path
  cimg_library::cimg, 60

HSV_LUT256
  cimg_library::CImg, 238
haar
  cimg_library::CImg, 273
height
  cimg_library::CImg, 164
  cimg_library::CImgDisplay, 359
hide_mouse
  cimg_library::CImgDisplay, 370
histogram
  cimg_library::CImg, 235
hot_LUT256
  cimg_library::CImg, 239

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, 217, 227
imagemagick_path
  cimg_library::cimg, 58
images
  cimg_library::CImgList, 426
index
  cimg_library::CImg, 236
info
  cimg_library::cimg, 43
insert
  cimg_library::CImgList, 424, 425
invert
  cimg_library::CImg, 219
invert_endianness
  cimg_library::cimg, 46
is_CImg3d
  cimg_library::CImg, 191
is_closed
  cimg_library::CImgDisplay, 357
is_directory
  cimg_library::cimg, 54
is_empty
  cimg_library::CImg, 183
is_file
  cimg_library::cimg, 55
is_inf
  cimg_library::CImg, 183
is_key
  cimg_library::CImgDisplay, 357, 358
is_key_sequence
  cimg_library::CImgDisplay, 358
is_keyESC
  cimg_library::CImgDisplay, 359
is_nan
  cimg_library::CImg, 183
is_object3d
  cimg_library::CImg, 191
is_overlapped
  cimg_library::CImg, 190
is_sameNXYZC
  cimg_library::CImgList, 418
is_sameXYZC
  cimg_library::CImg, 188
  cimg_library::CImgList, 418
is_sameXYC
  cimg_library::CImg, 187
is_sameXYZ
  cimg_library::CImg, 186
is_sameXZC
  cimg_library::CImg, 187
is_sameXC
  cimg_library::CImg, 185
is_sameXY
  cimg_library::CImg, 184
is_sameYXC
  cimg_library::CImg, 184
is_sameYZC
  cimg_library::CImg, 187, 188
is_sameY
  cimg_library::CImg, 185, 186
is_sameZC
  cimg_library::CImg, 186
is_sameN
  cimg_library::CImgList, 417, 418
is_saveable
  cimg_library::CImgList, 436
is_shared
  cimg_library::CImg, 183
isoline3d
  cimg_library::CImg, 279
isosurface3d
  cimg_library::CImg, 280
iterator
  cimg_library::CImg, 125
  cimg_library::CImgList, 390
jet_LUT256
  cimg_library::CImg, 239

key
  cimg_library::CImgDisplay, 363
keycode
  cimg_library::CImgDisplay, 364
kth_smallest
  cimg_library::CImg, 210
label
  cimg_library::CImg, 237, 238
linear_atXYZC
  cimg_library::CImg, 176
linear_atXYZ
  cimg_library::CImg, 175
linear_atXY
cimg_library::CImg, 174, 175
linear_atX
cimg_library::CImg, 173, 174
lines_LUT256
cimg_library::CImg, 238
load
cimg_library::CImg, 317
cimg_library::CImgList, 430
load_analyze
cimg_library::CImg, 322
load_ascii
cimg_library::CImg, 318
load_bmp
cimg_library::CImg, 318
load_camera
cimg_library::CImg, 329
load_cimg
cimg_library::CImg, 323
cimg_library::CImgList, 430
load_dcraw_external
cimg_library::CImg, 329
load_dlm
cimg_library::CImg, 318
load_exr
cimg_library::CImg, 324
load_ffmpeg_external
cimg_library::CImg, 327
cimg_library::CImgList, 432
load_gif_external
cimg_library::CImg, 327
cimg_library::CImgList, 433
load_graphicsmagick_external
cimg_library::CImg, 328
load_gzip_external
cimg_library::CImg, 328
cimg_library::CImgList, 433
load_imagemagick_external
cimg_library::CImg, 328
load_inr
cimg_library::CImg, 324
load_jpeg
cimg_library::CImg, 319
load_magick
cimg_library::CImg, 319
load_medcon_external
cimg_library::CImg, 329
load_minc2
cimg_library::CImg, 322
load_network
cimg_library::cimg, 64
load_off
cimg_library::CImg, 326
load_other
cimg_library::CImg, 330
load_pandore
cimg_library::CImg, 324
load_parrec
cimg_library::CImg, 325
cimg_library::CImgList, 431
load_pfm
cimg_library::CImg, 321
load_png
cimg_library::CImg, 319
load_pnm
cimg_library::CImg, 319
load_raw
cimg_library::CImg, 325
load_rgb
cimg_library::CImg, 321
load_rgba
cimg_library::CImg, 321
load_tiff
cimg_library::CImg, 322
cimg_library::CImgList, 433
load_video
cimg_library::CImg, 327
cimg_library::CImgList, 432
load_yuv
cimg_library::CImg, 326
cimg_library::CImgList, 431
log
cimg_library::CImg, 193
log10
cimg_library::CImg, 194
log2
cimg_library::CImg, 193
MSE
cimg_library::CImg, 211
magnitude
cimg_library::CImg, 213
map
cimg_library::CImg, 237
matchpatch
cimg_library::CImg, 271
matrix
cimg_library::CImg, 225, 226
max
cimg_library::CImg, 203, 205
max_min
 cimg_library::CImg, 209
cimg_library::CImgList, 423
maxabs
cimg_library::CImg, 205, 207
medcon_path
cimg_library::cimg, 59
min
cimg_library::CImg, 202, 203
min_max
 cimg_library::CImg, 209
cimg_library::CImgList, 423
minabs
cimg_library::CImg, 205, 207
minmod
cimg_library::CImg, 50
mirror
cimg_library::CImg, 243
mod
  cimg_library::cimg, 49
mouse_x
  cimg_library::CImgDisplay, 361
mouse_y
  cimg_library::CImgDisplay, 361
move
  cimg_library::CImgDisplay, 366
move_to
  cimg_library::CImg, 139, 140
  cimg_library::CImgList, 405, 406
mul
  cimg_library::CImg, 199
noise
  cimg_library::CImg, 232
norm
  cimg_library::CImg, 233
normalization
  cimg_library::CImgDisplay, 359
normalize
  cimg_library::CImg, 233
object3dToCImg3d
  cimg_library::CImg, 285
offset
  cimg_library::CImg, 167
openmp_mode
  cimg_library::cimg, 44
operator &
  cimg_library::CImg, 154
operator &=
  cimg_library::CImg, 153, 154
operator bool
  cimg_library::CImg, 153, 154
operator CImg
  cimg_library::CImgList, 408
operator T
  cimg_library::CImg, 146, 147
  cimg_library::CImgList, 409
operator++
  cimg_library::CImg, 146
operator+=
  cimg_library::CImg, 144, 145
operator,
  cimg_library::CImg, 162, 163
  cimg_library::CImgList, 409, 410
operator--
  cimg_library::CImg, 148, 149
operator--
  cimg_library::CImg, 148
operator=
  cimg_library::CImg, 147, 148
operator/
  cimg_library::CImg, 152
operator/=,
  cimg_library::CImg, 161, 162
operator<
  cimg_library::CImg, 163
  cimg_library::CImgList, 410
operator<<
  cimg_library::CImg, 158
operator<<=
  cimg_library::CImg, 157, 158
operator>
  cimg_library::CImg, 157, 158
operator>>
  cimg_library::CImgList, 410
operator>>=
  cimg_library::CImg, 159
operator*=
  cimg_library::CImg, 158, 159
operator*=
  cimg_library::CImg, 150, 151
operator~
  cimg_library::CImg, 160
operator^,
  cimg_library::CImg, 157
operator^=
  cimg_library::CImg, 156
operator()
  cimg_library::CImg, 141, 142
  cimg_library::CImgList, 407
operator+
  cimg_library::CImg, 146, 147
  cimg_library::CImgList, 409
operator++
  cimg_library::CImg, 146
operator+=
  cimg_library::CImg, 144, 145
operator,
  cimg_library::CImg, 162, 163
  cimg_library::CImgList, 409, 410
operator--
  cimg_library::CImg, 148, 149
operator--
  cimg_library::CImg, 148
operator=
  cimg_library::CImg, 147, 148
operator/=
  cimg_library::CImg, 151
operator=
  cimg_library::CImg, 143, 144
  cimg_library::CImgDisplay, 356
  cimg_library::CImgList, 408, 409
operator==
  cimg_library::CImg, 160
operator%
  cimg_library::CImg, 153
operator%=
  cimg_library::CImg, 152, 153
operator |
  cimg_library::CImg, 155, 156
operator |=
  cimg_library::CImg, 155
output
  cimg_library::cimg, 42
PSNR
  cimg_library::CImg, 211
paint
  cimg_library::CImgDisplay, 374
permute_axes
  cimg_library::CImg, 244
pixel_type
  cimg_library::CImg, 164
  cimg_library::CImgList, 411
plane3d
  cimg_library::CImg, 283
pow
  cimg_library::CImg, 200, 201
print

Generated by Doxygen
cimg_library::CImgList, 437
screenshot
cimg_library::CImgList, 437
select
cimg_library::CImg, 317
sequence
cimg_library::CImg, 218, 227
set_button
cimg_library::CImgDisplay, 371
setFullscreen
cimg_library::CImgDisplay, 369
set_key
cimg_library::CImgDisplay, 372
set_linear_atXYZ
cimg_library::CImg, 182
set_linear_atXY
cimg_library::CImg, 182
set_linear_atX
cimg_library::CImg, 181
set_matrix_at
cimg_library::CImg, 215
set_mouse
cimg_library::CImgDisplay, 370
set_normalization
cimg_library::CImgDisplay, 368
set_tensor_at
cimg_library::CImg, 217
set_title
cimg_library::CImgDisplay, 369
set_vector_at
cimg_library::CImg, 215
set_wheel
cimg_library::CImgDisplay, 371, 372
Setting Environment Variables, 18
sharpen
cimg_library::CImg, 268
shift
cimg_library::CImg, 244
shift_object3d
cimg_library::CImg, 275
show
cimg_library::CImgDisplay, 366
show_mouse
cimg_library::CImgDisplay, 370
sign
cimg_library::CImg, 194
sin
cimg_library::CImg, 195
sinc
cimg_library::CImg, 195
sinh
cimg_library::CImg, 196
size
cimg_library::CImg, 165
cimg_library::CImgList, 411
sleep
cimg_library::cimg, 48
snapshot
time
cimg_library::cimg, 48
title
cimg_library::CImgDisplay, 360
toc
cimg_library::cimg, 48
toggleFullscreen
cimg_library::CImgDisplay, 370
torus3d
cimg_library::CImg, 282
transpose
cimg_library::CImg, 218
Tutorial : Getting Started., 20
unroll
cimg_library::CImg, 244
Using Display Windows., 28
Using Image Loops., 22
value_string

cimg_library::CImg, 182
value_type

cimg_library::CImg, 126
cimg_library::CImgList, 390
vanvliet

cimg_library::CImg, 261
variance

cimg_library::CImg, 210
variance_mean

cimg_library::CImg, 210
variance_noise

cimg_library::CImg, 211
vector

cimg_library::CImg, 217, 224, 225
wait

cimg_library::CImgDisplay, 373
cimg_library::cimg, 49
warn

cimg_library::cimg, 45
warp

cimg_library::CImg, 247
watershed

cimg_library::CImg, 261
wgetPath

cimg_library::cimg, 61
wheel

cimg_library::CImgDisplay, 362
width

cimg_library::CImg, 164
cimg_library::CImgDisplay, 359
cimg_library::CImgList, 411
windowHeight

cimg_library::CImgDisplay, 360
windowWidth

cimg_library::CImgDisplay, 360
windowX

cimg_library::CImgDisplay, 361
windowY

Generated by Doxygen