# 3DLDF: The Program

**Version 1.1.5.1**

by Laurence D. Finston

January 2004

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copyright and License</td>
<td>1</td>
</tr>
<tr>
<td><strong>Introduction</strong> <em>(3DLDF.web)</em></td>
<td>1</td>
</tr>
<tr>
<td>Formatting commands</td>
<td>2</td>
</tr>
<tr>
<td><strong>Preprocessor variables and library files</strong> <em>(loader.web)</em></td>
<td>3</td>
</tr>
<tr>
<td>Configuration file</td>
<td>3</td>
</tr>
<tr>
<td>Library files</td>
<td>4</td>
</tr>
<tr>
<td>Putting loader together</td>
<td>5</td>
</tr>
<tr>
<td><strong>Global items</strong> <em>(psglb.web)</em></td>
<td>6</td>
</tr>
<tr>
<td>Include files</td>
<td>6</td>
</tr>
<tr>
<td>Type definitions</td>
<td>6</td>
</tr>
<tr>
<td>Utility classes</td>
<td>7</td>
</tr>
<tr>
<td>Global variables</td>
<td>7</td>
</tr>
<tr>
<td>For the header file</td>
<td>7</td>
</tr>
<tr>
<td>Global constants</td>
<td>11</td>
</tr>
<tr>
<td>For compilation</td>
<td>11</td>
</tr>
<tr>
<td>For the header file</td>
<td>11</td>
</tr>
<tr>
<td>Utility functions</td>
<td>12</td>
</tr>
<tr>
<td>Solve quadratic equation</td>
<td>12</td>
</tr>
<tr>
<td>System information</td>
<td>13</td>
</tr>
<tr>
<td><strong>Declare namespace System</strong></td>
<td>13</td>
</tr>
<tr>
<td>Endianness</td>
<td>13</td>
</tr>
<tr>
<td>Get endianness</td>
<td>14</td>
</tr>
<tr>
<td>Is big endian</td>
<td>14</td>
</tr>
<tr>
<td>Is little endian</td>
<td>15</td>
</tr>
<tr>
<td>Register width</td>
<td>15</td>
</tr>
<tr>
<td>Get register width</td>
<td>15</td>
</tr>
<tr>
<td>Is 32 bit</td>
<td>16</td>
</tr>
<tr>
<td>Is 64 bit</td>
<td>16</td>
</tr>
<tr>
<td>Forward declarations</td>
<td>16</td>
</tr>
<tr>
<td>Putting psglb together</td>
<td>16</td>
</tr>
<tr>
<td><strong>Dynamic allocation for Shapes</strong></td>
<td>16</td>
</tr>
<tr>
<td>Include files</td>
<td>17</td>
</tr>
<tr>
<td>Dynamic allocation for Shapes</td>
<td>17</td>
</tr>
<tr>
<td>Pointer argument</td>
<td>17</td>
</tr>
<tr>
<td>Reference argument</td>
<td>17</td>
</tr>
<tr>
<td>Putting creatnew together</td>
<td>18</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Get second-largest real value</td>
<td>63</td>
</tr>
<tr>
<td>Include files</td>
<td>64</td>
</tr>
<tr>
<td>Declare namespace System</td>
<td>65</td>
</tr>
<tr>
<td>Get second largest</td>
<td>66</td>
</tr>
<tr>
<td>Calculate second-largest Real</td>
<td>69</td>
</tr>
<tr>
<td>Loop for testing bits</td>
<td>70</td>
</tr>
<tr>
<td>Template function instantiations</td>
<td>71</td>
</tr>
<tr>
<td>Putting gaittempit together</td>
<td>73</td>
</tr>
<tr>
<td>I/O (io.web)</td>
<td>76</td>
</tr>
<tr>
<td>Include files</td>
<td>77</td>
</tr>
<tr>
<td>Global variables</td>
<td>78</td>
</tr>
<tr>
<td>I/O functions</td>
<td>80</td>
</tr>
<tr>
<td>Initialize I/O</td>
<td>81</td>
</tr>
<tr>
<td>Write footers</td>
<td>84</td>
</tr>
<tr>
<td>Begin figure</td>
<td>86</td>
</tr>
<tr>
<td>End figure</td>
<td>88</td>
</tr>
<tr>
<td>Putting I/O together</td>
<td>90</td>
</tr>
<tr>
<td>Color (colors.web)</td>
<td>93</td>
</tr>
<tr>
<td>Include files</td>
<td>94</td>
</tr>
<tr>
<td>Color class definition</td>
<td>95</td>
</tr>
<tr>
<td>Constructors and setting functions</td>
<td>96</td>
</tr>
<tr>
<td>Default constructor</td>
<td>97</td>
</tr>
<tr>
<td>Copy constructor</td>
<td>99</td>
</tr>
<tr>
<td>Name and unsigned short arguments</td>
<td>101</td>
</tr>
<tr>
<td>Constructor</td>
<td>102</td>
</tr>
<tr>
<td>Setting function</td>
<td>104</td>
</tr>
<tr>
<td>Three real arguments</td>
<td>106</td>
</tr>
<tr>
<td>Constructor</td>
<td>107</td>
</tr>
<tr>
<td>Setting function</td>
<td>109</td>
</tr>
<tr>
<td>Pseudo-constructor for dynamic allocation</td>
<td>111</td>
</tr>
<tr>
<td>Pointer argument</td>
<td>112</td>
</tr>
<tr>
<td>Reference argument</td>
<td>113</td>
</tr>
<tr>
<td>Assignment</td>
<td>114</td>
</tr>
<tr>
<td>Equality</td>
<td>116</td>
</tr>
<tr>
<td>Inequality</td>
<td>118</td>
</tr>
<tr>
<td>Modifying</td>
<td>120</td>
</tr>
<tr>
<td>Set on free store</td>
<td>121</td>
</tr>
<tr>
<td>Set name</td>
<td>123</td>
</tr>
<tr>
<td>Set use name</td>
<td>125</td>
</tr>
<tr>
<td>Modify</td>
<td>127</td>
</tr>
<tr>
<td>Set red part</td>
<td>129</td>
</tr>
<tr>
<td>Set green part</td>
<td>131</td>
</tr>
<tr>
<td>Set blue part</td>
<td>133</td>
</tr>
<tr>
<td>Show</td>
<td>135</td>
</tr>
<tr>
<td>Returning elements and information</td>
<td>137</td>
</tr>
<tr>
<td>Is on free store</td>
<td>138</td>
</tr>
<tr>
<td>Get Color parts</td>
<td>140</td>
</tr>
<tr>
<td>Get red part</td>
<td>141</td>
</tr>
<tr>
<td>Get green part</td>
<td>142</td>
</tr>
<tr>
<td>Get blue part</td>
<td>143</td>
</tr>
<tr>
<td>Get use name</td>
<td>144</td>
</tr>
<tr>
<td>Get name</td>
<td>146</td>
</tr>
<tr>
<td>Topic</td>
<td>Pages</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>Output operator</td>
<td>147</td>
</tr>
<tr>
<td>Define Colors in METAPOST</td>
<td>149</td>
</tr>
<tr>
<td>Initialize Colors</td>
<td>151</td>
</tr>
<tr>
<td>Namespace Colors</td>
<td>153</td>
</tr>
<tr>
<td>Major Colors</td>
<td>155</td>
</tr>
<tr>
<td>Internal (with initialization)</td>
<td>156</td>
</tr>
<tr>
<td>External</td>
<td>157</td>
</tr>
<tr>
<td>All Colors</td>
<td>158</td>
</tr>
<tr>
<td>Global constants</td>
<td>159</td>
</tr>
<tr>
<td>Putting Color together</td>
<td>161</td>
</tr>
<tr>
<td>Transformations (transform.web)</td>
<td>164</td>
</tr>
<tr>
<td>Include files</td>
<td>165</td>
</tr>
<tr>
<td>Transform class definition</td>
<td>166</td>
</tr>
<tr>
<td>Constructors</td>
<td>167</td>
</tr>
<tr>
<td>Default constructor</td>
<td>168</td>
</tr>
<tr>
<td>Constructor with one real argument</td>
<td>170</td>
</tr>
<tr>
<td>Constructor with 16 real arguments</td>
<td>172</td>
</tr>
<tr>
<td>Assignment</td>
<td>174</td>
</tr>
<tr>
<td>Reset to identity matrix</td>
<td>176</td>
</tr>
<tr>
<td>Setting values</td>
<td>178</td>
</tr>
<tr>
<td>Clean</td>
<td>180</td>
</tr>
<tr>
<td>Epsilon</td>
<td>182</td>
</tr>
<tr>
<td>Test for identity matrix</td>
<td>184</td>
</tr>
<tr>
<td>Non-const version</td>
<td>185</td>
</tr>
<tr>
<td>const version</td>
<td>187</td>
</tr>
<tr>
<td>Querying</td>
<td>189</td>
</tr>
<tr>
<td>Get element</td>
<td>190</td>
</tr>
<tr>
<td>Show</td>
<td>192</td>
</tr>
<tr>
<td>Affine transformations</td>
<td>194</td>
</tr>
<tr>
<td>Scale</td>
<td>195</td>
</tr>
<tr>
<td>Shear</td>
<td>197</td>
</tr>
<tr>
<td>Shift</td>
<td>199</td>
</tr>
<tr>
<td>real arguments</td>
<td>200</td>
</tr>
<tr>
<td>Point argument</td>
<td>202</td>
</tr>
<tr>
<td>Shift with multiplication</td>
<td>203</td>
</tr>
<tr>
<td>Rotation around the main axes</td>
<td>205</td>
</tr>
<tr>
<td>Rotation around an arbitrary axis</td>
<td>210</td>
</tr>
<tr>
<td>Point arguments</td>
<td>211</td>
</tr>
<tr>
<td>Path argument</td>
<td>212</td>
</tr>
<tr>
<td>Alignment with an axis</td>
<td>213</td>
</tr>
<tr>
<td>Matrix multiplication</td>
<td>214</td>
</tr>
<tr>
<td>With assignment</td>
<td>215</td>
</tr>
<tr>
<td>real argument</td>
<td>216</td>
</tr>
<tr>
<td>Transform argument</td>
<td>218</td>
</tr>
<tr>
<td>Plain multiplication</td>
<td>220</td>
</tr>
<tr>
<td>real argument</td>
<td>221</td>
</tr>
<tr>
<td>Transform argument</td>
<td>223</td>
</tr>
<tr>
<td>Matrix inversion</td>
<td>225</td>
</tr>
<tr>
<td>const version (no assignment)</td>
<td>226</td>
</tr>
<tr>
<td>Non-const version (with assignment)</td>
<td>232</td>
</tr>
<tr>
<td>Global variables</td>
<td>234</td>
</tr>
<tr>
<td>Global constants</td>
<td>236</td>
</tr>
</tbody>
</table>
## Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Putting <strong>Transform</strong> together</td>
<td>238</td>
</tr>
<tr>
<td><strong>Shape</strong> (shapes.web)</td>
<td>241</td>
</tr>
<tr>
<td>Include files</td>
<td>242</td>
</tr>
<tr>
<td><strong>Shape</strong> class definition</td>
<td>243</td>
</tr>
<tr>
<td>Static data members</td>
<td>246</td>
</tr>
<tr>
<td>Putting <strong>Shape</strong> together</td>
<td>247</td>
</tr>
<tr>
<td><strong>Picture and Label</strong> (pictures.web)</td>
<td>250</td>
</tr>
<tr>
<td>Include files</td>
<td>251</td>
</tr>
<tr>
<td><strong>Label</strong></td>
<td>252</td>
</tr>
<tr>
<td><strong>Label</strong> class definition</td>
<td>253</td>
</tr>
<tr>
<td>Static data members</td>
<td>254</td>
</tr>
<tr>
<td>Declarations for <strong>Label</strong> functions</td>
<td>255</td>
</tr>
<tr>
<td><strong>namespace</strong> Projections</td>
<td>256</td>
</tr>
<tr>
<td><strong>namespace</strong> Sorting</td>
<td>258</td>
</tr>
<tr>
<td><strong>Picture</strong></td>
<td>260</td>
</tr>
<tr>
<td><strong>Picture</strong> class definition</td>
<td>261</td>
</tr>
<tr>
<td><strong>Constructors</strong></td>
<td>262</td>
</tr>
<tr>
<td>Default constructor</td>
<td>263</td>
</tr>
<tr>
<td>Copy constructor</td>
<td>265</td>
</tr>
<tr>
<td><strong>Destructor</strong></td>
<td>266</td>
</tr>
<tr>
<td><strong>Assignment</strong></td>
<td>267</td>
</tr>
<tr>
<td>Adding elements</td>
<td>268</td>
</tr>
<tr>
<td><strong>Add Picture</strong></td>
<td>269</td>
</tr>
<tr>
<td><strong>Add Shape</strong></td>
<td>270</td>
</tr>
<tr>
<td><strong>Add Label</strong></td>
<td>272</td>
</tr>
<tr>
<td><strong>Suppress Labels</strong></td>
<td>274</td>
</tr>
<tr>
<td><strong>Unsuppress Labels</strong></td>
<td>275</td>
</tr>
<tr>
<td><strong>Kill Labels</strong></td>
<td>276</td>
</tr>
<tr>
<td>Transformations</td>
<td>278</td>
</tr>
<tr>
<td><strong>Affine transformations</strong></td>
<td>279</td>
</tr>
<tr>
<td>Scale</td>
<td>280</td>
</tr>
<tr>
<td>Shift</td>
<td>282</td>
</tr>
<tr>
<td><strong>real version</strong></td>
<td>283</td>
</tr>
<tr>
<td><strong>Point version</strong></td>
<td>285</td>
</tr>
<tr>
<td>Rotation around the main axes</td>
<td>286</td>
</tr>
<tr>
<td>Rotation around an arbitrary axis</td>
<td>288</td>
</tr>
<tr>
<td><strong>Set transform</strong></td>
<td>289</td>
</tr>
<tr>
<td><strong>Multiplying transform</strong></td>
<td>291</td>
</tr>
<tr>
<td><strong>Show</strong></td>
<td>293</td>
</tr>
<tr>
<td><strong>Show transform</strong></td>
<td>295</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>297</td>
</tr>
<tr>
<td><strong>Focus argument</strong></td>
<td>298</td>
</tr>
<tr>
<td>No <strong>Focus</strong> argument</td>
<td>299</td>
</tr>
<tr>
<td><strong>Clear</strong></td>
<td>300</td>
</tr>
<tr>
<td><strong>Reset transform</strong></td>
<td>301</td>
</tr>
<tr>
<td>Global variables</td>
<td>302</td>
</tr>
<tr>
<td>Putting <strong>Picture</strong> and <strong>Label</strong> together</td>
<td>304</td>
</tr>
<tr>
<td><strong>Point</strong> (points.web)</td>
<td>307</td>
</tr>
<tr>
<td>Include files</td>
<td>308</td>
</tr>
<tr>
<td><strong>Point</strong> class definition</td>
<td>309</td>
</tr>
<tr>
<td>Type definitions and utility structures</td>
<td>311</td>
</tr>
</tbody>
</table>
point_pair and bool_point_pair ........................................ 312 81
bool_point ............................................................... 313 81
bool_point_quadruple ..................................................... 315 83
  Default Constructor for bool_point_quadruple ...................... 316 84
bool_real_point .......................................................... 317 84
  Default Constructor for bool_real_point .......................... 318 85
Global constants .................................................................... 319 85
Constructors and setting functions ........................................ 321 85
  Initialize coordinates and limits ...................................... 323 86
  Default version ......................................................... 324 86
Three real values .................................................................. 326 87
  Constructor ..................................................................... 327 87
  Setting function ............................................................. 329 87
Copy constructor .................................................................... 331 88
  Setting function ............................................................. 333 88
Pseudo-constructor for dynamic allocation ............................... 335 89
  Pointer argument .......................................................... 336 89
  Reference argument ....................................................... 337 89
Destructor ........................................................................... 338 89
Assignment ......................................................................... 340 90
Set on free store .................................................................... 342 90
Clear .................................................................................. 344 91
Clean ................................................................................... 346 91
Returning elements and information ........................................ 348 92
  Is identity ......................................................................... 349 92
  Epsilon ............................................................................ 350 92
  Get Line .......................................................................... 352 92
Getting coordinates .................................................................. 353 92
  Get all coordinates .......................................................... 354 92
    Non-const version ....................................................... 355 93
    const version .................................................................. 357 93
  Get coord ........................................................................ 359 93
    Non-const version ....................................................... 360 94
    const version .................................................................. 362 94
  Get x ............................................................................. 364 95
    Non-const version ....................................................... 365 95
    const version .................................................................. 367 95
  Get y ............................................................................. 369 95
    Non-const version ....................................................... 370 95
    const version .................................................................. 372 96
  Get z ............................................................................. 374 96
    Non-const version ....................................................... 375 96
    const version .................................................................. 377 96
  Get w ............................................................................. 379 96
    Non-const version ....................................................... 380 97
    const version .................................................................. 382 97
Get transform ......................................................................... 384 97
Get copy ............................................................................... 385 97
Is on free store ....................................................................... 387 98
Slope .................................................................................. 389 98
Is on segment ......................................................................... 392 99
  Non-const version .......................................................... 393 100
| const version | 396 102 |
| Is on line | 398 103 |
| Is on Plane | 400 103 |
| Is in triangle | 401 103 |
| Transformations | 402 103 |
| Affine transformations | 403 103 |
| Rotation around the main axes | 404 104 |
| Scale | 406 104 |
| Shear | 408 104 |
| Shift | 410 104 |
| Point versions | 411 104 |
| Three real arguments | 412 105 |
| Point argument | 414 105 |
| Transform version | 416 105 |
| Picture version | 417 105 |
| Shift times | 418 105 |
| Three real arguments | 419 106 |
| Point argument | 421 106 |
| Alignment with an axis | 423 106 |
| Normalize point | 433 110 |
| Rotation around an arbitrary axis | 434 110 |
| Point versions | 435 110 |
| Point arguments | 436 111 |
| Path argument | 438 111 |
| Transform version | 439 112 |
| Picture version | 440 113 |
| Projection | 441 113 |
| Focus argument | 442 113 |
| Parallel projection | 444 115 |
| Perspective projection | 445 116 |
| No Focus argument | 446 117 |
| Applying transformations | 448 117 |
| Set transform to identity | 450 118 |
| Drawing | 452 119 |
| Drawdot | 453 119 |
| Normal version | 454 119 |
| Picture argument first | 456 120 |
| Undrawdot | 458 120 |
| Picture argument first | 460 120 |
| Draw | 462 121 |
| Normal version | 463 121 |
| Picture argument first | 464 121 |
| Draw arrow | 465 121 |
| Normal version | 466 121 |
| Picture argument first | 467 122 |
| Undraw | 468 122 |
| Normal version | 469 122 |
| Picture argument first | 470 122 |
| Draw help | 471 122 |
| Normal version | 472 122 |
| Picture argument first | 473 123 |
| Showing | 474 123 |
### TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show</td>
<td>475</td>
</tr>
<tr>
<td>Show transform</td>
<td>477</td>
</tr>
<tr>
<td>Outputting</td>
<td>479</td>
</tr>
<tr>
<td>Output operator</td>
<td>480</td>
</tr>
<tr>
<td>Suppress output</td>
<td>482</td>
</tr>
<tr>
<td>Unsuppress output</td>
<td>484</td>
</tr>
<tr>
<td>Extract</td>
<td>486</td>
</tr>
<tr>
<td>Get extremes</td>
<td>488</td>
</tr>
<tr>
<td>Get minimum z</td>
<td>489</td>
</tr>
<tr>
<td>Get maximum z</td>
<td>491</td>
</tr>
<tr>
<td>Get mean z</td>
<td>493</td>
</tr>
<tr>
<td>Set extremes</td>
<td>495</td>
</tr>
<tr>
<td>Comparison classes</td>
<td>497</td>
</tr>
<tr>
<td>Compare minimum z</td>
<td>498</td>
</tr>
<tr>
<td>Compare maximum z</td>
<td>499</td>
</tr>
<tr>
<td>Compare mean z</td>
<td>500</td>
</tr>
<tr>
<td>Output</td>
<td>501</td>
</tr>
<tr>
<td>Label</td>
<td>503</td>
</tr>
<tr>
<td><code>string</code> argument</td>
<td>504</td>
</tr>
<tr>
<td><code>short</code> argument</td>
<td>505</td>
</tr>
<tr>
<td>Dotlabel</td>
<td>507</td>
</tr>
<tr>
<td><code>string</code> argument</td>
<td>509</td>
</tr>
<tr>
<td><code>short</code> argument</td>
<td>510</td>
</tr>
<tr>
<td>Get copy of Label</td>
<td>512</td>
</tr>
<tr>
<td>Output Labels</td>
<td>515</td>
</tr>
<tr>
<td>Matrix operations</td>
<td>516</td>
</tr>
<tr>
<td>Multiplication by a Transform with assignment</td>
<td>517</td>
</tr>
<tr>
<td>Vector operations</td>
<td>518</td>
</tr>
<tr>
<td>Vector addition</td>
<td>520</td>
</tr>
<tr>
<td>Vector addition with assignment</td>
<td>521</td>
</tr>
<tr>
<td>Vector subtraction</td>
<td>523</td>
</tr>
<tr>
<td>Vector subtraction with assignment</td>
<td>525</td>
</tr>
<tr>
<td>Vector-scalar multiplication with assignment</td>
<td>527</td>
</tr>
<tr>
<td>Vector-scalar multiplication</td>
<td>529</td>
</tr>
<tr>
<td>Member version (Point first)</td>
<td>531</td>
</tr>
<tr>
<td>Non-member version (scalar first)</td>
<td>532</td>
</tr>
<tr>
<td>Unary minus</td>
<td>534</td>
</tr>
<tr>
<td>Vector-scalar division with assignment</td>
<td>536</td>
</tr>
<tr>
<td>Vector-scalar division</td>
<td>538</td>
</tr>
<tr>
<td>Dot product</td>
<td>540</td>
</tr>
<tr>
<td>Cross product</td>
<td>542</td>
</tr>
<tr>
<td>Magnitude</td>
<td>544</td>
</tr>
<tr>
<td>Angle between two vectors</td>
<td>546</td>
</tr>
<tr>
<td>Unit vector</td>
<td>548</td>
</tr>
<tr>
<td>With assignment</td>
<td>550</td>
</tr>
<tr>
<td><code>const</code> (no assignment)</td>
<td>551</td>
</tr>
<tr>
<td>Mediation</td>
<td>553</td>
</tr>
<tr>
<td>Get normal</td>
<td>555</td>
</tr>
<tr>
<td>Comparison</td>
<td>557</td>
</tr>
<tr>
<td>Equality</td>
<td>558</td>
</tr>
<tr>
<td>Non-<code>const</code> version</td>
<td>559</td>
</tr>
<tr>
<td>const version</td>
<td>567 151</td>
</tr>
<tr>
<td>---------------</td>
<td>--------</td>
</tr>
<tr>
<td>Inequality</td>
<td>569 151</td>
</tr>
<tr>
<td>Intersection</td>
<td>571 152</td>
</tr>
<tr>
<td>Vector version</td>
<td>572 152</td>
</tr>
<tr>
<td>Trace version</td>
<td>573 152</td>
</tr>
<tr>
<td>Picture functions</td>
<td>586 159</td>
</tr>
<tr>
<td>Assignment operator</td>
<td>587 159</td>
</tr>
<tr>
<td>Copy constructor</td>
<td>588 159</td>
</tr>
<tr>
<td>Combining Pictures</td>
<td>589 160</td>
</tr>
<tr>
<td>Clear Picture</td>
<td>590 160</td>
</tr>
<tr>
<td>Output</td>
<td>591 161</td>
</tr>
<tr>
<td>Focus argument</td>
<td>592 161</td>
</tr>
<tr>
<td>No Focus argument</td>
<td>598 165</td>
</tr>
<tr>
<td>Focus</td>
<td>599 165</td>
</tr>
<tr>
<td>Focus class definition</td>
<td>600 165</td>
</tr>
<tr>
<td>Constructors and setting functions</td>
<td>601 165</td>
</tr>
<tr>
<td>Default constructor</td>
<td>602 166</td>
</tr>
<tr>
<td>real arguments</td>
<td>603 166</td>
</tr>
<tr>
<td>Constructor</td>
<td>604 166</td>
</tr>
<tr>
<td>Setting function</td>
<td>606 168</td>
</tr>
<tr>
<td>Point arguments</td>
<td>608 168</td>
</tr>
<tr>
<td>Constructor</td>
<td>609 168</td>
</tr>
<tr>
<td>Setting function</td>
<td>611 168</td>
</tr>
<tr>
<td>Assignment</td>
<td>613 168</td>
</tr>
<tr>
<td>Reset angle</td>
<td>615 169</td>
</tr>
<tr>
<td>Show</td>
<td>617 170</td>
</tr>
<tr>
<td>Returning elements and information</td>
<td>619 171</td>
</tr>
<tr>
<td>Get position</td>
<td>620 171</td>
</tr>
<tr>
<td>Get direction</td>
<td>621 171</td>
</tr>
<tr>
<td>Get distance</td>
<td>622 171</td>
</tr>
<tr>
<td>Get up</td>
<td>623 171</td>
</tr>
<tr>
<td>Get transform</td>
<td>624 172</td>
</tr>
<tr>
<td>Get transform element</td>
<td>625 172</td>
</tr>
<tr>
<td>Get persp</td>
<td>627 173</td>
</tr>
<tr>
<td>Get persp element</td>
<td>628 173</td>
</tr>
<tr>
<td>Global variables</td>
<td>630 173</td>
</tr>
<tr>
<td>Putting Point and Focus together</td>
<td>632 174</td>
</tr>
<tr>
<td>This is what's compiled</td>
<td>633 174</td>
</tr>
<tr>
<td>This is what's written to points.h</td>
<td>634 174</td>
</tr>
<tr>
<td>Line (lines.web)</td>
<td>635 174</td>
</tr>
<tr>
<td>Include files</td>
<td>636 175</td>
</tr>
<tr>
<td>Line struct definition</td>
<td>637 175</td>
</tr>
<tr>
<td>Constructors</td>
<td>638 175</td>
</tr>
<tr>
<td>Default constructor</td>
<td>639 175</td>
</tr>
<tr>
<td>Copy constructor</td>
<td>641 176</td>
</tr>
<tr>
<td>Assignment</td>
<td>643 176</td>
</tr>
<tr>
<td>Get Line</td>
<td>645 177</td>
</tr>
<tr>
<td>Get Path</td>
<td>646 177</td>
</tr>
<tr>
<td>Intersection</td>
<td>647 177</td>
</tr>
<tr>
<td>Get distance</td>
<td>648 178</td>
</tr>
<tr>
<td>Show</td>
<td>652 181</td>
</tr>
<tr>
<td>Global constants for Line</td>
<td>654 181</td>
</tr>
</tbody>
</table>
Putting Line together .......................................................................................... 656 181
Plane \texttt{(planes.web)} .................................................................................. 659 182
Include files ..................................................................................................... 660 182
Plane struct definition ..................................................................................... 661 182
Constructors .................................................................................................... 662 182
    Default constructor .................................................................................... 663 182
    Copy constructor ....................................................................................... 665 183
Point arguments ............................................................................................... 667 184
Assignment ...................................................................................................... 669 185
Comparing Planes ............................................................................................ 671 186
    Equality ...................................................................................................... 672 186
    Inequality ................................................................................................. 674 186
Get distance ...................................................................................................... 676 187
    Point argument ......................................................................................... 677 187
    No argument ............................................................................................. 679 187
Point is on Plane ............................................................................................. 681 188
Intersection ...................................................................................................... 682 188
    Intersection with a line ............................................................................. 683 188
        Point arguments .................................................................................. 684 188
        Path argument .................................................................................... 686 189
    Intersection of two Planes ....................................................................... 687 189
Show .................................................................................................................. 689 191
Global constants for Plane ............................................................................. 691 191
Putting Plane together .................................................................................... 693 191
Path \texttt{(paths.web)} .................................................................................... 696 193
Include files ..................................................................................................... 697 193
Path class definition ....................................................................................... 698 193
Static member variable definitions .................................................................. 699 194
Assignment ...................................................................................................... 700 194
Constructors and setting functions ................................................................ 702 195
    Discard points and connectors ............................................................... 703 195
    Default constructor .................................................................................. 704 195
Lines .................................................................................................................. 706 196
    Constructor .............................................................................................. 707 196
    Setting function ...................................................................................... 709 197
Points and one type of connector .................................................................. 711 197
    Constructor .............................................................................................. 712 198
    Setting function ...................................................................................... 714 198
Variable number of Points and connectors ................................................... 716 199
    Constructor .............................................................................................. 717 199
    Setting function ...................................................................................... 719 200
Copy constructor ............................................................................................. 721 201
Pseudo-constructor for dynamic allocation ....................................................... 723 202
    Pointer argument ..................................................................................... 724 202
    Reference argument .................................................................................. 725 202
Destructor ........................................................................................................ 726 202
Clear .................................................................................................................. 728 203
Get copy ........................................................................................................... 730 204
Set on free store ............................................................................................. 732 205
Setting drawing and filling data ..................................................................... 734 205
    Set \texttt{fill\_draw\_value} ......................................................................... 735 205
    Set draw color ....................................................................................... 737 205
<p>| Color version | 738 205 |
| Color pointer version | 740 206 |
| Set fill color | 742 206 |
| Color version | 743 206 |
| Color pointer version | 745 206 |
| Set dash pattern | 747 206 |
| Set pen | 749 207 |
| Set connectors | 751 207 |
| Transformations | 753 207 |
| Affine transformations | 754 207 |
| Rotation | 755 207 |
| Rotation around the main axes | 756 207 |
| Rotation around an arbitrary axis | 758 208 |
| Transform version | 759 208 |
| Point version | 760 209 |
| Path versions | 761 209 |
| Path arguments | 762 209 |
| Point arguments | 764 210 |
| Scale | 766 210 |
| Shear | 768 210 |
| Shift | 770 211 |
| real arguments | 771 211 |
| Point argument | 773 211 |
| Shift times | 775 211 |
| real arguments | 776 212 |
| Point argument | 778 212 |
| Applying transformations | 780 212 |
| Multiplying by a Transform | 781 212 |
| Applying transform to points | 783 212 |
| Projection | 785 213 |
| Functions for lines | 787 213 |
| Alignment with an axis | 788 213 |
| For lines | 789 213 |
| No assignment | 790 213 |
| With assignment | 792 214 |
| For non-lines | 794 215 |
| Adding Points to Paths | 796 215 |
| With assignment | 797 215 |
| Without assignment | 799 215 |
| Adding connectors to Paths | 801 216 |
| Concatenating Paths | 803 216 |
| Versions using “&amp;” | 804 216 |
| With assignment | 805 216 |
| Without assignment | 810 217 |
| Appending with a connector argument | 812 218 |
| Drawing and filling | 815 218 |
| Draw | 816 218 |
| Path versions | 817 219 |
| Normal version | 818 219 |
| Picture argument first | 820 220 |
| Point versions | 822 220 |
| Normal version | 823 220 |
| Picture argument first | 824 220 |
| Draw arrow | 825 220 |
| Path versions | 826 220 |
| Normal version | 827 221 |
| Picture argument first | 829 221 |
| Point versions | 831 221 |
| Normal version | 832 221 |
| Picture argument first | 833 222 |
| Draw help | 834 222 |
| Path versions | 835 222 |
| Normal version | 836 222 |
| Picture argument first | 838 222 |
| Point versions | 840 223 |
| Normal version | 841 223 |
| Picture argument first | 842 223 |
| Fill | 843 223 |
| Normal version | 844 223 |
| Picture argument first | 846 224 |
| Filldraw | 848 225 |
| Normal version | 849 225 |
| Picture argument first | 851 226 |
| Undraw | 853 227 |
| Path versions | 854 227 |
| Normal version | 855 227 |
| Picture argument first | 857 227 |
| Point versions | 859 227 |
| Normal version | 860 228 |
| Picture argument first | 861 228 |
| Unfill | 862 228 |
| Normal version | 863 228 |
| Unfilldraw | 865 229 |
| Normal version | 866 229 |
| Picture argument first | 868 230 |
| Labelling | 870 231 |
| Label | 871 231 |
| Normal version | 872 231 |
| Picture argument first | 874 232 |
| Dotlabel | 876 232 |
| Normal version | 877 233 |
| Picture argument first | 879 233 |
| Outputting | 881 233 |
| Extract | 882 233 |
| Set extremes | 884 234 |
| Get extremes | 888 236 |
| Get minimum z | 889 236 |
| Get maximum z | 891 237 |
| Get mean z | 893 237 |
| Suppress output | 895 237 |
| Unsuppress output | 897 238 |
| Output | 899 238 |
| Showing | 908 244 |
| Show | 909 244 |</p>
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Colors</td>
<td>911</td>
</tr>
<tr>
<td>Returning elements and information</td>
<td>913</td>
</tr>
<tr>
<td>Is on free store</td>
<td>914</td>
</tr>
<tr>
<td>Is planar</td>
<td>916</td>
</tr>
<tr>
<td>Is linear</td>
<td>918</td>
</tr>
<tr>
<td>Get line switch</td>
<td>920</td>
</tr>
<tr>
<td>Test for cycles</td>
<td>921</td>
</tr>
<tr>
<td>Size (number of points)</td>
<td>922</td>
</tr>
<tr>
<td>Slope</td>
<td>923</td>
</tr>
<tr>
<td>Subpath</td>
<td>925</td>
</tr>
<tr>
<td>Get point</td>
<td>931</td>
</tr>
<tr>
<td>non-const version</td>
<td>932</td>
</tr>
<tr>
<td>const version</td>
<td>934</td>
</tr>
<tr>
<td>Get last point</td>
<td>936</td>
</tr>
<tr>
<td>Get size</td>
<td>938</td>
</tr>
<tr>
<td>Get normal</td>
<td>939</td>
</tr>
<tr>
<td>Path version</td>
<td>940</td>
</tr>
<tr>
<td>Point version</td>
<td>945</td>
</tr>
<tr>
<td>Get plane</td>
<td>946</td>
</tr>
<tr>
<td>Point lies within triangle</td>
<td>948</td>
</tr>
<tr>
<td>Manipulating Paths</td>
<td>951</td>
</tr>
<tr>
<td>Set cycle</td>
<td>952</td>
</tr>
<tr>
<td>Reverse</td>
<td>954</td>
</tr>
<tr>
<td>With assignment</td>
<td>955</td>
</tr>
<tr>
<td>No assignment</td>
<td>959</td>
</tr>
<tr>
<td>Equality</td>
<td>961</td>
</tr>
<tr>
<td>Intersection</td>
<td>963</td>
</tr>
<tr>
<td>Intersection of two linear Paths</td>
<td>964</td>
</tr>
<tr>
<td>Intersection of a linear Path with a Plane</td>
<td>966</td>
</tr>
<tr>
<td>Drawing axes</td>
<td>967</td>
</tr>
<tr>
<td>Length argument first</td>
<td>968</td>
</tr>
<tr>
<td>Color argument first</td>
<td>973</td>
</tr>
<tr>
<td>Paths and Lines</td>
<td>975</td>
</tr>
<tr>
<td>Get Line</td>
<td>976</td>
</tr>
<tr>
<td>Get Path</td>
<td>978</td>
</tr>
<tr>
<td>Putting Path together</td>
<td>979</td>
</tr>
<tr>
<td>Curves (curves.web)</td>
<td>982</td>
</tr>
<tr>
<td>Include files</td>
<td>983</td>
</tr>
<tr>
<td>Regular closed plane curve</td>
<td>984</td>
</tr>
<tr>
<td>Reg_Cl_Plane_Curve class definition</td>
<td>985</td>
</tr>
<tr>
<td>Returning elements and information</td>
<td>986</td>
</tr>
<tr>
<td>Is quadratic</td>
<td>987</td>
</tr>
<tr>
<td>Is cubic</td>
<td>988</td>
</tr>
<tr>
<td>Is quartic</td>
<td>989</td>
</tr>
<tr>
<td>Get coefficients</td>
<td>990</td>
</tr>
<tr>
<td>Solve</td>
<td>991</td>
</tr>
<tr>
<td>Location of a point</td>
<td>992</td>
</tr>
<tr>
<td>Angle point</td>
<td>994</td>
</tr>
<tr>
<td>Intersection points</td>
<td>996</td>
</tr>
<tr>
<td>Point arguments</td>
<td>997</td>
</tr>
<tr>
<td>Degenerate cases, error handling</td>
<td>999</td>
</tr>
<tr>
<td>Parallel and coplanar cases</td>
<td>1000</td>
</tr>
</tbody>
</table>
### Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coplanar case</td>
<td>1001</td>
</tr>
<tr>
<td>Parallel case</td>
<td>1006</td>
</tr>
<tr>
<td>Perpendicular and non-coplanar cases</td>
<td>1007</td>
</tr>
<tr>
<td><strong>Path arguments</strong></td>
<td>1008</td>
</tr>
<tr>
<td><strong>Reg_Cl_Plane_Curve segments</strong></td>
<td>1010</td>
</tr>
<tr>
<td>Segment</td>
<td>1011</td>
</tr>
<tr>
<td>Half</td>
<td>1013</td>
</tr>
<tr>
<td>Quarter</td>
<td>1014</td>
</tr>
<tr>
<td>Putting <strong>Reg_Cl_Plane_Curve</strong> together</td>
<td>1015</td>
</tr>
<tr>
<td><strong>Polygon (polygons.web)</strong></td>
<td>1017</td>
</tr>
<tr>
<td>Include files</td>
<td>1018</td>
</tr>
<tr>
<td><strong>Polygon class definition</strong></td>
<td>1019</td>
</tr>
<tr>
<td>Returning elements and information</td>
<td>1020</td>
</tr>
<tr>
<td>Get center</td>
<td>1021</td>
</tr>
<tr>
<td>non-const version</td>
<td>1022</td>
</tr>
<tr>
<td>const version</td>
<td>1024</td>
</tr>
<tr>
<td><strong>Intersections</strong></td>
<td>1026</td>
</tr>
<tr>
<td>Intersection with a line</td>
<td>1027</td>
</tr>
<tr>
<td><strong>Point version</strong></td>
<td>1028</td>
</tr>
<tr>
<td>Degenerate cases, error handling</td>
<td>1030</td>
</tr>
<tr>
<td>Parallel and coplanar cases</td>
<td>1031</td>
</tr>
<tr>
<td>Coplanar case</td>
<td>1032</td>
</tr>
<tr>
<td>Parallel case</td>
<td>1033</td>
</tr>
<tr>
<td>Perpendicular and non-coplanar cases</td>
<td>1034</td>
</tr>
<tr>
<td>End of definition</td>
<td>1036</td>
</tr>
<tr>
<td><strong>Path version</strong></td>
<td>1037</td>
</tr>
<tr>
<td>Intersection with another <strong>Polygon</strong></td>
<td>1039</td>
</tr>
<tr>
<td>Coplanar case</td>
<td>1041</td>
</tr>
<tr>
<td>Parallel case</td>
<td>1042</td>
</tr>
<tr>
<td>Non-parallel, non-coplanar case</td>
<td>1043</td>
</tr>
<tr>
<td><strong>Transformations</strong></td>
<td>1044</td>
</tr>
<tr>
<td>Applying a transformation</td>
<td>1045</td>
</tr>
<tr>
<td>Rotation around the main axes</td>
<td>1047</td>
</tr>
<tr>
<td>Rotate around an arbitrary axis</td>
<td>1049</td>
</tr>
<tr>
<td><strong>Point arguments</strong></td>
<td>1050</td>
</tr>
<tr>
<td><strong>Path argument</strong></td>
<td>1052</td>
</tr>
<tr>
<td>Scale</td>
<td>1054</td>
</tr>
<tr>
<td>Shear</td>
<td>1056</td>
</tr>
<tr>
<td>Shift</td>
<td>1058</td>
</tr>
<tr>
<td>real arguments</td>
<td>1059</td>
</tr>
<tr>
<td><strong>Point argument</strong></td>
<td>1061</td>
</tr>
<tr>
<td>Shift times</td>
<td>1063</td>
</tr>
<tr>
<td>real arguments</td>
<td>1064</td>
</tr>
<tr>
<td><strong>Point argument</strong></td>
<td>1066</td>
</tr>
<tr>
<td><strong>Reg_Polygon (polygons.web)</strong></td>
<td>1068</td>
</tr>
<tr>
<td><strong>Reg_Polygon class definition</strong></td>
<td>1069</td>
</tr>
<tr>
<td>Assignment</td>
<td>1070</td>
</tr>
<tr>
<td>Constructors and setting functions</td>
<td>1072</td>
</tr>
<tr>
<td>Default constructor</td>
<td>1073</td>
</tr>
<tr>
<td>Center, sides, diameter, and angles</td>
<td>1075</td>
</tr>
<tr>
<td>Constructor</td>
<td>1076</td>
</tr>
<tr>
<td>Setting function</td>
<td>1079</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Pseudo-constructor for dynamic allocation</td>
<td>1082</td>
</tr>
<tr>
<td>Pointer argument</td>
<td>1083</td>
</tr>
<tr>
<td>Reference argument</td>
<td>1084</td>
</tr>
<tr>
<td>Destructor</td>
<td>1085</td>
</tr>
<tr>
<td>Returning elements and information</td>
<td>1086</td>
</tr>
<tr>
<td>Get radius</td>
<td>1087</td>
</tr>
<tr>
<td>Circles</td>
<td>1088</td>
</tr>
<tr>
<td>Enclosed circle</td>
<td>1089</td>
</tr>
<tr>
<td>Draw enclosed circle</td>
<td>1090</td>
</tr>
<tr>
<td>Normal version</td>
<td>1091</td>
</tr>
<tr>
<td>Picture argument first</td>
<td>1092</td>
</tr>
<tr>
<td>Surrounding circle</td>
<td>1093</td>
</tr>
<tr>
<td>Normal version</td>
<td>1094</td>
</tr>
<tr>
<td>Picture argument first</td>
<td>1095</td>
</tr>
<tr>
<td>Putting polygons together</td>
<td>1096</td>
</tr>
<tr>
<td><strong>Rectangle (rectangles.web)</strong></td>
<td>1097</td>
</tr>
<tr>
<td>Include files</td>
<td>1100</td>
</tr>
<tr>
<td><strong>Rectangle class definition</strong></td>
<td>1101</td>
</tr>
<tr>
<td>Constructors and setting functions</td>
<td>1102</td>
</tr>
<tr>
<td>Default constructor</td>
<td>1103</td>
</tr>
<tr>
<td>Center, lengths, and angles</td>
<td>1105</td>
</tr>
<tr>
<td>Constructor</td>
<td>1106</td>
</tr>
<tr>
<td>Setting function</td>
<td>1108</td>
</tr>
<tr>
<td><strong>Four Points</strong></td>
<td>1110</td>
</tr>
<tr>
<td>Constructor</td>
<td>1111</td>
</tr>
<tr>
<td>Setting function</td>
<td>1113</td>
</tr>
<tr>
<td>Pseudo-constructor for dynamic allocation</td>
<td>1115</td>
</tr>
<tr>
<td>Pointer argument</td>
<td>1116</td>
</tr>
<tr>
<td>Reference argument</td>
<td>1117</td>
</tr>
<tr>
<td>Destructor</td>
<td>1118</td>
</tr>
<tr>
<td>Assignment</td>
<td>1119</td>
</tr>
<tr>
<td>Returning Elements and information</td>
<td>1121</td>
</tr>
<tr>
<td>Is rectangular</td>
<td>1122</td>
</tr>
<tr>
<td>Returning Points</td>
<td>1124</td>
</tr>
<tr>
<td>Corner</td>
<td>1125</td>
</tr>
<tr>
<td>Get Mid-point</td>
<td>1127</td>
</tr>
<tr>
<td>Getting axes</td>
<td>1129</td>
</tr>
<tr>
<td>Get axis _h</td>
<td>1130</td>
</tr>
<tr>
<td>Get axis _v</td>
<td>1132</td>
</tr>
<tr>
<td><strong>Ellipses</strong></td>
<td>1134</td>
</tr>
<tr>
<td>Surrounding Ellipse</td>
<td>1135</td>
</tr>
<tr>
<td>Enclosed Ellipse</td>
<td>1136</td>
</tr>
<tr>
<td>Draw surrounding Ellipse</td>
<td>1137</td>
</tr>
<tr>
<td>Draw enclosed Ellipse</td>
<td>1138</td>
</tr>
<tr>
<td>Putting Rectangle together</td>
<td>1139</td>
</tr>
<tr>
<td><strong>Ellipse (ellipses.web)</strong></td>
<td>1141</td>
</tr>
<tr>
<td>Include files</td>
<td>1142</td>
</tr>
<tr>
<td><strong>Ellipse class definition</strong></td>
<td>1143</td>
</tr>
<tr>
<td>Static data members</td>
<td>1144</td>
</tr>
<tr>
<td>Constructors</td>
<td>1145</td>
</tr>
<tr>
<td>Default constructor</td>
<td>1146</td>
</tr>
</tbody>
</table>
Center, lengths, and angles of rotation .................................................. 1148 319
Constructor .................................................................................. 1149 319
Setting function ............................................................................. 1151 321
Pseudo-constructor for dynamic allocation ........................................ 1153 321
  Pointer argument ........................................................................ 1154 321
  Reference argument ..................................................................... 1155 322
Destructor ...................................................................................... 1156 322
Assignment .................................................................................. 1157 322
Labelling ....................................................................................... 1159 322
Label ........................................................................................... 1160 322
dotlabel ......................................................................................... 1162 323
Returning elements and information .................................................. 1163 323
  Is elliptical ............................................................................... 1164 323
  Is quadratic .............................................................................. 1166 325
  Is cubic ..................................................................................... 1167 325
  Is quartic ................................................................................... 1169 326
Solve .............................................................................................. 1171 326
Get coefficients ............................................................................ 1174 328
Get center ..................................................................................... 1176 328
  Non-const version .................................................................... 1177 329
  const version ........................................................................... 1179 329
Get focus ...................................................................................... 1181 329
  Non-const version .................................................................... 1182 329
  const version ........................................................................... 1184 330
Get linear eccentricity ................................................................... 1186 331
Get numerical eccentricity .............................................................. 1188 331
Get axes ....................................................................................... 1190 332
  Get vertical axis ...................................................................... 1191 332
    const version ....................................................................... 1192 332
    Non-const version ................................................................ 1194 332
  Get horizontal axis .................................................................. 1196 332
    const version ....................................................................... 1197 333
    Non-const version ................................................................ 1199 333
Angle point .................................................................................... 1201 333
Equality ......................................................................................... 1203 334
Location of a point ........................................................................ 1205 335
Intersection points ......................................................................... 1207 336
  Point arguments ....................................................................... 1208 336
  Path argument ......................................................................... 1210 337
  Ellipse argument ..................................................................... 1212 337
    Check intersection point locations ....................................... 1228 355
Transformations .............................................................................. 1229 355
  Performing a transformation ..................................................... 1230 355
    Do transform ......................................................................... 1231 356
    Operator ............................................................................... 1233 358
  Rotation around the main axes ............................................... 1235 358
  Scale ....................................................................................... 1237 358
  Shear ....................................................................................... 1239 359
  Shift ......................................................................................... 1241 359
    real arguments ..................................................................... 1242 359
    Point argument ..................................................................... 1244 359
  Shift times ............................................................................... 1246 360
<table>
<thead>
<tr>
<th>TABLE OF CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>real arguments ...........................................</td>
</tr>
<tr>
<td>Point argument ...........................................</td>
</tr>
<tr>
<td>Rotation around an arbitrary axis .......................</td>
</tr>
<tr>
<td>Point arguments ..........................................</td>
</tr>
<tr>
<td>Path arguments ............................................</td>
</tr>
<tr>
<td>Rectangles ...............................................</td>
</tr>
<tr>
<td>Surrounding rectangle ....................................</td>
</tr>
<tr>
<td>Inscribed rectangle ......................................</td>
</tr>
<tr>
<td>Draw surrounding rectangle ..............................</td>
</tr>
<tr>
<td>Draw inscribed rectangle ...............................</td>
</tr>
<tr>
<td><strong>Rectangle functions</strong> ................................</td>
</tr>
<tr>
<td>Ellipses ................................................</td>
</tr>
<tr>
<td>Surrounding Ellipse .....................................</td>
</tr>
<tr>
<td>Enclosed Ellipse ........................................</td>
</tr>
<tr>
<td>Draw surrounding Ellipse ................................</td>
</tr>
<tr>
<td>Draw enclosed Ellipse ...................................</td>
</tr>
<tr>
<td>Putting Ellipse together ................................</td>
</tr>
<tr>
<td>Circle (circles.web) ....................................</td>
</tr>
<tr>
<td>Include files ............................................</td>
</tr>
<tr>
<td>Circle class definition ................................</td>
</tr>
<tr>
<td>Constructors and setting functions .....................</td>
</tr>
<tr>
<td>Default constructor ....................................</td>
</tr>
<tr>
<td>Center, diameters and angles ...........................</td>
</tr>
<tr>
<td>Constructor ..............................................</td>
</tr>
<tr>
<td>Setting function ........................................</td>
</tr>
<tr>
<td>Pseudo-constructor for dynamic allocation .............</td>
</tr>
<tr>
<td>Pointer argument .......................................</td>
</tr>
<tr>
<td>Reference argument .....................................</td>
</tr>
<tr>
<td>Destructor ...............................................</td>
</tr>
<tr>
<td>Assignment ...............................................</td>
</tr>
<tr>
<td>Circle argument ........................................</td>
</tr>
<tr>
<td>Ellipse argument ........................................</td>
</tr>
<tr>
<td>Returning elements and information ....................</td>
</tr>
<tr>
<td>Is circular ..............................................</td>
</tr>
<tr>
<td>Get radius ..............................................</td>
</tr>
<tr>
<td>Get diameter ............................................</td>
</tr>
<tr>
<td>Intersections ............................................</td>
</tr>
<tr>
<td>Point argument .........................................</td>
</tr>
<tr>
<td>Path argument ..........................................</td>
</tr>
<tr>
<td>Circle argument ........................................</td>
</tr>
<tr>
<td><strong>Reg_Polygon functions</strong> ..............................</td>
</tr>
<tr>
<td>Enclosed circle ..........................................</td>
</tr>
<tr>
<td>Draw enclosed circle ....................................</td>
</tr>
<tr>
<td>Normal version .........................................</td>
</tr>
<tr>
<td>Picture argument first ..................................</td>
</tr>
<tr>
<td>Surrounding circle .....................................</td>
</tr>
<tr>
<td>Draw surrounding circle ................................</td>
</tr>
<tr>
<td>Normal version .........................................</td>
</tr>
<tr>
<td>Picture argument first ..................................</td>
</tr>
<tr>
<td>Putting Circle together ................................</td>
</tr>
<tr>
<td><strong>Patterns</strong> (patterns.web) .............................</td>
</tr>
<tr>
<td>Include files ............................................</td>
</tr>
<tr>
<td>Topic</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>Plane tessellations</td>
</tr>
<tr>
<td>Hexagonal tessellation 1</td>
</tr>
<tr>
<td>patterns</td>
</tr>
<tr>
<td>Epicycloid pattern 1</td>
</tr>
<tr>
<td>Epicycloid pattern 2</td>
</tr>
<tr>
<td>Epicycloid pattern 3</td>
</tr>
<tr>
<td>Putting patterns together</td>
</tr>
<tr>
<td>Solid (solids.web)</td>
</tr>
<tr>
<td>Include files</td>
</tr>
<tr>
<td>Solid class definition</td>
</tr>
<tr>
<td>Define static const Solid data members</td>
</tr>
<tr>
<td>Constructors</td>
</tr>
<tr>
<td>Default constructor</td>
</tr>
<tr>
<td>Copy constructor</td>
</tr>
<tr>
<td>Pseudo-constructor for dynamic allocation</td>
</tr>
<tr>
<td>Pointer argument</td>
</tr>
<tr>
<td>Reference argument</td>
</tr>
<tr>
<td>Destructor</td>
</tr>
<tr>
<td>Assignment</td>
</tr>
<tr>
<td>Copying</td>
</tr>
<tr>
<td>Set on free store</td>
</tr>
<tr>
<td>Returning elements and information</td>
</tr>
<tr>
<td>Get center</td>
</tr>
<tr>
<td>Getting Shapes</td>
</tr>
<tr>
<td>Get Shape pointer</td>
</tr>
<tr>
<td>Get Circle pointer</td>
</tr>
<tr>
<td>Get Ellipse pointer</td>
</tr>
<tr>
<td>Get Path pointer</td>
</tr>
<tr>
<td>Get Rectangle pointer</td>
</tr>
<tr>
<td>Get Reg_Polygon pointer</td>
</tr>
<tr>
<td>Getting Shape centers</td>
</tr>
<tr>
<td>Get Shape center</td>
</tr>
<tr>
<td>Get Circle center</td>
</tr>
<tr>
<td>Get Ellipse center</td>
</tr>
<tr>
<td>Get Rectangle center</td>
</tr>
<tr>
<td>Get Reg_Polygon center</td>
</tr>
<tr>
<td>Is on free store</td>
</tr>
<tr>
<td>Show</td>
</tr>
<tr>
<td>Clear</td>
</tr>
<tr>
<td>Transformations</td>
</tr>
<tr>
<td>Multiplying by a Transform</td>
</tr>
<tr>
<td>Applying a transformation</td>
</tr>
<tr>
<td>Scale</td>
</tr>
<tr>
<td>Shear</td>
</tr>
<tr>
<td>Shift</td>
</tr>
<tr>
<td>real arguments</td>
</tr>
<tr>
<td>Point argument</td>
</tr>
<tr>
<td>Rotation around the main axes</td>
</tr>
<tr>
<td>Rotation around an arbitrary axis</td>
</tr>
<tr>
<td>Outputting</td>
</tr>
<tr>
<td>Extract</td>
</tr>
<tr>
<td>Set extremes</td>
</tr>
<tr>
<td>Topic</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>Get extremes</td>
</tr>
<tr>
<td>Get minimum z</td>
</tr>
<tr>
<td>Get maximum z</td>
</tr>
<tr>
<td>Get mean z</td>
</tr>
<tr>
<td>Suppress output</td>
</tr>
<tr>
<td>Uns suppressing output</td>
</tr>
<tr>
<td>Output</td>
</tr>
<tr>
<td>Drawing and filling</td>
</tr>
<tr>
<td>Draw</td>
</tr>
<tr>
<td>Fill</td>
</tr>
<tr>
<td>Filledraw</td>
</tr>
<tr>
<td>Uindraw</td>
</tr>
<tr>
<td>Unfill</td>
</tr>
<tr>
<td>Unfilldraw</td>
</tr>
<tr>
<td>Putting Solid together</td>
</tr>
<tr>
<td>Solid_Faced (soldfaced_web)</td>
</tr>
<tr>
<td>Include files</td>
</tr>
<tr>
<td>Solid_Faced class definition</td>
</tr>
<tr>
<td>Putting Solid_Faced together</td>
</tr>
<tr>
<td>Cuboid (cuboid_web)</td>
</tr>
<tr>
<td>Include files</td>
</tr>
<tr>
<td>Cuboid class definition</td>
</tr>
<tr>
<td>Constructors and setting functions</td>
</tr>
<tr>
<td>Default constructor</td>
</tr>
<tr>
<td>Copy constructor</td>
</tr>
<tr>
<td>Center, height, width, depth, and angles</td>
</tr>
<tr>
<td>Constructs for dynamic allocation</td>
</tr>
<tr>
<td>Pointer argument</td>
</tr>
<tr>
<td>Reference argument</td>
</tr>
<tr>
<td>Destructor</td>
</tr>
<tr>
<td>Assignment</td>
</tr>
<tr>
<td>Putting Cuboid together</td>
</tr>
<tr>
<td>Polyhedra (polyhedra_web)</td>
</tr>
<tr>
<td>Include files</td>
</tr>
<tr>
<td>Polyhedron class definition</td>
</tr>
<tr>
<td>Intersection</td>
</tr>
<tr>
<td>Regular Platonic Polyhedra</td>
</tr>
<tr>
<td>Tetrahedron</td>
</tr>
<tr>
<td>Tetrahedron class definition</td>
</tr>
<tr>
<td>Define static const Tetrahedron data members</td>
</tr>
<tr>
<td>Constructors and setting functions</td>
</tr>
<tr>
<td>Default constructor</td>
</tr>
<tr>
<td>Center, diameter of triangle, and angles</td>
</tr>
<tr>
<td>Constructor</td>
</tr>
<tr>
<td>Setting function</td>
</tr>
<tr>
<td>Get net</td>
</tr>
<tr>
<td>Draw net</td>
</tr>
<tr>
<td>Dodecahedron</td>
</tr>
<tr>
<td>Dodecahedron class definition</td>
</tr>
<tr>
<td>Define static const Dodecahedron data members</td>
</tr>
<tr>
<td>Constructors and setting functions</td>
</tr>
<tr>
<td>Default constructor</td>
</tr>
<tr>
<td>---------------------</td>
</tr>
<tr>
<td>Center, diameter of pentagon, and angles</td>
</tr>
<tr>
<td>Constructor</td>
</tr>
<tr>
<td>Get net</td>
</tr>
<tr>
<td>Draw net</td>
</tr>
<tr>
<td>Icosahedron</td>
</tr>
<tr>
<td>Icosahedron class definition</td>
</tr>
<tr>
<td>Define static const Icosahedron data members</td>
</tr>
<tr>
<td>Constructors and setting functions</td>
</tr>
<tr>
<td>Default constructor</td>
</tr>
<tr>
<td>Center, diameter of triangle, and angles</td>
</tr>
<tr>
<td>Constructor</td>
</tr>
<tr>
<td>Get net</td>
</tr>
<tr>
<td>Draw net</td>
</tr>
<tr>
<td>Semi-Regular Archimedean Polyhedra</td>
</tr>
<tr>
<td>Truncated Octahedron</td>
</tr>
<tr>
<td>Trunc_Octahedron class definition</td>
</tr>
<tr>
<td>Define static const Trunc_Octahedron data members</td>
</tr>
<tr>
<td>Constructors and setting functions</td>
</tr>
<tr>
<td>Default constructor</td>
</tr>
<tr>
<td>Center, diameter of hexagon, and angles</td>
</tr>
<tr>
<td>Constructor</td>
</tr>
<tr>
<td>Get net</td>
</tr>
<tr>
<td>Putting polyhedra together</td>
</tr>
<tr>
<td>Parsing (parser.web)</td>
</tr>
<tr>
<td>Parse</td>
</tr>
<tr>
<td>Putting the parser together</td>
</tr>
<tr>
<td>Main (main.web)</td>
</tr>
<tr>
<td>Include files</td>
</tr>
<tr>
<td>Get input</td>
</tr>
<tr>
<td>Actions in main</td>
</tr>
<tr>
<td>Process command line options</td>
</tr>
<tr>
<td>Print version, copyright, and license information</td>
</tr>
<tr>
<td>Main itself</td>
</tr>
<tr>
<td>Putting Main together</td>
</tr>
<tr>
<td>Appendices</td>
</tr>
<tr>
<td>References</td>
</tr>
<tr>
<td>GNU Free Documentation License</td>
</tr>
<tr>
<td>GNU General Public License</td>
</tr>
<tr>
<td>Index</td>
</tr>
</tbody>
</table>
1. Copyright and License.
Copyright © 2003 Laurence D. Finston.

See the section (GNU Free Documentation License 1.561) for the copying conditions that apply to this document.

The program 3DLDF documented here is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. See the section (GNU General Public License 1.562) in this document.

3DLDF is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with 3DLDF; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

3DLDF is a GNU package. It is part of the GNU Project of the Free Software Foundation and is published under the GNU General Public License. See the website http://www.gnu.org for more information. 3DLDF is available for downloading from http://www.gnu.org/3DLDF. It is also available from http://www.dante.de/software/ctan/, the Dante www-server and from http://wwwuser.gwdg.de/~lfinsto1, the author's website.

Please send bug reports to:
bug-3DLDF@gnu.org

The mailing list help-3DLDF@gnu.org is available for people to ask other users for help. The mailing list info-3DLDF@gnu.org is for sending announcements to users. To subscribe to these mailing lists, send an email with "subscribe (email-address)" as the subject.

The author can be contacted at:
Laurence D. Finston
Kraußbergstr 41
D-37075 Goettingen
Germany
lfinsto1@gwdg.de
s246794@stud.uni-goettingen.de

Web site: http://wwwuser.gwdg.de/~lfinsto1

2. Introduction (3DLDF.web).

This book contains the program code of 3DLDF, along with explanations. For information on using 3DLDF see the 3DLDF User and Reference Manual, which should have been in the distribution along with this book.

3DLDF is a free software package for three-dimensional drawing written by Laurence D. Finston, who is also the author of this manual. It is written in C++ using CWEWEB and it outputs MetaPost code.

In the text sections of the CWEWEB documentation, I note things about C++ that may be of interest even if they are obvious to people with experience.

The various files that make up 3DLDF are tangled and compiled separately (see the chapter "Compiling" in the 3DLDF User and Reference Manual) and the resulting object files are then linked. However, the file 3DLDF.web, which contains no C++ code and is not tangled, includes the other .web files, so that cweave processes them as if they were all one file.
To write my .web files, I wrote a cweb-mode for Emacs and a number of Emacs-Lisp functions to go with it. It is not currently included in the 3DLDF distribution (Version 1.1.5.1), but I may include it in a later version. However, GNU is at work at an official cweb-mode of its own, so you might want to use it instead, if it's available.

Plurals of types are typeset with the “s” in the same font as the type, e.g., “Points” and not “Points”. It's not considered good typographical practice to typeset words with letters from different fonts. The second example does have the advantage that it’s somewhat clearer what the actual name of the type is, but I think the first argument is weightier.

See http://www-cs-faculty.stanford.edu/knuth/cweb.html for more information about CWEB. The WEB (for Pascal) and CWEB packages are available from the CTAN archive, ftp.dante.de and http://www.dante.de.

Donald Knuth’s books \TeX: The Program and METAFONT: The Program each include a section “How to read a WEB”, which may be helpful.

3. Formatting commands. cweave formats “==” as “\equal”, “!=” as “\neq”, and “!” as “\textasciitilde”. Programmers who use these tokens must type them as “==”, “!=”, and “!”.

The following formatting commands are for types defined in C++ or in the C++ Standard Library, but not handled correctly by cweave.

```c
format bitset int
format bool int
format bools bool
format ifstream int
format key_type int
format mapped_type int
format map int
format ofstream int
format stat int
format stringstream int
format numeric_limits int
format pair int
format string char
format tm int
format valarray int
format vector int
```
4. This section contains commands for inputting the CWEB source files, which are invisible in the cweb output.

5. Preprocessor variables and library files (loader.web).

[LDF 2002.10.15.] It would, of course, be possible to put this code into a .h file directly, but it's convenient to have a CWEB file so that it can be cwebbed along with the rest of 3DLDF.

[LDF 2003.07.18.] Set the preprocessor macro LDF_GCC_3_3 to 1 in order to compile using gcc version 3.3 20030226 (prerelease) (SuSE Linux). Set it to 0 in order to compile using GCC version 2.95.3 20010315 (SuSE). This can be faster than using GCC 3.3, especially with respect to linking.

Log

[LDF 2003.08.21.] Now including plfmtar.h. It contains #define and #undef preprocessor commands for conditional compilation. There's a different version of this file in each of the subdirectories used for compiling with a different combination of operating system, compiler, and compiler version.

[LDF 2003.07.25.] Modified the conditional constructions governing compilation slightly.

[LDF 2003.08.14.] Now including getopt.h for the GCC versions under Linux. It's for processing the command line options.

[LDF 2003.08.14.] Now including streambuf.h, if LDF_GCC_2_95 is defined, otherwise ios. This is for stream formatting.

[LDF 2003.08.29.] Removed getopt.h to main.web, because it's only used there.

[LDF 2003.09.03.] Added #define LDF_PUBLIC in order to be able to conditionally include plfmtar.h. The latter is not included in the version for distribution. Instead, the preprocessor variables are defined or undefined here.

[LDF 2003.11.12.] Removed the version control identifiers from the CWEB files for the distribution of 3DLDF 1.1. They're still used in my development versions.

(Version control identifier 5) \equiv
static string rev_id = "$Id:\uer-lcend web, v1.4, 2004/01/12, 21:30:27\uerinst01, Exp\$";

See also sections 13, 53, 63, 76, 93, 164, 241, 250, 307, 635, 659, 696, 982, 1017, 1099, 1141, 1273, 1317, 1331, 1443, 1451, 1470, 1536, and 1543.

This code is used in sections 11, 51, 61, 74, 91, 162, 239, 248, 305, 633, 657, 694, 980, 1015, 1097, 1139, 1271, 1315, 1329, 1441, 1449, 1468, 1534, 1541, and 1558.

6. Configuration file. This section includes config.h, which is generated by configure. This is new in 3DLDF 1.1. The configure script generated by Autoconf tests whether certain library files are present, and defines preprocessor variables in config.h accordingly. These can be used for conditionally compiling code, so that library files are only included if they are really present. However, it will be necessary to add code for handling the case that they aren't present. I haven't done this yet, although I have put in conditional code using these variables in a couple of places. TO DO: Work on this. [LDF 2003.11.12.]

Autoconf does not per default check the version of the compiler that's used, and I'm not sure whether this would really be sensible. 3DLDF already contains conditionally compiled code based on whether the DEC C++ compiler, or the GNU C++ compiler (GCC) version 2.95 or version 3.3 is used. If the DEC compiler is used, the preprocessor variables LDF_GCC_2_95 or LDF_GCC_3_3 must be undefined by hand below. It defines __DECCXX itself. If GCC is used, one of them must be defined, and the other undefined. Per default, LDF_GCC_3_3 is defined and LDF_GCC_2_95 is undefined. This is because GCC 3.3 is, in general, an improvement over GCC 2.95. If I use GCC 2.95, I usually use GCC 2.95 myself, because linking is significantly faster on the computer I use. [LDF 2003.11.12.]

Log

[LDF 2003.11.12.] Added this section.
[LDF 2003.12.17.] config.h is now not included if I'm with the DEC C++ compiler. This is because building with Autoconf, etc., doesn't work on the DEC Alpha machine I'm using.

7. Library files.

[LDF 2003.12.17.] Changed the conditional in which _GNU_SOURCE, LDF_GCC_3_3, and LDF_GCC_2_95 are defined or undefined. Working on compiling with the DEC C++ compiler.
```c
#include LDF_GCC_2_95
#endif
#include <new.h>

#include <sstream>
#include <stdarg.h>
#include <stdexcept>
#ifdef __GNUC__
#include <stdio.h>
#endif
#endif __DECCXX
#include <stdlib.h>
#else HAVE_STDLIB_H
#include <stdlib.h>
#endif

8. streambuf.h is included above, if LDF_GCC_2_95 is defined, instead of ios, which is included in all
other cases. [LDF 2003.08.14.]

#include <vector>

9. Log

[LDF 2003.07.18.] Added "using namespace std". This is needed with GCC 3.3, but not with GCC 2.95
or the DEC C++ compiler.

#include <vector>
using namespace std;

10. Putting loader together.

11. This is what’s compiled. It’s not necessary to include stdlib.h when using GCC 3.3, but I think it’s
safer. [LDF 2004.01.06.] Log

[LDF 2004.01.06.] Added this section. It simplifies the rules for building the executable 3dlldf if loader.c
is compiled.

#ifdef __DECCXX
#include <stdlib.h>
#else HAVE_STDLIB_H
#include <stdlib.h>
#endif
#endif __DECCXX
#include <string>
using namespace std;
(Version control identifier 5)
12. This is what’s written to the loader.h.

[Log]
LDF 2004.01.06. Added this section.

{loader.h 12} ≡
{Include files 6}

13. Global items (pspg1b.web). Typedefs, global variables and constants, and some non-class functions. (pspg1b.web)

[Log]
LDF 2003.11.12. Removed the version control identifiers from the CWB files for the distribution of 3DLDF 1.1.1. They’re still used in my development versions.
LDF 2003.12.10. Put the version control identifiers back into my release versions for 3DLDF 1.1.4. I’ve already put some of them back in, now I’m doing the rest of them. However, the release versions are now in their own RCS repository.

{Version control identifier 5} ≡
static string rcv_id = "$Id: pspg1b.web,v.1.14,2004/01/16,16:30:41,lfinsto1,Exp$";

{Include files 6} ≡
#include "loader.h"
#if _DECCXX
#include <limits>
#else
#endif LDF GCC 3.3
#include <limits>
#else
#endif LDF GCC 2.95
#if HAVE_LIMITS_H
#include <limits.h>
#endif
#endif
#endif
#include <bitset>

15. Type definitions. [LDF 2002.10.15.] Currently, all floating point variables are declared as reals. I’ve defined real in order to make it easy to switch between using floats and doubles simply by changing the value of the #if expression.

I try to avoid using preprocessor commands (see Introduction), but this is one of the cases where there’s no better alternative to using the preprocessor (I don’t consider commenting out the unwanted version preferable to using the preprocessor).

[Log]
LDF 2002.04.10. Added formatting commands.
LDF 2002.04.10. Added declaration of bool_real.
LDF 2002.12.11. Added the macros LDF_REAL_FLOAT and LDF_REAL_DOUBLE. They’re needed below, where MAX_REAL and INVALID_REAL are declared in the GNU/Linux version (using GCC).
LDF 2003.06.03. Added real_short. It’s the return type of Plane::get_distance().
§15  3DLDF-1.15.1

LDF 2003.12.30. Now using \#define instead of @d for the definitions of LDF_REAL_FLOAT and LDF_REAL_DOUBLE. If @d is used, the definitions are only written to ppglb.c. Using \#define, and writing (Type definitions 15) to ppglb.h makes the definitions available to the files that include ppglb.h.

```c
format real float
format real_pair real
format bool_pair real_pair
format real_pair real_short
format Matrix int

(Type definitions 15) \equiv
\#define LDF_REAL_FLOAT 1
\#define LDF_REAL_DOUBLE 0
\#if LDF_REAL_FLOAT
  typedef float real;
\#elif LDF_REAL_DOUBLE
  typedef double real;
\#else  /* Default. LDF 2003.12.17. */
  typedef float real;
\#endif
  typedef real Matrix[4][4];
  typedef pair<real,real> real_pair;
  typedef pair<real,signed short> real_short;
  typedef pair<bool,bool> bool_pair;
  typedef pair<bool,real> bool_real;
```

See also sections 312, 313, 315, and 317.
This code is cited in section 15.
This code is used in sections 51, 52, 633, and 634.

16. Utility classes.

(Utility classes 16) \equiv
```c
struct real_triple {
  real first;
  real second;
  real third;

  real_triple()
  : first(0), second(0), third(0) {}

  real_triple(real a, real b, real c)
  : first(a), second(b), third(c) {}  
};
```
This code is used in sections 51 and 52.

17. Global variables.

18. For compilation. LDF 2003.08.25. GCC 2.95 doesn’t have the numeric_limits template, and GCC 3.3 doesn’t seem to have it either.

Log

LDF 2002.12.11. BUG FIX: Discovered that the way this was before, \texttt{MAX\_REAL = INVALID\_REAL - realLimits.\_epsilon() } caused \texttt{MAX\_REAL} and \texttt{INVALID\_REAL} to be equal! I didn’t notice the problem until
I started to port 3DLDF to GNU/Linux. It also doesn’t work to use \texttt{\texttt{\texttt{MAX\_REAL} = INVALID\_REAL - realLimits.min()}}.

\begin{verbatim}
(Global variables 18) \equiv
    valarray(real) null_coordinates(4);
See also sections 19, 20, 78, 234, 302, and 630.
This code is cited in section 25.
This code is used in sections 51, 91, 239, 305, and 633.

19. \texttt{MAX\_REAL} is the second largest \texttt{real} value. \texttt{MAX\_REAL\_SQRT} is convenient to have for testing when computing distances.

   \texttt{!! KLUDGE: Using the macros FLT\_MAX or DBL\_MAX because the numeric\_limits template doesn’t seem to be available under GNU/Linux using GCC, at least not on the computer I’m using. [LDF 2002.12.11.]}\end{verbatim}

\begin{verbatim}
[\texttt{LDF 2003.12.08.}] Changed the definition of \texttt{MAX\_REAL}. Previously, it was calculated using \texttt{.00000003 * FLT\_MAX}, which was a kludge.
[\texttt{LDF 2003.12.29.}] Changed the way \texttt{MAX\_REAL} and \texttt{MAX\_REAL\_SQRT} are declared. They can no longer be \texttt{const}, because the value of \texttt{MAX\_REAL} is set at the beginning of \texttt{main()} using \texttt{get\_second\_largest < Real > ()}. The value of \texttt{MAX\_REAL\_SQRT} is set after this. Their values should never change after this!
   \texttt{MAX\_REAL\_SQRT} must be initialized here, because it’s used in \textbf{Point::magnitude()}. [LDF 2003.12.29.]
\end{verbatim}

\begin{verbatim}
(Global variables 18) \equiv
#ifdef __DECCXX
    numeric\_limits(real) realLimits;
    extern const real INVALID\_REAL = realLimits.max();
#else
    #if LDF\_REAL\_DOUBLE
        extern const real INVALID\_REAL = DBL\_MAX;
    #else  /* LDF\_REAL\_FLOAT, or not specified. LDF 2003.12.08. */
        extern const real INVALID\_REAL = FLT\_MAX;
    #endif
#endif
real MAX\_REAL = 0;
real MAX\_REAL\_SQRT = 0;

20. [LDF 2003.08.14.] \texttt{VERBOSE\_GLOBAL} is \texttt{false} by default. It is set to \texttt{true} by the command line option "--verbose". If \texttt{VERBOSE\_GLOBAL} is \texttt{true}, the local \texttt{verbose} variables in functions are set to \texttt{true}.

\begin{verbatim}
[\texttt{LDF 2003.08.14.}] Added \texttt{VERBOSE\_GLOBAL} and \texttt{SILENT\_GLOBAL}.
\end{verbatim}

\begin{verbatim}
(Global variables 18) \equiv
    bool VERBOSE\_GLOBAL = false;
    bool SILENT\_GLOBAL = false;
\end{verbatim}
21.  
(Declarations for the header file 21) ≡
  extern bool VERBOSE_GLOBAL;
  extern bool SILENT_GLOBAL;
  extern const bool ldf_realfloat;
  extern const bool ldf_realdouble;
  extern real MAX_REAL;
  extern real MAX_REAL_SQRT;

See also sections 23, 26, 29, 235, 237, 303, 320, 631, 655, and 692.
This code is cited in section 24.
This code is used in sections 52, 240, 306, 634, 658, and 695.

22.  
Log
[LD 2003.08.14.] Added VERSION_3DLDF and COPYRIGHT_3DLDF.

(Declarations for the header file 22) ≡
#if LDF_REAL_FLOAT
  extern const bool ldf_realfloat = 1;
  extern const bool ldf_realdouble = 0;
#else LDF_REAL_DOUBLE
  extern const bool ldf_realfloat = 0;
  extern const bool ldf_realdouble = 1;
#else /* Defaults. LDF 2003.12.17. */
  extern const bool ldf_realfloat = 1;
  extern const bool ldf_realdouble = 0;
#endif
  extern const string VERSION_3DLDF = "1.1.5.1";
  extern const string COPYRIGHT_3DLDF = "Copyright (c) 2003, 2004, by Laurence D. Finston."
  extern const string DISCLAIMER_3DLDF = "This free software, and you are welcome into redis\n  distribution of 3DLDF 1.1.5.1. In order to have received\n  distribution, you should have received\n  distribution under certain conditions, please read again. See the file COPYING.\n  Please send bug reports to the authors at: n\n  Email: ldfinston@ds.gwdg.de\n  URL: http://wwwuser.gwdg.de/ldfinston";

See also sections 28, 159, 236, and 319.
This code is used in sections 51, 162, 239, and 633.

23.  
Log
[LD 2003.11.28.] Changed VERSION_3DLDF from a real to a string. This is necessary, because I now
have versions with three digits separated by periods.

(Declarations for the header file 21) ≡
  extern const string VERSION_3DLDF;
  extern const string COPYRIGHT_3DLDF;
  extern const string DISCLAIMER_3DLDF;
  extern const bool ldf_realfloat;
  extern const bool ldf_realdouble;
24. TO DO: Find out why the library version of `trunc()` can't be found in the version for GCC 2.95 under Linux! [LDF 2002.12.10]
   The problem doesn't exist for GCC 3.3 under Linux. [LDF 2003.08.14]

[Log]

[LDF 2003.08.14.] Put this function declaration in `(Declare utility functions 24)`. Formerly, it was in `(Declarations for the header file 21).

   Changed the conditional from `#ifdef __GNUC__` to `#ifdef LDF_GCC_2_95`, because the library version of `trunc()` is found when compiling with GCC 3.3 under Linux.

`(Declare utility functions 24) ≡
  #ifdef LDF_GCC_2_95
      double trunc(double d);
  #endif

See also section 31.
This code is cited in section 24.
This code is used in section 52.
25. Log

[2003.08.14] Put this function definition into \{Define utility functions 25\}. Formerly, it was in \{Global variables 18\}.

\{Define utility functions 25\} \equiv

```c
#ifdef LDF_GCC_2_95
/* KLUDGE!! [LDF 2002.12.1] \texttt{trunc()} isn't available on the Linux machine gwdg-wb02.gwdg.de! Find out why not! */

double \texttt{trunc(double \texttt{d})}
{
  int \texttt{i};
  \texttt{i = static_cast<int>(d)};
  return \texttt{static_cast<double}(i)};
}
#endif
```

See also section 32.

This code is cited in section 25.

26. For the header file,

\{Declarations for the header file 21\} \equiv

```c
extern \texttt{valarray(real) null_coordinates};
#endif
extern \texttt{numeric_limits(real) realLimits};
#endif
```

27. Global constants. \texttt{INVALID_REAL} is the largest possible \texttt{real} value, where \texttt{real} is either a synonym for \texttt{float} or for \texttt{double}, depending on how it's defined. Values are set to \texttt{INVALID_REAL} or functions return it when something has gone wrong. \texttt{INVALID_REAL} is also used for the \texttt{real} values in \texttt{INVALID_TRANSFORM} and \texttt{INVALID_POINT}. Another possibility would be to use exception handling, but so far I've found it convenient to use \texttt{INVALID_REAL} instead. Since the largest \texttt{float} is so large, and \texttt{epsilon()} for \texttt{floats} is so small, the loss of the largest possible valid value is insignificant. Using exception handling has its advantages, and if it turns out to be useful, I'll put in exception handling code, but using an otherwise valid value to signal exceptional conditions or errors does have the advantage of simplifying the path of execution through the program code. [LDF 2002.10.16] Modified [LDF 2002.10.20].

Log

[2002.09.25.] Added this section. Previously, I declared and initialized my global constants in the header file, This meant that each compilation unit that loaded pspg1b.h had its own version of \texttt{PI}, \texttt{INVALID_REAL}, etc. I didn't know that \texttt{consts} had internal linkage by default and that I could make their linkage external by using \texttt{extern} in the declaration with the initialization, and put a second declaration, also with \texttt{extern}, in the header file. This is what I've done now.

[2003.06.03.] Added \texttt{INVALID_REAL_SHORT}.

28. For compilation,

\{Global constants 22\} \equiv

```c
extern \texttt{const real PI = 4.0 * atan(1.0)};
extern \texttt{const real_pair INVALID_REAL_PAIR(INVALID_REAL, INVALID_REAL)};
extern \texttt{const real_short INVALID_REAL_SHORT(INVALID_REAL, 0)};
```
29. For the header file.
(declarations for the header file 21 ) \[ \begin{align*}
   \text{extern const real } & \pi; \\
   \text{extern const real } & \text{INVALID-REAL;} \\
   \text{extern const real_pair } & \text{INVALID-REAL_PAIR;} \\
   \text{extern const real_short } & \text{INVALID-REAL_SHORT;} \\
\end{align*} \]

30. Utility functions.

31. Solve quadratic equation. [LDF 2002.09.03] TO DO: Maybe add functions for solving cubic and quartic equations, if this is practicable.

---

[LDF 2003.06.1] Changed return type from \texttt{pair(real,real)} to \texttt{real_pair}, which is equivalent.

---

(declare utility functions 24 ) \[ \begin{align*}
   \text{real_pair } & \text{solve_quadratic( real } a, \text{real } b, \text{real } c); \\
\end{align*} \]
32. Define utility functions 25) +≡
real_pair solve_quadratic(real a, real b, real c)
{
    real_pair p;
    try {
        p.first = (-b + sqrt((b * b) - (4 * a * c)))/(2 * a);
    }
    catch(...) {
        p.first = INVALID_REAL;
    }
    try {
        p.second = (-b - sqrt((b * b) - (4 * a * c)))/(2 * a);
    }
    catch(...) {
        p.second = INVALID_REAL;
    }
    return p;
}

33. System information.

--- Log ---

[LDF 2003.12.29.] Added this section.

34. Declare namespace System.

--- Log ---

[LDF 2003.12.29.] Added this section.

(Declare namespace System 34) ≡
namespace System {
    (Declare System functions 36)
}

See also section 65.
This code is used in sections 51, 52, 74, and 75.

35. Endianness.

--- Log ---

[LDF 2003.12.29.] Added this section.
36. **Get endianness.** `get_endianness()` returns the following values:
0 if the processor is little-endian,
1 if the processor is big-endian.
-1 if the endianness cannot be determined.

It is called by `is_little_endian()` and `is_big_endian()`.

[LDF 2003.12.21.]

---

(Declare System functions 36) ≡

```cpp
signed short get_endianness(const bool verbose = false);
```

See also sections 38, 40, 43, 45, 47, 66, 71, and 72.

This code is used in sections 34, 52, 65, and 75.

37.

(Define System functions 37) ≡

```cpp
signed short System::get_endianness(const bool verbose)
{
    union {
        long Long;
        char Char[sizeof(long)];
    } u;
    u.Long = 1;
    if (u.Char[0] == 1) {
        if (verbose) cout << "Processor is little-endian." << endl << endl << flush;
        return 0;
    }
    else if (u.Char[sizeof(long) - 1] == 1) {
        if (verbose) cout << "Processor is big-endian." << endl << endl << flush;
        return 1;
    }
    else {
        cerr << "ERROR in System::get_endianness():\n        " << "Can’t determine endianness. Returning -1" << endl << endl << flush;
        return -1;
    }
}
```

See also sections 39, 41, 44, 46, 48, 67, and 68.

This code is used in sections 51, 74, and 75.

38. **Is big endian.**

---

(Declare System functions 36) ≡

```cpp
bool is_big_endian(const bool verbose = false);
```
39.  
(Define System functions 37) \[\equiv\]  
```cpp
bool System::is_big_endian(const bool verbose)  
{  
    return (get_endianness(verbose) ≡ 1);  
}
```

40. Is little endian.  

[LDF 2003.12.29.] Added this function.

(Declare System functions 36) \[\equiv\]  
```cpp
bool is_little_endian(const bool verbose = false);
```

41.  
(Define System functions 37) \[\equiv\]  
```cpp
bool System::is_little_endian(const bool verbose)  
{  
    return (get_endianness(verbose) ≡ 0);  
}
```

42. Register width.  

[LDF 2003.12.29.] Added this section.

43. Get register width.  

[LDF 2003.12.29.] Added this function.  
[LDF 2004.1.2.] Changed the name of this function from `get_processor_size()` to `get_register_width()`.

(Declare System functions 36) \[\equiv\]  
```cpp
unsigned short get_register_width();
```

44.  
(Define System functions 37) \[\equiv\]  
```cpp
unsigned short System::get_register_width()  
{  
    return (sizeof(void*) * CHAR_BIT);  
}
```

45. Is 32 bit.  

[LDF 2003.12.29.] Added this function.

(Declare System functions 36) \[\equiv\]  
```cpp
bool is_32bit();
```
46. (Define System functions 37) \( \equiv \)
   ```
   bool System::is_32_bit()
   {
     return (get_register_width() \equiv 32);
   }
   ```
47. Is 64 bit.

   [LDF 2003.12.29.] Added this function.

   (Declare System functions 36) \( \equiv \)
   ```
   bool is_64_bit();
   ```
48. (Define System functions 37) \( \equiv \)
   ```
   bool System::is_64_bit()
   {
     return (get_register_width() \equiv 64);
   }
   ```
49. Forward declarations, [LDF 2002.10.16] In the files that are compiled first, some classes refer to
other classes that haven’t been defined yet. Forward declarations make it possible to do this. TO DO: GET
CITATION from Stroustrup.

   [LDF 2002.04.10.] Added the forward declaration of bool real_point. It’s needed because it’s used as
the return value of Point::intersection_point(), which is, of course, declared within the declaration of class
Point. However, bool real_point can only be defined after Point is defined. This forward declaration
solves the problem.

   [LDF 2003.07.16.] Added forward declaration of Ellipse. It’s needed, because I’ve declared Ellipse to be
a friend of Path. Formerly, Circle was a friend of Path, but now it must be Ellipse, because I’ve made
the “segment” functions segment(), half(), and quarter() members of Ellipse instead of Circle.

   (Forward declarations 49) \( \equiv \)
   ```
   struct bool_point;
   struct bool_real_point;
   class Circle;
   class Ellipse;
   struct Focus;
   struct Line;
   class Path;
   class Picture;
   struct Plane;
   class Point;
   ```

   This code is used in sections 51 and 52.
50. Putting pspglb together.
51. This is what’s compiled.

(Include files 6)
(Version control identifier 5)
(Type definitions 15)
(Utility classes 16)
(Global variables 18)
(Global constants 22)
(Define utility functions 25)
(Declare namespace System 34)
(Define System functions 37)
(Forward declarations 49)

52. This is what’s written to the pspglb.h.

(pspglb.h 52) ≡
(Type definitions 15)
(Utility classes 16)
(Declarations for the header file 21)
(Declare namespace System 34)
/* This doesn’t work, apparently because it’s incompatible with the use of sstream. */
#if 0
#elif _DECCXX /* Using the DEC C++ Compiler. */
   const real PI = _CXL_PI;
#endif
#endif
(Declare utility functions 24)
(Forward declarations 49)
(Declare System functions 36)

53. Dynamic allocation for Shapes.

--- Log

[LD 2003.12.29.] Added this template function.
[LD 2004.1.2.] Moved the code in this file from pspglb.web to creatnew.web.

... 

54. Include files.

(Include files 6) ≡
#include "loader.h"
#include "pspglb.h"

55. Dynamic allocation for Shapes.

56. Pointer argument.

(Declare create_new() 56) ≡
   template<class C> C*create_new(const C*arg);
See also section 58.
57. (Define create_new ( ) 57 \( \equiv \))
   \[
   \text{template}\langle\text{class } C\rangle \text{ } C\ast \text{create_new(const } C\ast \text{ary)}
   \]
   \[
   \{ \text{ C * obj = new(C); } \text{ obj.set_on_free_store(); } \text{ if (ary} \neq 0) \text{ *obj = ary;} \text{ return obj; } \}
   \]
   See also section 59.
   This code is used in sections 61 and 62.

58. Reference argument.
   (Declare create_new ( ) 56 \( \equiv \))
   \[
   \text{template}\langle\text{class } C\rangle \text{ } C\ast \text{create_new(const } C\& \text{ ary)};
   \]

59. (Define create_new ( ) 57 \( \equiv \))
   \[
   \text{template}\langle\text{class } C\rangle \text{ } C\ast \text{create_new(const } C\& \text{ ary)}
   \]
   \[
   \{ \text{ C * obj = new(C); } \text{ obj.set_on_free_store(); } \text{ *obj = ary;} \text{ return obj; } \}
   \]

60. Putting createnew together.

61. This is what’s compiled. I don’t really need to compile the definition of create_new ( ) here, because it must be included in all of the files that instantiate it, anyway. However, that may become unnecessary later, in which case it will have to be compiled here. In addition, if there’s something wrong with the definition, it may be helpful to catch the error here. [LDF 2004.1.2]
   \[
   \{ \text{Include files 6} \}
   \{ \text{Version control identifier 5} \}
   \{ \text{Define create_new ( ) 57} \}
62. This is what’s written to the `creatnew.h`. The file `creatnew.h` must be included by all files that define specializations of `create_new()`. [LDF 2003.12.29]

```c
#include "../gsltmplt.web" ;
```

63. Get second-largest real value.

```c
static string res_id = "$Id: gstreamer_0.4_2004/01/12_29:56_gistoi_Exp\$";
```

64. Include files.

```c
#include "loader.h"
#include "pspglb.h"
```

65. Declare namespace System.

```c
[Log]
```

```c
[Declare namespace System 34]
```

```c
namespace System {

(Declare System functions 36)
}
```

66. Get second largest. This function calculates the second-largest real value. It should be called using `float` or `double` as a parameter, e.g., `get_secondLargest < float > (FLT_MAX)` or `get_secondLargest < double > (DEL_MAX)`. FLT_MAX or DEL_MAX must be passed as an argument. On systems with the `numeric_limits` template, `realLimits.max()` could be used instead. [LDF 2003.12.29]

```c
get_secondLargest() determines which unsigned integral type has the same size as the template parameter `Real`. The locally declared type `i_type` is defined to be a synonym for this type using `typedef`. `ip` is a pointer to `i_type`. It is assigned a value by casting a pointer to `MAX_VAL` to the type of `ip`, i.e., a pointer to `i_type`. Then, `1` is subtracted from `*ip`, and the `ip` is cast back to a pointer to `Real`. This is the second largest `Real` value. [LDF 2003.12.29]

This algorithm works on all of the machines I’ve tested. It doesn’t matter whether they are big or little-endian, whether they have 32 or 64-bit processors. If the exponent of a floating point type is stored in its lowest order byte or bytes, then this will fail. I haven’t run into this problem yet, though. The commented-out code in (Loop for testing bits 70) may help in finding the second-largest `Real` value in this case. [LDF 2003.12.29]

```c
[Log]
```

```c
(Declare System functions 36)
```

```c
template<class Real> Real get_secondLargest(Real MAX_VAL, bool verbose = false);
float get_secondLargest(float, bool);
```

67. (Define System functions 37)
68.

(Define System functions 37) \(+\)

\texttt{template\langle class\ Real\ \rangle \ Real\ System:: get\_second\_largest(RealMAX\_VAL, bool verbose)\ }
\{
    const unsigned short USHORT\_SIZE = sizeof(unsigned short);
    const unsigned short UINT\_SIZE = sizeof(unsigned int);
    const unsigned short ULONG\_SIZE = sizeof(unsigned long);
    const unsigned short ULONG\_LONG\_SIZE = sizeof(unsigned long long);
    const unsigned short Real\_SIZE = sizeof(Real);
    const unsigned short FLT\_SIZE = sizeof(float);
    const unsigned short DBL\_SIZE = sizeof(double);
    const unsigned short LONG\_DBL\_SIZE = sizeof(long double);
    const bool RealEQ\_USHORT = (Real\_SIZE == USHORT\_SIZE);
    const bool RealEQ\_UINT = (Real\_SIZE == UINT\_SIZE);
    const bool RealEQ\_ULONG = (Real\_SIZE == ULONG\_SIZE);
    const bool RealEQ\_ULONG\_LONG = (Real\_SIZE == ULONG\_LONG\_SIZE);
    if (verbose) {
        cout << "USHORT\_SIZE=\u" << USHORT\_SIZE << endl << flush;
        cout << "UINT\_SIZE=\u" << UINT\_SIZE << endl << flush;
        cout << "ULONG\_SIZE=\u" << ULONG\_SIZE << endl << flush;
        cout << "ULONG\_LONG\_SIZE=\u" << ULONG\_LONG\_SIZE << endl << flush;
        cout << "FLT\_SIZE=\u" << FLT\_SIZE << endl << flush;
        cout << "DBL\_SIZE=\u" << DBL\_SIZE << endl << flush;
        cout << "LONG\_DBL\_SIZE=\u" << LONG\_DBL\_SIZE << endl << flush;
        cout << "Real\_SIZE=\u" << Real\_SIZE << endl << flush;
    }
    Real* rp;
    if (RealEQ\_USHORT) {
        if (verbose) cout << "Real\_EQ\_USHORT\n";
        typedef unsigned short i\_type;
        (Calculate second-largest Real 69)
    }
    else if (RealEQ\_UINT) {
        if (verbose) cout << "Real\_EQ\_UINT\n";
        typedef unsigned int i\_type;
        (Calculate second-largest Real 69)
    }
    else if (RealEQ\_ULONG) {
        if (verbose) cout << "Real\_EQ\_ULONG\n";
        typedef unsigned long i\_type;
        (Calculate second-largest Real 69)
    }
    else if (RealEQ\_ULONG\_LONG) {
        if (verbose) cout << "Real\_EQ\_ULONG\_LONG\n";
        typedef unsigned long long i\_type;
        (Calculate second-largest Real 69)
    }
    else {
        // Handle other cases
    }
}
```c
cerr << "ERROR! In main():" << endl << " Apparently Real doesn’t have the same size."
   << " as any unsigned integral type."
   << " There must be some mistake." << endl
   << "Exiting with return value 1." << endl << flush;
   return -1;
 }
if (verbose) {
   cout << " MAX VAL_u = " << MAX_VAL_u << endl
   << " MAX VAL < endl < " << *rp_u = u << endl
   << " (MAX VAL _ u * rp _ u = u) < (MAX VAL + * rp _ u = u) < (MAX VAL > * rp _ u) < endl
   << " MAX VAL < rp _ u * rp _ u = u" < (MAX VAL > * rp) < endl < flush;
 }
if (MAX_VAL == *rp) {
   cerr << "ERROR! In System::get_second_largest<Real>():" << endl
   << " MAX VAL = u * rp. Exiting with return value 1" << endl < endl << flush;
   exit(1);
 }
else if (MAX.VAL < *rp) {
   cerr << "ERROR! In System::get_second_largest<Real>():" << endl
   << " MAX VAL < rp. Exiting with return value 1" << endl < endl < flush;
   exit(1);
 }
return *rp;
}

69. Calculate second-largest Real.
(Calculate second-largest Real 69) ≡
 { i_type *ip = reinterpret_cast<i_type*>(&MAX_VAL);
   if (verbose) cout << " *ip_u = u" << *ip << endl << flush;
   i_type bit_pattern_i_type;
   i_type result;
   bit_pattern_i_type = 1;
   bitset<sizeof(i_type) * CHAR_BIT> b;
   b = *ip;
   if (verbose) cout << "b_u (MAX VAL) _ u = u" << b << endl < flush;
   b = bit_pattern_i_type;
   if (verbose) cout << "b_u (bit_pattern_i_type) _ u = u" << b << endl < flush;
   result = bit_pattern_i_type & *ip;
   if (verbose) cout << "result_u = u" << result << endl < flush;
   b = result;
   if (verbose) cout << "b_u (result) _ u = u" << b << endl < flush;
   rp = reinterpret_cast<Real*>(&result);
   if (verbose) cout << " *rp_u = u" << *rp << endl < flush;
 } }
This code is used in section 68.
```
70. Loop for testing bits.

(Loop for testing bits 70) 

```c
#if 0
    int counter; for (int i = 0; i < (sizeof (Real) * CHAR_BIT); ++i) {
        if (verbose) cout << "i<<u" << i << endl << flush; /* This has only been needed on the DEC Alpha, so far. */
        if (verbose) cout << "mantissa is 1, and all of the other bits (in particular, all the bits of the exponent) are 0. In this case, *rp */
        if (verbose) cout << "*rp */ is not a number (NAN). The GNU compiler copes with this, */
        if (verbose) cout << "the DEC compiler signals a floating point error and dumps */
        if (verbose) cout << "core (I believe). I wasn’t able to catch the error with */
        if (verbose) cout << "try and catch. */
        if (verbose) cout << "START HERE. Change (8 + 1) to (FLT_EXP + 1), except */
        if (verbose) cout << "FLT_EXP */
        if (verbose) cout << "isn’t the right name. Find it, and put here. This */
        if (verbose) cout << "assumes the exponent is at the left, which may not be */
        if (verbose) cout << "true. Skipping this bit pattern is necessary on the DEC */
        if (verbose) cout << "ALPHA, because the float is not a number. */
        if (verbose) cout << "It must be 23 for float, and 52 for double. */
    }

    if (i == (sizeof (Real) * CHAR_BIT) - (12)) {
        if (verbose) cout << "This produces NaN. Continuing. 
        if (verbose) cout << "n" << flush;
        continue;
    }

    bit_pattern_i_type = 1;
    bit_pattern_i_type <<= i;
    if (verbose) cout << "bit_pattern_i_type" << endl << flush;
    b = bit_pattern_i_type;
    if (verbose) cout << "b" << endl << flush;
    result = b;
    if (verbose) cout << "result" << endl << flush;
    rp = reinterpret_cast (Real *) (result);
    if (verbose) cout << "*rp" << endl << flush;
    if (*rp < 0) {
        if (verbose) cout << "*rp" << endl << flush;
        continue;
    }
    else if (*rp >= MAX_VAL) {
        if (verbose) cout << "*rp" << MAX_VAL << endl << flush;
        continue;
    }
    else if (secondLargest_real >= *rp) {
        if (verbose) cout << "secondLargest_real" "nContinuing." << endl << flush;
        continue;
    }
    else if (*rp > secondLargest_real && *rp < MAX_VAL) {
        if (verbose) cout << "secondLargest_real" "nContinuing." << endl << flush;
        secondLargest_real = *rp;
        counter = i;
    }
    else {
        if (verbose) cout << "Some other condition. Continuing." "n";
        continue;
    }

```

71. Template function instantiations.

[Declare System functions 36] +≡
float get_second_largest(float MAX_VAL, bool verbose);

72. (Declare System functions 36) +≡
double get_second_largest(double MAX_VAL, bool verbose);

73. Putting galtmplt together.

74. This is what’s compiled. I don’t really need to compile the definition of get_second_largest() here, because it must be included in all of the files that instantiate it, anyway. However, that may become unnecessary later, in which case it will have to be compiled here. In addition, if there’s something wrong with the definition, it may be helpful to catch the error here. [LDF 2004.1.2.]

(Include files 6)
(Version control identifier 5)
(Declare namespace System 34)
(Define System functions 37)
75. This is what’s written to the galsplit.h. The file galsplit.h must be included by all files that define specializations of get_second_largest(). [LDF 2003.12.29.]

\begin{verbatim}
galsplit.h 75) ==
(Declare namespace System 34)
(Declare System functions 36)
(Define System functions 37)
\end{verbatim}

76. I/O (io.web).

\begin{verbatim}
[Log] [LDF 2003.11.12.] Removed the version control identifiers from the CWEB files for the distribution of 3DLDF 1.1. They’re still used in my development versions.
[Log] [LDF 2003.12.10.] Put the version control identifiers back into my release versions for 3DLDF 1.1.4. I’ve already put some of them back in, now I’m doing the rest of them. However, the release versions are now in their own RCS repository.

{Version control identifier 5) ==
static string res_id = "$Id: io.web,v 1.4 2004/01/12 21:30:03 ulfinsto1 Exp $";
\end{verbatim}

77. Include files.

\begin{verbatim}
(Include files 6) ==
#include "loader.h"
#include "pspglb.h"
#include <time.h>
\end{verbatim}

78. Global variables. [LDF 2002.10.16.] in_stream is an input stream attached to a file with user code for input. Currently, it is used, but it fulfills no useful function, because I haven’t defined an input routine yet. out_stream is an output stream attached to the file of METAPOST code that 3DLDF currently produces as its output. tex_stream is an output stream attached to a file of TeX code. The user can write TeX code to this file and load it into persp.tex or use it for some other purpose. 3DLDF makes no use of it itself.

\begin{verbatim}
[Log] [LDF 2002.08.30.] Added tex_stream so that I can include TeX code in my user code. Code written by 3DLDF to tex_stream will be loaded by persp.tex, or whatever TeX file includes the PostScript file generated by METAPOST from the output of 3DLDF. User code is currently in main.web. In production versions user code will be in user.web.
[Log] [LDF 2003.07.16.] Added fig_num.

format ifstream int
format ofstream int
\end{verbatim}

(GLOBAL VARIABLES 18) ==
ifstream in_stream;
ofstream out_stream;
ofstream tex_stream;
unsigned short fig_num;
79. **extern declarations for the global variables.**

\[
\begin{align*}
\text{extern variable declarations 79) } & \equiv \\
\text{extern ifstream } & \text{ in}\_\text{stream}; \\
\text{extern ofstream } & \text{ out}\_\text{stream}; \\
\text{extern ofstream } & \text{ tex}\_\text{stream}; \\
\text{extern unsigned short } & \text{ fig}\_\text{num};
\end{align*}
\]

This code is used in section 92.

80. **I/O functions.**

81. **Initialize I/O.**

<table>
<thead>
<tr>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>[LDF 2003.08.29.] Changed, so that in_stream isn’t opened.</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{Declare I/O functions 81) } & \equiv \\
\text{void initialize_io(string in}\_\text{stream}_\text{name}, \text{string out}\_\text{stream}_\text{name}, \text{string tex}\_\text{stream}_\text{name}, \text{char } & *\text{program}_\text{name});
\end{align*}
\]

See also sections 84, 86, and 88.

This code is used in section 92.

82. **Define I/O functions 82) \equiv**

\[
\begin{align*}
\text{void initialize_io(string in}\_\text{stream}_\text{name}, \text{string out}\_\text{stream}_\text{name}, \text{string tex}\_\text{stream}_\text{name}, \text{char } & *\text{program}_\text{name})\{ \text{time}_t & tt; \\
& \text{tm } *tt; \\
& tt = \text{time}(0); \\
& tt = \text{localtime}(\&tt); \\
& \text{string datestamp(asctime(tt));} \\
& \text{datestamp._erase(datestamp._size()} - 1); /* \text{Remove terminal line-feed. */}
\end{align*}
\]

See also sections 83, 85, 87, and 89.

This code is used in section 91.
83. Open `out_stream` and `tex_stream`. `in_stream` is currently not opened. [LDF 2003.08.29.]

(Define I/O functions 82) \(\equiv\)

```c
#if 0
  in_stream.open(in_stream_name.c_str());
#endif
out_stream.open(out_stream_name.c_str());
#endif
out_stream.set(ios_base::fixed, ios_base::floatfield);
#else
#endif
out_stream.set(ios::fixed, ios::floatfield);
#endif
```

```
ext_stream.open(tex_stream_name.c_str()); /* Write datestamp to out_stream. */
out_stream << "\texttt{This is an out_stream name} \" << out_stream_name << "\" << endl << "\texttt{Generated on} \" << datestamp << "\" << program_name << "\" << endl << "\texttt{Date from} \" << program_name << "\" << endl << "\texttt{Datestamp} \" << program_name << "\"
```

84. Write footers. [LDF 2002.10.16.] Footers can be written to `output_stream` and `tex_stream`. I use them for Local Variables lists for Emacs. Other users may not want this, which is why this code is commented out here.

(Declare I/O functions 81) \(\equiv\)

```c
void write_footers();
```
85.  
{Define I/O functions s2} +
  void write_footers()
  {
    #if 0
      out_stream << endl << "% Variables:" << endl << "% mode:Metafont" 
          << endl << "% eval:(if metamap-nil (load "metamap\n"))" 
          << endl << "% eval:(use-local-map metapost-mode-map)" 
          << endl << "% eval:(local-set-key [f9]'mp-file)" 
          << endl << "% run-cweb-on-file:"persp.mp" 
          << endl << "% run-cweb-on-file:"main.web" 
          << endl << "% run-tex-on-file:"persp.tex" 
          << endl << "% run-dvips-on-file:"persp.ps"
    #endif
    return;
  }

86.  Begin figure.

<table>
<thead>
<tr>
<th>Log</th>
</tr>
</thead>
</table>

[LDF 2003.07.16] Added silent argument, and a message printed conditionally to stdout, saying which figure is being started. This should help in finding where errors occur.


[LDF 2003.08.17] Made silent non-const. Setting it to true, if SILENT_GLOBAL is true.

(Declare I/O functions s1) +
  void beginfig(unsigned short i, bool silent = false);
87. Define I/O functions 82) +≡
   void beginfig(unsigned short i, bool silent)
   {
     if (SILENT_GLOBAL) silent = true;
     fig_num = i;
     out_stream << "beginfig(" << fig_num << ");\n";
     if (!silent) cout << "Beginning\figure\" << fig_num << "." << endl << flush;
     return;
   }

88. End figure. The unsigned short argument is “syntactic sugar”. It’s ignored by endfig(), but may
be convenient for a user for keeping track of what figure is being ended.

---

Log

[LDF 2003.07.16.] Added silent argument, and a message printed conditionally to stdout, saying which
figure is being ended. This should help in finding where errors occur.
[LDF 2003.07.16.] Made non-inline.
[LDF 2003.08.17.] Made silent non-const. Setting it to true, if SILENT_GLOBAL is true.

(Declare I/O functions 81) +≡
   void endfig(unsigned short i = 0, bool silent = false);

89. Define I/O functions 82) +≡
   void endfig(unsigned short i, bool silent)
   {
     if (SILENT_GLOBAL) silent = true;
     out_stream << "endfig" << ";\n";
     if (!silent) cout << "Ending\figure\" << fig_num << "." << endl << endl << flush;
     return;
   }

90. Putting I/O together.

91. This is what’s compiled.
   (Include files 6)
   (Version control identifier 5)
   (Global variables 18)
   (Define I/O functions 82)
92. This is what’s written to io.h,

\{ io.h 92 \} ≡
\{ extern variable declarations 79 \}
\{ Declare I/O functions 81 \}

93. Color (colors.web).

--- Log

[LDF 2003.11.12] Removed the version control identifiers from the CWEB files for the distribution of 3DLDF 1.1. They’re still used in my development versions.
[LDF 2003.12.10] Put the version control identifiers back into my release versions for 3DLDF 1.1.4. I’ve already put some of them back in, now I’m doing the rest of them. However, the release versions are now in their own RCS repository.

\{ Version control identifier 5 \} ≡
static string res_id = "$Id: colors.web,v1.7.2004/01/12_21:27:38_1finsto1_Exp\$";

94. Include files.

\{ Include files 6 \} ≡
\#include "loader.h"
\#include "pspglb.h"
\#include "creatnew.h"
\#include "io.h"

95. Color class definition, [LDF 2002.09.25] "" Remember to change the constructors, setting functions, and assignment operator if I add or change anything here!!

--- Log

[LDF 2002.10.06] Added on_free_store.

\{ Define class Color 95 \} ≡
class Color {
    string name;
    bool use_name;
    bool on_free_store;
    real red_part;
    real green_part;
    real blue_part;
    public: \{ Declare Color functions 97 \}
};

This code is used in sections 162 and 163.

96. Constructors and setting functions,

97. Default constructor, [LDF 2002.10.06] Added code to definition. Previously, it was empty.

\{ Declare Color functions 97 \} ≡
Color();

See also sections 99, 102, 104, 107, 109, 114, 116, 118, 121, 123, 125, 127, 129, 131, 133, 135, 138, 141, 142, 143, 144, 146, 149, and 151.

This code is used in section 95.
98. (Define Color functions 98) ≡
   Color::Color()
   {
      red_part = green_part = blue_part = 0.0;
      name = "";
      use_name = false;
      on_free_store = false;
   }

See also sections 100, 103, 105, 108, 110, 115, 119, 122, 124, 126, 128, 130, 132, 134, 136, 139, 145, 150, and 152.
This code is used in section 162.

99. Copy constructor.  !! Remember to add or change code here if I add or change anything in the class definition!!

   [LDF 2002.09.25.]  Added this function.

(Declare Color functions 97) +≡
   Color(const Color &c, const string n = "", const bool u = true);

100. (Define Color functions 98) +≡
   Color::Color(const Color &c, const string n, const bool u)
   {
      name = n;
      red_part = c.get_red_part();
      green_part = c.get_green_part();
      blue_part = c.get_blue_part();
      if (n != "" && u) {
         use_name = true;
      } else use_name = false;
      on_free_store = false;  /* LDF 2002.10.06. Added. */
      return;
   }

101. Name and unsigned short arguments.

102. Constructor.

(Declare Color functions 97) +≡
   Color(const string n, const unsigned short r, const unsigned short g, const unsigned short b, const bool u = true);
103.  
(Define Color functions 98) +≡

    Color : Color(const string n, const unsigned short r, const unsigned short g, const unsigned short b, const bool u)
    : name(n) {
        name = n;
        if (n != "" && u == true) {
            use_name = true;
        }
        else use_name = false;
        on_free_store = false;  /* LDF 2002.10.06. Added. */
        red_part = r / 255.0;
        green_part = g / 255.0;
        blue_part = b / 255.0;
    }

104.  Setting function.  
(Declare Color functions 97) +≡

    void set(const string n, const unsigned short r, const unsigned short g, const unsigned short b, const bool u = false);

105.  
(Define Color functions 98) +≡

    void Color::set(const string n, const unsigned short r, const unsigned short g, const unsigned short b, const bool u)
    {
        name = n;
        if (n != "" && u == true) {
            use_name = true;
        }
        else use_name = false;
        red_part = r / 255.0;
        green_part = g / 255.0;
        blue_part = b / 255.0;
    }

106.  Three real arguments.  [LDF 2002.10.06.] Added the following constructor and setting function. They are for unnamed Colors. The DEC compiler can't distinguish between real and unsigned short arguments, so the overloaded functions must differ in another way. In this case, these versions have no name argument. I believe that users are most likely to declare Colors using real arguments when they plan to modify them, in which case the output() function should write the red, green and blue values to ostream rather than name. If it turns out to be necessary, more constructors can be added or the existing ones can be changed.

107.  Constructor.  
(Declare Color functions 97) +≡

    Color(const real r, const real g, const real b);
108. Define Color functions +⇒

```cpp
Color(const real r, const real g, const real b) {
    name = "";
    use_name = false;
    on_free_store = false;
    if (r < 0) {
        cerr << "WARNING! In Color:Color() with three real arguments:\n" <<
            "Red part argument r<0. Setting red part to 0.\n"
        ;
        red_part = 0;
    } else if (r > 1) {
        cerr << "WARNING! In Color:Color() with three real arguments:\n" <<
            "Red part argument r>1. Setting red part to 1.\n"
        ;
        red_part = 1;
    } else red_part = r;
    if (g < 0) {
        cerr << "WARNING! In Color:Color() with three real arguments:\n" <<
            "Green part argument g<0. Setting green part to 0.\n"
        ;
        green_part = 0;
    } else if (g > 1) {
        cerr << "WARNING! In Color:Color() with three real arguments:\n" <<
            "Green part argument g>1. Setting green part to 1.\n"
        ;
        green_part = 1;
    } else green_part = g;
    if (b < 0) {
        cerr << "WARNING! In Color:Color() with three real arguments:\n" <<
            "Blue part argument b<0. Setting blue part to 0.\n"
        ;
        blue_part = 0;
    } else if (b > 1) {
        cerr << "WARNING! In Color:Color() with three real arguments:\n" <<
            "Blue part argument b>1. Setting blue part to 1.\n"
        ;
        blue_part = 1;
    } else blue_part = b;
    return;
}
```

109. Setting function.

Declare Color functions +⇒

```cpp
void set(const real r, const real g, const real b);
```
§110.  
(Define Color functions 98) +≡

```cpp
void Color::set(const real r, const real g, const real b)
{
    name = "";
    use_name = false;
    on_free_store = false;
    if (r < 0) {
        cerr << "WARNING! \( \text{\textbackslash Color(\textbackslash three_\textbackslash real_\textbackslash arguments)} \): \n" <<
            "Red_\textbackslash part_\textbackslash argument_\textless_0.0_\textless real_\textgreater_0.0.0_\textless 0.0_\textless 0.0\";
        red_part = 0;
    } else if (r > 1) {
        cerr << "WARNING! \( \text{\textbackslash Color(\textbackslash three_\textbackslash real_\textbackslash arguments)} \): \n" <<
            "Red_\textbackslash part_\textbackslash argument_\textgreater_1.0_\textless real_\textgreater_0.0_\textless 0.0\";
        red_part = 1;
    } else red_part = r;
    if (g < 0) {
        cerr << "WARNING! \( \text{\textbackslash Color(\textbackslash three_\textbackslash real_\textbackslash arguments)} \): \n" <<
            "Green_\textbackslash part_\textbackslash argument_\textless_0.0_\textless real_\textgreater_0.0_\textless 0.0\";
        green_part = 0;
    } else if (g > 1) {
        cerr << "WARNING! \( \text{\textbackslash Color(\textbackslash three_\textbackslash real_\textbackslash arguments)} \): \n" <<
            "Green_\textbackslash part_\textbackslash argument_\textgreater_1.0_\textless real_\textgreater_0.0_\textless 0.0\";
        green_part = 1;
    } else green_part = g;
    if (b < 0) {
        cerr << "WARNING! \( \text{\textbackslash Color(\textbackslash three_\textbackslash real_\textbackslash arguments)} \): \n" <<
            "Blue_\textbackslash part_\textbackslash argument_\textless_0.0_\textless real_\textgreater_0.0_\textless 0.0\";
        blue_part = 0;
    } else if (b > 1) {
        cerr << "WARNING! \( \text{\textbackslash Color(\textbackslash three_\textbackslash real_\textbackslash arguments)} \): \n" <<
            "Blue_\textbackslash part_\textbackslash argument_\textgreater_1.0_\textless real_\textgreater_0.0_\textless 0.0\";
        blue_part = 1;
    } else blue_part = b;
}
```

111.  Pseudo-constructor for dynamic allocation,

```
Log

[LD 2003.12.30.] Replaced Color::create_new_color() with specializations of template<class C> C*create_new().
```

112.  Pointer argument.

(Declare non-member template functions for Color 112) +≡

```cpp
Color *create_new(const Color &c);
```
113. **Reference argument.**

<table>
<thead>
<tr>
<th>Log</th>
</tr>
</thead>
</table>

[LDF 2004.1.2.] Added this declaration.

\[
\text{\{Declare non-member template functions for Color 112\}} + \equiv \\
\text{Color} *create\_new(\text{const Color} &c); \\
\]

114. **Assignment.** [LDF 2002.09.24.] Added this operator function.

\[
\text{\{Declare Color functions 97\}} + \equiv \\
\text{void operator=}=(\text{const Color} &c); \\
\]

115.

\[
\text{\{Define Color functions 98\}} + \equiv \\
\text{void Color::operator=}=(\text{const Color} &c) \\
\text{\{} \\
\text{name} = "\text{un}"; \\
\text{use\_name} = \text{false}; \\
\text{red\_part} = \text{c\_red\_part}; \\
\text{green\_part} = \text{c\_green\_part}; \\
\text{blue\_part} = \text{c\_blue\_part}; \\
\text{\}} \\
\]

116. **Equality.** [LDF 2002.09.25.] Changed so that only red\_part, green\_part and blue\_part are compared. This way, Colors that differ only in name and/or use\_name are considered to be equal. [LDF 2002.09.24.] Added this operator function.

\[
\text{\{Declare Color functions 97\}} + \equiv \\
\text{bool operator}\equiv(\text{const Color} &c) \text{\ const}; \\
\]

117.

\[
\text{\{Define Color functions 98\}} + \equiv \\
\text{bool Color::operator}\equiv(\text{const Color} &c) \text{\ const} \\
\text{\{} \\
\text{return} ((\text{red\_part} \equiv \text{c\_red\_part}) \land (\text{green\_part} \equiv \text{c\_green\_part}) \land (\text{blue\_part} \equiv \text{c\_blue\_part})); \\
\text{\}} \\
\]

118. **Inequality.** [LDF 2002.09.24.] Added this operator function.

\[
\text{\{Declare Color functions 97\}} + \equiv \\
\text{bool operator}\neq(\text{const Color} &c) \text{\ const}; \\
\]
119.  
(Define Color functions 98) \[\begin{align*}
\text{bool Color::operator\neq(const Color &c) const} \\
\{ \\
\text{ return \neg(*this \equiv c);} \\
\}
\end{align*}\]

120.  
Modifying.

121.  
Set on free store.

[Log]  
[LDF 2003.12.30.] Added this function.

(Declare Color functions 97) \[\begin{align*}
\text{bool set_on_free_store(bool b = true);} \\
\end{align*}\]

122.  
(Define Color functions 98) \[\begin{align*}
\text{bool Color::set_on_free_store(bool b)} \\
\{ \\
\text{ on_free_store = b;} \\
\text{ return b;} \\
\}
\end{align*}\]

123.  
Set name.

(Declare Color functions 97) \[\begin{align*}
\text{void set_name(const string s);} \\
\end{align*}\]

124.  
(Define Color functions 98) \[\begin{align*}
\text{void Color::set_name(const string s)} \\
\{ \\
\text{ name = s;} \\
\}
\end{align*}\]

125.  
Set use name.

(Declare Color functions 97) \[\begin{align*}
\text{void set_use_name(const bool b);} \\
\end{align*}\]

126.  
(Define Color functions 98) \[\begin{align*}
\text{void Color::set_use_name(const bool b)} \\
\{ \\
\text{ use_name = b;} \\
\}
\end{align*}\]

127.  
Modify.

(Declare Color functions 97) \[\begin{align*}
\text{void modify(const real r, const real g = 0, const real b = 0);} \\
\end{align*}\]
128.  
(Define Color functions 98) +⇒

```c
void Color::modify(const real r, const real g, const real b)
{
    red_part += r;
    green_part += g;
    blue_part += b;
    if (red_part > 1) {
        cerr << "WARNING! In Color::modify():\n" << "red_part is greater than 1: \n"
        endl << "Setting red_part to 1. \n"
        red_part = 1;
    } else if (red_part < 0) {
        cerr << "WARNING! In Color::modify():\n" << "red_part is less than 0: \n"
        endl << "Setting red_part to 0. \n"
        red_part = 0;
    }
    if (green_part > 1) {
        cerr << "WARNING! In Color::modify():\n" << "green_part is greater than 1: \n"
        green_part = 1;
    } else if (green_part < 0) {
        cerr << "WARNING! In Color::modify():\n" << "green_part is less than 0: \n"
        green_part = 0;
    }
    if (blue_part > 1) {
        cerr << "WARNING! In Color::modify():\n" << "blue_part is greater than 1: \n"
        blue_part = 1;
    } else if (blue_part < 0) {
        cerr << "WARNING! In Color::modify():\n" << "blue_part is less than 0: \n"
        blue_part = 0;
    }
    return;
}
```

129.  Set red_part.
(Declare Color functions 97) +⇒

```c
void set_red_part(const real r);
```
130. 
(Define Color functions 98) \(\equiv\)

```c
void Color::set_red_part(const real r) {
    if (r > 1) {
        cerr \ll "WARNING! In Color::set_red_part():\n        \ll "r is greater than 1:\n        \ll "r \ll endl \ll "Setting red_part to 1.:\n        \ll \n\r;    
        red_part = 1;
    } else if (r < 0) {
        cerr \ll "WARNING! In Color::set_red_part():\n        \ll "r is less than 0:\n        \ll "r \ll endl \ll "Setting red_part to 0.:\n        \ll \n\r;    
        red_part = 0;
    } else red_part = r;
    return;
}
```

131. Set green part.
(Declare Color functions 97) \(\equiv\)

```c
void set_green_part(const real g);
```

132. 
(Define Color functions 98) \(\equiv\)

```c
void Color::set_green_part(const real g) {
    if (g > 1) {
        cerr \ll "WARNING! In Color::set_green_part():\n        \ll "g is greater than 1:\n        \ll "g \ll endl \ll "Setting green_part to 1.:\n        \ll \n\r;    
        green_part = 1;
    } else if (g < 0) {
        cerr \ll "WARNING! In Color::set_green_part():\n        \ll "g is less than 0:\n        \ll "g \ll endl \ll "Setting green_part to 0.:\n        \ll \n\r;    
        green_part = 0;
    } else green_part = g;
    return;
}
```

133. Set blue part.
(Declare Color functions 97) \(\equiv\)

```c
void set_blue_part(const real b);
```
134. (Define Color functions 98) +≡

```cpp
void Color::set_blue_part(const real b)
{
  if (b > 1) {
    cerr << "WARNING! In Color::set_blue_part(): \n" << "b is greater than 1: \n" << endl << "Setting blue_part to 1. \n\n";
    blue_part = 1;
  } else if (b < 0) {
    cerr << "WARNING! In Color::set_blue_part(): \n" << "b is less than 0: \n" << endl << "Setting blue_part to 0. \n\n";
    blue_part = 0;
  } else blue_part = b;
  return;
}
```

135. Show.

(Declare Color functions 97) +≡

```cpp
void show(string text = "") const;
```

136. (Define Color functions 98) +≡

```cpp
void Color::show(string text) const
{
  if (text == "") text = "Color:";
  cout << text << endl;
  cout << "name" = "" << get_name() << endl;
  cout << "use_name" = "" << get_use_name() << endl;
  cout << "red_part" = "" << get_red_part() << endl;
  cout << "green_part" = "" << get_green_part() << endl;
  cout << "blue_part" = "" << get_blue_part() << endl << endl << endl;
  return;
}
```

137. Returning elements and information.

138. Is on free store.

---

[LDFO 2004.01.06] Made non-inline.

(Declare Color functions 97) +≡

```cpp
bool is_on_free_store() const;
```
139.  
<Define Color functions 98> +≡
  bool Color::is_on_free_store() const
  {
    return on_free_store;
  }

140. Get Color parts.  [LDF 2002.09.24.] These functions always return a real; the argument decimal can’t make them return an unsigned short.

141. Get red part.
<Declare Color functions 97> +≡
  inline real get_red_part(bool decimal = false) const
  {
    if (decimal) return trunc((red_part * 255) + .5);
    else return red_part;
  }

142. Get green part.
<Declare Color functions 97> +≡
  inline real get_green_part(bool decimal = false) const
  {
    if (decimal) return trunc((green_part * 255) + .5);
    else return green_part;
  }

143. Get blue part.
<Declare Color functions 97> +≡
  inline real get_blue_part(bool decimal = false) const
  {
    if (decimal) return trunc((blue_part * 255) + .5);
    else return blue_part;
  }

144. Get use name.
<Declare Color functions 97> +≡
  bool get_use_name() const;
145. 
(Define Color functions 98) \(\equiv\)

\[
\text{bool Color\_get\_use\_name() const}
\]

\[
\text{return use\_name;}
\]

146. Get name.

(Declare Color functions 97) \(\equiv\)

\[
\text{inline string get\_name() const}
\]

\[
\text{return name;}
\]

147. Output operator.

(Declare non-member non-template functions for Color 147) \(\equiv\)

\[
\text{ostream & operator \ll (ostream & o, const Color &c)};
\]

This code is used in section 163.

148. 
(Define non-member non-template functions for Color 148) \(\equiv\)

\[
\text{ostream & operator \ll (ostream & o, const Color &c)}
\]

\[
\text{\{ if (c.get\_use\_name() \equiv true) \{
\text{o \ll c.get\_name();
\text{\}}}
\text{else \{
\text{o \ll "" \ll c.get\_red\_part() \ll ",\" \ll c.get\_green\_part() \ll ",\" \ll c.get\_blue\_part() \ll "";}
\text{\}}
\text{return o;}
\}\]

This code is used in section 162.

149. Define Colors in METAPOST.

(Declare Color functions 97) \(\equiv\)

\[
\text{void define\_color\_mp() const;}
\]
§150. Define Color functions 98) ≡
    void Color::define_color_mp() const
    {
        if (!out_stream.is_open()) {
            cerr << "ERROR! In Color::define_color_mp();\n" <<
                "out_stream.is_closed! Returning.\n" << flush;
            return;
        }
        if (name == "") {
            cerr << "ERROR! In Color::define_color_mp();\n" <<
                "name is empty. Not doing anything and returning.\n" << flush;
            return;
        }
        out_stream << "color" << name << ";" << "," << name << "," 
            << get_red_part() << "," <<
            get_green_part() << "," <<
            get_blue_part() << ";\n" << flush;
        return;
    }

151. Initialize Colors. [LDF 2002.09.25.] This function presupposes the existence of namespace Colors.
(Declare Color functions 97) ≡
    static void initialize_colors();
152.  
(Define Color functions 98) ==

```cpp
void Color::initialize_colors()
{
    using namespace Colors;
    if (!out_stream.is_open()) { 
        cerr << "ERROR! Color::initialize_colors():\n" << "out_stream is closed! Returning \n" << flush;
        return;
    }
    out_stream << "\n\nColor definitions.\n\n";
#if 0  /* [LDF 2002.09.25.] These colors are already defined in METAPOST and their definitions are 
not likely to change. However, if they do, I can comment these function calls back in. */
    red.define_color_map();
    green.define_color_map();
    blue.define_color_map();
    black.define_color_map();
    white.define_color_map();
    background.define_color_map();
#endif
    yellow.define_color_map();
    cyan.define_color_map();
    magenta.define_color_map();
    orange.define_color_map();
    violet.define_color_map();
    purple.define_color_map();
    yellowgreen.define_color_map();
    greenyellow.define_color_map();
    blueviolet.define_color_map();
    gray.define_color_map();
    lightgray.define_color_map();
    violetred.define_color_map();
    default_background.define_color_map();
    /* [LDF 2002.09.25.] Currently, this function does nothing if I'm using all of the colors. */
    out_stream << "\n\nEnd of Color definitions.\n\n";
    return;
}
```

153. Namespace Colors. Here I can put either (Major Colors 156) or (All Colors 0) into (Declare namespace Colors 153), and comment out the other, depending on what I want. This prevents too much unneeded code from being processed. (All Colors 0) is very long, so I neither want to compile it, write the extern declarations from it to colors.h, nor print out the code when I run cweave, unless I really want to use it. [LDF 2002.09.25.]

(Declare namespace Colors 153) ==

```cpp
(Major Colors 156)  /* (All Colors 0) */
```

This code is cited in section 153.
This code is used in section 162.
154. [LDF 2002.09.25.] Here I can put either \texttt{(extern Major Colors 157)} or \texttt{(extern All Colors 0)} into \texttt{(extern namespace Colors declaration 154)}
\texttt{(extern namespace Colors declaration 154) \equiv}
\texttt{(extern Major Colors 157)}
This code is cited in sections 154 and 158.
This code is used in section 163.

155. Major Colors. The colors "red", "green", "blue", "black", and "white" are already defined in METAPOST, however, we need them here in order to access the \texttt{Color} functions for them.
!! [LDF 2002.09.24.] If this definition isn’t explicitly written to the header file, as it is below, this causes real problems!! It took me awhile to find out that this was the cause.

156. Internal (with initialization). [LDF 2002.09.25] !! If I add \texttt{Colors} here, remember to add them in the “External” section below, and in the definition of \texttt{Color::initialize_colors()} below.

\begin{verbatim}
[LDF 2002.10.26.] Added \texttt{help\_color}.

(Major Colors 156) \equiv
namespace Colors
  extern const Color red("red", 255, 0, 0, true);
  extern const Color green("green", 0, 255, 0, true);
  extern const Color blue("blue", 0, 255, true);  /* Primaries, subtractive. */
  extern const Color cyan("cyan", 0, 255, 255, true);
  extern const Color yellow("yellow", 255, 255, 0, true);
  extern const Color magenta("magenta", 255, 0, 255, true);  /* LDF 2002.09.27. */ The convention that I use is that colors like “orange\_red” are reds and colors like “red\_orange” are oranges. */
  /* Red. */
  extern const Color orange\_red("orange\_red", 255, 69, 0);
  extern const Color violet\_red("violet\_red", 208, 32, 144);  /* Pink. */
  extern const Color pink("pink", 255, 192, 203);  /* Blue. */
  extern const Color green\_yellow("green\_yellow", 173, 255, 47);  /* Yellow. */
  extern const Color orange("orange", 255, 165, 0, true);  /* Orange. */
  extern const Color violet("violet", 238, 130, 238, true);
  extern const Color purple("purple", 160, 32, 240, true);
  extern const Color \texttt{blue\_violet}("blue\_violet", 138, 43, 226);  /* Green. */
  extern const Color yellow\_green("yellow\_green", 154, 205, 50);  /* “Unbunt” Colors (blacks, whites, and grays). */
  extern const Color black("black", 0, 0, true);
  extern const Color white("white", 255, 255, 255, true);
  extern const Color gray("gray", 192, 192, 192);
  extern const Color light\_gray("light\_gray", 211, 211, 211);  /* Defaults. [LDF 2002.09.27.] Note that \texttt{default\_color}, \texttt{help\_color} and \texttt{background\_color} are pointers and that \texttt{default\_background} is a plain \texttt{Color}. It can be used to access the original background color (currently white), if the user points the \texttt{background\_color} at some other \texttt{Color}. */
  extern const Color default\_background("default\_background", 255, 255, 255, true);
  extern const Color \texttt{*default\_color} = \&black;
  extern const Color \texttt{*background\_color} = \&default\_background;
  extern const Color \texttt{*help\_color} = \&green;  /* LDF 2002.10.26. Added. */
  */ [LDF 2002.09.25.] !! TO DO: \texttt{default\_background} is a convenience, in case I change “background” in the METAPOST code. Check METAPOST documentation!! I believe it has something similar. */
\end{verbatim}
157. External.

\texttt{\{extern Major Colors 157\} \equiv}
\texttt{
namespace Colors \{ /* [LDF 2002.09.27.] The ordering should be as above for the internal declarations. */ /* Primaries, additive. */
    extern const Color red;
    extern const Color green;
    extern const Color blue; /* Primaries, subtractive. */
    extern const Color cyan;
    extern const Color yellow;
    extern const Color magenta; /* Red. */
    extern const Color orange_red;
    extern const Color violet_red; /* Pink. */
    extern const Color pink; /* Blue. */ /* Yellow. */
    extern const Color green_yellow; /* Orange. */
    extern const Color orange; /* Violet. */
    extern const Color violet;
    extern const Color purple;
    extern const Color blue_violet; /* Green. */
    extern const Color yellow_green; /* "Unbunt" Colors (black, white, and grays). */
    extern const Color black;
    extern const Color white;
    extern const Color gray;
    extern const Color light_gray; /* Defaults. */
    extern const Color default_background;
    extern Color *\texttt{default\_color};
    extern Color *\texttt{background\_color};
\}
}

This code is cited in section 153 and 158.
This code is used in section 153.

158. All Colors. \texttt{\{extern Major Colors 156\}}

\texttt{namespace Colors \{ /* [LDF 2002.09.27.] The ordering should be as above for the internal declarations. *//* Primaries, additive. */
    extern const Color red;
    extern const Color green;
    extern const Color blue; /* Primaries, subtractive. */
    extern const Color cyan;
    extern const Color yellow;
    extern const Color magenta; /* Red. */
    extern const Color orange_red;
    extern const Color violet_red; /* Pink. */
    extern const Color pink; /* Blue. */ /* Yellow. */
    extern const Color green_yellow; /* Orange. */
    extern const Color orange; /* Violet. */
    extern const Color violet;
    extern const Color purple;
    extern const Color blue_violet; /* Green. */
    extern const Color yellow_green; /* "Unbunt" Colors (black, white, and grays). */
    extern const Color black;
    extern const Color white;
    extern const Color gray;
    extern const Color light_gray; /* Defaults. */
    extern const Color default_background;
    extern Color *\texttt{default\_color};
    extern Color *\texttt{background\_color};
\}
\}

This code is cited in section 154.
This code is used in section 154.

159. Global constants.

\texttt{namespace Colors \{
    extern const vector<const Color *> \texttt{default\_color\_vector}(1, \texttt{default\_color});
\}
extern const vector<const Color *> help_color_vector(1, help_color);
    /* LDF 2002.10.26. Added. */
extern const vector<const Color *> background_color_vector(1, background_color);
}

160.
{extern global constant declarations 160} ≡
namespace Colors {
    extern const vector<const Color *> default_color_vector;
    extern const vector<const Color *> help_color_vector;  /* LDF 2002.10.26. Added. */
    extern const vector<const Color *> background_color_vector;
}

This code is used in section 163.

161. Putting Color together.

162. This is compiled.

{Include files 6}
{Version control identifier 5};
{Define class Color 95}
{Declare namespace Colors 153}
{Define Color functions 98}
{Declare non-member template functions for Color 112}
{Define non-member non-template functions for Color 148}
{Global constants 22}
163. This is written to colors.h.
(colors.h 163) ≡
(Define class Color 95)
(extern namespace Colors declaration 154)
(Declare non-member template functions for Color 112)
(Declare non-member non-template functions for Color 147)
(extern global constant declarations 160)

164. Transformations (transform.web).

Log
[LDF 2003.11.12.] Removed the version control identifiers from the CWEB files for the distribution of
3DLDF 1.1. They're still used in my development versions.
[LDF 2003.12.10.] Put the version control identifiers back into my release versions for 3DLDF 1.1.4. I've
already put some of them back in, now I'm doing the rest of them. However, the release versions are now in
their own RCS repository.

(Version control identifier 5) ≡
static string rsid = "$Id: transform.web,v1.1.5,04/01/12,21:33:44,1finside1,Exp$";

165. Include files.

(Include files 6) ≡
#include "loader.h"
#include "pspglb.h"
#include "io.h"
#include "colors.h"

166. Transform class definition. The Transform class has a 4 × 4 transformation matrix as its only
data member and a number of member functions. Points, Pictures, and Focuses contain Transforms as
data members.

!! Remember to add items to operator=() if I add them to the class definition here.

Log
[LDF 2003.07.04.] Removed friend declaration for Focus. I've added set_element() and get_element()
which are used in the Focus functions, so the latter need no longer be a friend of Transform.

format Transform int
(Define class Transform 166) ≡
class Transform {
friend class Point;
Matrix matrix; /* When I've got things working, I can try to optimize use of storage by not
storing the parts of the matrix that I don't need. This is a little complicated, because the row or
column which isn't needed differs between the affine and perspective transformations. */
public: (Declare Transform functions 168)
};
This code is used in sections 239 and 240.

167. Constructors.

168. Default constructor. (No arguments). Initializes a new transformation matrix as the identity
matrix
§168  3DLDF-1.1.5.1

(Declare Transform functions 168) \(\equiv\)
\[\text{Transform}();\]
See also sections 170, 172, 174, 176, 178, 180, 182, 185, 187, 190, 192, 195, 197, 200, 202, 203, 205, 211, 212, 213, 216, 218, 221, 223, 226, and 292.

This code is used in section 166.

169.
(Define Transform functions 169) \(\equiv\)
\[\text{Transform}();\text{Transform}();\]
\{\n    \text{reset}();\n\}

This code is used in sections 239, 633, and 980.

170. Constructor with one real argument. All elements of matrix are set to the real argument \(r\).
(Declare Transform functions 168) \(\equiv\)
\[\text{Transform}(\text{real } r);\]

171.
(Define Transform functions 169) \(\equiv\)
\[\text{Transform}();\text{Transform}(\text{real } r)\]
\{\n    \text{for} (\text{int } i = 0; i < 4; i++)\n    \text{for} (\text{int } j = 0; j < 4; j++)\n    \text{matrix}[i][j] = r;\n\}

172. Constructor with 16 real arguments. [LDF 2002.09.06] Added this constructor. This constructor makes it possible to specify all of the elements of matrix.
(Declare Transform functions 168) \(\equiv\)
\[\text{Transform}(\text{real } r_0, \text{real } r_1, \text{real } r_2, \text{real } r_3, \text{real } r_4, \text{real } r_5, \text{real } r_6, \text{real } r_7, \text{real } r_8, \text{real } r_9, \text{real } r_{10}, \text{real } r_{11}, \text{real } r_{12}, \text{real } r_{13}, \text{real } r_{14}, \text{real } r_{15}, \text{real } r_{16}, \text{real } r_{17}, \text{real } r_{18}, \text{real } r_{19}, \text{real } r_{20}, \text{real } r_{21}, \text{real } r_{22}, \text{real } r_{23}, \text{real } r_{24}, \text{real } r_{25}, \text{real } r_{26}, \text{real } r_{27});\]
173. (Define Transform functions 169) +≡

\[
\text{Transform: } \text{Transform}(\text{real } r_0_0, \text{real } r_0_1, \text{real } r_0_2, \text{real } r_0_3, \text{real } r_1_0, \text{real } r_1_1, \text{real } r_1_2, \text{real } r_1_3, \text{real } r_2_0, \text{real } r_2_1, \text{real } r_2_2, \text{real } r_2_3, \text{real } r_3_0, \text{real } r_3_1, \text{real } r_3_2, \text{real } r_3_3)
\]

\[
\{
\begin{align*}
\text{matrix}[0][0] &= r_0_0; \\
\text{matrix}[0][1] &= r_0_1; \\
\text{matrix}[0][2] &= r_0_2; \\
\text{matrix}[0][3] &= r_0_3; \\
\text{matrix}[1][0] &= r_1_0; \\
\text{matrix}[1][1] &= r_1_1; \\
\text{matrix}[1][2] &= r_1_2; \\
\text{matrix}[1][3] &= r_1_3; \\
\text{matrix}[2][0] &= r_2_0; \\
\text{matrix}[2][1] &= r_2_1; \\
\text{matrix}[2][2] &= r_2_2; \\
\text{matrix}[2][3] &= r_2_3; \\
\text{matrix}[3][0] &= r_3_0; \\
\text{matrix}[3][1] &= r_3_1; \\
\text{matrix}[3][2] &= r_3_2; \\
\text{matrix}[3][3] &= r_3_3;
\end{align*}
\}
\]

174. Assignment. Sets matrix to be identical to the matrix of another Transform. !! Remember to add items here if I add them to the class definition.

(Declare Transform functions 168) +≡

\[
\text{Transform operator}=\text{(const Transform } \& t); \\
\]

175. (Define Transform functions 169) +≡

\[
\text{Transform Transform :: operator=}(\text{const Transform } \& t)
\]

\[
\{
\begin{align*}
&\text{for (int } i = 0; \ i < 4; \ i++) \\
&\quad \text{for (int } j = 0; \ j < 4; \ j++) \text{ matrix}[i][j] = t.\text{matrix}[i][j]; \\
&\text{return } t;
\end{align*}
\}
\]

176. Reset to identity matrix.

(Declare Transform functions 168) +≡

\[
\text{void reset();}
\]
177. (Define Transform functions 169) \(\equiv\) 
\[
\text{void Transform::reset()}
\{
    \text{for (int } i = 0; i < 4; i++) \quad \text{/* Rows */}
    \text{for (int } j = 0; j < 4; j++) \quad \text{/* Columns */}
    \{ \text{matrix}[i][j] = (i \equiv j) ? 1 : 0; \}
\}
\]

178. Setting values.

\[\text{LDF 2003.07.04.} \quad \text{Added this function.}\]

(Declare Transform functions 168) \(\equiv\) 
\[
\text{void set_element(const unsigned short row, const unsigned short col, real r);} \]

179. (Define Transform functions 169) \(\equiv\) 
\[
\text{void Transform::set_element(const unsigned short row, const unsigned short col, real r)}
\{ \text{matrix[row][col] = (fabs(r) < epsilon()) ? 0 : r; return;} \}
\]

180. Clean. \(\text{clean()}\) changes elements in matrix whose absolute values are \(\text{< epsilon()}\) to 0.
(Declare Transform functions 168) \(\equiv\) 
\[
\text{void clean();} \]
181. 
(Define Transform functions 169) \(\equiv\)

```c
void Transform::clean()
{
    real eps = epsilon();
    for (int i = 0; i < 4; i++)  /* Rows. */
        for (int j = 0; j < 4; j++)  /* Columns. */
            if (fabs(matrix[i][j]) < eps) matrix[i][j] = 0;
}
```

182. Epsilon. Minimum magnitude of values stored in matrix. [LDF 2002.10.16.] The value returned by `epsilon()` has to be fairly large because of the poor precision resulting from the use of `floats` and the Standard Library versions of the trigonometric functions. There is currently no equality operator for `Transform`, but the precision of the `transforms` affects that of Points. TO DO: I hope to be able to solve the problem by finding routines for calculating the trigonometric functions more accurately (and faster) by using integers and bitwise shifts. If this doesn’t work out, I could try redefining `real` as `double` (which I don’t want to do), or I could try to use `double` explicitly when using the trigonometric functions. In the latter case, I would have to truncate the `doubles` to `floats` eventually, so I don’t know if this would have any benefit.

[LDF 2004.1.2.] Now returning different values, depending on whether `real` is `float` or `double`. TO DO: Try to find out what values would be best. It will be necessary to check how good the value for `double` is.

[LDF 2004.1.2.] Made `epsilon()` `static` and non-inline.

(Declare Transform functions 168) \(\equiv\)

```c
static real epsilon();
```

183.

(Define Transform functions 169) \(\equiv\)

```c
real Transform::epsilon()
{
    #if LDF_REAL_DOUBLE
        return .000000001;
    #else
        return .00001;
    #endif
}
```

184. Test for identity matrix. [LDF 2002.11.16.] TO DO: I should check the elements on the main diagonal for whether they differ from 1 by an amount \(<\text{epsilon()}\). If so, they should be set to 1.

185. Non-const version.

(Declare Transform functions 168) \(\equiv\)

```c
bool is_identity();
```
186. Define Transform functions 169 +≡
   bool Transform::is_identity()
   {
      clean();
      for (int i = 0; i < 4; i++)
         for (int j = 0; j < 4; j++)
            if ((i == j) && matrix[i][i] != 1) || ((i != j) && matrix[i][j] != 0)) return false;
      return true;
   }

187. const version.
   Declare Transform functions 168 +≡
   bool is_identity() const;

188. Define Transform functions 169 +≡
   bool Transform::is_identity() const
   {
      Transform t;
      t = *this;
      t.clean();
      for (int i = 0; i < 4; i++)
         for (int j = 0; j < 4; j++)
            if ((i == j) && matrix[i][i] != 1) || ((i != j) && matrix[i][j] != 0)) return false;
      return true;
   }

189. Querying.

190. Get element.

   Log

   [LDF 2003.07.04.] Added this function.

   Define Transform functions 168 +≡
   real get_element(const unsigned short row, const unsigned short col) const;

191. Define Transform functions 169 +≡
   real Transform::get_element(const unsigned short row, const unsigned short col) const
   {
      return matrix[row][col];
   }

192. Show.
   Declare Transform functions 168 +≡
   void show(string text ="") const;
193. Define Transform functions 169 \( + \equiv \)

```cpp
void Transform::show (string text) const
{
    if (text == "") text = "Transform:";
    cout << text << ":n";
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {* LDF 2002.02.07. Can’t use left here, and I can’t include
            ios. This causes an error with a reference to something in the C++ manual. Must write to K.
            Heuer and ask if he can fix it. */
            cout << setw(7) << setprecision(3) << matrix[i][j] << ":n";
        }
    }
    cout << endl;
}
```

194. Affine transformations. [LDF 2002.10.16.] The functions for the affine transformations all return a Transform representing the transformation, not *this. This makes it possible to chain expressions using operator*=( ), e.g.,

```cpp
Point p0;  // Start with some initial point
Point p1 (1, 2, 3);  // Set p1 to some other point
Point p2 (3, 4, 5);
Transform t;  // t is our initial transformation
    t.rotate(90, 90, 90);  // Apply a 90° rotation to each point
    p0 *= p1 *= p2 *= t;  // Catenate all transformations
```

`p0`, `p1`, and `p2` are all rotated 90° around the x, y, and z-axes.

195. Scale. [LDF 2002.10.15.] Bug fix: If the absolute value of an argument is < epsilon ( ), the argument is now set to 0 instead of 1. Setting it to 1 causes no scaling to take place, which is not the effect of multiplying the corresponding coordinate by a number of very small magnitude.

[LDF 2003.03.25.] Bug fix: Fixed conditional that tests whether all the arguments are 1.

(Declare Transform functions 168) \( + \equiv \)

```cpp
Transform scale (real x, real y = 1, real z = 1);
```
§196. 
(Define Transform functions 169) +≡

Transform Transform : $scale(\text{real } x, \text{real } y, \text{real } z)$
{
  Transform t;
  if $(x \equiv 1 \land y \equiv 1 \land z \equiv 1)$ {
    cerr << "WARNING! In Transform::scale():\n    " << endl << endl << flush;
    return t;
  }
  real eps = epsilon();
  t.matrix[0][0] = (fabs(x) > eps) ? x : 0;
  t.matrix[1][1] = (fabs(y) > eps) ? y : 0;
  t.matrix[2][2] = (fabs(z) > eps) ? z : 0;
  *this += t;
  clean();
  return t;
}

197. Shear. [LDF 2002.10.15.] Replaced the dummy definition of this function with a proper one.
(Declare Transform functions 168) +≡

Transform shear(\text{real } xy, \text{real } zz = 0, \text{real } yx = 0, \text{real } yz = 0, \text{real } zx = 0, \text{real } yz = 0);
198.
(Define Transform functions 169) +≡

Transform Transform :: shear(real xy, real xx, real yz, real xx, real yy)
{
    Transform t;
    real eps = epsilon();
    if (fabs(xy) < eps) xy = 0;
    if (fabs(xx) < eps) xx = 0;
    if (fabs(yz) < eps) yz = 0;
    if (fabs(xx) < eps) yy = 0;
    if (xy <= 0 && xx <= 0 && yz <= 0 && xx <= 0 && yz <= 0) {
        cerr << "WARNING! Invalid Transform::shear()\n" << endl << endl << flush;
        return t;
    }
    t.matrix[0][0] = xy;
    t.matrix[0][1] = xx;
    t.matrix[0][2] = yz;
    t.matrix[1][0] = xx;
    t.matrix[1][1] = yz;
    t.matrix[1][2] = xy;
    (*this) *= t;
    clean();
    return t;
}

199. Shift. (Translation.)

200. real arguments.
(Declare Transform functions 168) +≡

Transform shift(real x, real y, real z = 0);

201.
(Define Transform functions 169) +≡

Transform Transform :: shift(real x, real y, real z)
{
    real eps = epsilon();
    Transform t;
    if (x <= 0 && y <= 0 && z <= 0) return t;
    t.matrix[3][0] = (fabs(x) > eps) ? x : 0;
    t.matrix[3][1] = (fabs(y) > eps) ? y : 0;
    t.matrix[3][2] = (fabs(z) > eps) ? z : 0;
    t.clean();
    (*this) *= t;
    clean();
    return t;
}
202. **Point argument.** [LDF 2002.04.24.] Added this function. It must be defined in points.web, because `Point` is an incomplete type here.

(Declare `Transform` functions 168) +≡

```
Transform shift(const Point &p);
```

203. **Shift with multiplication.** [LDF 2002.08.22.] Added this function. It takes real arguments and multiplies the appropriate elements of `matrix` by them.

(Declare `Transform` functions 168) +≡

```
Transform shift_times(real x, real y = 1, real z = 1);
```

204.

(Define `Transform` functions 169) +≡

```
Transform Transform::shift_times(real x, real y, real z)
{
    bool DEBUG = false;  /* true */
    if (DEBUG) {
        show("Before multiplication");
    }
    matrix[3][0] *= x;
    matrix[3][1] *= y;
    matrix[3][2] *= z;
    if (DEBUG) {
        show("After multiplication");
    }
    return *this;
}
```

205. **Rotation around the main axes.** `rotate()` will perform rotation about the x, y and z-axes in that order if called with multiple, non-zero arguments. Rotation only about the y and/or z-axis requires one or two dummy 0 arguments so that `rotate()` “knows” about which axis (or axes) to rotate.

(Declare `Transform` functions 168) +≡

```
Transform rotate(real x, real y = 0, real z = 0);
```
206.

Define Transform functions 169 \{\#
    Transform Transform::rotate(real x, real y, real z)\{ bool DEBUG = false; /* true */
    Transform t_all;
    real eps = epsilon();
    if (x \equiv 0 \land y \equiv 0 \land z \equiv 0) {
        if (DEBUG) cerr \ll "In rotate(real, real, real)\n" \ll "0.0 rotation about all axes. Returning identity matrix.\n" \ll flush;
        return t_all;
    }
    Transform t_x;
    Transform t_y;
    Transform t_z;
    real ssin;
    real ccos;
    real temp1;
    real temp2;
    int i;

207. Rotation around the x-axis.

Define Transform functions 169 \{\#
    if (x \neq 0) { /* ! Reversed direction of rotation because I didn’t like the way it was. */
        x *= -PI/180.0; /* Convert to radians. */
        ssin = sin(x);
        ccos = cos(x);
        for (i = 0; i < 4; i++) {
            temp1 = (t_x.matrix[i][1] * ccos) - (t_x.matrix[i][2] * ssin);
            temp2 = (t_x.matrix[i][1] * ssin) + (t_x.matrix[i][2] * ccos);
            t_x.matrix[i][2] = (fabs(temp2) \geq eps) ? temp2 : 0;
            t_x.matrix[i][1] = (fabs(temp1) \geq eps) ? temp1 : 0;
        }
    } /* if */

208. Rotation around the y-axis.

Define Transform functions 169 \{\#
    if (y \neq 0) { /* ! Reversed direction of rotation because I didn’t like the way it was. */
        y *= -PI/180.0;
        ssin = sin(y);
        ccos = cos(y);
        for (i = 0; i < 4; i++) {
            temp1 = (t_y.matrix[i][0] * ccos) + (t_y.matrix[i][2] * ssin);
            temp2 = (-t_y.matrix[i][0] * ssin) + (t_y.matrix[i][2] * ccos);
            t_y.matrix[i][2] = (fabs(temp2) \geq eps) ? temp2 : 0;
            t_y.matrix[i][0] = (fabs(temp1) \geq eps) ? temp1 : 0;
        }
    } /* if */
209. Rotation around the z-axis.

\[
\begin{align*}
\text{(Define Transform functions 169) +} & \\
\text{if } (z \neq 0) \{ & \\
\quad z = P1/180.0; & \\
\quad \sin = \sin(z); & \\
\quad \cos = \cos(z); & \\
\quad \text{for } (\text{int } i = 0; i < 4; i++) \{ & \\
\quad \quad \text{temp1} = (t_x \cdot \text{matrix}[i][0] \ast \cos) - (t_x \cdot \text{matrix}[i][1] \ast \sin); & \\
\quad \quad \text{temp2} = (t_x \cdot \text{matrix}[i][0] \ast \sin) + (t_x \cdot \text{matrix}[i][1] \ast \cos); & \\
\quad \quad t_x \cdot \text{matrix}[i][1] = (\text{fabs}(\text{temp2}) \geq \text{eps}) \? \text{temp2} : 0; & \\
\quad \quad t_x \cdot \text{matrix}[i][0] = (\text{fabs}(\text{temp1}) \geq \text{eps}) \? \text{temp1} : 0; & \\
\quad \} & \quad \text{/* if */} & \\
\quad t_x = t_x; & \\
\quad t_y = t_y; & \\
\quad t_z = t_z; & \\
\quad t_x \cdot \text{clean}(); & \\
\quad (* \text{this}) = t_x; & \\
\quad \text{clean}(); & \\
\quad \text{return } t_x; & \quad \text{/* End of rotate(). */} & \\
\end{align*}
\]

210. Rotation around an arbitrary axis.

211. Point arguments. Defined in points.web because Point is an incomplete type here.

Log

[LDF 2002.4.7.] Added default value for \text{angle} \equiv 180.

[LDF 2003.06.02.] Changed name of this function from \text{rotate\_around()} to \text{rotate()}. This function now overloads \text{rotate()} with three real arguments.

\[
\begin{align*}
\text{(Declare Transform functions 168) +} & \\
\text{Transform rotate(Point p0, Point p1, const real angle = 180);} & \\
\end{align*}
\]
212. Path argument. [LDF 2002.05.03.] Defined in paths.web because Path is an incomplete type here.

[Log]

[LDF 2002.05.03.] Added this function.
[LDF 2003.06.02.] Changed name of this function from rotate_around() to rotate(). This function now overloads rotate() with three real arguments.

(Declare Transform functions 168) +≡
Transform rotate(const Path &p, const real angle = 180);

213. Alignment with an axis. Defined in points.web, because it uses Points, which haven’t been defined yet.

(Declare Transform functions 168) +≡
Transform align_with_axis(Point p0, Point p1, char axis = ’z’);  // Default is the z-axis. */

214. Matrix multiplication.

215. With assignment.

216. real argument. [LDF 2002.11.19.] This function multiplies each element of Matrix by the real argument r and returns r. This makes it possible to chain invocations of this function. Not currently used anywhere, but it may turn out to be useful for something.

[Log]

[LDF 2002.08.22.] Added this function.
[LDF 2002.11.19.] Changed return value from *this to r.

(Declare Transform functions 168) +≡
real operator*=(real r);

217.

(Declare Transform functions 169) +≡
real Transform::operator*=(real r)
{
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++) matrix[i][j] *= r;
    clean();
    return r;
}


[Log]

[LDF 2002.11.06.] If t is the identity Transform, it is returned right away. If *this is, it is set to t using operator=(t). BUG FIX: Now t is always returned, instead of *this. This makes it possible to chain expressions using this function.

(Declare Transform functions 168) +≡
Transform operator*=(const Transform &t);
219.
\[\text{Define Transform functions 169} \implies\]
\text{Transform Transform::operator\*= (const Transform& t)}
\{  
  \text{bool DEBUG = false; } \quad /\ast \text{ true } */\n  \text{if (DEBUG) cout \ll "Entering\_Transform::operator\*=().n" \ll flush; }\n  \text{if (t.is\_identity()) } \{\n    \text{if (DEBUG) cout \ll "t is the identity transformation.\n      
      Returning t. n" \ll flush; }\n    \text{return t; }\n  }\n  \text{else if (is\_identity()) } \{\n    \text{if (DEBUG) cout \ll "*this, is the identity \_transformation.\n      
      Setting *this to t and returning t. n" \ll flush; }\n    \text{return (*this = t); }\n  \}
  \text{Matrix temp\_matrix = \{0,0,0,0\}, \{0,0,0,0\}, \{0,0,0,0\}, \{0,0,0,0\}; }\n  \text{for (int i = 0; i < 4; i++ ) }\n  \text{for (int k = 0; k < 4; k++) }\n    \text{for (int j = 0; j < 4; j++) temp\_matrix[i][k] += matrix[i][j] \* t.matrix[j][k]; }\n  \text{for (int i = 0; i < 4; i++ ) }\n    \text{for (int j = 0; j < 4; j++) }\n      \text{if (DEBUG) cout \ll "Entering\_Transform::operator\*=().n" \ll flush; }\n      \text{return t; }\n\}

220. Plain multiplication.

221. real argument. [LDF 2002.08.22.] Added this function. Not currently used anywhere, but it may turn out to be useful for something.
\[\text{Define Transform functions 168} \implies\]
\text{Transform operator\*= (const real r) const; }

222.
\[\text{Define Transform functions 169} \implies\]
\text{Transform Transform::operator\*= (const real r) const}
\{  
  \text{Transform t = *this;}\n  \text{t *= r;}\n  \text{t.clean(); }\n  \text{return t; }\n\}

223. Transform argument.
\[\text{Define Transform functions 168} \implies\]
\text{Transform operator\*= (const Transform t) const; }
224.

\[
\text{Define Transform functions 169}\quad +\equiv \\
\text{Transform Transform \_\_ operator\_\_ (const Transform t) const} \\
\text{\{ \\
\text{Transform a = \_\_this; } \\
\text{a *= t; } \\
\text{a.clean(); } \\
\text{return a; } \\
\text{\}}}
\]

225. Matrix inversion.

226. \begin{description}
\item[const version (no assignment).] It would be easy to generate the inverses of the transformations that I call explicitly using \textit{rotate()}, \textit{shift()}, etc., as I go along. However, it is not possible to do this for the ones produced using \textit{operator*()} and \textit{operator+=()}. So, since a matrix inversion routine is needed anyway, I don’t bother to generate the inverses as I go along.
\item[TO DO:] Get format for references! \textit{inverse()} uses the Gaß-Jordan algorithm with column pivot search. I’ve taken the algorithm from Stoer, Josef. \textit{Numerische Mathematik} 1\textsuperscript{1} and adapted it to C++.
\end{description}

\[\text{Log}\]

[LDF 2002.12.01.] !! Changed \textit{hi} from \textit{real} to \textit{int} because of a warning, when I tried to compile under GNU/Linux. I think \textit{hi} can be an \textit{int}, but test to be sure!

\[
\text{\langle Define Transform functions 168\rangle +\equiv} \\
\text{Transform \_\_ inverse() const;}
\]

227.
(Define Transform functions 169) +≡

Transform Transform::inverse() const { bool DEBUG = false; /* true */
    if (DEBUG) cout << "Entering Transform::inverse()." << endl << flush;
    int i;
    int j;
    int k;
    int row;
    const int n = 4;
    real max;
    real hr;
    int hi; /* [LDF 2002.12.01] See above in “Log”. */
    real hv[n];
    Transform t;
    if (DEBUG) {
        cout << "matrix_1=" << n << ":n;"
    }
    for (i = 0; i < 4; i++) {
        for (j = 0; j < 4; j++) {
            t.matrix[i][j] = matrix[i][j];
            if (DEBUG) {
                cout << matrix[i][j] << ":n;"
            }
        }
    }
    if (DEBUG) {
        cout << ":n;"
        t.show("t");
        cout << "Enter<RETURN>to continue." << ":n\n\n" << flush;
        getchar();
    }
    int p[n];
    for (int j = 0; j < n; j++) p[j] = j;
    for (j = 0; j < n; j++) {

228. Pivot search.

\begin{Verbatim}
\{ Define Transform functions 169 \} + \equiv  
\begin{verbatim}
max = fabs(t.matrix[j][j]);
row = j;
for (i = j + 1; i < n; i++) {
  if (fabs(t.matrix[i][j]) > max) {
    max = fabs(t.matrix[i][j]);
    row = i;
  }
} /* inner for */
if (DEBUG) {
  cout << "max_i==_j" << max << endl << flush;
  cout << "row_i==_j" << row << endl << flush;
}
if (max == 0) {
  cerr << "ERROR! In Transform::inverse() : n" <<
       "Matrix is singular. Returning INVALID_TRANSFORM.\n" << flush;
  return INVALID_TRANSFORM;
}
\end{verbatim}
\end{Verbatim}

229. Row exchange.

\begin{Verbatim}
\{ Define Transform functions 169 \} + \equiv  
\begin{verbatim}
if (row > j) {
  for (k = 0; k < n; k++) {
    hr = t.matrix[j][k];
    t.matrix[j][k] = t.matrix[row][k];
    t.matrix[row][k] = hr;
  } /* for */
  hi = p[j];
  p[j] = p[row];
  p[row] = hi;
} /* if */
if (DEBUG) cout << "Finished row exchange.\n" << flush;
\end{verbatim}
\end{Verbatim}


\begin{Verbatim}
\{ Define Transform functions 169 \} + \equiv  
\begin{verbatim}
if (DEBUG) cout << "t.matrix[" << j << "][" << j << "] == \n" << t.matrix[j][j] << endl << flush;
hr = 1/t.matrix[j][j];
for (i = 0; i < n; i++) t.matrix[i][j] = hr * t.matrix[i][j];
\end{verbatim}
\end{Verbatim}

\begin{Verbatim}
\begin{verbatim}
t.matrix[j][j] = hr;
for (k = 0; k < n; k++)
  if (k != j) {
    for (i = 0; i < n; i++)
      if (i != j) t.matrix[i][k] -= t.matrix[i][j] * t.matrix[j][k];
    t.matrix[j][k] *= -hr;
  }
} /* outer for */
if (DEBUG) cout << "Finished Transformation.\n" << flush;
\end{verbatim}
\end{Verbatim}
231. Column exchange.

\[ \text{Define Transform functions 169} \Rightarrow \]
\[
\text{for } (i = 0; i < n; i++) \{
    \text{for } (k = 0; k < n; k++) \; \text{hv} [i][k] = \text{matrix}[i][k];
    \text{for } (k = 0; k < n; k++) \; \text{matrix}[i][k] = \text{hv}[k];
\} /* \text{for */}
if (\text{DEBUG}) \{
    \text{cout } \ll \text{"Finished column exchange. \n" } \ll \text{flush;} \n
    \text{cout } \ll \text{"Exiting Transform::inverse(). \n" } \ll \text{endl } \ll \text{flush;} \n
    \}
if (\text{DEBUG}) \text{cout } \ll \text{"Exiting Transform::inverse(). \n" } \ll \text{endl } \ll \text{flush;} \n
\text{return } \text{t;} \}

232. Non-const version (with assignment). [LDF 2002.10.20.] Added this function. I thought of calling in “invert()”, but I decided against it, because I thought having two functions called “inverse()” and “invert()” would be confusing.

There is no point in calling this function with assign \equiv false, since it is equivalent to the const version above with no argument. If it is called with assign \equiv false, a warning is issued, the const version is invoked, and its return value is returned.

\[ \text{Declare Transform functions 168} \Rightarrow \]
\[
\text{Transform inverse(bool assign);} \n
233.

\[ \text{Define Transform functions 169} \Rightarrow \]
\[
\text{Transform Transform::inverse(bool assign)} \}
\[
\text{bool DEBUG = false; } /* \text{true */}
\text{if (DEBUG) cout } \ll \text{"Entering Transform::inverse(bool assign). \n" } \ll \text{flush;} \n
\text{if (assign } \equiv \text{false }) \{
    \text{cerr } \ll \text{"WARNING: In Transform::inverse(bool assign): \n" } \ll
    \text{"assign = false. \nThere’s no reason to do any harm. Continuing. \n\n" } \ll \text{flush;}
    \text{return } \text{inverse();}
\}
\text{if (DEBUG) cout } \ll \text{"Exiting Transform::inverse(bool assign). \n" } \ll \text{flush;} \n
\text{return } (* \text{this } = \text{inverse()}); \}

234. Global variables.

\[ \text{Global variables 18} \Rightarrow \]
\[
\text{Transform user_transform; \n}
235.  (Declarations for the header file 21) +≡
    extern Transform user_transform;

236. Global constants.
    (Global constants 22) +≡
    extern const Transform INVALID_TRANSFORM(INVALID_REAL);
    extern const Transform IDENTITY_TRANSFORM;

237.  (Declarations for the header file 21) +≡
    extern const Transform INVALID_TRANSFORM;
    extern const Transform IDENTITY_TRANSFORM;

238. Putting Transform together.

239.  This is what's compiled.
    (Include files 6)
    (Version control identifier 5)
    (Define class Transform 166)
    (Global variables 18)
    (Global constants 22)
    (Define Transform functions 169)
240. This is what’s written to *transfor.h*,

\{transfor.h 240\} ≡
\{Define class Transform 166\}
\{Declarations for the header file 21\}

241. **Shape** (*shapes.web*). [LDF 2002.10.20] *Shape* is an abstract class. This means that no objects of type *Shape* may be declared. *Shape* is used as a base class for all “drawable” classes, e.g., *Point*, *Path*, and *Dodecahedron*. All objects that are put onto a *Picture* must be either *Shapes* or *Labels*.

[LDF 2003.11.12] Removed the version control identifiers from the CWB files for the distribution of 3DLDF 1.1. They’re still used in my development versions.

[LDF 2003.12.10] Put the version control identifiers back into my release versions for 3DLDF 1.1.4. I’ve already put some of them back in, now I’m doing the rest of them. However, the release versions are now in their own RCS repository.

\{Version control identifier 5\} ≡
\{static string resLdID = "$Id: shapes.web,v 1.4 2004/01/12 21:32:30 lfinsto1 Exp$\};

242. Include files.

\{Include files 6\} ≡
\#include "loader.h"
\#include "pspglb.h"
\#include "io.h"
\#include "colors.h"
\#include "transfor.h"

243. **Shape** class definition.

244. *class Point* is known when *shapes.c* is compiled, because it’s declared (but not defined) in *transfor.web*, which is processed by **cmpl** first.

?? Apparently, both the return value and the argument types of pure virtual functions must be the same, otherwise the derived classes will cause compiler errors. Check where this is stated.

[LDF 2003.06.16] Added declarations of *getMinimum_z()* and *getMean_z()*.

\{Define Shape class 244\} ≡
\class Shape\{
\protected:
\static const signed short DRAWDOT;  /* const values used for output. */
\static const signed short DRAW;
\static const signed short FILL;
\static const signed short FILLDRAW;
\static const signed short UNDRAW;
\static const signed short UNDRAWDOT;
\static const signed short UNFILL;
\static const signed short UNFILLDRAW;
\}

See also section 245.

This code is used in sections 248 and 249.
245. Shape function declarations. All Shape functions are pure virtual functions.

[LDF 2002.10.20.] I've thought about getting rid of get_copy() a couple of times, and using
create_new_(type)() instead, but it's not possible: get_copy() is used in Picture functions for objects
of types derived from Shape where the type is not known. The compiler must resolve to the correct
version of get_copy(), so a virtual Shape function is needed. The "create_new_(type)()" functions are
not virtual Shape functions, and can't be, because the names of the types are part of the name of the
functions. I could solve this problem by renaming get_copy() create_new(), but what I wanted to do was have
a template function create_new() (or just create()). So far, I haven't been able to get this to work. So, for
the time being, I'm leaving things as they are.

[LDF 2002.10.23.] The default arguments to show() are necessary, since

\( \text{Define Shape class 244} = \equiv \)

public: virtual void show(string text = "", char coords = '.w', const bool do_persp = true, const
bool do_apply = true, Focus *f = 0, const unsigned short proj = 0, const real factor = 1)
  const = 0;
virtual Shape *get_copy() const = 0;
virtual bool is_on_free_store() const = 0;
virtual bool set_on_free_store(bool b = true) = 0;
virtual void clear() = 0;
virtual void output() = 0;
virtual vector<Shape *> extract(const Focus &, const unsigned short proj, real factor) = 0;
virtual Transform rotate(const real, const real, const real) = 0;
virtual Transform scale(real, real, real) = 0;
virtual Transform shear(real xy, real xz, real yx, real yz, real zx, real zy) = 0;
virtual Transform shift(real, real, real) = 0;
virtual Transform rotate(const Point &, const Point &, const real) = 0;
virtual Transform operator=(const Transform &) = 0;
virtual void apply_transform(void) = 0;
virtual bool set_extremes() = 0;
virtual real get_minimum_z() const = 0;
virtual real get_maximum_z() const = 0;
virtual real get_mean_z() const = 0;
virtual const valarray<real> get_extremes() const = 0;
virtual void suppress_output() = 0;
virtual void unsuppress_output() = 0; \) ;

246. Static data members.

\( \text{Define static Shape member variables 246} = \equiv \)

const signed short Shape::DRAWDOT = 1;
const signed short Shape::DRAW = 2;
const signed short Shape::FILL = 3;
const signed short Shape::FILLDRAW = 4;
const signed short Shape::UNDRAWDOT = -1;
const signed short Shape::UNDRAW = -2;
const signed short Shape::UNFILL = -3;
const signed short Shape::UNFILLDRAW = -4;

This code is used in section 248.

247. Putting Shape together.
248. This is what's compiled.

(Include files 6)
(Version control identifier 5)
(Define Shape class 244)
(Define static Shape member variables 246)
249. This is what's written to shapes.h.

```
{shapes.h 249} ≡
  {Define Shape class 244}
```

250. Picture and Label (pictures.web).

[LDF 2003.11.12] Removed the version control identifiers from the CWEB files for the distribution of 3DLDF 1.1. They’re still used in my development versions.

[LDF 2003.12.10] Put the version control identifiers back into my release versions for 3DLDF 1.1.4. I’ve already put some of them back in, now I’m doing the rest of them. However, the release versions are now in their own RCS repository.

```
{Version control identifier 5} +≡
  static string rc5_id = "$Id: pictures.web,v 1.4.12 2004/01/12 21:31:45_1finsto1_Exp0$";
```

251. Include files.

```
{Include files 6} +≡
#include "loader.h"
#include "pspglb.h"
#include "io.h"
#include "colors.h"
#include "transfor.h"
#include "shapes.h"
```

252. Label. [LDF 2002.10.20] Labels are the only objects, other than Shapes, that can be put onto Pictures. They are created by the functions label() and dotlabel(), which are currently defined for Points and Paths.

253. Label class definition. A Label contains a pointer to a Point, which is its location, a bool to indicate whether the label should have a dot or not, a string for the text of the label and a string for positioning the text with respect to the label. pt must be a pointer, because Point is an incomplete type here. position can be any of the strings used in METAFONT, i.e., "top", "bot", "lt", "rt", "llft", "rlft", "llt", "rlt", or "urt".

[LDF 2002.10.09] Labels are currently only ever created on the free store.

```
format Label
{Define classes 253} ≡
  class Label {
    friend class Point;
    friend class Picture;
    Point *pt;
    bool dot;
    string text;
    string position;
    public: static bool DO_LABELS;
    {Declare Label functions 255}
  };
```

See also section 261.

This code is used in sections 305 and 306.
254. **Static data members.** `DO_LABELS` is used for globally enabling or suppressing putting Labels onto Pictures. If `DO_LABELS` is `false`, then `label()` and `dotlabel()` have no effect, i.e., no Label is put onto the Picture. Note that Picture has a `private` data member `do_labels`, which is for enabling or suppressing output of Labels for a single Picture (see below). Log

[LDF 2002.10.20.] Added this section. `DO_LABELS` was formerly a global variable defined in `psglb.web`. 

```cpp
// Initialize static Label data members 254
bool Label::DO_LABELS = true;
```

This code is used in section 305.

255. **Declarations for Label functions.** These must be defined in `points.web`, because they require operations on `pt`, and `Point` is an incomplete type in this file. Log

[LDF 2002.10.23.] Added arguments `proj` and `factor`. 

```cpp
// Declare Label functions 255
void output(const Focus &f, const unsigned short proj, real factor, const Transform &t);
Label *geCopy() const;
```

This code is used in section 253.

256. **namespace Projections.** Log

[LDF 2003.06.11.] Added AXON. 

```cpp
// Declare namespace Projections 256
namespace Projections {
    extern const unsigned short PERSP = 0;
    extern const unsigned short PARALLEL_X_Y = 1;
    extern const unsigned short PARALLEL_X_Z = 2;
    extern const unsigned short PARALLEL_Z_Y = 3;
    extern const unsigned short AXON = 4;
    extern const unsigned short ISO = 5;
};
```

This code is used in section 305.
257. **External**, 

\[
\text{\texttt{\{external declaration of namespace \textbf{Projections} 257\}}} \equiv \\
\text{\texttt{namespace Projections \{}} \\
\quad \text{\texttt{extern const unsigned short PERSP;}} \\
\quad \text{\texttt{extern const unsigned short PARALLEL_{X,Y};}} \\
\quad \text{\texttt{extern const unsigned short PARALLEL_{X,Z};}} \\
\quad \text{\texttt{extern const unsigned short PARALLEL_{Z,Y};}} \\
\quad \text{\texttt{extern const unsigned short AXON;}} \\
\quad \text{\texttt{extern const unsigned short ISO;}}} \\
\text{\texttt{\}}} \\
\]

This code is used in section 306.

258. **namespace Sorting.** This namespace contains constants that are passed to \texttt{Picture::output()} for determining how the \texttt{Shapes} on the \texttt{Picture} are sorted in order to determine the order in which they are output.

\[
\text{\texttt{\{Declare namespace \textbf{Sorting} 258\}}} \equiv \\
\text{\texttt{namespace Sorting \{}} \\
\quad \text{\texttt{extern const unsigned short NO\_SORT = 0;}} \\
\quad \text{\texttt{extern const unsigned short MAX\_Z = 1;}} \\
\quad \text{\texttt{extern const unsigned short MIN\_Z = 2;}} \\
\quad \text{\texttt{extern const unsigned short MEAN\_Z = 3;}}} \\
\text{\texttt{\}}} \\
\]

This code is used in section 305.
259. External,
\[
\begin{align*}
\text{extern declaration of namespace } & \text{Sorting } 259 \equiv \text{ namespace Sorting } \{ \\
& \quad \text{extern const unsigned short NO\_SORT;} \\
& \quad \text{extern const unsigned short MAX\_Z;} \\
& \quad \text{extern const unsigned short MIN\_Z;} \\
& \quad \text{extern const unsigned short MEAN\_Z;} \\
& \}\;.
\end{align*}
\]
This code is used in section 306.

260. Picture,
\[
\text{format } \text{Picture } \text{int}
\]

261. Picture class definition. [LDF 2002.08.06.] Note that Label has a public static data member named DO\_LABELS, which is used for globally enabling or suppressing putting Labels onto Pictures (see above).

Picture::do\_labels, on the other hand, is for enabling or suppressing the output of Labels for a single Picture. If a Picture is output when do\_labels is false for that Picture, the Labels are not output. However, the Labels are still on the Picture. If do\_labels is reset to true and the Picture is output again, the Labels will be output this time.

[LDF 2002.04.25.] Added do\_labels. It’s set to true in the constructors and can be set to false using suppress\_labels() .

\[
\begin{align*}
\text{Define classes } 253 \equiv & \text{ class Picture } \{ \\
& \quad \text{Transform } \text{transform;} \\
& \quad \text{vector } \text{Shape } * \text{ shapes;} \\
& \quad \text{vector } \text{Label } * \text{ labels;} \\
& \quad \text{bool } \text{do\_labels;} \\
& \quad \text{public: } \{ \text{Declare Picture functions } 263 \} \\
& \}\;.
\end{align*}
\]

262. Constructors.

263. Default constructor. (No arguments).
\[
\text{Declare Picture functions } 263 \equiv \text{ Picture();}
\]
This code is used in section 261.
264. Define Picture functions 264) ≡
   Picture::Picture()
   : do_labels(true) {}
See also sections 271, 273, 277, 281, 284, 287, 290, 292, 294, 296, 417, 440, 587, 588, 589, 590, 592, 593, 594, 595, 596, 597,
   and 598.
This code is used in sections 305 and 633.

265. Copy constructor.  "! PORTING. [LDF 2002.12.05.] Moved to points.web because I've moved
   Picture::operator=() to points.web, so the latter is undeclared in this file. I've had to do these things
   because of differences between the DEC compiler and the GNU compiler.
(Declare Picture functions 263) +≡
   Picture(const Picture &p);

266. Destructor.  [LDF 2002.10.20.] Picture does not currently have a destructor. clear() takes care of
   deallocating memory and clearing shapes and labels. Defining a destructor would probably cause problems.

267. Assignment. TO DO: Add \LaTeX macro for "PORTING". !! PORTING. [LDF 2002.12.05.] Moved
   to points.web because Picture::clear() and Label::get_copy() are undeclared in this file. This didn't cause
   a problem with the DEC compiler, but it does with the GNU Compiler.
(Declare Picture functions 263) +≡
   void operator=(const Picture &p);

268. Adding elements.

269. Add Picture. This function must be defined in points.web, because it uses Point, which is an
   incompletely defined class here.
   [LDF 2002.04.17.] It seems to be most useful to have the argument Picture p be non-const, in order to
   be able to shift it and add it to *this multiple times. For this to work, it must be possible to set p.transform
   to the identity matrix afterwards. It is possible to do this explicitly by calling reset_transform() on the
   Picture following the call to operator+=(), but it's more convenient to have it done automatically. If it
   turns out to be useful, I can add a const version of this function.

Log

[ Declare Picture functions 263 ] +≡
   void operator+=const Picture &p);

270. Add Shape.
(Declare Picture functions 263) +≡
   void operator+=(Shape *s);
271. Define Picture functions 264) +≡
   void Picture::operator+=(Shape *s)
   {
       shapes.push_back(s);
   }

272. Add Label.
   (Declare Picture functions 263) +≡
   void operator+=(Label *label);

273. Define Picture functions 264) +≡
   void Picture::operator+=(Label *label)
   {
       labels.push_back(label);
   }

274. Suppress Labels. [LDF 2002.04.25.] Added this function. Sometimes it’s irritating to have the labels when a Picture is copied and transformed, and both the original and the transformed versions are output.
   (Declare Picture functions 263) +≡
   inline void suppress_labels()
   {
       do_labels = false;
   }

275. Unsuspend Labels.

   [LDF 2002.12.20.] Added this function.

   (Declare Picture functions 263) +≡
   inline void unsuppress_labels()
   {
       do_labels = true;
   }

276. Kill Labels.

   [LDF 2003.06.07.] Added this function.

   (Declare Picture functions 263) +≡
   void kill_labels();
277. (Define Picture functions 264) +≡
   void Picture::killLabels()
     {
       labels.clear();
     }

278. Transformations. Transformations for Pictures are saved up, and then performed when the Picture is output.

279. Affine transformations.

280. Scale.
   (Declare Picture functions 263) +≡
   Transform scale(real x, real y = 1, real z = 1);

281. (Define Picture functions 264) +≡
   Transform Picture::scale(real x, real y, real z)
     {
       return transform.scale(x, y, z);
     }

282. Shift. (Translation.)

283. real version.
   (Declare Picture functions 263) +≡
   Transform shift(real x, real y = 0, real z = 0);

284. shift() returns a Transform representing the shift, not *this. This makes it possible to apply the transformation to other objects.
   (Define Picture functions 264) +≡
   Transform Picture::shift(real x, real y, real z)
     {
       return transform.shift(x, y, z);
     }

285. Point version. This function must defined in points.web, because Point is an incompletely defined type here.
   (Declare Picture functions 263) +≡
   Transform shift(const Point &p);

286. Rotation around the main axes. rotate() will perform rotation about the x, y and z-axes in that order if called with multiple, non-zero arguments. Rotation only about the y and/or z-axis requires one or two dummy 0 arguments so that rotate() “knows” about which axis (or axes) to rotate.
   (Declare Picture functions 263) +≡
   Transform rotate(const real x, const real y = 0, const real z = 0);
287. Define Picture functions 264) +≡
   Transform Picture::rotate(const real x, const real y, const real z)
   {
      return transform.rotate(x, y, z);
   }

288. Rotation around an arbitrary axis. [LDF 2002.05.03.] This function is defined in points.web, because it has Point arguments, and Point is an incomplete type in this file.

   Log

   [LDF 2002.05.03.] Added this declaration.
   [LDF 2003.05.02.] Changed name of this function from rotate around() to rotate(). This function now overloads rotate() with three real arguments.

   (Declare Picture functions 263) +≡
   Transform rotate(const Point &p0, const Point &p1, const real angle = 180);
   /* Remember to also shear! */

289. Set transform. Log

   [LDF 2003.01.17.] Made non-inline and changed t from plain Transform to const Transform &.

   (Declare Picture functions 263) +≡
   Transform set_transform(const Transform &t);

290. Define Picture functions 264) +≡
   Transform Picture::set_transform(const Transform &t)
   {
      transform = t;
      return t;
   }

291. Multiplying transform. Log

   [LDF 2003.01.17.] Changed t from plain Transform to const Transform &.

   (Declare Picture functions 263) +≡
   Transform operator+=(const Transform &t);
292.  
{Define Picture functions 264} +≡

    Transform Picture::operator*(const Transform &t)  
    {
        transform *= t;
        return t;
    }

293.  Show.

{Declare Picture functions 263} +≡

    void show(string text = "", bool stop = false);

294.  
{Define Picture functions 264} +≡

    void Picture::show(string text, bool stop)
    {
        cout << "Showing picture:\n" << text << "\n" << flush;
        transform.show("transform:");
        cout << "shapes.size() == \n" << shapes.size() << endl << flush;
        cout << "labels.size() == \n" << labels.size() << endl << flush;
        cout << "do_labels == \n" << do_labels << endl << flush;
        cout << "Showing shapes.\n";
        for (vector<Shape>::iterator iter = shapes.begin(); iter != shapes.end(); ++iter) (**iter).show();
        if (stop) {
            cout << "Hit return to continue.\n" << flush;
            getchar();
        }
        cout << "Done showing picture.\n" << flush;
    }

295.  Show transform.

{Declare Picture functions 263} +≡

    void show_transform(string text = "Transform from Picture:");
296. (Define Picture functions 264) +≡
   void Picture::show_transform(string text)
   {
      transform.show(text);
   }

297. Output. [LDF 2002.09.18.] Added the optional real arguments min_x_proj, max_x_proj, etc. The purpose of these is to suppress output of Shapes whose projective extremes fall outside of these limits, whereby the "z" values are not currently checked. They are not set for a particular Focus or Picture, but for a particular invocation of output(). I believe the default values are sufficiently generous, but they can always be changed if it turns out that they're not. Alternatively, I could store them in the Picture or the Focus, if that turns out be more convenient. They are checked in Picture::check_projection_limits().

298. Focus argument.
   (Declare Picture functions 263) +≡
   void output(const Focus &f, const unsigned short projection = Projections::PERSP, real factor = 1, const unsigned short sort_value = Sorting::MAX_Z, const bool do_warnings = true, const real min_x_proj = -40, const real max_x_proj = 40, const real min_y_proj = -40, const real max_y_proj = 40, const real min_z_proj = -40, const real max_z_proj = 40);

299. No Focus argument.
   (Declare Picture functions 263) +≡
   void output(const unsigned short proj = Projections::PERSP, real factor = 1, const unsigned short sort_value = Sorting::MAX_Z, const bool do_warnings = true, const real min_x_proj = -40, const real max_x_proj = 40, const real min_y_proj = -40, const real max_y_proj = 40, const real min_z_proj = -40, const real max_z_proj = 40);

   (Declare Picture functions 263) +≡
   void clear();

301. Reset transform. [LDF 2002.04.17.] Added this function.
   (Declare Picture functions 263) +≡
   inline void reset_transform()
   {
      transform.reset();
   }

302. Global variables.
   (Global variables 18) +≡
   Picture current_picture;

303. (Declarations for the header file 21) +≡
   extern Picture current_picture;

304. Putting Picture and Label together.
305. This is what's compiled.

   (Include files 6)
   (Version control identifier 5)
   (Declare namespace Projections 256)
   (Declare namespace Sorting 258)
   (Define classes 253)
   (Initialize static Label data members 254)
   (Global variables 18)
   (Define Picture functions 264)
306. This is what’s written to pictures.h,

\begin{verbatim}
(pictures.h 306) ≡
 (extern declaration of namespace Projections 257)
 (extern declaration of namespace Sorting 259)
 (Define classes 253)
 (Declarations for the header file 21)
\end{verbatim}

307. Point (points.web).

[LDF 2002.10.20.] **Point** is the most basic drawable (not fillable) type. All of the other **Shapes** contain **Points** and are ultimately defined by their **Points** and the relationships among them. It is therefore understandable that points.web is by far the largest of the source files of 3DLD and that **Point** has the most functions of any class in 3DLD. Many of the functions in the other classes do little more than apply the **Point** version of the function to their **Points**.

---

Log

[LDF 2003.11.12.] Removed the version control identifiers from the CWEB files for the distribution of 3DLD 1.1. They’re still used in my development versions.

[LDF 2003.12.10.] Put the version control identifiers back into my release versions for 3DLD 1.1.4. I’ve already put some of them back in, now I’m doing the rest of them. However, the release versions are now in their own RCS repository.

\begin{verbatim}
 (Version control identifier $) +≡
 static string res_id = "$Id: pictures.web,v1.6.2004/01/12,21:32:01$";
\end{verbatim}

308. Include files.

\begin{verbatim}
 (Include files 6) +≡
 #include "loader.h"
 #include "pspg1b.h"
 #include "crea.new.h"
 #include "io.h"
 #include "colors.h"
 #include "transfor.h"
 #include "shapes.h"
 #include "pictures.h"
\end{verbatim}

309. Point class definition. [LDF 2002.10.20.]

- **world Coordinates** contains the coordinates of the **Point** in the global coordinate system.
- **user Coordinates** and **view Coordinates** are not currently used.
- **user Coordinates** is intended for use with a user-defined coordinate system. For example, it may be convenient to define a coordinate system based on a plane defined by an object in a drawing. The user-defined coordinate system will be defined in terms of the global coordinate system and the **world Coordinates** can be derived from the **user Coordinates** by using the appropriate transformation.
- **projective Coordinates** is used for projecting the three-dimensional **Points** onto two-dimensions for output. Currently, the only projection routines are for perspective and parallel projection. I plan to add others (axonometric, etc.) soon.
- **transform** is used for storing the transformations that are applied the **Point**. It is not necessary to update **world Coordinates** (or **user Coordinates**, if they’re being used) every time a **Point** is transformed. The transformations can be saved up and their result applied to the **Point** when needed. This will be when the **Point** or the **Shape** or **Label** containing it is output, or when a function (such as get_x()) requires up-to-date coordinate values.
- **drawdot_value**, **drawdot_color**, and **pen** are used in the drawing and undrawing functions and in **Point::output()**. They are explained below.
• `projective_extremes` is used in outputting `Pictures`. It’s for culling `Points` that are invisible using a particular `Focus`, or that lie outside the boundaries passed as arguments to `Picture::output()`.

• `do_output` is for enabling or suppressing output of a `Point`. It’s needed when a `Point` has been culled, as described above. Culling does not actually remove a `Point` (or any other `Shape`) from a `Picture`, so a way of suppressing output is needed. However, it must be possible to enable output again, because the `Picture` may be output again using a different `Focus` and/or different values for the boundaries.

• `measurement_units` is a `string` that is attached to the numerical values of the `projective_coordinates` when the `METAPOST` code is output to `out_stream`. It is currently "cm" and at the present time (2002.10.20), it’s not a good idea to use more than one value for a single drawing. Changing this will only become urgent when I start writing the input routine. TO DO.

---

[LDF 2003.04.01.] Added `WORLD_VALUES`, `PROJ_VALUES`, `USER_VALUES`, and `VIEW_VALUES`. They are used in `label()` for labelling `Points` using the values in `world_coordinates`, `projective_coordinates`, etc.

[LDF 2003.06.06.] Changed `WORLD_VALUE`, `PROJ_VALUE`, `USER_VALUE`, and `VIEW_VALUE` to `WORLD_VALUES`, `PROJ_VALUES`, `USER_VALUES`, and `VIEW_VALUES`. Added `WORLD_VALUES_X_Y`, `PROJ_VALUES_X_Y`, `USER_VALUES_X_Y`, and `VIEW_VALUES_X_Y` for suppressing the z-coordinate.

[LDF 2003.06.20] Added `WORLD_VALUES_Z`.

---

```c
format Point Shape
(Define class Point 309) ≡
class Point : protected Shape {
  friend Transform Transform::align_with_axis(Point, Point, char);
private: Transform transform;
  bool on_free_store;
  valarray<real> world_coordinates;
  valarray<real> user_coordinates;
  valarray<real> view_coordinates;
  valarray<real> projective_coordinates;
  signed short drawdot_value;
  const Color *drawdot_color;
  string pen;
protected: /* LDF 2002.09.18. Added `projective_extremes`. It contains the minimum and maximum values for x, y, and z in `projective_coordinates`. */
  valarray<real> projective_extremes;
  bool do_output; /* LDF 2002.09.18. Added this data member. */
public: static string measurement_units;
  static real CURR_Y;
  static real CURR_Z;
  static const short WORLD_VALUES;
  static const short PROJ_VALUES;
  static const short USER_VALUES;
  static const short VIEW_VALUES;
  static const short WORLD_VALUES_X_Y;
  static const short PROJ_VALUES_X_Y;
  static const short USER_VALUES_X_Y;
  static const short VIEW_VALUES_X_Y;
  static const short WORLD_VALUES_Z;
  (Declare Point constructors 324)
  (Declare Point functions 329)
};
```

This code is used in sections 633 and 634.
310.

Log

LDF 2003.04.01. Added initialization of WORLD_VALUES, PROJ_VALUES, USER_VALUES, and VIEW_VALUES. They are used in label() for labelling Points using the values in world_coordinates, projective_coordinates, etc. !! KLUDGE: Using the macro SHRT_MAX because the numeric_limits template doesn’t seem to be available under GNU/Linux using GCC, at least not on the computer I’m using.

LDF 2003.05.06. Added initialization of WORLD_VALUES_X_Y, PROJ_VALUES_X_Y, USER_VALUES_X_Y, and VIEW_VALUES_X_Y.

LDF 2003.05.22. BUG FIX: Changed WORLD_VALUES_Z so that it’s one less than VIEW_VALUES_X_Y. Previously, it had the same value.

(Define static Point data members 310) \equiv

| string Point :: measurement_units = "cm"; |
| real Point :: CURR_Y = 0; |
| real Point :: CURR_Z = 0; |
| const short Point :: WORLD_VALUES = SHRT_MAX; |
| const short Point :: PROJ_VALUES = WORLD_VALUES - 1; |
| const short Point :: USER_VALUES = WORLD_VALUES - 2; |
| const short Point :: VIEW_VALUES = WORLD_VALUES - 3; |
| const short Point :: WORLD_VALUES_X_Y = WORLD_VALUES - 4; |
| const short Point :: PROJ_VALUES_X_Y = WORLD_VALUES - 5; |
| const short Point :: USER_VALUES_X_Y = WORLD_VALUES - 6; |
| const short Point :: VIEW_VALUES_X_Y = WORLD_VALUES - 7; |
| const short Point :: WORLD_VALUES_Z = WORLD_VALUES - 8; |

This code is used in section 633.

311. Type definitions and utility structures. Some of the types are simple enough to be defined using typedef, but others require struct definitions.

Log

[LDF 2002.04.10.] Added these formatting instructions. They are duplicated using “@s” in cwdriver.web.

format point_pair Point
format bool_point Point
format bool_point_pair Point
format bool_point_quadruple Point
format bool_point Point

312. point_pair and bool_point_pair.

(\textit{Type definitions 15} \equiv

typedef pair(Point, Point) point_pair;

typedef pair(bool_point, bool_point) bool_point_pair;

313. bool_point.

Log

LDF 2002.04.15. Added this section. bool_point was formerly a simple typedef. I’ve had to change it to a struct, in order for Point::intersection_points() to return one.
LDF 2003.05.30. Removed the definition of the default constructor to the new section (Define bool_point functions 314). See below for an explanation.

```c
< Type definitions 15 > +≡
struct bool_point {
  bool b;
  Point pt;
  bool_point();
  bool_point(bool bb, const Point &ppt) : b(bb), pt(ppt) {}
  void operator=(const bool_point &bp) {
    b = bp.b;
    pt = bp.pt;
  }
};
```
314. Log

LDF 2003.05.30. Added this section, and the definition of bool_point(void). Previously, b and pt were not set, so their values were unpredictable. I had to remove the definition from the declaration of bool_point, because INVALID_POINT isn’t defined, when the declaration is read by the compiler.

\[
\begin{align*}
\text{(Define bool_point functions 314)} & \equiv \\
\text{bool_point \colon bool_point()} & \\
& \{ \\
& \quad b = false; \\
& \quad pt = INVALID_POINT; \\
& \} \\
\text{This code is cited in section 313.} \\
\text{This code is used in section 633.}
\end{align*}
\]

315. bool_point_quadruple. It would be possible to define this as a pair of pairs, but then the individual element s would be nested inconveniently.

\[
\begin{align*}
\text{(Type definitions 15) \equiv} & \\
\text{struct bool_point_quadruple} & \{ \\
\quad \text{bool_point first;} \\
\quad \text{bool_point second;} \\
\quad \text{bool_point third;} \\
\quad \text{bool_point fourth;} \\
\quad \text{bool_point_quadruple();} \\
\quad \text{bool_point_quadruple(bool_point a, bool_point b, bool_point c, bool_point d) : first(a), second(b), third(c), fourth(d) \{ \}} \\
\quad \text{void operator=(const bool_point_quadruple &arg) \{} \\
\quad \quad \text{first.b = arg.first.b;} \\
\quad \quad \text{first.pt = arg.first.pt;} \\
\quad \quad \text{second.b = arg.second.b;} \\
\quad \quad \text{second.pt = arg.second.pt;} \\
\quad \quad \text{third.b = arg.third.b;} \\
\quad \quad \text{third.pt = arg.third.pt;} \\
\quad \quad \text{fourth.b = arg.fourth.b;} \\
\quad \quad \text{fourth.pt = arg.fourth.pt;} \\
\quad \} \\
\};
\end{align*}
\]
316. Default Constructor for bool_point_quadruple.

LDF 2003.06.1. Added this section. Redefined the default constructor bool_point_quadruple(void), so that first, second, third, and fourth are all set to INVALID_BOOL_POINT. In order to do this, it was necessary to remove the definition from the declaration of bool_point_quadruple, because when the compiler sees it, INVALID_BOOL_POINT isn't defined yet.

(Define bool_point_quadruple functions 316) ≡
bool_point_quadruple::bool_point_quadruple()
  : first(INVALID_BOOL_POINT), second(INVALID_BOOL_POINT),
    third(INVALID_BOOL_POINT),
    fourth(INVALID_BOOL_POINT) { }
This code is used in section 633.

317. bool_real_point. [LDF 2002.04.10] Added this type. Line::intersection_point() returns a bool_real_point. I may change Point::intersection_point() so that it calls Line::intersection_point() and returns a bool_real_point, too.

[LDF 2002.10.26] !! KLUDGE: \newline inserted in the text above to avoid overfull boxes.
(Define definitions 15) +≡
struct bool_real_point {
  bool b;
  real r;
  Point pt;
  bool_real_point(); /* Default constructor. */
  bool_real_point(const bool_real_point &brp)
    : b(brp.b), r(brp.r), pt(brp.pt) {} /* Copy constructor. */
  bool_real_point(const bool &bb, const real &rr, const Point &ppt)
    : b(bb), r(rr), pt(ppt) {} /* Constructor with bool, real, and Point arguments. */
  void operator=(const bool_real_point &brp) /* Assignment operator. */
  {
    b = brp.b;
    r = brp.r;
    pt = brp.pt;
  }
};
318. Default Constructor for bool\_real\_point.

LDF 2003.06.1. Added this section. Redefined the default constructor bool\_real\_point(void), so that \( b \) is set to false, \( r \) is set to INVALID\_REAL, and \( pt \) is set to INVALID\_BOOL\_POINT. In order to do this, it was necessary to remove the definition from the declaration of bool\_real\_point, because when the compiler sees it, INVALID\_REAL and INVALID\_POINT aren't defined yet.

\[
\text{Define bool\_real\_point functions 318} \equiv \\
\text{bool\_real\_point::bool\_real\_point() : } b(\text{false}), r(\text{INVALID\_REAL}), pt(\text{INVALID\_POINT}) \{ \}
\]

This code is used in section 633.

319. Global constants. [LDF 2002.09.25.] Changed this section. I now know that \( \text{const} \)s have internal linkage by default and that I must declare them with \text{extern} in order to give them external linkage.

\[
\text{Global constants 22} + \equiv \\
\text{extern const Point INVALID\_POINT(INVALID\_REAL,INVALID\_REAL,INVALID\_REAL);} \\
\text{extern const Point origin(0,0,0);} \\
\text{extern const bool\_point INVALID\_BOOL\_POINT(false,INVALID\_POINT);} \\
\text{extern const bool\_point\_pair INVALID\_BOOL\_POINT\_PAIR(INVALID\_BOOL\_POINT, INVALID\_BOOL\_POINT);} \\
\text{extern const bool\_real\_point INVALID\_BOOL\_REAL\_POINT(false,INVALID\_REAL,INVALID\_POINT);} \\
\text{extern const bool\_point\_quadruple INVALID\_BOOL\_POINT\_QUADRUPLE(INVALID\_BOOL\_POINT, INVALID\_BOOL\_POINT,INVALID\_BOOL\_POINT,INVALID\_BOOL\_POINT);} \\
\]

320. (Declarations for the header file 21) + \equiv \\
\text{extern const Point INVALID\_POINT;} \\
\text{extern const Point origin;} \\
\text{extern const bool\_point INVALID\_BOOL\_POINT;} \\
\text{extern const bool\_point\_pair INVALID\_BOOL\_POINT\_PAIR;} \\
\text{extern const bool\_real\_point INVALID\_BOOL\_REAL\_POINT;} \\
\text{extern const bool\_point\_quadruple INVALID\_BOOL\_POINT\_QUADRUPLE;} \\

321. Constructors and setting functions.
322. The \texttt{varrays} I use for the various sets of coordinates can be declared in the \texttt{class} declaration, but neither can their size be set nor can they be initialized. \texttt{nullcoordinates} is defined in \texttt{papglb.web} and is a \texttt{varray} of \texttt{reals} with 4 elements = 0. Setting \texttt{worldcoordinates}, etc. to \texttt{nullcoordinates} makes them the right size.

323. Initialize coordinates and limits. \hfill Log

[LDf 2002 4.3.] Now setting \texttt{worldcoordinates}[3], \texttt{usercoordinates}[3], and \texttt{viewcoordinates}[3] = 1. It fixes a bug that showed up when I tried to shift a \texttt{Point} with coordinates \equiv 0.

\begin{verbatim}
<Initialize coordinates and limits 323>

#include <DECCXX>

worldcoordinates = nullcoordinates;
usercoordinates = nullcoordinates;
viewcoordinates = nullcoordinates;
projective_coordinates = nullcoordinates;
#endif
#endif

worldcoordinates.resize(4,0);
usercoordinates.resize(4,0);
viewcoordinates.resize(4,0);
projective_coordinates.resize(4,0);
#endif

if 1
pen = "";
#endif
transform.reset();
#endif

worldcoordinates[3] = 1;
usercoordinates[3] = 1;
viewcoordinates[3] = 1;
projective_extremes.resize(6,0);

This code is used in sections 325, 328, and 332.


<Declare Point constructors 324>

Point();

See also sections 327 and 331.

This code is used in section 309.
325.
(Define Point constructors 325) ≡
Point::Point()
{
  (Initialize coordinates and limits 323)
on_free_store = false;
do_output = true;
}
See also sections 328 and 332.
This code is used in section 633.

326. Three real values.

327. Constructor.

[LDF 2002.12.01.] Made arguments const.

(Declare Point constructors 324) ≡
Point(const real x, const real y = CURR_Y, const real z = CURR_Z);

328.
(Define Point constructors 325) ≡
Point::Point(const real x, const real y, const real z)
{
  (Initialize coordinates and limits 323)
on_free_store = false;
do_output = true;
#if 0  /* [LDF 2002.10.23.] user_transform is not currently in use. It is intended for use in implementing
user-defined coordinate systems. */
  if (user_transform.is_identity())
#endif
  world_coordinates[0] = x;
  world_coordinates[1] = y;
  world_coordinates[2] = z;
  world_coordinates[3] = 1;
}

329. Setting function.

[LDF 2002.12.01.] Made arguments const.
[LDF 2003.03.25.] Changed this function, so that it returns *this instead of void. This makes it possible to
chain invocations of this function.

(Declare Point functions 329) ≡
const Point &set(const real x, const real y = CURR_Y, const real z = CURR_Z);
387, 389, 393, 396, 398, 400, 401, 404, 406, 408, 412, 414, 419, 421, 436, 438, 442, 444, 446, 448, 450, 454, 456, 458, 460, 463,
464, 466, 467, 469, 470, 472, 473, 475, 477, 482, 484, 486, 488, 489, 491, 493, 495, 501, 505, 507, 510, 512, 518, 521, 523,
525, 527, 529, 592, 536, 538, 540, 542, 544, 546, 548, 551, 553, 555, 557, 560, 567, 569, 572, and 573.
This code is used in section 309.
330.  
(Define Point functions 330) \equiv  
const Point &Point :: set(const real x, const real y, const real z)  
{  
  Point p(x, y, z);  
  *this = p;  
  do_output = true;  
  return *this;  /* LDF 2003.03.25. Added this. Formerly, the return value was void. */  
}  


This code is used in sections 633, 657, 694, and 980.  

331.  Copy constructor.  
(Declare Point constructors 324) \equiv  
Point(const Point &p);  

332.  (Define Point constructors 325) \equiv  
Point :: Point(const Point &p)  
{  
  (Initialize coordinates and limits 323)  
  *this = p;  
  on_free_store = false;  
  do_output = true;  
}  

333.  Setting function.  [LDF 2002.10.23.]  This function is unnecessary, because it does nothing that the assignment operator can’t do. However, I’ve tried to use set() a couple of times with a Point argument, so it’s convenient to have it. If nothing else, it prevents compilation from failing occasionally.  

[LDF 2002.10.23.]  Added this function.  

(Declare Point functions 329) \equiv  
void set(const Point &p);
334. Define Point functions 330 ) +≡
   void Point::set(const Point &p)
   {
     *this = p;
     do_output = true;
   }

335. Pseudo-constructor for dynamic allocation, create_new < Point > ( ) is meant to be used
   instead of new ( ) for dynamic allocation of Points. It calls the default constructor (without arguments)
   and then sets on_free_store to true.
   [LDF 2002.10.11.] It is used in various Point functions, and in Path and some classes derived from Path,
   currently Ellipse, Reg_Polygon, and Rectangle. It is intended that objects of these types be declared, i.e.,
   unlike Reg_Cl_Plane_Curve, they are not meant to be used only as base classes. Reg_Cl_Plane_Curve
   does not use create_new < Point > ( ) and it is unlikely that other classes of this kind will use it.

336. Pointer argument.
   (Declare non-member template functions for Point 336 ) +≡
   Point *create_new(const Point *p);
   See also section 337.
   This code is used in sections 633 and 634.

337. Reference argument.

338. Destructor.
   [LDF 2003.08.27.] Added a virtual destructor with an empty definition, because GCC with the “-Wall”
   option issued the following warning: “class Point’ has virtual functions but non-virtual destructor”.
   (Declare Point functions 329 ) +≡
   virtual ~Point();
339. (Define Point functions 330) +≡
   Point :: ~Point ()
   {} 

340. Assignment.

   [LDF 2003.03.25.] Changed this function, so that it returns p instead of void. This makes it possible to
   chain invocations of this function.

   (Declare Point functions 329) +≡
   const Point &operator=(const Point &p);

341. (Define Point functions 330) +≡
   const Point &Point :: operator=(const Point &p)
   {
      transform = p.transform;
      drawdot_value = p.drawdot_value;
      drawdot_color = p.drawdot_color;
      #if 1
      pen = p.pen;
      #endif
      world_coordinates = p.world_coordinates;
      user_coordinates = p.user_coordinates;
      view_coordinates = p.view_coordinates;
      projective_coordinates = p.projective_coordinates;
      projective_extremes = p.projective_extremes;  /* LDF 2002.09.18. Added this line. I think that it
         shouldn't be necessary to do this, but I've added it just to be sure. */
      do_output = true;  /* LDF 2002.09.18.] Probably not necessary, but I might as well. */
      return p;  /* LDF 2003.03.25. Added this. Formerly, the return value was void. */
   }

342. Set on free store.


   (Declare Point functions 329) +≡
   virtual bool set_on_free_store(bool b = true);
(Define Point functions 330) \(\equiv\)

```cpp
bool Point::set_on_free_store(bool b)
{
    on_free_store = b;
    return b;
}
```

344. Clear. I need this function because it’s a virtual function in Shape.

---

[LDF 2002.10.27] Redefined this function. Formerly, it was inline and empty. Now it sets all of the \(x, y,\) and \(z\) coordinates to 0, and resets \(\text{transform}\). It doesn’t seem worthwhile to set \(\text{draw.dot.value}, \text{draw.dot.color,}\) or \(\text{pen}\) to any particular values.

(Declare Point functions 329) \(\equiv\)

```cpp
void clear();
```

345.

(Define Point functions 330) \(\equiv\)

```cpp
void Point::clear()
{
    for (int i = 0; i < 4; i++)
    {
        world_coordinates[i] = user_coordinates[i] = view_coordinates[i] = projective_coordinates[i] = 0;
    }
    transform.reset();
    return;
}
```

346. Clean.

(Declare Point functions 329) \(\equiv\)

```cpp
void clean(int factor = 1);
```
347. (Define Point functions 330) +≡
    void Point::clean(int factor)
    {
        apply_transform();
        real eps = epsilon() * factor;
        for (int i = 0; i < 4; i++)
            if (fabs(world_coordinates[i]) < eps) world_coordinates[i] = 0.0;
    }

348. Returning elements and information.

349. Is identity.
(Declare Point functions 329) +≡
    inline bool is_identity()
    {
        return (transform.is_identity());
    }

350. Epsilon.

---

Log

[LDF 2004.1.2.] Now returning different values, depending on whether real is float or double. TO DO: Try to find out what values would be best. It will be necessary to check how good the value for double is.
[LDF 2004.1.2.] Made epsilon() non-inline.

(Declare Point functions 329) +≡
    static real epsilon();

351. (Define Point functions 330) +≡
    real Point::epsilon()
    {
#if LDF_REAL_DOUBLE
        return .000000001;
#else
        return .0001;
#endif
    }

352. Get Line. Defined in lines.web. Must be defined there, because Line is an incomplete type here.
[LDF 2002.04.12.] Removed this function to lines.web.
(Declare Point functions 329) +≡
    Line get_line(const Point &pt) const;

353. Getting coordinates. ?? Change get_x(), etc., back to inline ??

354. Get all coordinates. [LDF 2002.09.19.] Added this function. ?? [LDF 2002.12.01.] Can I make the Focus * argument const? What is the syntax for a pointer to a const, as opposed to a const pointer? Look up!! Make sure I change this is 3DLDF.texi if I change it here!
355. **Non-const version.** [LDF 2002.09.19.] Added this function.

(Declare Point functions 329) +≡

```cpp
valarray(real) get_all_coords(char coords = 'w', const bool do_persp = true, const bool
do_apply = true, Focus *f = 0, const unsigned short proj = Projections::PERSP, real
factor = 1);
```

356.

(Define Point functions 330) +≡

```cpp
valarray(real) Point::get_all_coords(char coords, const bool do_persp, const bool do_apply, Focus
+f, const unsigned short proj, real factor)
{
    if (do_apply) apply_transform();
    coords = tolower(coords);
    if (coords == 'w') return world_coordinates;
    else if (coords == 'v') return view_coordinates;
    else if (coords == 'u') return user_coordinates;
    else if (coords == 'p') {
        if (f != nullptr) f = &default_focus;
        project(*f, proj, factor);
        return projective_coordinates;
    }
    else {
        cerr << "ERROR\nIn Point::get_all_coords():\n""Argument coords has invalid value: \n""Returning world coordinates.\n"" << flush;
        return world_coordinates;
    }
}
```

357. **const version.** [LDF 2002.09.19.] Added this function.

(Declare Point functions 329) +≡

```cpp
valarray(real) get_all_coords(char coords = 'w', const bool do_persp = true, const bool
do_apply = true, Focus *f = 0, const unsigned short proj = Projections::PERSP, real
factor = 1) const;
```

358.

(Declare Point functions 330) +≡

```cpp
valarray(real) Point::get_all_coords(char coords, const bool do_persp, const bool do_apply, Focus
+f, const unsigned short proj, real factor) const
{
    Point p(*this);
    valarray(real) v = p.get_all_coords(coords, do_persp, do_apply, f);
    return v;
}
```

359. **Get coord.** [LDF 2002.09.14] Added get_coord(). Fixing a bug that caused get_x('p'), etc., to call project() multiple times when doing Path::output().
360. **Non-const version.**  
[LDF 2002.10.27.] The argument \( c \) refers to either the \( x, y, z, \) or \( w \) coordinate.

(Declare Point functions 329) +

```c
real get_coord(char c, char coords = 'w', const bool do_persp = true, const bool do_apply = true, Focus *f = 0, const unsigned short proj = Projections::PERSP, real factor = 1);
```

361.

(Define Point functions 330) +

```c
real Point::get_coord(char c, char coords, const bool do_persp, const bool do_apply, Focus *f, const unsigned short proj, real factor)
```

```c
{
  if (do_apply) apply_transform();
  if (f == 0) f = &default_focus;

  unsigned short ctr;
  c = tolower(c);
  if (c == 'x') ctr = 0;
  else if (c == 'y') ctr = 1;
  else if (c == 'z') ctr = 2;
  else if (c == 'w') ctr = 3;
  else {
    cerr << "ERROR! Invalid coordinate: " << c << "\n" <<
         "Using default\n" << flush;
    ctr = 0;
  }

  coords = tolower(coords);
  if (coords == 'w') return world_coordinates[ctr];
  else if (coords == 'u') return user_coordinates[ctr];
  else if (coords == 'p') {
    if (do_persp) project(*f, proj, factor);
    return projective_coordinates[ctr];
  }

  else if (coords == 'v') return view_coordinates[ctr];
  else {
    cerr << "ERROR! Invalid coordinate: " << coords << endl << "Returning INVALID_REAL.\n\n" << flush;
    return INVALID_REAL;
  }
}
```

362. **const version.**

(Declare Point functions 329) +

```c
real get_coord(char c, char coords = 'w', const bool do_persp = true, const bool do_apply = true, Focus *f = 0, const unsigned short proj = Projections::PERSP, real factor = 1) const;
```
363. \( \text{(Define Point functions 330)} + \equiv \)

\[
\begin{align*}
\text{real Point} & : \text{get coord}(\text{char c, char coords, const bool do_persp, const bool do_apply, Focus *f, const unsigned short proj, real factor}) \text{ const} \\
& \quad \{ \\
& \quad \quad \text{Point p(*this);} \\
& \quad \quad \text{return p.get coord(c, coords, do_persp, do_apply, f, proj, factor);}
\}
\end{align*}
\]

364. Get x.


\( \text{(Declare Point functions 329)} + \equiv \)

\[
\begin{align*}
\text{real get x}(\text{char coords = 'u', const bool do_persp = true, const bool do_apply = true, Focus} \\
& \quad \quad \quad \quad *f = 0, \text{const unsigned short proj = Projections::PERS, real factor = 1}) \text{ const};
\end{align*}
\]

366. \( \text{(Define Point functions 330)} + \equiv \)

\[
\begin{align*}
\text{real Point} & : \text{get x}(\text{char coords, const bool do_persp, const bool do_apply, Focus *f, const unsigned short proj, real factor}) \\
& \quad \{ \\
& \quad \quad \text{return get coord('x', coords, do_persp, do_apply, f, proj, factor);}
\}
\end{align*}
\]

367. const version.

\( \text{(Declare Point functions 329)} + \equiv \)

\[
\begin{align*}
\text{real get x}(\text{char coords = 'u', const bool do_persp = true, const bool do_apply = true, Focus} \\
& \quad \quad \quad \quad *f = 0, \text{const unsigned short proj = Projections::PERS, real factor = 1}) \text{ const;}
\end{align*}
\]

368. \( \text{(Define Point functions 330)} + \equiv \)

\[
\begin{align*}
\text{real Point} & : \text{get x}(\text{char coords, const bool do_persp, const bool do_apply, Focus *f, const unsigned short proj, real factor}) \text{ const} \\
& \quad \{ \\
& \quad \quad \text{return get coord('x', coords, do_persp, do_apply, f, proj, factor);}
\}
\end{align*}
\]

369. Get y.


\( \text{(Declare Point functions 329)} + \equiv \)

\[
\begin{align*}
\text{real get y}(\text{char coords = 'u', const bool do_persp = true, const bool do_apply = true, Focus} \\
& \quad \quad \quad \quad *f = 0, \text{const unsigned short proj = Projections::PERS, real factor = 1});
\end{align*}
\]
371.  
(Define Point functions 330) +≡

real Point::get_y(char coords, const bool do_persp, const bool do_apply, Focus *f, const unsigned short proj, real factor)
{
    return get_coord('y', coords, do_persp, do_apply, f, proj, factor);
}

372.  const version,
(Declare Point functions 329) +≡

real get_y(char coords = 'v', const bool do_persp = true, const bool do_apply = true, Focus *f = 0, const unsigned short proj = Projections::PERSP, real factor = 1) const;

373.  
(Define Point functions 330) +≡

real Point::get_x(char coords, const bool do_persp, const bool do_apply, Focus *f, const unsigned short proj, real factor) const
{
    return get_coord('x', coords, do_persp, do_apply, f, proj, factor);
}

374.  Get z,

375.  Non-const version,
(Declare Point functions 329) +≡

real get_z(char coords = 'w', const bool do_persp = true, const bool do_apply = true, Focus *f = 0, const unsigned short proj = Projections::PERSP, real factor = 1);

376.  
(Define Point functions 330) +≡

real Point::get_z(char coords, const bool do_persp, const bool do_apply, Focus *f, const unsigned short proj, real factor)
{
    return get_coord('z', coords, do_persp, do_apply, f, proj, factor);
}

377.  const version.
(Declare Point functions 329) +≡

real get_z(char coords = 'w', const bool do_persp = true, const bool do_apply = true, Focus *f = 0, const unsigned short proj = Projections::PERSP, real factor = 1) const;

378.  
(Define Point functions 330) +≡

real Point::get_z(char coords, const bool do_persp, const bool do_apply, Focus *f, const unsigned short proj, real factor) const
{
    return get_coord('z', coords, do_persp, do_apply, f, proj, factor);
}

379.  Get w.

\[
\text{(Declare Point functions 329) } \equiv \\
\text{real } \text{get}_w(\text{char } \text{coords} = 'w', \text{const bool } \text{do_persp} = \text{true}, \text{const bool } \text{do_apply} = \text{true}, \text{Focus } *f = 0, \text{const unsigned short } \text{proj} = \text{Projections::PERSP, real factor} = 1);\\
\]

381.

\[
\text{(Define Point functions 330) } \equiv \\
\text{real Point::get}_w(\text{char coord}, \text{const bool do_persp}, \text{const bool do_apply}, \text{Focus } *f, \text{const unsigned short proj}, \text{real factor}) \\
\quad \{ \\
\quad \quad \text{return get}_w(\text{coord}, \text{coords}, \text{do_persp}, \text{do_apply}, f, \text{proj}, \text{factor}); \\
\quad \}
\]

382. const version.

\[
\text{(Declare Point functions 329) } \equiv \\
\text{real get}_w(\text{char coord} = 'w', \text{const bool do_persp} = \text{true}, \text{const bool do_apply} = \text{true}, \text{Focus } *f = 0, \text{const unsigned short proj} = \text{Projections::PERSP, real factor} = 1) \text{const};\\
\]

383.

\[
\text{(Define Point functions 330) } \equiv \\
\text{real Point::get}_w(\text{char coord} = 'w', \text{const bool do_persp}, \text{const bool do_apply}, \text{Focus } *f, \text{const unsigned short proj}, \text{real factor}) \text{const} \\
\quad \{ \\
\quad \quad \text{return get}_w(\text{coord} = 'w', \text{coords}, \text{do_persp}, \text{do_apply}, f, \text{proj}, \text{factor}); \\
\quad \}
\]

384. Get transform.

\[
\text{Log} \quad \text{[LDF 2002.10.27.] Made this function const.} \\
\]

\[
\text{(Declare Point functions 329) } \equiv \\
\text{inline Transform get}_w(\text{() const} \\
\quad \{ \\
\quad \quad \text{return transform;} \\
\quad \}
\]

385. Get copy.

\[
\text{Log} \quad \text{[LDF 2002.10.27.] Made this function const.} \\
\]

\[
\text{(Declare Point functions 329) } \equiv \\
\text{Shape get}_w(\text{() const};
386.
(Define Point functions 330) +≡

Shape *Point :: get_copy() const { Point *p = create_new < Point > (0);
  *p = *this;
  return static_cast(Shape *)(*p); }

387. Is on free store.

 Log

[LDf 2004.01.06.] Made non-inline.

(Declare Point functions 329) +≡

bool is_on_free_store() const;

388.
(Define Point functions 330) +≡

bool Point :: is_on_free_store() const
{
  return on_free_store;
}

389. Slope. [LDf 2002.10.27.] slope() returns the slope of the trace of the line from *this to p on the plane indicated by the char arguments m and n. These should be 'x', 'y', 'z', 'X', 'Y', or 'Z'.

 Log

[LDf 2002.10.27.] Now using world coordinates directly instead of "get" functions.
[LDf 2002.10.27.] Changed argument p from const Point & to Point.

(Declare Point functions 329) +≡

real slope(Point p, char m = 'x', char n = 'y') const;
390. (Define Point functions 330) +≡

real Point::slope(Point p, char m, char n) const { bool DEBUG = false; /* true */
    Point a(*this);
    a.apply_transform();
    p.apply_transform();
    if (a == p) {
        cerr << "ERROR! In Point::slope():\n" << "Points are the same. Returning INVALID_REAL\n" << flush;
        if (DEBUG) {
            a.show("a");
            p.show("p");
        }
        return INVALID_REAL;
    }
    m = tolower(m);
    n = tolower(n);
    if (−(m == 'x' || m == 'y' || m == 'z') && (n == 'x' || n == 'y' || n == 'z') && (m != n)) {  
        cerr << "ERROR! In Point::slope():\n" << "One or both char arguments are invalid\n" << "or they are the same.\n" << m << "," << n << endl << "Returning INVALID_REAL\n" << flush;
        return INVALID_REAL;
    }
    int ctr = m - 'x';
    real a_m_coord = a.world_coordinates[ctr];
    real p_m_coord = p.world_coordinates[ctr];
    ctr = n - 'x';
    real a_n_coord = a.world_coordinates[ctr];
    real p_n_coord = p.world_coordinates[ctr];

391. We often use slope() in order to find out whether a line has slope or not, so an error message is out of place here. A warning is too, probably, but I’m leaving this in here for now, just in case I change my mind.

(Define Point functions 330) +≡

if (a.m_coord ≡ p.m_coord) {
    #if 0
        cerr << "WARNING! In Point::slope():\n" << m << 
            "coordinates of points are equal (no slope)!\n" << "Returning INVALID_REAL\n";
    #endif
        return INVALID_REAL;
    }
    return (a_n_coord - p_n_coord)/(a_m_coord - p_m_coord); }

392. Is on segment.
393. Non-const version.  [LDF 2002.10.29.] is_on_segment() returns a bool_real with the bool indicating whether *this lies on the line segment between p0 and p1, and a real value t representing the distance of *this on the way from p0 to p1. If the bool is true, then 0 <= t <= 1. If t < 0 or t > 1, then *this lies on the line passing through p0 and p1, but not on the segment. If *this doesn’t lie on the line, t will be INVALID_REAL.

[LDF 2002.10.29.] To check whether *this lies on the line, use is_on_line().

Log

[LDF 2002.10.29.] BUG FIX: Added code to check whether the unit vectors *this - p0 and p1 - *this are equal before calculating r. Before I did this, true was returned for Points that weren’t on the line segment.

[LDF 2002.10.29.] Now using world_coordinates directly instead of get_x(), get_y(), and get_z().

(Declare Point functions 329) +

bool_real is_on_segment(Point p0, Point p1);
394.

\{ Define Point functions 330 \} +

```cpp
bool real Point::is_on_segment(Point p0, Point p1) { bool DEBUG = false; /* true */
    if (DEBUG) cout << "Entering: Point::is_on_segment().\n" << flush;
    apply_transform();
    p0.apply_transform();
    p1.apply_transform();
    if (DEBUG) {
        show("this");
        p0.show("p0");
        p1.show("p1");
    }
    if (*this == INVALID_POINT || p0 == INVALID_POINT || p1 == INVALID_POINT) {
        cerr << "ERROR! In Point::is_on_segment():\n" << "One_of_the_points_is_invalid!\n" 
             "Returning: false and INVALID_REAL.\n" << flush;
        return pair<bool, real>(false, INVALID_REAL);
    }
    bool b;
    real r;
    if (p0 == p1 && *this == p0) {
        cerr << "ERROR! In Point::is_on_segment():\n" << "*this_and_the_arguments_p0_and_p1_are_all_equal.\n" 
             "Returning: false and INVALID_REAL.\n";
        return pair<bool, real>(false, INVALID_REAL);
    }
    else if (p0 == p1) {
        cerr << "ERROR! In Point::is_on_segment():\n" << "Arguments_p0_and_p1_are_equal.\n" 
             "Returning: false and INVALID_REAL.\n";
        return pair<bool, real>(false, INVALID_REAL);
    }
    else if (*this == p0) {
        return pair<bool, real>(true, 0.0);
    }
    else if (*this == p1) {
        return pair<bool, real>(true, 1.0);
    }
    /* [LDF 2002.10.29.] Beginning of new code. */
    Point v0(*this - p0);
    Point v1(p1 - *this);
    v0.apply_transform();
    v1.apply_transform();
    if (DEBUG) {
        v0.show("v0");
        v1.show("v1");
    }
    v0.unit_vector(true);
    v1.unit_vector(true);
    if (DEBUG) {
        v0.show("v0");
        v1.show("v1");
    }
```
Point v2(−v1);
if(DEBUG) {
    v2.show("v2");
}
if(v0 ≠ v1 ∧ v0 ≠ v2) {
    if(DEBUG) cout << "Not on line.\n";
    return pair<bool,real>(false, INVALID_REAL);
}
/* [LDF 2002.10.29.] End of new code. */

395. [LDF 2002.10.29.] Calculate how far *this* is on the way from p0 to p1.

LDF Undated. The value $t$ can be calculated from either the x, y, or z-coordinates. We try them in order and return the first one that works. Because of the limited precision with which we are working, it’s possible that the value of $t$ can differ, depending on what coordinates are used to calculate it. In general, this will not be significant, since we’ll mainly be needing this function to determine whether a Point is on a line segment; the exact value of $t$ will usually not be significant.

(Define Point functions 330) +≡
if(p1.world_coordinates[0] ≠ p0.world_coordinates[0])
    $r = (\text{world}\_\text{coordinates}[0] - p0.\text{world}\_\text{coordinates}[0]) / (p1.\text{world}\_\text{coordinates}[0] - p0.\text{world}\_\text{coordinates}[0])$;
else if(p1.world_coordinates[1] ≠ p0.world_coordinates[1])
    $r = (\text{world}\_\text{coordinates}[1] - p0.\text{world}\_\text{coordinates}[1]) / (p1.\text{world}\_\text{coordinates}[1] - p0.\text{world}\_\text{coordinates}[1])$;
else if(p1.world_coordinates[2] ≠ p0.world_coordinates[2])
    $r = (\text{world}\_\text{coordinates}[2] - p0.\text{world}\_\text{coordinates}[2]) / (p1.\text{world}\_\text{coordinates}[2] - p0.\text{world}\_\text{coordinates}[2])$;
else {
    cerr << "ERROR! In Point::is_on_segment()\n" <<
    "Can’t calculate $t$. Returning false and INVALID_REAL.\n\n"
    << flush;
    return pair<bool,real>(false, INVALID_REAL);
}
if($r ≥ 0 ∧ r ≤ 1$) $b = true$;
else $b = false$;
return pair<bool,real>(b, r); }

396. const version.

________________________ Log __________________________

[LDF 2002.10.29.] Added this function.

(Declare Point functions 329) +≡
bool_real is_on_segment(const Point &p0, const Point &p1) const;
397. (Define Point functions 330) +≡
    bool_real Point::is_on_segment(const Point &p0, const Point &p1) const
    {
        Point a(*this);
        return a.is_on_segment(p0, p1);
    }

398. Is on line. [LDF 2002.10.29] TO DO: Maybe add a non-const version. This isn’t urgent, though.

[LDF 2002.10.29] Added this function.

(Declare Point functions 329) +≡
    bool_real is_on_line(const Point &p0, const Point &p1) const;

399. (Define Point functions 330) +≡
    bool_real Point::is_on_line(const Point &p0, const Point &p1) const
    {
        bool_real br = is_on_segment(p0, p1);
        if (br.second != INVALID_REAL) br.first = true;
        return br;
    }

400. Is on Plane. [LDF 2003.06.04] This function returns true, if *this lies on the Plane p, otherwise false. It must be defined in planes.web, because Plane is an incomplete type here.

[LDF 2003.06.04] Added this function.

(Declare Point functions 329) +≡
    bool is_on_plane(const Plane &p) const;

401. Is in triangle. [LDF 2003.06.11] This function returns true, if *this lies within the triangle defined by the three Point arguments, otherwise false. Defined in paths.web, because it uses class Path, which is an incompletely defined type here.

[LDF 2003.06.11] Added this function.

[LDF 2003.06.24] Removed the argument test_points.

(Declare Point functions 329) +≡
    bool is_in_triangle(const Point &p0, const Point &p1, const Point &p2, bool verbose = false) const;

402. Transformations.

403. Affine transformations.
404. Rotation around the main axes.

[Log]

[LDF 2003.01.22.] Replaced body of function. \texttt{Transform::rotate()} returns a \texttt{Transform} representing the rotation only, so I don’t need to use a locally declared \texttt{Transform t} in this function.

(Declare Point functions 329) +≡

\texttt{Transform rotate(const real x, const real y = 0, const real z = 0);}  

405.

(Define Point functions 330) +≡

\texttt{Transform Point::rotate(const real x, const real y, const real z)}  

\{
  \texttt{return transform.rotate(x,y,z);}  
\}

406. Scale.

[Log]

[LDF 2003.01.22.] Replaced body of function. \texttt{Transform::scale()} returns a \texttt{Transform} representing the rotation only, so I don’t need to use a locally declared \texttt{Transform t} in this function.

(Declare Point functions 329) +≡

\texttt{Transform scale(real x, real y = 1, real z = 1);}  

407.

(Define Point functions 330) +≡

\texttt{Transform Point::scale(real x, real y, real z)}  

\{
  \texttt{return transform.scale(x,y,z);}  
\}

408. Shear.

[Log]

[LDF 2003.01.22.] Replaced body of function. \texttt{Transform::shear()} returns a \texttt{Transform} representing the rotation only, so I don’t need to use a locally declared \texttt{Transform t} in this function.

(Declare Point functions 329) +≡

\texttt{Transform shear(real xy, real xz = 0, real yz = 0, real zx = 0, real zy = 0);}  

409.

(Define Point functions 330) +≡

\texttt{Transform Point::shear(real xy, real xz, real yz, real zx, real zy)}  

\{
  \texttt{return transform.shear(xy, xz, yz, zx, zy);}  
\}

410. Shift.

411. Point versions.
§412. Three real arguments.
(Declare Point functions 329) +≡
  Transform shift(real x, real y = 0, real z = 0);

413.
(Define Point functions 330) +≡
  Transform Point::shift(real x, real y, real z)
  {
    Transform t;
    if (x ≠ 0 ∨ y ≠ 0 ∨ z ≠ 0) transform *= t.shift(x, y, z);
    return t;
  }

414. Point argument.
(Declare Point functions 329) +≡
  Transform shift(const Point &p);

415.
(Define Point functions 330) +≡
  Transform Point::shift(const Point &p)
  {
    return shift(p.get_x(), p.get_y(), p.get_z());
  }

416. Transform version. Point argument. [LDF 2002.04.24.] Added this function. It’s declared in transform.web, but must be defined here, because Point is an incomplete type there.
(Define Transform functions 169) +≡
  Transform Transform::shift(const Point &p)
  {
    return shift(p.get_x(), p.get_y(), p.get_z());
  }

417. Picture version. Point argument. [LDF 2002.08.08.] Added this function. It’s declared in pictures.web, but must be defined here, because Point is an incomplete type there.
(Define Picture functions 264) +≡
  Transform Picture::shift(const Point &p)
  {
    return shift(p.get_x(), p.get_y(), p.get_z());
  }

418. Shift times.
[LDF 2003.01.19.] Note that shift times() will only have an effect if it’s called after a call to shift() and before an operation is applied that causes apply_transform() to be called.

[Log]

[LDF 2003.01.19.] Added this section.
419. Three real arguments.

[LDF 2003.01.19.] Added this function.
[LDF 2003.01.22.] Got rid of local Transform t. It wasn’t needed. Now just returning the return value of transform.shift_times().

(Declare Point functions 329) +≡

Transform shift_times(real x, real y = 1, real z = 1);

420.

(Define Point functions 330) +≡

Transform Point::shift_times(real x, real y, real z)
{
    return transform.shift_times(x, y, z);
}

421. Point argument.

[LDF 2003.01.19.] Added this function.

(Declare Point functions 329) +≡

Transform shift_times(const Point &p);

422.

(Define Point functions 330) +≡

Transform Point::shift_times(const Point &p)
{
    return transform.shift_times(p.get_x(), p.get_y(), p.get_z());
}


[LDF 2002.10.23.] align::with::axis() returns the Transform needed to align \( \overrightarrow{OP} \) with one of the main axes.
[LDF 2003.06.04.] BUG: TO DO: Try to find out why I sometimes get erroneous results with rotate(Point, Point, real) (formerly rotate::around()). I think the problem may be here.

[LDF 2002.12.10.] Made this function a friend in class Point. Now calling p0.apply::transform() and p1.apply::transform() at the beginning of this function and using p0.world_coordinates and p1.world_coordinates directly instead of get::x(), get::y(), and get::z().
424.  
(Define |Transform| functions 169) \(\equiv\)

|Transform| Transform :: align_with_axis(Point p0, Point p1, char axis)

```c
/* Default is the z-axis. */
{ bool DEBUG = false; /* true */
  if (DEBUG) cout << "Entering Transform::align_with_axis." << endl << flush;
p0.apply_transform(); /* LDF 2002.12.10. Added these two lines. */
p1.apply_transform();
Transform t;
axis = tolower(axis); /* Upper- or lowercase is permitted for axis. */
if (axis != 'x' && axis != 'y' && axis != 'z') {
  cerr << "ERROR! In Transform::align_with_axis() \"axis\" argument; " <<
        axis << endl << "Returning identity matrix." << endl << flush;
  return t;
}
real angle;
```

425.  [LDF 2002.10.23.] Shift \(p0\) to origin, and shift \(p1\) the same way, so that the relationship between them remains constant.

(Define |Transform| functions 169) \(\equiv\)

```c
if (p0 != origin) {
  if (DEBUG) {
    p0.show("p0");
p1.show("p1");
  }
t.shift(-p0);
p1 *= t;
p1.apply_transform();
p0.apply_transform();
}
```

(Normalize point 433)  /* [LDF 2002.10.23.] Transform the |Point| so that it’s \(x, y, z\) \(\equiv\) positive. See below for the explanation. */

```c
if (DEBUG) {
t.show("\$t\$ outside of normalization");
}
```

|Point| proj_on_xz_plane(p1); /* LDF 2002.10.23.] Get the projection of \(p1\) on the \(x\)-\(z\) plane. */

```c
proj_on_xz_plane.shift(0, -p1.world_coordinates[1]);
if (DEBUG) proj_on_xz_plane.show("proj_on_xz_plane");
```
426. [LDF 2002.10.23.] If we're aligning with the x or y-axis, rotate \( p \) onto the x-y plane and then to the x-axis.

\[
\begin{align*}
\text{(Define Transform functions 169) +=} \\
\text{if } (\text{axis } \equiv 'x') \lor (\text{axis } \equiv 'y') 
\{ \\
\text{Point pt_on_x_axis;} \\
\text{pt_on_x_axis.set(1);} \\
\text{angle = proj_on_x_z_plane.angle(pt_on_x_axis);} \\
\text{if (DEBUG) cout }<< \text{"angle of projection: " }<< \text{angle }<< \text{endl }<< \text{flush;} \\
\text{if (angle }\not\equiv 0 \land \text{angle }\not\equiv \text{INVALID_REAL) }t != \text{pt.rotate}(0,-angle);} \\
\text{if (DEBUG) pt.show("pi.after\_rotation\_to\_x\_y\_plane");} \\
\text{angle = pt.angle(pt_on_x_axis);} \\
\text{if (DEBUG) cout }<< \text{"angle to x-axis: " }<< \text{angle }<< \text{endl }<< \text{flush;} \\
\text{if (angle }\not\equiv 0 \land \text{angle }\not\equiv \text{INVALID_REAL) }t != \text{pt.rotate}(0,0,-angle);} \\
\text{if (DEBUG) pt.show("pi.after\_rotation\_to\_x\_axis");} \\
\}
\end{align*}
\]

427. [LDF 2002.10.23.] If we're aligning with the z-axis, rotate \( p \) onto the y-z plane and then to the z-axis.

\[
\begin{align*}
\text{(Define Transform functions 169) +=} \\
\text{else if (axis }\equiv 'z') 
\{ \\ 
\text{Point pt_on_z_axis;} \\
\text{pt_on_z_axis.set(0,0,1);} /* [LDF 2002.10.23.] This assumes that proj_on_x_z_plane.get_z() }\geq 0. It should be, but if it isn't, the following error handling code takes care of the problem. */ \\
\text{if (proj_on_x_z_plane.get_z() < 0) } \\
\text{cerr }<< \text{"ERROR in Transform::align_with_axis():\n" }<< \\
\text{"proj_on_x_z_plane.get_z() }\leq 0\text{\n" }<< \text{"\@Normalize point\@ should ensure that\" }<< \\
\text{"this value is\@ }\geq 0\text{\n" }<< \text{"Handling the error, but find\out why it happened!\" }<< \\
\text{endl }<< \text{flush;} \\
\text{pt_on_z_axis.set(0,0,-1);} \\
\}
\end{align*}
\]

428. Log

[LDF 2002.12.10.] Added the following conditional. Trying to fix a bug that occurred while porting to GNU/Linux.

\[
\begin{align*}
\text{(Define Transform functions 169) +=} \\
\text{if (proj_on_x_z_plane.world_coordinates[0] }\equiv 0 \land \text{proj_on_x_z_plane.world_coordinates[1] }\equiv 0) \text{ angle } = 0; \\
\text{else angle } = \text{proj_on_x_z_plane.angle(pt_on_z_axis);} \\
\end{align*}
\]

429. (Define Transform functions 169) +=

\[
\begin{align*}
\text{if (DEBUG) cout }<< \text{"angle of projection: " }<< \text{angle }<< \text{endl }<< \text{flush;} \\
\text{if (angle }\not\equiv 0 \land \text{angle }\not\equiv \text{INVALID_REAL) }t != \text{pt.rotate}(0,\text{angle);} \\
\text{if (DEBUG) pt.show("pi.after\_rotation\_to\_z\_y\_plane");} \\
\text{pt.apply\_transform();} \\
\end{align*}
\]
430.  Log

[LDF 2002.12.10.] Added the following conditional. Trying to fix a bug that occurred while porting to GNU/Linux.

[LDF 2003.06.13.] BUG FIX: Changed \texttt{proj_{on z-plane}} to \texttt{p1} in the "if" part of the following conditional. The \texttt{y-coordinate} of \texttt{proj_{on z-plane}} is always 0, so \texttt{angle} was always set to 0. I discovered this bug when I tried rotating a \texttt{Point} in the plane of a \texttt{Reg_Polygon} about a line from the center of the \texttt{Reg_Polygon} in the direction of its normal, and the resulting \texttt{Point} was not in the same plane.

\begin{verbatim}
(Define \texttt{Transform} functions 169) \equiv
  if \((p1\.world\texttt{coordinates}[1] \equiv 0)\) \texttt{angle} = 0;
  else \texttt{angle} = \texttt{p1.angle(p1.on_z_axis)};
\end{verbatim}

431.  

(Define \texttt{Transform} functions 169) \equiv

\begin{verbatim}
  if (\texttt{DEBUG}) \texttt{cont} \leftarrow "\texttt{angle}_\texttt{to}z\texttt{-axis}:u" \leftarrow \texttt{angle} \leftarrow \texttt{endl} \leftarrow \texttt{flush};
  if (\texttt{angle} \neq 0 \land \texttt{angle} \neq \texttt{INVALIDREAL}) \texttt{t} \leftarrow \texttt{p1.rotate(-angle)};
  if (\texttt{DEBUG}) \texttt{p1.show("p1\texttt{after rotation to z-axis}");}
\end{verbatim}

432.  [LDF 2002.10.23.] If we’re aligning with the y-axis, \texttt{p1} must be rotated from the x-axis (where it is now) around the z-axis by 90°. Then it will be on the y-axis.

\begin{verbatim}
(Define \texttt{Transform} functions 169) \equiv
  if (\texttt{axis} \equiv \textit{’y’}) {
    \texttt{t} \leftarrow \texttt{p1.rotate(0,0,90)};
    if (\texttt{DEBUG}) \texttt{p1.show("p1\texttt{after rotation to y-axis}");}
  }
  if (\texttt{DEBUG}) {
    \texttt{cont} \leftarrow "\texttt{p1.magnitude()}=u" \leftarrow \texttt{p1.magnitude()} \leftarrow \texttt{endl} \leftarrow \texttt{flush};
    \texttt{t.show("t at end of align with axis");}
  }
  if (\texttt{DEBUG}) \texttt{cont} \leftarrow "Exiting\texttt{Transform::align with axis}" \leftarrow \texttt{endl} \leftarrow \texttt{flush};
  \texttt{return t;}
\end{verbatim}
433. Normalize point. It makes it easier to determine the correct direction of rotation toward the x-y or y-z plane if \( p_l \) ’s coordinates are all \( > 0 \), so we rotate it in order to make them so. The only case that requires more than a rotation around a single axis is the case that \( x_{pl} \), \( y_{pl} \), and \( z_{pl} \) are all \( < 0 \). It would be nice if I could replace this long conditional with a more elegant construction, but I don’t know one.

\[
\langle \text{Normalize point } 433 \rangle \equiv \\
\{
\text{if (DEBUG) } p_l \text{.show("p}_l\text{ before normalization");
if (} p_l \text{.world_coordinates}[0] < 0 \land p_l \text{.world_coordinates}[1] \geq 0 \land p_l \text{.world_coordinates}[2] \geq 0
\text{/* x negative, y and z positive. */
\text{t} \text{**} p_l \text{.rotate}(0, -90);
\text{else if (} p_l \text{.world_coordinates}[0] \geq 0 \land p_l \text{.world_coordinates}[1] < 0 \land p_l \text{.world_coordinates}[2] \geq 0
\text{/* x positive, y negative, z positive. */
\text{t} \text{**} p_l \text{.rotate}(90);
\text{else if (} p_l \text{.world_coordinates}[0] \geq 0 \land p_l \text{.world_coordinates}[1] \geq 0 \land p_l \text{.world_coordinates}[2] < 0
\text{/* x positive, y positive, z negative. */
\text{t} \text{**} p_l \text{.rotate}(-90);
\text{else if (} p_l \text{.world_coordinates}[0] < 0 \land p_l \text{.world_coordinates}[1] < 0 \land p_l \text{.world_coordinates}[2] \geq 0
\text{/* x negative, y negative, z positive. */
\text{t} \text{**} p_l \text{.rotate}(0, 0, 180);
\text{else if (} p_l \text{.world_coordinates}[0] < 0 \land p_l \text{.world_coordinates}[1] \geq 0 \land p_l \text{.world_coordinates}[2] < 0
\text{/* x negative, y positive, z negative. */
\text{t} \text{**} p_l \text{.rotate}(0, 180);
\text{else if (} p_l \text{.world_coordinates}[0] \geq 0 \land p_l \text{.world_coordinates}[1] < 0 \land p_l \text{.world_coordinates}[2] < 0
\text{/* x positive, y negative, z negative. */
\text{t} \text{**} p_l \text{.rotate}(180);
\text{else if (} p_l \text{.world_coordinates}[0] < 0 \land p_l \text{.world_coordinates}[1] < 0 \land p_l \text{.world_coordinates}[2] < 0
\text{/* All negative. */
\text{a} = p_l \text{.world_coordinates}[0];
\text{t} \text{**} p_l \text{.rotate}(180, 180);
\text{t} \text{**} p_l \text{.shift(a)};
\text{t} \text{**} p_l \text{.rotate}(0, 180);
\text{t} \text{**} p_l \text{.shift(-a)};
\}
\text{p}_l \text{.apply_transform();}
\text{if (DEBUG) }
\{
\text{p}_0 \text{.show("p}_0\text{ after normalization");
\text{p}_l \text{.show("p}_l\text{ after normalization");
\text{t}.show("t after normalization");
\}
\}
\}
\] This code is used in section 425.

434. Rotation around an arbitrary axis.

435. Point versions. [LDF 2002.4.7] Added default value for \( \text{angle} \equiv 180 \).
§436. **Point arguments.** This function first checks to see if \( \ast \text{this} \) lies on the axis. It does this by creating unit vectors in the directions of \( p1 - p0 \) and \( \ast \text{this} - p0 \). If they are equal, or the latter is the former multiplied by -1, then we don’t bother to perform the rotation. Otherwise, we call \texttt{Transform::rotate()} (defined below).

[Log]

[LDF 2002.4.7.] Added default value for \( \text{angle} \equiv 180 \).

[LDF 2003.06.02.] Changed name of this function from \texttt{rotate\_around()} to \texttt{rotate()}. This function now overloads \texttt{rotate()} with three \texttt{real} arguments.

(Declare Point functions 329) +≡

\texttt{Transform rotate(const Point &p0, const Point &p1, const real angle = 180);}  

437.

(Define Point functions 330) +≡

\texttt{Transform Point::rotate(const Point &p0, const Point &p1, const real angle)}

\{
  
  Point \( a = p1 - p0 \);
  Point \( b = \ast \text{this} - p0 \);
  a.\texttt{unit}\_\texttt{vector}(true);
  b.\texttt{unit}\_\texttt{vector}(true);
  
  Transform \( t \);
  if \((a \equiv b \lor a \equiv -b)\) {
    
    \texttt{cerr} \ll \"WARNING! In Point::rotate().\n\ll \"Point to rotate lies on axis.\n\ll \"Returning identity \ll return \( t \);
  }
  
  return \texttt{transform.rotate(p0,p1,angle)};
  
\}

438. **Path argument.** Defined in \texttt{paths\.web}, because \texttt{Path} is still an incomplete type in this compilation unit.

[Log]

[LDF 2002.04.07.] Added default value for \( \text{angle} \equiv 180 \).

[LDF 2003.06.02.] Changed name of this function from \texttt{rotate\_around()} to \texttt{rotate()}. This function now overloads \texttt{rotate()} with three \texttt{real} arguments.

(Declare Point functions 329) +≡

\texttt{Transform rotate(const Path &p, const real angle = 180);}
439. Transform version. Declared in transfor.web. [LDF 2002.09.29] TO DO: Possible BUG!! Actually, the problem that occurred may just have to do choosing the direction of rotation. I've changed the place where the problem occurred, so I'll have to write a routine to test this.

Log

[LDF 2002.10.23.] Changed, so that the direction of \( \overrightarrow{OP} \) is tested. If it is parallel to the \( x \) or \( y \)-axis, then that axis is used for alignment. Otherwise, the \( z \)-axis is used. This may help reduce inaccuracies caused by rotations. Haven't tested it yet. TO DO: Test this!
[LDF 2002.11.03.] TO DO: See if I can't make Point arguments const.
[LDF 2003.08.02.] Changed name of this function from rotate\_around() to rotate(). This function now overloads rotate() with three real arguments.

(Define Transform functions 169) +

Transform Transform::rotate(Point p0, Point p1, const real angle)
{
    bool DEBUG = false; /* true */
    if (DEBUG) cout << "Entering Transform::rotate() \n";
    p0.apply\_transform();
    p1.apply\_transform();
    Point a = p1 - p0;
    a\_unit\_vector(true);
    char axis;
    if (a.get\_x() \equiv 1 \lor a.get\_x() \equiv -1) axis = 'x';
    else if (a.get\_y() \equiv 1 \lor a.get\_y() \equiv -1) axis = 'y';
    else axis = 'z';
    Transform t;
    t.align\_with\_axis(p0, p1, axis);
    Transform i = t.inverse();
    if (axis \equiv 'x') t.rotate(angle);
    else if (axis \equiv 'y') t.rotate(0, angle);
    else t.rotate(0, 0, angle);
    t *= i;
    t\_clean();
    *this *= t;
    clean();
    return t;
}
440. **Picture version.** [LDF 2002.10.20] *angle* is in degrees.

---

[LDF 2003.06.02.] Changed name of this function from `rotate_around()` to `rotate()`. This function now overloads `rotate()` with three real arguments.

---

(Define Picture functions 264) +≡

*Transform Picture*: `rotate(const Point &p0, const Point &p1, const real angle)`

\[
\text{Log}
\]

---

441. **Projection.** [LDF 2002.10.23] For stylistic reasons, and for the sake of clean programming, I believe that the programmer who uses `project()` should ensure that `apply_transform()` has been invoked first. However, `transform` is checked in `project()` and `apply_transform()` is invoked, if required, so invoking `apply_transform()` explicitly beforehand is not strictly speaking necessary.

---

[LDF 2002.09.09.] The new version now almost works Added division of `projective_coordinates` by the value calculated for \( w \). However, it doesn’t work when I use `hex_pattern1()` (and it doesn’t work when I use `hex_pattern1()`). Find out why not!! TO DO: Add routine for calculating \( z \). Then I can add sorting routine in `Picture::output()`.

[LDF 2002.09.14.] I believe I’ve gotten the new version to work now. LOOK UP: Do I need to divide the derived \( z \) by \( w \)? I don’t think it’s necessary. Since the \( z \) values of all of the `Points` would be divided by the same amount, their relative positions would remain the same, since only the relationship “closer or further away” matters, not the exact amounts.

[LDF 2002.09.16.] Added `Focus` argument to this function. Default is `default_focus`, but it was necessary to write a dummy version of this function in order to make this work, because `default_focus` doesn’t exist at the time that this declaration is compiled.

[LDF 2002.09.18.] Changed name of this function from `persp_transform()` to `project` and added `Transform` argument.

[LDF 2003.06.09.] BUG FIX: Added loop, setting all elements of `projective_coordinates` to 0. This was done in the conditionally compiled code for the DEC compiler, but I forgot to do it for GCC when added the declaration of `temp_coordinates` and resized it. It took me about 6–7 hours to find this bug!

---

442. **Focus argument.**

---

[LDF 2003.07.11.] Added defaults for `proj` and `factor`.

---

(Declare Point functions 329) +≡

```
bool project(const Focus &f, const unsigned short proj = Projections::PERSP, real factor = 1);
```
443.
(Define Point functions 330) +

```c
bool Point::project(const Focus &f, const unsigned short proj, real factor){ bool DEBUG = false;
    /* true */
    if (DEBUG) cout << "Entering\nproject()\n" << flush;
    if (!transform.is_identity()) /* LDF 2002.10.23. Added, just to be sure. */
        apply_transform();
    #ifdef __GNUC__
        valarray<real> temp_coordinates;
        temp_coordinates.resize(4, 0);
        for (int i = 0; i < 4; ++i) /* LDF 2003.05.09. Added this loop. */
            projective_coordinates[i] = 0;
    #else
    #ifdef __DECCXX
        valarray<real> temp_coordinates = projective_coordinates = null_coordinates;
    #endif
    #endif
    int i;
    int j; /* LDF 2002.09.18. Transform temp_coordinates by Focus::transform. */
    for (i = 0; i < 4; i++) {
        for (j = 0; j < 4; j++) {
            temp_coordinates[i] += world_coordinates[j] * f.get_transform_element(j, i);
        }
    }
```

```
444. Parallel projection.

[LDF 2002.11.06.] TO DO: Add a way of projecting onto a plane other than the x-y plane. It's possible to achieve the same effect by rotating the Picture before outputting it, but it would be nice to do so without changing the Picture.

The x and y projective coordinates are simply taken from the world coordinates.

Log

[LDF 2002.10.23.] Added this section.
[LDF 2002.12.18.] Changed PARALLEL to PARALLEL_X_Y and added PARALLEL_X_Z and PARALLEL_Z_Y.

(Define Point functions 330) +=

if (proj \equiv Projections::PARALLEL_X_Y \lor proj \equiv Projections::PARALLEL_X_Z \lor proj \equiv Projections::PARALLEL_Z_Y) {
using namespace Projections;
if (factor \equiv 0) {
    cerr \ll "ERROR! In Point::project():\n    "factor\u003d\u003d0, multiplying coordinates by\n    y\u003d0 doesn't make sense.\n    "Using\u003d\u003d1 instead.\n    \n    factor = 1;
}
unsigned short horizontal;
unsigned short vertical;
if (proj \equiv PARALLEL_X_Y \lor proj \equiv PARALLEL_X_Z) /* [LDF 2002.12.18.] Explain this!! */
    horizontal = 0;
else horizontal = 2;
if (proj \equiv PARALLEL_X_Y \lor proj \equiv PARALLEL_Z_Y) vertical = 1;
else vertical = 2;
projective_coordinates[0] = world_coordinates[horizontal] * factor;
projective_coordinates[1] = world_coordinates[vertical] * factor;
projective_coordinates[2] = 0;
projective_coordinates[3] = 1;
if (DEBUG) {
    cout \ll "projective_coordinates:\n    \"projective_coordinates[0] \ll ",\n    \"projective_coordinates[1] \ll ",\n    \"projective_coordinates[2] \ll ",\n    \"projective_coordinates[3] \ll \") \ll endl \ll endl \ll flush;
}
return true;
}
445. Perspective projection. !KLUDGE: See below. [LDF 2002.11.08.] TO DO: Get numbers to output using only decimal notation!

\{ Define Point functions 330 \} +

if (temp_coordinates[2] + f.get_distance() \exists 0) {
  cerr \ll "ERROR! In Point::project():n\" \ll "temp_coordinates[2]\ll " \ll "uf.distance\ll " \ll f.get_distance() \ll endl \ll "Sum\ll " \ll 0. \ll Can't perform division.\" \ll endl \ll "Setting\ll u\ll projection_coordinates\ll u\ll INVALID\ll REAL\" \ll "\ll u\ll and\ll returning.\" \ll flush;
  for (i = 0; i < 4; i++) projection_coordinates[i] = INVALID_REAL;
  return false;
}

else if (temp_coordinates[2] + f.get_distance() < 0) {
  cerr \ll "ERROR! In Point::project():n\" \ll "temp_coordinates[2]\ll " \ll temp_coordinates[2] \ll "uf.distance\ll " \ll f.get_distance() \ll endl \ll "Their\ll sum\ll 0.\ll Point\ll behind\ll focus.\" \ll endl \ll "Setting\ll projection_coordinates\ll INVALID\ll REAL\" \ll "\ll and\ll returning.\" \ll flush;
  for (i = 0; i < 4; i++) projection_coordinates[i] = INVALID_REAL;
  return false;
}

if (DEBUG) {
  cout \ll "temp_coordinates[2]\ll " \ll temp_coordinates[2] \ll "f.get_distance() \ll endl \ll flush;
  cout \ll "save_z\ll " \ll save_z \ll "f.get_distance() \ll endl \ll flush;
  for (i = 0; i < 4; i++) {
    for (j = 0; j < 4; j++) {
      projection_coordinates[i] += temp_coordinates[j] \* f.get_persp_element(j, i);
    }
  }
}

real eps = epsilon();
if (projection_coordinates[3] \exists 0) {
  cerr \ll "ERROR! In Point::project():n\" \ll "projection_coordinates[3]\ll " \ll "\ll endl \ll "This will cause a floating point error.\" \ll endl \ll "Setting\ll projection_coordinates\ll INVALID\ll REAL\" \ll "\ll and\ll returning.\" \ll flush;
  for (i = 0; i < 4; i++) projection_coordinates[i] = INVALID_REAL;
  return false;
}

for (i = 0; i < 4; i++) {
  projection_coordinates[i] /= projection_coordinates[3];
  if (fabs(projection_coordinates[i]) < eps) projection_coordinates[i] = 0;
}  // [LDF 2002.09.14] Set the z value of the perspective coordinates in order to be able to use it for my hidden surface algorithm. */

projection_coordinates[2] = (fabs(save_z) > eps) ? save_z : 0;
if (DEBUG) {
  cout \ll "Perspective_coordinates:\n\";
  for (i = 0; i < 4; i++) {
    cout \ll projection_coordinates[i];
    if (i < 3) cout \ll ",\";
  }
  cout \ll ");\n\" \ll flush;
§445 3DLDF-1.1.5.1  PERSPECTIVE PROJECTION  117

for (i = 0; i < 2; i++)
    /* [LDF 2002.11.07.] KLUDGE. Added this loop. The value used for comparison is slightly larger
    than one that arose while I was testing the constructor of TruncOctahedron. eps was too small.
    */
    if (fabs(projective_coordinates[i]) <= 10.0 * 10^-05) projective_coordinates[i] = 0;
    if (DEBUG) cout << "Exiting,project().\n" << flush;
    return true; }

446. No Focus argument.  [LDF 2002.09.13.] Added this function. This dummy function just passes
default_foci to project(const Focus &f...). This is necessary because it’s impossible make the argument
f optional with default_foci as the default. This is because project() must be declared inside the class
declaration of Point, whereas the declaration of Focus must be later, because it contains Points. It’s not
a problem to use default_foci inside of project(), as long as default_foci is defined before the function is
called.

  (Declare Point functions 329) +=
      bool project(const unsigned short &proj = Projections::PERSP, real factor = 1);

447.

  (Define Point functions 330) +=
      bool Point::project(const unsigned short &proj, real factor)
      {
          return project(default_foci, proj, factor);
      }

448. Applying transformations. This version applies the transformation stored in Point::transform.
!! Add a version that applies a Transform supplied as an argument!! [LDF 2002.12.08.] BUG FIX: See below.
  (Declare Point functions 329) +=
     void applyTransform();
449. (Define Point functions 330) +≡

void Point::apply_transform()
{
  bool DEBUG = false;  /* true */
  if (transform.is_identity())  /* If transform.matrix is the identity matrix, we don’t need to bother
                         to perform the matrix multiplication. */
    return;
  int i;
  int j;
  valarray<real> new_coordinates;
  new_coordinates.resize(4,0);  /* [LDF 2002.12.08] BUG FIX. For GNU CC. */
  if (DEBUG) {
    cout ≡ "x.w.u=" world_coordinates[0] ≡ endl ≡ flush;
    cout ≡ "y.w.u=" world_coordinates[1] ≡ endl ≡ flush;
    cout ≡ "z.w.u=" world_coordinates[2] ≡ endl ≡ flush;
    cout ≡ "w.w.u=" world_coordinates[3] ≡ endl ≡ flush;
  }
  for (i = 0; i < 4; i++) {
    for (j = 0; j < 4; j++) {
      new_coordinates[i] += world_coordinates[j] * transform.matrix[j][i];
      if (new_coordinates[i] ≠ 0) {
        if (DEBUG) {
          cout ≡ "new_coordinates[" ≡ i ≡ "]w.u=" new_coordinates[i] ≡ endl ≡ flush;
        }
      }
  }
  real eps = epsilon();
  for (i = 0; i < 4; i++) {
    if (DEBUG) {
      cout ≡ "new_coordinates[" ≡ i ≡ "]w.u=" new_coordinates[i] ≡ endl ≡ flush;
    }
    world_coordinates[i] = (fabs(new_coordinates[i]) > eps) ? new_coordinates[i] : 0;
  }
  transform.reset();
}

450. Set transform to identity.
(Declare Point functions 329) +≡

void reset_transform();
451. \(\text{Define Point functions 330}\) +\=

\begin{verbatim}
void Point::reset_transform()
{
    transform.reset();
}
\end{verbatim}

452. Drawing.

453. Drawdot.

[LDF 2002.10.26.] \texttt{drawdot()} copies \texttt{*this} and puts the copy onto the vector \texttt{shapes} of the \texttt{Picture} argument \texttt{picture}. The data members \texttt{drawdot_value}, \texttt{drawdot_color}, and \texttt{pen} are only set on the copy, not on \texttt{*this}. All of the drawing and filling functions behave similarly.

[LDF 2003.06.30.] TO DO: Add code for allocating new \texttt{Color}, if \texttt{ddrawdot_color.use_name} is \texttt{false}, as in the drawing and filling functions for \texttt{Path} and \texttt{Solid}.

454. Normal version.

\begin{verbatim}
[Log]
[LDF 2003.07.11.] Made \texttt{ppen} and \texttt{drawdot()} itself \texttt{const}.
\end{verbatim}

(Declare Point functions 329) +\=

\begin{verbatim}
void drawdot(const Color &ddrawdot_color = *Colors::default_color, const string ppen = "", Picture &picture = current_picture) const;
\end{verbatim}

455. \(\text{Define Point functions 330}\) +\=

\begin{verbatim}
void Point::drawdot(const Color &ddrawdot_color, const string ppen, Picture &picture) const{
    bool DEBUG = false;  // true
    if (DEBUG) cout << "Entering drawdot()" << \\
        "\n" << flush;
    Point *pt = create_new < Point> (0);
    *pt = *this;
    pt->drawdot_value = DRAWDOT;
    pt->drawdot_color = &ddrawdot_color;
    #if 1
    pt->pen = ppen;
    #endif
    picture += static_cast<Shape *>(pt);
    if (DEBUG) cout << "Exiting drawdot()" << \\
        "\n" << flush;
}
\end{verbatim}
456. Picture argument first.

[LD 2002.01.24.] Added this version.
[LD 2003.01.31.] Removed default for picture. Having a default made calls to drawdot() with no arguments ambiguous.

(Declare Point functions 329) +

void drawdot(Picture &picture, const Color &ddrawdot_color = Colors::default_color, const string ppen = "") const;

457. (Define Point functions 330) +

void Point::drawdot(Picture &picture, const Color &ddrawdot_color, const string ppen) const
{
    drawdot(ddrawdot_color, ppen, picture);
}

458. Undrawdot. [LD 2002.10.26.] undraw() does not remove a dot from picture, but causes the METAPOST command undrawdot to be written to outStream when picture is output.

(Declare Point functions 329) +

void undrawdot(string ppen = "", Picture &picture = current_picture);

459. (Define Point functions 330) +

void Point::undrawdot(string ppen, Picture &picture){ Point *pt = create_new < Point > (0);
    *pt = *this;
    pt->drawdot_value = UNDRAWDOT;
    pt->drawdot_color = Colors::background_color;
    #if 1
    pt->pen = ppen;
    #endif
    picture += static_cast<Shape *>(pt); }

460. Picture argument first.

[LD 2002.01.24.] Added this version.
[LD 2003.01.31.] Removed default for picture. Having a default made calls to undrawdot() with no arguments ambiguous.

(Declare Point functions 329) +

void undrawdot(Picture &picture, string ppen = "");
461. (Define Point functions 330) +≡
   void Point::undrawdot(Picture &picture, string ppen)
   {
      undrawdot(ppen, picture);
   }

462. Draw. [LDF 2002.10.26] draw() creates a Path with the two Points *this and the argument
   p, and the connector "---", calls Path::draw() for it, and returns the Path. The latter is a line, i.e.,
   Path::get_line_switch() returns true for it. 
   draw() must be defined in paths.web, because Path is an incomplete type here.

463. Normal version. Log

   [LDF 2003.01.15.] Added the argument aarrow.

(Declare Point functions 329) +≡
Path draw(const Point &p, const Color &ddraw_color = *Colors::default_color, string
   ddashed = "", string ppen = "", Picture &picture = current_picture, bool aarrow = false) const;

464. Picture argument first. [LDF 2003.01.15.] This function is convenient for when I want to pass a
   Picture argument.

   [LDF 2002.09.17.] Added this function.
   [LDF 2003.01.15.] Added the argument aarrow.

(Declare Point functions 329) +≡
Path draw(Picture &picture, const Point &p, const Color &ddraw_color = *Colors::default_color,
   string ddashed = "", string ppen = "", bool aarrow = false);

465. Draw arrow.


   [LDF 2003.01.15.] Added this function.
   [LDF 2003.06.03.] Made drawarrow() const.

(Declare Point functions 329) +≡
Path drawarrow(const Point &p, const Color &ddraw_color = *Colors::default_color, string
   ddashed = "", string ppen = "", Picture &picture = current_picture) const;
467. Picture argument first.  [LDF 2003.01.15.] Defined in paths.web,

Log

[LDF 2003.06.03.] Made drawarrow() const.

(Declare Point functions 329) +⇒
Path drawarrow(Picture &picture, const Point &p,
    const Color &ddraw_color = *Colors::default_color, string ddashed = "", string ppen = "")
    const;

468. Undraw.

469. Normal version.  This function must be defined in paths.web, because it uses Path, which is an incomplete type here.

Log

[LDF 2002.4.8.] Added this function.
[LDF 2002.11.03.] Changed this function, so that it returns the Path pa, instead of void.

(Declare Point functions 329) +⇒
Path undraw(const Point &pt,string ddashed = ",",string ppen = ",";Picture
    &picture = current_picture);

470. Picture argument first.

Log

[LDF 2002.09.17.] Added this function. It’s convenient for when I want to pass a Picture argument.
[LDF 2002.11.03.] Changed this function, so that it returns the Path pa, instead of void.

(Declare Point functions 329) +⇒
Path undraw(Picture &picture, const Point &pt,string ddashed = ",",string ppen = ",");

471. Draw help.  [LDF 2002.10.26.] drawHelp() is like draw(), except that the Path is only drawn if the static Path data member doHelpLines ≡ true. This is convenient for drawing construction lines that shouldn’t be output in the final version of a drawing. Also, the default color is *Colors::help_color.

472. Normal version.  [LDF 2002.4.8.] This function must be defined in paths.web, because it uses Path, which is an incomplete type here.

Log

[LDF 2002.4.8.] Added this function.
[LDF 2003.07.13.] Made this function const.

(Declare Point functions 329) +⇒
Path drawHelp(const Point &pt, const Color &ddraw_color = *Colors::help_color, string
    ddashed = ",",string ppen = ",";Picture &picture = current_picture) const;
473. Picture argument first.  [LDF 2002.09.17.] This version is convenient for when I want to pass a Picture argument. Log

[LDF 2002.09.17.] Added this function.
[LDF 2003.07.13.] Made this function const.

(Declare Point functions 329) +≡
Path draw_help(Picture &picture, const Point &pt,
    const Color &ddraw_color = +Colors::help_color, string ddashed = "", string ppen = "") const;

474. Showing.

475. Show.  [LDF 2002.10.26.] The arguments:

string text
If text is non-empty, (i.e., not ""), it's written to standard output (stdout). If it is empty, or show() is called without any arguments, the default is used, namely "Point: ".

char coords
One of the characters 'w', 'p', 'u', or 'v' should be used, to indicate which set of coordinates should be shown: world_coordinates, projective_coordinates, user_coordinates, or view_coordinates, respectively. The latter two exist, but are not currently used. The corresponding uppercase characters can also be used.

const bool do_persp
Only meaningful if the projective_coordinates are being shown (coords argument 'p'). If do_persp ≡ true, then project() is called on *this before projective_coordinates are shown. This is usually what one wants. However, it may sometimes be useful to show the contents of projective_coordinates, without calling project(), in which case do_persp should be false.

const bool do_apply
Usually, apply_transform() should be called on *this before showing a set of coordinates, so the default for do_apply is true. However, it may sometimes be useful to show the values of the coordinates without applying transform, in which case do_apply should be false.

Focus *f
Only meaningful if the projective_coordinates are being shown (coords argument 'p'). Refers to the Focus used for projection. If the default is used, or 0 is passed as the argument explicitly, then the global variable default_focus is used.

const unsigned short proj
Only meaningful if the projective_coordinates are being shown (coords argument 'p'). Refers to the projection used. Currently, I've only programmed the perspective and the parallel projections. The default is the perspective projection.

const real factor
Only meaningful if the projective_coordinates are being shown (coords argument 'p') and the parallel projection is being used. The x and y values in projective_coordinates are multiplied by factor, so it can be used to magnify or shrink the projected image. The default is 1 (no magnification or shrinking).

[LDF 2002.10.26.] TO DO: Add case 'a' for coords for showing all of the sets of coordinates.
[LDF 2002.10.26.] !! KLUDGE: In the text above, I've had to typeset "projective_coordinates" using "\it" explicitly as a couple of places, in order to get the hyphenation to work.
[LDF 2002.11.12.] Added \"relax\" after the arguments to \"\texttt{\textbackslash ARG}\" in the \TeX\ code above in order to suppress a space at the beginning of the first line of the indented paragraph. I couldn't figure out a way of suppressing the space within the definition of \texttt{\textbackslash ARG}.
[LDF 2003.04.30.] Changed, so that a newline is not output following \texttt{text}.

\begin{verbatim}
(Declare Point functions 329) \equiv
  void \texttt{show} (string \texttt{text} = \"\", \texttt{char} \texttt{coords} = \'w\', \texttt{const bool} \texttt{do_persp} = \texttt{true}, \texttt{const bool}
  \texttt{do_apply} = \texttt{true}, \texttt{Focus} \texttt{f} = 0, \texttt{const unsigned short} \texttt{proj} = \texttt{Projections::PERSP}, \texttt{const real}
  \texttt{factor} = 1) \texttt{const} ;

476.
(Define Point functions 330) \equiv
  void \texttt{Point} :: \texttt{show} (string \texttt{text}, \texttt{char} \texttt{coords}, \texttt{const bool} \texttt{do_persp}, \texttt{const bool} \texttt{do_apply}, \texttt{Focus}
  \texttt{f}, \texttt{const unsigned short} \texttt{proj}, \texttt{const real factor}) \texttt{const}
  
  { \texttt{bool} \texttt{DEBUG} = \texttt{false}; \ /* true */
    if (\texttt{text} \equiv \"\") \texttt{text} = \"Point:\";
    \texttt{cout} \ll \texttt{text} \ll \\"\";
    \texttt{coords} = \texttt{tolower} (\texttt{coords});
    if (\texttt{coords} \equiv \texttt{\'w\'}) ; \ /* Do nothing. */
    else if (\texttt{coords} \equiv \texttt{\'p\'}) \texttt{cout} \ll \"Projective coordinates.\n\" \ll \texttt{flush};
    else if (\texttt{coords} \equiv \texttt{\'u\'}) \texttt{cout} \ll \"User coordinates.\n\" \ll \texttt{flush};
    else if (\texttt{coords} \equiv \texttt{\'v\'}) \texttt{cout} \ll \"View coordinates.\n\" \ll \texttt{flush};
    \texttt{else} {
      \texttt{cerr} \ll \"WARNING! in \texttt{\textbackslash show}(): Invalid coordinate \texttt{coords}, argument.\n\" \ll \"Showing world coordinates.\n\" \ll \texttt{flush};
      \texttt{coords} = \texttt{\'w\'};
    }
    if (*\texttt{this} \equiv \texttt{INVALID\_POINT}) {
      \texttt{cerr} \ll \"Point is an INVALID\_POINT. \texttt{\textbackslash cannot show.\n\" \ll \texttt{flush};
      \texttt{return};
    }
    \texttt{if (DEBUG)} \texttt{transform.show (\"Transform before apply_transform\")};
    \texttt{valoray (real) v = get_call_coords} (\texttt{coords}, \texttt{do_persp}, \texttt{do_apply}, \texttt{f}, \texttt{proj}, \texttt{factor});
    \texttt{cout} \ll \"(\" \ll \texttt{v[0]} \ll \".\" \ll \texttt{v[1]} \ll \".\" \ll \texttt{v[2]} \ll \")\n\" \ll \texttt{flush};
    \texttt{if (DEBUG)} {
      \texttt{transform.show (\"Transform after apply_transform\")};
      \texttt{cout} \ll \\"on_free_store\u=\u\" \ll \texttt{on_free_store} \ll \"\";
    }
  }

477. Show transform.
(Declare Point functions 329) \equiv
  void \texttt{show_transform} (string \texttt{text} = \"\");
478.  
(Define Point functions 330) \begin{verbatim}
  void Point::show_transform(string text)
  {
    if (text == "") text = "transform:";
    cout << text << endl;
    transform.show();
  }
\end{verbatim}

479. Outputting.

480. Output operator. [LDF 2002.10.26.] This function is used in Path::output() for writing the x and y values of the projective_coordinates to ostream. All code using this function must ensure that apply_transform() and project() are called first!

\begin{verbatim}
  [LDF 2002.09.16.] Removed calls to apply_transform() and project().
\end{verbatim}

(Declare non-member non-template functions for Point 480) \begin{verbatim}
  ostream & operator<<(ostream & o, Point &p);
\end{verbatim}

See also section 534.
This code is used in section 634.

481.  
(Define non-member non-template functions for Point 481) \begin{verbatim}
  ostream & operator<<(ostream & o, Point &p)
  {
    o << "(" << p.get_x(‘p’, false, false) << Point::measurement_units << "," << p.get_y(‘p’, false, false) << Point::measurement_units << ")";
    return o;
  }
\end{verbatim}

See also section 535.
This code is used in section 633.

482. Suppress output. [LDF 2002.09.18.] Added this function. It’s needed because trying to erase a Shape * from elements in Picture::output() causes a memory fault.
(Declare Point functions 329) \begin{verbatim}
  virtual void suppress_output();
\end{verbatim}

483.  
(Define Point functions 330) \begin{verbatim}
  void Point::suppress_output()
  {
    do_output = false;
  }
\end{verbatim}

484. Unsuppress output. [LDF 2002.09.18.] Added this function. It’s needed because trying to erase a Shape * from elements in Picture::output() causes a memory fault.
(Declare Point functions 329) \begin{verbatim}
  virtual void unsuppress_output();
\end{verbatim}
Define Point functions 330 \[\equiv\]

```cpp
void Point::unsuppress_output()
{
    do_output = true;
}
```

486. Extract. [LDF 2002.10.26] `extract()` is a pure virtual function in Shape. It's called by `Picture::output()`. Each of the Shape pointers on the vector `shapes` in the Picture must be "extracted". For Points, this means projecting the Point using the Focus passed to `extract()` as an argument. If `project()` succeeds, `extract()` returns a vector containing this. Otherwise, it returns an empty vector.

[LDF 2002.10.26] A vector is returned rather than this by itself because it may sometimes be useful to return a collection of Shape pointers rather than a single one. This was formerly the case for Cuboid, but at the present time, no version of `extract()` returns a vector with more than one pointer to Shape.

---

[LDF 2002.09.17] Added const Focus &f argument and error handling code. Now, if the Point cannot be projected onto the projection plane using the Focus f, it is not put onto the vector(Shape *) `Picture::elements`, and consequently never reaches `Picture::output()` and `Point::output()`.

(Declare Point functions 329) \[\equiv\]

```cpp
vector<Shape*> extract(const Focus &f, const unsigned short proj, real factor);
```
487.  
(Define Point functions 330) +≡

```cpp
vector<Shape *> Point::extraction(const Focus &f, const unsigned short proj, real factor)
{
    bool DEBUG = false; /* true */
    vector<Shape *> v; /* [LDF 2002.09.16] Added this error checking code. Check *this first, to
                        make sure that it can be drawn with the current value of default_focus. */
    apply_transform();
    if (!project(f, proj, factor))
        if (DEBUG) {
            cerr << "WARNING! Point::extract(): n" << "Point::onPath cannot be projected. v" <<
                 "Returning an empty vector of Shape*: n" << flush;
        }
    if (DEBUG) {
        cout << "world_coordinates": "" << world_coordinates[0] << ", u" <<
             ";" << "projective_coordinates": "" << projective_coordinates[0] <<
             projective_coordinates[3] << ") n;";
    }
    v.push_back(this);
    return v;
}
```

488. Get extremes. [LDF 2002.09.18.] Added this function. Any code that calls get_extremes() must ensure that project() has been invoked first.

(Declare Point functions 329) +≡

```cpp
virtual inline const valarray<real> get_extremes() const
{
    return projective_extremes;
}
```

489. Get minimum z. [LDF 2002.09.17.] Added this function.

(Declare Point functions 329) +≡

```cpp
virtual real get_minimum_z() const;
```
490. (Define Point functions 330) +≡
real Point::get_minimum_z() const
{
    bool DEBUG = false;           /* true */
    if (DEBUG) cout << "Entering Point::get_minimum_z()" << endl << flush;
    if (DEBUG) cout << "minimum_z==u" << projective_extremes[4] << endl << flush;
    if (DEBUG) cout << "Exiting Point::get_minimum_z()" << endl << flush;
    return projective_extremes[4];
}

491. Get maximum z. [LDF 2002.09.17.] Added this function.
(Declare Point functions 329) +≡
virtual real get_maximum_z() const;

492. (Define Point functions 330) +≡
real Point::get_maximum_z() const
{
    bool DEBUG = false;           /* true */
    if (DEBUG) cout << "Entering Point::get_maximum_z()" << endl << flush;
    if (DEBUG) cout << "maximum_z==u" << projective_extremes[5] << endl << flush;
    if (DEBUG) cout << "Exiting Point::get_maximum_z()" << endl << flush;
    return projective_extremes[5];
}

493. Get mean z. [LDF 2003.03.16.] Added this function.
(Declare Point functions 329) +≡
virtual real get_mean_z() const;

494. (Define Point functions 330) +≡
real Point::get_mean_z() const
{
    return ((projective_extremes[4] + projective_extremes[5])/2);
}

495. Set extremes. This function sets "extreme" values for the x, y, and z-coordinates. This is, of course, trivial for Points, because they only have one x, y and z-coordinate. So the maxima and minima for each coordinate are always the same.
[LDF 2002.10.20.] The programmer who uses setExtremes() must ensure that apply_transform() and project() are invoked before setExtremes().

[Log]
[LDF 2002.09.17.] Added this function.
[LDF 2002.09.18.] Changed the name of this function from set_minimum_z() to set_extremes().

(Declare Point functions 329) +≡
virtual bool set_extremes();
496.
(Define Point functions 330) +3

    bool Point::set_extremes()
    
    { bool DEBUG = false; /* true */
      if (DEBUG) cout << "EnteringPoint::set_extremes()" << endl << flush;
      for (int i = 0; i < 4; i++) {
        if (projective_coordinates[i] == INVALID_REAL) {
          cerr << "ERROR in Point::set_extremes():\n" "projective_coordinates[]\" " i"const )" << ":\n" "Setting every element in\projective_extremes[]" << ":\n" "to\INVALID_REAL\and\returning\false.\n""; return false;
        }
        projective_extremes[i] = projective_coordinates[i]; /* min x */
        projective_extremes[1] = projective_coordinates[0]; /* max x */
        projective_extremes[2] = projective_coordinates[1]; /* min y */
        projective_extremes[3] = projective_coordinates[1]; /* max y */
        projective_extremes[4] = projective_coordinates[2]; /* min z */
        projective_extremes[5] = projective_coordinates[2]; /* max z */
      }
      if (DEBUG) cout << "ExitingPoint::set_extremes()" << endl << flush;
      return true;
    }

497. Comparison classes. [LDF 2003.06.16.] The function classes in this section are used in
Picture::output() to sort the pointers to Shape in vector(Shape *) elements. The argument sort_value,
which should be one of the constants in namespace Sorting, determines which one is used, or if elements
shouldn’t be sorted.
• If Sorting::MIN_Z is passed to Picture::output(), Compare_minimum_z is used for sorting. The elements
are sorted in descending order of the minimum z-value from their projective_extremes.
• If Sorting::MAX_Z is passed to Picture::output(), Compare_maximum_z is used for sorting. The elements
are then sorted in descending order of the maximum z-value from their projective_extremes.
• If Sorting::MEAN_Z is passed to Picture::output(), Compare_mean_z is used for sorting. The elements
are then sorted in descending order of the mean of the minimum and maximum z-values from their
projective_extremes.

In all three of these cases, the Shapes that are furthest from the Focus are output first, so that they can
be covered, if necessary, by Shapes that are closer.
• If Sorting::NO_SORT is passed to Picture::output(), elements is not sorted, and the Shapes are output
in the order in which they were drawn or filled.
498. Compare minimum z.

Log

[LD 2003.05.16.] Added this class.

{Define comparison classes 498} \equiv

\begin{verbatim}
class Compare_minimum_z {
    public: int operator()(const Shape *s1, const Shape *s2) const
    {
        return s1->get_minimum_z() > s2->get_minimum_z();
    }
};
\end{verbatim}

See also sections 499 and 500.

This code is cited in section 596.

This code is used in sections 633 and 634.

499. Compare maximum z.

Log

[LD 2002.09.17.] Added this class.

[LD 2002.09.21.] Changed from “minimum z” to “maximum z”. This works for the more common cases.

{Define comparison classes 498} \equiv

\begin{verbatim}
class Compare_maximum_z {
    public: int operator()(const Shape *s1, const Shape *s2) const
    {
        return s1->get_maximum_z() > s2->get_maximum_z();
    }
};
\end{verbatim}
500. Compare mean $z$. Log

[LDF 2002 09.17.] Added this class.

<Define comparison classes 498> +≡
class Compare_mean_z {
public: int operator() (const Shape *s1, const Shape *s2) const
{
    return (((s1->get_minimum_z() + s1->get_maximum_z()) / 2) >
            ((s2->get_minimum_z() + s2->get_maximum_z()) / 2));
}
};

501. Output. [LDF 2002.10.26.] $output()$ is a pure virtual function in $Shape$. After the $Shape$
pointers on the vector $Picture::shapes$ have been extracted, $output()$ is called for each of the $Shapes$
they point to (except for the ones, if any, where $project()$ failed). $output()$ writes the METAPOST code to
$outstream$.

[LDF 2002 09.16.] Added Focus argument $f$. I want the default to be $default_focus$, but I can’t put it in
the declaration, as I normally do, because $default_focus$ hasn’t been defined yet. I’ve put it in the definition,
and it seems to work. Sometimes it doesn’t, and I don’t know why, nor do I know why it works this time.
If I run into problems, this may be the reason. If necessary, I can make a dummy version of this function
with no argument that calls this version with $default_focus$ as its argument.

[LDF 2002 09.17.] Changed the argument $f$ from Focus to const Focus &. Removed the invocations of
apply_transform() and $project()$ and error handling code to $extract()$.

[LDF 2002.10.23.] Removed the argument $f$. Since $extract()$ takes care of applying $project()$, the
projective_coordinates are already set, so all $output()$ needs to do is write them to $outstream$ with the
proper METAPOST instructions.

(Declare Point functions 329) +≡

void $output()$;
502.

(Define Point functions 330) \equiv

```cpp
void Point::output()
{
    bool DEBUG = false;  // true */
    if (DEBUG) cout << "Entering\nPoint::output().\n" << flush;
    if (do_output \equiv false) {
        if (DEBUG) cout << "Returning.\n" << flush;
        return;
    }
    if (drawdot_value \equiv DRAWDOT) out_stream << "drawdot\n";
    else if (drawdot_value \equiv UNDRAWDOT) out_stream << "drawdot\n";
    else /* DRAWDOT */
    {
        if (drawdot_value \equiv DRAWDOT)
        {
            out_stream << "drawdot\n";
        }
        out_stream << "\n" << projective_coordinates[0] << measurement_units << ",\n" << projective_coordinates[1] << measurement_units << ");\n";
        if (drawdot_color \equiv Colors::default_color) out_stream << ";withcolor\n" << *drawdot_color;
        if (pen \equiv "") out_stream << ";withpen\n" << pen;
        out_stream << ");\n";
        if (DEBUG) cout << "Exiting\nPoint::output().\n" << flush;
    }
}

503. Labelling.

504. Label.
505. string argument. [LDF 2002.10.27.] The arguments:

- **string text_str**
  The text for the label.

- **string position_str**
  Indicates the position of the label text relative to the **Point**. The same **strings** are permitted as in METAPOST. They are written unchecked to **out_stream**, so if an invalid **string** is used, it won’t cause an error in 3DLDF, but it will in METAPOST. The permitted **strings** are: "top" (the default), "bot", "lft", "rt", "ulft" (upper left), "llft" (lower left), "urt" (upper right), "lrt" (lower right), and "" for putting the label right on top of the **Point**. The empty string must be used explicitly, because "top" is the default.

- **bool dot**
  If `true`, then `dotlin` is written to `out_stream` rather than `label`. This argument is mainly for use by the function `dotlin()`, which calls `label()` with `dot = true`.

- **Picture &picture**
  Indicates the **Picture** onto which the **Label** should be placed. The default is `current_picture`.

[LDF 2003.01.15.] TO DO: Add pen argument to `label()` and `dotlin()`!!

---

Log

[LDF 2002.06.14.] Changed `text_str` so that it is no longer optional. It doesn’t make any sense to print empty labels, so I’ve made it a required argument.

[LDF 2002.11.12.] Added \texttt{"\relax"} after the arguments to \texttt{"\textbackslash ARG"} in the \LaTeX{} code above in order to suppress a space at the beginning of the first line of the following indented paragraph. I couldn’t figure out a way of suppressing the space within the definition of \texttt{\textbackslash ARG}.

[LDF 2003.07.09.] Made `text_str`, `position_str`, and `dot` arguments \texttt{const}.

\begin{verbatim}
(Declare Point functions 329) +≡

void label(const string text_str,const string position_str = "top",const bool dot = false,Picure
&picture = current_picture) const;
\end{verbatim}
506. (Define Point functions 330) +

void Point::label(const string text_str, const string position_str, const bool dot, Picture &picture)
const { bool DEBUG = false; /* true */
if (DEBUG) cout << "Entering Point::label()" << "\n" << flush;
if (Label::DO_LABELS == false) {
    if (DEBUG) cout << "Label::DO_LABELS==false. Returning.\n" << "\n" << flush;
    return;
}
if (*this == INVALID_POINT) {
    cerr << "WARNING! In Point::label():\n" << "*this==INVALID_POINT.\nNot doing any\n" << "\n\n" << flush;
    return;
}
Label *lbl = new (Label); lbl->pt = create_new < Point>(0);
*(lbl->pt) = *this;
if (dot == true) lbl->dot = true;
lbl->position = position_str;
if (text_str!="") lbl->text = text_str;
else lbl->text = "Pt. ";
picture += lbl; /* [LDF 2002.10.27.] The Label is pushed onto the vector labels in picture. */
if (DEBUG) cout << "Exiting Point::label()" << "\n" << flush;
return; }

§507. short argument.  [LDF 2003.04.01.] TO DO: Make non-const version of this function! TO DO: Make it possible to use PROJ_VALUES to use the values in projective_coordinates for the label. This will require adding arguments for use by project().

Log

[LDF 2003.04.01.] Changed this function so that it tests whether text_short is equal to WORLD_VALUES, PROJ_VALUES, USER_VALUES, or VIEW_VALUES, which are public const static data members in Point. If text_short is equal to WORLD_VALUES, *this is copied and apply_transform() is called on the copy. This is necessary, because this function is const. Then, the updated values in the world_coordinates vector of the copy are used for the label.

[LDF 2003.05.06.] Added comparison of text_short with WORLD_VALUES_X_Y, PROJ_VALUES_X_Y, USER_VALUES_X_Y, or VIEW_VALUES_X_Y, which are used for suppressing the z-coordinate, when the values from one of the sets of coordinates are used for the label. Also, no longer copying *this, since get_x(), get_y(), and get_z() are const anyway.

[LDF 2003.05.20.] Added "WORLD_VALUES_Z" case.

[LDF 2003.05.22.] BUG FIX: The "WORLD_VALUES_Z" case started with if instead of else if. This caused str() to have an erroneous five-digit integer following the closing parenthesis, when WORLD_VALUES or WORLD_VALUES_X_Y was used. I don’t know why this should have been the case, but changing if to else if fixed the problem. It probably had something to do with the fact that WORLD_VALUES_Z had the same value as VIEW_VALUES_X_Y. I’ve fixed this above today, too.

[LDF 2003.06.06.] Changed the case, where text_short = WORLD_VALUES or text_short = WORLD_VALUES_X_Y: The coordinates surrounded by parentheses are now printed out using TeX’s math mode, i.e., “(x, y, z)” instead of “(x, y, z)”.

[LDF 2003.07.09.] Made text_short, position_str, and dot arguments const.

(Declare Point functions 329) +=

void label(const short text_short, const string position_str = "top", const bool dot = false, Picture &picture = current_picture) const;
508. (Define Point functions 330) +

```cpp
void Point::label(const short text_short, const string position_str, const bool dot, Picture &picture) const
{
    bool DEBUG = false; /* true */
    stringstream s;
    if (text_short == WORLD_VALUES || text_short == WORLD_VALUES_X_Y) {
        if (DEBUG) cout << "It's a WORLD_VALUES or WORLD_VALUES_X_Y \n";
        s << "\"" << getx() << ",\"", gety() << ";\"";
        if (text_short == WORLD_VALUES) s << ",\"" << getz() << ";\"";
        s << "\n";
    } else if (text_short == WORLD_VALUES_Z) {
        if (DEBUG) cout << "It's a WORLD_Z \n";
        s << getz() << ";\"";
    } else if (text_short == PROJ_VALUES) {
        if (DEBUG) cout << "It's a PROJ_VALUES\n";
        cerr << "WARNING! In Point::label():\n" << endl << "text_short==\"PROJ_VALUES\"<<
            "Haven't programmed this case\ yet. \n" << "Returning. \n\n" << flush;
        return;
    } else if (text_short == USER_VALUES) {
        if (DEBUG) cout << "It's a USER_VALUES\n";
        cerr << "WARNING! In Point::label():\n" << endl << "text_short==\"USER_VALUES\"<<
            "Haven't programmed this case\ yet. \n" << "Returning. \n\n" << flush;
        return;
    } else if (text_short == VIEW_VALUES) {
        if (DEBUG) cout << "It's a VIEW_VALUES\n";
        cerr << "WARNING! In Point::label():\n" << endl << "text_short==\"VIEW_VALUES\"<<
            "Haven't programmed this case\ yet. \n" << "Returning. \n\n" << flush;
        return;
    } else {
        if (DEBUG) cout << "It's a some other value. \n";
        s << text_short;
    }
    if (DEBUG) cout << "s.str()==\"" << s.str() << "\"" << endl << flush;
    label(s.str(), position_str, dot, picture);
    return;
}
```

509. Dotlabel. TO DO: Add an optional pen argument. If it's used, use drawdot() with the pen argument, together with label(). When I do this, I should also add real arguments (to both label() and dotlabel()) for shifting the position of the text, and a version with a Point argument for the same purpose. This is so that the dot won't cover the text. [LDF 2003.07.16.]
510. string argument.

[Log] [LDF 2003.07.09.] Made text_str and position_str arguments const.

\begin{verbatim}
(Declare Point functions 329) \equiv
  void dotlabel(const string text_str, const string position_str = "top", Picture &picture = current_picture) const;
\end{verbatim}

511.

(Define Point functions 330) \equiv

\begin{verbatim}
void Point :: dotlabel(const string text_str, const string position_str, Picture &picture) const
{
  label(text_str, position_str, true, picture);
}
\end{verbatim}

512. short argument.

[Log] [LDF 2003.07.09.] Made text_short and position_str arguments const.

\begin{verbatim}
(Declare Point functions 329) \equiv
  void dotlabel(const short text_short, const string position_str = "top", Picture &picture = current_picture) const;
\end{verbatim}

513.

(Define Point functions 330) \equiv

\begin{verbatim}
void Point :: dotlabel(const short text_short, const string position_str, Picture &picture) const
{
  label(text_short, position_str, true, picture);
}
\end{verbatim}
514. [LDF 2002.09.06.] Commented out \(\sim\)Label(). This was the cause of a bug that caused a memory fault when I tried to use a label in figure 2 (beginfofig(2)) after having used it in figure 1 and then invoking current\_picture\_clear() in between.

(Define Label functions 514) \(=\)

```cpp
#if 0
  Label::\(\sim\)Label()
  {
    if (pt \neq 0) delete pt;
  }
#endif
```

See also sections 515 and 516.

This code is used in section 633.

515. Get copy of Label.

(Define Label functions 514) \(=\)

```cpp
Label \+ Label::get_copy() const
{
  Label \+ lbl = new Label();
  lbl-\+ pt = create_new < Point > (0);
  *(lbl-\+ pt) = *pt;
  lbl-\+ dot = dot;
  lbl-\+ text = text;
  lbl-\+ position = position;
  return lbl;
}
```

516. Output Labels. [LDF 2002.10.23] Declared in pictures.web. Must be defined here, because Point is an incomplete type there.

517. Matrix operations.

518. Multiplication by a Transform with assignment.
[LDF 2002.11.06.] BUG FIX: This function now returns \( t \) instead of \textit{transform}. This makes it possible to chain expressions using \texttt{operator \*=()}. \\

\begin{verbatim}
(Declare Point functions 329) +\equiv
  Transform operator \*=((\text{const} \ Transform \ &t));
\end{verbatim}

519.

\begin{verbatim}
(Define Point functions 330) +\equiv
  Transform Point::operator\*=(\text{const} \ Transform \ &t)
  {
    return (transform \*= \( t \));
  }
\end{verbatim}

520. Vector operations. [LDF 2002.10.27.] Note that the vector operations don’t affect the w coordinate.

\begin{verbatim}
[\text{LDF 2002.10.27.}] In the functions \texttt{operator\+(\()}, \texttt{operator\*=(\()}, \texttt{operator\-()}, \texttt{and} \texttt{operator\-=()}: \text{It}
\text{doesn’t seem worth it to write non-\text{const} versions, although I could. Now using the elements of\texttt{p.world\_coordinates} directly instead of using \texttt{get\_x()}}, \texttt{get\_y()}}, \text{and} \texttt{get\_z()}. \text{This is safe, as is calling}\texttt{apply\_transform()} \text{on} \texttt{p}, \text{and saves the cost of three function calls.}
\end{verbatim}

521. Vector addition.

\begin{verbatim}
(Declare Point functions 329) +\equiv
  Point operator\+=(Point \ p) \text{\ const;}
\end{verbatim}

522.

\begin{verbatim}
(Define Point functions 330) +\equiv
  Point Point::operator\+=(Point \ p) \text{\ const}
  {
    Point \ a;
    a = *this;
    p.apply\_transform();
    a.shift(p.world\_coordinates[0], p.world\_coordinates[1], p.world\_coordinates[2]);
    return a;
  }
\end{verbatim}

523. Vector addition with assignment.

\begin{verbatim}
(Declare Point functions 329) +\equiv
  void operator\+=((Point \ p);
\end{verbatim}
524.  
(Define Point functions 330 ) +≡
void Point::operator+(Point p)
{
    p.apply_transform();
    shift(p.world_coordinates[0], p.world_coordinates[1], p.world_coordinates[2]);
}

525. Vector subtraction.
(Declare Point functions 329 ) +≡
Point operator−(Point p) const;

526.  
(Define Point functions 330 ) +≡
Point Point::operator−(Point p) const
{
    Point a(*this);
    p.apply_transform();
    a.shift(−p.world_coordinates[0], −p.world_coordinates[1], −p.world_coordinates[2]);
    return a;
}

527. Vector subtraction with assignment.
(Declare Point functions 329 ) +≡
void operator−=(Point p);

528.  
(Define Point functions 330 ) +≡
void Point::operator−=(Point p)
{
    p.apply_transform();
    shift(−p.world_coordinates[0], −p.world_coordinates[1], −p.world_coordinates[2]);
}

529. Vector-scalar multiplication with assignment.

[Log: LDF 2002.10.27. Made argument r const. Changed return value from Point & to void.]
[Log: LDF 2003.06.14. Changed return value from void to real. It now returns the argument r. This makes it possible to chain invocations of this function.]

(Declare Point functions 329 ) +≡
real operator*=(const real r);
530. ?? I'm not sure whether multiplication with a scalar is commutative with transformations. I doubt it. Therefore, I apply transform before multiplying.

\[
\text{Define Point functions 330 } 
\text{ real Point \& operator \( * \)(const real \( r \))}
\]

\[
\{
\text{ apply\_transform();}
\text{ for (int i = 0; i < 3; i++) world\_coordinates[i] *= r;}
\text{ return r;}
\}
\]

531. Vector-scalar multiplication.

532. Member version (Point first).

\[\text{Log} \quad \text{[LDF 2002.10.27.] Made this function and the argument } r \text{ const.}\]

\[
\text{Declare Point functions 329 } 
\text{ Point operator \( + \)(const real \( r \)) const;}
\]

533.

\[
\text{Define Point functions 330 } +
\text{ Point Point \& operator \( * \)(const real \( r \)) const}
\]

\[
\{
\text{ Point a(*this);}
\text{ a.apply\_transform();}
\text{ a *= r;}
\text{ return a;}
\}
\]

534. Non-member version (scalar first).

\[\text{Declare non-member non-template functions for Point 480} +
\text{ Point operator \( + \)(const real, const Point \&p);}
\]

535.

\[\text{Define non-member non-template functions for Point 481} +
\text{ Point operator \( * \)(const real \( r \), const Point \&p)}
\]

\[
\{
\text{ return p \& r;}
\}
\]

536. Unary minus.

\[\text{Log} \quad \text{[LDF 2002.10.27.] Made this function const.}\]

\[
\text{Declare Point functions 329} +
\text{ Point operator \(-\)() const;}
\]
537.  Define Point functions 330) +
   Point Point::operator－( ) const
   {
      Point a(*this);
      a.apply_transform();
      a *= -1;
      return a;
   }

538.  Vector-scalar division with assignment.  ?? I'm not sure whether division with a scalar is
       commutative with transformations. I doubt it. Therefore, I apply transform before dividing.

[Log]
[LDF 2002.10.27.] Made the argument r const.

539.  void operator/(const real r);

540.  Vector-scalar division.

[Log]
[LDF 2002.10.27.] Made this function and the argument r const.

541.  Declare Point functions 329) +
   Point operator/(const real r) const;
541.  (Define Point functions 330) \(+\equiv\)

\[
\text{Point Point::operator/(const real } r \text{) const}
\]

\[
\begin{align*}
&\text{Point } a(*\text{this}); \\
&\text{a.apply\_transform();} \\
&\text{a }/= \text{ } r; \\
&\text{return } a;
\end{align*}
\]

542. Dot product.

\[
\begin{align*}
&\text{Log} \\
&\text{[LDF 2002.10.27.] Changed this function and argument } p \text{ to } \text{const}. \text{ Now using } \text{world\_coordinates} \text{ directly} \\
&\text{instead of } \text{get\_x()}, \text{get\_y()}, \text{and } \text{get\_z}(). \\
&\text{[LDF 2003.07.11.] Changed, so that if the dot product is less than } \text{Point::epsilon()}, 0 \text{ will be returned.}
\end{align*}
\]

\[
\begin{align*}
&\text{(Declare Point functions 329) } +\equiv \\
&\text{real } \text{dot\_product(Point } p \text{) const;}
\end{align*}
\]

543.  (Define Point functions 330) \(+\equiv\)

\[
\begin{align*}
&\text{real Point::dot\_product(Point } p \text{) const}
\end{align*}
\]

\[
\begin{align*}
&\text{Point } a(*\text{this}); \\
&\text{a.apply\_transform();} \\
&\text{p.apply\_transform();} \\
&\text{real } r = ((a.\text{world\_coordinates}[0] * p.\text{world\_coordinates}[0]) + (a.\text{world\_coordinates}[1] * \\
&\hspace{1cm} p.\text{world\_coordinates}[1]) + (a.\text{world\_coordinates}[2] * p.\text{world\_coordinates}[2])); \\
&\text{if } (\text{fabs}(r) < \text{Point::epsilon()} ) r = 0; \\
&\text{return } r;
\end{align*}
\]

544. Cross product.

\[
\begin{align*}
&\text{Log} \\
&\text{[LDF 2002.10.27.] Changed this function and argument } p \text{ to } \text{const}. \text{ Now using } \text{world\_coordinates} \text{ directly} \\
&\text{instead of } \text{get\_x()}, \text{get\_y()}, \text{and } \text{get\_z}().
\end{align*}
\]

\[
\begin{align*}
&\text{(Declare Point functions 329) } +\equiv \\
&\text{Point cross\_product(Point } p \text{) const;}
\end{align*}
\]
545.  
(Define Point functions 330) +≡

Point Point::cross_product(Point p) const
{
    Point a(*this);
    a.apply_transform();
    p.apply_transform();
    Point r;
    r.world_coordinates[2] = (a.world_coordinates[0] * p.world_coordinates[1]) - (a.world_coordinates[1] * p.world_coordinates[0]);  /* z. */
    return r;
}

546. Magnitude.  [LDF 2002.10.27.]  
The magnitude of a Point is its distance from the origin and is equal to \( \sqrt{x^2 + y^2 + z^2} \).

Since floats are so large anyway, and since I can easily redefine real to use double or double double, (or whatever it’s called Look up!), it’s not really necessary to use an algorithm to approximate \( \sqrt{x^2 + y^2 + z^2} \) (viz., “Pythagorean addition” in Knuth, Metafont: The Program. (Get reference!)) However, it might be nice to use it anyway.

[Log]  Made this function const. Now using world_coordinates directly instead of get_x(), get_y(), and get_z().

( Declare Point functions 329 ) +≡

real magnitude() const;
547. (Define Point functions 330) +≡
real Point::magnitude() const
{
  bool DEBUG = true;  /* false */
  real r;
  real temp;
  Point a(*this);
  a.apply_transform();
  if ((a.world_coordinates[0] > MAX_REAL_SQRT) ∨ (a.world_coordinates[1] >
       MAX_REAL_SQRT) ∨ (a.world_coordinates[2] > MAX_REAL_SQRT))
  {
    cerr << "ERROR::Point::magnitude().\n"   ("Point\u00a0has\u00a0coordinate\u00a0too\u00a0large\u00a0for\u00a0squaring!\n"   ("Returning\u00a0INVALID\_REAL.\n"
      return INVALID_REAL;
  }
  r = a.world_coordinates[0] ∗ a.world_coordinates[0];
  temp = a.world_coordinates[1] ∗ a.world_coordinates[1];
  if (MAX_REAL − r < temp) {
    cerr << "IN\u00a0magnitude().\n";
    cerr << "Point\u00a0has\u00a0too\u00a0great\u00a0magnitude!\n";  /* !! This show() outputs to stdout. It
would be nice to output it to stderr instead. Must write function for this. */
    cerr << "Returning\u00a0INVALID\_REAL.\n"
      return INVALID_REAL;
  }
  r += temp;
  if (MAX_REAL − r < temp) {
    cerr << "IN\u00a0magnitude().\n";
    cerr << "Point\u00a0has\u00a0too\u00a0great\u00a0magnitude!\n";  /* !! This show() outputs to stdout. It
would be nice to output it to stderr instead. Must write function for this. */
    cerr << "Returning\u00a0INVALID\_REAL.\n"
      return INVALID_REAL;
  }
  r += temp;
  return sqrt(r);
}

548. Angle between two vectors.

Log

[LD 2002.10.27.] Made this function const.
[LD 2003.07.27.] Made the argument p a const Point &. No longer copying *this. Now using
dot_product() instead of calculating the angle "by hand". Simplified the code of the function.

(Declare Point functions 329) +≡
real angle (const Point &p) const;
549. 
(Define Point functions 330) +≡
real Point::angle(const Point &p) const
{
    bool DEBUG = false; /* true */
    real mag = magnitude();
    real p_mag = p.magnitude();
    if (mag == INVALID_REAL) {
        cerr << "WARNING! In angle(). magnitude() failed." << "Returning INVALID_REAL.
";
        return INVALID_REAL;
    } else if (mag == 0) {
        if (DEBUG) cerr << "WARNING! In angle().\n" << "this has magnitude 0.\n" << "Returning INVALID_REAL.\n";
        return INVALID_REAL;
    } else if (p_mag == INVALID_REAL) {
        cerr << "WARNING! In angle().\n" << "p has magnitude 0.\n" << "Returning INVALID_REAL.\n";
        return INVALID_REAL;
    } else if (p_mag == 0) {
        if (DEBUG) cerr << "WARNING! In angle().\n" << "p has magnitude 0.\n" << "Returning INVALID_REAL.\n";
        return INVALID_REAL;
    } else return (180/PI * acos(dot_product(p)/(mag * p_mag)));
}

550. Unit vector. 

[Log] Added a second version. If assign is not used, unit_vector() can be const, so I now have a const version with no argument and a non-const one for assignment that should be called with the argument true.

551. With assignment. This version should only ever be called with true as its argument. Using false will work, unless *this is const, in which case it will cause a compilation error. [LDF 2002.10.27.]

If the optional silent argument is true, warning messages will be suppressed, otherwise, they will be issued. The const version below can't have an optional silent argument, because that would make a call to this function with one argument ambiguous.

[Log] If magnitude() fails, unit_vector() now returns INVALID_POINT instead of origin.

[LDF 2003.07.01.] Added the silent argument to suppress warning messages. I kept getting warnings when this function was called from intersection functions, in cases where it wasn't a problem, that a Point (vector) had 0 magnitude.

(Declare Point functions 329) +≡
Point unit_vector(const bool assign, const bool silent = false);
552.

(Define Point functions 330) ⊨≡

Point Point::unit_vector(const bool assign, const bool silent)
{
    if (assign ≡ false) {
        if (silent) {
            cerr ≡ "WARNING! In Point:unit_vector:
" ≡ "Don’t call this function with false as its argument.
" ≡ "Use unit_vector() without an argument instead.
" ≡ "Calling unit_vector() without an argument.

flush;
        }
        return unit_vector();
    }

apply_transform();

real m = magnitude();

if (m ≡ 0)
    /* LDF 2002.04.10. Added this error handling code for the case where *this has no magnitude. */
{
    if (silent) {
        cerr ≡ "WARNING! In Point:unit_vector:
" ≡ "Point(vector) has no magnitude. Returning INVALID_POINT.

flush;
    }
    return INVALID_POINT;
}

for (int i = 0; i < 3; i++) world_coordinates[i] /= m;

world_coordinates[3] = 1;   /* LDF 2002.10.27. Setting the w-coordinate to 1, just to be sure. */

return *this;
}

553. const (no assignment).

(Declare Point functions 329) ⊨≡

Point unit_vector() const;

554.

(Define Point functions 330) ⊨≡

Point Point::unit_vector() const
{
    Point a(*this);

    return a.unit_vector(true);
}

555. Mediation.

Log

[LDF 2003.12.09.] Changed from a non-member to a const member function.

(Declare Point functions 329) ⊨≡

Point mediate(Point p, const real r = .5) const;
556.  

\( \text{Define Point functions 330) +\equiv} \)

\[
\text{Point Point::mediate(Point p, const real r) const}
\{
\quad \text{Point } t(*this);
\quad t *= (1 - r);
\quad p *= r;
\quad \text{return } (t + p);
\}
\]

557.  Get normal.  \( get\text{\textunderscore normal()} \) must be defined in \texttt{paths\textunderscore web}, because it uses a \texttt{Path} in its definition, which is an incompletely defined type in this file.

---

558.  Comparison.

559.  Equality.  \( \text{!! I may have to adjust to value of } eps. \text{ It would be nice to be able to use } \text{epsilon()} \), but for other purposes \( \text{epsilon()} \) must be smaller.  Transformations seem to cause fairly large inaccuracies in the values of the coordinates, so I need greater tolerance in the functions testing for equality and inequality.

This function could be formulated more succinctly, but I had some trouble getting it to work properly, so I'm leaving it in its more verbose form, in case I have to debug it some more.


---

\[ \text{[LDF 2002.10.27.] Revised this function. Now using } \texttt{Point } a \text{ and } \texttt{Point } q. \text{ Added } \texttt{factor} \text{ and using it as the argument to } \texttt{clean()} \text{ and for calculating } eps. \text{ Since this function is an operator, it's not possible to pass } \texttt{factor} \text{ as an argument, unfortunately. Using } \texttt{clean(factor)} \text{ makes it possible to compare the coordinates with 0 directly rather than using } \text{fabs()} \text{ and } eps. \text{ Also, } \texttt{operator\texttt{\equiv}()} \text{ now uses } \texttt{world\textunderscore coordinates} \text{ directly rather than } get_x(), get_y(), \text{ and } get_z(). \]

\[ \text{[LDF 2003.07.09.] Made this function } \texttt{non\texttt{-const}}, \text{ and added } \texttt{const} \text{ version below.} \]

\[ \text{[Declare Point functions 329) +\equiv} \]

\[
\text{bool operator\equiv(Point p);}\]
561.  
(Define Point functions 330) +≡

```cpp
bool Point::operator==(Point p) { bool DEBUG = false;  /* true */
    if (DEBUG) cout << "Entering Point::operator==() \n" << flush;
    unsigned short factor = 10;  /* LDF 2002.10.27. Added. */
    clean(factor);
    p.clean(factor);
    real eps = epsilon() * factor;  /* This currently makes eps ≡ .0001. */
    real tx = world_coordinates[0];
    real ty = world_coordinates[1];
    real tz = world_coordinates[2];
    real px = p.world_coordinates[0];
    real py = p.world_coordinates[1];
    real pz = p.world_coordinates[2];
```

562.  Points are frequently compared to INVALID_POINT, so it’s best to suppress debugging output for these comparisons, because they’re probably not the ones we’re interested in.

(Define Point functions 330) +≡

```cpp
if (tx ≡ INVALID_REAL ∨ ty ≡ INVALID_REAL ∨ tz ≡ INVALID_REAL ∨ px ≡ INVALID_REAL ∨ py ≡ INVALID_REAL ∨ pz ≡ INVALID_REAL) DEBUG = false;
```

563.  Debugging output.

(Define Point functions 330) +≡

```cpp
if (DEBUG) {
    cout << "tx=\n" << tx << endl << flush;
    cout << "ty=\n" << ty << endl << flush;
    cout << "tz=\n" << tz << endl << flush;
    cout << "px=\n" << px << endl << flush;
    cout << "py=\n" << py << endl << flush;
    cout << "pz=\n" << pz << endl << flush;
}
```

564.  Check whether the coordinates of both Points are all 0.

[Log] Now that clean(10) and p.clean(10) are called above, it’s no longer necessary to compare the absolute values of the coordinates. I can just compare them with 0 instead.

(Define Point functions 330) +≡

```cpp
if (tx ≡ 0 ∧ px ≡ 0 ∧ ty ≡ 0 ∧ py ≡ 0 ∧ tz ≡ 0 ∧ pz ≡ 0) {
    if (DEBUG) cout << "All coordinates are 0, returning true. \n";
    return true;
}
```
Get the signs of the coordinates.

[2002 10.27] As in the previous section, changed so that the coordinates are compared with 0, instead of using fabs() and eps.

```c
// Define Point functions 330
signed short t_x_sign;
signed short t_y_sign;
signed short t_z_sign;
signed short p_x_sign;
signed short p_y_sign;
signed short p_z_sign;
if (t_x ≡ 0) t_x_sign = 0;
else if (t_x < 0) t_x_sign = -1;
else t_x_sign = 1;
if (t_y ≡ 0) t_y_sign = 0;
else if (t_y < 0) t_y_sign = -1;
else t_y_sign = 1;
if (t_z ≡ 0) t_z_sign = 0;
else if (t_z < 0) t_z_sign = -1;
else t_z_sign = 1;
if (p_x ≡ 0) p_x_sign = 0;
else if (p_x < 0) p_x_sign = -1;
else p_x_sign = 1;
if (p_y ≡ 0) p_y_sign = 0;
else if (p_y < 0) p_y_sign = -1;
else p_y_sign = 1;
if (p_z ≡ 0) p_z_sign = 0;
else if (p_z < 0) p_z_sign = -1;
else p_z_sign = 1;
if (DEBUG) {
    cout << "t_x_sign=" << t_x_sign << endl << flush;
    cout << "t_y_sign=" << t_y_sign << endl << flush;
    cout << "t_z_sign=" << t_z_sign << endl << flush;
    cout << "p_x_sign=" << p_x_sign << endl << flush;
    cout << "p_y_sign=" << p_y_sign << endl << flush;
    cout << "p_z_sign=" << p_z_sign << endl << flush;
}
if ((t_x_sign ≠ p_x_sign) ∨ (t_y_sign ≠ p_y_sign) ∨ (t_z_sign ≠ p_z_sign)) {
    if (DEBUG)
        cout << "At least one coordinate pair has signs that differ." << "Returning false.
        return false;
    ```
566. Get the difference between each pair of \( x, y, \) and \( z \)-coordinates.

\[
\begin{align*}
  \text{Define Point functions } & \quad + \equiv \\
  t_x &= \text{fabs}(t_x) ; \\
  t_y &= \text{fabs}(t_y) ; \\
  t_z &= \text{fabs}(t_z) ; \\
  p_x &= \text{fabs}(p_x) ; \\
  p_y &= \text{fabs}(p_y) ; \\
  p_z &= \text{fabs}(p_z) ; \\
  \text{real delta}_x &= \text{fabs}(t_x - p_x) ; \\
  \text{real delta}_y &= \text{fabs}(t_y - p_y) ; \\
  \text{real delta}_z &= \text{fabs}(t_z - p_z) ; \\
  \text{if } (\text{DEBUG}) \{ \\
    \text{cout } \ll \text{"delta}_x==0\text{" } \ll \text{delta}_x \ll \text{endl } \ll \text{flush} ; \\
    \text{cout } \ll \text{"delta}_y==0\text{" } \ll \text{delta}_y \ll \text{endl } \ll \text{flush} ; \\
    \text{cout } \ll \text{"delta}_z==0\text{" } \ll \text{delta}_z \ll \text{endl } \ll \text{flush} ; \\
  \} \\
  \text{bool } r ; \quad /* \text{LDF 2002.10.27.} \text{ The return value. It’s only needed for the sake of the debugging code.} \\
  \text{Otherwise, this function could just return the result of the following expression. } */ \\
  r &= (\text{delta}_x < \text{eps } \land \text{delta}_y < \text{eps } \land \text{delta}_z < \text{eps}) ; \\
  \text{if } (\text{DEBUG}) \text{ cout } \ll \text{"r==0\text{" } } \ll r \ll \text{endl } \ll \text{flush} ; \\
  \text{if } (\text{DEBUG}) \text{ cout } \ll \text{"Exiting Point::operator()==()\text{" } } \ll \text{"n\text{" } } \ll \text{flush} ; \\
  \text{return } r ; \} \\
\end{align*}
\]

567. **\text{const version}.** \text{[LDF 2003.07.09.]} \text{This function merely copies } *this \text{ and calls the non-\text{const} version on the copy. Here, } p \text{ can be a } \text{const Point \&}, \text{ because this function does nothing but pass it to non-\text{const} version, where it is passed by value.}

\[
\begin{align*}
  \text{Log} \\
  \text{[LDF 2003.07.09.]} \text{ Added this version. Made the original version non-\text{const}.} \\
\end{align*}
\]

\[
\begin{align*}
  \text{Declare Point functions } & \quad + \equiv \\
  \text{bool operator\&\&(const Point \&p) const} ; \\
\end{align*}
\]

568.

\[
\begin{align*}
  \text{Define Point functions } & \quad + \equiv \\
  \text{bool Point::operator\&\&(const Point \&p) const} \\
  \{ \\
    \text{Point copy \*(this)} ; \\
    \text{return } (\text{copy } \equiv p) ; \\
  \} \\
\end{align*}
\]

569. **Inequality.**

\[
\begin{align*}
  \text{Declare Point functions } & \quad + \equiv \\
  \text{bool operator\&\!(const Point \&p) const} ; \\
\end{align*}
\]
570. (Define Point functions 330) +≡

\begin{verbatim}
  bool Point::operator==(const Point &p) const
  {
    return !(*this == p);
  }
\end{verbatim}

571. Intersection.

[LDF 2002.10.27.] intersection_point() takes four Point arguments. It assumes that the first and second represent one line segment and the third and fourth another. It calculates the intersection point of the two lines, if any, and returns a bool_point. If the intersection point exists and is on both line segments, the bool is true and the Point is the intersection point. If the intersection point exists, but not on both line segments, the bool is false and the Point is the intersection point. If no intersection point exists, i.e., the line segments are congruent or parallel, then the bool is false and the Point is INVALID_POINT.

LDF Note? Note that intersection_point() had to be defined as a static member function, because Path::intersection_point() was not able to resolve the call of this version, when it was not callable as “Point::intersection_point()”. I got a compiler error, because the call of intersection_point() inside Path::intersection_point() with four references to Points as arguments was deemed to have “too many arguments” and the references to Point couldn’t be converted to const references to Path.

572. Vector version. [LDF 2003.06.29.] Defined in lines.web, because Line is an incomplete type here.


!! It may be necessary or desirable to add try...catch blocks where calculations are performed below, just in case overflow occurs.

!! Under Linux, both this function and the version of intersection_points() using traces have failed in the same cases, which involved coplanar lines which had been rotated about the z-axis or the y and z-axes. I suspect it has to do with the routines for sine and cosine, since I’ve had trouble with rotation in the constructors for Polyhedra. [LDF 2003.06.29.]

Log

[LDF 2002.04.10.] Added this function. It replaces the old version, below.

[LDF 2002.04.12.] Removed the definition of this function to lines.web, because it requires the use of Lines, and Line is an incomplete type here.

[LDF 2003.06.29.] Started using this version again. Bug fixes I’ve made elsewhere seem to have made it function.

(Declare Point functions 329) +≡

\begin{verbatim}
  static bool_point intersection_point(const Point &pp0, const Point &pp1, const Point &qq0, const Point &qq1);
\end{verbatim}

573. Trace version. This function finds the intersection point of two lines by finding the intersection points of the traces of the lines on the major planes. I originally wrote it, because the vector version didn’t work. Bug fixes elsewhere seem to have fixed the problem, so this version isn’t really needed anymore. [LDF 2003.06.29.]

The bool argument trace serves only to distinguish this function from the vector version. It doesn’t matter whether it’s true or false. [LDF 2003.06.29.]

Log

[LDF 2002.10.27.] Changed the const Point & arguments to Point, because I had to copy them anyway in order to call apply_transform() on them.
[LDF 2003:06:29.] Added the bool argument trace, in order to be able to use both the vector and trace versions. Previously, the vector version didn’t work, and was commented-out. Now, bug fixes elsewhere seem to have made the vector version work. Both versions, however, failed under Linux. See the TeX section for the vector version, above, for more information.

(Declare Point functions 329) +≡
static bool_point intersection_point(Point p0, Point p1, Point q0, Point q1, const bool trace);

574.
(Define Point functions 330) +≡
bool_point Point::intersection_point(Point p0, Point p1, Point q0, Point q1, const bool trace){
  bool DEBUG = false; /* true */
  if (DEBUG) {
    cout << "Entering Point::intersection_point().\n" << flush;
    p0 . show("p0");
    p1 . show("p1");
    q0 . show("q0");
    q1 . show("q1");
  }
  bool bp; /* Return value. */
  if (DEBUG) cout << "Error\after\here\0.\n" << flush;

575. Apply the transformations, so we have the correct values for x, y, and z in each of the Points. Then assign them to variables.

(Define Point functions 330) +≡
p0 . apply_transform();
p1 . apply_transform();
q0 . apply_transform();
q1 . apply_transform();
real p0.x = p0 . world_coordinates[0];
real p0.y = p0 . world_coordinates[1];
real p0.z = p0 . world_coordinates[2];
real p1.x = p1 . world_coordinates[0];
real p1.y = p1 . world_coordinates[1];
real p1.z = p1 . world_coordinates[2];
real q0.x = q0 . world_coordinates[0];
real q0.y = q0 . world_coordinates[1];
real q0.z = q0 . world_coordinates[2];
real q1.x = q1 . world_coordinates[0];
real q1.y = q1 . world_coordinates[1];
real q1.z = q1 . world_coordinates[2];
if (DEBUG) cout << "Error\after\here\1.\n" << flush;
576. Get deltas.
\begin{verbatim}
(Define Point functions 330) +≡
  real delta_xp = p1.x - p0.x;
  real delta_yq = p1.y - p0.y;
  real delta_zp = p1.z - p0.z;
  real delta_wq = q1.w - q0.w;
  real delta_yq = q1.y - q0.y;
  real delta_wq = q1.w - q0.w;
  if (DEBUG) cout << "Error after here 2 \n" << flush;
\end{verbatim}

577. Slopes for line \( \overrightarrow{pq} \).
\begin{verbatim}
(Define Point functions 330) +≡
  real slope_{p.x=q} = (delta_{q.x} ≠ 0) ? delta_{q.x} / delta_{x.p} : INVALID_REAL;
  real slope_{p.y=q} = (delta_{q.y} ≠ 0) ? delta_{q.y} / delta_{y.p} : INVALID_REAL;
  real slope_{p.z=q} = (delta_{q.z} ≠ 0) ? delta_{q.z} / delta_{z.p} : INVALID_REAL;
  if (DEBUG) cout << "Error after here 3 \n" << flush;
\end{verbatim}

578. Slopes for line \( \overrightarrow{pq} \).
\begin{verbatim}
(Define Point functions 330) +≡
  real slope_{q.x=p} = (delta_{p.x} ≠ 0) ? delta_{p.x} / delta_{x.q} : INVALID_REAL;
  real slope_{q.y=p} = (delta_{p.y} ≠ 0) ? delta_{p.y} / delta_{y.q} : INVALID_REAL;
  real slope_{q.z=p} = (delta_{p.z} ≠ 0) ? delta_{p.z} / delta_{z.q} : INVALID_REAL;
  if (DEBUG) cout << "Error after here 4 \n" << flush;
\end{verbatim}

579. The traces on the x-y plane, \( x.i, y.i, z.i, y.int.p \), and \( y.int.q \) are set to INVALID_REAL so that I can test for whether the routines below succeed in setting them correctly.
\begin{verbatim}
(Define Point functions 330) +≡
  real x.i = INVALID_REAL;  /* x-coordinate of the intersection point. */
  real y.i = INVALID_REAL;  /* y-coordinate of the intersection point. */
  real z.i = INVALID_REAL;  /* z-coordinate of the intersection point. */
  real y.int.p = INVALID_REAL;  /* y-intercept of \( \overrightarrow{pq} \). */
  real y.int.q = INVALID_REAL;  /* y-intercept of \( \overrightarrow{pq} \). */
  if (DEBUG) cout << "Error after here 5 \n" << flush;
  if (slope_{p.x=q} ≠ INVALID_REAL)  /* \( \Delta x_p ≠ 0 \) */
  {      
    y.int.p = p0.y - (slope_{p.x=q} * p0.x);
  }
  if (slope_{q.x=p} ≠ INVALID_REAL)  /* \( \Delta x_q ≠ 0 \) */
  {      
    y.int.q = q0.y - (slope_{q.x=p} * q0.x);
  }
  if (DEBUG) cout << "Error after here 6 \n" << flush;
\end{verbatim}
580.
• If both of the traces of \( \overrightarrow{p} \) and \( \overrightarrow{q} \) in the x-y plane are parallel to the y-axis (i.e., \( \Delta x = 0 \)), we test whether \( p_y = q_y \). If they are, then we set \( x_j \) to that value. If they’re not, the lines don’t intersect, so we return INVALID_POINT.
• If the trace of \( \overrightarrow{p} \) or the trace of \( \overrightarrow{q} \) in the x-y plane is parallel to the y-axis, we set \( x_j \) to its x-value, because in this case, the intersection point must have this x-value.
• If \( \Delta x_p \neq 0 \) and \( \Delta x_q \neq 0 \), we derive \( x_j \) using the slope and y-intercept of the lines.

```cpp
{Define Point functions 330} +=
if (y_int_p \equiv INVALID_REAL \land y_int_q \equiv INVALID_REAL)
  /* \( \overrightarrow{p}_z \) and \( \overrightarrow{q}_z \) are both parallel to the z-axis. */
  
  if (DEBUG) cout << "Error after here\_7.\n" << flush;
  if (p[0] \equiv q[0])        /* They have the same value for x. */
  
  if (DEBUG) cout << "Error after here\_8.\n" << flush;
  if (DEBUG) cout << "Traces on x-y plane are coincident.\n" << flush;
  x_j = p[0]_z;
  real y_int_p_x = INVALID_REAL;
  real y_int_q_x = INVALID_REAL;
  
  if (slope_p-x-y \neq INVALID_REAL) y_int_p_x = p[0]_y - slope_p-x-y \* p[0]_z;
  if (slope_q-x-y \neq INVALID_REAL) y_int_q_x = q[0]_y - slope_q-x-y \* q[0]_z;
  if (DEBUG) cout << "Error after here\_9.\n" << flush;
  if (slope_p-x-y \equiv INVALID_REAL \land slope_q-x-y \equiv INVALID_REAL) {
    if (DEBUG) cout << "Both traces on x-y plane are vertical\n" << flush;
    if (DEBUG) cout << "Error after here\_10.\n" << flush;
    if (p[0]_z \equiv q[0]_z) cerr << "Lines are coincident.\n";
    else cerr << "Lines do not intersect.\n";
    cerr << "Returning INVALID_BOOL_POINT.\n" << flush;
    return INVALID_BOOL_POINT;
  }
else if (slope_p-x-y \equiv INVALID_REAL) {
  if (DEBUG) cout << "The p-trace is vertical\n" << flush;
  if (DEBUG) cout << "Error after here\_11.\n" << flush;
  z_i = p[0]_z;
  y_i = z_i \* slope_p-x-y + y_int_p_x;
  if (DEBUG) {
    cout << "x_\_i=u" << x_i << endl << flush;
    cout << "y_\_i=u" << y_i << endl << flush;
    cout << "z_\_i=u" << z_i << endl << flush;
    cout << "slope_q-x-y=u" << slope_q-x-y << endl << flush;
    cout << "y_int_q_x=u" << y_int_q_x << endl << flush;
  }
else if (slope_q-x-y \equiv INVALID_REAL) {
  if (DEBUG) cout << "The q-trace is vertical\n" << flush;
  if (DEBUG) cout << "Error after here\_12.\n" << flush;
  z_i = q[0]_z;
  y_i = z_i \* slope_q-x-y + y_int_q_x;
  if (DEBUG) {
    cout << "x_\_i=u" << x_i << endl << flush;
    cout << "y_\_i=u" << y_i << endl << flush;
  }
    
    
}
2D Linear Interpolation

```cpp
    cout << "z_i==\n" << z_i << endl << flush;
    cout << "slope_p_x_y==\n" << slope_p_x_y << endl << flush;
    cout << "y_int_p_x==\n" << y_int_p_x << endl << flush;

    z_i = (y_int+ x_int_p_y)/(slope_p_x_y - slope_q_x_y);
    y_i = slope_p_x_y * x_i + y_int_p_x;
    if (DEBUG) { cout << "x_i==\n" << x_i << endl << flush;
    cout << "y_i==\n" << y_i << endl << flush;
    cout << "z_i==\n" << z_i << endl << flush;
    cout << "slope_p_x_y==\n" << slope_p_x_y << endl << flush;
    cout << "y_int_p_x==\n" << y_int_p_x << endl << flush;
    cout << "slope_q_x_y==\n" << slope_q_x_y << endl << flush;
    cout << "y_int_q_x==\n" << y_int_q_x << endl << flush;
    }
    }  
else  
    
    if (DEBUG) cout << "Error after here\n" << flush;
    cerr << "Line\ndoes not intersect\n" << flush;
    cerr << "x==p0.x && y==p0.y && \n     x==p1.x && y==p1.y && \n     x==q0.x && y==q0.y && \n     x==q1.x && y==q1.y \nReturning INVALID_BOOL_POINT\n" << flush;
    return INVALID_BOOL_POINT;
}
else if (y_int.p == INVALID_REAL)  
    /* \tilde{p}_{xy} is parallel to the x-axis. */
{
    if (DEBUG) cout << "Error after here\n" << flush;
    x_i = p0.x;
    y_i = slope_q_x_y * x_i + y_int_q;
}
else if (y_int.q == INVALID_REAL)  
    /* \tilde{q}_{xy} is parallel to the y-axis. */
{
    if (DEBUG) cout << "Error after here\n" << flush;
    x_i = q0.x;
    y_i = slope_p_x_y * x_i + y_int_p;
}
```
581. [LDF 2002.11.12.] !! BUG: Occurred when I tried to find an intersection of two lines in the x-z plane. This code shouldn’t be reached. Rotating the objects 90° around the x-axis, putting them into the x-y plane, fixed the problem. Obviously, the case that the objects are in the x-z plane already isn’t handled properly.

(Define Point functions 330) +≡

else /* Neither \(q_x\) nor \(q_y\) is parallel to the y-axis. */

{ if (DEBUG) /* [LDF 2002.11.12.] Start working on finding bug here. */
    { count ≜ "Error after here.\n" ≜ flush;
      count ≜ "slope \(p.X\ y\) == u" ≜ slope \(p.x\ y\) ≜ endl ≜ flush;
      count ≜ "slope \(q.X\ y\) == u" ≜ slope \(q.x\ y\) ≜ endl ≜ flush;
    }
    if (slope \(p.x\ y\) ≠ slope \(q.x\ y\))
      { x̂ = (y \(\text{int}\) \(p\) − y \(\text{int}\) \(p\))/(slope \(p.x\ y\) − slope \(q.x\ y\));
        ŷ = slope \(p.x\ y\) * x̂ + y \(\text{int}\) \(p\);
      }
  }

582. The trace on the x-z plane. We don’t need to do this if we’ve calculated \(x̂\) \(\hat{\text{i}}\) above, in the case that the traces on the x-y plane are coincident.

(Define Point functions 330) +≡

if (x̂ \(\hat{\text{i}}\) ≜ INVALID-REAL) { if (DEBUG) count ≜ "Error after here.\n" ≜ flush;
    real z \(\text{int}\) \(p\) = INVALID-REAL; /* z-intercept of \(\text{dy}/\text{dx}\). */
    real z \(\text{int}\) \(q\) = INVALID-REAL; /* z-intercept of \(\text{dy}/\text{dx}\). */
    if (slope \(p.x\ y\) ≠ INVALID-REAL) /* Δx̂ ≠ 0 */
      { z \(\text{int}\) \(p\) = p \(\hat{\text{i}}\) − (slope \(p.x\ y\) * p \(\hat{\text{i}}\));
      }
    if (slope \(p.x\ y\) ≠ INVALID-REAL) /* Δx̂ ≠ 0 */
      { z \(\text{int}\) \(q\) = q \(\hat{\text{i}}\) − (slope \(q.x\ y\) * q \(\hat{\text{i}}\));
      }
    if (DEBUG) {
      count ≜ "z \(\text{int}\) \(p\) == u" ≜ z \(\text{int}\) \(p\) ≜ endl ≜ flush;
      count ≜ "z \(\text{int}\) \(q\) == u" ≜ z \(\text{int}\) \(q\) ≜ endl ≜ flush;
    }
  }

583. [LDF 2003.06.24.] \(x̂\) \(\hat{\text{i}}\) will be equal to INVALID-REAL, if the traces of the lines on the x-y plane were colinear.

Log

[LDF 2003.06.24.] Added this conditional.

(Define Point functions 330) +≡

if (x̂ \(\hat{\text{i}}\) ≜ INVALID-REAL ∧ (z \(\text{int}\) \(p\) ≜ INVALID-REAL ∨ z \(\text{int}\) \(q\) ≜ INVALID-REAL))
  { x̂ \(\hat{\text{i}}\) = (z \(\text{int}\) \(q\) − z \(\text{int}\) \(p\))/(slope \(p.x\ y\) − slope \(q.x\ y\));
    ŷ = p \(\hat{\text{i}}\) \(\hat{\text{j}}\);
  }
In the following case, \( \vec{p} \) and \( \vec{q} \) are both parallel to the z-axis. They have the same value for \( x \). We've set \( x, z \) above, so there's no need to do so here again.

```c
(Define Point functions 330) pl ≡ INVALID_REAL \land \( z_{int} \) ≡ INVALID_REAL) {
  if (DEBUG) cout ≡ "Error after here 19.\n" \( \ll \) flush;
  if (DEBUG) cout ≡ "Error after here 20.\n" \( \ll \) flush;
  return INVALID_BOOL_POINT;
}
else if (z_{int} \neq INVALID_REAL) {
  if (DEBUG) cout ≡ "Error after here 21.\n" \( \ll \) flush;
  z_{j} = p_{i} \_z;
}
else if (z_{int} \neq INVALID_REAL) {
  if (DEBUG) cout ≡ "Error after here 22.\n" \( \ll \) flush;
  z_{j} = q_{0} \_z;
}
else {
  if (DEBUG) cout ≡ "Error after here 23.\n" \( \ll \) flush;
  z_{j} = slope \_p \_x \_z * x_{j} + z_{int} \_p;
}
```
585. [LDF 2002.10.27.] If \( x_j, y_j, \) and \( z_j \) are all valid, set \( \text{bp}.pt \) using those values. Otherwise, set it to \( \text{INVALID\_POINT} \). If this Point is on both of the line segments \( \overline{p_0p_1} \) and \( \overline{q_0q_1} \), set \( \text{bp}.b \) to \text{true}, otherwise set it to \text{false}.

(Define Point functions 330) \( \equiv \)

```cpp
if (DEBUG) {
    cout << "Error after here, 24. \n" << flush;
    cout << \"x_.i\" = i \ " << x_j << endl << flush;
    cout >> \"y_.i\" = i \ " << y_j << endl << flush;
    cout >> \"z_.i\" = i \ " << z_j << endl << flush;
}
if (x_j \equiv \text{INVALID\_REAL} \lor y_j \equiv \text{INVALID\_REAL} \lor z_j \equiv \text{INVALID\_REAL}) {
    return \text{INVALID\_BOOL\_POINT};
}
else {
    \text{bp}.pt.set(x_j, y_j, z_j);
    if (DEBUG) \text{bp}.pt.show("bp.pt");
}
if (\text{bp}.pt.is_on_segment(p0, p1).first \equiv false \lor \text{bp}.pt.is_on_segment(q0, q1).first \equiv false) \text{bp}.b = false;
else \text{bp}.b = true;
if (DEBUG) {
    cout << \"bp.b = \" i \ " << bp.b << endl;
    cout >> \"bp.b = \" i \ " << bp.b << endl;
}
\text{return} \text{bp};
```

586. Picture functions. These functions must be defined here, because they use types which are incompletely defined in pictures.web.

587. Assignment operator. \( \equiv \) PORTING [LDF 2002.12.05.] Moved here from pictures.web. See that file for explanation.

(Define Picture functions 264) \( \equiv \)

```cpp
void \text{Picture}::operator=(\text{const Picture \&p})
{
    \text{clear}();
    transform = p.transform;
    for (\text{vector(Shape \*)::const_iterator iter} = p.shapes.begin(); \text{iter} \ne p.shapes.end(); \text{iter}++) {
        \text{shapes.push.back}((*iter).get_copy());
    }
    Label *lbl;
    for (\text{vector(Label \*)::const_iterator iter} = p.labels.begin(); \text{iter} \ne p.labels.end(); \text{iter}++) {
        \text{labels.push.back}(lbl);
        \text{labels.push}.back(lbl);
    }
}
```

588. Copy constructor. \( \equiv \) PORTING [LDF 2002.12.05.] Moved here from pictures.web. See that file for explanation.

(Define Picture functions 264) \( \equiv \)

```cpp
\text{Picture}::\text{Picture}(\text{const Picture \&p})
: \text{do_labels} (true) \{
```

[LDF 2002.10.29.] Made p const and fixed bugs that changed p (see below).
[LDF 2002.10.29.] BUG FIX: Now, p.transform is applied to *(shapes.back()), previously it was applied to **iter, which was not what I wanted.
[LDF 2002.10.29.] BUG FIX: Now, p.transform is applied to *(labels.back()–pt), previously it was applied to *((*iter)–pt), which is not what I wanted.
[LDF 2002.10.29.] BUG FIX: Now, p.transform.reset() is no longer called.

```cpp
// Define Picture functions 264) ++
void Picture::operator+=(const Picture &p)
{
    for (vector<Shape> *::const_iterator iter = shapes.begin(); iter != p.shapes.end(); iter++) {
        shapes.push_back((*iter).get_copy());
        /* [LDF 2002.10.29.] Normally, transform in a Picture is applied to its Shapes when it's output,
         * however, it must be done now for the copies of the Shapes from Picture p that are copied
         * onto *this, because p.transform is only known within p the Shapes don't "know" about it. */
        *(shapes.back()) *= p.transform;
    }
    for (vector<Label> *::const_iterator iter = labels.begin(); iter != p.labels.end(); iter++) {
        labels.push_back((*iter).get_copy());
        *(labels.back()–pt) *= p.transform;
    }
}
```

590. Clear Picture.

```cpp
// Define Picture functions 264) ++
void Picture::clear()
{
    bool DEBUG = false;  /* true */
    if (DEBUG) cout << "Entering Picture::clear().\n" << flush;
    if (shapes.size() < 0 && labels.size() < 0) return;
    transform.reset();
    for (vector<Shape> *::iterator iter = shapes.begin(); iter != shapes.end(); iter++) {
        (*iter).clear();
        delete (*iter);
    }
    for (vector<Label> *::iterator iter = labels.begin(); iter != labels.end(); iter++) {
        /* ?? I tried to use ~Label() here, but it didn't work. I got run-time errors having to do with
         * "Unaligned access pid=299273..." (didn't understand). This works, though. If I change the
         * definition of Label, I'll have to make corresponding changes here. */
        delete (*iter); pt;
        delete (*iter);
    }
    shapes.clear();
```
labels.clear();
if (DEBUG) cout << "Exiting Picture::clear().\n" << flush;
}

591. Output. The arguments:
• sort_value is used to determine how to sort the Shapes. The values to be used are found in namespace Sorting. Sorting::NO_SORT is used, if they shouldn't be sorted, because we will have already drawn them in the order we want them rendered. Sorting::MAX_Z is used for sorting them according to their maximum z-coordinate for "furthest-first" output. Sorting::MIN_Z is used for sorting them according to their maximum z-coordinate for "nearest-last" output. So far, no other types of sorting have been defined.

The simple painter's algorithm implemented here for surface hiding fails for Shapes where one Shape is partly in front of and partly behind another. For these cases, it will be necessary to find the intersection points and divide the Shapes into parts. TO DO: Implement a routine for dividing up Shapes. This will not be done soon!

do_warnings: Sometimes we'll want use the min_x_proj, max_x_proj, etc., arguments to cut off parts of the image, or we'll deliberately place the Focus where it won't be able to "see" part of the image. In these cases, it will be annoying to see the warnings.

Log

[LDF 2002.09.21.] Added the arguments do_sort and do_warnings.
[LDF 2003.06.16.] Changed bool do_sort to const unsigned short sort_value. About to add namespace Sorting with constants for different ways of sorting, i.e., "no sort", "nearest-last", or "furthest-first".

592. Focus argument.

```cpp
#include Picture::output(const Focus &f, const unsigned short proj, real factor, const
unsigned short sort_value, const bool do_warnings, const real min_x_proj, const real
max_x_proj, const real min_y_proj, const real max_y_proj, const real min_z_proj, const real
max_z_proj){ bool DEBUG = false; /* true */
using namespace Sorting;
if (DEBUG) {
  cout << "Entering Picture::output(const Focus &...).\n" << flush;
  cout << "min_x_proj=\"" << min_x_proj << endl << flush;
  cout << "max_x_proj=\"" << max_x_proj << endl << flush;
  cout << "min_y_proj=\"" << min_y_proj << endl << flush;
  cout << "max_y_proj=\"" << max_y_proj << endl << flush;
  cout << "min_z_proj=\"" << min_z_proj << endl << flush;
}
/* Check whether the vector shapes has anything in it. If it doesn't, return. */
if (shapes.size() <= 0 && labels.size() <= 0) {
  if (DEBUG) cout << "Picture is empty. Returning.\n" << flush;
  return;
}
593. [LDF 2002.09.17.] Some Shapes may consist of other Shapes, and not have an output() function of their own, so we must extract their contents recursively until we get to Shapes that have one. So far, only Point, Path, and Solid have output() functions, and all other Shapes reduce to Paths or SOLIDS.

[LDF 2002.09.17.] extract() checks that all of the Points contained in the Shape can be projected with the Focus that is being used. If any of them cannot be, then extract() returns an empty vector of Shapes. This means that any Shape must be entirely projectable; partial Shapes will not be output. problem, too.

---

Log

---

(Define Picture functions 264) +=

  vector (Shape *) v;
  vector (Shape *) elements;
  vector (Shape *) :: iterator iter;

  bool do_transform = !transform.is_identity();

  DEBUG = false; /* true */
  if (DEBUG) {
    if (do_transform) {
      cout << "Applying \n"
      transform.show("transform:");
    }
    else cout << "Not applying transform.\n"
  }
  for (iter = shapes.begin(); iter != shapes.end(); ++iter) {
    if (do_transform) {
      (**iter) *= transform;
    }
    v = (*iter)-extract(f, proj_factor);
    if (DEBUG && v.size() == 0)
      cerr << "WARNING! In Picture::output():\n" << extract() "\n" << empty_vector << "Continuing.\n" << flush;
    for (vector(Shape *) :: iterator i = v.begin(); i != v.end(); ++i) {
      elements.push_back(*i);
    }
  }
  DEBUG = false;
594. Set the extremes for the *Shape* and handle the error if it returns `false`. (LDF Undated)

Log

[LDF 2002.09.18.] Changed the error handling code below. Formerly, `get_minimum_z()` was invoked, but this is unnecessary, since `set_extremes()` returns `false` if something goes wrong with setting the extreme values for the *Shape*.

```c
{ Define Picture functions 264 } +\equiv

valarray<real> extremes(6, 0); for (iter = elements.begin(); iter != elements.end(); ++iter) {
  if (DEBUG) cout << "About to set extremes.\n" << flush;
  if (!(*++iter).set_extremes()) {
    cerr << "ERROR! In Picture::output():\n" << "set_extremes() returned false.\n" << "Suppressing output for this Shape.\n" << flush;
    (**iter).suppress_output();
  }
}
```

595. [LDF 2002.09.18.] Added this routine. It checks for whether the values in the `valarray<real>` `projective_extremes` in the *Shape* fall within the limits given by the `min_x_proj`, `max_x_proj`, `min_y_proj`, and `max_y_proj` arguments to this function (Picture::output()). (Note that `min_x_proj` and `max_x_proj` are currently not checked.) If they don’t, the *Shape* is removed from `elements`. Note that the projected z-coordinates are not currently checked, but they are used for ordering the Shapes for output (furthest away first).

```c
{ Define Picture functions 264 } +\equiv

extremes = (**iter).set_extremes();
if (DEBUG)  // [LDF 2002.09.21.] Show the extremes for this Shape. */
{
  for (int i = 0; i < 4; i++) {
    cout << "extremes[" << i << "]\n" << extremes[i] << ";\n";
  }
  cout << "extremes[0] < min_x_proj < u\n" << (extremes[0] < min_x_proj) << endl << flush;
  cout << "extremes[1] < max_x_proj < u\n" << (extremes[1] > max_x_proj) << endl << flush;
  cout << "extremes[3] < max_y_proj < u\n" << (extremes[3] > max_y_proj) << endl << flush;
    if (do_warnings == true) {
      cerr << "WARNING! In Picture::output():\n" << "Shape lies outside the limits for this\n" << invocation_of_output() << " Suppressing output for this Shape.\n" << flush;
    }
    (**iter).suppress_output();
  }
  // for */
}  // End of group. */
```
596. [LDF 2003.05.16.] Sorting can be performed in different ways, depending on the sort_value argument. This is explained in (Define comparison classes 498).

[LDF 2002.09.18.] It's necessary to make sure that sorting is only performed if elements is non-empty. It could be empty now, if the error handling code above has removed all of the elements because set_extremes() returned false for all of them. We can't just return, because there might still be Labels on the Picture.

(Define Picture functions 264) \[\equiv\]
if (elements.size() > 0) {
    if (sort_value \equiv \text{MIN}_Z) sort(elements.begin(), elements.end(), \text{Compare}_{\text{minimum}_Z}());
    else if (sort_value \equiv \text{MAX}_Z) sort(elements.begin(), elements.end(), \text{Compare}_{\text{maximum}_Z}());
    else if (sort_value \equiv \text{MEAN}_Z) sort(elements.begin(), elements.end(), \text{Compare}_{\text{mean}_Z}());
    if (DEBUG) {
        if (sort_value \equiv \text{MIN}_Z) cout << " \text{\textquotedblleft}_u***u\text{MIN}_Z_u***\text{n}\n";
        else if (sort_value \equiv \text{MAX}_Z) cout << " \text{\textquotedblleft}_u***u\text{MAX}_Z_u***\text{n}\n";
        else if (sort_value \equiv \text{MEAN}_Z) cout << " \text{\textquotedblleft}_u***u\text{MEAN}_Z_u***\text{n}\n";
        for (iter = elements.begin(); iter \neq elements.end(); ++iter) {
            cout << "\text{\textquotedblleft}_u***u\text{Result}_u***\text{n}\n"; ++iter)
        }
        cout << "\text{\textquotedblright}_u***u\text{Sort}_u***\text{n}\n" << flush;
    }
    for (iter = elements.begin(); iter \neq elements.end(); ++iter) {
        (\ast\astiter).\text{output}();
        (**iter).\text{unsuspend_output}(); /* [LDF 2002.09.18.] With a different Focus or different limiting values for the projection, this Shape might be projectable, so we reset do_output to true. If it can't be projected the next time, suppress_output() will be invoked again. */
    }
}

597. Output the labels. LDF Undated. It is necessary to output the labels last because they might otherwise be drawn over by fill() or filldraw() commands.

[LDF 2002.09.17.] I'm not bothering to sort the labels so that the ones behind can be hidden by the ones in front. Labels should all be visible and are not put into perspective, so they shouldn't overlap one another.

[LDF 2002.04.25.] Added following conditional. Sometimes it's irritating to have the labels when a Picture is copied and transformed, and both the original and the transformed versions are output.

(Define Picture functions 264) \[\equiv\]
if (do_labels \equiv \text{true}) {
    for (vector(Label \ast): \text{iterator} = labels.begin(); i \neq labels.end(); \text{++i}) {
        \text{/* [LDF 2002.09.17.] Simplified the following code. Formerly, there was a conditional here that chose which version of Label::output() to call. I've removed the version without a Transform argument and invoke Transform::is_identity() in Label::output(const Focus &\ast, const Transform &\ast. [LDF 2002.04.25.] This applies transform to the Point Label::pt. */}
        (**i).output(f, proj, factor, transform);
    }
    \text{/* [LDF 2002.04.25.] Added following line. This fixes a bug. If I don't reset transform to identity, it will be applied again each time I output a Picture, which is not what I want. */}
    transform.reset();
    if (DEBUG) cout << "Exiting\text{Picture::output(const Focus&\ast)} \text{n}\n" << flush;
}
}
598. No Focus argument.

(Define Picture functions 264) +≡
void Picture::output(const unsigned short proj, real factor, const unsigned short
  sort_value, const bool do_warnings, const real min_x_proj, const real max_x_proj,
  const real min_y_proj, const real max_y_proj, const real min_z_proj, const real max_z_proj)
{
  output(default_focus, proj, factor, sort_value, do_warnings, min_x_proj, max_x_proj,
         min_y_proj, max_y_proj, min_z_proj, max_z_proj);
}

599. Focus.

600. Focus class definition. [LDF 2002.09.18.] Made Focus a class (it was formerly a struct). Added
c char axis data member. It indicates the axis to which position should be transformed when it’s put in
standard position. This determines which plane the image is projected onto. If axis == 'x' (the default),
the image is projected onto the x-y plane, if axis == 'y', the z-y plane, if 'z', the x-z plane.

At this time, I’m not adding the routines that will do this, which will entail changing transform and
possibly persp, I’m just adding axis. I must also add a function for changing axis without changing any of
the other data members. TO DO: Add these routines!!

[LDF 2002.09.14.] Added Transform persp. It’s needed because I need to get the z value of the
world coordinates after the transformation that puts the Focus into standard position, but before the
perspective transformation is performed. This z value can be used in an algorithm for surfaces hiding. If
this were the case, I could combine the transformations, because matrix multiplication is associative (it
is not, however, commutative, except with special matrices).

[LDF 2002.09.11.] Added class Focus and the following constructors: The default constructor with no
arguments, the one with two Point arguments and a real argument, and the one with seven real arguments.

format Focus int
(Define class Focus 600) ≡
class Focus
{
  Point position;
  Point direction;
  Point up;
  real distance;
  real angle;
  char axis;
  Transform transform;
  Transform persp;

  public: { Declare Focus functions 602 }
};

This code is used in sections 633 and 634.

601. Constructors and setting functions. [LDF 2002.09.22.] TO DO: Check magnitude of direction —
position and make sure it’s non-zero!!

[LDF 2002.10.13.] The effect of using an angle ≠ 0 is similar to that of rotating a camera about an axis
through its aperture and perpendicular to the surface of the lens. Because this is possible, it is necessary to
indicate the upward direction of a projection. The Point up does this. It is determined in the constructors
and setting functions by the vector direction - position and angle. up is first set to (0, 1, 0) if axis == 'x' or
axis == 'z', or (1, 0, 0) if axis == 'y'. If angle ≠ 0, up and transform are then rotated by - angle. Then, up
is transformed by the inverse of transform, in order to put it in the correct location with respect to position.
This location is “above” position by definition.
[LDF 2002.10.13.] Changed all of the constructors and setting functions except for the default constructor and the first non-default constructor. Now, all the others use the latter to create a Focus locally and use Focus::operator=(), which I’ve defined today, to assign to this. This eases maintenance and cuts down on the potential for error through inconsistencies in the different constructors and setting functions.

602. Default constructor. (No arguments).

(Declare Focus functions 602) ≡

    Focus()

{} See also sections 604, 606, 609, 611, 613, 615, 617, 620, 621, 622, 623, 624, 625, 627, and 628.

This code is used in section 600.

603. real arguments. The first three real arguments are for the coordinates of the center of projection (the focus in the narrowest sense) (position), the fourth through the sixth are for the coordinates of the direction of view (direction), dist is for the distance of position to the projection plane (distance), ang is for the angle of rotation around the axis \( \vec{p} \) where \( p \) stands for position and \( d \) for direction (angle), and the char argument ax indicates the axis with which \( \vec{p} \) is to be aligned, and around which \( \vec{u} \) is to be rotated (axis).

[LDF 2003.07.04.] Now calling persp.set_element() instead of accessing the elements of persp directly. The latter is no longer possible, because Focus is no longer a friend of Transform.

604. Constructor.

(Declare Focus functions 602) +≡

    Focus(const real pos_x, const real pos_y, const real pos_z, const real dir_x, const real dir_y, const real dir_z, const real dist, const real ang = 0, char ax = 'z');
605.

```c
(Define Focus functions 605 )

Focus(const real pos_x, const real pos_y, const real pos_z, const real dir_x, const real dir_y, const real dir_z, const real dist, const real ang, char ax)

  : distance(dist), angle(ang), axis(ax)
  
  bool DEBUG = false;  /* true */
  if (DEBUG) cout << "Entering Focus() with real arguments. \n" << flush;
  axis = tolower(axis);
  if (axis != 'x' && axis != 'y' && axis != 'z') {
    cerr << "WARNING! userFocus() with real arguments: \n" << "axis argument has invalid value: \n" << axis << "Using 'z' \n" << flush;
    axis = 'z';
  }
  position.set(pos_x, pos_y, pos_z);
  direction.set(dir_x, dir_y, dir_z);
  
  if (0)
    transform.reset();
  /* [LDF 2002.12.10] This doesn't seem to be necessary. I believe I added it while debugging. */
  
  if (DEBUG) {
    transform.show("transform before alignment.");
  }
  transform.align_with_axis(position, direction, axis);
  if (DEBUG) {
    transform.show("transform after alignment.");
    cout << "Enter<RETURN> to continue. \n" << flush;
  }
  Transform unalign_up = transform.inverse();  /* Use the positive y-axis for the "up" direction,
phantom{0} if axis = 'x' or 'z', and the positive x-axis if axis = 'y'. */
  if (axis != 'z' || axis != 'x') up.set(0, 1, 0);
  else up.set(1, 0, 0);
  if (angle != 0) {
    if (axis != 'z') up *= transform.rotate(0, 0, -angle);
    else if (axis != 'x') up *= transform.rotate(-angle);
    else if (axis != 'y') up *= transform.rotate(0, -angle);
    else {
      cerr << "ERROR! userFocus::Focus() with axis has invalid value: \n" << axis << " endl << "Rotating around z-axis. \n" << "Enter<RETURN> to try to continue. \n" << flush;
      up *= transform.rotate(0, 0, -angle);
    }
  }
  if (DEBUG) up.show("up after rotation");
  }
  up *= unalign_up;
  up.apply_transform();
  transform.shift(0, 0, -distance);
  persp.set_element(2, 2, 0);
  persp.set_element(2, 3, 1 / distance);
  if (DEBUG) cout << "Exiting Focus() with real arguments. \n" << flush;

See also sections 607, 610, 612, 614, 616, 618, 626, and 629.
```
\begin{verbatim}
(Declare Focus functions 602) +≡
  void set(const real pos_x, const real pos_y, const real pos_z, const real dir_x, const real
  dir_y, const real dir_z, const real dist, const real ang = 0.0, char ax = 'z');
\end{verbatim}

607.
\begin{verbatim}
(Define Focus functions 605) +≡
  void Focus::set(const real pos_x, const real pos_y, const real pos_z, const real dir_x, const real
  dir_y, const real dir_z, const real dist, const real ang, char ax)
  {
    Focus f(pos.x, pos.y, pos.z, dir.x, dir.y, dir.z, dist, ang, ax);
    *this = f;
  }
\end{verbatim}

608. Point arguments.

609. Constructor.
\begin{verbatim}
(Declare Focus functions 602) +≡
  Focus(const Point &pos, const Point &dir, const real dist, const real ang = 0.0, char ax = 'z');
\end{verbatim}

610.
\begin{verbatim}
(Define Focus functions 605) +≡
  Focus::Focus(const Point &pos, const Point &dir, const real dist, const real ang, char ax)
  {
    Focus f(pos.get_x(), pos.get_y(), pos.get_z(), dir.get_x(), dir.get_y(), dir.get_z(), dist, ang, ax);
    *this = f;
  }
\end{verbatim}

611. Setting function. [LDF 2002.09.17.] Added this function.
\begin{verbatim}
(Declare Focus functions 602) +≡
  void set(const Point &pos, const Point &dir, const real dist, const real ang = 0.0, char ax = 'z');
\end{verbatim}

612.
\begin{verbatim}
(Define Focus functions 605) +≡
  void Focus::set(const Point &pos, const Point &dir, const real dist, const real ang, char ax)
  {
    Focus f(pos.get_x(), pos.get_y(), pos.get_z(), dir.get_x(), dir.get_y(), dir.get_z(), dist, ang, ax);
    *this = f;
  }
\end{verbatim}

613. Assignment. [LDF 2002.10.13.] Added this function. Now using it in all but the first of the non-default constructors. This saves on duplicating code and reduces the probability of bugs that might arise from inconsistencies among the constructors and setting functions.
\begin{verbatim}
(Declare Focus functions 602) +≡
  const Focus &operator=(const Focus &);
\end{verbatim}
614.  
(Define **Focus** functions 605) +≡

```cpp
const Focus &Focus::operator=(const Focus &f) 
{
    if (this == &f) /* [LDF 2002.10.13] Prevent self-assignment. */
        return *this;
    position = f.position;
    direction = f.direction;
    up = f.up;
    distance = f.distance;
    angle = f.angle;
    axis = f.axis;
    transform = f.transform;
    persp = f.persp;
    return *this;
}
```


(Declare **Focus** functions 602) +≡

```cpp
void reset_angle(const real ang);
```
616. (Define Focus functions 605 ) +≡
void Focus::reset_angle(const real ang)
{
    angle = ang;
    transform.reset();
    persp.reset();
    transform.align_with_axis(position, direction, axis);
    Transform unalign_up = transform.inverse(); /* Use the positive y-axis for the “up” direction, */
    if (axis == 'x' or 'z', and the positive x-axis if axis == 'y'. */
    if (axis == 'z' or axis == 'x') up.set(0,1,0);
    else up.set(1,0,0);
    if (angle != 0) {
        if (axis == 'z') up *= transform.rotate(0,0,-angle);
        else if (axis == 'x') up *= transform.rotate(-angle);
        else if (axis == 'y') up *= transform.rotate(0,-angle);
        else {
            cerr << "ERROR in Focus::Focus():\n" << "This can’t happen!\n" << "axis has invalid value: u\" << axis << endl << "Rotating around z-axis.\n" << "Enter <RETURN> to try to continue.\n" << "\nup *= transform.rotate(0,0,-angle);\n        }
    }
    up *= unalign_up;
    up.apply_transform();
    transform.shift(0,0,-distance);
    persp.set_element(2,2,0);
    persp.set_element(2,3,1/distance);
}

617. Show.

Log

LDF 2002.09.17 Added this function.
LDF 2003.07.09 Made the arguments const.

(Declare Focus functions 602 ) +≡
void show(const string text_str = "Focus:", const bool show_transforms = false) const;


618. \(\text{Define Focus functions 605 } +\equiv\)
void Focus::show(const string text_str, const bool show_transforms) const
{
    cout \ll text_str \ll endl;
    position.show("position: ");
    direction.show("direction: ");
    up.show("up: ");
    cout \ll "distance_u\;=\;u" \ll distance \ll "axis_u\;=\;u" \ll axis \ll endl \ll flush;
    if (show_transforms \equiv true) {
        transform.show("transform: ");
        persp.show("persp: ");
    }
    return;
}

619. Returning elements and information. [LDF 2002.09.18.] Added this section. The functions in
this section are now necessary, since I've made Focus a class (it was formerly a struct), and the data
members private.

620. Get position.

| Log | [LDF 2002.09.18.] Added this function. |

(Declare Focus functions 602 ) \equiv
inline const Point &get_position() const
{
    return position;
}

621. Get direction.

| Log | [LDF 2003.07.09.] Added this function. |

(Declare Focus functions 602 ) \equiv
inline const Point &get_direction() const
{
    return direction;
}


(Declare Focus functions 602 ) \equiv
inline const real &get_distance() const
{
    return distance;
}

623. Get up. [LDF 2002.09.18.] Added this function.
624. Get transform.

\[ \text{(Declare Focus functions 602) } \equiv \]
\[
\text{inline const Point } \& \text{get_up() const}
\{
    \text{return up;}
\}
\]

Log

[LDF 2002.09.18.] Added this function.

\[
\text{(Declare Focus functions 602) } \equiv \]
\[
\text{inline const Transform } \& \text{get_transform() const}
\{
    \text{return transform;}
\}
\]

625. Get transform element.

Log

[LDF 2002.09.18.] Added this function.
[LDF 2003.07.04.] Made non-inline. It now calls Transform::get_element() instead of accessing transform::matrix directly. This is no longer possible, because Focus is no longer a friend of Transform.
[LDF 2003.07.09.] Changed the const unsigned int arguments to const unsigned short.

\[
\text{(Declare Focus functions 602) } \equiv \]
\[
\text{real get_transform_element(const unsigned short row, const unsigned short column) const;}
\]
626.  

(Define Focus functions 605 ) +≡

real Focus : get_transform_element( const unsigned short row , const unsigned short column ) const

{  
   return transform .get_element( row , column );
}

627.  Get persp.

Log

[ LDF 2002.09.18. ] Added this function.

(Declare Focus functions 602 ) +≡

inline const Transform & get_persp () const  

{  
   return persp ;
}

628.  Get persp element.

Log

[ LDF 2002.09.18. ] Added this function.
[ LDF 2003.07.04. ] Made non-inline. It now calls Transform::get_element() instead of accessing transform . matrix directly. This is no longer possible, because Focus is no longer a friend of Transform.
[ LDF 2003.07.09. ] Changed the const unsigned int arguments to const unsigned short.

(Declare Focus functions 602 ) +≡

real get_persp_element( const unsigned short row , const unsigned short column ) const;

629.

(Define Focus functions 605 ) +≡

real Focus : get_persp_element( const unsigned short row , const unsigned short column ) const

{  
   return persp .get_element( row , column );
}


Default value. Can be changed.

( Global variables 18 ) +≡

Focus default_focus ( 0 , 10 , -10 , 0 , 10 , 0 , 10 );
633. **This is what’s compiled.**

   (Include files 6)
   (Version control identifier 5)
   (Define **class** **Point** 309)
   (Declare non-member template functions for **Point** 336)
   (Define **Point** constructors 325)
   (Define **class** **Focus** 600)
   (Define comparison classes 498)
   (Define static **Point** data members 310)
   (Type definitions 15)
   (Global constants 22)
   (Global variables 18)
   (Define **Transform** functions 169)
   (Define **Point** functions 330)
   (Define non-member non-template functions for **Point** 481)
   (Define **bool**,**_**point** functions 314)
   (Define **bool**,**_**point**,**_**quadruple** functions 316)
   (Define **bool**,**_**real**,**_**point** functions 318)
   (Define **Focus** functions 605)
   (Define **Label** functions 514)
   (Define **Picture** functions 264)

634. **This is what’s written to points.h.**

    (points.h 634) +=
    (Define **class** **Point** 309)
    (Define **class** **Focus** 600)
    (Define comparison classes 498)
    (Declare non-member template functions for **Point** 336)
    (Declare non-member non-template functions for **Point** 480)
    (Type definitions 15)
    (Declarations for the header file 21)

635. **Line (lines.web).** [LDF 2002.10.29] **Lines are not Shapes.** They are used for performing vector operations. A **Line** is defined by a **Point** representing a position vector and a **Point** representing a direction vector.

    [LDF 2003.06.03] TO DO: Add setting functions.

---

**Log**


[LDF 2003.11.12] Removed the version control identifiers from the CWEB files for the distribution of 3DLDF 1.1. They’re still used in my development versions.

[LDF 2003.12.10] Put the version control identifiers back into my release versions for 3DLDF 1.1.4. I’ve already put some of them back in, now I’m doing the rest of them. However, the release versions are now in their own RCS repository.
636. Include files.

```c
#include "loader.h"
#include "pspglb.h"
#include "io.h"
#include "colors.h"
#include "transform.h"
#include "shapes.h"
#include "pictures.h"
#include "points.h"
```

637. Line struct definition.

```c
struct Line {
    public: Point position;
    Point direction;
    // Declare Line constructors 639
    // Declare Line functions 643
};
```

This code is used in sections 657 and 658.

638. Constructors, [LDF 2002.10.29.] The constructors and assignment operator take Point arguments for position and direction. If you want to get the Line between two Points, use Point ::getLine().

639. Default constructor. This constructor takes two optional Point arguments. The default for the Point arguments is origin.

```c
Line(const Point &pos = origin, const Point &dir = origin);
```

See also section 641.

This code is used in section 637.

---

[Log]

[LDF 2002.04.12.] It took me a while to figure out why I was having problems with Lines. The constructor was making the opposite assumption, namely, that it was supposed to calculate the Line from its arguments, rather than just taking them as they were. This caused a problem in Plane::intersectionLine().
640.  
(Define Line constructors 640) \equiv 
 Line::Line(const Point &pos, const Point &dir)  
  : position(pos), direction(dir) {  
    position.apply_transform();  
    direction.apply_transform();  
  }
See also section 642.
This code is used in section 657.

641. Copy constructor. [LDF 2002.10.29] Calling apply_transform() on position and direction is probably unnecessary, because it will already have been called on l-position and l-direction when l was declared or assigned to. But maybe some function has affected l-position.transform or l-direction.transform, so I'm doing it just to be sure.
(Declare Line constructors 639) \equiv 
 Line(const Line &l);

642.  
(Define Line constructors 640) \equiv 
 Line::Line(const Line &l)  
  : position(l.position), direction(l.direction) {  
    position.apply_transform();  
    direction.apply_transform();  
  }

643. Assignment.
(Declare Line functions 643) \equiv 
 void operator=(const Line &l);
See also sections 646, 648, and 652.
This code is used in section 637.
644.  
(Define Line functions 644) \( \equiv \)
void Line::operator=(const Line &l)
{
    position = l.position;
    direction = l.direction;
}
See also sections 649, 650, 651, 653, and 978.
This code is used in sections 657 and 980.

645. Get Line.  (Point function). LDF Undated. Declared in points.web. Must be defined here, because Line is an incomplete type there.
[LDF 2003.06.06.]  get_line() returns a Line \( l \) corresponding to the line from \(*this\) to \( p \), where \( l.position \) is a Point on the Line, and \( l.direction \) is a direction vector. \( l.position \) will be \(*this\), and \( l.direction \) will be \( pt - *this \).

Log

[LDF 2003.06.06.]  BUG FIX: Changed the call to Line(), so that the argument for direction is \( pt - *this \) instead of \( pt \).

646. Get Path.  [LDF 2003.06.06.] Returns a linear Path consisting of two Points, and corresponding to the Line. Must be defined in paths.web, because Path is an incomplete type here.

Log

[LDF 2003.06.06.]  Added this function.

647. Intersection.  [LDF 2003.06.06.] Commented-out. This function doesn’t work. Using a different version, that finds the intersection points of the traces of the lines on two or all of the major axes. TO DO: Fix it!

LDF Undated. Declared in points.web, but must be defined here, because Line is an incomplete type in points.web.

Log

[LDF 2002.04.12.]  Moved this function definition here from points.web because it requires the use of Lines, and Line is an incomplete type there.
[LDF 2002.04.15.]  Commented-out, because I’m having problems with it. Commented old version in points.web back in. I don’t quite understand this, because it seemed to be working.
[LDF 2002.04.22.]  Changed return value to bool_point, to correspond with the old version. This facilitates testing, since all I have to do is to comment-out whichever version I don’t want to use, and uncomment-out the other one. Made a few changes in the function definition in order to be able to return a bool_point.


648. Get distance. [LDF 2002.04.22] Renamed this get_distance() from intersection_point(). The old version of Point::intersection_point(), which I am currently using again, since the new version wasn’t working, returns a bool_point, which is sensible. If I start using the commented-out version above again, I should have it return a bool_point too, instead of a bool_real_point. This will make it easier to switch back to the old version, if I have problems again.

[LDF 2003.06.11] START HERE. TO DO: get_distance() may be working now, due to changes I’ve made to Line elsewhere. Read this through and see how it works. Then test. Also, check where it’s used.

[LDF 2003.06.03] When I’ve fixed it, add description to line.texi.
(Declare Line functions 643) \(\equiv\)
   \[\text{bool real point get\_distance (const Line &l) const;}\]

649.
(Define Line functions 644) \(\equiv\)
\[
\begin{align*}
\text{bool real point Line::get\_distance (const Line &l) const} & \{ \text{bool DEBUG = false; } /* true */ \\
& \quad \text{if (DEBUG) cout \ll "Entering Line::get\_distance()\n";} \\
& \quad \text{bool real point brp;} \\
& \quad \text{Point normal = direction.cross\_product(l.direction);} \\
& \quad \text{if (DEBUG) normal.show("normal after cross\_product.");} \\
& \quad \text{Point normal\_unit = normal.unit\_vector();} \\
& \quad \text{if (DEBUG) normal\_unit.show("normal\_unit.");} \\
& \quad \text{if (normal\_unit \equiv origin) \{} \\
& \quad \quad \text{if (DEBUG) cout \ll "Lines are parallel.\n" \ll flush;} \\
& \quad \quad \text{brp.b = false; } /* No intersection. */ \\
& \quad \quad \text{brp.pt = INVALID\_POINT;} \\
& \quad \quad \text{Point temp\_pt(l\_position);} \\
& \quad \quad \text{temp\_pt \(\equiv\) position;} \\
& \quad \quad \text{temp\_pt = temp\_pt.cross\_product(direction);} \\
& \quad \quad \text{brp.r = temp\_pt.magnitude() / direction.magnitude(); } /* [LDF 2002.10.29.] Distance. */ \\
& \quad \quad \text{if (DEBUG) \{} \\
& \quad \quad \quad \text{cout \ll "distance\_\equiv =\" \ll brp.r \ll endl \ll flush;} \\
& \quad \quad \\text{\}} \\
& \quad \quad \text{if (DEBUG) cout \ll "Exiting Line::get\_distance()\n";} \\
& \quad \quad \text{return brp;} \\
& \quad \\text{\}} \\
& \quad \text{else \{} \\
& \quad \quad \text{if (DEBUG) cout \ll "Lines are not parallel.\n" \ll flush;} \\
& \quad \quad \text{brp.r = \textit{fabs}((l\_position - position).dot\_product(normal\_unit));} \\
& \quad \quad \text{if (DEBUG) cout \ll "distance\_\equiv =\" \ll brp.r \ll endl \ll flush;} \\
\end{align*}
\]
Lines have an intersection.

[LDF 2003-08-27] Commented-out the declarations of \( u_x \), \( u_y \), and \( u_z \), since they are not used. I haven't deleted them, in case I need them someday.

```c
(Define Line functions 644) +\equiv
if (brp.r < Point::epsilon()) {
    if (DEBUG) cout << "Lines do not intersect. \n" << flush;
    brp.r = 0;
    brp.b = true;
real ax = position.getx();
real ay = position.gety();
real az = position.getz();
real bx = l.position.getx();
real by = l.position.gety();
real bz = l.position.getz();
#endif
real wx = l.direction.getx();
real wy = l.direction.gety();
real wz = l.direction.getz();
real ux = normal.getx();
real uy = normal.gety();
real uz = normal.getz();
real t;
if (ux \neq 0) {
    if (DEBUG) cout << "u_x=/0\n";
    t = (((bx - ax) * wx) - ((by - ay) * wx)) / ux;
}
else if (uy \neq 0) {
    if (DEBUG) cout << "u_y=/0\n";
    t = (((by - ay) * wy) - ((bz - ax) * wy)) / uy;
}
else if (uz \neq 0) {
    if (DEBUG) cout << "u_z=/0\n";
    t = (((bz - ax) * wz) - ((bz - ax) * wz)) / uz;
}
else {
    cerr << "This case should not happen! \n" << "Lines do not intersect. \n" << "normal=\ne\nThis case should have been dealt with.\n" << "caught above, so something is really wrong. \n" << "Returning INVALID_BOOL_REAL_POINT. \n" << flush;
    return INVALID_BOOL_REAL_POINT;
}
if (DEBUG) cout << "t=\n" << t << endl << flush;
brp.pt = direction;
brp.pt += t;
brp.pt += position;
```
if (DEBUG) {
    brp.pt.show("intersection_point:");
} else {
    if (DEBUG) cout << "Lines are not parallel, but do not intersect. \n"
        << flush;
    brp.b = false;
    brp.pt = INVALID_POINT;
    if (DEBUG) cout << "Exiting Line::get_distance()\n"
        << flush;
    return brp;
} }

651. Lines are not parallel, but do not intersect.
}<Define Line functions 644> +\equiv

void show(string text = "");

652. Show.
<Declare Line functions 643> +\equiv

653.  
<Define Line functions 644> \equiv

void Line::show(string text)
{
    if (text == "") cout << "Line:\n";
    else cout << text << endl;
    position.show("position: ");
    direction.show("direction: ");
}

654. Global constants for Line.
<Line global constants 654> \equiv
extern const Line INVALID_LINE(INVALID_POINT,INVALID_POINT);
This code is used in section 657.

655. 
<Definitions for the header file 21> +\equiv
extern const Line INVALID_LINE;

656. Putting Line together.

657. This is what’s compiled.
<Include files 6>
<Version control identifier 5>
<Define struct Line 637>
<Define Line constructors 640>
<Line global constants 654>
<Define Line functions 644>
<Define Point functions 330>
This is what’s written to lines.h.

(Define struct Line 637)
(Declarations for the header file 21)

Planes are not Shapes. They are used for performing vector operations. A Plane is defined by a Point representing a point on the plane, a Point representing the normal to the plane, and the distance of the plane from the origin.

---

Log

[LDF 2002.04.12] Created this file. Removed the code for Plane from points.web and put it here.
[LDF 2003.11.12] Removed the version control identifiers from the CWEB files for the distribution of 3DLDF 1.1. They’re still used in my development versions.
[LDF 2003.12.10] Put the version control identifiers back into my release versions for 3DLDF 1.1.4. I’ve already put some of them back in, now I’m doing the rest of them. However, the release versions are now in their own RCS repository.

(Version control identifier 5) +≡

static string res_id = "$Id: planes.web,v 1.4 2004/01/12 21:31:54 ulminst oExp$";

Include files.

(Include files 6) +≡

#include "loader.h"
#include "pspglb.h"
#include "io.h"
#include "colors.h"
#include "transfor.h"
#include "shapes.h"
#include "pictures.h"
#include "points.h"
#include "lines.h"

Plane struct definition.

(Define struct Plane 661) +≡

struct Plane {
    public: Point normal;
    Point point;
    real distance;

    (Declare Plane functions 663)
};

This code is used in sections 694 and 695.

Constructors.

Default constructor. [LDF 2003.06.06] Creates a degenerate Plane with point≡normal≡origin, and distance≡0. I could have made the Plane be equal to INVALID_PLANE, but there’s probably no reason for doing so. A Plane constructed using this constructor will probably be set using the assignment operator or Path::get_plane() immediately, or very soon after being declared.
§663  3DLDF-1.1.5.1  

Log

[LDF 2003.06.06.]  Added this function.

(Declare Plane functions 663) ≡
  Plane(void);
See also sections 665, 667, 669, 672, 674, 677, 679, 684, 686, 687, and 689.
This code is used in section 661.

664.
(Define Plane functions 664) ≡
  Plane::Plane(void)
  {
    normal = point = origin;
    distance = 0;
  }
See also sections 666, 668, 670, 673, 675, 678, 680, 685, 688, 690, and 966.
This code is used in sections 694 and 980.

665. Copy constructor.

Log

[LDF 2003.06.06.]  Added this function.

(Declare Plane functions 663) ≡
  Plane(const Plane &p);

666. Define Plane functions 664) +
   Plane::Plane(const Plane &p)
   {
      *this = p;
      return;
   }

667. Point arguments.

LOG

LDF 2003.06.03.] Changed this function, BUG FIX: distance is now calculated, instead of being passed
as an argument. normal is now made a unit vector.
LDF 2003.06.06. Changed, so that if point or normal is equal to INVALID_POINT, the other one is also
set to INVALID_POINT, and distance is set to INVALID_REAL.
LDF 2003.06.06. Arguments are no longer optional. I've made this change, because I've added a default
constructor.
LDF 2003.06.06. Added conditional to test for case that point=normal. In this case, a warning message
is printed to standard error, they are both set to INVALID_POINT, and distance is set to INVALID_REAL.
LDF 2003.06.24.] BUG FIX. Formerly, INVALID_PLANE was returned, if point == normal. This has been
changed, so that INVALID_PLANE is returned, if normal == origin. There is, of course, no reason why point
shouldn't be equal to normal.

(Declare Plane functions 663) +
   Plane(const Point &p, const Point &n);
\section{Point Arguments}

668. \hspace{1em} (Define Plane functions 664) \hfill \equiv

\begin{verbatim}
Plane::Plane(const Point &p, const Point &n)
    : normal(n), point(p) {
        point.apply_transform();
        normal.apply_transform();
        if (point == INVALID_POINT) {
            normal = INVALID_POINT;
            distance = INVALID_REAL;
            return;
        } else if (normal == INVALID_POINT) {
            point = INVALID_POINT;
            distance = INVALID_REAL;
            return;
        } else if (normal == origin) {
            cerr << "WARNING! In Plane():\n            "normal=" << point << "\nplane is INVALID PLANE.\n            " distance = INVALID_REAL;
            return;
        }
        normal.unit_vector(true);
        distance = -point.dot_product(normal);
        if (fabs(distance) < Point::epsilon()) distance = 0;
        return;
    }
\end{verbatim}

669. Assignment.

\begin{verbatim}
  const Plane &operator=(const Plane &p);
\end{verbatim}

\[\text{Log}
\begin{tabular}{l}
\hline
[LDF 2003.06.06] Added this function.
\hline
\end{tabular}\]
670. Define Plane functions 664 +≡
   const Plane &Plane::operator=(const Plane &p)
   {
     if (this ≡ &p)   /* Make sure it’s not self-assignment. [LDF 2003.06.06.] */
       return *this;
     point = p.point;
     normal = p.normal;
     distance = p.distance;
     return p;
   }

671. Comparing Planes.

[LDF 2003.06.06.] Added this section.

672. Equality.

[LDF 2003.06.06.] Added this function.

Declare Plane functions 663 +≡
   bool operator==(const Plane &p) const;

673. Define Plane functions 664 +≡
   bool Plane::operator==(const Plane &p) const
   {
     return (point ≡ p.point) ∧ (normal ≡ p.normal) ∧ (distance ≡ p.distance));
   }

674. Inequality.

[LDF 2003.06.06.] Added this function.

Declare Plane functions 663 +≡
   bool operator!=(const Plane &p) const;
675.  (Define Plane functions 664) \[\equiv\]
   bool Plane::operator\((\text{const Plane } & p)\) const
   \{
      return \!(\text{operator\(\equiv\)(p)});\}

676.  Get distance.

677.  Point argument.  \[\text{LDF 2003.06.03.}\] This function returns a real_short \(r\), where \(r_{\text{first}}\) is the
distance of the Point from the Plane. \(r_{\text{first}}\) is always positive. \(r_{\text{second}}\) can take on the following values:
0 If the Point lies in the Plane,
1 If it lies on the side of the Plane pointed at by the normal to the Plane, considered to be the “outside”.
-1 If it lies on the side of the Plane not pointed at by the normal to the Plane, considered to be the “inside”.

[\text{LDF 2003.06.03.}\] Changed the definition of this function. The old definition was incorrect. Also changed
return type from real to real_short.

[\text{LDF 2003.06.04.}\] BUG FIX: In the case that \(r_{\text{fabs}} < \text{Point}::\text{epsilon}()\), now \(r_{\text{fabs}}\) is set to 0. Previously, \(r\) was, which was wrong, because \(r_{\text{fabs}}\) is returned, not \(r\). Also, I now set \(r_{\text{fabs}}\) and \(s\) to 0 separately,
because they are of different types. I don’t believe any compiler would have trouble with this, but I think
it’s cleaner if they are assigned to separately.

(Declare Plane functions 663) \[\equiv\]
   real_short get_distance(const Point & p) const;

678.  (Define Plane functions 664) \[\equiv\]
   real_short Plane::get_distance(const Point & p) const
   \{
      real \(r = (p - \text{point}).\text{dot.product}()\);
      real \(r_{\text{fabs}} = \text{fabs}(r)\);
      signed short \(s\);
      if (\(r_{\text{fabs}} < \text{Point}::\text{epsilon}()\)) \{\n         \(r_{\text{fabs}} = 0;\)
         \(s = 0;\)
      \} \else \(s = \text{static_cast}(\text{signed short})(r/r_{\text{fabs}})\);
      return real_short(r_{\text{fabs}},s);
   \}

679.  No argument.  \[\text{LDF 2003.06.03.}\] This version of get_distance() returns the data member distance
and its sign, i.e., the distance of origin to the Plane, and which side of the Plane it lies on. I'm not using
origin as the default for an optional Point argument, because of problems that may arise, when I implement
user_coordinates and view_coordinates.

[\text{LDF 2003.06.03.}\] Added this function.

(Declare Plane functions 663) \[\equiv\]
   real_short get_distance(void) const;
680.  
(Define Plane functions 664) \(\equiv\)

```cpp
real_short Plane::get_distance(void) const
{
    real f = fabs(distance);
    signed short s = static_cast(signed short)(distance/f);
    return real_short(f, s);
}
```

681. Point is on Plane.  [LDF 2003.06.04] This function returns \textit{true}, if the Point lies on the Plane \(p\), otherwise \textit{false}. Declared in \texttt{points.web}. Must be defined here, because \texttt{Plane} is an incomplete type in that file.

---

[LDF 2003.06.04] Added this function.

---

682. Intersection.

683. Intersection with a line.

684. Point arguments.

---

[LDF 2003.06.03] Added this function.

---

(Declare Plane functions 663) \(\equiv\)

```cpp
bool_point intersection_point(const Point &p0, const Point &p1) const;
```
685. \(\text{Define Plane functions 664} \implies\)

\[
\begin{align*}
\text{bool_point Plane::intersection_point(const Point &p0, const Point &p1) const} \\
\{
\text{bool_point bp; real denominator = (p0 - p1).dot_product(normal);} \\
\text{if (denominator \equiv 0) /* !! TO DO: Handle cases: Path is in Plane, and Path is in a parallel Plane. [LDF 2003.06.03] */} \\
\{
\text{cerr \ll "ERROR! In Plane::intersection_point(const Point&, const Point&, \):" \ll} \\
\text{"denominator is \equiv 0. Can’t divide." \ll "Path is either in Plane, or parallel to Plane." \ll endl \ll} \\
\text{"Returning INVALID_BOOL_POINT." \ll endl \ll flush; return INVALID_BOOL_POINT;} \\
\}
\text{real numerator = p0.dot_product(normal) + distance;} \\
\text{bp.pt = p0 + ((numerator/denominator) \ast (p1 - p0));} \\
\text{bp.b = (bp.pt \neq INVALID_POINT) ? true : false;} \\
\text{return bp;}
\}
\]

686. \text{Path argument. [LDF 2003.06.03.] Defined in paths.web, because Path is an incomplete type in this file.}

\begin{tabular}{ll}
\hline
\text{[LDF 2003.06.03.] Added this function.} & Log \\
\hline
\end{tabular}

\begin{tabular}{ll}
\text{(Declare Plane functions 663) \implies} & Log \\
\text{bool_point intersection_point(const Path &p) const;} & \\
\end{tabular}

687. \text{Intersection of two Planes. [LDF 2002.10.29.] TO DO: Look up and explain!}

\begin{tabular}{ll}
\hline
\text{[LDF 2003.06.04.] Changed to const.} & Log \\
\hline
\end{tabular}

\begin{tabular}{ll}
\text{(Declare Plane functions 663) \implies} & Log \\
\text{Line intersection_line(const Plane &p1) const;} & \\
\end{tabular}
Define Plane functions 664) +

688.

Line Plane::intersection_line(const Plane &pl) const
{
    bool DEBUG = false;  /* true */
    if (DEBUG) cout << "Entering Plane::intersection_line()\n";
    Point pl_normal(pl_normal);
    Point direction_vector = normal.cross_product(pl_normal); /* [LDF 2002.10.29.] Needed?? */
    if (DEBUG) direction_vector.show("direction_vector");
    if (direction_vector == origin) {
        cerr << "In Plane::intersection_line().\n" << "Planes are parallel\n";
        if (distance == pl.distance) {
            /* Ellipse::intersection_points(Ellipse & ) calls this function to find out whether two Planes
            are coincident, so sometimes we don’t want to see these messages. I may decide to add an
            argument bool silent to this function. ?? Add bool silent ?? */
            cerr << "Planes are coincident.\n" << "Returning INVALID_LINE.\n\n" << "and INVALID_POINT as direction.\n\n" << flush;
            return INVALID_LINE;
        } else {
            cerr << "Planes are not coincident.\n" << "Returning INVALID_LINE.\n\n" << flush;
            return INVALID_LINE;
        }
    } else {  /* Outer if. */
        if (DEBUG) cout << "Planes are not parallel\n" << flush;
        /* At least one of the x, y, or z components of direction_vector must be non-zero, otherwise, this
        function would have exited by now. */
        real x, y, z;
        real d = distance;
        real e = pl.distance;
        real nx = normal.getx();
        real ny = normal.gety();
        real nz = normal.getz();
        real mx = pl.normal.getx();
        real my = pl.normal.gety();
        real mz = pl.normal.getz();
        real wx = direction_vector.getx();
        real wy = direction_vector.gety();
        real wz = direction_vector.getz();
        if (direction_vector.getx() != 0) {
            x = 0;
            y = -1 * ((d * mz - e * nz) / wx);
            z = (d * my - e * ny) / wz;
        } else if (direction_vector.gety() != 0) {
            x = (d * mz - e * nz) / wy;
            y = 0;
            z = -1 * ((d * mx - e * nx) / wy);
        } else {
            x = -1 * ((d * my - e * ny) / wz);
        }
    }
}
\[ y = (d * m - e * n) / u; \]
\[ z = 0; \]
}
Point point_on_line(x, y, z);
if (DEBUG) {
    point_on_line.show("point_on_line: ");
    direction_vector.show("direction_vector:");
    getchar();
}
if (DEBUG) {
    cout << "Exiting Plane::intersection_line() \n";
    getchar();
}
return Line(point_on_line, direction_vector);

689. Show.

Log

[LDF 2003.06.06] Minor change to the conditional that handles text.
[LDF 2003.06.06] Made show() const.

\langle\text{Declare Plane functions 663}\rangle +\equiv
\begin{verbatim}
void show(string text = "") const;
\end{verbatim}

\langle\text{Define Plane functions 664}\rangle +\equiv
\begin{verbatim}
void Plane::show(string text) const
{
    if (text == "") text = "Plane: ";
    cout << text << endl;
    if (*this == INVALID_PLANE) {
        cout << "INVALID_PLANE::Can't show. " << endl << endl << flush;
        return;
    }
    normal.show("normal:");
    point.show("point:");
    cout << "distance\_u = \_u" << distance << endl << endl << flush;
}
\end{verbatim}

691. Global constants for Plane.
\langle\text{Plane global constants 691}\rangle +\equiv
\begin{verbatim}
extern const Plane INVALID_PLANE(INVALID_POINT, INVALID_POINT);
\end{verbatim}
This code is used in section 694.

692. \langle\text{Declarations for the header file 21}\rangle +\equiv
\begin{verbatim}
extern const Plane INVALID_PLANE;
\end{verbatim}

693. Putting Plane together.
694. This is what's compiled.
   (Include files 6)
   (Version control identifier 5)
   (Define struct Plane 661)
   (Plane global constants 691)
   (Define Plane functions 664)
   (Define Point functions 330)
695. This is what’s written to planes.h.

(planes.h 695) \equiv

(Define struct Plane 661)

(Declarations for the header file 21)

696. Path (paths.web).

Log

[LDF 2003.11.12.] Removed the version control identifiers from the CWEB files for the distribution of 3DLDF 1.1. They’re still used in my development versions.

[LDF 2003.12.10.] Put the version control identifiers back into my release versions for 3DLDF 1.1.4. I’ve already put some of them back in, now I’m doing the rest of them. However, the release versions are now in their own RCS repository.

format Path Shape

(Version control identifier 5) \equiv

static string res_id = "$Id::paths.web,vU1.7\u2004/01/12\u201d30:51@1!1f1nsto1\u201dExp\u201d$";

697. Include files.

(Include files 6) \equiv

#include "loader.h"
#include "psplb.h"
#include "creatnew.h"
#include "io.h"
#include "colors.h"
#include "transform.h"
#include "shapes.h"
#include "pictures.h"
#include "points.h"
#include "lines.h"
#include "planes.h"

698. Path class definition.

Log

[LDF 2002.09.18.] Added projective.extremes. It contains the minimum and maximum values for x, y, and z of the Points in points. It’s used in Picture::output() for surface hiding.

[LDF 2002.08.14.] Added static variables for help lines and curves: help_color, help_dash_pattern, do_help_lines. The variables for help lines (or curves) are part of Path’s interface and can be set anywhere by anyone.

(Define class Path 698) \equiv

class Path : protected Shape {
protected:
  bool line_switch;
  bool cycle_switch;
  bool on_free_store;
  bool do_output; /* LDF 2002.09.18. Added. */
  signed short fill_draw_value; /* Variables for drawing and filling. */
  const Color *draw_color;
  const Color *fill_color;
  string dashed;
string pen;
bool arrow;      /* LDF 2003.01.15. Added. Needed for drawarrow(). */
valarray<real> projective_extremes;  /* LDF 2002.09.18. Added. */
vector<Point *) points;
vector<string> connectors;
public: static const Color *help_color;
static string help_dash_pattern;
static bool do_help_lines;
(Declare Path functions 700)
}

This code is used in sections 980 and 981.

699. Static member variable definitions.

(Define static class Path data members 699) ≡
const Color *Path::help_color = &Colors::red;
string Path::help_dash_pattern = "evenly";
bool Path::do_help_lines = true;

This code is used in section 980.

700. Assignment.

Log

[LDF 2002.10.23.] Now all of the data members of class Path are assigned to except for on_free_store. This has become necessary because of changes in Solid::output(), where temporary Paths have to be created in order to sort them.

[LDF 2002.12.18.] Moved here. With the DEC compiler under Compaq Tru64 on the DEC Alpha computer, it worked to have this following the constructors. With the GNU C++ compiler (GCC) under GNU/Linux on the Intel i686 computer, it didn’t. The copy constructor used the default assignment operator instead of this function, presumably because this function wasn’t known at the time the copy constructor was compiled, although it had been declared previously! URGENT: Move assignment operators for the other classes before the constructors!

[LDF 2003.04.09.] ?? BUG FIX: Now resizing projective_extremes, if after setting it to p:projective_extremes, projective_extremes.size() ≡ 0. This prevents a Memory Fault error at run-time. I don’t know why it should be necessary, though, since all of the constructors of Path and its derived classes resize projective_extremes at least, I thought they did.

(Declare Path functions 700) ≡
virtual Path &operator=(const Path &p);

This code is used in section 698.
701.
(Define Path functions 701) ≡

Path &Path::operator=(const Path &p) {
    if (this == &p) /* Make sure it's not self-assignment. */
        return *this;
    (Discard points and connectors 703)
    line_switch = p.line_switch;
    cycle_switch = p.cycle_switch;
    do_output = p.do_output;
    fill_draw_value = p.fill_draw_value;
    draw_color = p.draw_color; /* LDF 2002.10.23. draw_color and fill_color point to the same
    Color as p.draw_color and p.fill_color. No memory allocation is performed. */
    fill_color = p.fill_color;
    dashed = p.dashed;
    pen = p.pen;
    projective_extremes = p.projective_extremes; /* LDF 2002.09.18. Added this line. */
    if (projective_extremes.size() == 0) /* LDF 2003.04.09. Added this conditional. */
        projective_extremes.resize(6, 0);
    for (vector(Point *)::const_iterator p_iter = p.points.begin(); p_iter != p.points.end(); p_iter++)
        points.push_back (create_new < Point > (*p_iter));
    for (vector<string>::const_iterator c_iter = p.connectors.begin(); c_iter != p.connectors.end();
        c_iter++) {
        connectors.push_back (*c_iter);
    }
    return *this;
}


This code is used in section 980.

702. Constructors and setting functions. Each constructor taking an argument has a corresponding
function for setting an already existing Path.

Log

[LDF 2003.04.06.] BUG FIX: Now setting dashed = "", pen = ", and arrow = false in all constructors
and setting functions. This fixed a problem I was having with Icosahedron: One of the Reg_Polygons
was drawn with an arrow.

703. Discard points and connectors. This is useful in the setting functions.
(Discard points and connectors 703) ≡

if (points.size() > 0) {
    for (vector<Point>::iterator iter = points.begin(); iter != points.end(); ++iter) {
        delete iter;
    }
    points.clear();
}

if (connectors.size() > 0) connectors.clear();

This code is used in sections 701, 710, 715, 720, and 729.
(Declare Path functions 700) \( \equiv \)
\[
\text{Path();}
\]

705.  
(Define Path functions 701) \( \equiv \)
\[
\text{Path::Path()}
\]
\[
\begin{aligned}
& \quad \text{bool DEBUG = false; /* true */}
& \quad \text{if (DEBUG) cout << "Entering Path() \n\nversion. \n\nflush;}
& \quad \text{on_free_store = false;}
& \quad \text{line_switch = false;}
& \quad \text{cycle_switch = false;}
& \quad \text{fill_draw_value = 0;}
& \quad \text{dashed = "}n"; /* LDF 2003.04.06. Added these three lines. */}
& \quad \text{pen = "n";}
& \quad \text{arrow = false;}
& \quad \text{draw_color = 0;}
& \quad \text{fill_color = 0;}
& \quad \text{do_output = true; /* LDF 2002.09.18. Added this line. */}
& \quad \text{projective_extremes.resize(6, 0); /* LDF 2002.09.18. Added this line. */}
& \quad \text{if (DEBUG) cout << "Exiting Path() \n\nflush;}
& \quad \text{return;}
\end{aligned}
\]

706. Lines. [LDF 2002.10.15.] Lines in this sense are Paths containing two Points and the connector “—” They should not be confused with the struct Line, which is for vector operations (where the word “vector” is used in its mathematical sense).

707. Constructor.  
(Declare Path functions 700) \( \equiv \)
\[
\text{Path(const Point \& p0, const Point \& p1);}
708. (Define Path functions 701) \[IELDS\]

`Path::Path(const Point &p0, const Point &p1) { bool DEBUG = false; /* true */
  if (DEBUG) cout << "Entering Path() \n line\nversion(\nversion(\n\n\n      line\nversion(\n      cycle\nswitch = false;
  do_output = true; /* LDF 2002.09.18. Added this line. */
  projective_extremes.resize(6,0); /* LDF 2002.09.18. Added this line. */
  fill_draw_value = 0;
  draw_color = 0;
  fill_color = 0;
  dashed = "\n"; /* LDF 2003.04.06. Added these three lines. */
  pen = "\n";
  arrow = false; points.push_back ( create_new < Point > (p0) );
  points.push_back ( create_new < Point > (p1) );
  connectors.push_back ("--");
  if (DEBUG) cout << "Exiting Path() \n line\nversion(\nversion(\n      return; }
``

709. Setting function. (Declare Path functions 700) \[IELDS\]

`void set(const Point &p0, const Point &p1);`

710. (Define Path functions 701) \[IELDS\]

`void Path::set(const Point &p0, const Point &p1) { line\nswitch = true;
  cycle\nswitch = false;
  do_output = true; /* LDF 2002.09.18. Added this line. */
  fill_draw_value = 0;
  draw_color = 0;
  fill_color = 0;
  dashed = "\n"; /* LDF 2003.04.06. Added these three lines. */
  pen = "\n";
  arrow = false;
  (Discard points and connectors 703)
  projective_extremes = 0; /* LDF 2002.09.18. Added this line. */
  points.push_back ( create_new < Point > (p0) );
  points.push_back ( create_new < Point > (p1) );
  connectors.push_back ("--");
  line\nswitch = true; }
``

711. Points and one type of connector. This constructor takes a variable number of Point * arguments, but only allows one type of connector. The argument list must end with 0. If the order of the named arguments is reversed, the compiler can’t resolve certain calls to Path(). It couldn’t resolve between Path (bool cycle, string connector ... ) and Path (Point * first, Point * ptr ... ). I don’t know why it should have had trouble, though, since pointers to Points are not bools.

[LDF 2002.4.6] Probably it couldn’t distinguish between a pointer and an int on the one hand and a bool and an int on the other. I hope that bools are more efficiently implemented than as ints, though!

[LDF 2002.10.29] ?? I don’t know why create needs instructions to put thin spaces after the "bool" in the declaration and definition below. Maybe it’s because of the "...".
712. Constructor.

(Declare Path functions 700) \(\equiv\)

\[\text{Path(string connector, bool cycle ...);}\]

713.

(Define Path functions 701) \(\equiv\)

\[\text{Path::Path(string connector, bool cycle ...){ bool DEBUG = false; }\]

\[\text{if (DEBUG) cout }<"\text{EnteringPath}() \_\text{connector, cycle, ...}.n" \text{\ll flush; }\]

\[\text{line\_switch = false; }\]

\[\text{draw\_color = }true; \]

\[\text{draw\_color = cycle; }\]

\[\text{connectors.push\_back (connector); }\]

\[\text{do\_output = true; }\]

\[\text{projective\_extremes.resize(6, 0); }\]

\[\text{fill\_draw\_value = 0; }\]

\[\text{fill\_color = 0; }\]

\[\text{dashed = }true; \]

\[\text{pen = }true; \]

\[\text{arrow = false; }\]

\[\text{va\_list ap; }\]

\[\text{va\_start(ap, cycle); }\]

\[\text{Point }*\text{arg\_ptr; while (arg\_ptr = va\_arg(ap, Point *)) }\}

\[\text{points.push\_back (create\_new < Point > (arg\_ptr)); }\]

\[\text{va\_end(ap); }\]

\[\text{if (DEBUG) cout }<"\text{ExitingPath}() \_\text{connector, cycle, ...}.n" \text{\ll flush; }\]

\[\text{return; }\]

714. Setting function.

(Declare Path functions 700) \(\equiv\)

\[\text{void set(string connector, bool cycle ...);}\]
715.  
(Define Path functions 701) +≡

void Path::set (string connector, bool cycle ... ) { on_free_store = false;
  line_switch = false;
  cycle_switch = cycle;
  do_output = true;  /* LDF 2002.09.18. Added this line. */
  (Discard points and connectors 703)
  filldraw_value = 0;
  draw_color = 0;
  fill_color = 0;  /* LDF 2003.04.06. Added these three lines. */
  pen = "n";
  arrow = false;
  projective_extremes = 0;  /* LDF 2002.09.18. Added this line. */
  connectors.push_back (connector);
  va_list ap;  /* For the variable length argument list. */
  va_start (ap, cycle);
  Point *arg_ptr; while ((arg_ptr = va_arg (ap, Point *)) != static_cast < Point > (0))
    points.push_back (create_new < Point > (arg_ptr));
  va_end (ap); }

716. Variable number of Points and connectors. These functions takes a variable number of
alternating Point * and connector arguments, starting with a Point *. The argument list must end with
0. We don't need an argument for whether it's a cycle or not, because if it is, it will have a connector at the
end.

[LDF 2002.10.29.] BUG FIX: No longer pushing first_point_ptr onto points. Copying it instead.

717. Constructor.
(Declare Path functions 700) +≡

Path ( Point *first_point_ptr ... ) ;
718.  
(Define Path functions 701) +≡

Path::Path (Point *first_point_ptr ... ) { bool DEBUG = false; /* true */
    if (DEBUG) cout << "Entering Path() with Point*..." << flush;
    on_free_store = false;
    line_switch = false;
    cycle_switch = false;
    do_output = true; /* LDF 2002.09.18. Added this line. */
    projective_extremes.resize(6, 0); /* LDF 2002.09.18. Added this line. */
    fill_draw_value = 0;
    draw_color = 0;
    fill_color = 0;
    dashed = "n"; /* LDF 2003.04.06. Added these three lines. */
    pen = "n";
    arrow = false; points.push_back (create_new <Point> (first_point_ptr));
    va_list ap; /* For the variable length argument list. */
    va_start (ap, first_point_ptr);
    Point *point_ptr;
    char *connector_ptr;
    string connector_string; while ((connector_ptr = va_arg (ap, char *))
        ! static_caster (char *)(0)) {
        connectors.push_back ((connector_string = connector_ptr));
        if ((point_ptr = va_arg (ap, Point *)) == static_caster (Point *)(0)) {
            cycle_switch = true;
            break;
        }
        points.push_back (create_new <Point> (point_ptr)); }
    va_end (ap);
    if (DEBUG) cout << "Exiting Path() with Point*..." << flush;
}

719. Setting function.
(Declare Path functions 700) +≡

void set (Point *first_point_ptr ... );
§720. Define Path functions 701 \( \text{\equiv} \)

\[ \text{void Path::set (Point *first_point_ptr ... ) \{ on_free_store = false; line_switch = false; cycle_switch = false; do_output = true; /* LDF 2002.09.18. Added this line. */ (Discard points and connectors 703) projective_extremes = 0; /* LDF 2002.09.18. Added this line. */ fill_draw_value = 0; draw_color = 0; dashed = "\"; /* LDF 2003.04.06. Added these three lines. */ pen = "\"; arrow = false; points.push_back (create_new <Point> (first_point_ptr)); va_list ap; /* For the variable length argument list. */ va_start(ap,first_point_ptr); Point *point_ptr; char *connector_ptr; string connector_string; while ((connector_ptr = va.argv(ap,char *)) \( \neq \) (char *) 0) \{ connectors.push_back ((connector_string = connector_ptr)); if ((point_ptr = va.argv(ap,Point *)) \( \equiv \) static_cast (Point *) (0)) \{ cycle_switch = true; break; \} points.push_back (create_new <Point> (point_ptr)); \} va_end(ap); \} \]

721. Copy constructor. [LDF 2003.04.06] ??!! BUG: Got a memory fault when I tried to use this function. Haven’t tested it yet. It worked to use the default constructor and then the assignment operator. Maybe it’s not kosher to use \( \text{\*this} = p \) in a copy constructor.

\[ \text{Log} \]

[\text{LDF 2002.10.15.}] Rewrote this function. The old version caused a memory fault when I tried to use it. I’ve taken code from the default constructor and the assignment operator and put it here without bothering to see what was causing the problem. Probably the old version didn’t account for changes I’ve made in other places, perhaps in the class definition.

[\text{LDF 2002.11.03.}] Rewrote this function. Now just using the assignment operator.

\[ \langle \text{Declare Path functions 700} \rangle \text{\equiv} \]

\[ \text{Path(const Path &p);} \]
722. Define Path functions 701 ) +\equiv
Path := Path ( const Path &p )
{
   bool DEBUG = false;   /* true */
   if ( DEBUG ) cout << "Entering Path\copy\constructor.\n";
   *this = p;
   on_free_store = false;
   if ( DEBUG ) cout << "Exiting Path()\copy\constructor.\n" << flush;
   return;
}

723. Pseudo-constructor for dynamic allocation.

724. Pointer argument.

Log

[LDF 2002.10.29.] Added argument const Path *p. If p \neq 0, the new Path is assigned to using the values
from p.
[LDF 2003.12.30.] Replaced this version of Path::create_new() with a specialization of template<class
C> C*create_new().
[LDF 2003.12.30.] Changed the argument. It’s now a const Path *.
[LDF 2003.12.30.] Removed default argument "0", because this caused a compiler error when using the
DEC C++ compiler. Apparently, it suffices to declare a default argument in the template declaration.

(Declare non-member template functions for Path 724 ) +\equiv
Path *create_new ( const Path *p );

See also section 725.
This code is used in sections 980 and 981.

725. Reference argument.

Log

[LDF 2002.10.29.] Added this function.
[LDF 2003.12.30.] Replaced this version of Path::create_new() with a specialization of template<class
C> C*create_new().
[LDF 2003.12.30.] Changed argument from Path to const Path &.

(Declare non-member template functions for Path 724 ) +\equiv
Path *create_new ( const Path &p );

726. Destructor.

Log

[LDF 2003.08.27.] Made virtual, because GCC with the "-Wall" option issued the following warning:
"class Path" has virtual functions but non-virtual destructor.

(Declare Path functions 700 ) +\equiv
virtual ~Path ( );
727. !! Make sure to delete anything else that I allocate dynamically!

(Define Path functions 701) +≡

Path:: ~Path() {
    bool DEBUG = false; /* true */
    if (DEBUG) {
        cout << "Entering~Path() \n" << flush;
        show("Path:");
        getchar();
    }
    for (vector<Point> :: iterator iter = points.begin(); iter != points.end(); iter++) {
        delete *iter;
    }
    points.clear(); /* [LDF 2002.11.03] This replaces a while loop in which pop_back() was used to empty points. */
    connectors.clear(); /* LDF 2002.11.03. Added. */
    /* LDF 2002.10.07. Added code for handling draw_color and fill_color. */
    if (draw_color != 0 && draw_color_is_on_free_store() == true) {
        if (DEBUG) cout << "Deleting draw_color\n";
        delete draw_color;
        draw_color = 0;
    } else if (DEBUG) {
        cout << "Not deleting draw_color\n";
    }
    if (fill_color != 0 && fill_color_is_on_free_store() == true) {
        if (DEBUG) cout << "Deleting fill_color\n";
        delete fill_color;
        fill_color = 0;
    } else if (DEBUG) {
        cout << "Not deleting fill_color\n";
    }
    if (DEBUG) {
        cout << "Exiting~Path() \n" << flush;
        getchar();
    }
}

728. Clear. LDF Undated. This function is needed because it's a pure virtual function in Shape, and for getting rid of items in Picture::clear().

[LDF 2002.10.07] clear() is needed because it's called on the Shapes that are stored in Pictures, and I don't know of a way of overloading destructors. That is, in Picture::clear(), the actual types of the Shapes are unknown, so I can't call ~Path(), ~Circle(), or other destructors directly. But a named function such as clear() can serve the same purpose.

Log

[LDF 2002.10.07] Added code for deallocating the memory allocated for draw_color and fill_color, if any.

?? I tried calling ~Path() inside Path::clear(), but I got a memory fault. Don't know why. TO DO: Try to find out. However, this isn't urgent.
(Declare Path functions 700) +≡
  virtual void clear();

729.
(Define Path functions 701) +≡
void Path::clear()
{
  bool DEBUG = false;  /* true */
  if (DEBUG) cout << "Entering Path::clear().\n";
  (Discard points and connectors 703)
  /* LDF 2002.10.07. Added code for handling draw_color and fill_color. */
  if (draw_color != 0 \ draw_color\is_on_free_store()) +≡ true) {
    if (DEBUG) cout << "Deleting draw_color\n"
    delete draw_color;
    draw_color = 0;
  }
  else if (DEBUG) {
    cout << "Not deleting draw_color\n";
  }
  if (fill_color != 0 \ fill_color\is_on_free_store()) +≡ true) {
    if (DEBUG) cout << "Deleting fill_color\n"
    delete fill_color;
    fill_color = 0;
  }
  else if (DEBUG) {
    cout << "Not deleting fill_color\n";
  }
  if (DEBUG) cout << "Exiting Path::clear().\n";
}

730. Get copy.

Log
[ LDF 2002.11.03 ] Made virtual. Changed dynamic\_cast() to static\_cast(). This may not work.
[ LDF 2003.01.29 ] It seems to work. At least, I haven’t had any problems with it.

(Declare Path functions 700) +≡
  virtual Shape *get_copy() const;
731.  
(Define Path functions 701) +≡  
Shape *Path::getCopy() const { Path *p = create_new < Path > (0);  
    *p = *this;  
    return static_cast<Shape *>(p); }

732. Set on free store.  

[Log 2004.01.06.] Made non-inline.

(Declare Path functions 700) +≡  
    virtual bool setOnFreeStore(bool b = true);

733.  
(Define Path functions 701) +≡  
    bool Path::setOnFreeStore(bool b)  
    {  
        on_free_store = b;  
        return b;  
    }

734. Setting drawing and filling data.

735. Set fill_draw_value.  
(Declare Path functions 700) +≡  
    virtual void setFillDrawValue(const signed short s);

736.  
(Define Path functions 701) +≡  
    void Path::setFillDrawValue(const signed short s)  
    {  
        fill_draw_value = s;  
        return;  
    }

737. Set draw color.

738. Color version.  
(Declare Path functions 700) +≡  
    virtual void setDrawColor(const Color &c);

739.  
(Define Path functions 701) +≡  
    void Path::setDrawColor(const Color &c)  
    {  
        draw_color = &c;  
        return;  
    }
740. Color pointer version.

(Declare Path functions 700) +≡
    virtual void set_draw_color(const Color *c);

741.

(Define Path functions 701) +≡
    void Path::set_draw_color(const Color *c)
    {
        if (draw_color ≠ 0 ∧ draw_color—is_on_free_store() ≡ true) {
            delete draw_color;
        }
        draw_color = c;
        return;
    }

742. Set fill color.

743. Color version.

(Declare Path functions 700) +≡
    virtual void set_fill_color(const Color &c);

744.

(Define Path functions 701) +≡
    void Path::set_fill_color(const Color &c)
    {
        fill_color = &c;
        return;
    }

745. Color pointer version.

(Declare Path functions 700) +≡
    virtual void set_fill_color(const Color *c);

746.

(Define Path functions 701) +≡
    void Path::set_fill_color(const Color *c)
    {
        if (fill_color ≠ 0 ∧ fill_color—is_on_free_store() ≡ true) {
            delete fill_color;
        }
        fill_color = c;
        return;
    }

747. Set dash pattern.

(Declare Path functions 700) +≡
    virtual void set_dash_pattern(const string s = "");
748. Define Path functions 701 \(+ \equiv \)
    void Path::set_dash_pattern(const string s)
    {
        dashed = s;
        return;
    }

749. Set pen.
    (Declare Path functions 700) \(+ \equiv \)
    virtual void set_pen(const string s = "n");

750. Define Path functions 701 \(+ \equiv \)
    void Path::set_pen(const string s)
    {
        pen = s;
        return;
    }

751. Set connectors. [LDF 2003.02.08.] TO DO: Overload with a version taking a vector(string) as
its argument, arguments.

    [LDF 2003.02.08.] Added this function.

    (Declare Path functions 700) \(+ \equiv \)
    virtual void set_connectors(const string s = ". ..");

752. Define Path functions 701 \(+ \equiv \)
    void Path::set_connectors(const string s)
    {
        connectors.clear();
        connectors.push_back(s);
    }

753. Transformations. [LDF 2002.11.03.] All of the transformations return a Transform, so that the
same Transform can be applied to multiple objects by chaining expressions.

754. Affine transformations.

755. Rotation.

756. Rotation around the main axes.
    (Declare Path functions 700) \(+ \equiv \)
    virtual Transform rotate(const real x, const real y = 0, const real z = 0);
757.  
(Define Path functions) +≡

Transform Path :: rotate(const real x, const real y, const real z)
{
    Transform t;
    t.rotate(x, y, z);
    return *this *= t;
}

758.  Rotatation around an arbitrary axis.

759.  Transform version.  Declared in transform.web.  Must be defined here, because Path is an
      incomplete type there.

Log

[LDF 2003.05.02.] Changed name of this function from rotate::around() to rotate().  This function now
overloads rotate() with three real arguments.
[LDF 2003.07.06.] Changed, so that is_linear() is used, instead of get_line_switch().

(Define Transform functions) +≡

Transform Transform :: rotate(const Path &p, const real angle)
{
    if (!p.is_linear()) {
        cerr << "ERROR! In Transform::rotate(Path, const real).\n" <<
             "Path is not linear. Returning INVALID_TRANSFORM.\n"
        return INVALID_TRANSFORM;
    }
    Transform t;
    t.rotate(p.get_point(0), p.get_last_point(), angle);
    return (*this *= t);
}
760. **Point version.** Declared in points.web. Must be defined here, because Path is an incomplete type there.

[Log]

[LDF 2003.05.02.] Changed name of this function from `rotate_around()` to `rotate()`. This function now overloads `rotate()` with three real arguments.

(Define **Point** functions 330) 

```cpp
Transform Point::rotate(const Path &p, const real angle)
{
    if (!p.get_line_switch()) {
        cerr << "ERROR! In Point::rotate(Path, real). \n" <<
            "Path not a line. Returning INVALID_TRANSFORM. \n\n";
        return INVALID_TRANSFORM;
    }
    Point pt0 = p.get_point(0);
    Point pt1 = p.get_point(1);
    return rotate(pt0, pt1, angle);
}
```

761. **Path versions.**

762. **Point arguments.**

[Log]

[LDF 2002.4.7.] Added default value for `angle` ≡ 180.
[LDF 2003.05.02.] Changed name of this function from `rotate_around()` to `rotate()`. This function now overloads `rotate()` with three real arguments.

(Declare **Path** functions 700) 

```cpp
virtual Transform rotate(const Point &p0, const Point &p1, const real angle = 180);
```

763. **TO DO:** Change this, so that I use `operator*(Transform)` here and in the other transformation functions.

(Define **Path** functions 701) 

```cpp
Transform Path::rotate(const Point &p0, const Point &p1, const real angle)
{
    Transform t;
    t.rotate(p0, p1, angle);
    return (*this += t);
}
```
764. Path arguments.

---

[LDF 2002.4.7.] Added default value for angle ≡ 180.
[LDF 2002.11.03.] Got rid of local Points p0 and p1.
[LDF 2003.06.02.] Changed name of this function from `rotate_around()` to `rotate()`. This function now overloads `rotate()` with three `real` arguments.
[LDF 2003.07.13.] Changed, so that `is_linear()` is used instead of checking the return value of `get_line_switch()`. Also, `get_last_point()` passed as the second argument to `rotate()`, instead of `get_point(1)`.

```c++
(Declare Path functions 700) +≡
    Transform rotate(const Path &p, const real angle = 180);
```

765. (Define Path functions 701) +≡

```c++
    Transform Path :: rotate(const Path &p, const real angle)
    {
        if (!p.is_linear()) {
            cerr << "ERROR! In Path:rotate(Path,real).\n" <<
                 "Path::in::not::a::line::u::Returning::INVALID_TRANSFORM.\n\n";
            return INVALID_TRANSFORM;
        }
        return rotate(p.get_point(0), p.get_last_point(), angle);
    }
```


```c++
(Declare Path functions 700) +≡
    Transform scale(real x, real y = 1, real z = 1);
```

767. (Define Path functions 701) +≡

```c++
    Transform Path :: scale(real x, real y, real z)
    {
        Transform t;
        t.scale(x, y, z);
        return (+this += t);
    }
```

768. Shear.

```c++
(Declare Path functions 700) +≡
    Transform shear(real xy, real xz = 0, real yx = 0, real yz = 0, real zx = 0, real zy = 0);
```
769.  \(\textit{Define Path} \text{ functions 701} \) \(\equiv\)
\(\text{Transform Path} :: \textit{shear} (\text{real } xy, \text{real } xz, \text{real } yx, \text{real } yz, \text{real } zx, \text{real } zy)\)
\{
  \text{bool DEBUG} = \text{false};  \text{ /* true */}
  \text{if} (\text{DEBUG}) \text{cout} \ll \text{"Entering Path::shear().\n"} ;
  \text{Transform } t ;
  t.\text{shear}(xy, xz, yx, yz, zx, zy);
  \text{if} (\text{DEBUG}) \text{cout} \ll \text{"Exiting Path::shear().\n"} ;
  \text{return} (*this += t) ;
\}

770.  \textit{Shift}.  

771.  \(\text{Declare Path} \text{ functions 700} \) \(\equiv\)
\(\text{Transform shift} (\text{real } x, \text{real } y = 0, \text{real } z = 0) ;\)

772.  \(\text{Define Path} \text{ functions 701} \) \(\equiv\)
\(\text{Transform Path} :: \textit{shift} (\text{real } x, \text{real } y, \text{real } z)\)
\{
  \text{Transform } t ;
  t.\text{shift}(x, y, z);
  \text{return} (*this += t) ;
\}

773.  \textit{Point argument}.  

774.  \(\text{Declare Path} \text{ functions 700} \) \(\equiv\)
\(\text{Transform shift} (\text{const Point} & p) ;\)

775.  \textit{Shift times}.  
\[\text{LDF 2003.01.19.}\] \textit{shift::times} () returns \texttt{void}, because \textit{Path} doesn’t have a \textit{Transform} data member, and there’s no guarantee that all of the \textit{Points} on \textit{points} will have identical \textit{transforms}.
\[\text{LDF 2003.01.19.}\] Note that \textit{shift::times} () will only have an effect on the \textit{Points} on a \textit{Path} if it’s called \textit{after} a call to \textit{shift} () and \textit{before} an operation is applied that causes \textit{Point :: apply\_transform} () to be called.

\[\text{LDF 2003.01.19.}\] Added this section.
776. real arguments.

---

[LDF 2003.01.19.] Added this function.

(Declare Path functions 700) +≡

virtual void shift_times(real x, real y = 1, real z = 1);

777. (Define Path functions 701) +≡

void Path::shift_times(real x, real y, real z)
{
    for (vector<Point>::iterator iter = points.begin(); iter != points.end(); ++iter)
        (**iter).shift_times(x, y, z);
    return;
}

778. Point argument.

---

[LDF 2003.01.19.] Added this function.

(Declare Path functions 700) +≡

virtual void shift_times(const Point &p);

779. (Define Path functions 701) +≡

void Path::shift_times(const Point &p)
{
    return shift_times(p.get_x(), p.get_y(), p.get_z());
}

780. Applying transformations.

781. Multiplying by a Transform.

(Declare Path functions 700) +≡

virtual Transform operator*(const Transform &t);

782. (Define Path functions 701) +≡

Transform Path::operator*(const Transform &t)
{
    for (vector<Point>::iterator iter = points.begin(); iter != points.end(); iter++) (**iter) *= t;
    return t;
}

783. Applying transform to points.

(Declare Path functions 700) +≡

virtual void apply_transform();
784. (Define Path functions 701) +≡
    void Path::apply_transform()
    {
      for (vector<Point>::iterator iter = points.begin(); iter != points.end(); iter++)
        (**iter).apply_transform();
      return;
    }

785. Projection. [LDF 2002.12.20] TO DO: Make this function virtual!
    bool project(const Focus &f, const unsigned short proj, real factor);

786. (Define Path functions 700) +≡
    bool Path::project(const Focus &f, const unsigned short proj, real factor)
    {
      for (vector<Point>::iterator iter = points.begin(); iter != points.end(); iter++) {
        if (!(**iter).project(f, proj, factor)) {
          cerr << "ERROR! InPath::project()" << "Point::project() returned false."
               << "Returning false."
               << flush;
          return false;
        }
      }
      return true;
    }

787. Functions for lines.

788. Alignment with an axis.

789. For lines.

790. No assignment. (Axis argument only). [LDF 2002.11.03] This function returns the Transform
that would transform Path such that it would come to lie on the major axis indicated by its argument (by
default, the z-axis). It does not actually perform the transformation on the Path.

    Transform align_with_axis(const char axis = 'z') const;

[Log]
[LDF 2002.11.03] Changed char argument to const char.
791.  (Define Path functions 701) +≡
    Transform Path :: align_with_axis(const char axis) const
    {
        Transform t;
        if (~get_line_switch()) {
            cerr ≡ "ERROR! In Path::align_with_axis().\n            "Path.in not a line.\n            Returning INVALID_TRANSFORM.\n            "\n            return INVALID_TRANSFORM;
        }
        Point p0 (*points[0]);
        Point p1 (*points[1]);
        return t.align_with_axis(p0, p1, axis);
    }

792.  With assignment.  [LDF 2002.11.03]  This function should never be called with the bool argument assign ≡ false. It won’t cause any harm, though, since it will just call the const version above.

[LDF 2002.11.03]  Added this function.
[LDF 2003.07.18]  Changed, so that is_linear() is used, rather than get_line_switch(). Also, changed the way Transform t is set. The latter change was necessary, because GCC 3.3 couldn’t compile this file the way it was before.

(Declare Path functions 700) +≡
    Transform align_with_axis(bool assign, const char axis = 'z');

793.  (Define Path functions 701) +≡
    Transform Path :: align_with_axis(bool assign, const char axis)
    {
        if (~is_linear()) {
            cerr ≡ "ERROR! In Path::align_with_axis().\n            "Path in not a linear.\n            Returning INVALID_TRANSFORM.\n            "\n            return INVALID_TRANSFORM;
        }
        Transform t;
        t.align_with_axis(get_point(0), get_last_point(), axis);
        if (assign ≡ false) {
            cerr ≡ "WARNING! In Path::align_with_axis():\n            "Don’t call this function with the assign=false.\n            "It won’t cause any harm, though.\n            "Continuing.\n            "\n            flush;
            return t;
        }
        return (*this == t);
    }
794. For non-lines, (Point and axis arguments). [LDF 2002.11.03.] This function finds the transformation that would align the line segment $p_0\overrightarrow{p_1}$ with the major axis indicated by the axis argument, and applies it to *this. $p_0$ and $p_1$ are not changed.

[LDF 2002.11.03.] Changed Point arguments to const Point & and char argument to const char.

(Declare Path functions 700) +≡
Transform align_with_axis(const Point &, const Point &, const char axis);

795. (Define Path functions 701) +≡
Transform Path::align_with_axis(const Point &, const Point &, const char axis = 'z')
{
    Transform t;
    t.align_with_axis(p0, p1, axis);
    return (*this += t);
}

796. Adding Points to Paths.

797. With assignment.

[LDF 2002.4.6.] Added this function. Currently, it doesn’t return a Path. If it turns out that it would be useful to return *this, I can change it.

(Declare Path functions 700) +≡
void operator+=(const Point &pt);

798. (Define Path functions 701) +≡
void Path::operator+=(const Point &pt){ points.push_back ( create_new < Point > (pt) );
    return; }

799. Without assignment.

[LDF 2002.4.6.] Added this function.

(Declare Path functions 700) +≡
Path operator+(const Point &pt) const;
800.  
(Define Path functions 701) +Ξ
Path Path::*operator+(const Point &pt) const { Path pa(*this); pa.points.push_back ( create_new < Point > (pt) );
    return pa; }

801.  Adding connectors to Paths.

[Log]  [LDF 2003.02.09] Added this function.

(Declare Path functions 700) +Ξ
void operator+=(const string s);

802.  
(Define Path functions 701) +Ξ
void Path::*operator+=(const string s)
{
    connectors.push_back(s);
    return;
}

803.  Concatenating Paths.

804.  Versions using "&".

805.  With assignment.  This function appends the Path argument pa to *this.

[Log]  [LDF 2002.4.6] Added this function.
[Log]  [LDF 2002.11.03] Made non-inline.

(Declare Path functions 700) +Ξ
void operator&=(const Path &pa);

806.  
(Define Path::operator&=(const Path &pa){
    if (is_cycle()) || pa.is_cycle()) /* Return if either one of the Paths is a cycle. */
    {
        cerr << "ERROR! _In Path::operator=(Path&).\n" << 
            "One of the Paths is a cycle. Can’t concatenate.\n\n" << "Returning *this.\n\n";
        return;
    }
    string last_connector;
807. [LDF 2002.4.6] If there isn’t an explicit connector for every pair of Points in this Points, then we have to fill up connectors so that there are enough. Otherwise, the “&” will be at the wrong place. We don’t have to worry about the connectors for *pa*.

(Define Path functions 701) ≡
if (connectors.size() == 0) last_connector = "--";
else last_connector = connectors.back();
while (connectors.size() < points.size() - 1) connectors.push_back(last_connector);

808. [LDF 2002.4.6] If the Paths don’t touch, they are joined using “..” instead of “&”. This mimics the behavior of METAFONT.

[Log]
[LDF 2002.11.03] Now using *(points.back()) instead of get_point(points.size() - 1).

(Define Path functions 701) ≡
if (*points.back()) != pa.get_point()) {
  cerr << "ERROR\u001b[2mIn Path::operator&(Path&)": Paths don’t touch.\n" << "Using:. . .\u001b[2m\".join\" instead of\". \n" << flush;
  connectors.push_back("..");
} else connectors.push_back("&");
for (vector<Point *>::const_iterator iter = pa.points.begin(); /* [LDF 2002.4.6] Copy the Points in pa and put the copies onto points. */ iter != pa.points.end(); iter++) points.push_back (create_new < Point > (*iter) );

809. [LDF 2002.4.6] Put the connectors from *pa* onto the new Path. Since they’re strings, and not pointers, we don’t have to copy them. I tested this to make sure it’s true. I don’t know how strings are implemented, but they seem to be handled like string literals.

(Define Path functions 701) ≡
for (vector<string>::const_iterator iter = pa.connectors.begin(); iter != pa.connectors.end(); iter++) {
  connectors.push_back(*iter);
} return;

810. Without assignment.

[Log]
[LDF 2002.4.6] Added this function. It behaves the way the operator “&” does in METAFONT.

(Declare Path functions 700) ≡
Path operator&(const Path &pa) const;
811.  
(Define Path functions 701) +≡
   Path Path::operator&(const Path &pa) const
   {  
      Path r(*this);
      r & = pa;
      return r;
   }

812.  Appending with a connector argument.  [LDF 2002.4.7] It would not have been possible to 
   specify a connector if I'd defined this function as a binary operator, e.g., operator+=( ), so I've made it 
   a named function. It can be useful when, for instance, rotation causes two Points, which should be identical, 
   to differ by a small amount, like 1/10,000 in one coordinate. This has actually happened, which is why I've 
   added this function. METAFONT can recover gracefully by using ".." instead of "&" to connect the paths, 
   but it issues an error message and stops to wait for a response. Using this function can help to avoid such 
   problems.

   [LDF 2002.4.7.] Added this function.

(Declare Path functions 700) +≡
   Path append(const Path &pa,string connector = "--",bool assign = true);

813.  
(Define Path functions 701) +≡
   Path Path::append(const Path &pa,string connector, bool assign) { Path r(*this);
      string last_connector;  /* [LDF 2002.4.6.] If there isn’t an explicit connector for every pair of
      Points in this-points, then we have to fill up connectors so that there are enough. Otherwise, 
      the "&" will be at the wrong place. We don’t have to worry about the connectors for pa. */
      if (r.connectors.size() ≥ 0) last_connector = "--";
      else last_connect = r.connectors.back();
      while (r.connectors.size() < r.points.size() - 1) r.connectors.push_back(last_connect);
      r.connectors.push_back (connector);
      /* [LDF 2002.4.6.] Copy the Points in pa and put the copies onto points. */
      for (vector(Point *): const iterator = pa.points.begin(); iter ≠ pa.points.end(); iter++)
         r.points.push_back (create_new < Point > (*iter) );

814.  Put the connectors from pa onto the new Path. Since they’re strings, and not pointers, we don’t 
   have to copy them. I tested this to make sure it’s true. I don’t know how strings are implemented, but 
   they seem to be handled like string literals. [LDF 2002.4.6.]

(Define Path functions 701) +≡
   for (vector(string): const_iterator = pa.connectors.begin(); iter ≠ pa.connectors.end(); iter++) {
      r.connectors.push_back (*iter);
   }  
   if (assign = true) *this = r;
   return r; }

815.  Drawing and filling.

816.  Draw.
§817  PATH VERSIONS


[LDF 2002.10.07.] Added code for handling draw_color and fill_color.

(Declare Path functions 700) ++

virtual void draw(const Color &ddraw_color = *Colors::default_color, const string ddashed = "", const string ppen = "", Picture &picture = current_picture, bool arrow = false) const;

819.

(Define Path functions 701) ++

void Path::draw(const Color &ddraw_color, const string ddashed, const string ppen, Picture &picture, bool arrow) const{ bool DEBUG = false; /* true */
if (DEBUG) cout << "Entering Path::draw().\n" << flush;
if (points.size() == 0)
/* LDF 2002.09.27. Added this error handling code. If the Path is empty, don’t draw it. */
{
  cerr << "WARNING! Path::draw() does not contain any Points.\n" << "Not doing anything.\n" << flush;
  return;
}
Path *p = create_new < Path > (*this);
p->fill_color = DRAW;
p->arrow = arrow;
if (DEBUG)
  cout << "draw_color.get_use_name() == " << ddraw_color.get_use_name() << endl << flush;
if (ddraw_color.get_use_name() == false) {
if (DEBUG) cout << "Allocating memory for Color.\n" << flush;
Color *c = create_new < Color > (0);
  *c = ddraw_color;
p->draw_color = c; }
else {
  if (DEBUG) cout << "draw_color.get_name() == " << ddraw_color.get_name() << endl << flush;
p->draw_color = &ddraw_color;
}
p->fill_color = Colors::background_color;
p->dashed = ddashed;
p->pen = ppen;
picture += static_cast<Shape *>(p);
/* LDF 2002.11.03. Changed dynamic_cast() to static_cast(). */
if (DEBUG) cout << "Exiting Path::draw().\n" << flush;
return; }

820. Picture argument first. [LDF 2002.09.17.] Added this function. It’s convenient for when I want to pass a Picture argument. If I want to declare it inline, I must define it within the declaration of class Path. Otherwise, it causes a compiler error. I’ve decided to declare it non-inline, and hope that the compiler will inline it by itself.

\[
\text{void Path::draw(Picture &picture, const Color &\textcolor{#00558b}{\textit{ddraw\_color}} = Colors::\textcolor{#00558b}{\textit{default\_color}}, string \textcolor{#00558b}{\textit{ddashed}} = "", string \textcolor{#00558b}{\textit{ppen}} = ", bool aarrow = false) const;}
\]

821.

822. Point versions. Declared in points.web, but must be defined here, because Path is an incomplete type here.

823. Normal version.

824. Picture argument first. [LDF 2002.09.17.] Added this function. It’s convenient for when I want to pass a Picture argument.

825. Draw arrow.

826. Path versions.
827. Normal version.

Log

[LDF 2003.01.15.] Added this function.

(Declare Path functions 700) +≡

virtual void drawarrow (const Color &ddraw_color = *Colors::default_color, string 
     dashed = "", string ppen = "", Picture &picture = current_picture) const;

828.

(Define Path functions 701) +≡

void Path :: drawarrow (const Color &ddraw_color, string dashed, string ppen, Picture &picture)
     const

{ 
    draw (ddraw_color, dashed, ppen, picture, true);
}

829. Picture argument first.

Log

[LDF 2003.01.15.] Added this function.

(Declare Path functions 700) +≡

virtual void drawarrow (Picture &picture, const Color &ddraw_color = *Colors::default_color, string 
     dashed = "", string ppen = "") const;

830.

(Define Path functions 701) +≡

void Path :: drawarrow (Picture &picture, const Color &ddraw_color, string dashed, string ppen)
     const

{ 
    draw (picture, ddraw_color, dashed, ppen, true);
}

831. Point versions.

832. Normal version.

Log

[LDF 2003.01.15.] Added this function.
[LDF 2003.06.03.] Made drawarrow () const.

(Define Point functions 330) +≡

Path Point :: drawarrow (const Point &p, const Color &ddraw_color, string dashed, string 
     ppen, Picture &picture) const

{ 
    Path pa (*this, p);
    pa.drawarrow (ddraw_color, dashed, ppen, picture);
    return pa;
}
833. **Picture argument first.**

- [LDF 2003.01.15.] Added this function.
- [LDF 2003.06.03.] Made `drawarrow()` const.

<Define Point functions 330> +≡

```cpp
Path Point::drawarrow(Picture &picture, const Point &p, const Color &ddraw_color, string dashed, string ppen) const
{
    return drawarrow(p, ddraw_color, dashed, ppen, picture);
}
```

834. **Draw help.**

- [LDF 2002.06.10.] Changed the way the default arguments are handled. The way it was didn’t work for both versions, i.e., the `Path` version and the `Point` version.
- [LDF 2002.4.8.] Added this section. !! It would be nice to do something to make sure that the help lines and curves are not drawn over by filling commands. Maybe it will be possible to take care of this when I implement the hidden surface algorithm in `output()`. [LDF 2002.11.03.] I could have help lines outputted last, if I put them on a vector of their own.
- [LDF 2003.07.13.] Made all versions of `draw_help()` const.

835. **Path versions.** [LDF 2002.12.20.] ?? Could these functions be const?

836. **Normal version.** [LDF 2002.4.8.] Added this function.

<Declare Path functions 700> +≡

```cpp
void draw_help(const Color &ddraw_color = *help_color, string dashed = help_dash_pattern, string ppen = "", Picture &picture = current_picture) const;
```

837.

<Define Path functions 701> +≡

```cpp
void Path::draw_help(const Color &ddraw_color, string dashed, string ppen, Picture &picture) const
{
    if (do_help_lines == false) return;
    draw(ddraw_color, dashed, ppen, picture);
}
```

838. **Picture argument first.**

<Declare Path functions 700> +≡

```cpp
void draw_help(Picture &picture, const Color &ddraw_color = *help_color, string dashed = help_dash_pattern, string ppen = "") const;
```
839.
(Define Path functions 701) +≡
void Path :: draw_help(Picture &picture, const Color &ddraw_color, string dashed, string ppen) const
{
    draw_help(ddraw_color, dashed, ppen, picture);
}

840. Point versions.

841. Normal version. [LDF 2002.4.8.] Added this function. Declared in points.web, but must be
defined here, because Path is an incomplete type here.
(Define Point functions 330) +≡
Path Point :: draw_help(const Point &pt,const Color &ddraw_color, string dashed, string
ppen, Picture &picture) const
{
    Path pa(*this, pt);
    pa.draw_help(ddraw_color, dashed, ppen, picture);
    return pa;
}

842. Picture argument first. [LDF 2002.09.17.] Added this function. It's convenient for when I want
to pass a Picture argument.
[LDF 2002.10.26.] Declared in points.web. Must be defined here, because Path is an incomplete type
there.
(Define Point functions 330) +≡
Path Point :: draw_help(Picture &picture, const Point &pt,const Color &ddraw_color, string
dashed, string ppen) const
{
    return draw_help(pt, ddraw_color, dashed, ppen, picture);
}

843. Fill.
!! [LDF 2003.02.02.] Filling doesn't use a pen!! Change everywhere!!.


Log

[Declare Path functions 700) +≡
void fill(const Color &fill_color = *Colors::default_color, Picture &picture = current_picture) const;
845. *(Define Path functions 701) +≡

    void Path::fill(const Color &fill_color, Picture &picture) const {
        bool DEBUG = false;
        /* true */
        if (DEBUG) cout << "Entering Path::fill().\n" << flush;
        if (points.size() == 0)
            /* LDF 2002.09.27. Added this error handling code. If the Path is empty, don’t fill it. */
            {
                cerr << "WARNING! Null Path::fill().\n" << "Path doesn’t contain any points.\n" << "Not doing anything.\n" << flush;
                return;
            }
        Path *p = create_new < Path > (0);
        *p = *this;
        p->fill_draw_value = FILL;
        if (DEBUG)
            cout << "fill_color.get_use_name() = \n" << fill_color.get_use_name() << endl << flush;
        if (fill_color.get_use_name() == false) {
            if (DEBUG) cout << "Allocating memory for Color.\n" << flush;
            Color *c = create_new < Color > (0);
            *c = fill_color;
            p->fill_color = c;
        } else {
            if (DEBUG) cout << "fill_color.get_name() = \n" << fill_color.get_name() << endl << flush;
            p->fill_color = &fill_color;
        }
        p->pen = "";
        p->dashed = "";
        p->draw_color = Colors::background_color;
        picture += static_cast<Shape *>(p);
        if (DEBUG) cout << "Exiting Path::fill().\n" << flush;
        return;
    }


    (Declare Path functions 700) +≡

    void fill(Picture &picture, const Color &fill_color = Colors::default_color);
§847.  (Define Path functions 701) +≡

void Path::fill(Picture &picture, const Color &fillcolor)
{
    fill(fillcolor, picture);
}

§848. Filldraw.

[LDF 2002.03.25] [LDF 2002.11.03] Revised the following text. At the present time, filldraw() differs from the filldraw command in METAFONT and METAPOST. In the default case, the outline is drawn in the default color (currently black) and the Path is filled with the background color (currently white by default). This makes it possible to hide objects that are behind the Path by using the painter’s algorithm when rendering. If you want a Path to be filled with another color, you will have to use explicit arguments for draw_color and fill_color. Either or both of these can be "", which causes the default color (currently black) to be used. Of course, plain fill() followed by plain draw() will produce the same result.

[LDF 2003.07.16] Made both versions const.


[LDF 2002.10.07] Added code for handling draw_color and fill_color.

(Declare Path functions 700) +≡

void filldraw(const Color &draw_color = *Colors::default_color, const Color &fill_color = *Colors::background_color, string ddashed = "", string pp'en = "", Picture &picture = current_picture) const;
850.  
(Define Path functions 701) \( \equiv \) 
void Path::filldraw(const Color &draw_color, const Color &fill_color, string dashed, string ppen, Picture &picture) const { bool DEBUG = false;  
if (DEBUG) cout << "Entering Path::filldraw().\n" << flush;  
if (points.size() == 0)  
  /* LDF 2002.09.27. Added this error handling code. If the Path is empty, don’t filldraw it */  
  {  
    cerr << "WARNING! In Path::filldraw():\n" << "Path doesn’t contain any points.\n" << "Not doing filldrawing.\n\n" << flush;  
    return;  
  }  
Path *p = create_new < Path > (0);  
*p = *this;  
p->draw_value = FILLDRAW;  
if (DEBUG)  
  cout << "draw_color.get_use_name() ==\n" << draw_color.get_use_name() << endl << flush;  
if (DEBUG)  
  cout << "fill_color.get_use_name() ==\n" << fill_color.get_use_name() << endl << flush;  
if (draw_color.get_use_name() == false)  
  if (DEBUG) cout << "Allocating\nmemory for\nColor.\n" << flush;  
Color *c = create_new < Color > (0);  
*c = draw_color;  
p->draw_color = c;  
else  
  if (DEBUG) cout << "draw_color.get_name() ==\n" << draw_color.get_name() << endl << flush;  
  p->draw_color = &draw_color;  
}  
if (fill_color.get_use_name() == false)  
  if (DEBUG) cout << "Allocating\nmemory for\nColor.\n" << flush;  
Color *c = create_new < Color > (0);  
*c = fill_color;  
p->fill_color = c;  
else  
  if (DEBUG) cout << "fill_color.get_name() ==\n" << fill_color.get_name() << endl << flush;  
    p->fill_color = &fill_color;  
}  
p->dashed = dashed;  
p->pen = ppen;  
picture += static_cast<Shape *>(p);  
if (DEBUG) cout << "Exiting Path::filldraw().\n" << flush;  
return; }  

851.  Picture argument first.  [LDF 2002.09.17.] Added this function. It’s convenient for when I want to pass a Picture argument.  
(Declare Path functions 700) \( \equiv \) 
void filldraw(Picture &picture, const Color &draw_color = *Colors::default_color, const Color &fill_color = *Colors::background_color, string dashed = "", string ppen = ") const;
§852.  
(Define Path functions 701) +≡
  void Path::filldraw(Picture &picture, const Color &draw_color, const Color &fill_color, string ddashed, string ppens) const
  {
    filldraw(draw_color, fill_color, ddashed, ppens, picture);
  }

§853.  Undraw.

§854.  Path versions.

§855.  Normal version.
(Declare Path functions 700) +≡
  void undraw(string ddashed = "", string ppens = "", Picture &picture = current_picture);

§856.  (Define Path functions 701) +≡
  void Path::undraw(string ddashed, string ppens, Picture &picture)
  {
    if (points.size() == 0)
      /* LDF 2002.09.27. Added this error handling code. If the Path is empty, don’t undraw it. */
      {
        cerr ⩶ "WARNING!\nIn Path::undraw():\n" ⩶ "Path doesn’t contain any points.\n" ⩶ "Not doing anything.\n" ⩶ flush;
        return;
      }
    Path *p = create_new Path (this);
    p->fill_draw_value = UNDRAW;
    p->draw_color = 0;
    p->fill_color = 0;
    p->dashed = ddashed;
    p->pen = ppens;
    picture += static_cast<Shape*>(p);
  }

(Declare Path functions 700) +≡
  void undraw(Picture &picture, string ddashed = "", string ppens = "");

§858.
(Define Path functions 701) +≡
  void Path::undraw(Picture &picture, string ddashed, string ppens)
  {
    undraw(ddashed, ppens, picture);
  }

§859.  Point versions.
860. Normal version. This function is declared in points.web, but must be defined here, because `Path` is an incomplete type here.

Log

[LDF 2002.4.8.] Added this function.
[LDF 2002.11.03.] Changed this function, so that it returns the `Path pa`, instead of `void`.

```cpp
(Path Point functions 330) +≡
Path Point::undraw(const Point &pt, string ddashed, string ppen, Picture &picture)
{
    Path pa(*this, pt);
    pa.undraw(ddashed, ppen, picture);
    return pa;
}
```

861. Picture argument first.

Log

[LDF 2002.09.17.] Added this function. It’s convenient for when I want to pass a `Picture` argument.
[LDF 2002.09.17.] Added this function. It’s convenient for when I want to pass a `Picture` argument.

```cpp
(Path Point functions 330) +≡
Path Point::undraw(Picture &picture, const Point &pt, string ddashed, string ppen)
{
    return undraw(pt, ddashed, ppen, picture);
}
```

862. Unfill.

863. Normal version.

```cpp
(Declare Path functions 700) +≡
void unfill(Picture &picture = current_picture);
```
§864.  
(Define Path functions 701) +=

```cpp
void Path::unfill(Picture &picture){
    if (points.size () == 0)
        /* LDF 2002.09.27. Added this error handling code. If the Path is empty, don't unfill it. */
    {  
        cerr << "WARNING! In Path::unfill():n" << "Path doesn't contain any Points.n" << "Not doing anything.n" << flush;
        return;
    }
    Path *p = create_new < Path > (this);
    p->fill_draw_value = UNFILL;
    p->draw_color = 0;
    p->fill_color = 0;
    p->dashed = "n";
    p->pen = "n";
    picture += static_cast (Shape *)(p); }
```

§865.  Unfilldraw.

§866.  Normal version,  
(Declare Path functions 700) +=

```cpp
void unfilldraw(const Color &ddraw_color = *Colors::background_color, string ddashed = "n", string dpen = "n", Picture &picture = current_picture);
```
867. (Define Path functions 701) +⇒

```cpp
void Path::unfilldraw(const Color &draw_color, string dashed, string ppen, Picture &picture){
  bool DEBUG = false; /* true */
  if (DEBUG) cout << "Entering Path::unfilldraw().\n" << flush;
  if (points.size() == 0) /* LDF 2002.09.27. Added this error handling code. If the Path is empty, don’t unfilldraw it. */
    { err << "WARNING! In Path::unfilldraw():\n" << "Path doesn’t contain any Points.\n" << "Not doing anything.\n" << flush;
      return;
    }
  Path *p = create_new < Path > (this);
  pfill_draw_value = UNFILLDRAW;
  /* LDF 2002.10.07. Added code for handling draw_color and fill_color. Will get rid of this if I do
   actually change it to make it act more like unfilldraw in METAPOST. */
  if (draw_color.get_use_name() == false) {
    Color *c = create_new < Color > (0);
    *c = draw_color;
    pdraw_color = c;
  } else {
    if (DEBUG) cout << "draw_color.get_name()" << endl << flush;
    pdraw_color = &draw_color;
  }
  pfill_color = 0;
  pdashed = dashed;
  ppen = ppen;
  picture += static_cast<Shape *>(p);
  if (DEBUG) cout << "Exiting Path::unfilldraw().\n" << flush;
  return;
}
```

868. Picture argument first. [LDF 2002.09.17.] Added this function. It’s convenient for when I want to pass a Picture argument.

(Declare Path functions 700) +⇒

```cpp
void unfilldraw(Picture &picture, const Color &draw_color = *Colors::background_color, string dashed = "", string ppen = "");
```
869. (Define Path functions 701) \equiv
void Path::unfilldraw(Picture &picture, const Color &ddraw_color, string ddashed, string ppen)
{
  unfilldraw(ddraw_color, ddashed, ppen, picture);
}

870. Labelling.

871. Label.

[Log]
[LDF 2002.03.25.] Added argument dot and changed definition of dotlabel() below so that it just calls label().
[LDF 2003.04.01.] BUG FIX: Got rid of the first argument unsigned int i, and made the third argument short text_short the first argument. Formerly, the Points in Paths were always numbered starting from 0, because the argument text_short was passed to Point::label(), not i. Also changed the following versions of label() and dotlabel(), that call this function.

872. Normal version.

[Log]
[LDF 2003.06.06.] Changed the conditional, where text_short is compared with WORLD_VALUES, PROJ_VALUES, etc. I had to change it, because I've added WORLD_VALUES_X_Y, etc. Now, the conditional tests for VIEW_VALUES_X_Y\leq text_short \leq WORLD_VALUES. Of course, this makes an assumption about the values that are used to signal that coordinate values should be used for the label, but I think it's worth it, to avoid testing text_short against each value individually.
[LDF 2003.07.09.] Made position_string and dot arguments const.

(Declare Path functions 700) \equiv
void label(short text_short = 0, const string position_string = "top", const bool dot = false, Picture &picture = current_picture) const;
873.  
\{Define \textbf{Path} functions 701\} +\equiv 
\begin{verbatim}
void Path::label(short text_short, const string position_string, const bool dot, Picture &picture)
    const
    {
        bool DEBUG = false;  /* true */
        if (DEBUG) cout << "Entering\text::Path::label()" << "\n" << flush;
        if (Label::DQ_LABELS \equiv false) return;
        if (points.size() \equiv 0)
            /* LDF 2002.09.27. Added this error handling code. If the Path is empty, don't label it. */
            {
                cerr << "WARNING! In Path::label():\n" << "Path doesn't contain any Points.\n" << "Not doing anything.\n" << flush;
                return;
            }
        for (vector(Point *)::const_iterator iter = points.begin(); iter \neq points.end(); iter++)
            (**iter).label(text_short, position_string, dot, picture);
            /* LDF 2003.05.06. Changed this conditional. */
        ; /* Do nothing. */
        else ++text_short;
    }
    if (DEBUG) cout << "Exiting\text::Path::label()" << "\n" << flush;
\end{verbatim}

874.  \textbf{Picture argument first}.  [LDF 2002.09.17.]  Added this function. It's convenient for when I want to pass a \textbf{Picture} argument.

[LDF 2003.07.09.]  Made \textit{position\_string} and \textit{dot} arguments \textbf{const}.

\begin{verbatim}
\end{verbatim}

875.  
\{Define \textbf{Path} functions 701\} +\equiv 
\begin{verbatim}
void Path::label(Picture &picture, short text_short = 0, const string position_string = "top", const bool dot = false) const;
\end{verbatim}

876.  \textbf{Dotlabel}.

Log

[LDI 2003.07.09.] Made text_short and position_string arguments const.

(Declare Path functions 700) +≡
void dotlabel(const short text_short = 0, const string position_string = "top", Picture &picture = current_picture) const;

878.

(Define Path functions 701) +≡
void Path::dotlabel(const short text_short, const string position_string, Picture &picture) const
{
label(text_short, position_string, true, picture);
}

879. Picture argument first.

Log

[LDI 2002.09.17.] Added this function. It's convenient for when I want to pass a Picture argument.
[LDI 2003.07.09.] Made text_short and position_string arguments const.

(Declare Path functions 700) +≡
void dotlabel(Picture &picture, const short text_short = 0, const string position_string = "top")
const;

880.

(Define Path functions 701) +≡
void Path::dotlabel(Picture &picture, const short text_short, const string position_string) const
{
dotlabel(text_short, position_string, picture);
}

881. Outputting.

882. Extract. This is needed for outputting a Picture.
[LDI 2003.01.31.] ?? Do I need to call Point::project() on the Points here and in Path::project()?

Log

[LDI 2002.09.17.] Added const Focus &f argument and error handling code. Now, if any of the Points on vector(Point *) points cannot be projected onto the projection plane using the Focus f, the Path is not put onto the vector(Shape *) Picture::elements, and consequently never reaches Picture::output() and Path::output().
[LDI 2003.06.09.] Rewrote this function. It now calls Point::extract() instead of calling apply_transform() and project() on the Points directly. This makes much more sense, since any changes to Point::extract() would otherwise not have been applied to Points on Paths.

(Declare Path functions 700) +≡
vector(Shape *) extract(const Focus &f, const unsigned short proj, real factor):
883.  
(Define Path functions 701) \(\equiv\) 

```cpp
vector<Shape *> Path::extract(const Focus &f, const unsigned short proj, real factor) 
{
    bool DEBUG = false;  // true */
    vector<Shape *> v;
    int i = 0;
    for (vector<Point *>::iterator iter = points.begin(); iter != points.end(); iter++) {
        if (DEBUG) cout << "Point:
          <i++ << ":" << endl;
        v = (**iter).extract(f, proj, factor);
        if (DEBUG) cout << endl;
        if (v.size() == 0)  // Point::extract() failed. LDF 2003.05.09. */
            return v;
    }
    vector<Shape *> r;
    r.push_back(this);
    return r;
}
```

884. Set extremes. [LDF 2002.09.18] set_extremes() doesn't check that the projective_coordinates in all of the Points on the Path are valid. This is done already in project() and extract(), so I don't think it's necessary to repeat it here, since extract() (which invokes project()), is called before set_extremes() in Picture::output(). The latter is the only place where set_extremes() is invoked.


(Declare Path functions 700) \(\equiv\) 

```cpp
virtual bool set_extremes();
```

885.  
(Define Path functions 701) \(\equiv\) 

```cpp
bool Path::set_extremes() { bool DEBUG = false;  // true */
    if (DEBUG)
        cout << "Entering
          Path::set_extremes()" << endl << flush;  // If there are no Points on
    if (points.size() <= 0) {
        cerr << "ERROR! In Path::set_extremes():\n          " << points.size() << " Points\n          Setting projective_extremes to INVALID_REAL and returning.\n          " << flush;
        projective_extremes = INVALID_REAL;
        return false;
    }
```
§886. [LDF 2002.09.18.] Added this routine. Set the minimum values to MAX_REAL and the maximum values to -MAX_REAL. This way, any valid perspective coordinates will replace them on the first iteration of the for loop.

[LDF 2002.09.18.] I had some difficulty debugging this because instead of using -MAX_REAL, I defined and used MIN_REAL = numeric_limits(real) :: min(). However, this isn’t the negative real with the largest magnitude, but the smallest positive real.

(Define Path functions 701)  +

projective_extremes[0] = MAX_REAL; /* Minima. */
projective_extremes[2] = MAX_REAL;
projective_extremes[4] = MAX_REAL;
projective_extremes[1] = -MAX_REAL; /* Maxima. */
projective_extremes[3] = -MAX_REAL;
projective_extremes[5] = -MAX_REAL;

for (vector(Point *):: iterator iter = points.begin(); iter != points.end(); ++iter) {
  if (DEBUG) {
    cout << "min.x["u"] = " << projective_extremes[0] << endl;
    cout << "max.x["u"] = " << projective_extremes[1] << endl;
    cout << "x["world"] = " << (**iter).get.x() << endl;
    cout << "x["persp"] = " << (**iter).get.x(’p’,false,false) << endl;
    cout << "min.y["u"] = " << projective_extremes[2] << endl;
    cout << "max.y["u"] = " << projective_extremes[3] << endl;
    cout << "y["world"] = " << (**iter).get.y() << endl;
    cout << "y["persp"] = " << (**iter).get.y(’p’,false,false) << endl;
    cout << "min.z["u"] = " << projective_extremes[4] << endl;
    cout << "max.z["u"] = " << projective_extremes[5] << endl;
    cout << "z["world"] = " << (**iter).get.z() << endl;
    cout << "z["persp"] = " << (**iter).get.z(’p’,false,false) << endl;
  }

  projective_extremes[0] = min(projective_extremes[0], (**iter).get.x(’p’,false,false)); /* Min x */
  projective_extremes[1] = max(projective_extremes[1], (**iter).get.x(’p’,false,false)); /* Max x */
}

if (DEBUG) {
  for (int i = 0; i < 6; i++)
    cout << "projective_extremes[" << i << "] = " << projective_extremes[i] << endl << flush;
}
887. [LDF 2002.09.18.] Added this error handling code. There is a remote chance that a valid Point could have a coordinate $\equiv$ MAX_REAL or $\pm$MIN_REAL, however, it is virtually impossible that it would be projectable. If it’s the $x$ or $y$-coordinate, it would probably lie outside the limits defined for the invocation of Picture::output(), and if it was the $z$, it would either be behind the Focus or so far away as to be practically invisible. I believe that this is the case, even though the $z$-coordinates are made smaller by applying the equation $z_p = z/(z + p)$.

```cpp
(Define Path functions 701) +\equiv
  for (int i = 0; i < 6; i += 2) {
    if (projective_extremes[i] $\equiv$ MAX_REAL /* Minima */
        $$i.$$ projective_extremes[i + 1] $\equiv$ -MAX_REAL) /* Maxima */
    {
      if (DEBUG) {
        cout $<$ "i.=u" $<$ i $<$ endl $<$ flush;
        cout $<$ "projective_extremes[] $<$ i $<$ "].=u" $<$ projective_extremes[i] $<$ endl $<$ flush;
      } // comment
      cerr $<$ "ERROR\u2018In\u2019Path::set_extremes()$n" $<$ "maxima\_and\_minima\_could\_not\_be\_set\_prop\erly\_u" $<$ "Setting\_them\_all\_to\_INVALID\_REAL\_and\_returning\_false.\n" $<$ flush;
      projective_extremes = INVALID_REAL;
      return false;
      break;
    }
  }
  if (DEBUG) cout $<$ "Exiting\u2018Path::set_extremes()$n" $<$ endl $<$ flush;
  return true; }

888. Get extremes. [LDF 2002.09.18.] Added this function. Any code that calls get_extremes() must ensure that project() has been invoked first.

```cpp
(Declare Path functions 700) +\equiv
  virtual inline const valarray<real) get_extremes( ) const
  {
    return projective_extremes;
  }
```

889. Get minimum $z$. [LDF 2003.05.16.] Added this function.

```cpp
(Declare Path functions 700) +\equiv
  virtual real get_minimum_z( ) const;
```
§890  3DLDF-1.15.1  GET MINIMUM Z  237

890.  
(Define Path functions 701) +≡  
real Path::get_minimum_z() const  
{  
  bool DEBUG = false;  /* true */  
  if (DEBUG) {  
    cout << "Entering Path::get_minimum_z()" << endl << flush;  
    cout << "projective_extremes[4]_u=0" << projective_extremes[4] << endl << flush;  
  }  
  return projective_extremes[4];  
}

891. Get maximum z.  [LDF 2002.09.17.] Added this function.  
(Declare Path functions 700) +≡  
virtual real get_maximum_z() const;

892.  
(Define Path functions 701) +≡  
real Path::get_maximum_z() const  
{  
  bool DEBUG = false;  /* true */  
  if (DEBUG) {  
    cout << "Entering Path::get_maximum_z()" << endl << flush;  
    cout << "projective_extremes[5]_u=0" << projective_extremes[5] << endl << flush;  
  }  
  return projective_extremes[5];  
}

893. Get mean z.  [LDF 2003.05.16.] Added this function.  
(Declare Path functions 700) +≡  
virtual real get_mean_z() const;

894.  
(Define Path functions 701) +≡  
real Path::get_mean_z() const  
{  
  return ((projective_extremes[4] + projective_extremes[5])/2);  
}

895. Suppress output.  [LDF 2002.09.18.] Added this function. It’s needed because trying to erase a Shape * from elements in Picture::output() causes a memory fault.  
(Declare Path functions 700) +≡  
virtual void suppress_output();
896. (Define Path functions 701) +≡
    void Path::suppress_output()
    {
        do_output = false;
    }

897. Unsuppress output. [LDF 2002.09.18] Added this function. It’s needed because trying to erase a Shape * from elements in Picture::output() causes a memory fault.
    (Declare Path functions 700) +≡
    virtual void unsuppress_output();

898. (Define Path functions 701) +≡
    void Path::unsuppress_output()
    {
        do_output = true;
    }

899. Output. [LDF 2002.09.17] Removed error checking code to extract(). Now output() assumes that all of the Points in points can be projected using focus. If they can’t be, extract() will already have ensured that the Path is not on vector(Shape *) elements in the Picture.

    [LDF 2003.01.15.] Added code for writing “drawarrow” to out_stream, if arrow ≡ true. This is for the drawarrow() functions for Path and Point that I’ve added today.

    (Declare Path functions 700) +≡
    virtual void output();

900. (Define Path functions 701) +≡
    void Path::output() { bool DEBUG = false; /* true */
        if (DEBUG) cout ≡ "Entering Path::output()" ≡ "\n" ≡ flush;
        if (do_output && ishape)
            if (DEBUG) cout ≡ "In Path::output(): do_output == false." ≡ flush;
            return;
    }
§901. [LDF 2002.09.27.] Added this error handling code. If the Path is empty, don’t output it. This code should never be reached, because the case of a Path containing no Points should be caught in Path::draw() and the other drawing and filling commands. If it should reach set_extremes() and Picture::output(), which also shouldn’t be possible, they would catch it, too.

(Define Path functions 701 ) +≡

\[
\text{if (points.size() == 0) } \{
\text{cerr }\ll \text{ "THIS CAN’T HAPPEN! In Path::output():\n"
}\ll \text{ "This code should never be reached.\n"
}\ll \text{ "However, I may be able to recover.\n"
}\ll \text{ "Type <RETURN> to continue.\n"
}\ll \text{ flush; }
\text{getchar();
}\text{cerr }\ll \text{ "WARNING! In Path::output():\n"
}\ll \text{ "Path doesn’t contain any Points.\n"
}\ll \text{ "Not doing anything.\n"
}\ll \text{ flush; }
\text{return;}}
\]
902.
(Define Path functions 701) \( \equiv \)

```cpp
vector<Point> &iterator point_iter = points.begin();
vector<string> &iterator connector_iter = connectors.begin();

string connector_string;

if (connectors.size() > 0) connector_string = *connector_iter++;
else connector_string = "--";
if (fill_draw_value \equiv DRAW) {
    if (DEBUG) cout \ll "Drawing. \n" \ll flush;
    if (arrow \equiv true) out_stream \ll "drawarrowu" \ll ++point_iter++;
    else out_stream \ll "drawu" \ll ++point_iter++;
    (Output Path 907) /* [LDF 2002.09.26] Comparing pointers seems to work here. I think it
    should, but I wasn't sure that it really would. */
    if (draw_color \(!\equiv Colors::default_color) {
        out_stream \ll "\nwithcoloru" \ll *draw_color;
    }
    if (dashed \(!\equiv "\n") out_stream \ll "\ndashedu" \ll dashed;
    if (pen \(!\equiv "\n") out_stream \ll "\nwithpenu" \ll pen;
    out_stream \ll ";\n" \ll flush;
}
else if (fill_draw_value \equiv FILL) {
    if (DEBUG) cout \ll "Filling. \n" \ll flush;
    out_stream \ll "fillu" \ll ++point_iter++;
    (Output Path 907)
    if (fillcolor \(!\equiv Colors::default_color) out_stream \ll "\nwithcoloru" \ll *fillcolor;
    out_stream \ll ";\n" \ll flush;
}
else if (fill_draw_value \equiv FILLDRAW) {
    if (DEBUG) cout \ll "Filldrawing. \n" \ll flush;
    if (draw_color \equiv fillcolor) {
        out_stream \ll "filldrawu" \ll ++point_iter++;
        (Output Path 907)
        if (draw_color \(!\equiv Colors::default_color) out_stream \ll "\nwithcoloru" \ll *draw_color;
        if (dashed \(!\equiv "\n") out_stream \ll "\ndashedu" \ll dashed;
        if (pen \(!\equiv "\n") out_stream \ll "\nwithpenu" \ll pen;
        out_stream \ll ";\n" \ll flush;
    }
    else /* We have two different colors, so we have to fill once and draw once. */
    {
        out_stream \ll "\nfillu" \ll ++point_iter++;
        (Output Path 907)
        out_stream \ll "\nwithcoloru" \ll *fillcolor;
        out_stream \ll ";\n" \ll flush;
        point_iter = points.begin();
    }
```

903.

[LD 2002.06.10.] Added the code in this section. It fixes a bug. If it's not done, then the correct connectors are not output when the Path is output the second time.

```cpp
(Define Path functions 701) +\equiv
   connector_iter = connectors.begin();
   if (connectors.size() > 0) connector_string = *connector_iter++;
   else connector_string = "--";

904.

(Define Path functions 701) +\equiv
   if (arrow == true) out_stream <"drawarrow\" <**point_iter++;
   else out_stream <"draw\" <**point_iter++;
   (Output Path 907)
   if (draw_color != Colors::default_color) out_stream <"withcolor\" <draw_color;
   if (dashed != ") out_stream <"dashed\" <dashed;
   if (pen != ") out_stream <"withpen\" <pen;
   out_stream <";\n" <flush; } }
   else if (fill_draw_value == UNDRAW) {
      if (DEBUG) cout <"Undrawing.\n" <flush;
      out_stream <"undraw\" <**point_iter++;
      (Output Path 907)
      if (dashed != ") out_stream <"dashed\" <dashed;
      if (pen != ") out_stream <"withpen\" <pen;
      out_stream <";\n" <flush;
   }
   else if (fill_draw_value == UNFILL) {
      if (DEBUG) cout <"Unfilling.\n" <flush;
      out_stream <"unfill\" <**point_iter++;
      (Output Path 907)
      out_stream <";\n" <flush;
   }
```
Filldraw case.

[LDF 2003.03.25.] Changed this section, so that the outline of the Path is drawn, if draw_color ≠ Colors::background_color.

(Define Path functions 701) +≡
else
if (fill_draw_value ≡ UNFILLDRAW) {
    if (DEBUG) out ≡ "Unfilldrawing.\n" ≡ flush;
    if (draw_color ≡ Colors::background_color) {
        out_stream ≡ "unfilldraw\n" ≡ *point_iter++;
        // Output Path 907
        if (dashed ≠ "") out_stream ≡ "\ndashed\n" ≡ dashed;
        if (pen ≠ "") out_stream ≡ "\nwithpen\n" ≡ pen;
        out_stream ≡ ";\n" ≡ flush;
    }
    else {
        out_stream ≡ "unfill\n" ≡ *point_iter++;
        // Output Path 907
        out_stream ≡ ";\n" ≡ flush;
        point_iter = points begin();
        connector_iter = connectors.begin();
        if (connectors.size() > 0) connector_string = *connector_iter++;
        else connector_string = "--";
        out_stream ≡ "draw\n" ≡ *point_iter++;
        // Output Path 907
        if (draw_color ≠ Colors::default_color) out_stream ≡ "\nwithcolor\n" ≡ *draw_color;
        if (dashed ≠ "") out_stream ≡ "\ndashed\n" ≡ dashed;
        if (pen ≠ "") out_stream ≡ "\nwithpen\n" ≡ pen;
        out_stream ≡ ";\n" ≡ flush;
    }
} /* End of UNFILLDRAW case. [LDF 2003.03.25.] */
906. Default case. [LDF 2003.03.25.]

(Define Path functions 701) +\n    else /* Use DRAW as default. [LDF 2003.03.25.] */
    {
        cin << "WARNING! Invalid fill_draw_value: " << fill_draw_value << ". Using " << draw"\n" << flush;
    #if 0 /* !!! Define a class for information on the run state. */
    if (!Run_State::non_stop) getchar();
    #endif
        if (arrow) out_stream << "drawarrow\n" << **point_iter++;
        else out_stream << "draw\n" << **point_iter++;
        (Output Path 907)
        if (draw_color == default_color) out_stream << "\nwithcolor\n" << *draw_color;
        if (dashed) out_stream << "\ndashed\n" << dashed;
        if (pen) out_stream << "\nwithpen\n" << pen;
        out_stream << ";\n" << flush;
    }
    if (DEBUG) cout << "Exiting Path::output(Focus)\n" << ";\n" << flush;
    return; }
907. When \texttt{fill\_color} and \texttt{draw\_color} are different, this will have to be performed twice, so I’ve made it a named section.

\begin{verbatim}
Log

[LDF 2002.11.03] \texttt{counter} is now initially set to 2 instead of 1. This makes each line have at most two
Points. Previously, the first line had 3 Points (if the \texttt{Path} had at least three Points on it).

[LDF 2002.12.20] Using the manipulator “fixed” below. It solves the problem of Points being output in
scientific format, which Metapost doesn’t understand.

[LDF 2002.12.20] I had to add preprocessor code for conditional compilation, because “fixed” is unknown
to the GNU C++ Compiler. However, it doesn’t need it in this case, since the problem only occurred when
using the DEC C++ compiler on a DEC Alpha computer under Compaq Tru64.
\end{verbatim}

\begin{verbatim}
\{Output Path 907\} \equiv
if (DEBUG) cout \ll \"Entering\_Output\_Path\|\n\" \ll flush;
for (unsigned short counter = 2; point\_iter \neq points\_end(); ) {
    out\_stream \ll \".u\" \ll connector\_string \ll \".u\" \ll \*point\_iter++;
    /* This breaks the line and indents after two points */
    if (counter \equiv 2 \wedge point\_iter \neq points\_end()) {
        out\_stream \ll \"\n\";
        counter = 1;
    }
    ++counter;
} if (connector\_iter \neq connectors\_end()) connector\_string = \*connector\_iter ++;
if (is\_cycle ()) out\_stream \ll \".u\" \ll connector\_string \ll \".ucycle\";
if (DEBUG) cout \ll \"Exiting\_Output\_Path\|\n\" \ll flush;
\end{verbatim}

This code is used in sections 902, 904, 905, and 906.

908. Showing.

909. Show.

\begin{verbatim}
Log

[LDF 2003.07.13] Commented-out the line that prints \texttt{fill\_draw\_value} to stdout.

[LDF 2003.08.20] Now printing \texttt{points\_size()} and \texttt{connectors\_size()} to stdout. If the latter is 0, a message
is printed, that "--" will be used as the connector.
\end{verbatim}

\begin{verbatim}
\{Declare Path functions 790\} \equiv
void show (string text = \"\", char coords = \'v\', const bool do\_persp = true, const bool
do\_apply = true, Focus *f = 0, const unsigned short proj = Projections::PERSP, const real
factor = 1) const;
\end{verbatim}
910.
(Define Path functions 701)  

```c
void Path::show(string text, char coords, const bool do_persp, const bool do_apply, Focus
+ f, const unsigned short proj, const real factor) const
{
    if (text == "") text = "Path:";
    cout << text << endl;
# ifdef 0
    cout << "fill_draw_value_u="u" << fill_draw_value << endl << flush;
# endif
    coords = tolower(coords);
    if (coords == 'w')  /* Do nothing. */
        else if (coords == 'p') cout << "Projective,coordinates.\n" << flush;
        else if (coords == 'u') cout << "User,coordinates.\n" << flush;
        else if (coords == 'v') cout << "View,coordinates.\n" << flush;
        else {
            cout << "WARNING!In show():\n" << "Invalid character for coords argument.\n" << "Showing world,coordinates.\n" << flush;
            coords = 'w';
        }
    valarray<real> v;
    v.resize(4,0);  /* LDF 2002.12.13. Added this line. Needed for porting to Intel Linux (i686). */
    string connector,string;
    vector<string>::const_iterator connectors_iter = connectors.begin();
    cout << "points.size()u==u" << points.size() << endl << flush;
    cout << "connectors.size()u==u" << connectors.size() << endl << flush;
    if (connectors.size() == 0) {
        cout << "Using\n" << "as,connector.\n" << flush;
        connector_string = "--";
    }
    int loop_ctr = 0;
    for (vector<Point *>::const_iterator points_iter = points.begin(); points_iter != points.end();
        ++points_iter) {
        if (points_iter != points.begin()) {
            cout << "u" << connector_string << "u";
        }
        if (connectors_iter != connectors.end()) connector_string = *connectors_iter++;
        if (loop_ctr == 2)  /* Break each line after 2 Points. */
        {
            cout << endl;
            loop_ctr = 0;
        }
        ++loop_ctr;
        v = (**points_iter).get_all_coords(coords, do_persp, do_apply, f, proj, factor);
    }
    if (cycle_switch) {
        if (connectors_iter != connectors.end()) connector_string = *connectors_iter++;
        cout << "u" << connector_string << "cycle;" << endl;
    }
```
911. Show Colors.
(Declare Path functions 700) +≡
void show_colors(bool stop = false);

912.
(Define Path functions 701) +≡
void Path::show_colors(bool stop)
{
    if (draw_color != 0) draw_color_show("draw_color");
    else cout << "draw_color==0.\n";
    if (fill_color != 0) fill_color_show("fill_color");
    else cout << "fill_color==0.\n";
    if (stop) getc();
}

913. Returning elements and information.

914. Is on free store.

---

[LDF 2004.01.06.] Made non-inline.

---

(Declare Path functions 700) +≡
bool is_on_free_store() const;

915.
(Define Path functions 701) +≡
bool Path::is_on_free_store() const
{
    return on_free_store;
}

916.  Is planar. [LDF 2002.11.05.] is_planar() uses the return value of get_normal() to determine
whether *this lies in a plane or not. If it does, is_planar() returns true, otherwise, it returns false. If *this
is linear, is_planar() issues a warning and returns true.

---

[LDF 2002.11.03.] Rewrote this function. It should now work for all Paths.
[LDF 2002.11.05.] Rewrote this function again. It now uses the new version of get_normal().
[LDF 2002.11.06.] Added optional const bool verbose and string text arguments for writing a message
to the standard output.
[LDF 2003.08.14.] Made verbose non-const. Setting it to true if VERBOSE_GLOBAL is true.
Added VERBOSE_GLOBAL to pspglb.web today.

---

(Declare Path functions 700) +≡
virtual bool is_planar(bool verbose = false, string text = "") const;
917. (Define Path functions 701) +≡
bool Path::is_planar(bool verbose, string text) const
{
    bool DEBUG = false; /* true */
    if (VERBOSE_GLOBAL) verbose = true;
    if (DEBUG ∨ verbose) cout ≈ "EnteringPath::is_planar() \n";
    Point p(get_normalized());
    if (p ≡ INVALID_POINT) {
        if (DEBUG) cout ≈ "ExitingPath::is_planar(). _Returning_\false. \n\n" ≈ flush;
        if (verbose) {
            if (text ≡ "") text = "Path";
            cout ≈ text ≈ "is_\non-planar.\n\n";
        }
        return false;
    }
    else if (p ≡ origin) {
        cerr ≈ "WARNING!\nInPath::is_planar():\n\n" ≈ "Path\nis_linear. _Returning_\true. \n\n" ≈ flush;
        if (verbose) {
            if (text ≡ "") text = "Path";
            cout ≈ text ≈ "is_planar.\n\n";
        }
        return true;
    }
    else {
        if (verbose) {
            if (text ≡ "") text = "Path";
            cout ≈ text ≈ "is_planar.\n\n";
        }
        if (DEBUG) cout ≈ "ExitingPath::is_planar(). _Returning_\true. \n\n" ≈ flush;
        return true;
    }
}

918. Is linear. [LDF 2003.04.09] is_linear() first checks whether line_switch is true. If it is, it returns true right away. Otherwise, it uses the return value of get_normalized() to determine whether *this is linear or not. If it is, is_linear() returns true, otherwise, it returns false.

---

[LDF 2002.11.05.] Added this function.
[LDF 2002.11.06.] Added optional const bool verbose and string text arguments for writing a message to the standard output.
[LDF 2003.04.09.] Now checking whether line_switch is true before calling get_normalized(). !! If a Path whose line_switch ≡ true is modified such that it's no longer linear, the programmer must ensure that line_switch is set to false!
[LDF 2003.08.14.] Made verbose non-const. Setting it to true if VERBOSE_GLOBAL is true.
Added VERBOSE_GLOBAL to psglib.web today.

(Declare Path functions 700) +≡
bool is_linear(bool verbose = false, string text = "") const;
919.  
(Define Path functions 701) \( \equiv \)

```cpp
bool Path::is_linear(bool verbose, string text) const
{
  bool DEBUG = false;  /* true */
  if (VERBOSE_GLOBAL) verbose = true;
  if (DEBUG \( \lor \) verbose) cout \( \ll \) "Entering Path::is_linear().\n"
; if (line.switch) /* LDF 2003.04.09. Added this conditional. */
    return true;
  Point p(getNormal());
  if (p \( \equiv \) origin) {
    if (verbose) {
      if (text \( \equiv \) "") text = "Path";
      cout \( \ll \) text \( \ll \) "is_linear.\n";
    }
    if (DEBUG) cout \( \ll \) "Exiting Path::is_linear().\n"
; return true;
  }
  else {
    if (verbose) {
      if (text \( \equiv \) "") text = "Path";
      cout \( \ll \) text \( \ll \) "is_non-linear.\n";
    }
    if (DEBUG) cout \( \ll \) "Exiting Path::is_linear().\n"
; return false;
  }
}
```

920. Get line switch. [LDF 2002.11.03] This function returns \texttt{true} for Paths that are created or set using two Points only, and no connectors, as arguments.

Log

[LDF 2002.11.03.] Renamed this function \texttt{getLineSwitch()} from \texttt{isLine()}. About to add \texttt{isLinear()}, which will test whether all the Points are collinear or not.

(Declare Path functions 700) \( \equiv \)

```cpp
inline bool getLineSwitch() const
{
  return line.switch;
}
```

921. Test for cycles.

(Declare Path functions 700) \( \equiv \)

```cpp
inline bool isCycle() const
{
  return cycle_switch;
}
```

922. Size (number of points).

(Declare Path functions 700) \( \equiv \)
inline int size()
{
    return points.size();
}

923. Slope.  [LDF 2002.11.05.] slope() can only be used for linear Paths. It returns a real value representing the slope of the trace of a line on the major plane represented by the char arguments, or INVALID_REAL, if the Path is non-linear. For example, if $\frac{a}{b}$ is a Path $p$ and $\frac{a}{b}$ is the trace of $p$ on the x-y plane, then $p$ slope('x', 'y') returns a real $m$ such that $m = (b - y)/x$ where $b$ is the y-intercept of $\frac{a}{b}$ and $x$ and $y$ are the x and y-coordinates of points on $\frac{a}{b}$.

[LDF 2002.11.05.] Changed this function, so that is_linear() is used instead of get_line_switch() (formerly "is_line()”). Now, it can be used for all linear Paths, not just ones created using the constructor for lines. Also, it was commented-out.

(Declare Path functions 790) +=
real slope(char a = 'x', char b = 'y');

924.
(Define Path functions 791) +=
real Path::slope(char a, char b)
{
    if (!is_linear()) {
        cerr << "ERROR! " << Path::is_not_linear!" <<
        "Returning INVALID_REAL
        return INVALID_REAL;
    }
    return points[1]-slope(*points[0], a, b);
}

925. Subpath.  [LDF 2002.11.05.] subpath() returns a new Path using points[start] through points[end - 1] from *this. If the optional bool argument cycle is used, then the new Path will be a cycle, whether *this is or not. One optional connector argument can be used. If it is, it will be the only connector. Otherwise, the connectors from *this are used.

[LDF 2002.11.05.] start must be < end. It is not possible to have start > end, even if *this is a cycle.

[LDF 2002.11.05.] Rewrote this function. Made subpath() itself and its arguments const. Added error handling code.

[LDF 2003.07.16.] Please note that start and end cannot be made const.
[LDF 2003.08.27.] Changed int i to size_t i in the for loops that compare i to start and end. The way it was before caused GCC with the "-Wall" option to issue a warning.

(Declare Path functions 790) +=
Path subpath(size_t start, size_t end, const bool cycle = false, const string connector = "") const;
926.  
(Define Path functions 701) +≡

Path Path::subpath(size_t start, size_t end, const bool cycle, const string connector) const {
  bool DEBUG = false;  /* true */
  Path p;
}

927.  [LDF 2002.11.06.] There is no "INVALID_PATH", so I return an empty one, if start ≥ end. Since
operator≡() currently doesn’t exist, there’s not much point in defining INVALID_PATH, since there’s no
way to compare another Path to it.

(Define Path functions 701) +≡

if (start ≥ end) {
  cerr ≡ "ERROR!\nInPath::subpath():\n  "start"\nargument is ≥ the \nend\nargument.\n  "Returning empty Path.\n  "\n  "flush;
  return p;
}

928.  [LDF 2002.11.06.] More error handling. In these cases, it’s possible to recover.

(Define Path functions 701) +≡

if (start > points.size() - 1) {
  cerr ≡ "ERROR!\nInPath::subpath():\n  "start"\nargument is > \npoints.size()\n- 1.\n  "Will try to recover by setting start = 0.\n  "\n  "flush;
  start = 0;
}

if (end > points.size()) {
  cerr ≡ "ERROR!\nInPath::subpath():\n  "end"\nargument is > \npoints.size().\n  "Will try to recover by setting end = points.size().\n  "\n  "flush;
  end = points.size();
}

929.  [LDF 2002.11.06.] If a connector argument is specified, all we have to do is put the appropriate Points
from points onto p.points, put connector onto p.connectors, and return p.

(Define Path functions 701) +≡

if (connector ≠ "") { for (size_t i = start; i < end; i++) {
  if (i ≥ points.size()) {
    cerr ≡ "ERROR!\nInPath::subpath():\n  "end"\nargument is > \npoints.size().\n  "\n  "Breaking out of loop.\n  "\n  "flush;
    break;
  }
  p.points.push_back ( create_new < Point > (points[i]) );
} p.connectors.push_back (connector);
  p.set_cycle (cycle);
  return p;
}
930. [LDF 2002.11.05.] If no connector argument is specified, then we have to get the appropriate connectors from *this. This is slightly tricky, because connectors doesn’t have to contain a connector for each pair of Points that is joined in a Path. So, first we must fill up p.connectors so that we can tell which ones to use.

[LDF 2002.11.05.] Actually, with the constructors that exist, there will either be only one connector or a connector for each pair of Points that need to be joined. However, it would be easy to write functions that add or remove connectors, so it’s best to have this routine.

(Define Path functions 701) +≡

\[ p = *this; \]
\[ \text{unsigned short } a = \text{points.size}(); \]
\[ \text{if } (\sim \text{cycle}) \ a = = 1; \]
\[ \text{int } i; \]
\[ \text{string } s = \text{connectors.back}(); \]
\[ \text{for } (i = \text{connectors.size}(); i < a; i++) \{ \]
\[ \quad \text{p.connectors.push_back}(s); \]
\[ \} \]
\[ \text{if } (\text{DEBUG}) \text{ cout } < < \text{"p.connectors.size()}_i = = a \text{" } < < \text{p.connectors.size()} \text{\(<} \text{endl} \text{\<flush}; \]
\[ \text{Path q; for (size_t } i = \text{start} ; i < \text{end}; i++) \{ \text{q.points.push_back (create_new < Point > (p.points[i])} \]
\[ \quad \} \]
\[ \text{if } (i < \text{p.connectors.size}()) \text{ q.connectors.push_back(p.connectors[i]);} \]
\[ \text{return q;} \]

931. Get point.

932. non-const version.

Log

[LDF 2002.11.05.] Made non-inline. Changed return value to const Point &.

[LDF 2003.11.28.] BUG FIX: Changed, so that apply_transform() is called on the Point. This entailed making this function non-const. Added const version below. This may actually be a bug, rather than a bug fix, depending on how this function is used elsewhere. However, I really think apply_transform() should be called.

(Declare Path functions 700) +≡

\[ \text{const Point &get_point(const unsigned short a);} \]
933. (Define Path functions 701) +≡
    const Point &Path::get_point(const unsigned short a)
    {
        if (a < points.size()) {
            (points[a]).apply_transform();
            return *points[a];
        } else {
            cerr << "ERROR! In Path::get_point(): \n" << "Argument\n" << "Returning INVALID_POINT. \n" << flush;
            return INVALID_POINT;
        }
    }

934. const version.

    [LDF 2003.11.28.] Added this version.

(Declare Path functions 700) +≡
    Point get_point(const unsigned short a) const;

935. (Define Path functions 701) +≡
    Point Path::get_point(const unsigned short a) const
    {
        if (a < points.size()) {
            Point p = *(points[a]);
            p.apply_transform();
            return p;
        } else {
            cerr << "ERROR! In Path::get_point(): \n" << "Argument\n" << "Returning INVALID_POINT. \n" << flush;
            return INVALID_POINT;
        }
    }

936. Get last point.

    [LDF 2002.06.10.] Added this function.
    [LDF 2002.11.05.] Made non-inline. Changed return value to const Point &.

(Declare Path functions 700) +≡
    const Point &get_last_point() const;
937. Define Path functions 701
   
   const Point &Path::get_last_point() const
   {
      if (points.size() != 0) return *points[points.size() - 1];
      else { cerr << "ERROR in Path::get_last_point():
                  "Path is empty.\n                  "Returning INVALID_POINT.\n                  "flush; return INVALID_POINT; 
   }

938. Get size.
   
   Declare Path functions 700
   
   virtual inline size_t get_size() const
   {
      return points.size();
   }

939. Get normal.
940. Path version. [LDF 2002.11.05.] \( \text{get\_normal()} \) returns a unit vector representing the normal to the plane of the Path \( *this \), if \( *this \) is planar. If the Points on \( *this \) are colinear and there are no connectors that could make the Path non-planar, then origin \( ((0,0,0)) \) is returned. If the Path is neither planar nor linear, \( \text{get\_normal()} \) returns INVALID\_POINT.

[LDF 2002.11.05.]
- \( \text{get\_normal()} \) first checks whether a Path contains no Points or only one Point. If so, \( \text{get\_normal()} \) returns INVALID\_POINT.
- Then it checks whether the Path has connectors that might make the Path non-planar, even if the Points lie in a plane. If it does, it returns INVALID\_POINT. Note that there is no guarantee that the connectors actually will make the Path non-planar.
- Then it checks whether the Path has only two Points. If it does, \( \text{get\_normal()} \) returns the Point \( (0,0,0) \), because the Path will be linear.
- Then it gets the cross product \( b_0 \) of \( \overrightarrow{p_0p_2} \) and \( \overrightarrow{p_0p_2} \), where \( p_0 \) and \( p_2 \) are the first and second Points on the Path, and \( p_2 \) is the next Point on the Path such that \( b_0 \neq (0,0,0) \). If no Points on the Path fulfill this condition, then all of the Points are colinear, so \( \text{get\_normal()} \) returns \( \text{origin} \).
- If, however, \( b_0 \neq (0,0,0) \) exists, then cross products \( b_0 \) are calculated using \( \overrightarrow{p_0p_2} \) and the direction vectors \( \overrightarrow{p_0p_2} \) for the rest of the Points \( p_k \) on the Path. If and only if \( b_0 = b_0 \), \( b_0 = -b_0 \), or \( b_0 = (0,0,0) \) for all \( b_0 \), then the Path is planar, and \( \text{get\_normal()} \) returns \( -b_0 \) (see explanation of sign below). Otherwise, the Path is non-planar, and \( \text{get\_normal()} \) returns INVALID\_POINT.

[LDF 2003.06.04.] Reversing the sign of \( b_0 \) ensures that the normal will point in the direction of the positive y-axis, when a plane figure is created in the x-z plane, using one of the constructors taking a Point argument for the center, real arguments for the dimensions, and three real arguments for the rotation about the major axes. If non-zero arguments are used for rotation, the normal will be rotated accordingly. This direction considered to be "outside". In 3DLDF, the constructors generally generate Points moving about the figure in the counter-clockwise direction (as seen from a Point with a positive y-coordinate). However, according to Hsu Jones, *Computer Graphics Through Key Mathematics*, p. 197, "outside" is considered to be the side of a plane, where the Points are meant to be traversed in the clockwise direction. !! Watch out for problems that may arise from this discrepancy!

Log

[LDF 2002.11.05.] Rewrote this function.
[LDF 2003.06.04.] Changed sign of the normal, when it's returned, in the cases where a proper normal is found (not INVALID\_POINT or origin). See explanation above.

(Declare \textbf{Path functions} 700) \( + \equiv \)

\textbf{virtual Point get\_normal()}\ \textbf{const};

941.
(Define \textbf{Path functions} 701) \( + \equiv \)

\textbf{Point Path::get\_normal()}\ \textbf{const}\{ \textbf{bool DEBUG = false; /* true */}
  \textbf{if} (DEBUG) \textbf{cout} \ll "Entering\_Path::get\_normal() \n\n" \ll \textbf{flush};
  \textbf{if} (\textbf{points}\_\textbf{size}() \leq 0) \{ \textbf{cerr} \ll "WARNING! In\_Path::get\_normal():n" \ll 
    "Path\_is\_empty\_or\_contains\_only\_one\_Point.\n\n" \ll "Returning\_INVALID\_POINT.\n\n" \ll \textbf{flush};
    \textbf{return} INVALID\_POINT; \}
}
Connectors other than the ones in the conditional below could cause the Path to be non-linear or non-planar, even if the Points lie on a line or a plane.

```cpp
for (vector<string>::const_iterator iter = connectors.begin(); iter != connectors.end(); ++iter) {
    if (-(iter == "...") || iter == "---") {
        cerr << "WARNING! Un-Path: get_normal(): \n" << "Connector may make Path non-linear or non-planar.
        return INVALID_POINT;
    }
}
```

Two points determine a line.

```cpp
if (points.size() == 2) {
    cerr << "WARNING! Un-Path: get_normal(): \n" << 
        "Path has 2 Points. Returning origin. \n\n" << flush;
    return origin;
}
```
944.

(Define Path functions 701) \( \equiv \)
\[
\text{vector(Point *):: const_iterator iter = points.begin();}
\]
Point p0(**iter++);
Point p1(**iter++);
Point p2;
Point a0(p1 - p0);
Point a1;
Point b0;

while \((b0 \equiv \text{origin} \land \text{iter} \neq \text{points.end()})\) {
  p2 = **iter ++;
  a1 = p2 - p0;
  b0 = a0.cross_product(a1);
}
if (iter \equiv \text{points.end()} \land b0 \equiv \text{origin}) {
  if (DEBUG) cout \ll "Exiting\_\_Path::get_normal() \_\_" \ll "Points\_\_are\_\_all\_\_collinear.\n\n:\ll "Returning\_\_origin.\n\n:\ll flush;
      return origin;
  }
  if (DEBUG) b0.show("b0");
  b0.unitVector(true);
if (iter \equiv \text{points.end()} \land b0 \neq \text{origin}) {
  if (DEBUG) {
    cout \ll "Exiting\_\_Path::get_normal() \_\_" \ll "Points\_\_are\_\_all\_\_collinear\_\_except\_\_for\_\_one.\n\n:\ll "Returning\_\_normal.\n\n:\ll flush;
  }
  return -b0;
}

Point b1;
if (DEBUG) cout \ll "Entering\_\_second\_\_while.\n\n:\ll flush;
while (iter \neq \text{points.end()}) {
  p2 = **iter ++;
  a1 = p2 - p0;
  b1 = a0.cross_product(a1);
  if (b1 \neq \text{origin})
    \text{ /* [LDF 2002.11.03] This if merely prevents a warning from being issued by unitVector(). */}
      b1.unitVector(true);
  if (DEBUG) b1.show("b1");
  if (\(-(b1 \equiv \text{origin} \lor b1 \equiv b0 \lor b1 \equiv -b0))\) {
    if (DEBUG)
      cout \ll "Exiting\_\_Path::get_normal() \_\_" \ll "Returning\_\_\_INVALID\_\_POINT.\n\n:\ll flush;
      return INVALID\_\_\_POINT;
  }
}
if (DEBUG) cout \ll "Exiting\_\_Path::get_normal() \_\_" \ll "Returning\_\_\_normal.\n\n:\ll flush;
return -b0; \}
945. **Point version.** `Point::get_normal()` is declared in `points.def`, but it must be defined here, because it calls `Path::get_normal()`. [LDF 2003.07.11.]  

```
[Log]

(Define Point functions 330) +=
Point Point::get_normal(const Point &p, const Point &q) const
{
    Path r;
    r.set_connectors("--");
    r += *this;
    r += p;
    r += q;
    if (!r.is_planar()) {
        cerr << "ERROR! In Point::get_normal():
             The Points do not determine a plane."
             "Returning INVALID_POINT."
             "flushing output..."
        return INVALID_POINT;
    } else return r.get_normal();
}
```

946. **Get plane.**

```
[Log]

(Declare Path functions 700) +=
virtual Plane get_plane() const;
```
947. (Define Path functions 701) +≡
   Plane Path::get_plane() const
   {
     Point normal(get_normal());
     if (normal ≡ INVALID_POINT ∨ normal ≡ origin) {
       cerr ≡ "WARNING! PlanePath::get_plane().\n" ≡
         "Path not in Plane..Returning INVALID_PLANE.\n\n" ≡ flush;
       return INVALID_PLANE;
     }
     Point point(get_point());
     return Plane(point, normal);
   }

948. Point lies within triangle. [LDF 2003.06.11.] Declared in points.web. Must be defined here, because Path is an incompletely defined type there.

---

[LDF 2003.06.11.] Added this function.
[LDF 2003.06.24.] Removed the argument test_points. Now, planarity is always tested.
[LDF 2003.06.24.] BUG FIX. When the Points all lay in the x-z plane, or a plane parallel to it, lambda_denominator was 0. This caused is_in_triangle() to return false, even when *this did lie in the triangle. Now, if lambda_denominator or mu_denominator is equal to 0, the y and z-coordinates are exchanged, and lambda_denominator and mu_denominator are recalculated. If either of the new values is 0, the x and z-coordinates are exchanged (based on the original coordinate values), and lambda_denominator and mu_denominator are again recalculated. Only one exchange has been needed in the cases I've tested so far.
[LDF 2003.06.14.] Setting verbose to true if VERBOSE_GLOBAL is true. Added VERBOSE_GLOBAL to pspgb.web today.

(Define Point functions 330) ≡
   bool Point::is_in_triangle(const Point &p0, const Point &p1, const Point &p2, bool verbose)
   const { bool DEBUG = false; /* true */
     if (VERBOSE_GLOBAL) verbose = true;
     Path q;
     q += p0;
     q += p1;
     q += p2;
     Plane q.pl = q.get_plane();
     if (q.pl ≡ INVALID_PLANE) {
       if (verbose) {
         cerr ≡ "WARNING! Point::is_in_triangle():\n" ≡
           "The arguments do not determine a plane.\n" ≡ "Returning false.\n\n" ≡ flush;
       }
       return false;
     } else if (!is_on_plane(q.pl)) {
       if (verbose) {

--
cerr << "WARNING! In Point::is_in_triangle():\n" << "this doesn't lie in the plane determined by the arguments.\n" << "Returning false. \n" << flush;

return false;
}
Point t(*this);
Point c(p0);
Point d(p1);
Point e(p2);
t.apply_transform();
c.apply_transform();
d.apply_transform();
e.apply_transform();
if (DEBUG) {
    show("t:");
    c.show("c:");
    d.show("d:");
    e.show("e:");
}
real tx = t.world_coordinates[0];
real ty = t.world_coordinates[1];
real tz = t.world_coordinates[2];
real cx = c.world_coordinates[0];
real cy = c.world_coordinates[1];
real cz = c.world_coordinates[2];
real dx = d.world_coordinates[0];
real dy = d.world_coordinates[1];
real dz = d.world_coordinates[2];
real ex = e.world_coordinates[0];
real ey = e.world_coordinates[1];
real ez = e.world_coordinates[2];
real lambda_denominator = (((cx - tx) * (dy - cy)) - ((cy - ey) * (dx - cx))) - ((cy - ey) * (dx - cx));
real mu_denominator = (((cx - tx) * (dy - cy)) - ((cy - ey) * (dx - cx))) - ((cy - ey) * (dx - cx));
bool exchange_yz = false;
bool exchange_xz = false; if (lambda_denominator ≜ 0 ∨ mu_denominator ≜ 0) {
    if (DEBUG) cout << "lambda_denominator_or_mu_denominator = 0.0" << "Exchanging y and z-coordinates. \n" << flush;
    real temp;
temp = ty;
    ty = tx;
    tx = temp;
temp = cy;
    cy = cz;
    cz = temp;
temp = dy;
    dy = dz;
    dz = temp;
temp = ey;
    ey = ez;
    ez = temp;
\begin{verbatim}
lambda_denominator = ((e.x - c.x) * (d.y - c.y)) - ((e.y - c.y) * (d.x - c.x));
mu_denominator = ((e.x - c.x) * (d.y - c.y)) - ((e.y - c.y) * (d.x - c.x));
if (-(lambda_denominator \equiv 0 \lor mu_denominator \equiv 0)) {
    if (DEBUG) cout << "Exchanging u, y, and z-coordinates worked.\n" <<
        "lambda_denominator and mu denominator are no longer 0.\n" << flush;
    exchange,y,z = true;
} else {
    if (DEBUG) cout << "Exchanging u, y, and z-coordinates didn't work.\n" <<
        "Exchanging u, x, and z-coordinates.\n" << flush;
}
\end{verbatim}

949. First, put things back the way they were. It's wasteful, but less confusing. [LDF 2003.06.24]

(Define Point functions 330) +≡

\begin{verbatim}
temp = t.y;
t.y = t.x;
t.x = temp;
temp = c.y;
c.y = c.x;
c.x = temp;
temp = d.y;
d.y = d.x;
d.x = temp;
temp = e.y;
e.y = e.x;
e.x = temp;
e.x = temp;
\end{verbatim}
950. Now, exchange the x and z-coordinates. [LDF 2003.06.24.]

(Define Point functions 330) +

```c
temp = t.x;
t.x = t.z;
t.z = temp;
temp = c.x;
c.x = c.z;
c.z = temp;
temp = d.x;
d.x = d.z;
d.z = temp;
temp = e.x;
e.x = e.z;
e.z = temp;
```

```c
lambda.denominator = (((e.x - c.x) * (d.y - c.y)) - ((e.y - c.y) * (d.x - c.x)));
mu.denominator = (((e.x - c.x) * (d.y - c.y)) - ((e.y - c.y) * (d.x - c.x)));
```

```c
if ((lambda.denominator == 0) || (mu.denominator == 0)) {
    if (DEBUG) cout << "Exchanging x and z-coordinates worked. \n" << "lambda_denominator and mu_denominator are no longer 0. \n" << flush;
    exchange_xz = true;
} else {
    if (verbose ∨ DEBUG) {
        cerr << "WARNING! InPoint::is_in_triangle(): \n" << "lambda_denominator and mu_denominator are 0. \n" << "Returning false. \n" << flush;
    }
    return false;
}
```

```c
real lambda = (((t.x - c.x) * (d.y - c.y)) - ((t.y - c.y) * (d.x - c.x)))/lambda.denominator;
```

```c
real mu = -(((t.x - c.x) * (e.y - c.y)) - ((t.y - c.y) * (e.x - c.x)))/mu.denominator;
```

```c
if (DEBUG) {
    cout << "lambda = " << lambda << " lambda_endl << flush;"
    cout << "mu = " << mu << " mu_endl << flush;"
    cout << "lambda + mu = " << lambda + mu << " endl << flush;"
    cout << "(lambda >= 0 ∧ mu >= 0 ∧ (lambda + mu) <= 1) \n" << "(lambda >= 0 ∧ mu >= 0 ∧ (lambda + mu) <= 1) \n" << "endl << flush;"
}
```

```c
bool b = (lambda >= 0 ∧ mu >= 0 ∧ ((lambda + mu) <= 1));
```

```c
if (verbose) {
    cout << "InPoint::is_in_triangle: \n";
    if (b) cout << "The Point is within the triangle. \n" << "Returning true. \n";
    else cout << "The Point doesn't lie within the triangle. \n" << "Returning false. \n";
    cout << endl << endl << flush;
}
```

```c
return b; }
```
[LDF 2002.11.05.] Made `bool c` argument `const`.

(Declare `Path` functions 700) +≡

```cpp
void set_cycle(const bool c = true);
```

953.

(Define `Path` functions 701) +≡

```cpp
void Path::set_cycle(const bool c)
{
    cycle_switch = c;
}
```

954. Reverse.

955. With assignment.

[Log]

[LDF 2002.4.6.] Added this function.
[LDF 2003.07.16.] Added error handling code for the case that this function is called with `assign ≡ false`. I've now added a `const` version, so there's no need to call this version with `assign ≡ false`. If `assign` is `false`, the `const` version is called, so I could simplify the code in this version.

(Declare `Path` functions 700) +≡

```cpp
Path reverse(bool assign);
```

956.

(Define `Path` functions 701) +≡

```cpp
Path Path::reverse(bool assign){ bool DEBUG = false; /* true */
    if (is_cycle()) /* Return *this if *this is a cycle. */
    {
        cerr ≡ "ERROR! In Path::reverse().\n" ≡ "*this is a cycle. Can't reverse.\n" ≡ "Returning *this.\n"
        return *this;
    }
    if (~assign)
    {
        cerr ≡ "WARNING! In Path::reverse(bool):\n" ≡ "assign is false. Do not call this function.\n" ≡ "with false as it as a argument.\n" ≡ "Use reverse() without an argument instead.\n" ≡ "Calling reverse(void).\n"
        flush;
        return reverse();
    }
```
957. [LDF 2002.4.6.] If there is more than one connector, but there isn’t an explicit connector for every pair of Points in points, then we have to fill up connectors so that there is one for each pair of Points. Otherwise, the connectors and the Points won’t match up properly when we reverse them.

```cpp
<Define Path functions 701> +≡
  if (connectors.size() > 1 ∧ connectors.size() ≠ points.size() - 1) {
    string lastConnector = connectors.back();
    while (connectors.size() < points.size() - 1) connectors.push_back(lastConnector);
  }
```

958. [LDF 2002.4.7.] If I don’t explicitly refer to the std namespace here, this function is called, and since the arguments are different from the one used for this function, this causes an error at compile time.

```cpp
<Define Path functions 701> +≡
  if (DEBUG) cout "Reversing connectors and points. \n" ≪ flush;
  std::reverse(connectors.begin(), connectors.end());
  std::reverse(points.begin(), points.end());
  if (DEBUG) {
    cout "Showing connectors:\n";
    for (vector<string>::iterator iter = connectors.begin(); iter ≠ connectors.end(); iter ++)
      cout ≪ *iter ≪ endl;
    cout "Showing points:\n";
    for (vector<Point *>::iterator iter = points.begin(); iter ≠ points.end(); iter ++)
      (**iter).show();
    getchar();
  } /* if (DEBUG) */
  return *this;
```

959. No assignment. This version merely copies *this and calls reverse(true) on the copy, returning the return value of that function call.

```cpp
<Declare Path functions 700> +≡
  Path reverse( void ) const;
```

960.

```cpp
<Define Path functions 701> +≡
  Path Path::reverse( void ) const
  {
    Path p = *this;
    return p.reverse(true);
  }
```

961. Equality. TO DO: I’ll need to make all connectors explicit in order to make this work. See operator&() for an example of how to make this work.

```cpp
<Declare Path functions 700> +≡
#if 0
  virtual bool operator==(Path &p);
#endif
```
962.
(Define Path functions 701) +≡
#if 0
  virtual bool Path::operator≡(Path &p)
  {}
#endif

963. Intersection.

964. Intersection of two linear Paths. If *this is a line and the argument pa is a line,
intersection_point() calls the version for Points in points.web.
    Other kinds of Paths and other classes will need their own versions of this function.
    I may have a problem with the constancy of *this and pa. If I do, just remove it.

Log
[LDf 2002.04.15.] Changed return value from bool real point to bool point, since I've had to comment-out the version of Point::intersection_point() that uses the Line version.
[LDf 2002.04.10.] Changed return type to bool real point to correspond with the same change to Point::intersection_point().
[LDf 2003.07.04.] Added trace argument. Added conditional using trace to choose which version of Point::intersection_point() should be called. Changed so that is_linear() is used instead of get_line_switch(). Now using get_last_point() instead of +points[1].

(Declare Path functions 700) +≡
  bool point intersection_point(const Path &pa, const bool trace = false) const;
965. (Define Path functions 701) \( \equiv \)

```cpp
bool_point Path::intersection_point(const Path &pa, const bool trace) const
{
    if (is_linear() && pa.is_linear()) {
        if (trace) return Point::intersection_point(*points[0], get_last_point(), *pa.points[0],
                                             pa.get_last_point(), trace);
        else return Point::intersection_point(*points[0], get_last_point(), *pa.points[0], pa.get_last_point(),
                                             trace);
    } else {
        cout << "Haven’t coded this case yet…” << "Returning INVALID_BOOL_POINT.\n" << flush;
        return INVALID_BOOL_POINT;
    }
}
```

966. Intersection of a linear Path with a Plane. [LDF 2003.06.03] This function must be defined here, because Path is an incomplete type in planes.web.

[LDF 2003.06.03] Added this function.

967. Drawing axes. This function draws and labels arrows for the main axes at the origin. It can be helpful for determining whether the “up” direction is correct for a Focus.

[LDF 2003.04.01] Sometimes placeholders are needed for the dist and position arguments. If dist is a number \( z < 0 \), then it’s set to the default, currently 2.5. If a position argument (pos.x, pos.y, or pos.z) is "d", it’s set to the default.

[LDF 2003.02.05] Moved this function from main.web to here, so I can use it in my examples for the Texinfo documentation. Also, added additional arguments specifying the positions of labels and suppressing drawing the axes (and their labels).

[LDF 2003.04.01] Added arguments for dash pattern (\textit{dashed}) and pen (\textit{pen}). Rearranged order of arguments. Also, got rid of the arguments suppress.x, suppress.y, and suppress.z. Now using the empty string (""") in the arguments pos.x, pos.y, and pos.z to indicate that the corresponding axes should be suppressed. Added error handling code that prints a warning to stderr if all axes are suppressed. (LDF 2003.05.06). Note that "" will never be needed for labelling an axis, because putting the label on top of the Point would interfere with the arrow.)

[LDF 2003.04.01] Added arguments \textit{shift.x}, \textit{shift.y}, and \textit{shift.z} for adjusting the position of the labels. Note that the adjustment affects the position of the three-dimensional Point within the Label, not the two-dimensional projected point. Therefore, it’s not possible to adjust the position of the Label precisely
without changing the Metapost code. TO DO: Change label(), so that it’s possible to adjust the position of the points in the projection! This may open a can of worms, though, especially if the same code is used to generate drawings using different projections and/or different Focuses.

[LDF 2003.07.13.] Made \textit{dashed} and \textit{pen} \texttt{const} in both versions.

\section*{Exercise 968.} Length argument first.

\begin{verbatim}
(Declare \texttt{draw_axes()} 968) \equiv
  void \texttt{draw_axes}(real \texttt{dist} = 2.5, string \texttt{pos}_x = "bot", string \texttt{pos}_y = "1ft", string
  \texttt{pos}_z = "bot", const Color &\texttt{draw_color} = \ast\texttt{Colors}::\texttt{default_color}, const string
  \texttt{dashed} = "", const string \texttt{pen} = "", const Point &\texttt{shift}_x = \texttt{origin}, const Point
  &\texttt{shift}_y = \texttt{origin}, const Point &\texttt{shift}_z = \texttt{origin}, \texttt{Picture} &\texttt{picture} = \texttt{current_picture});
\end{verbatim}

See also section 973.

This code is used in section 981.

\section*{Exercise 969.}

\begin{verbatim}
(Define \texttt{draw_axes()} 969) \equiv
  void \texttt{draw_axes}(real \texttt{dist}, string \texttt{pos}_x, string \texttt{pos}_y, string \texttt{pos}_z, const Color &\texttt{draw_color}, const
  string \texttt{dashed}, const string \texttt{pen}, const Point &\texttt{shift}_x, const Point &\texttt{shift}_y, const Point
  &\texttt{shift}_z, \texttt{Picture} &\texttt{picture})
\end{verbatim}

See also sections 970, 971, 972, and 974.

This code is used in section 986.

\section*{Exercise 970.} Remember to change this if you change any of the defaults!

[Add this section.

\section*{Exercise 971.}

\begin{verbatim}
(Define \texttt{draw_axes()} 969) \equiv
  if (\texttt{dist} \leq 0) \texttt{dist} = 2.5;
  if (\texttt{pos}_x \equiv "d") \texttt{pos}_x = "bot";
  if (\texttt{pos}_y \equiv "d") \texttt{pos}_y = "1ft";
  if (\texttt{pos}_z \equiv "d") \texttt{pos}_z = "bot";
\end{verbatim}

[Add this error handling code.

\begin{verbatim}
(Define \texttt{draw_axes()} 969) \equiv
  if (\texttt{pos}_x \equiv "n" \land \texttt{pos}_y \equiv "n" \land \texttt{pos}_z \equiv "n") {
    \texttt{cerr} \ll \"WARNING! In \texttt{draw_axes():} \ll \texttt{endl} \ll \"All axes are suppressed. Returning." \ll
    \texttt{endl} \ll \texttt{endl} \ll \texttt{flush};
    \texttt{return};
  }
\end{verbatim}
972.  
(Define `draw_axes()` 969) \(\equiv\)
if \(\text{pos}_x \neq \text{""}\) {
    Point \(x0(-\text{dist});\)
    Point \(x1(\text{dist});\)
    \(x0.\text{drawarrow}(x1, \text{draw_color}, \text{dashed}, \text{ppen}, \text{picture});\)
    \(x1 += \text{shift}_x;\)
    \(x1.\text{label}(\text{"x"}, \text{pos}_x, \text{false}, \text{picture});\)
}
if \(\text{pos}_y \neq \text{""}\) {
    Point \(y0(0,-\text{dist});\)
    Point \(y1(0,\text{dist});\)
    \(y0.\text{drawarrow}(y1, \text{draw_color}, \text{dashed}, \text{ppen}, \text{picture});\)
    \(y1 += \text{shift}_y;\)
    \(y1.\text{label}(\text{"y"}, \text{pos}_y, \text{false}, \text{picture});\)
}
if \(\text{pos}_z \neq \text{""}\) {
    Point \(z0(0,0,-\text{dist});\)
    Point \(z1(0,0,\text{dist});\)
    \(z0.\text{drawarrow}(z1, \text{draw_color}, \text{dashed}, \text{ppen}, \text{picture});\)
    \(z1 += \text{shift}_z;\)
    \(z1.\text{label}(\text{"z"}, \text{pos}_z, \text{false}, \text{picture});\)
}
return; }

973.  Color argument first.

---

Log

[LDF 2003.06.02] Added this function.

---

(Declare `draw_axes()` 968) \(\equiv\)

`void draw_axes(const Color &\text{draw_color}, \text{real dist} = 2.5, \text{string pos}_x = \text{"bot"}, \text{string pos}_y = \text{"ft"}, \text{string pos}_z = \text{"bot"}, \text{const string dashed} = \text{""}, \text{const string ppen} = \text{""}, \text{const Point &shift}_x = \text{origin}, \text{const Point &shift}_y = \text{origin}, \text{const Point &shift}_z = \text{origin}, \text{Picture &picture} = \text{current.picture});`
974.
(Define \( \text{draw\_axes()} \) 969) \( \equiv \)

\[
\begin{align*}
\text{void } \text{draw\_axes}(\text{const Color &draw\_color, real dist, string pos\_x, string pos\_y, string pos\_z, const string dashed, const string pen, const Point &shift\_x, const Point &shift\_y, const Point &shift\_z, Picture &picture}) \\
\{
\text{draw\_axes}(\text{dist, pos\_x, pos\_y, pos\_z, draw\_color, dashed, pen, shift\_x, shift\_y, shift\_z, picture)};
\}
\end{align*}
\]

975. Paths and Lines.

\[\text{Log} \]

[LDF 2003.06.06.] Added this heading.

976. Get Line. Returns a Line corresponding to *this, if *this is linear. Otherwise, getLine() returns INVALID_LINE.

\[\text{Log} \]

[LDF 2003.06.06.] Added this function.

(Declare Path functions 700) \( \equiv \)

\[
\begin{align*}
\text{Line get\_line(\text{void}) const;}
\end{align*}
\]

977.
(Define Path functions 701) \( \equiv \)

\[
\begin{align*}
\text{Line Path::get\_line(\text{void}) const}
\{
\text{if (is\_linear()) return points\_front()&get\_line(*points\_back());}
\text{else }
\{\text{cerr \ll "ERROR! In Path::get\_line():\n" \ll "Path is not linear. Returning INVALID\_LINE.\n\" \ll flush;\nreturn INVALID\_LINE;\}}
\}
\end{align*}
\]

978. Get Path. Declared in lines.web. Must be defined here, because Path is an incomplete type there.

\[\text{Log} \]

[LDF 2003.06.06.] Added this function.

(Define Line functions 644) \( \equiv \)

\[
\begin{align*}
\text{Path Line::get\_path(\text{void}) const}
\{
\text{Point p(position + direction);}
\text{return Path(position, p);}
\}
\end{align*}
\]

979. Putting Path together.
This is what’s compiled.

{Include files 6}
{Version control identifier 5}
{Define class Path 698}
{Define static class Path data members 699}
{Define Transform functions 169}
{Define Point functions 330}
{Define Plane functions 664}
{Define Path functions 701}
{Define Line functions 644}
{Define draw_axes() 969}
{Declare non-member template functions for Path 724}
981. This is what's written to paths.h.

```
(paths.h 981) ≡
  (Define class Path 698)
  (Declare draw_axes() 968)
  (Declare non-member template functions for Path 724)
```

982. Curves (curves.web).

Log

[LDF 2003.11.12.] Removed the version control identifiers from the CWEB files for the distribution of 3DLDF 1.1. They're still used in my development versions.

[LDF 2003.12.10.] Put the version control identifiers back into my release versions for 3DLDF 1.1.4. I've already put some of them back in, now I'm doing the rest of them. However, the release versions are now in their own RCS repository.

```
format Curve Path
```

```
static string res_id = "$Id: curves.web,v 1.5 2004/01/12 21:27:59 linsto1 Exp $";
```

983. Include files.

```
(Include files 6) ≡

#include "loader.h"
#include "pspglb.h"
#include "io.h"
#include "colors.h"
#include "transform.h"
#include "shapes.h"
#include "pictures.h"
#include "points.h"
#include "lines.h"
#include "planes.h"
#include "paths.h"
```

984. Regular closed plane curve.

Log

[LDF 2002.11.12.] Changed the name "Regular_Closed_Plane_Curve" to "Reg_Cl_Plane_Curve", because the former caused too many "Overfull boxes" when running cweave.

985. Reg_Cl_Plane_Curve class definition. A Reg_Cl_Plane_Curve is assumed to be closed, planar, convex, and have at least 3 points. The functions that create and modify Reg_Cl_Plane_Curves must ensure that these assumptions are correct!

[LDF 2002.11.05.] Reg_Cl_Plane_Curve is intended to be used as a base class. No objects of type Reg_Cl_Plane_Curve should be defined, however, it is not an abstract class, so it is possible to do so.

```
#define Reg_Cl_Plane_Curve  Curve
```

```
(Define class Reg_Cl_Plane_Curve 985) ≡
  class Reg_Cl_Plane_Curve : public Path { 
  protected: Point center;
  unsigned short number_of_points;
  
  public: (Declare Reg_Cl_Plane_Curve functions 987)
```
\section{\texttt{Reg\_Cl\_Plane\_Curve} Class Definition}

This code is used in sections 1015 and 1016.

\section*{986. Returning elements and information.} [LDF 2002.11.06.] The \textit{virtual} functions in this section are meant to be overloaded by member functions of types derived from \texttt{Reg\_Cl\_Plane\_Curve}.

\begin{verbatim}

Log

[LDF 2002.11.03.] Removed \texttt{Reg\_Cl\_Plane\_Curve::is\_planar}. A \texttt{Reg\_Cl\_Plane\_Curve} can be manipulated into a non-planar state, so it's safer to use the \texttt{Path} version, which tests whether it's really planar or not.

\end{verbatim}

\section*{987. Is quadratic.}

\begin{verbatim}
(Declare \texttt{Reg\_Cl\_Plane\_Curve} functions 987) \equiv
inline virtual bool is\_quadratic() const
{
    return false;
}

See also sections 988, 989, 990, 991, 992, 994, 997, 1008, 1011, 1013, and 1014.

This code is used in section 985.

988. Is cubic.

\begin{verbatim}
(Declare \texttt{Reg\_Cl\_Plane\_Curve} functions 987) \equiv
inline virtual bool is\_cubic() const
{
    return false;
}

989. Is quartic.

\begin{verbatim}
(Declare \texttt{Reg\_Cl\_Plane\_Curve} functions 987) \equiv
inline virtual bool is\_quartic() const
{
    return false;
}

990. Get coefficients.

\begin{verbatim}
(Declare \texttt{Reg\_Cl\_Plane\_Curve} functions 987) \equiv
inline virtual real\_triple get\_coefficients(real, real) const
{
    return real\_triple(INVALID\_REAL, INVALID\_REAL, INVALID\_REAL);
}

991. Solve. [LDF 2002.11.05.] This \textit{virtual} function is meant to be overloaded by member functions of types derived from \texttt{Reg\_Cl\_Plane\_Curve}.

\begin{verbatim}
(Declare \texttt{Reg\_Cl\_Plane\_Curve} functions 987) \equiv
inline virtual pair(real, real) solve(char, real) const
{
    return pair(real, real)(INVALID\_REAL, INVALID\_REAL);
}
\end{verbatim}

\end{verbatim}
Location of a point, \( \text{location}() \) returns a signed short indicating the location of its Point argument with respect to the Reg_CiPlane_Curve.

[LDF 2002.11.05] TO DO: Currently, the programmer must ensure that a Reg_CiPlane_Curve is planar. It might be worthwhile to check that it really is by using Path::get_normal(), since some manipulations may cause a Reg_CiPlane_Curve to become non-planar.

[LDF 2002.11.05] The number of Points in a Reg_CiPlane_Curve must be a multiple of 4, and that the Point number_of_points/4 must be at 90° to Point 0. Also, ref_pt can’t be Point 0.

[LDF 2003.07.16] Reg_CiPlane_Curve now has a data member named center. However, a Reg_CiPlane_Curve need not have a meaningful center. Usually, when an object of a class derived from Reg_CiPlane_Curve calls this function, its center will be passed as the ref_pt argument. However, this need not be the case.

TO DO: Check whether it will work if ref0.x < 0. I think it should.

[LDF 2003.06.14] !! CHECK. Bug, when Reg_CiPlane_Curve is rotated about x and z-axes only.

The following values are returned if the Point is in the same plane as this and this function has worked properly:

-1 The Point lies outside the Reg_CiPlane_Curve.

0 The Point lies on the perimeter of the Reg_CiPlane_Curve.

1 The Point lies inside the perimeter of the Reg_CiPlane_Curve.

These values are returned in cases where errors have occurred:

-2 The Point is not in the same plane as the Reg_CiPlane_Curve.

-3 Something has gone terribly wrong.

-4 The normal to the Reg_CiPlane_Curve has 0 magnitude, i.e., the Points on the Reg_CiPlane_Curve are colinear.

-5 An error occurred in putting the Reg_CiPlane_Curve in one of the major planes.

-6 The Reg_CiPlane_Curve is non-planar.

Log

[LDF 2002.04.03] Added and tested all cases. Seems to work properly.

[LDF 2002.11.12] Added “relax” after the arguments to “\RV” in the TEX code above in order to suppress a space at the beginning of the first line of the following indented paragraph. I couldn’t figure out a way of suppressing the space within the definition of \RV (which is currently “\text” to \ARG).

[LDF 2003.06.03] Changed the line where Plane::get_distance() is called below. It now returns a real short, so “.first” has to be added, in order to get the real value.

[LDF 2003.06.13] Changed pt0.epsilon() to Point::epsilon().

[LDF 2003.06.14] Added error handling code for the case that get_plane() fails.

[LDF 2003.06.14] No longer taking absolute value of the real value r0 returned by Plane::get_distance(), since it will always be positive, anyway. Comment at place below, where I made this change.

[LDF 2003.07.01] Added argument suppress_warnings.

[LDF 2003.07.16] Changed name of center argument to ref_pt, because I’ve made center a data member of Reg_CiPlane_Curve.

(Declare Reg_CiPlane_Curve functions 987) \( \equiv \)

\[
\text{virtual signed short location(Point ref0, Point pt0, const bool suppress_warnings = false) const;}
\]
993.

(Define \texttt{Reg\_Cl\_Plane\_Curve} functions 993) \equiv
signed short \texttt{Reg\_Cl\_Plane\_Curve::location(Point ref\_pt, Point pt0, const bool suppress\_warnings)}
  const
  {
    bool DEBUG = false; /* true */
    if (DEBUG) {
      \texttt{cout} ≡ "Entering\texttt{Reg\_Cl\_Plane\_Curve::location}\n";
      \texttt{ref\_pt} show("ref\_pt");
      \texttt{pt0} show("pt0");
    }
    unsigned short orientation;
    const unsigned short X\_Y = 0;
    const unsigned short X\_Z = 1;
    const unsigned short Z\_Y = 2;
    Plane pl\texttt{(get\_plane());
    if (pl \equiv INVALID\_PLANE) /* LDF 2003.06.14. Added this error handling code. */
      \{
        \texttt{cerr} ≡ "ERROR!\texttt{Reg\_Cl\_Plane\_Curve::location()}:" ≡
          "The\texttt{Reg\_Cl\_Plane\_Curve} is\texttt{non-planar}.\n" ≡ "Returning\texttt{-6}\n" ≡ flush;
        \texttt{return -6;}
      \}
    real r0 = pl\texttt{.get\_distance(pt0).first;}
    if (r0 > Point::\texttt{epsilon()})
      /* LDF 2003.06.14. Changed. \texttt{r0} will always be positive, so I now longer take its absolute value. */
      \{
        if (!suppress\_warnings) \texttt{cerr} ≡ "WARNING!\texttt{Reg\_Cl\_Plane\_Curve::location()}:" ≡
          "Point\texttt{is not in plane of regular closed plane}\texttt{curve}.\n" ≡ "Returning\texttt{-2}\n" ≡ flush;
        \texttt{return -2;}
      \}
    \texttt{Reg\_Cl\_Plane\_Curve} copy(*this);
    if (ref\_pt \ne origin) pt0 \equiv ref\_pt \equiv \texttt{copy\_shift(-ref\_pt)}; /* LDF 2002.11.05. Simplified. */
    \texttt{Point copy\_normal = copy\_get\_normal();
    if (DEBUG) \texttt{copy\_normal}\texttt{.show("copy\_normal");
    if (copy\_normal\_magnitude() \equiv 0) \{
      \texttt{cerr} ≡ "ERROR!\texttt{Reg\_Cl\_Plane\_Curve::location()}:" ≡
        "Normal\texttt{has no magnitude}.\texttt{Returning\texttt{-4}}\n" ≡ flush;
      \texttt{return -4;}
    \}
    else if (fabs(copy\_normal\_get\_x()) < Point::\texttt{epsilon()} \&\&
      fabs(copy\_normal\_get\_y()) < Point::\texttt{epsilon()})
      \{
        if (DEBUG) \texttt{cout} ≡ "Regular\texttt{closed plane curve} is\texttt{already in} x-y plane.\n";
        \texttt{orientation} \equiv X\_Y;
      \}
    else if (fabs(copy\_normal\_get\_x()) < Point::\texttt{epsilon()} \&\&
      fabs(copy\_normal\_get\_z()) < Point::\texttt{epsilon()})
      \{
        if (DEBUG) \texttt{cout} ≡ "Regular\texttt{closed plane curve} is\texttt{already in} x-z plane.\n";
        \texttt{orientation} \equiv X\_Z;
      \}
else if (fabs(normal.get_x()) < Point::epsilon() && fabs(normal.get_y()) < Point::epsilon())
{
  if (DEBUG) cout << "Regular closed plane curve is already in y-z plane.\n";
  orientation = Z_Y;
}
else {
  if (DEBUG) cout << "Putting regular closed plane curve into x-z plane.\n";
  Transform t1;
  t1.align_with_axis(ref_pt, copy.get_point(0), 'x');
  copy *= ref_pt *= pt0 *= t1;
  Point pt34 (copy.get_point(number_of_points / 4));
  Transform t2;
  t2.align_with_axis(ref_pt, pt34, 'y');
  copy *= ref_pt *= pt0 *= t2;
  orientation = X_Z;
}
if (DEBUG) {
  cout << "orientation=" << orientation << endl << flush;
  copy.draw(Colors::blue);
  copy.show("copy");
  ref_pt.dotlabel("ref pt");
  pt0.dotlabel("pt 0");
  pt0.show("pt 0");
}
Transform t3;
Point pt1 = copy.get_point(0);
pt1 -= ref_pt;
pt1.unit_vector(true);
Point x_axis(1,0,0);
Point z_axis(0,0,1);
if (orientation == X_Y || orientation == X_Z) {
  if (pt1 != x_axis && pt1 != -x_axis) {
    t3.align_with_axis(ref_pt, pt1, 'x');
    copy *= ref_pt *= pt0 *= t3;
  }
}
else if (orientation == Z_Y) {
  if (pt1 != z_axis && pt1 != -z_axis) {
    t3.align_with_axis(ref_pt, pt1, 'y');
    copy *= t3;
    ref_pt *= t3;
    pt0 *= t3;
  }
}

real_pair rr;
real pt0_x; /* [LDF 2002.11.05.] Vertical */
real pt0_y; /* [LDF 2002.11.05.] Horizontal */
if (orientation == X_Y) {
  pt0_x = pt0.get_x();
  pt0_y = pt0.get_y();
} else if (orientation ≡ X_Z) {
  pt0.x = pt0.x();
  pt0.y = pt0.y();
}
else if (orientation ≡ Z_Y) {
  pt0.x = pt0.x();
  pt0.y = pt0.y();
}
else {
  cerr << "ERROR! In Reg Cl Plane Curve::location() \n"
        "orientation has invalid value: \n"
        "\nReturning -5 \n"
        "\nflush;"
  return -5;
}
rr = solve(ν’, pt0.h); /* [LDF 2002.11.05] TO DO: Explain. */
if (π_first ≡ INVALID_REAL ∧ rr.second ≡ INVALID_REAL) {
  if (DEBUG) {
    cout << "Point is outside regular, closed plane, curve. \n"
         "Returning -1. \n"
         "Exiting Reg Cl Plane Curve::location() \n";
    getchar();
  }
  return -1;
}
else if ((fabs(fabs(pt0.x) – fabs(rr.first)) < Point::epsilon()) ∨
         (fabs(fabs(pt0.y) – fabs(rr.second)) <
          Point::epsilon())) {
  if (DEBUG) {
    cout << "Point is on regular, closed plane, curve. \n"
         "Returning 0. \n"
         "Exiting Reg Cl Plane Curve::location() \n";
    getchar();
  }
  return 0;
}
else if (fabs(pt0.x) < fabs(rr.first)) {
  if (DEBUG) {
    cout << "Point is inside regular, closed plane, curve. \n"
         "Returning 1. \n"
         "Exiting Reg Cl Plane Curve::location() \n";
    getchar();
  }
  return 1;
}
else if (fabs(pt0.x) > fabs(rr.first))
  /* This case should never occur, I believe. [LDF 2002.11.05] Why not?? */
  {
  if (DEBUG) {
    cout << "Point is outside regular, closed plane, curve. \n"
         "Returning -1. \n"
         "Exiting Reg Cl Plane Curve::location() \n";
    getchar();
  }
  return -1;
}
else {
  cerr << "ERROR! In Reg Cl Plane Curve::location() \n"
        "This can't happen! Returning -3. \n"
        "flush;";
getchar();
    if (DEBUG) cout << "Exiting Reg_Cl_Plane_Curve::location()\n";
    return -3;
}

See also sections 995, 998, 999, 1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1009, and 1012.
This code is used in section 1015.

994. Angle point. [LDF 32.03.01.05.] TO DO: Find out why this function isn’t const!
(Declare Reg_Cl_Plane_Curve functions 987) +
    virtual Point angle_point(real angle);
995. 
(Define Reg_Cl_Plane_Curve functions 993) +≡
   Point Reg_Cl_Plane_Curve::angle_point(real angle)
   
   >>> errors <<
   "This virtual function doesn't have a real definition for u"
   "ordinary Reg_Cl_Plane_Curve::angle_point()
   .\n   Returning INVALID_POINT.\n   \n   return INVALID_POINT;
   
996. Intersection points.  [LDF 2002.11.05.] Intersection with a line. intersection_points() returns a
   bool_point_pair. The bool in a bool_point indicates whether the Point is on the line segment in
   question. If one of the bools is false, but the Point is not INVALID_POINT, then the line (as opposed
   to the segment) does intersect the Reg_Cl_Plane_Curve, and the Point indicates one of the intersection
   points. So, intersection_points() can be used whether you want to restrict the intersection points to ones
   that are actually on a particular line segment or not.
   TO DO: [LDF 2002.04.12.] In the specializations, I should check whether the intersection points are on
   the curve in question. I should also write a version of this for Lines, where there's no test for whether
   the intersection points are on the segment, since there's no segment.
   
   The versions of intersection_points() belonging to classes derived from Reg_Cl_Plane_Curve will most
   likely call on the functions described in this section, passing center as the ref_pt argument. However, this
   need not be the case, and Reg_Cl_Plane_Curves need not have a meaningful center. ref_pt in these functions
   merely refers to the Point which should be placed at the origin by the transformation. [LDF 2003.07.16]

997. Point arguments.  

---

[Log]

[LDF 2003.06.20.] Rewrote this function. The perpendicular and non-parallel, non-coplanar cases are
handled in exactly the same way. In these cases, there can only be one intersection point.

Plane::intersection_point() and Reg_Cl_Plane_Curve::location() are now used to find it, if it exists.

[LDF 2003.06.20.] The coplanar case was the one that was causing difficulty. The copy of *this is now
always put into the x-z plane, even if it is in one of the major planes, or in a plane parallel to one of these.
The advantage of this is that it simplifies the code. The disadvantage is, that additional reductions rotate
the accuracy of the calculation of the intersection points.

[LDF 2003.06.20.] Transform::align_with_axis() is no longer used for putting the copy of *this and the
line into the x-z plane. It might be possible to use it, but I used Point::angle() while debugging, in order
to see what was happening better. It might be possible to go back to using Transform::align_with_axis,
but I don't see any advantage to doing so.

[LDF 2003.06.20.] I've tested this function for coplanar lines for planes with various orientations. I hope
that it works properly for all planes now!

[LDF 2003.07.01.] Added true as silent argument to unit_vector() when I call it on cross. This prevents
unit_vector() from issuing a warning message, when cross has magnitude 0, which occurs when surface_vector
and pt_vector are colinear. Since this case is handled correctly, the warning messages are unnecessary and
distracting.

[LDF 2003.07.01.] BUG FIX: Made changes to the way on_segment is used in the coplanar case. The way
it was before handled certain cases wrong.

[LDF 2003.07.01.] Removed unreachable statement at end of function: return bpp. GCC didn't complain,
but the DEC compiler issued a warning.

[LDF 2003.07.16.] Changed name of center argument to ref_pt, because I've made center a data member
of Reg_Cl_Plane_Curve.

(Declare Reg_Cl_Plane_Curve functions 987) +≡
virtual bool_point_pair intersection_points(Point refPt, Point pt0, Point pt1) const;

998.
(Define Reg_Ci_Plane_Curve functions 993) +≡
bool_point_pair Reg_Ci_Plane_Curve::intersection_points(Point refPt, Point pt0, Point pt1)
    const { bool DEBUG = false; /* true */
        if (DEBUG)
            cout << "Entering Reg_Ci_Plane_Curve::intersection_points()\n" << flush;
        bool_point_pair bpp = INVALID_BOOL_POINT_PAIR;
        Plane pl = getPlane();
        if (DEBUG) pl.normal.show("pl.normal");
        Point surface_vector = (getPoint(0) - refPt);
        surface_vector.unit_vector(true);
        Point pt_vector(pt1 - pt0);
        pt_vector.unit_vector(true);
        Point cross = surface_vector.cross_product(pt_vector);
        cross.unit_vector(true, true);
        if (DEBUG) {
            surface_vector.show("surface_vector");
            pt_vector.show("pt_vector");
            cross.show("cross");
            pl.normal.show("pl.normal");
        }
        short distance = pl.get_distance(pt0).second;
    }

999. Degenerate cases, error handling.
(Define Reg_Ci_Plane_Curve functions 993) +≡
if (pt_vector ≡ INVALID_POINT ∨ pl.normal ≡ INVALID_POINT ∨ pt_vector ≡ origin ∨ pl.normal ≡ origin)
    {
        cerr << "ERROR! InvalidReg_Ci_Plane_Curve:intersection_points()\n"
             "Something is wrong with the normals: pt_vector:\n"
             pt_vector.show("pt_vector");
             pl.normal.show("pl.normal");
        cerr << "Returning INVALID_BOOL_POINT_PAIR.\n" << flush;
        if (DEBUG) cout << "Exiting Reg_Ci_Plane_Curve:intersection_points()\n" << flush;
        return INVALID_BOOL_POINT_PAIR;
    }

1000. Parallel and coplanar cases.
(Define Reg_Ci_Plane_Curve functions 993) +≡
else if (surface_vector ≡ pt_vector ∨ surface_vector ≡ -pt_vector ∨ cross ≡ pl.normal ∨ cross ≡ -pl.normal)
§1001.  Coplanar case.

(Define Reg_Cl_Plane_Curve functions 993) +\[\]
if (distance == 0) {
  if (DEBUG) cout << "Line and Reg_Cl_Plane_Curve are coplanar.\n";
  Transform tol:
  Reg_Cl_Plane_Curve copy(*this);
  Point curve_0 = copy.get_point(0);
  if (DEBUG) {
    cout << "After copying:" << endl;
    show("this:");
    copy.show("copy:");
    curve_0.show("curve_0");
  }
  t0 == curve_0 == pt0 == pt1 == copy.shift(-ref_pt);
  if (DEBUG) {
    cout << "After shift:" << endl;
    copy.show("copy:");
    curve_0.show("curve_0");
    pt0.show("pt0");
    pt1.show("pt1");
  }
  if (curve_0.get_x() < 0) {
    t0 == curve_0 == pt0 == pt1 == copy.rotate(0, 0, 180);
  }
  if (DEBUG) {
    cout << "After rotating, curve_0 has positive x:" << endl;
    copy.show("copy:");
    curve_0.show("curve_0");
    pt0.show("pt0");
    pt1.show("pt1");
  }
  Point trace_0 = curve_0;
  trace_0.shift(0, -curve_0.get_y());
  Point x_axis_pt(1);
  real ang = trace_0.angle(x_axis_pt);
  if (DEBUG) {
    cout << "ang_u=\n" << ang << endl << flush;
  }
  if (ang != 0) {
    if (curve_0.get_x() > 0) ang *= -1;
    t0 == curve_0 == pt0 == pt1 == copy.rotate(0, ang);
  }
  if (DEBUG) {
    cout << "After rotating, the trace of curve_0 is on x-axis:" << endl;
    copy.show("copy:");
    curve_0.show("curve_0");
    pt0.show("pt0");
    pt1.show("pt1");
  }
  ang = curve_0.angle(x_axis_pt);
if (ang \neq 0) 
  if (curve.A.get_y() > 0) ang *= -1;
  t0 *= curve.A *= pt0 *= pt1 *= copy.rotate(0, 0, ang);
}

if (DEBUG) 
  cout \ll \"After rotating, so curve 0 is on x-axis:\" \ll endl;
  copy.show("copy: ");
  pt0.show("pt0");
  pt1.show("pt1");

Point curve_A = copy.get_point(number_of_points / 4);
Point z_axis_pt(0, 0, 1);
ang = curve_A.angle(z_axis_pt);
if (DEBUG) 
  curve_A.show("curve_4");
  cout \ll \"ang_\_z=x\" \ll ang \ll endl \ll flush;
}

if (ang \neq 0) 
  if (curve_A.get_y() > 0) ang *= -1;
  t0 *= curve_A *= pt0 *= pt1 *= copy.rotate(ang);
}

if (DEBUG) 
  cout \ll \"After rotating, so curve 4 is on x-axis:\" \ll endl;
  copy.show("copy: ");
  curve_A.show("curve_4");
  pt0.show("pt0");
  pt1.show("pt1");

if (DEBUG) 
  copy.draw(Colors::blue);
  pt0.draw(pt1, Colors::black, "evenly");
  for (int i = 0; i < 16; i += 4) copy.get_point(i).dotlabel(i);
  pt0.dotlabel("pt0");
  pt1.dotlabel("pt1");
  copy.get_center().dotlabel("copy\_center");
#endif
draw_axes();

real pt0_x = pt0.get_x();
real pt0_y = pt0.get_y();
real pt1_x = pt1.get_x();
real pt1_y = pt1.get_y();
real pt1_z = pt1.get_z(); /* pt1_z isn't used. Leaving it here, just in case. [LD 03.08.27] */
real Slope = pt1.slope(pt0, pt1_x, pt1_z);
if (DEBUG) cout \ll \"Slope_\_z=x\" \ll Slope \ll endl \ll flush;
real rrr; /* BEGIN */
1002. Slope is 0 (line is horizontal).

\begin{verbatim}
(Define Reg_Cl_Plane_Curve functions 993 +\equiv
if (Slope \equiv 0) /* v is known, h is unknown. */
{
  if (DEBUG) {
    cout << "Slope\equiv0" << endl << flush;
  }
  rr = solve('h',ptl.v);
  if (rr.first \neq INVALID_REAL) {
    bpp.first.pt.set(rr.first,0,ptl.v);
  }
  else bpp.first.pt = INVALID_POINT;
  if (rr.second \neq INVALID_REAL) {
    bpp.second.pt.set(rr.second,0,ptl.v);
  }
  else bpp.second.pt = INVALID_POINT;
  if (DEBUG) {
    bpp.first.pt.show("bpp.first.pt");
    bpp.second.pt.show("bpp.second.pt");
  }
} /* End Slope \equiv 0. */
\end{verbatim}

1003. Slope is undefined (line is vertical).

\begin{verbatim}
(Define Reg_Cl_Plane_Curve functions 993 +\equiv
else
  if (Slope \equiv INVALID_REAL) {
    if (DEBUG) {
      cout << "Line is vertical.\n";
    }
    rr = solve('v',ptl.h);
    if (rr.first \neq INVALID_REAL) {
      bpp.first.pt.set(ptl.h,0,rr.first);
    }
    else bpp.first.pt = INVALID_POINT;
    if (rr.second \neq INVALID_REAL) {
      bpp.second.pt.set(ptl.h,0,rr.second);
    }
    else bpp.second.pt = INVALID_POINT;
} /* End Slope \equiv INVALID_REAL. */
\end{verbatim}
1004. Slope ∈ \texttt{real} is defined and ≠ 0.
(Define \texttt{Reg\_Cl\_Plane\_Curve} functions 993) + ⋆
else {
    \texttt{real v\_intercept;}
    v\_intercept = pt0.w - (Slope * pt0.z);
    if (DEBUG) {cout << "v\_intercept = " << v\_intercept; << endl; << flush;
    }
    \texttt{real\_triple coeffs = get\_coefficients(Slope, v\_intercept);}  /* New b-values. */
    if (is\_quadratic()) {
        if (DEBUG) {
            cout << "Solving quadratic. \n" << flush;
        }
        rr = solve\_quadratic(coeffs\_first, coeffs\_second, coeffs\_third);
    } else {
        cout << "Not a quadratic. \n" << "Haven’t programmed this case yet. \n" << flush;
    }
    \texttt{real v\_coord;}
    if (rr\_first ≠ INVALID\_REAL) {
        v\_coord = (Slope * rr\_first) + v\_intercept;
        bpp\_first\_pt.set(rr\_first, 0, v\_coord);
    } else bpp\_first\_pt = INVALID\_POINT;
    if (rr\_second ≠ INVALID\_REAL) {
        v\_coord = (Slope * rr\_second) + v\_intercept;
        bpp\_second\_pt.set(rr\_second, 0, v\_coord);
    } else bpp\_second\_pt = INVALID\_POINT;
1005. Common code for the “coplanar” case.

(Define Reg_C1 Plane_Curve functions 993) \(\equiv\)

```cpp
bool real on_segment;

if (bpp_first.pt \equiv INVALID_POINT) {
    on_segment.first = false;
    on_segment.second = INVALID_REAL;
}
else on_segment = bpp_first.pt.is_on_segment(pt0, pt1);
if (DEBUG) {
    cout \ll "on_segment.first=\n" \ll on_segment.first \ll endl \ll flush;
    cout \ll "on_segment.second=\n" \ll on_segment.second \ll endl \ll flush;
}
if (on_segment.first \equiv true) bpp_first.b = true;
else bpp_first.b = false;
Transform \(t\)_inverse;
\(t\)_inverse = \(t\)0.inverse();
if (bpp_first.pt \neq INVALID_POINT) {
    if (DEBUG) cout \ll "Transforming\(\)bpp_first.pt\n" \ll flush;
    bpp_first.pt *= \(t\)_inverse;
}
else {
    if (DEBUG) cout \ll "bpp.first.pt\is\invalid\n" \ll flush;
}
if (bpp_second.pt \equiv INVALID_POINT) {
    on_segment.first = false;
    on_segment.second = INVALID_REAL;
}
else on_segment = bpp_second.pt.is_on_segment(pt0, pt1);
if (on_segment.first \equiv true) bpp_second.b = true;
else bpp_second.b = false;
if (DEBUG) {
    cout \ll "on_segment.first=\n" \ll on_segment.first \ll endl \ll flush;
    cout \ll "on_segment.second=\n" \ll on_segment.second \ll endl \ll flush;
}
if (bpp_second.pt \neq INVALID_POINT) {
    if (DEBUG) cout \ll "Transforming\(\)bpp.second.pt\n" \ll flush;
    bpp_second.pt *= \(t\)_inverse;
}
else {
    bpp_second.pt = INVALID_POINT;
    if (DEBUG) cout \ll "bpp.second.pt\is\invalid\n" \ll flush;
}
if (DEBUG) {
    cout \ll "rr.first=\n" \ll rr.first \ll endl \ll flush;
    cout \ll "rr.second=\n" \ll rr.second \ll endl \ll flush;
    cout \ll "bpp.first.b=\n" \ll bpp.first.b \ll endl \ll flush;
    cout \ll "bpp.second.b=\n" \ll bpp.second.b \ll endl \ll flush;
    bpp_first.pt.show("bpp.first.pt");
    bpp_second.pt.show("bpp.second.pt");
}
return bpp; } /* End of coplanar case. */
```
1006. Parallel case.

(Define Reg_Cl_Plane_Curve functions 993) +≡
else {
  cerr ≡ "WARNING! In Reg_Cl_Plane_Curve::intersection_points():\n" ≡
"Line_ and Reg_Cl_Plane_Curve are in parallel planes.\n" ≡
"No intersections. Returning INVALID_BOOL_POINT_PAIR." ≡ endl ≡ endl ≡ flush;
  return INVALID_BOOL_POINTPAIR;
} /* End of parallel and coplanar cases. */

1007. Perpendicular and non-coplanar cases. [LDF 2003.06.13.] These cases are handled in exactly the same way.

(Define Reg_Cl_Plane_Curve functions 993) +≡
else {
  if (pl.normal ≡ pt.vector ∨ pl.normal ≡ -pt.vector) {
    if (DEBUG)
      cout ≡ "The line is perpendicular to the_" ≡ "Reg_Cl_Plane_Curve.\n" ≡ flush;
  } else {
    if (DEBUG)
      cout ≡ "The line and the_Reg_Cl_Plane_Curve_" ≡ "are non-coplanar.\n" ≡ flush;
  }
  bool_point bp = pl.intersection_point(pt0, pt1);
  if (DEBUG) {
    bp.pt.show("bp.pt");
  }
  short s = location(ref.pt, bp.pt);
  if (DEBUG) cout ≡ "location: _u_\n_u_" ≡ s ≡ endl ≡ flush;
  if (s > -1) {
    bpp.first.pt = bp.pt;
    bpp.first.b = bp.pt.is_on_segment(pt0, pt1).first;
    if (DEBUG) cout ≡ "Un_{segment: _u_\n_u_" ≡ bpp.first.b ≡ endl ≡ flush;
    return bpp;
  } else return INVALID_BOOL_POINTPAIR;
} /* End of “Perpendicular and non-coplanar cases”. */

1008. Path arguments.

Log

[LDF 2003.06.20.] Added this function.
[LDF 2003.07.16.] Changed name of center argument to ref.pt, because I’ve made center a data member of Reg_Cl_Plane_Curve.

(Declare Reg_Cl_Plane_Curve functions 987) +≡
bool_point_pair intersection_points(const Point &ref.pt, const Path &p) const;
1009. (Define Reg_Cl_Plane_Curve functions 993) +
bool_point_pair Reg_Cl_Plane_Curve::intersection_points(const Point &ref_pt, const Path &p)
const
{
  if (!p.is_linear()) {
    cerr << "ERROR in Reg_Cl_Plane_Curve::intersection_points(): \n" << "Path argument is non-linear. Returning INVALID_BOOL_POINT_PAIR. \n" << flush;
    return INVALID_BOOL_POINTPAIR;
  }
  return intersection_points(ref_pt, p.get_point(0), p.get_last_point());
}

1010. Reg_Cl_Plane_Curve segments. The functions in this section require that the
Reg_Cl_Plane_Curve have a meaningful center, in order to make it possible to rotate the segments.
[LDF 2003.07.16.]

Log
[LDF 2003.07.16.] Added this section and its subsections, including the declarations and definitions of
segment(), half(), and quarter(). They were formerly members of Circle.

1011. Segment. [LDF 2002.10.11.] segment() returns a subpath of the Reg_Cl_Plane_Curve repre-
senting a segment of *this.*

int factor          Determines how large a segment of the Reg_Cl_Plane_Curve is returned. factor must
real angle          be > 1 and less than or equal to the number of points on the Reg_Cl_Plane_Curve.
bool closed         If true, the Path is made a “cycle” and the ends of the segment are joined by concatenating
                   the curved Path with the straight line segment from its last to its first Point using the
                   connector "&".

[LDF 2003.07.27.] TO DO: Make arguments const, if possible. angle can’t be, though. If factor \equiv
                   number_of_points, return *this, cast to a Path, with warning.

Log
[LDF 2002.11.12.] Added “\relax” after the arguments to “\ARG” in the \TeX code above in order to
suppress a space at the beginning of the first line of the following indented paragraph. I couldn’t figure out
a way of suppressing the space within the definition of \ARG.
[LDF 2003.06.20.] Changed the way the last connector is set when closed \equiv true.
[LDF 2003.07.27.] Made const.
[LDF 2003.08.20.] BUG FIX: Added unsigned short subpath_size. Changed the way the subpath is
created, when closed is true. Now concatenating the curved subpath with the straight line segment from
the last to the first Points of the subpath using "&".

(Declare Reg_Cl_Plane_Curve functions 987) +
Path segment(unsigned int factor, real angle = 0, bool closed = true) const;
1012. Define `Reg_C1_Plane_Curve` functions 993)

Path `Reg_C1_Plane_Curve::segment(unsigned int factor, real angle, bool closed) const`

```cpp
Path p
if (factor <= 1 || factor > number_of_points) {
    cerr << "ERROR! In Reg_C1_Plane_Curve::segment():\n" << "The argument factor has an invalid value: \n" << "\nReturning empty Path.\n";
    return p;
}
if (fabs(angle) > 360) {
    cerr << "WARNING! In Reg_C1_Plane_Curve::segment():\n" << "The argument angle is greater than 360; \n" << "It will be reduced.\n" << flush;
}
unsigned short subpath_size = (number_of_points/factor) + 1;
p = subpath(0, subpath_size, false, "...");
for (unsigned short i = 1; i < subpath_size - 1; ++i) p += ";";
if (closed) {
    p += ";n;
p += p.get_last_point();
p += ";n;
p += p.get_point(0);
p += ";n;
p.set_cycle();
}
angle = fmod(angle, 360);
if (angle != 0) {
    Point normal = get_normal();
    normal.shift(center);
    p.rotate(center, normal, angle);
}
return p;
}
```

1013. Half. `half()` creates a curve using half of the points in `points` starting from point 0. If the argument `angle` is not zero, the resulting `Path` is rotated by that amount about a line from `center` in the direction of the normal to the `Reg_C1_Plane_Curve`. If the argument `closed` is true, then the segment is closed and can be filled using `fill()` or `filldraw()`.

(Declare `Reg_C1_Plane_Curve` functions 987)

```cpp
inline Path half(real angle = 0, bool closed = true) const
{
    return segment(2, angle, closed);
}
```

1014. Quarter. `quarter()` creates a curve using a quarter of the points in `points` starting from point 0. If the argument `angle` is not zero, the resulting `Path` is rotated by that amount about a line from `center` in the direction of the normal to the `Reg_C1_Plane_Curve`. If the argument `closed` is true, then the segment is closed and can be filled using `fill()` or `filldraw()`.
§1014  3DLDF-1.1.5.1

{Declare Reg.Cl_Plane_Curve functions 987} +≡
   inline Path {quarter(real angle = 0, bool closed = true) const
   {
       return segment(4, angle, closed);
   }

   This is what’s compiled.
{Include files 6}
{Version control identifier 5}
{Define class Reg.Cl_Plane_Curve 985}
{Define Reg.Cl_Plane_Curve functions 993}
1016. This is what's written to curves.h.
   \{curves.h 1016\} ≡
   \{Define class Reg_Cl_Plane_Curve 985\}

1017. Polygon \{polygons.web\}.

Log

[LDF 2003.07.18.] Removed the transformation sections from Reg_Polygon, and made them members of Polygon. Also, removed the Rectangle versions in rects.web.
[LDF 2003.11.12.] Removed the version control identifiers from the CWEB files for the distribution of 3DLDF 1.1. They're still used in my development versions.
[LDF 2003.12.10.] Put the version control identifiers back into my release versions for 3DLDF 1.1.4. I've already put some of them back in, now I'm doing the rest of them. However, the release versions are now in their own RCS repository.

format Polygon Path
\{Version control identifier 5\} ≡
\{static string res_id = "$Id: polygons.web,v 1.4 u 2004/01/12 01:32:08 ufinsto u Exp$";\}

1018. Include files.
   \{Include files 6\} ≡
   \#include "loader.h"
   \#include "pspg1b.h"
   \#include "creatnew.h"
   \#include "io.h"
   \#include "colors.h"
   \#include "transfor.h"
   \#include "shapes.h"
   \#include "pictures.h"
   \#include "points.h"
   \#include "lines.h"
   \#include "planes.h"
   \#include "paths.h"
   \#include "curves.h"

1019. Polygon class definition. LDF Undated. Polygon is derived from Path. This makes sense, because a Polygon is really just a kind of Path. This way, we don't have to define the drawing and filling functions, or the transformations.
   \{Define class Polygon 1019\} ≡
   class Polygon: public Path {
     protected: Point center;
     public: \{Declare Polygon functions 1022\}
   };

This code is used in sections 1097 and 1098.
1020. Returning elements and information.

1021. Get center.

[Log]

[LDF 2003.07.18.] Moved these functions from Reg_Polygon to Polygon. Also removed the Rectangle
types, since Rectangle inherits the Polygon versions.

1022. non-const version.

[Log]

[LDF 2002.04.24.] Added this function.
[LDF 2003.06.09.] Changed return value from Point & to const Point &.

(Declare Polygon functions 1022) ≡

   virtual const Point &get_center();
See also sections 1024, 1028, 1037, 1039, 1045, 1047, 1050, 1052, 1054, 1056, 1059, 1061, 1064, and 1066.
This code is used in section 1019.

1023.

(Define Polygon functions 1023) ≡

   const Point &Polygon::get_center()
   {
      if (points.size() == 0)  /* LDF 2002.09.27. Added this error handling code. If the Polygon is
                                 empty, don’t return center. */
      {
         cerr << "WARNING! In Polygon::get_center():\n               \n              there are no Points.\n              \n              This is presumably bad.\n              \n              Invalid Point.\n              \n              Return const_cast(Point &)(INVALID_POINT);\n         return const_cast(Point &)(INVALID_POINT);
      }
      center.apply_transform();
      return center;
   }
See also sections 1025, 1029, 1030, 1031, 1032, 1033, 1034, 1035, 1036, 1038, 1040, 1041, 1042, 1043, 1046, 1048, 1051, 1053,
1055, 1057, 1060, 1062, 1065, and 1067.
This code is used in section 1097.

1024. const version. [LDF 2002.09.27.] Note that this version returns a Point whereas the non-const
version returns a Point &. That’s because p is a local variable in this function and it would be an error to
return a reference to it. [LDF 2002.04.24.] Added this function.

   (Declare Polygon functions 1022) ≡
   Point get_center() const;
1025.

(Define Polygon functions 1023) \(\equiv\)

Point Polygon::get_center() const
{
    if (points.size() == 0)  /* LDF 2002.09.27. Added this error handling code. If the Polygon is empty, don't return center. */
    {
        cerr \ll "WARNING! Polygon::get_center():\n        \ll "Polygon\n        \ll "doesn't have a center.\n        \ll "Returning INVALID_POINT.\n        \ll "flush;
        return const_cast(Point &)(INVALID_POINT);
    }
    Point p(center);
    p.apply_transform();
    return p;
}

1026. Intersections.

1027. Intersection with a line.  [LDF 2003.06.13] A line can intersect with a Polygon at two points at most.

1028. Point version.

<table>
<thead>
<tr>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>[LDF 2003.06.13] Added this function.</td>
</tr>
<tr>
<td>[LDF 2003.06.17] Minor change. Now using get_point(0) and center instead of get_point(0) and get_last_point() to generate surface_vector.</td>
</tr>
</tbody>
</table>

(Declare Polygon functions 1022) \(\equiv\)

bool_point_pair intersection_points(const Point &p0, const Point &p1) const;
1029. (Define Polygon functions 1023) \( \equiv \)

**bool point_pair intersection_points(const Point &pt0, const Point &pt1)**

\{ bool DEBUG = false; /* true */
if (DEBUG) cout << "Entering Polygon::intersection_points()\n" << flush;

**bool point_pair bpp = INVALID_BOOL_POINT_PAIR; /* The return value. [LDF 2003.06.13. */
** Plane pl = get_plane();
** Point pt_vector(pt1 - pt0);
** Point surface_vector(get_point(0) - center);

if (DEBUG) {
    pl.show("pl.point ");
    pl.show("pl.normal ");
    pt0.show("pt0 ");
    pt1.show("pt1 ");
    pt_vector.show("pt_vector ");
    surface_vector.show("surface_vector ");
}

**Point cross = surface_vector . cross_product(pt_vector);**

cross . unit_vector(true);
if (DEBUG) {
    cross.show("cross ");
    pl.show("pl.normal ");
}

short distance = pl . get_distance(pt0 . second);

1030. Degenerate cases, error handling.

(Define Polygon functions 1023) \( \equiv \)

if (pt_vector . equals(INVALID_POINT) \&\& pl.normal . equals(INVALID_POINT) \&\& pt_vector . equals(origin) \&\& pl.normal . equals(origin))
\
{ cerr << "ERROR! InvalidPolygon::intersection_points():\n" << "Something is wrong with the normals:\n";
    pt_vector.show("pt_vector ");
    pl.normal.show("pl.normal ");
    cerr << "Returning INVALID_BOOL_POINTPAIR.\n" << flush;
    if (DEBUG) cout << "Exiting Polygon::intersection_points() \n" << flush;
    return INVALID_BOOL_POINTPAIR;
}

1031. Parallel and coplanar cases.

[Log]

[LDf 2003.06.20.] Now checking surface_vector against pt_vector in the following conditional.

(Define Polygon functions 1023) \( \equiv \)

else if (surface_vector . equals(pt_vector) \&\& surface_vector . equals(-pt_vector) \&\& cross . equals(pl.normal) \&\& cross . equals(-pl.normal))
\
{
1032. Coplanar case. [LDF 2003.06.13] Only those intersection points that are on the line segments making of the Polygon are returned in bpp. If an intersection Point lies on both segments, the bool part of the bool_point will be true; otherwise false. If a Point lies on a line segment belonging to the Polygon, but not to the line segment pt0pt1, the Point will be put into the bool_point, but the bool will be false. The reason for this is, that the angles of the sides of the Polygon can cause intersection points to be found, that the user probably won’t want.

{Define Polygon functions 1023} +
if (distance <= 0) {
    if (DEBUG) cout << "Line_\_and_\_Polygon_\_are_\_coplanar.\n";
    bool found = false;
    bool_point bp;
    Point q0;
    Point q1;
    if (DEBUG) show("this");
    for (vector<Point> ::const_iterator iter = points.begin(); iter != points.end(); ++iter) {
        if ((iter + 1) == points.end()) {
            if (DEBUG) cout << "Doing_\_last_\_segment_\n";
            q0 = *(points.back());
            q1 = *(points.front());
        } else {
            if (DEBUG) cout << "Doing_\_normal_\_segment_\n";
            q0 = **iter;
            q1 = **(iter + 1);
        }
        bp = Point :: intersection_point(pt0, pt1, q0, q1);
        if (bp.b) /* Intersection point is on both segments. */
            if (!found) {
                bpp.first.b = true;
                bpp.first.pt = bp.pt;
                found = true;
            } else if (DEBUG) cout << "Found_\_first_\_intersection_\n";
        else {
            bpp.second.b = true;
            bpp.second.pt = bp.pt;
            if (DEBUG) cout << "Found_\_second_\_intersection_\nReturning_\n";
            return bpp;
        }
    }
}
else if (bp.pt != INVALID_\_POINT\_\_or_\_pt.is_on_\_segment(q0, q1).first) {
    if (!found) {
        bpp.first.b = false;
        bpp.first.pt = bp.pt;
        found = true;
        if (DEBUG) cout << "Found_\_first_\_intersection_\n";
    } else {
        bpp.second.b = false;
        bpp.second.pt = bp.pt;
    }
}
if (DEBUG) cout << "Found second intersection. Returning. \n";
    return bpp;
} else continue;
} /* for */
return bpp;
} /* End of coplanar case. */

1033. Parallel case.
(Define Polygon functions 1023) +=
else {
    cerr << "WARNING! In Polygon::intersection_points(): \n"
        << "Line and Polygon are in parallel planes. \n"
        << "No intersections. Returning INVALID_BOOL_POINT_PAIR. \n"
        << endl << endl << flush;
return INVALID_BOOL_POINT_PAIR;
} /* End of parallel and coplanar cases. */
1034. Perpendicular and non-coplanar cases. [LDF 2003.06.13.] These cases are handled in exactly the same way.

\begin{verbatim}
{Define Polygon functions 1023} +
else {
    if (pl.normal \equiv pt.vector \lor pl.normal \equiv -pt.vector) {
        if (DEBUG) cout \ll "The line is \perp to the Polygon.\n" \ll flush;
    } else {
        if (DEBUG) cout \ll "The line and the Polygon are non-coplanar.\n" \ll flush;
    }
    bool_point bp = pl.intersection_point(pt0, pt1);
    if (DEBUG) {
        bp.pt.show("bp.pt");
    }
    if (bp.pt \equiv center) {
        if (DEBUG) {
            cout \ll "bp.pt=\equiv center.\n" \ll endl \ll flush;
        }
        bpp.first.pt = bp.pt;
        bpp.first.b = bp.pt.is_on_segment(pt0, pt1).first;
        return bpp;
    }
    Point r0;
    Point r1;
    if (DEBUG) {
        show("this:");
        center.show("center");
        cout \ll "points.size()\equiv endl \ll flush;
    }
    for (vector<Point> &point : const_iterator iter = points.begin(); iter \neq points.end(); ++iter) {
        if ((iter + 1) \equiv points.end()) {
            r0 = *(points.back());
            r1 = *(points.front());
            if (DEBUG) {
                r0.show("r0");
                r1.show("r1");
            }
        } else {
            r0 = **iter;
            r1 = **(iter + 1);
        }
        if (bp.pt \equiv r0 \lor bp.pt \equiv r1) {
            if (DEBUG) {
                if (bp.pt \equiv r0) cout \ll "bp.pt=\equiv r0.\n" \ll endl \ll flush;
                else if (bp.pt \equiv r1) cout \ll "bp.pt=\equiv r1.\n" \ll endl \ll flush;
            }
            bpp.first.pt = bp.pt;
            bpp.first.b = bp.pt.is_on_segment(pt0, pt1).first;
            return bpp;
        }
    }
\end{verbatim}
1035. [LDF 2003.06.24] DEBUG is passed as the verbose argument to is_in_triangle(). So, if
intersection_points() is being debugged, is_in_triangle() will print more information. However, DEBUG in
is_in_triangle() will not be set to true.

{Define Polygon functions 1023} +=
else
  if (bp.pt.is_in_triangle(center, r0, r1, DEBUG)) {
    if (DEBUG) cout << "IntersectionPoint::is_within_triangle.\n";
    bpp.first.pt = bp.pt;
    bpp.first.b = bp.pt.is_on_segment(pt0, pt1).first;
    return bpp;
  }
} /* End of “Perpendicular and non-coplanar cases”. */

1036. End of definition.

{Define Polygon functions 1023} +=
if (DEBUG) cout << "Exiting Polygon::intersection_points.\n\n" << flush;
return bpp; }

1037. Path version.

{Declare Polygon functions 1022} +=
bool_point_pair intersection_points(const Path &p) const;

1038.
{Define Polygon functions 1023} +=
bool_point_pair Polygon::intersection_points(const Path &p) const
{
  if (~p.is_linear()) {
    cerr << "ERROR! In Polygon::intersection_points(const Path &p):\n";
    "Path p.is_non-linear. Returning INVALID_BOOL_POINT_PAIR.\n\n" << flush;
    return INVALID_BOOL_POINT_PAIR;
  }
  return intersection_points(p.get_point(0), p.get_last_point());
}

1039. Intersection with another Polygon. TO DO: Explain what this function does and how it
works. [LDF 2003.06.29]

TO DO: Find out where unit_vector() gets called, when this function is called, and try to pass true as its
silent argument. [LDF 2003.07.16]

Log

[LDF 2003.06.29] Replaced the dummy definition of this function with a real one.

{Declare Polygon functions 1022} +=
vector(Point) intersection_points(const Polygon &r) const;
### 1040. Define Polygon functions 1023

```cpp
vector<Point> Polygon::intersection_points(const Polygon &r) const
{
    bool DEBUG = false;
    /* true */
    vector<Point> v;
    Plane pl = get_plane();
    Plane r_pl = r.get_plane();
    if (DEBUG) {
        pl.normal.show("pl.normal");
        r_pl.normal.show("r_pl.normal");
        cout << "pl.distance_" << pl.distance << endl << flush;
        cout << "r_pl.distance_" << r_pl.distance << endl << flush;
    }
    real distance = fabs(fabs(pl.distance) - fabs(r_pl.distance));
    if (distance < Point::epsilon()) distance = 0;
    if (DEBUG) cout << "distance_" << distance << endl << flush;
    if (pl.normal == r_pl.normal) {
        1041. Coplanar case.

        (Define Polygon functions 1023) +
        if (distance == 0) {
            if (DEBUG) cout << "Coplanar.\n"
            bool_point bp;
            Point *ptr;
            Point *r_ptr;
            for (vector<Point>::const_iterator iter = points.begin(); iter != points.end(); ++iter) {
                if ((iter + 1) == points.end()) ptr = points.front();
                else ptr = *(iter + 1);
                for (vector<Point>::const_iterator r_iter = r.points.begin(); r_iter != r.points.end(); ++r_iter) {
                    if ((r_iter + 1) == r.points.end()) r_ptr = r.points.front();
                    else r_ptr = *(r_iter + 1);
                    bp = Point::intersection_point(*iter, *ptr, *r_iter, *r_ptr);
                    if (bp.b) v.push_back(bp.pt);
                } /* Inner for */
            } /* Outer for */
            return v;
        }
    }

    1042. Parallel case.

    (Define Polygon functions 1023) +=
    else /* Parallel. */
    {
        cerr << "WARNING! Polygon::intersection_points():\n" <<
             "The polygons lie in parallel planes.\n" << "Returning empty vector<Point>...\n"
             << flush;
        return v;
    }
```
Non-parallel, non-coplanar case. \( v \) will contain the intersection points of the Line \( l \) with *this and \( r \), if any. \( v \) can contain a maximum of four Points in this case. \( v[0] \) and \( v[1] \) will be the intersection points of the Line \( l \) with *this, and \( v[2] \) and \( v[3] \) the intersection points of \( l \) and \( r \), if they exist.

If any intersection point doesn't exist, INVALID_POINT will be stored in the corresponding element of \( v \) as a placeholder. [LDF 2003.06.29]

The values in \( v \) provide information about the relative positions of the Polygons, e.g., whether they touch, whether lies within the perimeter of the other, etc. However, it's not possible to include this information in the return value, since the latter is merely a vector(Point). The routine below may need to be put into another function in order to use this information. It may be of importance in breaking up Polygons and Solids for an improved surface hiding routine. [LDF 2003.06.29]

(Define Polygon functions 1023) »

```cpp
if (DEBUG) cout << "Non-coplanar, non-parallel. \n"
Line l = pl.intersection_line(r, pl);
bool_point_pair bpp = intersection_points(l.position, (l.position + l.direction));
v.push_back(bpp.first.pt);
if (bpp.first.pt == bpp.second.pt) {
  if (DEBUG) cout << "bpp.first.pt and bpp.second.pt are equal for \*this. \n"
  v.push_back(INVALID_POINT);
}
else v.push_back(bpp.second.pt);
 bpp = r.intersection_points(l.position, (l.position + l.direction));
v.push_back(bpp.first.pt);
if (bpp.first.pt == bpp.second.pt) {
  if (DEBUG) cout << "bpp.first.pt and bpp.second.pt are equal for \*r. \n"
  v.push_back(INVALID_POINT);
}
else v.push_back(bpp.second.pt);
{
  if (DEBUG)
    cout << "No intersection points found. \n"
    << "Returning empty vector<Point> \n"
    << flush;
  v.clear();
  return v;
}
bool_real br[4];
  if (v[0] != INVALID_POINT) br[0] = v[0].is_on_segment(v[2], v[3]);
  else {
    br[0].first = false;
    br[0].second = 0;
  }
  if (v[1] != INVALID_POINT) br[1] = v[1].is_on_segment(v[2], v[3]);
  else {
    br[1].first = false;
    br[1].second = 0;
  }
  if (!(v[0] == INVALID_POINT || v[1] == INVALID_POINT)) {
    if (v[2] != INVALID_POINT) br[2] = v[2].is_on_segment(v[0], v[1]);
```
else {
    br[2].first = false;
    br[2].second = 0;
}
if (v[3] != INVALID_POINT) br[3] = v[3], is_on_segment(v[0], v[1]);
else {
    br[3].first = false;
    br[3].second = 0;
}
if (br[0].first && br[1].first) {
    if (DEBUG) cout << "The intersection of this with the line is within the triangle."
    "the intersection of this with the line."
;
}
else if (br[2].first && br[3].first) {
    if (DEBUG) cout << "The intersection of this with the line is within the triangle."
    "the intersection of this with the line."
;
}
else if (br[0].first || br[1].first || br[2].first || br[3].first) {
    if (DEBUG) cout << "The intersections of this and the line overlap partially."
    "The intersections of this and the line don’t overlap at all."
;
}
return v;
/* else. End of non-parallel, non-coplanar case. */

1044. Transformations.

Log

[LDF 2002.08.07.] Copied the entire “Transformations” section from ellipses.web and made the appropriate changes.

[LDF 2003.04.27.] The previous comment was out-of-date. I may have removed the transformation functions. At any rate, there were only a couple here. I have now copied the rest of them from ellipses.web and made the appropriate changes.

[LDF 2003.07.18.] Moved “Transformations” section, including operators==(const Transform &) from Reg_Polygon to Polygon. Also removed the Rectangle versions in rectangle.web. The Polygon versions are now inherited by Reg_Polygon and Rectangle.

1045. Applying a transformation.

Log

[LDF 2002.11.06.] Now calling Path::operator*() instead of looping through points. This way, if I change Path::operator*(), the change will automatically be reflected here.

(Declare Polygon functions 1022 ) +≡

  virtual Transform operator*==(const Transform &t);
1046.  \(\text{Define Polygon functions 1023}) \equiv\)
   Transform Polygon::operator+=(const Transform &t)
   
   Path::operator+=\(t\);
   return (center += t);

1047.  Rotation around the main axes.
   \(\text{Declare Polygon functions 1022}) \equiv\)
   virtual Transform rotate(const real x, const real y, const real z = 0);

1048.  \(\text{Define Polygon functions 1023}) \equiv\)
   Transform Polygon::rotate(const real x, const real y, const real z)
   
   Transform t;
   t.rotate\(x, y, z\);
   return (*this += t);

1049.  Rotation around an arbitrary axis.

1050.  Point arguments,

\[\text{Log}\]

[LDF 2003.06.02.] Changed name of this function from rotate\_around() to rotate(). This function now overloads rotate() with three real arguments.

\[\text{Log}\]

\[\text{virtual Transform rotate(const Point &p0, const Point &p1, const real angle = 180);}\]

1051.  \(\text{Define Polygon functions 1023}) \equiv\)
   Transform Polygon::rotate(const Point &p0, const Point &p1, const real angle)
   
   Transform t;
   t.rotate\(p0, p1, angle\);
   return (*this += t);

1052.  Path argument,

\[\text{Log}\]

[LDF 2003.06.02.] Changed name of this function from rotate\_around() to rotate(). This function now overloads rotate() with three real arguments.

\[\text{virtual Transform rotate(const Path &p, const real angle = 180);}\]
1053.  
(Define Polygon functions 1023) +≡

Transform Polygon::rotate(const Path &P, const real angle)
{
    if (!p.is_linear())
        cerr << "ERROR:In Ellipse: rotate(Path, real)\n        Path intersecting line Returning INVALID_TRANSFORM;\n    return INVALID_TRANSFORM;
    return rotate(p.get_point(0), p.get_last_point(), angle);
}

1054.  Scale.
(Declare Polygon functions 1022) +≡

virtual Transform scale(real x, real y = 1, real z = 1);

1055.  (Define Polygon functions 1023) +≡

Transform Polygon::scale(real x, real y, real z)
{
    Transform t;
    t.scale(x, y, z);
    return (*this *= t);
}

1056.  Shear.
(Declare Polygon functions 1022) +≡

virtual Transform shear(real xy, real xz = 0, real yx = 0, real yz = 0, real zx = 0, real zy = 0);

1057.  (Define Polygon functions 1023) +≡

Transform Polygon::shear(real xy, real xz, real yx, real yz, real zx, real zy)
{
    Transform t;
    t.shear(xy, xz, yx, yz, zx, zy);
    return (*this *= t);
}

1058.  Shift.

1059.  real arguments.
(Declare Polygon functions 1022) +≡

virtual Transform shift(real x, real y = 0, real z = 0);
1060. 
(Define Polygon functions 1023) +≡
  Transform Polygon :: shift(real x, real y, real z)
  {
    Transform t;
    t.shift(x, y, z);
    return (*this == t);
  }

1061. Point argument.
(Declare Polygon functions 1022) +≡
  virtual Transform shift(const Point &p);

1062. 
(Define Polygon functions 1023) +≡
  Transform Polygon :: shift(const Point &p)
  {
    return shift(p.get_x(), p.get_y(), p.get_z());
  }

1063. Shift times.

1064. real arguments.
(Declare Polygon functions 1022) +≡
  virtual void shift_times(real x, real y = 1, real z = 1);

1065. 
(Define Polygon functions 1023) +≡
  void Polygon::shift_times(real x, real y, real z)
  {
    Path::shift_times(x, y, z);
    center.shift_times(x, y, z);
    return;
  }

1066. Point argument.
(Declare Polygon functions 1022) +≡
  virtual void shift_times(const Point &p);
1067.  \(\text{Define Polygon functions 1023} + \equiv\)
   void Polygon::shift_times(const Point &p)
   {
      return shift_times(p.get_x(), p.get_y(), p.get_z());
   }

1068.  Reg_Polygon (polygons.web).  [LDF 2003.04.15.] TO DO: It will be necessary to
   supply Reg_Polygon with a complete set of transformation functions, so that \textit{center} will be transformed
   along with the \textit{Points} pointed to by the pointers on \textit{points}. Some are present already, but not all.
   [LDF 2003.04.15.] TO DO: Add \texttt{in_circle()}, \texttt{out_circle()}. Align a line from \textit{center}
   in the direction of a normal with the \textit{y-axis}. Use the inverse of the \textbf{Transform} to transform the Circle.

   format Reg_Polygon Polygon

1069.  \textbf{Reg_Polygon class definition.}  \textbf{Reg_Polygon} is derived from \textbf{Polygon},
   \begin{verbatim}
   Log
   [LDF 2003.04.15.] Changed, so that \textbf{Reg_Polygon} is derived from \textbf{Path}. Previously, it was derived from \textbf{Reg_CL_Plane_Curve}.
   [LDF 2003.04.27.] Changed \texttt{protected} data members to \texttt{private}. They no longer need to be \texttt{protected},
   because \textbf{Rectangle} is no longer derived from \textbf{Reg_Polygon}.
   [LDF 2003.06.06.] Changed, so that \textbf{Reg_Polygon} is derived from \textbf{Polygon}, which I’ve just added above.
   \end{verbatim}

   \begin{verbatim}
   (Define class Reg_Polygon 1069) \equiv
   class Reg_Polygon : public Polygon {
      real internal_angle;
      real radius;
      unsigned short sides;
      bool on_free_store;

      public:  (Declare Reg_Polygon functions 1070)
   };
   \end{verbatim}
   \texttt{This code is used in sections 1097 and 1098.}

1070.  Assignment,
   \begin{verbatim}
   Log
   [LDF 2002.12.18.] Moved here. With the DEC compiler under Compaq Tru64 on the DEC Alpha computer,
   it worked to have the assignment operators following the constructors. With the GNU C++ compiler (GCC)
   under GNU/Linux on the Intel i686 computer, it didn’t. See \texttt{Path::operator=} (in paths.web) for more
   information.
   \end{verbatim}

   \begin{verbatim}
   (Declare Reg_Polygon functions 1070) \equiv
   const Reg_Polygon &operator=(const Reg_Polygon &p);
   \end{verbatim}
   \texttt{See also sections 1073, 1076, 1079, 1087, 1089, 1091, 1092, 1093, 1095, and 1096.}
   \texttt{This code is used in section 1069.}
1071.  
(Define Reg_Polygon functions 1071) \( \equiv \)
  
  const Reg_Polygon &Reg_Polygon::operator=(const Reg_Polygon &p)  
  {  
    clear();  
    Path::operator=(p);  
    internalAngle = p.internalAngle;  
    radius = p.radius;  
    sides = p.sides;  
    center = p.center;  
    return *this;  
  }

See also sections 1074, 1077, 1078, 1080, 1081, 1307, 1309, 1310, 1311, 1313, and 1314.
This code is used in sections 1097 and 1315.

1072.  Constructors and setting functions.

1073.  Default constructor.  No arguments.
(Declare Reg_Polygon functions 1070) \( + \equiv \)
  
  Reg_Polygon();

1074.  
(Define Reg_Polygon functions 1071) \( + \equiv \)
  
  Reg_Polygon::Reg_Polygon()  
  {  
    on_free_store = false;  
    line_switch = false;  
    cycle_switch = true;  
    projective_extremes.resize(6,0);  
    /* LDF 2003.04.09. Added this line. */
  }

1075.  Center, sides, diameter, and angles.

1076.  Constructor.
(Declare Reg_Polygon functions 1070) \( + \equiv \)
  
  Reg_Polygon(const Point &center, const unsigned short ssides, const real ddiameter, const  
  real angle\_{x} = 0, const real angle\_{y} = 0, const real angle\_{z} = 0);
1077. Log

[LD 2003.08.27] Reversed the order of the initializations following "\", because GCC with the "--Wall" option issued the following warning:
"Reg_Polygon:: sides' will be initialized after 'real Reg_Polygon:: radius'".

(Define Reg_Polygon functions 1071) ++
Reg_Polygon:: Reg_Polygon(const Point &center,const unsigned short ssides,const real
   d diameter, const real angle_x,const real angle_y,const real angle_z): radius(d diameter/2),
   sides(ssides) { bool DEBUG = false; /* true */
   if (DEBUG) cout << "Entering Reg_Polygon:: Reg_Polygon()" <<
   "(center, ssides, d diameter, angles).\n" << flush;
   center = center, on_free_store = false;
   internal_angle = 360.0/sides;
   cycle_switch = true;
   projective_extremes.resize(6,0); /* LD 2003.04.09. Added this line. */
   center.apply_transform();
}

1078. For regular polygons with an even number of sides, we rotate them so that a flat side is at the "top"
(in the direction of the positive z-axis, if angle_x, angle_y, and angle_z are all 0).
(Define Reg_Polygon functions 1071) ++
for (int i = 0; i < ssides; i++) { Point *vertex = create_new < Point > (0);
   vertex->set(0,0, radius);
   if (sides % 2 == 0) vertex->rotate(0, internal_angle /2, 0);
   if (i > 0) /* [LD 2002.11.06] Only rotate if the angle \neq 0, i.e., don't rotate the first time. */
      vertex->rotate(0, i * internal_angle, 0);
   if (angle_x \neq 0 || angle_y \neq 0 || angle_z \neq 0)
      /* Rotation around the x-axis, y-axis, and z-axis, if applicable. */
      vertex->rotate(angle_x, angle_y, angle_z);
   vertex->shift(center); /* Put in position around center. */
   points.push_back (vertex); }
if (DEBUG)
   cout << "Exiting Reg_Polygon:: Reg_Polygon()" << "(center, ssides, d diameter, angles).\n" << flush;
   return;
}

1079. Setting function.
(Declare Reg_Polygon functions 1070) ++
void set(const Point &ccenter,const unsigned short ssides,const real d diameter,const real
   angle_x = 0, const real angle_y = 0, const real angle_z = 0);

(Define Reg_Polygon functions 1071) ++
void Reg_Polygon:: set(const Point &ccenter,const unsigned short ssides,const real
   d diameter, const real angle_x, const real angle_y, const real angle_z) { bool DEBUG = false;
   /* true */
   if (DEBUG)
      cout << "Entering Reg_Polygon:: set()" << "(center, ssides, d diameter, angles).\n" << flush;
1081. ?? [LDF 2002.10.07] At exactly this place, Path::Path() (the default version with no arguments) is invoked. When set() exits, ~Path() is called on the empty Path. When DEBUG == true, the following message is printed before ~Path() is entered. I don’t know why Path() is invoked and this bothers me a bit. However, it’s destroyed cleanly, so I don’t have to worry about leakage.

(Define Reg_Polygon functions 1071) +≡

Reg_Polygon p(center, ssides, ddiameter, angle_x, angle_y, angle_z);
  *this = p;
  if (DEBUG)
    cout << "Exiting Reg_Polygon::set()" << "(center, ssides, ddiameter, angles).\n" << flush;
  return;

1082. Pseudo-constructor for dynamic allocation.

1083. Pointer argument.

Log

[LDF 2002.11.06] Added optional Reg_Polygon pointer argument, Made non-inline.

(Declare non-member template functions for Reg_Polygon 1083) +≡

Reg_Polygon *create_new(const Reg_Polygon *&r);

See also section 1084.
This code is used in sections 1097 and 1098.

1084. Reference argument.

Log

[LDF 2002.11.06] Added this function.

(Declare non-member template functions for Reg_Polygon 1083) +≡

Reg_Polygon *create_new(const Reg_Polygon &r);

1085. Destructor. [LDF 2002.10.09] Removed the destructor. Path::~Path() or Path::clear() should be used instead, unless I add dynamically allocated data members to Reg_Polygon (rather than Path).

1086. Returning elements and information.

Log

[LDF 2002.11.03] Removed Reg_Polygon::is_planar(). A Reg_Polygon can be manipulated into a non-planar state, so it’s safer to use the Path version, which tests whether it’s really planar or not.

1087. Get radius.

Log

[LDF 2003.06.13] Added this function.
1088. **Circles.** [LDF 2003.06.13] The functions in this section are all defined in circles.web, because Circle is an incomplete type in this file.

1089. **Enclosed circle.**

[LDF 2003.06.13] Added this function.

1090. **Draw enclosed circle.**

1091. **Normal version.**

[LDF 2003.07.04] Removed default argument for picture. Having one made it impossible for the compiler to resolve calls to draw_in_circle() with no arguments.

1092. **Picture argument first.**

1093. **Surrounding circle.**

1094. **Draw surrounding circle.**

1095. **Normal version.**
1096. Picture argument first.

[LDF 2003.07.01] Removed default argument for `picture`. Having one made it impossible for the compiler to resolve calls to `draw_out_circle()` with no arguments.

\[\text{Declare Reg-Polygon functions 1070}\] 
\[
\text{Circle draw_out_circle(Picture &picture, const Color &draw_color = *Colors::default_color, const string dashed = "", const string pen = "") const;}
\]

1097. Putting polygons together. This is what’s compiled.

\{Include files 6\} 
\{Version control identifier 5\} 
\{Define class Polygon 1019\} 
\{Define class Reg-Polygon 1069\} 
\{Define Reg-Polygon functions 1071\} 
\{DefinePolygon functions 1023\} 
\{Declare non-member template functions for Reg-Polygon 1083\}
1098. This is what’s written to polygons.h.

(polygons.h 1098) ≡
(Define class Polygon 1019)
(Define class Reg_Polygon 1069)
(Declare non-member template functions for Reg_Polygon 1083)

1099. Rectangle (rectangles.web).

Log
[LD 2003.07.18.] Removed the “Transformations” section, including operator=(const Transform &). Also moved the Reg_Polygon versions to Polygon in polygons.web. These are now inherited by Rectangle.
[LD 2003.11.12.] Removed the version control identifiers from the CWEB files for the distribution of 3DLDF 1.1. They’re still used in my development versions.
[LD 2003.12.10.] Put the version control identifiers back into my release versions for 3DLDF 1.1.4. I’ve already put some of them back in, now I’m doing the rest of them. However, the release versions are now in their own RCS repository.

format Rectangle Reg_Polygon
(Version control identifier 5) ≡
static string res_id = "$Id: rectangles.web,v,1.5,2004/01/12,21:32:24,11finsstoi1,Exp$";

1100. Include files.

(Includes files 6) ≡
#include "loader.h"
#include "pspglb.h"
#include "createnew.h"
#include "io.h"
#include "colors.h"
#include "transform.h"
#include "shapes.h"
#include "pictures.h"
#include "points.h"
#include "lines.h"
#include "planes.h"
#include "paths.h"
#include "curves.h"
#include "polygons.h"

1101. Rectangle class definition. [LD 2003.07.18.] TO DO: axis_h and axis_v are not recalculated when a Rectangle is transformed. I should do something about this.

(Define class Rectangle 1101) ≡
class Rectangle : public Polygon {
    real axis_h;
    real axis_v;
    bool on_free_store;

public: { Declare Rectangle functions 1103 }
};

This code is used in sections 1139 and 1140.

1102. Constructors and setting functions.
1103. **Default constructor.** No arguments.

(Declare Rectangle functions 1103) \equiv

    Rectangle();

See also sections 1106, 1108, 1111, 1113, 1119, 1122, 1125, 1127, 1130, 1132, 1135, 1136, 1137, and 1138.

This code is used in section 1101.

1104.

(Define Rectangle functions 1104) \equiv

    Rectangle::Rectangle()
    {
        on_free_store = false;
        line_switch = false;
        cycle_switch = true;
    }

See also sections 1107, 1109, 1112, 1114, 1120, 1123, 1126, 1128, 1131, 1133, 1267, 1268, 1269, and 1270.

This code is used in sections 1139 and 1271.

1105. **Center, lengths, and angles.** [LDF 2002.11.06] The following constructor and setting function create the Rectangle in the x-z plane and then rotate according to the arguments angle_x, angle_y, and angle_z, if at least one of them is non-zero.

1106. **Constructor.**

<table>
<thead>
<tr>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>[LDF 2002.11.06] Made real arguments const.</td>
</tr>
<tr>
<td>[LDF 2003.07.18] BUG FIX: Now, axis_h and axis_v are no longer divided by 2, when I initialize axis_h half and axis_v half. I mistakenly used / = instead of /.</td>
</tr>
</tbody>
</table>

(Declare Rectangle functions 1103) \equiv

    Rectangle(const Point &center, const real axis_h, const real axis_v, const real
        angle_x = 0, const real angle_y = 0, const real angle_z = 0);
1107.  
(Define Rectangle functions 1104) +≡

Rectangle::Rectangle(const Point &ccenter, const real axis_h, const real axis_v, const
real angle_x, const real angle_y, const real angle_z): axis_h(axis_h), axis_v(axis_v) {
    on_free_store = false;
    line_switch = false;
    cycle_switch = true;
    center = ccenter;
    center::apply_transform();

    real axis_h_half = axis_h/2;
    real axis_v_half = axis_v/2;
    Point bottom_left(-axis_h_half, 0, -axis_v_half);
    Point bottom_right(axis_h_half, 0, -axis_v_half);
    Point top_left(-axis_h_half, 0, axis_v_half);
    Point top_right(axis_h_half, 0, axis_v_half);

    if (angle_x ≠ 0 ∨ angle_y ≠ 0 ∨ angle_z ≠ 0)  /* Rotation around the x-axis, y-axis, and z-axis. */
    {
        bottom_left.rotate(angle_x, angle_y, angle_z);
        bottom_right.rotate(angle_x, angle_y, angle_z);
        top_left.rotate(angle_x, angle_y, angle_z);
        top_right.rotate(angle_x, angle_y, angle_z);
    }  /* Put around center. */
    bottom_left.shift(center);
    bottom_right.shift(center);
    top_left.shift(center);
    top_right.shift(center);
    top_left.shift(center);
    for (int i = 0; i < 4; i++) points.push_back ( create_new < Point > (0) ) ;
    *points[0] = bottom_left;
    *points[1] = bottom_right;
    *points[2] = top_right;
    *points[3] = top_left;
}

1108. Setting function.

[LD 2002.11.06.] Made real arguments const.

(Declare Rectangle functions 1103) +≡

void set(const Point &ccenter, const real axis_h, const real axis_v, const real angle_x = 0, const
real angle_y = 0, const real angle_z = 0);

1109.  
(Define Rectangle functions 1104) +≡

void Rectangle::set(const Point &ccenter, const real axis_h, const real axis_v, const real
angle_x, const real angle_y, const real angle_z)
{
    Rectangle r(ccenter, axis_h, axis_v, angle_x, angle_y, angle_z);
    *this = r;
}

1110. Four Points. The Point arguments must be so ordered, that they are contiguous in the resulting
Rectangle.
1111. Constructor.
(Declare Rectangle functions 1103) +≡
Rectangle(const Point &pt0, const Point &pt1, const Point &pt2, const Point &pt3);

1112.
(Define Rectangle functions 1104) +≡
Rectangle::Rectangle(const Point &pt0, const Point &pt1, const Point &pt2, const Point &pt3)
{ Point pt5 = pt1 - pt0;        /* Check that the Point arguments are coplanar. */
  Point pt6 = pt3 - pt0;
  Point pt7 = pt4.cross_product(pt5);
  pt7.unit_vector(true);
  Point pt8 = pt4.cross_product(pt6);
  pt8.unit_vector(true); if (pt7 ≡ pt8 ∨ pt7 ≡ -pt8) /* If they are, create a Rectangle. */
  { on_free_store = false;
    line_switch = false;
    cycle_switch = true;
    center = pt0.mediate(pt2);
    axis_x = (pt1 - pt0).magnitude();
    axis_y = (pt2 - pt1).magnitude();
    points.push_back ( create_new < Point > (pt0) ) ;
    points.push_back ( create_new < Point > (pt1) ) ;
    points.push_back ( create_new < Point > (pt2) ) ;
    connectors.push_back("--");
  } else {
    cerr << "ERROR! In Rectangle() with four Point arguments.\n" <<
         "Points are not coplanar.\nReturning.\n" << flush;
  }
  return;
}

1113. Setting function.
(Declare Rectangle functions 1103) +≡
void set(const Point &pt0, const Point &pt1, const Point &pt2, const Point &pt3);
1114.  
(Define \texttt{Rectangle} functions 1104) +\equiv 

\begin{verbatim}
void Rectangle::set(const Point &pt0, const Point &pt1, const Point &pt2, const Point &pt3) 
{
    Rectangle r(pt0, pt1, pt2, pt3);
    *this = r;
}
\end{verbatim}

1115.  Pseudo-constructor for dynamic allocation.

1116.  Pointer argument.

\begin{verbatim}
[LD 2003.12.30.] Replaced \texttt{Rectangle::create\_new\_rectangle()} with a specialization of \texttt{template<\texttt{class C}> C*create\_new(\texttt{}) for \texttt{Rectangle}}.
\end{verbatim}

1117.  Reference argument.

\begin{verbatim}
[LD 2003.12.30.] Replaced \texttt{Rectangle::create\_new\_rectangle()} with a specialization of \texttt{template<\texttt{class C}> C*create\_new(\texttt{}) for \texttt{Rectangle}}.
\end{verbatim}

1118.  Destructor.  [LD 2002.10.09.] Removed the destructor, \texttt{Path::~Path()} or \texttt{Path::clear()} should be used instead, unless I add dynamically allocated data members to \texttt{Rectangle} (rather than \texttt{Reg\_Polygon} or \texttt{Path}).

1119.  Assignment.

\begin{verbatim}
[LD 2002.11.06.] Changed return value from \texttt{void} to \texttt{const Rectangle \&}.
\end{verbatim}

(Declare \texttt{Rectangle} functions 1103) +\equiv 

\begin{verbatim}
const Rectangle \&operator=(\texttt{const Rectangle \&c});
\end{verbatim}
1120.  !! Remember to put anything specific to Rectangles in here!
(Define Rectangle functions 1104) +⇒
    const Rectangle &Rectangle :: operator=(const Rectangle &c)
    {
      clear();
      Path::operator=(c);
      center = c.center;
      axis1 = c.axis1;
      axis2 = c.axis2;
      return *this;
    }

1121.  Returning Elements and information.

[Log]
[LDL 2003.04.15.] Added this section. It’s become necessary, since I’m deriving Rectangle from Path
now, and not from Reg_Polygon.

1122.  Is rectangular.  is_rectangular() tests whether a Rectangle is rectangular. It first tests if the
Rectangle is planar. Then it creates vectors from the points on the Rectangle, and checks their angles to
one another. If they are within Point::epsilon() (exclusive) of 180° in one case, and 90° in the other two,
is_rectangular() returns 1, otherwise 0. [LDL 2003.12.02]

[Log]
[LDL 2003.11.28.] Added this function.
[LDL 2003.12.09.] Now using cross_product() to test for parallelity of the sides. TO DO: Add Path::is_parallel() and a version for Points.

(Declare Rectangle functions 1103) +⇒
    bool is_rectangular() const;
1123. Define Rectangle functions 1104 \( \equiv \)

```cpp
bool Rectangle::is_rectangular() const
{
  if (~is_planar()) return false;
  Point a = (get_point(1) - get_point(0));
  Point b = (get_point(2) - get_point(3));
  Point c = (get_point(3) - get_point(0));
  Point d = (get_point(2) - get_point(1));
  return (a.cross_product(b) \equiv \text{origin} \land c.cross_product(d) \equiv \text{origin} \land \text{fabs}(a.\text{angle}(d) - 90) < \text{Point::epsilon()});
}
```

1124. Returning Points.

Log

- [LDF 2002.11.06.]: Got rid of `get_center()`. It’s not needed, since `Reg_Polygon::get_center()` does the trick.
- [LDF 2003.04.15.]: Added `get_center()` again, since I’m no longer deriving `Rectangle` from `Reg_Polygon`, but from `Path`.
- [LDF 2003.07.18.]: Got rid of `get_center()` again, because `Rectangle` is now derived from `Polygon`, and I’ve moved the `Reg_Polygon` versions to `Polygon`.

1125. Corner. The argument \( c \) should be in the range \( 0 \leq c \leq 3 \).

```cpp
(Declare Rectangle functions 1103) \( \equiv \)

Point corner(unsigned short c);
```

1126. Define Rectangle functions 1104 \( \equiv \)

```cpp
Point Rectangle::corner(unsigned short c)
{
  if (c > 3) {
    cerr \leftarrow "ERROR:\u007eRectangles\u007e have\u007e4\u007ecorners,\u007eu"
     \leftarrow "numbered\u007e0\u007ethrough\u007e3.:\nReturning\u007eINVALID\_POINT.\n" \leftarrow flush;
    return INVALID\_POINT;
  }
  return *points[c];
}
```

1127. Get Mid-point. The argument \( c \) should be in the range \( 0 \leq c \leq 3 \).

Log

- [LDF 2002.11.06.]: Changed this function so that it uses `mediate()`.
- [LDF 2003.06.09.]: Renamed this function `get_mid_point()`. Formerly, it was called `mid_point()`.
- [LDF 2003.07.18.]: Made `const`.

```cpp
(Declare Rectangle functions 1103) \( \equiv \)

Point get_mid_point(unsigned short c) const;
```
1128.  
(Define Rectangle functions 1104) +≡
Point Rectangle::get_mid_point(unsigned short c) const
{
    if (c > 3) {
        cerr << "ERROR: Rectangles have 4 or more mid_points, " <<
            "numbered 0 through 3 \nReturning INVALID_POINT.\n" << flush;
        return INVALID_POINT;
    }
    Point p0;  
    Point p1;  
    Point p2;
    p0 = *points[c];
    p1 = (c < 3) ? *points[c + 1] : *points[0];
    return p0.mediate(p1);
}

1129. Getting axes.  [LDF 2003.07.18.] TO DO: axis_h and axis_v are not recalculated when a
Rectangle is transformed. I should do something about this.

Log

[LDF 2003.07.18.] Added this section.

1130. Get axis_h.
(Declare Rectangle functions 1103) +≡
real get_axis_h() const;

1131.  
(Define Rectangle functions 1104) +≡
real Rectangle::get_axis_h() const
{
    return axis_h;
}

1132. Get axis_v.
(Declare Rectangle functions 1103) +≡
real get_axis_v() const;
1133.
(Define Rectangle functions 1104) +≡
  real Rectangle :: get_axis_v() const
  { return axis_v;
  }

1134. Ellipses.

[LDF 2003.07.18.] Added this section. These functions must be defined in ellipses.web, because Ellipse
is an incomplete type in this file.

1135. Surrounding Ellipse.

[LDF 2003.07.18.] Added this function.

(Declare Rectangle functions 1103) ÷≡
  Ellipse out_ellipse() const;

1136. Enclosed Ellipse.

[LDF 2003.07.18.] Added this function.

(Declare Rectangle functions 1103) ÷≡
  Ellipse in_ellipse() const;

1137. Draw surrounding Ellipse.

[LDF 2003.07.18.] Added this function.

(Declare Rectangle functions 1103) ÷≡
  Ellipse draw_out_ellipse(const Color &draw_color = +Colors::default_color, string
  ddashed = "", string ppen = "", Picture &picture = current_picture) const;

1138. Draw enclosed Ellipse.

[LDF 2003.07.18.] Added this function.

(Declare Rectangle functions 1103) ÷≡
  Ellipse draw_in_ellipse(const Color &draw_color = +Colors::default_color, string
  ddashed = "", string ppen = "", Picture &picture = current_picture) const;

1139. Putting Rectangle together. This is what’s compiled.

(Include files 6)
(Version control identifier 5)
(Define class Rectangle 1101)
(Define Rectangle functions 1104)
(Declare non-member template functions for Rectangle 1116)
1140. This is what's written to rectangs.h.

(rectangs.h 1140) ⊆

{Define class Rectangle 1101}

{Declare non-member template functions for Rectangle 1116}

1141. Ellipse (ellipses.web).

[Log]

[LDF 2003.11.12] Removed the version control identifiers from the CWEB files for the distribution of 3DLDF 1.1. They're still used in my development versions.

[LDF 2003.12.10] Put the version control identifiers back into my release versions for 3DLDF 1.1.4. I've already put some of them back in, now I'm doing the rest of them. However, the release versions are now in their own RCS repository.

format Ellipse Path

{Version control identifier 5} ⊆

static string res_id = "$Id:\ellipses.web,v,1.7,2004/01/14,17:56:19,1finscio1,Exp,\$";

1142. Include files.

{Include files 6} ⊆

#include "loader.h"
#include "pspglb.h"
#include "createnew.h"
#include "io.h"
#include "colors.h"
#include "transfor.h"
#include "shapes.h"
#include "pictures.h"
#include "points.h"
#include "lines.h"
#include "planes.h"
#include "paths.h"
#include "curves.h"
#include "polygons.h"
#include "rectangs.h"

1143. Ellipse class definition.

[Log]

[LDF 2003.07.25] Added focus0, focus1, and linear_eccentricity.


{Define class Ellipse 1143} ⊆

class Ellipse : public Reg_CI_Plane_Curve {
protected: Point focus0;
    Point focus1;
    real linear_eccentricity;
    real numericalEccentricity;
    real axisJ;
    real axis_V;
    static unsigned short DEFAULT_NUMBER_OF_POINTS;
public: (Declare Ellipse functions 1146)
);
This code is used in sections 1271 and 1272.

1144. **Static data members.**
(Define static Ellipse data members 1144) ≡
    unsigned short Ellipse::DEFAULT_NUMBER_OF_POINTS = 16;
    /* [LDF 2002.11.06.] Must be a multiple of 4. */
This code is used in section 1271.

1145. **Constructors.**

1146. **Default constructor.** No arguments.
(Declare Ellipse functions 1146) ≡
    Ellipse();
See also sections 1149, 1151, 1157, 1160, 1162, 1164, 1166, 1167, 1169, 1171, 1174, 1177, 1179, 1182, 1184, 1186, 1188, 1192, 1194, 1197, 1199, 1201, 1203, 1205, 1208, 1210, 1214, 1231, 1233, 1235, 1237, 1239, 1242, 1244, 1247, 1249, 1252, 1254, 1257, 1259, 1261, and 1263.
This code is used in section 1143.

1147.
(Define Ellipse functions 1147) ≡
    Ellipse::Ellipse()
    {
        on_free_store = false;
        line_switch = false;
        cycle_switch = true;
    }
See also sections 1150, 1152, 1158, 1161, 1165, 1168, 1170, 1172, 1173, 1175, 1178, 1180, 1183, 1185, 1187, 1189, 1193, 1195, 1198, 1200, 1202, 1204, 1206, 1209, 1211, 1215, 1216, 1217, 1218, 1219, 1220, 1221, 1222, 1223, 1224, 1226, 1227, 1232, 1234, 1236, 1238, 1240, 1243, 1245, 1248, 1250, 1253, 1255, 1258, 1260, 1262, and 1264.
This code is used in section 1271.

1148. **Center, lengths, and angles of rotation.**

1149. **Constructor.** The ellipse is always generated in the x-z plane with the center at the origin. Then it is rotated about the main axes according to the values of the angle arguments and shifted to center.

   Log
   [LDF 2002.11.06.] Made real arguments const.
   [LDF 2003.07.25.] Added code for setting focus0, focus1, and linear_eccentricity.
   [LDF 2003.07.27.] Added code for setting numerical_eccentricity.

(Declare Ellipse functions 1146) +≡
    Ellipse(const Point &center, const real aaxis_h, const real aaxis_v, const real
    angle_x = 0, const real angle_y = 0, const real angle_z = 0, const unsigned short
    number_of_points = DEFAULT_NUMBER_OF_POINTS);
1150.
(Define Ellipse functions 1147) +

Ellipse::Ellipse(const Point & center, const real axis_h, const real axis_v, const real angle_x, const real angle_y, const real angle_z, const unsigned short number_of_points):
  axis_h(axis_h), axis_v(axis_v) { center = center;
  center.apply_transform();
  focus0 = origin;
  focus1 = origin;
  real axis_h_half = axis_h/2;
  real axis_v_half = axis_v/2;
  if (axis_h >= axis_v) {
    linear_eccentricity = sqrt((axis_h_half * axis_h_half) - (axis_v_half * axis_v_half));
    focus0.shift(-linear_eccentricity);
    focus1.shift(linear_eccentricity);
    numericalEccentricity = linear_eccentricity / axis_h_half;
  } else {
    linear_eccentricity = sqrt((axis_v_half * axis_v_half) - (axis_h_half * axis_h_half));
    focus0.shift(0, 0, -linear_eccentricity);
    focus1.shift(0, 0, linear_eccentricity);
    numericalEccentricity = linear_eccentricity / axis_v_half;
  }
  if (number_of_points % 4 != 0) {
    cerr << "WARNING! In Ellipse(): invalid value for number_of_points: " << number_of_points << ", using default instead: " <<
    DEFAULT_NUMBER_OF_POINTS << endl << "Using default instead: " <<
    number_of_points = DEFAULT_NUMBER_OF_POINTS;
  } else number_of_points = number_of_points;
  on_free_store = false;
  line_switch = false;
  cycle_switch = true;
  connectors.push_back ("...");

Transform t;
  if (angle_x != 0 || angle_y != 0 || angle_z != 0) t.rotate(angle_x, angle_y, angle_z);
  t.shift(center);
  focus0 += focus1 *= t;
real curr_angle;
real curr_x;
real curr_z;
  for (int i = 0; i < number_of_points; i++)
    /* LDF 2002.11.06. Modified this code. */
    /* curr_angle = 2*i*pi/number_of_points; */
    curr_x = axis_h/2 * cos(curr_angle);
    curr_z = axis_v/2 * sin(curr_angle);
    points.push_back ( create_new < Point > (0) );
  points.back()->set(curr_x, curr_z);
  *points.back() *= t;
    /* Rotate *points.back() around the x, y, and z-axes and shift it to center. */}
\textbf{1151. Setting function.}

\begin{center}
\begin{tabular}{ll}
\textbf{Log} & \\
\end{tabular}
\end{center}

[LDF 2003.03.01] Added this function.

\begin{verbatim}
\{Declare Ellipse functions 1146\} \equiv
void set(const Point &ccenter, const real axis_h, const real axis_v, const real
angle_x = 0, const real angle_y = 0, const real angle_z = 0, const unsigned short
nnumber_of_points = DEFAULT_NUMBER_OF_POINTS);
\end{verbatim}

\textbf{1152.}

\begin{verbatim}
\{Define Ellipse functions 1147\} \equiv
void Ellipse::set(const Point &ccenter, const real axis_h, const real axis_v, const real
angle_x, const real angle_y, const real angle_z, const unsigned short nnumber_of_points)
{  
    Ellipse e(ccenter, axis_h, axis_v, angle_x, angle_y, angle_z, nnumber_of_points);
    *this = e;
}
\end{verbatim}

\textbf{1153. Pseudo-constructor for dynamic allocation.}

\textbf{1154. Pointer argument.}

\begin{center}
\begin{tabular}{ll}
\textbf{Log} & \\
\end{tabular}
\end{center}

[LDF 2002.11.06] Added optional \texttt{const Ellipse *} argument.

[LDF 2003.12.30] Replaced \texttt{Ellipse::create_new()} with a specialization of \texttt{template(class C) C*create_new()} for \texttt{Ellipse}.

\begin{verbatim}
\{Declare non-member template functions for Ellipse 1154\} \equiv
    Ellipse *create_new(const Ellipse *e);
\end{verbatim}

See also section 1155.

This code is used in sections 1271 and 1272.
### 1155. Reference argument.

<table>
<thead>
<tr>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Added this function.</td>
</tr>
<tr>
<td>Replaced <code>Ellipse::create_new_ellipse()</code> with a specialization of <code>template&lt;class C&gt; C*create_new()</code> for <code>Ellipse</code>.</td>
</tr>
</tbody>
</table>

#### Declaration non-member template functions for `Ellipse` 1154

```cpp
Ellipse *create_new(const Ellipse &c);
```

### 1156. Destructor.

[LDF 2002.10.09] Removed the destructor. `Path::~Path()` or `Path::clear()` should be used instead, unless I add dynamically allocated data members to `Ellipse` (rather than `Path`).

### 1157. Assignment.

<table>
<thead>
<tr>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Added error handling code to prevent self-assignment.</td>
</tr>
<tr>
<td>Changed return value to <code>Ellipse &amp;</code>.</td>
</tr>
</tbody>
</table>

#### Declaration `Ellipse` functions 1146

```cpp
Ellipse &operator=(const Ellipse &c);
```

#### Definition `Ellipse` functions 1147

```cpp
Ellipse &Ellipse::operator=(const Ellipse &e)
{
  if (this == &e) /* [LDF 2002.11.06] Make sure it's not self-assignment. */
    return *this;
  Path::operator=(e);
  center = e.center; /* Ellipse members. */
  axis_h = e.axis_h;
  axis_v = e.axis_v;
  focus0 = e.focus0;
  focus1 = e.focus1;
  number_of_points = e.number_of_points; /* Reg_C1_Plane_Curve members. */
  return *this;
}
```

### 1159. Labelling.

### 1160. Label.

#### Declaration `Ellipse` functions 1146

```cpp
void label(const string pos = "top", const bool dot = false, Picture &picture = current_picture) const;
```
1161.  
(Define Ellipse functions 1147) +≡

```cpp
void Ellipse::label(const string pos, const bool dot, Picture &picture) const
{
  if (Label::D0_LABELS == false) {
    return;
  }
  string s;
  char c = 'a';
  for (vector<Point*>::const_iterator iter = points.begin(); iter != points.end(); iter++) {
    s = c;
    (**iter).label(s, pos, dot, picture);
    c++;
  }
}
```

1162. Dotlabel.

[LD 2002.11.06.] Changed this function so that it just calls Ellipse::label() with dot = true. Made it inline.

(Declare Ellipse functions 1146) +≡

```cpp
inline void dotlabel(string pos = "top", Picture &picture = current_picture) const
{
  label(pos, true, picture);
}
```

1163. Returning elements and information.

1164. Is elliptical. [LD 2003.07.20.] is_elliptical() first checks whether *this is planar by calling get_normal(). If the latter function returns INVALID_POINT, then is_elliptical() returns false. Otherwise, it makes a copy of *this called e, puts e into the x-z plane, and rotates it, so that *e.points[0] lies on the x-axis. Then, it plugs the x and z-coordinates of the Points on e into the ellipse equation, i.e., \( x^2/a^2 + z^2/b^2 = 1 \), where \( a \) is half of the horizontal axis of the ellipse, and \( b \) is half of the vertical axis. Let \( r = x^2/a^2 + z^2/b^2 \) and \( \epsilon \) stand for the return value of Point::epsilon(). If \(|r - 1| > \epsilon \) for any of the Points on e, is_elliptical() returns false, otherwise it returns true.

[LD 2003.07.20.] Added this function,
[LD 2003.07.25.] Now checking normal ≡ INVALID_POINT. If it is, is_elliptical() returns false.

(Declare Ellipse functions 1146) +≡

```cpp
bool is_elliptical() const;
```
1165.
(Define Ellipse functions 1147) +¥

```cpp
bool Ellipse::is_elliptical() const
{
    bool DEBUG = false;    /* true */
    Point normal = get_normal();
    if (normal ≡ INVALID_POINT)
    {
        if (DEBUG)
            cerr << "In Ellipse::is_elliptical():\n                    get_normal() returned INVALID_POINT.\n                    Returning false.\n            endl << endl << flush;
        return false;
    }
    Ellipse e = *this;
    normal shift(center);
    Transform t;
    t.align_with_axis(centroid, normal, 'y');
    e.do_transform(t);
    if (DEBUG) e.show("e after alignment:");
    Point x_axis_pt1();
    Point p0 = e.get_point(0);
    real ang = p0.angle(x.axis_pt);
    t.reset();
    t.rotate(0, ang);
    p0 *= e.do_transform(t);
    p0.unit_vector(true);
    if (p0 ≠ x.axis_pt) { t.reset(); t.rotate(0, -2 * ang); e.do_transform(t); }
    p0 = e.get_point(0);
    p0.unit_vector(true);
    if (p0 ≠ x.axis_pt) { if (DEBUG)
        cerr << "ERROR! In is_elliptical():\n                Rotation failed. Returning false\n            endl << flush;
    return false; }
    if (DEBUG) e.show("e after rotation in x-z plane:");
    real x;
    real z;
    real a = (e.get_point(0) - e.get_center()).magnitude();
    real b = (e.get_point(number_of_points / 4) - e.get_center()).magnitude();
    if (DEBUG) {
        cout << "a = " << a << endl << flush;
        cout << "b = " << b << endl << flush;
    }
    real r;
    int i = 0;
```
for (vector<Point *>::const_iterator iter = e.points.begin(); iter != e.points.end(); ++iter) {
    x = (**iter).get_x();
    z = (**iter).get_z();
    if (DEBUG) {
        cout << "Point_i" << i << ":_u" << endl << flush;
        (**iter).show("iter:");
        cout << "x_" == u" << x << endl << flush;
        cout << "z_" == u" << z << endl << flush;
    }
    r = ((x * x)/(a * a)) + ((z * z)/(b * b));
    if (DEBUG) cout << "r_" == u" << r << endl << flush;
    if (fabs(r - 1) > Point::epsilon()) {
        if (DEBUG) cout << "Point_i" << i << ":_u" doesn’t satisfy ellipse equation._n" << 
                     "Returning false._n\n" << flush;
            return false;
    }
    ++i;
}
if (DEBUG)
    cout << "Exiting Ellipse::is_elliptical()._u\nReturning true." << endl << endl << flush;
return true;
}

1166. Is quadratic.

(Declare Ellipse functions 1146) +≡
inline bool is_quadratic() const
{
    return true;
}

1167. Is cubic.

Log

[LDF 2003.07.27.] Made virtual and non-inline.

(Declare Ellipse functions 1146) +≡
virtual bool is_cubic() const;
1168.
(Define Ellipse functions 1147) +\equiv
    bool Ellipse::is_cubic() const
    {
        return false;
    }

1169. Is quartic.

    [LDF 2003.07.27.] Made virtual and non-inline.

(Declare Ellipse functions 1146) +\equiv
    virtual bool is_quartic() const;

1170.
(Define Ellipse functions 1147) +\equiv
    bool Ellipse::is_quartic() const
    {
        return false;
    }

1171. Solve. [LDF 2002.11.06] solve() assumes that the Ellipse lies in a major plane with its center at the origin. Code that calls it must ensure that these conditions are fulfilled. solve() returns the two possible values for either the horizontal or the vertical coordinate.

    TO DO: Read through, and then explain this function.

    [LDF 2003.07.20.] Now using get_axis_v() and get_axis_h(), instead of accessing axis_h and axis_v directly.
    [LDF 2003.07.25.] Removed some commented-out code, and an explanatory comment.

(Declare Ellipse functions 1146) +\equiv
    real_pair solve(char axis_unknown, real known) const;
1172.

(Define **Ellipse** functions 1147) +≡

```c
real_pair Ellipse::solve(char axis_unknown, real known) const {
    real radius_known;
    real radius_unknown;
    axis_unknown = tolower(axis_unknown);
    real_pair r;
    if (axis_unknown == 'h') {
        radius_known = get_axis_v() / 2;
        radius_unknown = get_axis_h() / 2;
    }
    else if (axis_unknown == 'v') {
        radius_known = get_axis_h() / 2;
        radius_unknown = get_axis_v() / 2;
    }
    else {
        cerr << "ERROR! In Ellipse::solve().\n" << "Invalid character for axis unknown: \n" << axis_unknown << "\nReturning INVALID_REAL_PAIR.\n" << flush;
        return INVALID_REAL_PAIR;
    }
    if (fabs(known) > radius_known) {
        return INVALID_REAL_PAIR;
    }
```
1173. The equation for an ellipse in the $x$-$y$ plane with its center at the origin is

$$x^2/a^2 + y^2/b^2 = 1$$

where $a$ is half the horizontal axis and $b$ is half the vertical axis. Therefore,

$$y = \sqrt{(1-x^2/a^2) \times b^2}$$

and

$$x = \sqrt{(1-y^2/b^2) \times a^2}.$$

(Define Ellipse functions 1147) +≡

```cpp
r.first = sqrt(((1 - (known*known)/(radius_known + radius_known)) * (radius_unknown + radius_unknown));
return r;
```

1174. Get coefficients. This is used for getting the coefficients of the quadratic equation that results from replacing $y$ with $mx + b$ from the line equation, where $m$ is the slope and $b$ the intercept with the vertical axis, ( whichever that might be in a particular case; it needn't be the y-axis) in the equation for the ellipse

$$x^2/a^2 + y^2/b^2 = 1$$

namely !! START HERE. Check this. I think the $x$ in the coefficient for $x^2$ is wrong.

$$x^2/a^2 + (mx + b)^2/b^2 - 1 = 0 \equiv (\beta^2 x + x^2 m^2) x^2 + 2abmx + (a^2 b^2 - a^2 \beta^2) = 0.$$  

The coefficients are returned in the `struct real_triple` in the order one would expect: `r.first` is the coefficient of $x^2$, `r.second` of $x$ and `r.third` of the constant term ($x^0$).

Log

[LDF 2003.07.20] Now using `get_axis_x()` and `get_axis_y()`, instead of accessing `axis_x` and `axis_y` directly.

[LDF 2003.07.27] Corrected a typo in the math mode material showing the coefficients.

(Declare Ellipse functions 1146) +≡

```cpp
real_triple get_coefficients(real Slope, real v_intercept) const;
```

1175.

(Define Ellipse functions 1147) +≡

```cpp
real_triple Ellipse :: get_coefficients(real Slope, real v_intercept) const
{
    real_triple r;
    real ax_j = get_axis_x();
    real ax_v = get_axis_y();
    r.first = ((ax_v/2) * (ax_v/2)) + ((ax_j/2) * (ax_j/2)) * (Slope * Slope);  /* a */
    r.second = 2 * Slope * v_intercept + ((ax_j/2) * (ax_j/2));  /* b */
    r.third = (((ax_j/2)*(ax_j/2))*(v_intercept * v_intercept)) - (((ax_v/2)*(ax_v/2))*(v_intercept * v_intercept)) - (((ax_j/2)*(ax_j/2))*(ax_j/2));  /* c */
    return r;
}
```

1176. Get center.

Log

[LDF 2002.11.10.] Made this function virtual and non-inline.
[LDF 2003.06.09.] Changed return value from Point & to const Point &.

(Declare Ellipse functions 1146) +≡

virtual const Point & get_center();

1178.

(Define Ellipse functions 1147) +≡

const Point & Ellipse::get_center()
{
    center.apply_transform();
    return center;
}

1179. const version.

Log

[LDF 2002.11.10.] Made this function virtual and non-inline.

(Declare Ellipse functions 1146) +≡

virtual Point get_center() const;

1180.

(Define Ellipse functions 1147) +≡

Point Ellipse::get_center() const
{
    Point p(center);
    p.apply_transform();
    return p;
}

1181. Get focus.

1182. Non-const version.

Log

[LDF 2003.07.25.] Added this function.

(Declare Ellipse functions 1146) +≡

const Point & get_focus(const unsigned short s);
1183.  
(Define Ellipse functions 1147) ⊆≡
  const Point &Ellipse::get_focus(const unsigned short s)
  {
    if (s ≡ 0) {
      focus0. apply_transform();
      return focus0;
    }
    else if (s ≡ 1) {
      focus1. apply_transform();
      return focus1;
    }
    else {
      cerr ≪ "ERROR! In Ellipse::get_focus():\nError: Invalid argument. s ≡ endl ≪
      "Valid arguments are 0 and 1.\nReturning INVALID_POINT.\n" ≪ flush;
      return INVALID_POINT;
    }
  }

1184.  const version.


(Declare Ellipse functions 1146) ⊆≡
  Point get_focus(const unsigned short s) const;
1185. (Define Ellipse functions 1147) +≡
   Point Ellipse::get_focus(const unsigned short s) const
   {
      Point p;
      if (s == 0) {
         p = focus0;
         p.apply_transform();
         return p;
      }
      else if (s == 1) {
         p = focus1;
         p.apply_transform();
         return p;
      }
      else {
         cerr << "ERROR! In Ellipse::get_focus():\n         Invalid argument: s \n         Valid arguments are 0 and 1.\n         Returning INVALID_POINT.\n         " << flush;
         return INVALID_POINT;
      }
   }


   (Declare Ellipse functions 1146) +≡
   real get_linear_eccentricity() const;

1187. (Define Ellipse functions 1147) +≡
   real Ellipse::get_linear_eccentricity() const
   {
      return linear_eccentricity;
   }


   (Declare Ellipse functions 1146) +≡
   real get_numerical_eccentricity() const;
1189. Define Ellipse functions +
   real Ellipse :: get_numerical_eccentricity ( ) const
   { return numerical_eccentricity; }

1190. Get axes.

[LDF 2003.07.20.] Rewrote the const versions of the functions in this section, and added non-const
versions. All of them now check whether *this is still elliptical using is_Ellipse ( ). If it is, the value axis_v
or axis_h should have is recalculated, and this value is returned. In the non-const versions, axis_v or axis_h
is set to the new value. If *this is no longer elliptical, the function returns INVALID_REAL, and axis_h or
axis_v is set to INVALID_REAL in the non-const versions.

[LDF 2003.07.20.] axis_h and axis_v are updated by the transformation functions, and these are presumably
the only ones that could cause an Ellipse to become non-elliptical. So, checking and recalculating them
here is probably redundant. However, this may change, so it’s safer to do this here.

[LDF 2003.07.25.] BUG FIX: axis_h and axis_v were too small by half. Now multiplying by 2 in all versions
of get_axis_h ( ) and get_axis_v ( ).

1191. Get vertical axis.

1192. const version.
   (Declare Ellipse functions +
   real get_axis_v ( ) const;

1193. Define Ellipse functions +
   real Ellipse :: get_axis_v ( ) const
   { if ( is_Elliptical ( ) ) return (2 * (get_point ( number_of_points / 4 ) - get_center ( ) ).magnitude ( ));
     else return INVALID_REAL; }

1194. Non-const version.
   (Declare Ellipse functions +
   real get_axis_v ( );

1195. Define Ellipse functions +
   real Ellipse :: get_axis_v ( )
   { if ( is_Elliptical ( ) ) axis_v = ((get_point ( number_of_points / 4 ) - get_center ( ) ).magnitude ( ) * 2);
     else axis_v = INVALID_REAL;
     return axis_v; }

1196. Get horizontal axis.
§197. const version.
(Declare Ellipse functions 1146) +≡
   real get_axis_h() const;

§198.
(Define Ellipse functions 1147) +≡
   real Ellipse::get_axis_h() const
   {
      if (is_elliptical()) return ((get_point(0) - get_center()).magnitude() + 2);
      else return INVALID_REAL;
   }

§199. Non-const version.
(Declare Ellipse functions 1146) +≡
   real get_axis_h();

§200.
(Define Ellipse functions 1147) +≡
   real Ellipse::get_axis_h()
   {
      if (is_elliptical()) axis_h = (get_point(0) - get_center()).magnitude() + 2;
      else axis_h = INVALID_REAL;
      return axis_h;
   }

§201. Angle point. angle_point() returns a point on the ellipse given an angle. Effectively, point[0] is rotated about the center in the plane of the ellipse and the intersection of the ray from the center through point[0] and the ellipse is returned.

[LDF 2003.07.27] TO DO: Try to get the rotation to always go in the direction I would like.

Log

[LDF 2003.07.01] BUG FIX: Now returning bpp.first.pt if it's not equal to INVALID_POINT. Otherwise, bpp.second.pt is returned. The latter may be a valid Point, or INVALID_POINT. Before, INVALID_POINT was returned if bpp.first.b and bpp.second.b were false, but this is the case, if the intersection points didn't lie on the line segment between center and pto.

BUG FIX: Now checking to make sure that the intersection point lies in the proper direction. Now that the intersection point doesn't have to be on the line segment, it's necessary to check this.

[LDF 2003.07.20] Now using get_axis_v() and get_axis_h(), instead of accessing axis_h and axis_v directly.


(Declare Ellipse functions 1146) +≡
   Point angle_point(const real angle) const;
1202.

(Define Ellipse functions 1147) +≡

**Point Ellipse::** angle_point(const real angle) const
{
    Point Center = get_center();
    Point normal = get_normal();
    normal.shift(Center);
    Point pt0 = get_point(0);
    pt0 = Center;
    pt0.unit_vector(true);
    pt0 = max(get_axis_h(), get_axis_v()) / 2;
    /* [LDF 2002.11.06] pt0 will either lie on the perimeter of the Ellipse or beyond it. */
    pt0.shift(Center);
    pt0.rotate(Center, normal, angle);
    bool_point_pair bpp = intersection_points(Center, pt0);
    Point direction_line(pt0 - Center);
    direction_line.unit_vector(true);
    Point direction_pt;
    if (bpp.first.pt != INVALID_POINT) {
        direction_pt = bpp.first.pt;
        direction_pt = Center;
        direction_pt.unit_vector(true);
        if (direction_pt == direction_line) return bpp.first.pt;
    }
    return bpp.second.pt;
}

1203. Equality. TO DO: I’ll need to define Path::operator≡() in order to be able to define this function.

(Declare Ellipse functions 1146) +≡

#if 0
    virtual bool operator≡(const Ellipse &e);
#endif
§1204.  3DLDF-1.1.5.1  EQUALITY  335

1204.  (Define Ellipse functions 1147) +≡

    #if 0
    virtual bool Ellipse::operator==(const Ellipse &e)
    {}  
    #endif

1205.  Location of a point.  [LDF 2003.07.25.]  This function overloads
        Reg_Cl_Plane_Curve::location().  It’s simpler, because it doesn’t need to
d        copy the Ellipse and transform the copy into a major plane.  Nor does it require
        the use of solve().

        [LDF 2003.07.25.]  If the Point argument P lies in the same plane as *this,
        location() compares the sum of the distances of P from the foci to 2 times
        the maximum of axis_h and axis_v.

        [LDF 2003.07.25.]  Let m stand for (P - focus0).magnitude() + (P - focus0).magnitude(),
        d for 2 * max(axis_h, axis_v), and ε for Point::epsilon().  The return values are
        as follows:

        0  |m - d| < ε.  P lies on the perimeter of the Ellipse.
        -1  m > d.  P lies outside the Ellipse.
        1  m < d.  P lies inside the perimeter of the Ellipse.
        -2  P is not in the same plane as the Ellipse.
        -3  The Ellipse is non-elliptical.

        [LDF 2003.07.25.]  Added this function.

(Declare Ellipse functions 1146) +≡

    virtual signed short location(const Point &p) const;

Log
1206.  
(Define Ellipse functions 1147) +≡

signed short Ellipse::location (const Point &p) const
{
  bool DEBUG = false;  /* true */
  real ar_h = get_axis_h();
  real ar_v = get_axis_v();
  if (ar_h ≡ INVALID_REAL ∨ ar_v ≡ INVALID_REAL)
  { cerr ≡ "ERROR!_U_in_Ellipse::location():\n" ≡
         "Ellipse_U_is_non-elliptical._Returning_U-3.\n" ≡ flush;
    return -3;
  }
  if (!p.is_on_plane (get_plane ()))
  { cerr ≡ "WARNING!_U_in_Ellipse::location():\n" ≡
    "Point_U_doesn’t_\lie_\in_\plane_\of_Ellipse.\n" ≡ "Returning_U-2.\n" ≡ flush;
    return -2;
  }
  real max_ax = max (ar_h, ar_v);
  Point q = p - get_focus (0);
  real mag = q.magnitude ();
  q = p - get_focus (1);
  mag += q.magnitude ();
  if (fabs (mag - max_ax) < Point::epsilon())
  { if (DEBUG)    cout ≡ "Point_\lie_\on_\perimeter_\of_Ellipse.\n";
    return 0;
  }
  else if (mag > max_ax)
  { if (DEBUG)    cout ≡ "Point_\lie_\outside_\of_\perimeter_\of_Ellipse.\n";
    return -1;
  }
  else 
  { if (DEBUG)    cout ≡ "Point_\lie_\inside_\of_\perimeter_\of_Ellipse.\n";
    return 1;
  }
}

1207.  Intersection points.

1208.  Point arguments.

Log

[1DF 2003.07.27] Made the arguments const Point &.

(Declare Ellipse functions 1146) +≡

virtual bool point_pair intersection_points (const Point &p0, const Point &p1) const;
1209.  (Define Ellipse functions 1147) +≡
  bool_point_pair Ellipse::intersection_points(const Point &pt0 , const Point &pt1 ) const
  {
      return Reg_Cl_Plane_Curve::intersection_points(center , pt0 , pt1 );
  }

1210.  Path argument.  This function just checks to be sure that Path p is a line, extracts the Points, and calls the version with Point arguments, returning the latter’s return value.

[LOG]
[LDF 2003.07.27] Made argument p a const Path &. Changed, so that is_linear() is used, and get_last_point() rather than get_point(1).

(Declare Ellipse functions 1146) +≡
  virtual bool_point_pair intersection_points(const Path &p) const;

1211.  (Define Ellipse functions 1147) +≡
  bool_point_pair Ellipse::intersection_points(const Path &p) const
  {
      if (! p.is_linear()) {
          cerr << "ERROR! In Ellipse::intersection_points(const Path&):\n" <<
                "Path argument is non-linear.\n" << "Haven’t programmed this case yet.\n" <<
                "Returning INVALID_BOOL_POINTPAIR.\n" << endl << endl << flush;
          return INVALID_BOOL_POINTPAIR;
      }
      return intersection_points( p.get_point(0) , p.get_last_point() );
  }

1212.  Ellipse argument.  TO DO: Read through and explain. [LDF 2002.11.06.]

The step argument is used in the case that the Ellipses have different centers and/or axis orientation. It is the number of degrees of rotation performed while the algorithm is searching for an intersection. The default is 3, which should work as long as the Ellipses don’t differ too much in size. [LDF 2004.01.12.]

If the verbose argument is true, information about the intersection points is printed to standard output. [LDF 2003.07.01.]

If the Ellipses are coplanar, the intersection points of the perimeters of the Ellipses are returned. If the planes of the Ellipses are perpendicular or skew, the intersection line of the planes is found. Then, the intersection points of this line with the Ellipses, if they exist.

TO DO: [LDF 2003.07.20] The following code found only one intersection:

#if 0
  Ellipse t(origin , 5, 4, 90);
  Circle c(origin , 3, 90);
  c.shift(3);
  c.rotate(0,0,30);
  bool_point_quadruple bpq = t.intersection_points( c );
#endif
1213. When \( c \) was rotated by 15° or 45°, intersection points() found both intersections. Try to find out why! I want to write a Circle version with an Ellipse argument, and vice versa. If I do, I may not have to worry about this problem.

1214. [LDF 2003.07.20.]

---

<table>
<thead>
<tr>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>[LDF 2002.04.12.] Actually, it looks as if the equations Dr. Schwarmann gave me won’t do the trick. They assume both ellipses are centered about the origin. When I tried to have Mathematica solve the equations for ( x^2/a^2 + y^2/b^2 = (x - m)^2/c^2 + (y - n)^2/d^2 ), Mathematica produced solutions that took up over 35,000 lines of text!</td>
</tr>
<tr>
<td>[LDF 2002.04.12.] I’ve rewritten the other cases (parallel, perpendicular, and skew) in this function completely, and it seems like it wasn’t really worth the effort. The new version does however make use of Planes and Lines, which is not a bad thing, and it checks whether the intersection Points are on the Ellipse, so it wasn’t a total loss.</td>
</tr>
<tr>
<td>[LDF 2002.04.12.] I’ve written to Dr. Schwarmann and asked him about this problem. It may just be my ignorance, or perhaps I’ve overlooked something simple. I’m not too hopeful, however, so I’ll probably have to implement the numerical solution I’d started, after all.</td>
</tr>
<tr>
<td>[LDF 2002.04.12.] Removed old definition. Dr. Ulrich Schwarmann of the Gesellschaft für wissenschaftliche Datenverarbeitung, Göttingen, Germany showed me how to use Mathematica’s “Solve” command to solve the systems of equations that describe the intersections between two ellipses. Therefore, I’ve removed the old definition of this function, which wasn’t complete, anyway, and I’ll put in a new definition soon.</td>
</tr>
<tr>
<td>[LDF 2002.04.14.] About to start work on implementing the coplanar case again. The equations produced by Mathematica will work for the case that the ellipses have the same center. I will catch the case that they are congruent and have the same center and the case that they are non-congruent and have the same center before going on to implement a numerical solution for the case that they have different centers.</td>
</tr>
<tr>
<td>[LDF 2002.04.14.] Changed argument from const Ellipse &amp; to Ellipse, since I was having to copy it, anyway.</td>
</tr>
<tr>
<td>[LDF 2003.07.01.] Added verbose argument.</td>
</tr>
<tr>
<td>[LDF 2003.07.01.] Changed “perpendicular and skew case”. Debugged coplanar case, where the Ellipses don’t have the same centers and/or axis orientation.</td>
</tr>
<tr>
<td>[LDF 2003.07.06.] Made a minor change to the conditional that determines whether the Ellipses are coplanar or not.</td>
</tr>
<tr>
<td>[LDF 2003.08.14.] Setting verbose to true if VERBOSE_GLOBAL is true. Added VERBOSE_GLOBAL to pappgib.web today.</td>
</tr>
<tr>
<td>[LDF 2004.01.12.] Added real step argument, with 3 as its default value.</td>
</tr>
</tbody>
</table>

Declare Ellipse functions 1146) +≡

virtual bool point quaduple intersection_points(Ellipse e, const real step = 3, bool verbose = false) const;
1215.  
OLVE Ellipse functions 1147) +3
bool_point_quadruple Ellipse::intersection_points(Ellipse e, const real step, bool verbose) const {
    bool DEBUG = false;  
    if (VERBOSE_GLOBAL) verbose = true;
    if (DEBUG \ verbose)
        cout << "Entering Ellipse::intersection_points(Ellipse, e, u, const, bool, verbose)" \ endl;
    bool_point_quadruple bpq = INVALID_BOOL_POINT_QUADRUPLE;
    Plane this_plane = this->get plane();
    Plane e_plane = e.get plane();
    if (DEBUG) {
        this_plane.show("this_plane:");
        e_plane.show("e_plane:");
    }
    if (this_plane.normal \ e_plane.normal \ this_plane.normal \ e_plane.normal) {

1216.  Coplanar case.

 Log  

[LDF 2003.07.20] Now using get_axis_h() and get_axis_v(), instead of accessing axis_h and axis_v directly.

OLVE Ellipse functions 1147) +3
if (fabs(fabs(this->plane.distance) - fabs(e_plane.distance)) < Point::epsilon()) {
    if (DEBUG) {
        cout << "Ellipses are coplanar.\n" \ flush;
    }
    real ax_h = get_axis_h();
    real ax_v = get_axis_v();
1217. Congruent, same location and axis orientation. [LDF 2002.04.15.] Added check for the axis orientation. If the axes are not all vertical or horizontal, then this fails (floating exception). This means that the circular case will not be caught here, but I plan to program a specialization for Circles, so this shouldn’t cause too much of a problem.

[LDF 2002.04.14.] Added this section.

This routine only works if the axis orientation of the Ellipses is the same, or rotated 90° or 180° about an axis through the center in the direction of the normal to the plane of the Ellipse, or in the opposite direction. The axis orientation is tested on the basis of the vector from the center to points[0]. If the axis orientation is the same, or rotated 180° about the above-mentioned axis, then axis_h of *this must equal axis_h of e and axis_v of *this must equal axis_v of e. If the axis orientation is rotated at an angle of ±90°, then axis_h of *this must equal axis_v of e and axis_v of *this must equal axis_h of e.

The maximum number of intersection points of two non-congruent ellipses is four, so checking five points eliminates the possibility that we could accidentally choose intersection points on two non-congruent ellipses and mistakenly conclude that they are congruent.

(Define Ellipse functions 1147) +=

Point this_axis_orientation = get_point(0);
this_axis_orientation = get_center();
this_axis_orientation.unit_vector(true);
Point e_axis_orientation = get_point(0);
e_axis_orientation = e.get_center();
e_axis_orientation.unit_vector(true);
if (DEBUG) {
    this_axis_orientation.show("this_axis_orientation");
e_axis_orientation.show("e_axis_orientation");
}
Point normal_point = this_plane.normal;
normal_point.unit_vector(true);
if (DEBUG) normal_point.show("normal_point");
Point e_axis_orientation.rotated(e_axis_orientation);
[LDF 2002.09.26.] Start here. Added the if condition. I think this should be done, but I should check to be sure. Apparently I haven’t programmed the case that the centers aren’t the same, but it would be easy enough to do, I think.

(Define Ellipse functions 1147) +

if (e_axis_orientation != this_axis_orientation) e_axis_orientation_rotated.rotate(origin, normal_point, 90);
if (DEBUG) e_axis_orientation_rotated.show("e_axis_orientation");
if (this_center == e_center && (this_axis_orientation == e_axis_orientation || (this_axis_orientation == e_axis_orientation_rotated)))

if (DEBUG) cout << "Centers and axis orientation are the same.\n" << flush;

/* [LDF 2002.04.14.] Pick the maximum of axis_h and axis_v and multiply it by 1.5. We’ll use a line segment of this length to find intersection points with the two Ellipses. Using this length guarantees we’ll find them. Actually, max(axis_h, axis_v) ought to do the trick. */

Point pt0;
if (ax_h >= ax_v) pt0 = get_point(0);
else pt0 = get_point(number_of_points / 4);
pt0 -= center;
pt0 *= 1.5;
pt0.shift(center);
if (DEBUG) {
    this_plane.normal.show("normal");
    center.show("center");
} /* [LDF 2002.04.14.] We’ll rotate pt0 around the normal to the plane of the ellipse from center, i.e., the line segment from center to pt1. */

Point pt1 (this_plane.normal);
pt1.shift(center);
if (DEBUG) pt1.show("pt1"); /* [LDF 2002.04.14.] pt2 is the intersection of the line from center to pt0 with *this, and pt3 is the intersection of the same line with e. */

Point pt2;
Point pt3;
bool_point_pair bpp_this;
bool_point_pair bpp_e;
bool congruent_flag = true; /* [LDF 2002.04.14.] We’ll find intersection points for five values of pt0. If one set of intersection points are not the same, this means we Ellipses are not congruent and in the same location, so we break out of the loop. */

for (int i = 0; i < 5; i++) {
    if (DEBUG) cout << "i=\n" << i << endl << flush;
    if (i != 0) pt0.rotate(center, pt1, 30);
    if (DEBUG) pt0.dotlabel("0");
    bpp_this = intersection_points(center, pt0);
    bpp_e = e.intersection_points(center, pt0);
    if (bpp_this.first.b == true) {
        if (DEBUG) cout << "first\nintersection_point\(this\).\n";
        pt2 = bpp_this.first.pt;
    } else if (bpp_this.second.b == true) {
        if (DEBUG) cout << "second\nintersection_point\(this\).\n";
        pt2 = bpp_this.second.pt;
    } else {
        /* code */
    }
}
ELLIPSE ARGUMENT

cerr << "In Ellipse::intersection_points(Ellipse).\n" <<
"This can't happen! (this)\n" << "will try to continue.\n" << flush;
pt3 = INVALID_POINT;
}
if (bpp_e.first.b == true) {
  if (DEBUG) cout << "first is an intersection point(e).\n";
  pt3 = bpp_e.first.pt;
}
else if (bpp_e.second.b == true) {
  if (DEBUG) cout << "second is an intersection point(e).\n";
  pt3 = bpp_e.second.pt;
}
else {
  cerr << "In Ellipse::intersection_points(Ellipse).\n" <<
"This can't happen! (this)\n" << "will try to continue.\n" << flush;
  pt3 = INVALID_POINT;
}
if (DEBUG) {
  pt2.show("2");
  pt2.dotlabel("2");
  pt3.show("3");
  pt3.dotlabel("3", "bot");
}
if (pt2 == INVALID_POINT || pt3 == INVALID_POINT || pt2 != pt3) {
  if (DEBUG) cout << "Ellipses are not congruent. Breaking.\n";
  congruent_flag = false;
  break;
}
else continue;
} /* for */
if (DEBUG) {
  cout << "congruent_flag==true\n" << congruent_flag << endl << flush;
}
if (congruent_flag == true) {
  cerr << "WARNING! In Ellipse::intersection_points(Ellipse).\n" <<
"Ellipses are congruent and in the same location.\n" <<
"Returning INVALID_BOOL_POINT_QUADRUPEL.\n" << flush;
  return INVALID_BOOL_POINT_QUADRUPEL;
}
else if (DEBUG) {
  cout << "All five points are not on both ellipses.\n" << flush;
}
} /* End of test of congruency and same location. */
/*/ [LDF 2002.09.26] START HERE, Program this case!! */
else if (DEBUG) {
  cout << "The centers are different, or the axis orientation\n" <<
"is different, or both.\n" << "Haven't programmed this case yet.\n"
}
Ellipse copy(*this);
Point copy_center(copy.get_center());
Transform t;
Transform t_inverse;
1219. Shift to origin (if necessary).
(Define Ellipse functions 1447) \(\equiv\)
if \((\text{copy\_center} \neq \text{origin})\) {
if (DEBUG) {
    cout << "Shifting copy to origin.\n" << flush;
}
t.shift(-copy\_center.get\_x(), -copy\_center.get\_y(), -copy\_center.get\_z());
copy *= t;
copy\_center *= t;
e *= t;
} else if (DEBUG) {
    cout << "copy is already at origin.\n" << flush;
}

1220. Get coordinates of normal.
(Define Ellipse functions 1447) \(\equiv\)
real normal\_x = this\_plane.normal.get\_x();
real normal\_y = this\_plane.normal.get\_y();
real normal\_z = this\_plane.normal.get\_z();
1221. Determine the orientation of the ellipse and rotate, if it's not already in a plane parallel to a major plane. Rotating the **Ellipse** can cause inaccuracies in the coordinate values, so if the ellipse is already in a major plane, (i.e., one perpendicular to a major axis), we leave it where it is.

---

(Define **Ellipse** functions 1147) +≡

```cpp
unsigned short orientation;
const unsigned short X_Y = 0;
const unsigned short X_Z = 1;
const unsigned short Z_Y = 2;
#else
const unsigned short OTHER = 3;
#endif
if (normal_x == 0 && normal_y == 0 && normal_z == 0) {
  cerr << "ERROR in Ellipse::intersection_points():\n" << "Normal = [0,0,0]" <<
       "Returning INVALID_BOOL_POINT_QUADRUPE\n" << flush;
  return INVALID_BOOL_POINT_QUADRUPE;
} else if (normal_x == 0 && normal_y == 0) /* Ellipse lies in a plane parallel to x-y plane. */ {
  if (DEBUG) cout << "Ellipse lies in a plane parallel to x-y plane\n" << flush;
  orientation = X_Y;
} else if (normal_x == 0 && normal_z == 0) /* Ellipse lies in a plane parallel to x-z plane. */ {
  if (DEBUG) cout << "Ellipse lies in a plane parallel to x-z plane\n" << flush;
  orientation = X_Z;
} else if (normal_y == 0 && normal_z == 0) /* Ellipse lies in a plane parallel to z-y plane. */ {
  if (DEBUG) cout << "Ellipse lies in a plane parallel to z-y plane\n" << flush;
  orientation = Z_Y;
} else /* Ellipse doesn’t lie in a plane parallel to a major plane. */ {
  if (DEBUG) cout << "Ellipse doesn’t lie in a plane parallel to a major plane.\n" << flush;
  /* Put it in x-z plane. */
  if (DEBUG) {
    copy_center.dotlabel("cn");
  }
  Point ellipse_pt0 = copy.get_point(0);
  Transform t0;
  t0.align_with_axis(copy_center, ellipse_pt0, ’x’);
  copy *= t0;
  copy_center *= t0;
  ellipse_pt0 *= t0;
  e *= t0;
  if (DEBUG) {
```

---

Log

[LDF 2003.08.27.] Commented-out the declaration of OTHER, because it's never used. I haven't deleted it, just in case.
ellipse_pt0.show("ellipse_pt0");
ellipse_pt0.dotlabel("0");
}
Point ellipse_pt4 = copy.get_point(number_of_points / 4);
if (DEBUG) ellipse_pt4.show("ellipse_pt4");

Transform t1;
t1.align_with_axis(copy_center, ellipse_pt4, 'z');
copy *= t1;

copy_center *= t1;
e *= t1;
ellipse_pt4 *= t1;
t0 *= t1;
t *= t1;
orientation = X_Z;
}
t.inverse = t.inverse();
Point e.center = e.get_center();
if (DEBUG) {
copy_center.show("copy_center");
e.center.show("e_center");
}
Ellipses have the same center and orientation. If they do, then there is an algebraic solution we can apply to find the intersection points.

(Define Ellipse functions 1147) +
Point copy_axis_orientation(copy.get_point());
copy_axis_orientation = copy_center;
copy_axis_orientation.unit_vector(true);
e.axis.orientation = e.get_point();
e.axis.orientation = e_center;
e.axis.orientation.unit_vector(true); if ((e_center == origin \land copy_center == origin) \land (copy_axis.orientation == e_axis.orientation \lor copy_axis.orientation ==
-e.axis.orientation) \lor (copy_axis.orientation == e_axis.orientation_rotated \lor copy_axis.orientation ==
-e.axis.orientation_rotated)) {
if (DEBUG) cout << "Both ellipses have the same center and axis orientation.\n";
Point pt20;
Point pt21;
Point pt22;
Point pt23;
real x;
real y;
real a = ax.h / 2.0;
real b = ax.v / 2.0;
real c = e.get_axis_h() / 2.0;
real d = e.get_axis_v() / 2.0;
if (DEBUG) {
    cout << "a = u\n" << a << endl << flush;
    cout << "b = u\n" << b << endl << flush;
    cout << "c = u\n" << c << endl << flush;
    cout << "d = u\n" << d << endl << flush;
}
real aa = (a * a);
real bb = (b * b);
real cc = (c * c);
real dd = (d * d);
if (DEBUG) {
    cout << "aa = u\n" << aa << endl << flush;
    cout << "bb = u\n" << bb << endl << flush;
    cout << "cc = u\n" << cc << endl << flush;
    cout << "dd = u\n" << dd << endl << flush;
}
real denominator;
real numerator;
if (DEBUG) cout << "x, coordinate.\n";
denominator = (aa - ((bb * cc) / dd)) * dd;
umerator = bb * (aa - cc);
if (DEBUG) {
    cout << "numerator = u\n" << numerator << endl << flush;
    cout << "denominator = u\n" << denominator << endl << flush;
}
if (denominator == 0) {
    if (DEBUG) cout << "x = 0, INVALID_REAL.\n";
\[ x = \text{INVALID\_REAL}; \]

else {
  try {
    x = -c * sqrt(1 - (numerator\_denominator));
  }
  catch(...) {
    x = \text{INVALID\_REAL};
    if (DEBUG) cout << "x_{u}=u_{u}\text{INVALID\_REAL}\n";
  }
}

if (DEBUG) {
  cout << "x_{u}=u_{u}\n" << x << endl << flush;
  cout << "y_{u}\_coordinate: \n";
}

numerator = b * sqrt(fabs(aa - cc));

denominator = sqrt(fabs(aa - ((bb + cc)/dd)));
if (DEBUG) {
  cout << "numerator_{u}=u_{u}\" << numerator << endl << flush;
  cout << "denominator_{u}=u_{u}\" << denominator << endl << flush;
}

if (denominator == 0) {
  if (DEBUG) cout << "y_{u}=u_{u}\text{INVALID\_REAL}\n";
  y = \text{INVALID\_REAL};
}
else {
  try {
    y = -\text{numerator/\text{denominator}};
  }
  catch(...) {
    y = \text{INVALID\_REAL};
    if (DEBUG) cout << "y_{u}=u_{u}\text{INVALID\_REAL}\n";
  }
}
if (DEBUG) {
  cout << "y_{u}=u_{u}\n" << y << endl << flush;
}
The ellipses can intersect at no points, two points, or four points.

- If they do not intersect, then one of the ellipses must be inside the other,
- If they intersect at two points, then either \( x = 0 \) or \( y = 0 \) (but not both),
- Otherwise, they intersect at four points.

```cpp
1223. \( \text{Define Ellipse functions} \ 1147 \) \+\-
if \((x \equiv \text{INVALID\_REAL} \lor y \equiv \text{INVALID\_REAL})\) {
    \text{cerr} \ll \text{"WARNING\!ln_0Ellipse::intersection_points(Ellipse).\n    "n}\ll
    "Ellipses\!don\!'t\!intersect.\!" \ll \text{"Returning\!\text{INVALID\_BOOL\_POINT\_QUADRUPLE.\n    n}\n    
    return \text{INVALID\_BOOL\_POINT\_QUADRUPLE;}
    }
else if \((\text{orientation} \equiv \text{X\_Y})\) {
    \text{pt20.set}(x, y, 0);
    \text{pt21.set}(-x, -y, 0);
    if \((y \neq 0 \land x \neq 0)\) {
        \text{pt22.set}(x, -y, 0);
        \text{pt23.set}(-x, y, 0);
    } else {
        \text{pt22 = INVALID\_POINT;}
        \text{pt23 = INVALID\_POINT;}
    }
}
else if \((\text{orientation} \equiv \text{X\_Z})\) {
    \text{pt20.set}(x, 0, y);
    \text{pt21.set}(-x, 0, -y);
    if \((y \neq 0 \land x \neq 0)\) {
        \text{pt22.set}(x, 0, -y);
        \text{pt23.set}(-x, 0, y);
    } else {
        \text{pt22 = INVALID\_POINT;}
        \text{pt23 = INVALID\_POINT;}
    }
}
else if \((\text{orientation} \equiv \text{Z\_Y})\) {
    \text{pt20.set}(0, y, x);
    \text{pt21.set}(0, -y, -x);
    if \((x \neq 0 \land y \neq 0)\) {
        \text{pt22.set}(0, y, -x);
        \text{pt23.set}(0, -y, x);
    } else {
        \text{pt22 = INVALID\_POINT;}
        \text{pt23 = INVALID\_POINT;}
    }
}
else {
    \text{cerr} \ll \text{"ERROR\!ln_0Ellipse::intersection_points(Ellipse).\n    "n}\ll
    "This\!can\!'t\!happen!orientation\!\has\!invalid\!value:\!
    " orientation \ll
    endl \ll \"Will\!\to\!\try\!\to\!\continue.\n    "n}\ll \text{flush;}
}
if \((\text{DEBUG})\) {
```
signed short ss_copy;  
signed short ss_e;
if (pt20 != INVALID_POINT) {
    ss_copy = copy_location(pt20);
    if (DEBUG) cout << "ss_copy==\u003d\u003e" << ss_copy << endl << flush;
    ss_e = e.location(pt20);
    if (DEBUG) cout << "ss_e==\u003d\u003e" << ss_e << endl << flush;
}
if (ss_copy == 0 && ss_e == 0) {
    pt20 = t_inverse;
    pt21 = t_inverse;
    bpg.first.b = true;
    bpg.first.pt = pt20;
    bpg.second.b = true;
    bpg.second.pt = pt21;
    if (pt22 == INVALID_POINT) {
        if (DEBUG) cout << "Ellipses intersect at 4 points.\n";
        pt22 = t_inverse;
        bpg.third.b = true;
        bpg.third.pt = pt22;
    } else {
        if (DEBUG) cout << "Ellipses intersect at 2 points.\n";
    }
    if (pt23 == INVALID_POINT) {
        pt23 = t_inverse;
        bpg.fourth.b = true;
        bpg.fourth.pt = pt23;
    }
    return bpg;
} else {
    cerr << "WARNING! In Ellipse::intersection_points(Ellipse).\n";
    ss_e = e.location(copy.get_point(0));
    if (ss_e == 1) cerr << "this is inside of \u003e\n";
    else if (ss_e == -1) cerr << "e is inside of \u003e this.\n";
    else {
        cerr << "This can’t happen! Invalid value for ss_e: \u003d\n" << ss_e << endl;
    }
    cerr << "No intersection.\n" << "Returning INVALID_BOOL_POINT_QUADRUPLE.\n\n";
    return INVALID_BOOL_POINT_QUADRUPLE;
}/* End of “Ellipses have the same center and axis orientation”. */
1224. Ellipses have different centers and/or axis orientation. There is no simple algebraic solution for this case, so I have to implement a numerical one here.

```cpp
Define Ellipse functions

Point currr_point;
signed short currr_location;
signed short location_switch;
unsigned short intersectionCtr = 0;
real save_angle;
real in_angle;
real out_angle;
real test_angle;

if (DEBUG) {
    e.show("e:");
    copy.show("copy:");
}

cout << "In Ellipse::intersection_points():\n" <<
    "Searching for Ellipse_intersections..\n" << "This can take some time...\n" << flush;

for (real i = 0; i < 360; i += step) {
    if (DEBUG) cout << "i=" << i << endl << flush;
    currr_point = copy.angle_point(i);
    if (DEBUG) currr_point.show("currr_point");
    currr_location = e.location(currr_point);
    if (DEBUG) cout << "currr_location=" << currr_location << endl << flush;
    if (currr_location == 0) {
        if (DEBUG) {
            cout << "Found an intersection point!\n";
            currr_point.show("currr_point:");
        }
        (Handle intersection point 1225)
        i += .5;
        save_angle = i;
        currr_point = copy.angle_point(i);
        location_switch = e.location(currr_point);
        continue;
    } // if End of “Found an intersection point” */
else if (i == 0) {
    location_switch = currr_location;
    save_angle = 0;
    continue;
}
else if (currr_location == 1) {
    if (DEBUG) cout << "Point is inside e.\n";
}
else if (currr_location == -1) {
    if (DEBUG) cout << "Point is outside e.\n";
}
else {
```
```cpp
if (curr_location != location_switch) {
    if (DEBUG) {
        cout << "Found a transition!\n";
        cout << "i = " << i << endl << flush;
        cout << "save_angle = " << save_angle << endl << flush;
        cout << "curr_location = " << curr_location << endl << flush;
        cout << "location_switch = " << location_switch << endl << flush;
    }
    if (location_switch == 1) {
        in_angle = save_angle;
        out_angle = i;
    } else {
        in_angle = i;
        out_angle = save_angle;
    }
    while (true) {
        test_angle = (in_angle + out_angle) / 2;
        curr_point = copy_angle_point(test_angle);
        curr_location = e.location(curr_point);
        if (curr_location == 0) {
            if (DEBUG) {
                cout << "Found an intersection point!\n";
                curr_point.show("curr_point:");
            }
            // Handle intersection point
            i = floor(test_angle);
            i += .5;
            if (i < test_angle) i += .5;
            save_angle = i;
            curr_point = copy_angle_point(i);
            location_switch = e.location(curr_point);
            break;
        } else if (curr_location == 1) {
            in_angle = test_angle;
            continue;
        } else if (curr_location == -1) {
            out_angle = test_angle;
            continue;
        } else {
            cerr << "ERROR! In Ellipse::intersection_points()\n" <<
                "This can't happen! Invalid value for curr_location: \n" << curr_location <<
                "Returning INVALID_BOOL_POINT_QUADRUPLE.\n\n" << flush;
            return INVALID_BOOL_POINT_QUADRUPLE;
        }
    }
}
```
1225. Handle intersection point.

```
(Handle intersection point 1225) ≡
{
    ++intersection_ctr;
    if (DEBUG) {
        cout ≡ "intersection_ctr_i=a_i " ≡ intersection_ctr ≡ endl ≡ flush;
    }
    if (intersection_ctr ≡ 1) {
        bpq.first.b = true;
        bpq.first.pt = curr_point;
        bpq.first.pt *= t_inverse;
    } else if (intersection_ctr ≡ 2) {
        bpq.second.b = true;
        bpq.second.pt = curr_point;
        bpq.second.pt *= t_inverse;
    } else if (intersection_ctr ≡ 3) {
        bpq.third.b = true;
        bpq.third.pt = curr_point;
        bpq.third.pt *= t_inverse;
    } else if (intersection_ctr ≡ 4) {
        bpq.fourth.b = true;
        bpq.fourth.pt = curr_point;
        bpq.fourth.pt *= t_inverse;
        if (DEBUG || verbose) {
            cout ≡ "InEllipse::intersection_points(Ellipse_e, const bool verbose)" ≡
                "Found four intersection points. \n" ≡ "Returning bpq:\n\n"
        }
        return bpq;
    } else {
        cerr ≡ "ERROR! InEllipse::intersection_points(Ellipse).\n" ≡
            "This can’t happen! Invalid value for intersection_ctr_i ≡
            intersection_ctr ≡ \
                "\nReturning INVALID_BOOL_POINT_QUADRUPE.\n\n"
        return INVALID_BOOL_POINT_QUADRUPE;
    }
}
```

This code is used in section 1224.
§1226. Parallel and non-coplanar case.

(Define Ellipse functions 1147) +

else {
    
    cerr << "WARNING! In Ellipse::intersection_points(Ellipse).\n" << "Ellipses are in parallel planes, so they don't intersect.\n" << "Returning INVALID_BOOL_POINT_QUADRUPLE.\n\n";
    return INVALID_BOOL_POINT_QUADRUPLE;
}
1227. Perpendicular and skew cases. These cases are handled in exactly the same way.

```cpp
(Define Ellipse functions 1147) + ·
else {
    if (DEBUG) cout << "Ellipses are in perpendicular or skew planes.\n" << flush;
    Line isect_line(this_plane.intersection_line(e_plane));
    if (DEBUG) isect_line.show("isect_line:");
    Point pt0(isect_line.direction);
    pt0.shift(isect_line.position);
    bool_point_pair bpp = intersection_points(isect_line.position, pt0);
    if (DEBUG) {
        cout << "bpp.first.b==u" << bpp.first.b << endl << flush;
        bpp.first.pt.show("bpp.first_point:");
        cout << "bpp.second.b==u" << bpp.second.b << endl << flush;
        bpp.second.pt.show("bpp.second_point:");
    }
    bpg.first.pt = bpp.first.pt;
    bpg.first.b = (bpp.first.pt == INVALID_POINT) ? false : true;
    bpg.second.pt = bpp.second.pt;
    bpg.second.b = (bpp.second.pt == INVALID_POINT) ? false : true;
    bpg = e.intersection_points(isect_line.position, pt0);
    bpg.third.pt = bpg.first.pt;
    bpg.third.b = (bpp.first.pt == INVALID_POINT) ? false : true;
    bpg.fourth.pt = bpp.second.pt;
    bpg.fourth.b = (bpp.second.pt == INVALID_POINT) ? false : true;
    signed short s;_
    signed short s.E;
    bool tempbool;
    string temp_string;
    if (bpg.first.b) {
        s. = location(bpg.first.pt);
        s.E = e.location(bpg.first.pt);
        tempbool = bpg.first.b;
        temp_string = "First";
        (Check intersection point locations 1228)
    }
    else if (verbose) cout << "First intersection point is INVALID_POINT.\n";
    if (bpg.second.b) {
        s. = location(bpg.second.pt);
        s.E = e.location(bpg.second.pt);
        tempbool = bpg.second.b;
        temp_string = "Second";
        (Check intersection point locations 1228)
    }
    else if (verbose) cout << "Second intersection point is INVALID_POINT.\n";
    if (bpg.third.b) {
        s. = location(bpg.third.pt);
        s.E = e.location(bpg.third.pt);
        tempbool = bpg.third.b;
        temp_string = "Third";
        (Check intersection point locations 1228)
    }
```
} else if (verbose) cout << "Third intersection point is INVALID_POINT.\n";
if (!bpq_fourth) {
    s_L = location(bpq_fourth.pt);
    s_R = e.location(bpq_fourth.pt);
    temp_bool = bpq_fourth.b;
    temp_string = "Fourth";
    // Check intersection point locations 1228
} else if (verbose) cout << "Fourth intersection point is INVALID_POINT.\n";
} // else. End of perpendicular and skew cases. */
if (DEBUG || verbose) cout << "Exiting Ellipse::intersection_points(Ellipse)\n" << flush;
return bpq; }

1228. Check intersection point locations. This is used in the "Perpendicular and Skew Cases" of intersection_points(Ellipse e, const bool verbose).

[LDf 2003.07.01.] Added this section.

(LCheck intersection point locations 1228) \equiv

if (temp_bool) {
    if (verbose) {
        if (s_L \equiv 0 \wedge s_R \equiv 0)
            cout \equiv temp_string \equiv "Point lies on the perimeter of both ellipses.\n"
        } else if (s_L \equiv 0)
            cout \equiv temp_string \equiv "Point lies on the perimeter of this.\n"
        } else if (s_L \equiv -1)
            cout \equiv temp_string \equiv "Point lies outside this.\n"
        } else if (s_L \equiv 1)
            cout \equiv temp_string \equiv "Point lies inside this.\n"
        } else if (s_R \equiv -2)
            cout \equiv temp_string \equiv "Point doesn’t lie in the plane of this.\n"
        } if (s_R \equiv 0 \wedge s_L \neq 0)
            cout \equiv temp_string \equiv "Point lies on the perimeter of this.\n"
        } if (s_R \equiv -1)
            cout \equiv temp_string \equiv "Point lies outside this.\n"
        } else if (s_R \equiv 1)
            cout \equiv temp_string \equiv "Point lies inside this.\n"
        } else if (s_R \equiv -2)
            cout \equiv temp_string \equiv "Point doesn’t lie in the plane of this.\n"
    }
}

This code is used in section 1227.

1229. Transformations.

[LDf 2003.04.27.] Finished adding the transformation functions. I already had shift(), now I have the rest of them.

1230. Performing a transformation.
1231. **Do transform.**  [LDF 2003.07.20] Performs a transformation on *this.

If `check ≡ true`, `is_elliptical()` is called on *this following the transformation. If the latter causes *this to become non-elliptical, `axis_h`, `axis_v`, `linear_eccentricity`, and `numerical_eccentricity` are set to INVALID_REAL, and a warning is issued to `stderr`. `center`, `focus0`, and `focus1` are not set to INVALID_POINT. They may no longer really be the center and foci of the (non-elliptical) Ellipse, but they may have some use for the programmer and/or user.

If `check ≡ true`, and the transformation does not cause *this to become non-elliptical, `axis_h` and `axis_v` are recalculated.

---

<table>
<thead>
<tr>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>[LDF 2003.07.20] Added this function. It makes it possible to perform a transformation on an Ellipse, optionally calling <code>is_elliptical()</code>. It is called in <code>is_elliptical()</code> with <code>check ≡ false</code>. This prevents <code>operator *=</code>() and <code>is_elliptical()</code> from calling each other ad infinitum.</td>
</tr>
<tr>
<td>[LDF 2003.07.25] Added code for recalculating <code>linear_eccentricity</code>, <code>focus0</code> and <code>focus1</code>.</td>
</tr>
<tr>
<td>[LDF 2003.07.25] BUG FIX: <code>axis_h</code> and <code>axis_v</code> were too small by half. Now multiplying by 2.</td>
</tr>
</tbody>
</table>

---

(Declare Ellipse functions 1146) +≡

**virtual Transform do_transform(const Transform &t, bool check = false);**
1232.

(Define Ellipse functions 1147) +\Xi

Transform Ellipse :: do_transform (const Transform &t, bool check)
{
    bool DEBUG = false; /* true */
    focus0 == focus1 == center == Path::operator+(t);
    real old_axis_h = axis_h;
    real old_axis_v = axis_v;
    if (check) {
        if (is_elliptical()) {
            Point c = get_center();
            axis_h = (get_point(0) - c).magnitude() * 2;
            axis_v = (get_point(number_of_points/4) - c).magnitude() * 2;
            real axis_h_half = axis_h/2;
            real axis_v_half = axis_v/2;
            if (fabs(axis_h - old_axis_h) > Point::epsilon() && fabs(axis_v - old_axis_v) > Point::epsilon()) {
                if (DEBUG) cout << "Recalculating linear eccentricity, u" <<
                    "numerical_eccentricity and foci.\n" << flush;
                if (axis_h > axis_v) {
                    linear_eccentricity = sqrt((axis_h_half * axis_h_half) - (axis_v_half * axis_v_half));
                    numerical_eccentricity = linear_eccentricity / axis_h_half;
                    focus1 = focus0 = (get_point(0) - get_center());
                }
                else {
                    linear_eccentricity = sqrt((axis_v_half * axis_v_half) - (axis_h_half * axis_h_half));
                    numerical_eccentricity = linear_eccentricity / axis_v_half;
                    focus1 = focus0 = (get_point(number_of_points/4) - get_center());
                }
                focus0.unit_vector(true);
                focus1.unit_vector(true);
                focus0 *= -linear_eccentricity;
                focus1 *= linear_eccentricity;
                focus1 *= focus0.shift(get_center());
            }
            else if (DEBUG)
                cout << "axis_h and axis_v are unchanged.\n" << "Not recalculating foci.\n" << flush;
        } /* if (is_elliptical() ) */
    } else {
        cerr << "WARNING\nIn Ellipse::do_transform(const Transform&):\n" <<
            "This transformation has made this non-elliptical!\n" << endl << endl << flush;
        axis_h = axis_v = linear_eccentricity = INVALID_REAL;
        numerical_eccentricity = INVALID_REAL;
    }
    return t;
}
1233. Operator.

<table>
<thead>
<tr>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>[LDF 2002.04.12.] Added this section.</td>
</tr>
<tr>
<td>[LDF 2003.07.20.] Changed, so that it calls <code>do_transform()</code> with <code>check == true</code>, so that <code>is_elliptical()</code> is called.</td>
</tr>
</tbody>
</table>
| (Declare Ellipse functions 1146) +≡  
  virtual Transform operator *(const Transform &t); |

1234.

<table>
<thead>
<tr>
<th>Log</th>
</tr>
</thead>
</table>
| (Define Ellipse functions 1147) +≡  
  Transform Ellipse :: operator *(const Transform &t) |
|   { |
|     return do_transform(t, true); |
|   } |

1235. Rotation around the main axes.

<table>
<thead>
<tr>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>[LDF 2003.07.25.] Changed, so that <code>do_transform()</code> is called with <code>check == false</code>. Rotation can neither change the lengths of <code>axis_h</code> or <code>axis_v</code>, nor make an Ellipse non-elliptical, so there’s no need to check <code>*this</code> after rotation.</td>
</tr>
</tbody>
</table>
| (Declare Ellipse functions 1146) +≡  
  virtual Transform rotate(const real x, const real y = 0, const real z = 0); |

1236.

<table>
<thead>
<tr>
<th>Log</th>
</tr>
</thead>
</table>
| (Define Ellipse functions 1147) +≡  
  Transform Ellipse :: rotate(const real x, const real y, const real z) |
|   { |
|     Transform t; |
|     t.rotate(x, y, z); |
|     return do_transform(t, false); |
|   } |

1237. Scale.

<table>
<thead>
<tr>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>[LDF 2003.07.20.] Added check for whether <code>*this</code> is still elliptical after the scaling operation.</td>
</tr>
</tbody>
</table>
| (Declare Ellipse functions 1146) +≡  
  virtual Transform scale(real x, real y = 1, real z = 1); |
1238. (Define Ellipse functions 1147) +≡
Transform Ellipse :: scale (real x, real y, real z)
{
    Transform t;
    t.scale (x, y, z);
    return (*this *= t);
}

1239. Shear. Log

[1DF 2003.07.20.] Added check for whether *this is still elliptical after the shearing operation.

(Declare Ellipse functions 1146) +≡
virtual Transform shear (real xy, real xx = 0, real yx = 0, real yz = 0, real zx = 0, real zy = 0);

1240. (Define Ellipse functions 1147) +≡
Transform Ellipse :: shear (real xy, real xx, real yx, real yz, real zx, real zy)
{
    Transform t;
    t.shear (xy, xx, yx, yz, zx, zy);
    return (*this *= t);
}

1241. Shift. Log

[1DF 2003.07.25.] Changed, so that do_transform() is called with check ≡ false. Shifting can neither change the lengths of axis_h or axis_v, nor make an Ellipse non-elliptical, so there’s no need to check *this after shifting.

(Declare Ellipse functions 1146) +≡
virtual Transform shift (real xx, real yy = 0, real zz = 0);

1243. (Define Ellipse functions 1147) +≡
Transform Ellipse :: shift (real xx, real yy, real zz)
{
    Transform t;
    t.shift (xx, yy, zz);
    return do_transform (t, false);
}

1244. Point argument. Log

(Declare Ellipse functions 1146) +≡
virtual Transform shift (const Point &p);
1245.  
(Define Ellipse functions 1147) 
   Transform Ellipse::shift(const Point &p) 
   { 
      return shift(p.get_x(), p.get_y(), p.get_z()); 
   }

1246. Shift times.

1247. real arguments.

[Log] Now performing shift_times() on focus0 and focus1, too.

(Declare Ellipse functions 1146) 
   virtual void shift_times(real x, real y = 1, real z = 1);

1248.  
(Define Ellipse functions 1147) 
   void Ellipse::shift_times(real x, real y, real z) 
   { 
      Path::shift_times(x, y, z); 
      focus1.shift_times(x, y, z); 
      focus0.shift_times(x, y, z); 
      center.shift_times(x, y, z); 
      return; 
   }

1249. Point argument.

(Declare Ellipse functions 1146) 
   virtual void shift_times(const Point &p);

1250.  
(Define Ellipse functions 1147) 
   void Ellipse::shift_times(const Point &p) 
   { 
      return shift_times(p.get_x(), p.get_y(), p.get_z()); 
   }

1251. Rotation around an arbitrary axis.
1252. Point arguments.

[LDF 2003.05.02] Changed name of this function from \texttt{rotate\_around()} to \texttt{rotate()}. This function now overloads \texttt{rotate()} with three \texttt{real} arguments.

[LDF 2003.07.25] Changed, so that \texttt{do\_transform()} is called with \texttt{check \equiv false}. Rotation can neither change the lengths of \texttt{axis\_h} or \texttt{axis\_v}, nor make an \texttt{Ellipse} non-elliptical, so there's no need to check *this after rotation.

\begin{verbatim}
(Declare Ellipse functions 1146) +≡
   virtual Transform rotate(const Point &p0, const Point &p1, const real angle = 180);
\end{verbatim}

1253.

(Define Ellipse functions 1147) +≡

\begin{verbatim}
Transform Ellipse::rotate(const Point &p0, const Point &p1, const real angle)
{
   Transform t;
   t.rotate(p0, p1, angle);
   return do_transform(t, false);
}
\end{verbatim}

1254. Path arguments.

[LDF 2003.05.02] Changed name of this function from \texttt{rotate\_around()} to \texttt{rotate()}. This function now overloads \texttt{rotate()} with three \texttt{real} arguments.

\begin{verbatim}
(Declare Ellipse functions 1146) +≡
   virtual Transform rotate(const Path &p, const real angle = 180);
\end{verbatim}
(Define Ellipse functions 1147) +≡

Transform Ellipse::rotate(const Path &p, const real angle)
{
    if (!p.is_linear()) {
        cerr << "ERROR! In Ellipse::rotate(Path, real).\n        "Path is not a line.\n        Returning INVALID_TRANSFORM.\n        \n        ";
        return INVALID_TRANSFORM;
    }
    return rotate(p.get_point(0), p.get_last_point(), angle);
}

1256. Rectangles.

1257. Surrounding rectangle.

(Declare Ellipse functions 1146) +≡

Rectangle out_rectangle() const;
1258. (Define Ellipse functions 1147) +

\begin{verbatim}
Rectangle Ellipse::out_rectangle() const
{
    Point C(get_center());
    Point pt0(get_point(0));
    Point pt1(get_point(number_of_points / 2));
    Path pa0(pt0, pt1);
    Point pt2(get_point(number_of_points / 4));
    Point pt3(get_point(3 * number_of_points / 4));
    Point pt4(pt1);
    pt4 -= C;
    Point pt5(pt2);
    pt5.shift(pt4);
    Point pt6(pt3);
    pt6.shift(pt4);
    Point pt7(pt0);
    pt7 -= C;
    Point pt8(pt2);
    pt8.shift(pt7);
    Point pt9(pt3);
    pt9.shift(pt7);

    #if 0
    pt0.dotlabel("0");
    pt1.dotlabel("1");
    pt2.dotlabel("2");
    pt3.dotlabel("3");
    pt5.dotlabel("5");
    pt6.dotlabel("6");
    pt8.dotlabel("8");
    pt9.dotlabel("9");
    #endif

    Rectangle r(pt6, pt9, pt8, pt5);
    return r;
}
\end{verbatim}

1259. Inscribed rectangle.

---

[Log]


---

(Declare Ellipse functions 1146) +

\begin{verbatim}
Rectangle in_rectangle() const;
\end{verbatim}
1260.  
(Define Ellipse functions 1147) +≡
Rectangle Ellipse::in_rectangle() const
{
  Rectangle r0 = out_rectangle();
  bool_point_pair bpp0 = intersection_points(r0.get_point(0), r0.get_point(2));
  bool_point_pair bpp1 = intersection_points(r0.get_point(1), r0.get_point(3));
  if (bpp0.first.pt == INVALID_POINT || bpp0.second.pt == INVALID_POINT || bpp1.first.pt ==
      INVALID_POINT || bpp1.second.pt == INVALID_POINT) {
    cerr << "Intersection didn’t work.\n" << "Returning empty rectangle.\n\n" << flush;
    Rectangle r2;
    return r2;
  }
  Rectangle r1(bpp0.second.pt, bpp0.first.pt, bpp0.first.pt, bpp1.second.pt);
  return r1;
}

1261.  Draw surrounding rectangle.

[Log]
[LD 2003.07.01.] Changed the return value from void to Rectangle. Now the surrounding Rectangle
is returned.

(Declare Ellipse functions 1146) +≡
Rectangle draw_out_rectangle(const Color &draw_color = *Colors::default_color, string
  ddashed = "", string ppens = "", Picture &picture = current_picture) const;

1262.
(Define Ellipse functions 1147) +≡
Rectangle Ellipse::draw_out_rectangle(const Color &draw_color, string ddashed, string
  ppens, Picture &picture) const
{
  Rectangle r = out_rectangle();
  r.draw(draw_color, ddashed, ppens, picture);
  return r;
}

1263.  Draw inscribed rectangle.

[Log]
[LD 2003.07.01.] Changed the return value from void to Rectangle. Now the inscribed Rectangle
is returned.

(Declare Ellipse functions 1146) +≡
Rectangle draw_in_rectangle(const Color &draw_color = *Colors::default_color, string
  ddashed = "", string ppens = "", Picture &picture = current_picture) const;
1264. 
(Define Ellipse functions 1147) +

  Rectangle Ellipse:: draw_in_rectangle (const Color &draw_color, string dashed, string ppem, Picture &picture) const

  {
   Rectangle r = in_rectangle();
   r.draw (draw_color, dashed, ppem, picture);
   return r;
  }

1265. Rectangle functions.  [LDF 2003.07.18.] TO DO: Add undraw_in_ellipse(), fillout_ellipse(), etc. Also, I should add versions with the Picture argument first.

   Log

   [LDF 2003.07.18.] Added this section. These functions are declared in rects.web, but must be defined here, because Ellipse is an incomplete type there.

1266. Ellipses.
1267. Surrounding Ellipse.

[LDIC 2003.07.18.] Added this function.

```cpp
{ Define Rectangle functions 104 } +
    Ellipse Rectangle::out_ellipse() const
    {
      Point C = get_center();
      Point p0 = get_point(0);
      Point M = get_mid_point(1);
      Point normal = get_normal();
      normal.shift(C);
      real out_distance = (p0 - C).magnitude();
      Transform t;
      real h_length = (get_point(1) - p0).magnitude();
      real v_length = (get_point(3) - p0).magnitude();
      p0 *= M *= t.align_with_axis(C, normal, 'y');
      Point x_axis_pt(1);
      real angle = M.angle(x_axis_pt);
      p0 *= M *= t.rotate(0, angle);
      if (M.unit_vector() != x_axis_pt)
        {
          cerr << "WARNING! in \n| Rectangle::in_ellipse() | in \n| std::is LINUX (1, 0, 0) !\n| "
          M.show("M: ");
          // I'd rather output this to stderr, but I don't have a way to do this yet. [LDIC 2003.07.18.]
          cout << endl << flush;
        }
      Ellipse e(origin, h_length, v_length);
      bool_point_pair bpp = e.intersection_points(origin, p0);
      real in_distance;
      if (bpp.first.b)
        in_distance = bpp.first.pt.magnitude();
      else if (bpp.second.b)
        in_distance = bpp.second.pt.magnitude();
      else
        {
          cerr << "ERROR! in \n| Rectangle::out_ellipse() | in \n| Couldn't find intersection point." << "Returning an empty Ellipse. \n| " << flush;
          Ellipse r;
          return r;
        }
      real scale_value = out_distance / in_distance;
      e.scale(scale_value, 0, scale_value);
      e *= t.inverse();
      return e;
    }
```
1268. Enclosed Ellipse.

Log

[LD 2003.07.18.] Added this function.

\{ Define Rectangle functions 104 \} \equiv

Ellipse Rectangle::in_ellipse() const
{
    Point C = get_center();
    Point M = get_mid_point(1);
    Point normal = get_normal();
    normal.shift(C);
    Transform t;
    real h_length = (get_point(1) - get_point(0)).magnitude();
    real v_length = (get_point(3) - get_point(0)).magnitude();
    M *= t.align_with_axis(C, normal, 'y');
    Point x_axis_pt(1);
    real angle = M.angle(x_axis_pt);
    M *= t.rotate(0, angle);
    if (M.unit_vector() \ne x_axis_pt) {
        cerr \ll "WARNING! Rectangle::in_ellipse() \n\n        "'\n        M.show("M: ");
        /* I'd rather output this to stderr, but I don't have a way to do this yet. [LD 2003.07.18. */
        cout \ll endl \ll flush;
    }
    Ellipse e(origin, h_length, v_length);
    e *= t.inverse();
    return e;
}


Log

[LD 2003.07.18.] Added this function.

\{ Define Rectangle functions 104 \} \equiv

Ellipse Rectangle::draw_out_ellipse(const Color &ddraw_color, string ddashed, string ppen, Picture &picture) const
{
    Ellipse e = out_ellipse();
    e.draw(ddraw_color, ddashed, ppen, picture);
    return e;
}

[LDF 2003.07.18.] Added this function.

```
{ Define Rectangle functions 104 } +≡
   Ellipse Rectangle::draw_in_ellipse(const Color &ddraw_color, string ddashed, string dpen, Picture &picture) const
   {
      Ellipse e = in_ellipse();
      e.draw(ddraw_color, ddashed, dpen, picture);
      return e;
   }
```

1271. Putting Ellipse together. This is what’s compiled.

```
{ Include files 6 }
{ Version control identifier 5 }
{ Define class Ellipse 1143 }
{ Define static Ellipse data members 1144 }
{ Define Ellipse functions 1147 }
{ Define Rectangle functions 1184 }
{ Declare non-member template functions for Ellipse 1154 }
```
1272. This is what's written to ellipses.h.

(ellipses.h 1272) ≡
(Define class Ellipse 1143)
(Declare non-member template functions for Ellipse 1154)

1273. Circle (circles.web). It won't be possible to make circles recede to the central vanishing point. !! Get quote from book!!

Log

[LDF 2003.11.12.] Removed the version control identifiers from the CWEB files for the distribution of 3DLDLF 1.1. They're still used in my development versions.
[LDF 2003.12.01.] Put the version control identifiers back into the release versions, because I've put them in their own RCS repository.

format Circle Shape

(Version control identifier 5) ≡
static string res_id = "$Id: circles.web,v1.8,2004/01/12,21:27:22,1f1msto1uExp,5$";

1274. Include files.

(Include files 6) ≡
#include "loader.h"
#include "pspglb.h"
#include "creatnew.h"
#include "io.h"
#include "colors.h"
#include "transform.h"
#include "shapes.h"
#include "pictures.h"
#include "points.h"
#include "lines.h"
#include "planes.h"
#include "paths.h"
#include "curves.h"
#include "polygons.h"
#include "rectangs.h"
#include "ellipses.h"

1275. Circle class definition.
1276.  
(Define class Circle 1276) \equiv  
class Circle : public Ellipse {  
    real radius;  
    public: (Declare Circle functions 1278)  
};  
This code is used in sections 1315 and 1316.

1277.  Constructors and setting functions.

1278.  Default constructor.  No arguments.  
(Declare Circle functions 1278) \equiv  
    Circle();  
See also sections 1281, 1283, 1290, 1292, 1295, 1297, 1298, 1300, 1302, and 1304.  
This code is used in section 1276.

1279.  
(Define Circle functions 1279) \equiv  
    Circle::Circle()  
    {  
        on_free_store = false;  
        line_switch = false;  
        cycle_switch = true;  
    }  
See also sections 1282, 1284, 1291, 1293, 1296, 1301, 1303, and 1305.  
This code is used in section 1315.

1280.  Center, diameters and angles.

1281.  Constructor,  
(Declare Circle functions 1278) +\equiv  
    Circle(const Point &ccenter, const real ddiameter, const real angle_x = 0, const real angle_y = 0,  
    const real angle_z = 0, const unsigned short number_of_points = DEFAULT_NUMBER_OF_POINTS);  

1282.  
(Define Circle functions 1279) +\equiv  
    Circle::Circle(const Point &ccenter, const real ddiameter, const real angle_x, const real  
    angle_y, const real angle_z, const unsigned short nnumber_of_points)  
    : radius(ddiameter/2) {  
        on_free_store = false;  
        line_switch = false;  
        cycle_switch = true;  
        center = ccenter;  
        center.apply_transform();  
        axis_h = axis_v = ddiameter;  
        number_of_points = nnumber_of_points;  
        Ellipse c(center, axis_h, axis_v, angle_x, angle_y, angle_z, number_of_points);  
        *this = c;  
    }
§1283  3DLDF-1.1.5.1

1283. Setting function.

\[ \text{Log} \]

[LDF 2003.05.05.] Added the argument \textit{number of points}. Without it, this setting function didn’t match the constructor above.

\[
\begin{align*}
\text{void set(const Point &center, const real diameter, const real angle_x = 0, const real angle_y = 0, } \\
&\quad \text{const real angle_x = 0, const unsigned short number_of_points = DEFAULT_NUMBER_OF_POINTS);}
\end{align*}
\]
1284.  
(Define Circle functions 1279 ) +≡

void Circle::set(const Point &center, const real diameter, const real angle_x, const real angle_y, const real angle_z, const unsigned short number_of_points)
{
    Circle c(center, diameter, angle_x, angle_y, angle_z, number_of_points);
    *this = c;
    return;
}

1285.  Pseudo-constructor for dynamic allocation.

1286.  Pointer argument.

(Declare non-member template functions for Circle 1286 ) +≡

Circle *create_new (const Circle *c);

See also section 1287.

This code is used in sections 1315 and 1316.

1287.  Reference argument.

(Declare non-member template functions for Circle 1286 ) +≡

Circle *create_new (const Circle &c);

1288.  Destructor.  [LDF 2002.10.09] Removed the destructor. Path:: ~Path() or Path:: clear() should be used instead, unless I add dynamically allocated data members to Circle (rather than Ellipse or Path).

1289.  Assignment.

1290.  Circle argument.  This function returns a reference to *this, which can be used for further assignment.

(LDF 2002.11.10] Changed and simplified this function. It now uses Ellipse::operator =().
(Declare Circle functions 1278) +≡
    Circle &operator=(const Circle &c);

1291.
(Define Circle functions 1279) +≡
    Circle &Circle::operator=(const Circle &c)
    {
        radius = c.radius;
        Ellipse::operator=(c);
        return *this;
    }

1292. Ellipse argument.  This function returns a reference to *this, which can be used for further assignment.

[Log]

[LDF 2002.11.10] Changed and simplified this function. It now uses Ellipse::operator=().
[LDF 2003.08.14] Added code for handling the case that e.axis_x and e.axis_y differ by a small amount, possible due to imprecision (see below).

(Declare Circle functions 1278) +≡
    Circle &operator=(const Ellipse &e);
1293. If \( e_{\text{axis},v} \neq e_{\text{axis},h} \), it’s quite possible that the difference is negligible, and the result of imprecision resulting from the representation of floating point numbers, or calculations performed on them. Therefore, we compare the absolute value of their difference with \( \text{Point :: epsilon()} \) instead of checking whether they're equal. [LDF 2003.08.14]

It’s also possible that one of them has an integer value, i.e., it has only zeroes following the decimal point, and the other deviates by a small amount. In this case, we want to set \( \text{radius} \) to half of the former, because it’s probably the correct value. So, if \( e_{\text{axis},v} \equiv \text{floor}(e_{\text{axis},v}) \), we set \( \text{radius} \) to \( e_{\text{axis},v}/2 \). Otherwise, we set \( \text{radius} \) to \( e_{\text{axis},h} \) without further ado. If \( e_{\text{axis},h} \equiv \text{floor}(e_{\text{axis},h}) \), so much the better, and if it isn’t, neither it nor \( e_{\text{axis},v} \) has an integer value, so it doesn’t matter which one we use to set \( \text{radius} \). [LDF 2003.08.14]

(Define Circle functions 1279) \( + \equiv \)
Circle &Circle::operator=(const Ellipse &e)
{
  real e_axis_v = e.getAxis_v();
  real e_axis_h = e.getAxis_h();
  if (e_axis_v != e_axis_h) {
    if (fabs(e_axis_v - e_axis_h) > Point::epsilon()) {
      cerr << "ERROR! In Circle::operator=Ellipse."
           << "Can’t perform assignment. Returning."
           << "Unable to determine axes.
           return *this;
    }
  } else if (e_axis_v == floor(e_axis_v))
  else if (e_axis_h == floor(e_axis_h))
  else
    radius = e.getAxis_v()/2.0;
  return *this;
}

1294. Returning elements and information.

1295. Is circular. Tests whether \*this is circular. Operations such as scale() and shear() can cause Circles to become non-circular. [LDF 2003.07.25]

isCircular() first tests whether \*this is planar using isPlanar(). If it’s not, false is returned. Otherwise, the Point \( p \) is set to \*points[0], and Point \( c = getCenter() \) is is subtracted from \( p \). \( p \text{mag} \text{magnitude}() \) is then stored in real \( \text{mag} \). [LDF 2003.07.25]

Then, \( p \) is set to each of the other Points on the Circle in turn, \( c \) is subtracted from \( p \), and \( p \text{mag} \text{magnitude}() \) is is stored in real \( \text{mag} \). If the absolute value of the difference \( \text{mag} - \text{mag} \) is greater than Point::epsilon(), isCircular() immediate returns false. If \( \text{fabs} (\text{mag} - \text{mag}) \leq \text{Point :: epsilon()} \) for all of the Points \*points[n] for \( n > 0 \), isCircular() returns true. [LDF 2003.07.25]

[Log]

[ Declare Circle functions 1278 ] \( + \equiv \)
bool isCircular(void) const;
1296. (Define Circle functions 1279) \[=\]
bool Circle::is_circular(void) const
{
    bool DEBUG = true;  /* false */
    if (DEBUG) cout << "Entering Circle::is_circular().\n";
    if (!is_planar()) {
        if (DEBUG)
            cerr << "In Circle::is_circular();\n" << "*this is non-planar. Returning false.\n" << endl << endl << flush;
        return false;
    }
    Point p;
    Point c = get_center();
    real mag0;
    real mag;
    vector(Point *)::const_iterator iter = points.begin();
    p = **iter++ - c;
    mag0 = p.magnitude();
    for (int i = 1; iter != points.end(); ++iter) {
        p = **iter - c;
        mag = p.magnitude();
        if (fabs(mag - mag0) > Point::epsilon()) {
            if (DEBUG) cerr << "Point\n" << i << " doesn't lie on Circle.\n" << "Exiting Circle::is_circular().\n" << "Returning false.\n" << flush;
            return false;
        }
        ++i;
    }
    if (DEBUG) cout << "Exiting Circle::is_circular(). Returning true.\n" << flush;
    return true;
}

1297. Get radius.

Log

[2002.05.10] Added this function.

(Declare Circle functions 1278) \[=\]
inline real get_radius()
{
    return radius;
}

1298. Get diameter. [2002.05.10] Added this function.
(Declare Circle functions 1278) \[=\]
inline real get_diameter()
{
    return (2 * radius);
}
1299. Intersections. Neither GCC nor the DEC compiler could resolve a call to `intersection_points()` with `Point` arguments to `Ellipse::intersection_points()`, after a `Circle` version with a `Circle` argument had been declared. TO DO: I didn’t think this would happen, so I should probably review the rules governing resolution of calls to functions on objects of derived classes. [LDF 2003.07.09]

Therefore, I’ve added `Circle` versions of this function, with `Point` and `Path` arguments, that simply call the `Ellipse` versions, and return their return values. This solves the problem. [LDF 2003.07.09]

The program executed correctly under Linux, after I recompiled with GCC. However, under Tru65, the program caused a “Memory fault” error. After I removed the object files, and recompiled (with the DEC compiler), the problem disappeared. [LDF 2003.07.09]

[LDF 2003.07.18.] TO DO: Add `Circle::intersection_points(const Ellipse &)` and `Ellipse::intersection_points(const Circle &)`.

1300. Point argument.

---

[Log]

---

(Declare Circle functions 1278) +≡

virtual bool_point_pair intersection_points(const Point &pt0, const Point &pt1) const;

1301.

(Define Circle functions 1279) +≡

bool_point_pair Circle::intersection_points(const Point &pt0, const Point &pt1) const
{
    return Ellipse::intersection_points(pt0, pt1);
}

1302. Path argument.

---

[Log]

---

(Declare Circle functions 1278) +≡

virtual bool_point_pair intersection_points(const Path &p) const;
§1303.  
(Define Circle functions 1279) \(\equiv\)

```cpp
bool_point_pair Circle::intersection_points(const Path &p) const
{
    return Ellipse::intersection_points(p);
}
```

1304.  Circle argument.

<table>
<thead>
<tr>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>[LDF 2003.07.20.] Wrote the definition of this function. Tested all cases. It should probably be tested more thoroughly.</td>
</tr>
<tr>
<td>[LDF 2003.08.14.] Made verbose argument non-const. Setting verbose to true if VERBOSE_GLOBAL is true. Added VERBOSE_GLOBAL to pspglea.web today.</td>
</tr>
<tr>
<td>[LDF 2003.08.27.] Removed the declaration real c.radius = c.radius, since c.radius was never used.</td>
</tr>
</tbody>
</table>

(Declare Circle functions 1278) \(\equiv\)

```cpp
virtual bool_point_quadruple intersection_points(const Circle &c, bool verbose = false) const;
```
1305. 
(Define Circle functions 1279 ) +

```cpp
bool_point_quadruple Circle::intersection_points(const Circle &c, bool verbose) const
{
    bool DEBUG = false;  /* true */
    if (VERBOSE_GLOBAL) verbose = true;
    if (DEBUG && verbose) cout << "****************************\n"
        "\nCircle::intersection_points(const Circle&)\n";

    bool_point_quadruple bpq = INVALID_BOOL_POINT_QUADRUPLE;
    Plane this_plane = get_plane();
    Plane c_plane = c.get_plane();
    if (!this_plane.normal || this_plane.normal && this_plane.normal || !c.plane.normal) {
        if (verbose && DEBUG) cout << "Circles are non-coplanar.\n"
            "\nCalling Circle::intersection_points()\n"
            "bpq = Ellipse::intersection_points(c, verbose);\n"
            if (DEBUG) cout << "Exiting Circle::intersection_points(const Circle&);" \n                "endl << endl << flush;
            return bpq;
    } else {
        if (verbose && DEBUG) cout << "Circles are non-coplanar.\n"
            real dist = (c.center - center).magnitude();
        if (DEBUG) cout << "dist=\n" << dist << endl << flush;
        if (dist == 0) {
            if (verbose && DEBUG) cout << "Circles have the same center.\n"
                if (radius == c.radius) {
                    if (verbose && DEBUG) cout << "Circles are congruent.\n"
                        "Returning INVALID_BOOL_POINT_QUADRUPLE.\n"
                        "\n";
                if (DEBUG) cout << "Exiting Circle::intersection_points(const Circle&)." \n                    "endl << endl << flush;
                return INVALID_BOOL_POINT_QUADRUPLE;
    } else if (radius > c.radius) {
        if (verbose && DEBUG) cout << "This circle is outside of each other.\n" << endl << "Returning INVALID_BOOL_POINT_QUADRUPLE.\n"
            "flush;
    } else if (radius < c.radius) {
        if (verbose && DEBUG) cout << "This circle is inside of each other.\n" << endl << "Returning INVALID_BOOL_POINT_QUADRUPLE.\n"
            "flush;
    } else if (radius == c.radius) {
        if (verbose && DEBUG) cout << "This circle is coincident.\n"
            "Returning INVALID_BOOL_POINT_QUADRUPLE.\n"
            "flush;
    } else if (radius < c.radius) {
        if (verbose && DEBUG) cout << "This circle is inside of each other.\n" << endl << "Returning INVALID_BOOL_POINT_QUADRUPLE.\n"
            "flush;
    } else if (radius > c.radius) {
        if (verbose && DEBUG) cout << "This circle is outside of each other.\n" << endl << "Returning INVALID_BOOL_POINT_QUADRUPLE.\n"
            "flush;
    } else if (radius == c.radius) {
        if (verbose && DEBUG) cout << "This circle is coincident.\n"
            "Returning INVALID_BOOL_POINT_QUADRUPLE.\n"
            "flush;
    } else if (radius < c.radius) {
        if (verbose && DEBUG) cout << "This circle is inside of each other.\n" << endl << "Returning INVALID_BOOL_POINT_QUADRUPLE.\n"
            "flush;
    } else if (radius > c.radius) {
        if (verbose && DEBUG) cout << "This circle is outside of each other.\n" << endl << "Returning INVALID_BOOL_POINT_QUADRUPLE.\n"
            "flush;
    } else if (radius == c.radius) {
        if (verbose && DEBUG) cout << "This circle is coincident.\n"
            "Returning INVALID_BOOL_POINT_QUADRUPLE.\n"
            "flush;
    }
```
CIRCLE Argument

§1305  3DLDF-1.15.1

} } 
else { 
  cerr << "ERROR! \nInu\n" << "|Circle::intersection_points(const Circle&)|" << endl << 
"This can't happen!" << "radius or \nc\radius have invalid values: \n" << 
"radius, \n" << radius << endl << "c\radius, \n" << c\radius << endl << 
"Returning INVALID_BOOL_POINT_QUADRUPLE.\n\n" << flush; 
if (DEBUG) cout << "\nExiting Circle::intersection_points(const Circle&)." << endl << 
endl << endl << flush; 
return INVALID_BOOL_POINT_QUADRUPLE; }
} /* if (dist % 0) */
else if (dist > (radius + c\radius)) { 
  if (verbose \ DEBUG) 
    cout << "this and c\radius lie outside of each other.\n" << 
"No intersections.\n" << 
"Returning INVALID_BOOL_POINT_QUADRUPLE.\n\n" << flush; 
if (DEBUG) cout << "\nExiting Circle::intersection_points(const Circle&)." << endl << 
endl << flush; 
return INVALID_BOOL_POINT_QUADRUPLE; }
else if (dist % (radius + c\radius)) { 
  if (verbose \ DEBUG) 
    cout << "this and c\radius have a tangent on the outside\n" << 
"(one\igestion\nointpoint)\n"; 
    bpq.first.pt.set(c\center - center); 
    bpq.first.pt.unitVector(true); 
    bpq.first.pt *= radius; 
    bpq.first.pt.shift(center); 
    bpq.first.b = true; 
if (DEBUG) cout << "\nExiting Circle::intersection_points(const Circle&)." << endl << 
endl << flush; 
return bpq; }
else if (dist % max(radius, c\radius) - min(radius, c\radius)) { 
  if (verbose \ DEBUG) 
    cout << "this and c\radius have a tangent on the inside\n" << 
"(one\igestion\nointpoint)\n"; 
if (radius > c\radius) { 
  if (verbose \ DEBUG) 
    cout << "c\radius lies outside this.\n"; 
    bpq.first.pt.set(c\center - center); 
} else { 
  if (verbose \ DEBUG) 
    cout << "this\lies\ninside c.\n"; 
    bpq.first.pt.set(center - c\center); 
} 
    bpq.first.pt.unitVector(true); 
    bpq.first.pt *= radius; 
    bpq.first.pt.shift(center); 
    bpq.first.b = true; 
if (DEBUG) cout << "\nExiting Circle::intersection_points(const Circle&)." << endl << 
endl << flush; 
return bpq; }
else if (dist < (radius + c\radius)) { 
  if (dist > max(radius, c\radius) - min(radius, c\radius)) {
if (verbose ∨ DEBUG) cout << "\n";
  real a = radius;
  real bb = c.radius * c.radius;
  real beta = 2 * atan(sqrt((bb - ((dist - a) * (dist - a)) / ((dist + a) * (dist + a)) - bb)));
  beta *= 180 / PI;
if (verbose ∨ DEBUG) cout << beta << endl << flush;

Point P(c.center - center);
P.unit_vector(true);
P *= radius;
P.shift(center);
Point normal = get_normal();
normal.shift(center);
if (DEBUG)
{
  P.dotlabel("op");
  P.show("p");
  normal.show("normal");
}
bpq.first.b = true;
bpq.first.pt = P;
bpq.first.pt.rotate(center, normal, beta);
if (DEBUG) bpq.first.pt.show("bpq.first.pt");
bpq.second.b = true;
bpq.second.pt = P;
bpq.second.pt.rotate(center, normal, -beta);
if (DEBUG) bpq.second.pt.show("bpq.second.pt");
if (DEBUG) cout << \nExitingCircle::intersection_points(const Circle&)." << 
  endl << endl << flush;
return bpq;
}
else if (radius > c.radius) 
{
  if (verbose ∨ DEBUG)
    cout << "Circle::intersection_points(const Circle&)." << 
      "Returning:INVALID_BOOL_POINT_QUADRUPLE." << 
    endl << endl << flush;
  return INVALID_BOOL_POINT_QUADRUPLE;
}
else 
{
  if (verbose ∨ DEBUG)
    cout << "Circle::intersection_points(const Circle&)." << 
      "Returning:INVALID_BOOL_POINT_QUADRUPLE." << 
    endl << endl << flush;
  return INVALID_BOOL_POINT_QUADRUPLE;
} /* else if (dist < (radius + c.radius)) */
else 
{
  cerr << "ERROR:Circle::intersection_points(const Circle&):" << 
    "This case isn’t accounted for." << 
  "Returning:INVALID_BOOL_POINT_QUADRUPLE." << endl << flush;
if (DEBUG) cout << \"\nExiting Circle::intersection_points(const Circle&).\" << endl << endl << flush;
    return INVALID_BOOL_POINT_QUADRUPLE;
}  /* else (Coplanar case). */

1306. **Reg_Polygon functions.** [LDF 2003.06.13] The functions in this section are declared in polygons.h. They must be defined here, because Circle is an incomplete type there.

[LDF 2003.06.13] Added this section.

1307. **Enclosed circle.**

[LDF 2003.06.13] Added this function.
[LDF 2003.12.09] Changed call to Point::mediate() below. It’s now a member function.

(Define **Reg_Polygon** functions 1071) \+=

Circle Reg_Polygon::in_circle() const
{
    Circle c;
    if (!is_planar()) {
        cerr << \"ERROR!\nReg_Polygon::in_circle():\n\" << \"Reg_Polygon\is\_non\_planar.\n\" << \
             \"Returning\_empty\_Circle.\n\" << flush;
        return c;
    }
    if (points.size() < 3) {
        cerr << \"ERROR!\nReg_Polygon::in_circle():\n\" << \
             \"Reg_Polygon\_has\_less\_than\_3\_Points.\n\" << \"Returning\_empty\_Circle.\n\" << flush;
        return c;
    }
    Point midPt = points[0].mediate(*points[1]);
    midPt -= center;
    real r = midPt.magnitude();
    c.set(origin, 2 * r);
    Point normal = get_normal();
    normal.shift(center);
    Transform t;
    t.align_with_axis(center, normal, ‘y’);
    c *= t.inverse();
    return c;
}

1308. **Draw enclosed circle.**
1309. Normal version.
(Define Reg_Polygon functions 1071) \( \implies \)

Circle Reg_Polygon::draw_in_circle(const Color &ddraw_color, const string &ddashed, const string &ppen, Picture &picture) const
{
  Circle c = in_circle();
  c.draw(ddraw_color, dashed, ppen, picture);
  return c;
}

1310. Picture argument first.
(Define Reg_Polygon functions 1071) \( \implies \)

Circle Reg_Polygon::draw_in_circle(Picture &picture, const Color &ddraw_color, const string &ddashed, const string &ppen) const
{
  return draw_in_circle(ddraw_color, dashed, ppen, picture);
}

1311. Surrounding circle.

Log

[Addition]


(Define Reg_Polygon functions 1071) \( \implies \)

Circle Reg_Polygon::out_circle() const
{
  Circle c;
  if (is_planar()) {
    cerr << "ERROR in Reg_Polygon::out_circle():\n" << "Reg_Polygon is non-planar.\n" << "Returning empty Circle.\n" << flush;
    return c;
  }
  if (points.size() < 3) {
    cerr << "ERROR in Reg_Polygon::out_circle():\n" << "Reg_Polygon has less than 3 Points.\n" << "Returning empty Circle.\n" << flush;
    return c;
  }
  Point normal = get_normal();
  normal.shift(center);
  c.set(origin, 2 * radius);
  Transform t;
  t.align_with_axis(center, normal, 'y');
  c *= t.invert();
  return c;
}

1312. Draw surrounding circle.

Log

[LD 2003.06.13.] Added this function.

```cpp
(Define Reg_Polygon functions 1071) +≡
Circle Reg_Polygon:: draw_out_circle(const Color &ddraw_color, const string ddashed, const string ppen, Picture &picture) const
{
    Circle c = out_circle();
    c.draw(ddraw_color, ddashed, ppen, picture);
    return c;
}
```

1314. Picture argument first.

Log

[LD 2003.06.13.] Added this function.

```cpp
(Define Reg_Polygon functions 1071) +≡
Circle Reg_Polygon:: draw_out_circle(Picture &picture, const Color &ddraw_color, const string ddashed, const string ppen) const
{
    return draw_out_circle(ddraw_color, ddashed, ppen, picture);
}
```

1315. Putting Circle together. This is what’s compiled.

{ Include files 6 }
{ Version control identifier 5 }
{ Define class Circle 1276 }
{ Define Circle functions 1279 }
{ Define Reg_Polygon functions 1071 }
{ Declare non-member template functions for Circle 1286 }
1316. This is what's written to circles.h,

#include "circles.h"

#include "loader.h"
#include "pspglb.h"
#include "io.h"
#include "colors.h"
#include "transform.h"
#include "shapes.h"
#include "pictures.h"
#include "points.h"
#include "lines.h"
#include "planes.h"
#include "paths.h"
#include "curves.h"
#include "polygons.h"
#include "rectangles.h"
#include "ellipses.h"
#include "circles.h"

1317. Patterns (patterns.web). [LDF 2002.09.21.] NOTE: When you add a new .web file and
move code to it by copying it from another .web file, remember to change the name of the header file
that it writes. Otherwise, this can cause problems and it's not obvious what's caused them.

Log

[LDF 2002.09.21.] Started using this file again. Moved hex_pattern_1 () here. Made the appropriate changes
to complspr.web and main.web.

[LDF 2003.11.12.] Removed the version control identifiers from the CWEB files for the distribution
of 3DLDF 1.1. They're still used in my development versions.

[LDF 2003.12.10.] Put the version control identifers back into my release versions for 3DLDF 1.1.4. I've
already put some of them back in, now I'm doing the rest of them. However, the release versions are now in
their own RCS repository.

#include files 6) +\=

static string rcs_id = "$Id: \$patterns.web, \$1.4:2004/01/12:21:31:19:0lfinsto1_lExp1$";

1318. Include files.

#include files 6) +\=

#include "loader.h"
#include "pspglb.h"
#include "io.h"
#include "colors.h"
#include "transform.h"
#include "shapes.h"
#include "pictures.h"
#include "points.h"
#include "lines.h"
#include "planes.h"
#include "paths.h"
#include "curves.h"
#include "polygons.h"
#include "rectangles.h"
#include "ellipses.h"
#include "circles.h"

1319. Plane tessellations.

1320. Hexagonal tessellation 1.

[LDF 2003.08.10.] TO DO: Change, so that it's possible to put the pattern into a rectangular area.

Log

[LDF 2002.09.22.] Added arguments. hex_pattern_1 () can now be used to make a pattern with up to three
nested hexagons which can be drawn and filled. Each hexagon has its own argument for the draw color, the
fill color and the pen to be used. hex_pattern_1 () does not have arguments for dash patterns, but they could
be added, if necessary. If the argument for the diameter of the middle or the inner hexagon is 0, then that
hexagon is not drawn or filled. In this case, the arguments for the draw and fill colors are ignored for that
hexagon, The default pen for the outer hexagon is thicker than the pens for the other two (.5mm and .3mm, respectively).

[LDF 2002-09-20] Rewrote this function. It now works with the new `project()` function. `hex_pattern_1()` makes a “honeycomb” pattern on the x-z plane using a single hexagon (i.e., there aren’t nested hexagons yet, as in the old version). In the next version, I plan to add arguments for optionally putting two smaller hexagons inside the large one, and for filling and unfilling.

This version contains arguments for the drawing command, including a `Picture` argument, so that `hex_pattern_1()` need not be put onto `current_picture`.

```c
(Declare Pattern functions 1320) ==
  unsigned int hex_pattern_1(real diameter_outer = 5, real diameter_middle = 0, real
diameter_inner = 0, unsigned short first_row = 5, unsigned short double_rows = 10, unsigned
short row_shift = 2,
  /* Arguments for the drawing and filling commands. */
Color draw_color_outer = *Colors::default_color,  /* Outer */
Color fill_color_outer = *Colors::background_color, Color draw_color_middle = *Colors::default_color,
/* Middle */
Color fill_color_middle = *Colors::background_color, Color draw_color_inner = *Colors::default_color,
/* Inner */
Color fill_color_inner = *Colors::background_color, string pen_outer = "pencircle_scaled_.5mm",
string pen_middle = "pencircle_scaled_.3mm", string pen_inner = "pencircle_scaled_.3mm",
  Picture &picture = current_picture, unsigned int max_hexagons = 1000);
```

See also sections 1323 and 1326.

This code is used in sections 1329 and 1330.
1321.

(Define Pattern functions 1321)

```c
unsigned int hex_pattern_1(real diameter_outer, real diameter_middle, real diameter_inner, unsigned short first_row, unsigned short double_rows, unsigned short row_shift,
    /* Arguments for the drawing and filling commands. */
    Color draw_color_outer, /* Outer */
    Color fill_color_outer, Color draw_color_middle, /* Middle */
    Color fill_color_middle, Color draw_color_inner, /* Inner */
    Color fill_color_inner, string pen_outer, string pen_middle, string pen_inner, Picture
        &picture, unsigned int max_hexagons)
{
    bool DEBUG = false; /* true */
    if (DEBUG) cout << "Entering hex_pattern_1().\n" << flush;
    if (first_row < 1) {
        cerr << "ERROR! In hex_pattern_1():\n" << "first_row has invalid value: \"" << first_row << "\n" << "It must be strictly positive.\n" << "Taking absolute value.\n" << flush;
        first_row = abs(first_row);
        cerr << "Now first_row = \"" << first_row << "\n" << endl << flush;
    }
    if (first_row > 25) /* [LDF 2002.09.22.] This can't be else if because the preceding condition might have produced a value > 25. */
    {
        cerr << "ERROR! In hex_pattern_1():\n" << "first_row has invalid value: \"" << first_row << "\n" << "It can be at most 25.\n" << "Setting first_row to 25.\n" << flush;
        first_row = 25;
    }
    if (fillColorOuter == Colors::default_color)
        /* LDF 2002.09.24. Changed this, because I'm now using class Color instead of strings. Now it looks like I'm going to have to use "Colors::background_color" as a placeholder. I'm going to have to check to see what the consequences of this change are here. !! [LDF 2002.09.22.] This is necessary, because Path::fill() interprets """" as "black". In hex_pattern_1(), it may be necessary to have a placeholder for a fillColor, and it's better to be able to use """" than to have to type "background". */
        fillColorOuter = Colors::background_color;
    if (fillColorMiddle == Colors::default_color) fillColorMiddle = Colors::background_color;
    if (fillColorInner == Colors::default_color) fillColorInner = Colors::background_color;
    bool doMiddle; /* [LDF 2002.09.22.] Having doMiddle and doInner is a convenience, so I don't have to check whether diameter_middle and diameter_inner are 0 below, which wouldn't be as clearly understandable. */
    bool doInner;
    doMiddle = (diameter_middle == 0) ? false : true;
    doInner = (diameter_inner == 0) ? false : true;
    Point pt0; /* origin. */
    Reg_Polygon pOuter(pt0, 6, diameterOuter);
    Reg_Polygon pMiddle;
    Reg_Polygon pInner; /* [LDF 2002.09.22.] The middle and inner hexagons are only set (and used) if doMiddle and/or doInner are true. */
    if (doMiddle) pMiddle.set(pt0, 6, diameter_middle);
    if (doInner) pInner.set(pt0, 6, diameter_inner);
    Reg_Polygon pOuterCopy; /* These Reg_Polygons are used for copying. */
```
Reg_Polygon p_middle_copy;
Reg_Polygon p_inner_copy;
#if 0
  if (DEBUG) p_outer.dotlabel();
#endif
real right_shift = p_outer.get_point(4).get_x() + p_outer.get_point(3).get_x() -
                 p_outer.get_point(1).get_x() - p_outer.get_point(2).get_x();
real left_shift = -right_shift;  /* Center the first row on the origin. */
Transform t;
t.shift((first_row/2) * left_shift);
if (first_row % 2 == 0) t.shift(.5 * right_shift);
p_outer == t;
if (do_middle) p_middle == t;
if (do_inner) p_inner == t;
Transform offset;
  /* [LDF 2002.09.22] offset is for moving the hexagons to the second row in the
double row, which is offset with respect to the first (and will contain one more set of nested hexagons). */
offset.shift(p_outer.get_point(4) - p_outer.get_point(2));
Transform move_back;
  /* [LDF 2002.09.22] move_back is for moving in the direction of the
positive z-axis before starting the next double row. */
move_back.shift(p_outer.get_point(5) - p_outer.get_point(3));
  /* [LDF 2002.09.22] The number of sets of hexagons in the rows differs, so we use i_min and i_max
for controlling the for loop that shifts and draws and fills the hexagons. */
signed short i_min = 0;
unsigned short i_max = first_row;
  /* [LDF 2002.09.20] I could just use first_row instead of declaring a new variable, but the name
i_max makes more sense in the loop, so I think it's worth doing for the sake of clarity. */
short i, j, k;
unsigned int hexagon_ctr = 0;  /* Each time a hexagon is drawn, hexagon_ctr is incremented.
hexagon_ctr is the return value of this function (hex_pattern_I()). */
for (k = 0; k < double_rows; ++k)  /* k is the number of double rows. [LDF 2002.09.22] This loop
takes care of moving back in the direction of the positive z-axis. */
  {
    /* [LDF 2002.09.20] If 0 is passed as the row_shift argument, don't do any shifting. Otherwise,
every row_shift rows, increase the number of hexagons in the rows by 2. The rows remain
centered around the z-axis. row_shift applies to the double rows. The offset row which is drawn
when j % 1 already has one hexagon more than the first row drawn, so when row_shift % 1, the
effect is that each single row is one hexagon longer than the last. This makes the edges recede
diagonally. */
    if (k != 0 && row_shift != 0 && k % row_shift == 0) {
      -i_min;
      ++i_max;
      t = p_outer.shift(left_shift);
      if (do_middle) p_middle == t;
      if (do_inner) p_inner == t;
    }
  for (j = 0; j < 2; ++j)  /* This loop makes the second line in each set of double lines. */
    {
      p_outer_copy = p_outer;
      if (do_middle) p_middle_copy = p_middle;
      if (do_inner) p_inner_copy = p_inner;
if (j == 1) {
    p_outer_copy *= offset;
    if (do_middle) p_middle_copy *= offset;
    if (do_inner) p_inner_copy *= offset;
    --i_min;
    t = p_outer_copy.shift(left_shift);
    if (do_middle) p_middle_copy *= t;
    if (do_inner) p_inner_copy *= t;
}
for (i = i_min; i < i_max; ++i)
/* [LDF 2002.09.22] This loop draws and/or fills the horizontal rows. */
{
    if (fill_color_outer == *Colors::background_color)
        p_outer_copy.draw(draw_color_outer,"",pen_outer,picture);
    else if (draw_color_outer == fill_color_outer) p_outer_copy.fill(fill_color_outer,picture);
    else p_outer_copy.filldraw(draw_color_outer,fill_color_outer,"",pen_outer,picture);
    if (do_middle) {
        if (fill_color_middle == fill_color_outer)
            p_middle_copy.draw(draw_color_middle,"",pen_middle,picture);
        else if (draw_color_middle == fill_color_middle) p_middle_copy.fill(fill_color_middle,picture);
        else p_middle_copy.filldraw(draw_color_middle,fill_color_middle,"",pen_middle,picture);
    }
    if (do_inner) {
        if (fill_color_inner == fill_color_middle)
            p_inner_copy.draw(draw_color_inner,"",pen_inner,picture);
        else if (draw_color_inner == fill_color_middle) p_inner_copy.fill(fill_color_inner,picture);
        else p_inner_copy.filldraw(draw_color_inner,fill_color_inner,"",pen_inner,picture);
    }
    ++hexagon_ctr;
    if (hexagon_ctr >= max_hexagons) {
        cerr << "ERROR! In hex_pattern_1():\n" << "Too many sets of hexagons:\n" << hexagon_ctr << ". Returning.\n" << endl << flush;
        return hexagon_ctr;
    }
    t = p_outer_copy.shift(right_shift);
    if (do_middle) p_middle_copy *= t;
    if (do_inner) p_inner_copy *= t;
}
p_outer *= move_back;
if (do_middle) p_middle *= move_back;
if (do_inner) p_inner *= move_back;
++i_min;
}
if (DEBUG) cout << "Exiting hex_pattern_1().\n" << flush;
return hexagon_ctr;

See also sections 1324 and 1327.
This code is used in section 1329.

1322. patterns.
1323. **Epicycloid pattern 1.** [LDF 2003.02.11.] This function works well for outer Circles with radii that are divisors (with no remainder) of the radius of the inner Circle. Each outer Circle is rolled around the inner Circle once only. If the radius of the outer Circle is a divisor of the inner Circle, the end of the epicycloid will meet up with the beginning. If the radius of the outer Circle is not a divisor of the inner Circle, it won't. See epicycloid_pattern_3() below, for a pattern that works well for outer Circles, whose radii are not divisors of the radius of the inner Circle. // START HERE. TO DO: Start from beginning of Color * vector, if I get to the end.

---

Log

[LDF 2003.02.09.] Added this function.
[LDF 2003.08.27.] Removed the declaration const Color * curr_color, since curr_color was never used.

(Declare Pattern functions 1320) +

```cpp
unsigned int epicycloid_pattern_1 (real diameter_inner,
    real diameter_outer_start, real diameter_outer_end, real step, unsigned int offsets, vector (const Color *) colors = Colors::default_color_vector, int arc_divisions = 72);
```
1324.

(Define Pattern functions 1321) +≡

```c
unsigned int epicycloid_pattern_1(real diameter_inner, real diameter_outer_start, real
diameter_outer_end, real step, unsigned int offsets, vector(const Color *) colors, int
arc_divisions)
{
    bool DEBUG = false;  /* true */
    if (diameter_inner < diameter_outer_start) {
        cerr << "WARNING! diameter inner < diameter outer!\n" <<
            "This is likely to lead to strange results.\n" << "Continuing.\n\n" << flush;
    }
    using namespace Colors;
    unsigned int spiral_counter = 0;
    real radius_outer;
    real phi;
    Path spiral;
    spiral += "..";
    real radius_inner = diameter_inner / 2;
    Circle inner_circle(origin, diameter_inner);
    inner_circle.draw();
    if (DEBUG) inner_circle.get_center().dotlabel("inner_circle" ,"rt");
    Circle outer_circle;
    Point outer_circle_center;
    Point normal;
    Point p0;
    real theta = 360.0 / arc_divisions;
    Circle temp_circle;
    Point p2;
    Point temp_circle_center;
    Point temp_circle_normal;
    if (offsets < 1) {
        cerr << "WARNING! offsets has invalid value: \n" << offsets << endl <<
            "offsets must be >0. Setting to 1.\n\n" << flush;
        offsets = 1;
    }
    real diameter_outer;
    vector(const Color *):iterator iter = colors.begin();
    for (diameter_outer = diameter_outer_start; diameter_outer ≥ diameter_outer_end;
        diameter_outer += step) {
        if (iter != colors.end() - 1) ++iter;
        radius_outer = diameter_outer / 2;
        outer_circle_center.set(0, 0, radius_inner + radius_outer);
        normal = outer_circle_center;
        normal.shift(0, 1);
        p0.set(0, 0, radius_inner + diameter_outer);
        outer_circle.set(outer_circle_center, diameter_outer);
        if (colors.size() == 0) colors.push_back(default_color);
        for (unsigned int i = 0; i < offsets; ++i) {
            if (i ≠ 0) outer_circle_center += normal += p0 += outer_circle.rotate(0, 360.0 / offsets);
            else {
```
if (DEBUG) {
    p0.dotlabel("p0");
    outer_circle.draw(blue);
}
if (DEBUG) {
    p0.dotlabel("p0","bot");
    outer_circle.draw(green,"evenly");
}
spiral += p0;
phi = theta * radius_inner/radius_outer;
temp_circle = outer_circle;
temp_circle.center = outer_circle.center;
temp_circle.normal = normal;
p2 = p0;
for (int j = 1; j <= arc_divisions; ++j) {
    p2 = temp_circle.normal *= temp_circle.rotate(0, theta);
    p2.rotate(temp_circle.center, temp_circle.normal, phi);
    spiral += p2;
    if (DEBUG) {
        p2.dotlabel("p2","left");
        temp_circle.draw(black,"evenly");
    }
    spiral.draw(***iter);
    ++spiral.counter;
    spiral.clear();
    spiral += ".";
}
return spiral.counter;

1325. **Epicycloid pattern 2.** [LDF 2003.02.11.] This pattern should be like `epicycloid_pattern_1()`, except that the offsets are not made by rotating the outer Circle around the center of the inner Circle, but by rotating the Point used for tracing the epicycloid about the center of the outer Circle.

1326. **Epicycloid pattern 3.** [LDF 2003.02.11.] This function works well for outer Circles with radii that are not even divisors of the radius of the inner Circle.

---

[LDF 2003.02.11.] Added this function.
[LDF 2003.08.27.] Removed the declaration `real radius_ratio = radius_outer/radius_inner`, since `radius_ratio` was never used.

(Declare Pattern functions 1320) +≡

```cpp
unsigned int epicycloid_pattern_3(real diameter_inner, real diameter_outer, vector<const Color *> colors = Colors::default_color_vector, unsigned int limit = 100, int arc_divisions = 72);
```
1327.

(Define Pattern functions 1321 ) +
unsigned int epicyclic_pattern_3(real diameter_inner, real diameter_outer, vector(const Color *)
          colors, unsigned int limit, int arc_divisions)
{
    using namespace Colors;
    bool DEBUG = false; /* true */
    vector(const Color *)::iterator color_iter = colors.begin();
    real radius_outer = diameter_outer/2;
    real radius_inner = diameter_inner/2;
    real theta = 360.0/arc_divisions;
    real phi = theta * radius_inner/radius_outer;
    real theta_total = 0;
    Circle inner_circle(origin, diameter_inner);
    inner_circle.draw();
    Point outer_circle_center(0,0, radius_inner + radius_outer);
    Circle outer_circle(outer_circle_center, diameter_outer);
    outer_circle.draw();
    Point normal(outer_circle_center);
    normal.shift(0,1);
    Point p0(0,0, radius_inner + diameter_outer);
    p0.dotlabel("p0");
    Path spiral;
    spiral += "...";
    spiral += p0;
    Point start_pt(p0);
    unsigned int spiral_counter = 1;
    unsigned int iter_ctr = 0;
    while (true) {
        if (theta_total > 360) {
            cout << "theta_total==" << theta_total << endl << "Reducing theta_total to 0" << endl << flush;
            theta_total -= 360;
            ++iter_ctr;
            spiral.draw(**color_iter++);
            spiral.clear();
            spiral += ". . .";
            if (color_iter == colors.end()) color_iter = colors.begin();
        }
        if (iter_ctr > limit) {
            cout << "Exceeded limit.. iter_ctr==" << iter_ctr << endl << "Breaking." << endl << flush;
            break;
        }
        else if (iter_ctr > 0 && (fmod((iter_ctr * radius_outer), radius_inner) == 0)) {
            cout << "Same out, even.." << iter_ctr << endl << "fmod(iter ctr * radius outer), radius inner) == 0" << fmod((iter_ctr * radius_outer),
                    radius_inner) << endl << "Breaking.." << flush;
            break;
        }
    }
outer\_circle\_center **= normal **= p0 **= outer\_circle\_rotate(0, theta);
p0 . rotate(outer\_circle\_center , normal , phi);
theta\_total += theta;
if (DEBUG) {
    outer\_circle . draw(black,"evenly");
p0 . dotlabel("p0");
}
spiral += p0;
++spiral\_counter;
}
return spiral\_counter;

1328. Putting patterns together.

1329. This is what’s compiled.
    (Include files 6)
    (Version control identifier 5)
    (Declare Pattern functions 1320)
    (Define Pattern functions 1321)
1330. This is what's written to patterns.h.

(declare Pattern functions 1320)

1331. Solid (solids.web).  [LDF 2002.11.12.] TO DO: Add get_center(). Must set center in the Polyhedra before this is useful.

Log

[LDF 2002.09.29.] Created this file.
[LDF 2003.11.12.] Removed the version control identifiers from the CWEB files for the distribution of 3DLDF 1.1. They're still used in my development versions.
[LDF 2003.12.10.] Put the version control identifiers back into my release versions for 3DLDF 1.1.4. I've already put some of them back in, now I'm doing the rest of them. However, the release versions are now in their own RCS repository.

(Version control identifier 5) +≡

static string res_id = "$Id: solids.web,v \1.5,2004/01/12,21:33:23,lnfsto\1.\Exp,\$";

1332. Include files.

(include files 6) +≡

#include "loader.h"
#include "pspglb.h"
#include "creatnew.h"
#include "io.h"
#include "colors.h"
#include "transform.h"
#include "shapes.h"
#include "pictures.h"
#include "points.h"
#include "lines.h"
#include "planes.h"
#include "paths.h"
#include "curves.h"
#include "polygons.h"
#include "rectangs.h"
#include "ellipses.h"
#include "circles.h"

1333. Solid class definition.

Log

[LDF 2002.09.30.] Added the data members circles, ellipses, polygons, and paths. On the one hand, this is wasteful, since most if not all Solids will contain only one kind of Path; on the other hand, it's an advantage to be able to have the drawing and filling functions be members of Solid, since they don't have to know what kind of a Path a Path is, in order to draw or fill it. This way, I don't have to define the drawing and filling functions for Sphere, Ellipsoid, Polyhedron, etc.

[LDF 2002.10.01.] Added the data member projective_extremes.
[LDF 2003.04.11.] Added the static const data members PATH, CIRCLE, ELLIPSE, REG_POLYGON, and RECTANGLE. Currently, their only use is as arguments to get_shape_ptr() and get_shape_center().

[LDF 2003.04.11.] Renamed polygons to reg_polygons. This is in case I decide to make it possible to have irregular polygons. In this case, I may define a class Polygon and derived Reg_Polygon from it.
§1333 3DLDF-1.1.5.1

Define class Solid 1333 \(\equiv\)

```
class Solid : public Shape {
    protected: bool on_free_store;
    Point center;
    bool do_output; /* LDF 2002.10.01. Added */
    vector(Circle *) circles;
    vector(Ellipse *) ellipses;
    vector(Path *) paths;
    vector(Rectangle *) rectangles;
    vector(Reg_Polygon *) reg_polygons;
    valarray<real> projective_extremes;

    public: static const unsigned short CIRCLE;
    static const unsigned short ELLIPSE;
    static const unsigned short PATH;
    static const unsigned short RECTANGLE;
    static const unsigned short REG_POLYGON;

    (Declare Solid functions 1336)
};
```

This code is used in sections 1441 and 1442.

1334. Define static const Solid data members.

```
[LD 2003.04.11.] Added this section.
```

```
(Define static const Solid data members 1334) \(\equiv\)
const unsigned short Solid::CIRCLE = 0;
const unsigned short Solid::ELLIPSE = 1;
const unsigned short Solid::PATH = 2;
const unsigned short Solid::RECTANGLE = 3;
const unsigned short Solid::REG_POLYGON = 4;
```

This code is used in section 1441.

1335. Constructors.

1336. Default constructor. (No arguments.) [LDF 2002.10.02] Solid will not normally be used in user code, since it is intended to be a base class only. Therefore, objects of type Solid will not normally be declared as automatic variables and there will be no static global Solids. However, create_new < Solid > () is used in the drawing and filling functions, in order to put Solids onto Pictures. If we didn’t have a constructor, projective_extremes wouldn’t initially have the right size and on_free_store and do_output would both be false (assuming the compiler set s the initial values of uninitialized bools to false). None of this would really matter, because presumably an assignment would follow immediately, which would take care of everything, but there’s no harm in making sure.

(Declare Solid functions 1336) \(\equiv\)

```
Solid();
```

See also sections 1338, 1343, 1345, 1348, 1350, 1353, 1356, 1358, 1360, 1362, 1364, 1366, 1369, 1371, 1373, 1375, 1377, 1379, 1381, 1383, 1386, 1388, 1390, 1392, 1395, 1397, 1399, 1401, 1404, 1406, 1408, 1409, 1411, 1413, 1415, 1417, 1419, 1423, 1426, 1429, 1432, 1435, and 1438.

This code is used in section 1333.
1337. (Define **Solid** functions 1337) \( \equiv \)

```
Solid::Solid()
{
  on_free_store = false;
  do_output = true;
  projective_extremes.resize(6, 0);
}
```

See also sections 1339, 1344, 1346, 1347, 1349, 1351, 1354, 1357, 1359, 1361, 1363, 1365, 1367, 1370, 1372, 1374, 1376, 1378, 1380, 1382, 1384, 1387, 1389, 1391, 1393, 1396, 1398, 1400, 1402, 1405, 1407, 1410, 1412, 1414, 1416, 1418, 1420, 1424, 1427, 1430, 1433, 1436, and 1439.

This code is used in section 1441.

1338. **Copy constructor.** [LDF 2002.10.02] (Declare **Solid** functions 1336) \( \equiv \)

```
Solid(const Solid &s);
```
1339.  
(Define Solid functions 1337) +≡
Solid::Solid(const Solid &s){ on_free_store = false;
do_output = true;
projective_extremes.resize(6,0); for (vector(Path *)::const_iterator iter = s.paths.begin();
iter != s.paths.end(); ++iter) { paths.push_back ( create_new < Path > (0) );
*(paths.back ()) = **iter; } for (vector(Circle *)::const_iterator iter = s.circles.begin();
iter != s.circles.end(); ++iter) { circles.push_back ( create_new < Circle > (0) );
*(circles.back ()) = **iter; } for (vector(Ellipse *)::const_iterator iter = s.ellipses.begin();
iter != s.ellipses.end(); ++iter) { ellipses.push_back ( create_new < Ellipse > (0) );
*(ellipses.back ()) = **iter; } for (vector(Reg_Polygon
*)::const_iterator iter = s.reg_polygons.begin(); iter != s.reg_polygons.end();
++iter) { reg_polygons.push_back ( create_new < Reg_Polygon > (0) );
*(reg_polygons.back ()) = **iter; } for (vector(Rectangle
*)::const_iterator iter = s.rectangles.begin(); iter != s.rectangles.end(); ++iter) {
rectangles.push_back ( create_new < Rectangle > (0) );
*(rectangles.back ()) = **iter; }
}

1340.  Pseudo-constructor for dynamic allocation.

1341.  Pointer argument.

Log

[LDN 2003.12.30.] Replaced Solid::create_new<solid>() with a specialization of template<class C> C*create_new() for Solid. The argument is now const.

(Declare non-member template functions for Solid 1341) +≡
Solid *create_new(const Solid *sc);
See also section 1342.
This code is used in sections 1441 and 1442.

1342.  Reference argument.

Log


(Declare non-member template functions for Solid 1341) +≡
Solid *create_new(const Solid &sc);

1343.  Destructor.

Log

[LDN 2003.08.27.] Added a virtual destructor with an empty definition, because GCC with the "-Wall" option issued the following warning: "class Solid has virtual functions but non-virtual destructor".

(Declare Solid functions 1336) +≡
virtual ~Solid();
1344.

(Define Solid functions 1337 ) +≡
  Solid:: ~Solid ()
  }

1345. Assignment.

(Declare Solid functions 1336 ) +≡
  virtual const Solid operator = ( const & s );

1346.

(Define Solid &Solid:: operator = (const Solid & s) ) +≡
  const Solid &Solid:: operator = (const Solid & s) { bool DEBUG = false; /* true */
    if (DEBUG) cout ≣ "Entering Solid : operator = (). \n";
    if (this ≡ & s) /* Make sure it's not self-assignment. */
      return *this;
    center = s . center; /* LDF 2002.10.06. Added this line, because center is now a member of
    Solid, */
    /* [LDF 2002.10.02.] First, call the destructor on all of the elements of paths, circles, ellipses, reg_polygons, and rectangles, because they've been allocated dynamically. Then clear out the vectors. */
    if (paths . size () > 0) {
      for (vector(Path *):: iterator = paths . begin () ; iter ≠ paths . end () ; ++ iter) (** iter) . clear ();
      paths . clear ();
    }
    if (circles . size () > 0) {
      for (vector (Circle *):: iterator = circles . begin () ; iter ≠ circles . end () ; ++ iter)
        (** iter) . clear ();
      circles . clear ();
    }
    if (ellipses . size () > 0) {
      for (vector(Ellipse *):: iterator = ellipses . begin () ; iter ≠ ellipses . end () ; ++ iter)
        (** iter) . clear ();
      ellipses . clear ();
    }
    if (reg_polygons . size () > 0) {
      for (vector (Reg_Polygon *):: iterator = reg_polygons . begin () ; iter ≠ reg_polygons . end () ;
        ++ iter) (** iter) . clear ();
      reg_polygons . clear ();
    }
    if (rectangles . size () > 0) {
      for (vector (Rectangle *):: iterator = rectangles . begin () ; iter ≠ rectangles . end () ; ++ iter)
        (** iter) . clear ();
      rectangles . clear ();
    }
1347. Now, create new Path, Circle, Ellipse, Reg_Polygon, and Rectangle pointers, allocate memory for them, assign values to the objects they point to from s, and push them onto the appropriate vectors. [LDF 2002.10.02.]

(Define Solid functions 1337) \(+\)

Path \(*p\) for (vector(Path *)::const_iterator iter = s.paths.begin(); iter \(!=\) s.paths.end(); ++iter) {
    p = create_new < Path > (0);
    paths.push_back (p);
    *(paths.back()) = **iter; } for (vector(Circle *)::const_iterator iter = s.circles.begin();
    iter \(!=\) s.circles.end(); ++iter) { circles.push_back ( create_new < Circle > (0) );
    *(circles.back()) = **iter; } for (vector(Ellipse *)::const_iterator iter = s.ellipses.begin();
    iter \(!=\) s.ellipses.end(); ++iter) { ellipses.push_back ( create_new < Ellipse > (0) );
    *(ellipses.back()) = **iter; } for (vector(Reg_Polygon *)::const_iterator iter = s.reg.polygons.begin();
    iter \(!=\) s.reg.polygons.end(); ++iter) { reg.polygons.push_back ( create_new < Reg_Polygon > (0) );
    *(reg.polygons.back()) = **iter; } for (vector(Rectangle *)::const_iterator iter = s.rectangles.begin();
    iter \(!=\) s.rectangles.end(); ++iter) { rectangles.push_back ( create_new < Rectangle > (0) );
    *(rectangles.back()) = **iter; } projective_extremes = 0; /* For output. */
do_output = true;
if (DEBUG) {
    cout \ll "paths.size() \(=\) \n" \ll paths.size() \ll endl \ll flush;
    cout \ll "circles.size() \(=\) \n" \ll circles.size() \ll endl \ll flush;
    cout \ll "ellipses.size() \(=\) \n" \ll ellipses.size() \ll endl \ll flush;
    cout \ll "reg.polygons.size() \(=\) \n" \ll reg.polygons.size() \ll endl \ll flush;
    cout \ll "rectangles.size() \(=\) \n" \ll rectangles.size() \ll endl \ll flush;
}
if (DEBUG) cout \ll "Exiting::Solid::operator=().\n"
return \*this; }

1348. Copying.

(Log [LDF 2003.05.06.] BUG FIX: Changed s from Shape * to Solid *. I noticed this bug when I tried to copy a Picture containing a Cuboid, and the copy contained a single empty Shape * on shapes.)

(Declare Solid functions 1336) \(+\)

virtual Shape *get_copy () const;

1349.

(Define Solid functions 1337) \(+\)

Shape *Solid::get_copy () const { Solid *s = create_new < Solid > (0);
    *s = \*this;
    return dynamic_cast<Shape >(*s); }

1350. Set on free store.

(Log [LDF 2004.01.06.] Made non-inline.)

(Declare Solid functions 1336) \(+\)

virtual bool set_on_free_store (bool b = true);
1351.  
(Define Solid functions 1337) +≡
    bool Solid::set_on_free_store(bool b)
    {
        on_free_store = b;
        return b;
    }

1352. Returning elements and information, [LDF 2003.04.11] The functions get_shape_ptr(), get_circle_ptr(), get_ellipse_ptr(), get_path_ptr(), get_rectangle_ptr(), and get_reg_polygon_ptr() all return const pointers to Shape, Circle, Ellipse, etc. Therefore, they must be invoked in such a way, that the const qualifier is not discarded. For example, following Dodecahedron d(origin, 5); two ways of invoking get_reg_polygon_ptr() are: const Reg_Polygon *ptr = d.get_reg_polygon_ptr(5); and

Reg_Polygon A = *d.get_reg_polygon_ptr(5);

[LDF 2003.06.09] Changed the names of get_shape(), get_circle(), get_ellipse(), get_path(), get_rectangle(), and get_reg_polygon() to get_shape_ptr(), get_circle_ptr(), get_ellipse_ptr(), get_path_ptr(), get_rectangle_ptr(), and get_reg_polygon_ptr(). The names without “_ptr” were confusing, because they didn’t make clear that the functions returned pointers.

1353. Get center.

[LDF 2003.05.06] Added this function,
[LDF 2003.08.10] Made this function const.

(Declare Solid functions 1336) +≡
    virtual const Point &get_center() const;

1354.
(Define Solid functions 1337) +≡
    const Point &Solid::get_center() const
    {
        return center;
    }

1355. Getting Shapes.

[LDF 2003.04.30] Changed the functions get_circle_ptr(), get_ellipse_ptr(), get_path_ptr(), get_rectangle_ptr(), and get_reg_polygon_ptr(). They no longer use get_shape_ptr(). There’s no good reason for casting pointers from one type to another. I rather doubt that get_shape_ptr() is needed, anyway.
Get Shape pointer.  [LDF 2003.05.30.] This function copies one of the objects on one of the vectors of Shape * belonging to the Solid, and returns a pointer to Shape that points to the copy. Currently, a Solid contains the vectors circles, ellipses, paths, rectangles, and reg_polygons. The argument shape_type indicates which vector should be accessed. Normally, the corresponding public static const data members CIRCLE, ELLIPSE, PATH, RECTANGLE, or REG_POLYGON should be passed as the shape_type argument, e.g., Circle *c_ptr = static_cast(Circle *>(get_shape_ptr(Solid::CIRCLE, 3)).

[LDF 2003.04.30.] This function was mainly intended for use in the functions get_circle_ptr(), get_ellipse_ptr(), etc., and was not intended for use in user code. I now doubt whether this function is needed at all, especially since it is no longer used in the functions mentioned above.

Log

[LDF 2003.04.11.] Added this function.
[LDF 2003.04.30.] Now using getCopy() instead of static_cast(const Shape *>(). The way it was caused compilation errors under Tru64 (DEC ALPHA).
[LDF 2003.05.30.] Changed return value to Shape * from const Shape *. The way it was before caused “Memory fault” errors at run-time.

(Declare Solid functions 1336) +≡

virtual Shape *get_shape_ptr(const unsigned short shape_type, const unsigned short s) const;

§1356  3DLDF-1.1.5.1  GET SHAPE POINTER  401
1357.

(Define Solid functions 1337) +≡
Shape *Solid::get_shape_ptr(const unsigned short shape_type, const unsigned short s) const
{
  bool DEBUG = false; /* true */
  if (DEBUG) {
    cout << "Entering*Solid::get_shape_ptr().\n" << flush;
  }
  if (shape_type == CIRCLE) {
    if (s < circles.size()) {
      return circles[s]->get_copy();
    } else {
      cerr << "ERROR!*In*Solid::get_shape_ptr():\n" "s" "\n" circles.size() "\n" circles.size() "\n" Returning*null*pointer(0).\n" << flush;
      return static_cast<Shape*>(0);
    }
  } else if (shape_type == ELLIPSE) {
    if (s < ellipses.size()) {
      return ellipses[s]->get_copy();
    } else {
      cerr << "ERROR!*In*Solid::get_shape_ptr():\n" "s" "\n" ellipses.size() "\n" ellipses.size() "\n" Returning*null*pointer(0).\n" << flush;
      return static_cast<Shape*>(0);
    }
  } else if (shape_type == PATH) {
    if (s < paths.size()) {
      return paths[s]->get_copy();
    } else {
      cerr << "ERROR!*In*Solid::get_shape_ptr():\n" "s" "\n" paths.size() "\n" paths.size() "\n" Returning*null*pointer(0).\n" << flush;
      return static_cast<Shape*>(0);
    }
  } else if (shape_type == RECTANGLE) {
    if (s < rectangles.size()) {
      return rectangles[s]->get_copy();
    } else {
      cerr << "ERROR!*In*Solid::get_shape_ptr():\n" "s" "\n" rectangles.size() "\n" rectangles.size() "\n" Returning*null*pointer(0).\n" << flush;
      return static_cast<Shape*>(0);
    }
  } else if (shape_type == REG_POLYGON) {
    if (s < reg_polygons.size()) {
      return reg_polygons[s]->get_copy();
    } else {
      cerr << "ERROR!*In*Solid::get_shape_ptr():\n" "s" "\n" reg_polygons.size() "\n" reg_polygons.size() "\n" Returning*null*pointer(0).\n" << flush;
      return static_cast<Shape*>(0);
    }
  }
}


return reg_polygons.size() > s ? return circles[s];
else {
  return static_cast(const Circle *)(0);
}

1358. Get Circle pointer.

{Declare Solid functions 1336} +≡

  virtual const Circle *get_circle_ptr(const unsigned short s) const;

1359.

{Define Solid functions 1337} +≡

  const Circle *Solid::get_circle_ptr(const unsigned short s) const
  {
    if (circles.size() > s) {
      return circles[s];
    } else {
      return static_cast(const Circle *)(0);
    }
  }

1360. Get Ellipse pointer.

{Declare Solid functions 1336} +≡

  virtual const Ellipse *get_ellipse_ptr(const unsigned short s) const;
1361. Define Solid functions 1337 ) +≡
   const Ellipse *Solid::get_ellipse_ptr(const unsigned short s) const
   {
     if (ellipses.size() > s) {
       return ellipses[s];
     }
     else {
       return static_cast<const Ellipse*>(0);
     }
   }

1362. Get Path pointer.

[Log]
LDF 2003.04.11.] Added this function.
LDF 2003.04.30. Changed this function, so that it no longer uses get_shape_ptr().

(Declare Solid functions 1336 ) +≡
   virtual const Path *get_path_ptr(const unsigned short s) const;

1363. Define Solid functions 1337 ) +≡
   const Path *Solid::get_path_ptr(const unsigned short s) const
   {
     if (paths.size() > s) {
       return paths[s];
     }
     else {
       return static_cast<const Path*>(0);
     }
   }

1364. Get Rectangle pointer.

[Log]
LDF 2003.04.11.] Added this function.
LDF 2003.04.30. Changed this function, so that it no longer uses get_shape_ptr().

(Declare Solid functions 1336 ) +≡
   virtual const Rectangle *get_rectangle_ptr(const unsigned short s) const;
1365.  
(Define Solid functions 1337) \(\equiv\)  
\[
\begin{align*}
\text{const Rectangle } \&\text{Solid::get_rectangle_ptr}(\text{const unsigned short } s) \text{ const} \\
\{} \\
\text{\quad if } (\text{rectangles.size()} > s) \{} \\
\text{\quad \quad return rectangles[s];} \\
\text{\quad } \} \\
\text{\quad else } \{} \\
\text{\quad \quad return \text{static_cast(Rectangle\*)}(0);} \\
\text{\quad } \} \\
\end{align*}
\]

1366. Get Reg_Polygon pointer.  

Log  

[LDF 2003.04.11.] Added this function.  
[LDF 2003.04.30.] Changed this function, so that it no longer uses \text{get_shape_ptr}().

(Declare Solid functions 1336) \(\equiv\)  
\[
\begin{align*}
\text{virtual const Reg_Polygon } \&\text{Solid::get_reg_polygon_ptr}(\text{const unsigned short } s) \text{ const;}
\end{align*}
\]

1367.  
(Define Solid functions 1337) \(\equiv\)  
\[
\begin{align*}
\text{const Reg_Polygon } \&\text{Solid::get_reg_polygon_ptr}(\text{const unsigned short } s) \text{ const} \\
\{} \\
\text{\quad if } (\text{reg_polygons.size()} > s) \{} \\
\text{\quad \quad return reg_polygons[s];} \\
\text{\quad } \} \\
\text{\quad else } \{} \\
\text{\quad \quad return \text{static_cast(Reg_Polygon\*)}(0);} \\
\text{\quad } \} \\
\end{align*}
\]

1368. Getting Shape centers.  

[LDF 2003.04.30.] TO DO: I think it might be possible to code the functions in this section more succinctly.  

Log  

[LDF 2003.04.11.] Added this section.
1369. **Get Shape center.** This function returns the center of the Circle, Ellipse, Rectangle, or Reg_Polygon number s in circles, ellipses, rectangles, or reg_polygons, respectively. If s is larger than `{vector}. size()`, an error message is issued and INVALID_POINT is returned.

One of the following **public static const** data members of Solid can (and probably should) be used as the `shape_type` argument: CIRCLE, ELLIPSE, RECTANGLE, and REG_POLYGON.

!! Note that this function will have to be changed, if new vectors of Shape pointers are added to class Solid!

[Declare Solid functions 1336] +≡

```
virtual const Point &get_shape_center(const unsigned short shape_type, const unsigned short s)
  const;
```
1370.

(Define Solid functions 1337) +≡

const Point &Solid::get_shape_center(const unsigned short shape_type, const unsigned short s)
const
{
    if (shape_type == CIRCLE) {
        if (s < circles.size()) return circles[s].get_center();
        else {
            cerr << "ERROR! In Solid::get_shape_center():\n" << "s(" << s << ")" >> circles.size()(" << circles.size() << ")\nReturning INVALID_POINT. \n\n" << flush;
            return INVALID_POINT;
        }
    }
    else if (shape_type == ELLIPSE) {
        if (s < ellipses.size()) return ellipses[s].get_center();
        else {
            cerr << "ERROR! In Solid::get_shape_center():\n" << "s(" << s << ")" >> ellipses.size()(" << ellipses.size() << ")\nReturning INVALID_POINT. \n\n" << flush;
            return INVALID_POINT;
        }
    }
    else if (shape_type == RECTANGLE) {
        if (s < rectangles.size()) return rectangles[s].get_center();
        else {
            cerr << "ERROR! In Solid::get_shape_center():\n" << "s(" << s << ")" >> rectangles.size()(" << rectangles.size() << ")\nReturning INVALID_POINT. \n\n" << flush;
            return INVALID_POINT;
        }
    }
    else if (shape_type == REG_POLYGON) {
        if (s < reg_polygons.size()) return reg_polygons[s].get_center();
        else {
            cerr << "ERROR! In Solid::get_shape_center():\n" << "s(" << s << ")" >> reg_polygons.size()(" << reg_polygons.size() << ")\nReturning INVALID_POINT. \n\n" << flush;
            return INVALID_POINT;
        }
    }
    else {
        cerr << "ERROR! In Solid::get_shape_center():\n" << "Invalid_argument for shape_type: \n" << shape_type << endl << "Returning INVALID_POINT. \n\n" << flush;
        return INVALID_POINT;
    }
}
1371. Get Circle center.

[LDF 2003.04.11.] Added this function.

(Declare Solid functions 1336 ) \(\equiv\)
  virtual const Point &get\_circle\_center( const unsigned short s ) const;

1372.
(Define Solid functions 1337 ) \(\equiv\)
  const Point &Solid::get\_circle\_center( const unsigned short s ) const
  {
    return get\_shape\_center( CIRCLE, s );
  }

1373. Get Ellipse center.

[LDF 2003.04.11.] Added this function.

(Declare Solid functions 1336 ) \(\equiv\)
  virtual const Point &get\_ellipse\_center( const unsigned short s ) const;

1374.
(Define Solid functions 1337 ) \(\equiv\)
  const Point &Solid::get\_ellipse\_center( const unsigned short s ) const
  {
    return get\_shape\_center( ELLIPSE, s );
  }

1375. Get Rectangle center.

[LDF 2003.04.11.] Added this function.

(Declare Solid functions 1336 ) \(\equiv\)
  virtual const Point &get\_rectangle\_center( const unsigned short s ) const;

1376.
(Define Solid functions 1337 ) \(\equiv\)
  const Point &Solid::get\_rectangle\_center( const unsigned short s ) const
  {
    return get\_shape\_center( RECTANGLE, s );
  }
1377. Get Reg_Polygon center.

[Log] Added this function.

(Declare Solid functions 1336) +=
  virtual const Point &get_reg_polygon_center(const unsigned short s) const;

1378. (Declare Solid functions 1337) +=
  const Point &Solid::get_reg_polygon_center(const unsigned short s) const
  {
    return get_shape_center(REG_POLYGON, s);
  }

1379. Is on free store.

(Declare Solid functions 1336) +=
  virtual bool is_on_free_store() const;

1380. (Declare Solid functions 1337) +=
  bool Solid::is_on_free_store() const
  {
    bool b = true;
    return b;
  }

1381. Show.

(Declare Solid functions 1336) +=
  virtual void show(string text = "", char coords = 'w', const bool do_persp = true, const bool do_apply = true, Focus *f = 0, const unsigned short proj = Projections::PERSP, const real factor = 1) const;
1382.

(Define Solid functions 1337 ) +

    void Solid::show (string text, char coords, const bool do_persp, const bool do_apply, Focus
        + f, const unsigned short proj, const real factor) const
    {
        if (text == "") text = "Solid: ";
        cout << text << endl;
        cout << "on_free_store = " << on_free_store << endl << flush;
        stringstream g;
        int i;
        if (paths.size() > 0) {
            cout << "Showing paths. \n";
            i = 0;
            for (vector<Path *>::const_iterator iter = paths.begin(); iter != paths.end(); ++iter) {
                g << "Path ", i++, " :: "
                (**iter).show (g, str(), coords, do_persp, do_apply, f, proj, factor);
                g.str(" ");
            }
        } else cout << "paths is empty. \n";
        if (circles.size() > 0) {
            cout << "Showing circles. \n";
            i = 0;
            for (vector<Circle *>::const_iterator iter = circles.begin(); iter != circles.end(); ++iter) {
                g << "Circle ", i++, " :: "
                (**iter).show (g, str(), coords, do_persp, do_apply, f, proj, factor);
                g.str(" ");
            }
        } else cout << "circles is empty. \n";
        if (ellipses.size() > 0) {
            cout << "Showing ellipses. \n"
            i = 0;
            for (vector<Ellipse *>::const_iterator iter = ellipses.begin(); iter != ellipses.end(); ++iter) {
                g << "Ellipse ", i++, " :: "
                (**iter).show (g, str(), coords, do_persp, do_apply, f, proj, factor);
                g.str(" ");
            }
        } else cout << "ellipses is empty. \n";
        if (reg_polygons.size() > 0) {
            cout << "Showing reg_polygons. \n"
            i = 0;
            for (vector<Reg_Polygon *>::const_iterator iter = reg_polygons.begin(); iter != reg_polygons.end();
                ++iter) {
                g << "Polygon ", i++, " :: "
                (**iter).show (g, str(), coords, do_persp, do_apply, f, proj, factor);
                g.str(" ");
            }
        } else cout << "reg_polygons is empty. \n";
if (rectangles.size() > 0) {
    cout << "Showing rectangles:\n";
    i = 0;
    for (vector(Rectangle *>::const_iterator iter = rectangles.begin(); iter != rectangles.end(); ++iter) {
        g << "Rectangle\n" << i++ << ":\n";
        (**iter).show(g.str(), coords, do_persp, do_apply, f, proj, factor);
        g.str("\n");
    }
    else cout << "rectangles is empty.\n";
    cout << endl << flush;
    return;
}

1383. Clear. [LDF 2002 10.07.] Replaced dummy definition with a real one. Now, clear() is called for all of the objects in the Solid.

Declare Solid functions 1336 \equiv virtual void clear();

1384.

(Define Solid functions 1337) \equiv

void Solid::clear()
{
    bool DEBUG = false; /* true */
    if (DEBUG) cout << "Entering Solid::clear().\n" << flush;
    for (vector(Path *>::iterator iter = paths.begin(); iter != paths.end(); ++iter) (**iter).clear();
    paths.clear();
    for (vector(Circle *>::iterator iter = circles.begin(); iter != circles.end(); ++iter) (**iter).clear();
    circles.clear();
    for (vector(Ellipse *>::iterator iter = ellipses.begin(); iter != ellipses.end(); ++iter)
    (**iter).clear();
    ellipses.clear();
    for (vector(Reg_Polygon *>::iterator iter = reg_polygons.begin(); iter != reg_polygons.end();
            ++iter) (**iter).clear();
    reg_polygons.clear();
    for (vector(Rectangle *>::iterator iter = rectangles.begin(); iter != rectangles.end(); ++iter)
    (**iter).clear();
    rectangles.clear();
    if (DEBUG) cout << "Exiting Solid::clear().\n" << flush;
    return;
}

1385. Transformations.

1386. Multiplying by a Transform.

Declare Solid functions 1336 \equiv

    virtual Transform operator\*=(const Transform &t);
1387.
(Define Solid functions 1337) +≡
Transform Solid::operator*=(const Transform &t)
{
center *= t;
for (vector<Path *>::iterator iter = paths.begin(); iter ≠ paths.end(); ++iter) **iter *= t;
for (vector<Ellipse *>::iterator iter = ellipses.begin(); iter ≠ ellipses.end(); ++iter) **iter *= t;
for (vector<Circle *>::iterator iter = circles.begin(); iter ≠ circles.end(); ++iter) **iter *= t;
for (vector<Reg_Polygon *>::iterator iter = reg_polygons.begin(); iter ≠ reg_polygons.end();
    ++iter) **iter *= t;
for (vector<Rectangle *>::iterator iter = rectangles.begin(); iter ≠ rectangles.end(); ++iter)
    **iter *= t;
return t;
}

1388. Applying a transformation.

Log

[LDF 2003.01.05.] Added this function. It’s now needed because I’ve made apply_transform() a pure
virtual function in class Shape. BUG FIX: I’ve done this in an attempt to fix a bug in Picture::output(),
where the Points on a Path were not transformed when I used “Transform t; current_picture *= t”.

(Declare Solid functions 1336) +≡

virtual void apply_transform(void);

1389.
(Define Solid functions 1337) +≡

void Solid::apply_transform(void)
{
center.apply_transform();
for (vector<Path *>::iterator iter = paths.begin(); iter ≠ paths.end(); ++iter)
    (**iter).apply_transform();
for (vector<Ellipse *>::iterator iter = ellipses.begin(); iter ≠ ellipses.end(); ++iter)
    (**iter).apply_transform();
for (vector<Circle *>::iterator iter = circles.begin(); iter ≠ circles.end(); ++iter)
    (**iter).apply_transform();
for (vector<Reg_Polygon *>::iterator iter = reg_polygons.begin(); iter ≠ reg_polygons.end();
    ++iter) (**iter).apply_transform();
for (vector<Rectangle *>::iterator iter = rectangles.begin(); iter ≠ rectangles.end(); ++iter)
    (**iter).apply_transform();
}

1390. Scale.
(Declare Solid functions 1336) +≡

virtual Transform scale(real xx, real yy = 0, real zz = 0);
§1391. 3DLDF-1.1.5.1

1391.  
\{ Define Solid functions 1337 \} +≡  
Transform Solid:: scale(real \( xx \), real \( yy \), real \( zz \))  
  \{  
    Transform \( t \);  
    \( t.\text{scale}(xx, yy, zz) \);  
    \( \ast \text{this} = \ast t \);  
    \text{return} \( t \);  
  \}  

1392.  Shear.  
\{ Declare Solid functions 1336 \} +≡  
virtual Transform shear(real \( xy \), real \( xx = 0 \), real \( yx = 0 \), real \( yz = 0 \), real \( zz = 0 \), real \( yz = 0 \), real \( zy = 0 \));  

1393.  
\{ Define Solid functions 1337 \} +≡  
Transform Solid:: shear(real \( xy \), real \( xx \), real \( yz \), real \( yz \), real \( zz \), real \( zz \), real \( zy \))  
  \{  
    Transform \( t \);  
    \( t.\text{shear}(xy, xx, yz, yz, zz, zz, zy) \);  
    \( \ast \text{this} = \ast t \);  
    \text{return} \( t \);  
  \}  

1394.  Shift.  

1395.  real arguments.  
\{ Declare Solid functions 1336 \} +≡  
virtual Transform shift(real \( xx \), real \( yy = 0 \), real \( zz = 0 \));  

1396.  
\{ Define Solid functions 1337 \} +≡  
Transform Solid:: shift(real \( xx \), real \( yy \), real \( zz \))  
  \{  
    Transform \( t \);  
    \( t.\text{shift}(xx, yy, zz) \);  
    \( \ast \text{this} = \ast t \);  
    \text{return} \( t \);  
  \}  

1397.  Point argument.  
\{ Declare Solid functions 1336 \} +≡  
virtual Transform shift(const Point \&pt);
1398. 
〈Define Solid functions 1337〉 +≡
  Transform Solid ::= shift (const Point &pt)
  
  Transform t;
  t.shift (pt);
  *this *= t;
  return t;
  
1399. Rotatation around the main axes.
〈Declare Solid functions 1336〉 +≡
  virtual Transform rotate (const real xx, const real yy = 0, const real zz = 0);

1400. 
〈Define Solid functions 1337〉 +≡
  Transform Solid ::= rotate (const real xx, const real yy, const real zz)
  
  Transform t;
  t.rotate (xx, yy, zz);
  *this *= t;
  return t;

1401. Rotatation around an arbitrary axis.

---

[LDF 2003.06.02] Changed name of this function from rotate around () to rotate (). This function now overloads rotate () with three real arguments.

〈Declare Solid functions 1336〉 +≡
  virtual Transform rotate (const Point &pθ, const Point &p1, const real angle = 180);

1402. 
〈Define Solid functions 1337〉 +≡
  Transform Solid ::= rotate (const Point &pθ, const Point &p1, const real angle)
  
  Transform t;
  t.rotate (pθ, p1, angle);
  *this *= t;
  return t;

1403. Outputting.

1404. Extract.
〈Declare Solid functions 1336〉 +≡
  virtual vector (Shape*) extract (const Focus &f, const unsigned short proj, real factor);
1405.

(Define Solid functions 1337 ) \( \equiv \)

\[ \text{vector(Shape *) Solid::extract(const Focus &f, const unsigned short proj, real factor)} \]

\{ 
\text{vector(Shape *) v;}
\text{for (vector(Path *)::iterator iter = paths.begin(); iter \neq paths.end(); ++iter) \{}
\text{(**iter).apply_transform();}
\text{if (-(**iter).project(f,proj,factor))} \}
\text{cerr << "WARNING! In Solid::extract():\n" << "Path cannot be projected.\n" << 
"Returning empty vector<Shape>.*\n" << flush;}
\text{return v;}
\text{break;}
\}
\text{for (vector(Ellipse *)::iterator iter = ellipses.begin(); iter \neq ellipses.end(); ++iter) \{}
\text{(**iter).apply_transform();}
\text{if (-(**iter).project(f,proj,factor))} \}
\text{cerr << "WARNING! In Solid::extract():\n" << "Ellipse cannot be projected.\n" << 
"Returning empty vector<Shape>.*\n" << flush;}
\text{return v;}
\text{break;}
\}
\text{for (vector(Circle *)::iterator iter = circles.begin(); iter \neq circles.end(); ++iter) \{}
\text{(**iter).apply_transform();}
\text{if (-(**iter).project(f,proj,factor))} \}
\text{cerr << "WARNING! In Solid::extract():\n" << "Circle cannot be projected.\n" << 
"Returning empty vector<Shape>.*\n" << flush;}
\text{return v;}
\text{break;}
\}
\text{for (vector(Reg_Polygon *)::iterator iter = reg_polygons.begin(); iter \neq reg_polygons.end(); ++iter) \{}
\text{(**iter).apply_transform();}
\text{if (-(**iter).project(f,proj,factor))} \}
\text{cerr << "WARNING! In Solid::extract():\n" << "Polygon cannot be projected.\n" << 
"Returning empty vector<Shape>.*\n" << flush;}
\text{return v;}
\text{break;}
\}
\text{for (vector(Rectangle *)::iterator iter = rectangles.begin(); iter \neq rectangles.end(); ++iter) \{}
\text{(**iter).apply_transform();}
\text{if (-(**iter).project(f,proj,factor))} \}
\text{cerr << "WARNING! In Solid::extract():\n" << "Rectangle cannot be projected.\n" << 
"Returning empty vector<Shape>.*\n" << flush;}
\text{return v;}
\text{break;}
\}
\text{v.push_back(this);} \]
return v;
}

1406. Set extremes.
(Declare Solid functions 1336) +E
virtual bool set_extremes();
1407.

(Define Solid functions 1337 )

```cpp
bool Solid::set_extremes()
{
  bool DEBUG = false; /* true */
  if (DEBUG) cout << "Entering Solid::set_extremes() \n flush;"
  valarray<real> v;
  v.resize(6, 0); /* LDF 2002.12.13. Added. Needed for compiling under GNU/Linux using GCC on
the Intel i686 computer gwd101.gwdg.de. */
  for (vector<Path>::iterator iter = paths.begin(); iter != paths.end(); ++iter) {
    if (!(*iter).set_extremes()) {
      cerr << "ERROR! In Solid::set_extremes(): \n " << "Path: set_extremes() returned false. \n"
      << "Returning false. \n" << flush;
      return false;
    }
    v = (**iter).get_extremes();
    for (int i = 0; i < 3; ++i) /* Minima. */
    {
      projective_extremes[i] = min(projective_extremes[i], v[i]);
    }
    for (int i = 3; i < 6; ++i) /* Maxima. */
    {
      projective_extremes[i] = max(projective_extremes[i], v[i]);
    }
  }
  for (vector<Ellipse>::iterator iter = ellipses.begin(); iter != ellipses.end(); ++iter) {
    if (!(*iter).set_extremes()) {
      cerr << "ERROR! In Solid::set_extremes(): \n " << "Path: set_extremes() returned false. \n"
      << "Returning false. \n" << flush;
      return false;
    }
    v = (**iter).get_extremes();
    for (int i = 0; i < 3; ++i) /* Minima. */
    {
      projective_extremes[i] = min(projective_extremes[i], v[i]);
    }
    for (int i = 3; i < 6; ++i) /* Maxima. */
    {
      projective_extremes[i] = max(projective_extremes[i], v[i]);
    }
  }
  for (vector<Circle>::iterator iter = circles.begin(); iter != circles.end(); ++iter) {
    if (!(*iter).set_extremes()) {
      cerr << "ERROR! In Solid::set_extremes(): \n " << "Path: set_extremes() returned false. \n"
      << "Returning false. \n" << flush;
      return false;
    }
    v = (**iter).get_extremes();
    for (int i = 0; i < 3; ++i) /* Minima. */
    {
      projective_extremes[i] = min(projective_extremes[i], v[i]);
    }
  }
  return true;
}
```
for (int i = 3; i < 6; ++i) /* Maxima. */
{
    projective_extremes[i] = max(projective_extremes[i], v[i]);
}

for (vector<Reg_Polygon *>::iterator iter = reg_polygons.begin(); iter != reg_polygons.end(); ++iter) {
    if (iter->set_extremes()) {
        cerr << "ERROR!In
Solid::set_extremes():\n""Path: \nSolid::set_extremes() returned false."
<< "Returning false.\n" << flush;
        return false;
    }
    v = (**iter).get_extremes();
    for (int i = 0; i < 3; ++i) /* Minima. */
    {
        projective_extremes[i] = min(projective_extremes[i], v[i]);
    }
    for (int i = 3; i < 6; ++i) /* Maxima. */
    {
        projective_extremes[i] = max(projective_extremes[i], v[i]);
    }
}

for (vector<Rectangle *>::iterator iter = rectangles.begin(); iter != rectangles.end(); ++iter) {
    if (iter->set_extremes()) {
        cerr << "ERROR!In
Solid::set_extremes():\n""Path: \nSolid::set_extremes() returned false."
<< "Returning false.\n" << flush;
        return false;
    }
    v = (**iter).get_extremes();
    for (int i = 0; i < 3; ++i) /* Minima. */
    {
        projective_extremes[i] = min(projective_extremes[i], v[i]);
    }
    for (int i = 3; i < 6; ++i) /* Maxima. */
    {
        projective_extremes[i] = max(projective_extremes[i], v[i]);
    }
}

if (DEBUG) cout << "Exiting \nSolid::set_extremes()" << endl << flush;
return true;

1408. Get extremes.
(Declare Solid functions 1336) +
inline virtual const valarray<real> get_extremes() const
{
    return projective_extremes;
}

1409. Get minimum z.
(Declare Solid functions 1336) +

virtual real get_minimum_z() const;

1410.
⟨Define Solid functions 1337 ⟩ +≡
real Solid::get_minimum_z() const
{
  bool DEBUG = false; /* true */
  if (DEBUG) {
    cout << "EnteringSolid::get_minimum_z()" << endl << flush;
    cout << "ExitingSolid::get_minimum_z()" << endl << flush;
  }
  return projective_extremes[4];
}

1411. Get maximum z.
⟨Declare Solid functions 1336 ⟩ +≡
virtual real get_maximum_z() const;

1412.
⟨Define Solid functions 1337 ⟩ +≡
real Solid::get_maximum_z() const
{
  bool DEBUG = false; /* true */
  if (DEBUG) {
    cout << "EnteringSolid::get_maximum_z()" << endl << flush;
    cout << "ExitingSolid::get_maximum_z()" << endl << flush;
  }
  return projective_extremes[5];
}

1413. Get mean z. [LDF 2003.06.16] Added this function.
⟨Declare Solid functions 1336 ⟩ +≡
virtual real get_mean_z() const;

1414.
⟨Define Solid functions 1337 ⟩ +≡
real Solid::get_mean_z() const
{
  return (projective_extremes[4] + projective_extremes[5])/2;
}

1415. Suppress output.
⟨Declare Solid functions 1336 ⟩ +≡
virtual void suppress_output();
1416. (Define Solid functions 1337 ) +≡
    void Solid::suppress_output()
    {
        do_output = false;
        return;
    }

1417. Unsuppress output.
(Declare Solid functions 1336 ) +≡
    virtual void unsuppress_output();

1418. (Define Solid functions 1337 ) +≡
    void Solid::unsuppress_output()
    {
        do_output = true;
        return;
    }

1419. Output. [LDF 2002.10.02.] In Picture::output(), shapes is sorted according to the values in projective_extremes for each Shape. However, it’s possible (and even likely) that the individual Paths in a Solid are not ordered in such a way that they will be output in the correct order. Therefore, I declare a vector (Shape *) s and put the Paths from paths, circles, ellipses, reg_polygons, and rectangles onto it. Then I sort s and call output() for each Shape. Currently, output() will resolve to Path::output(), because output() hasn’t been overloaded for Circle, Ellipse, Reg_Polygon, or Rectangle (and probably won’t be).

The invocation of push_back() in each of the four loops depends on the fact that Path::extract() returns a vector containing only one element. That’s why I use front(). There is no operator or function for concatenating vectors, at least I couldn’t find one in Stroustrup. TO DO: Get reference!
(Declare Solid functions 1336 ) +≡
    virtual void output();
1420. Define Solid functions

```cpp
void Solid::output()
{
    bool DEBUG = false;    /* true */
    if (DEBUG) cout << "Entering Solid::output.\n";  
    vector<Shape *> s;
    for (vector<Path *>::iterator iter = paths.begin(); iter != paths.end(); ++iter)
        s.push_back((**iter).get_copy());
    for (vector<Circle *>::iterator iter = circles.begin(); iter != circles.end(); ++iter)
        s.push_back((**iter).get_copy());
    for (vector<Ellipse *>::iterator iter = ellipses.begin(); iter != ellipses.end(); ++iter)
        s.push_back((**iter).get_copy());
    for (vector<Reg_Polygon *>::iterator iter = reg_polygons.begin(); iter != reg_polygons.end(); ++iter)
        s.push_back((**iter).get_copy());
    sort(s.begin(), s.end(), Compare_maximum_z());
    for (vector<Shape *>::iterator iter = s.begin(); iter != s.end(); ++iter) {
        (**iter).output();
        delete (*iter);
    }
    if (DEBUG) cout << "Exiting Solid::output.\n";
}
```

1421. Drawing and filling.
1422. Process vectors for `draw()`. [LDF 2002.10.09.] Added this section. The same things are done to each of the Shape * vectors paths, ellipses, circles, mg_polygons, and rectangles, so I've put the code in this named section. Each time it it's used, iter is an iterator for a different vector.

```cpp
{Process vectors for `draw()` 1422) ==
    if (c_iter != v.end()) {
        color_ptr = c_iter++;
        (**iter).set_fill.draw_value(DRAW); // LDF 2002.10.09. Added code for handling draw_color. */
        if (DEBUG) {
            cout << "color_ptr->get_use_name() = " << color_ptr->get_use_name() << endl << flush;
        }
        if (color_ptr->get_use_name() == false) {
            if (DEBUG) cout << "Allocating memory for Color.\n" << flush;
            Color *c = create_new < Color>(0);
            *c = *color_ptr;
            (**iter).set_draw_color(c); }
        else {
            if (DEBUG) cout << "color_ptr->get_name() = " << color_ptr->get_name() << endl << flush;
            (**iter).set_draw_color(color_ptr);
        }
    (**iter).set_fill_color(static_cast<Color*>(0));
    (**iter).set_dash_pattern(ddashed);
    (**iter).set_pen(ppen); }
This code is used in section 1424.
```

1423. Draw.

```
(Declare Solid functions 1336) ==
  virtual void draw(const vector< const Color * > v = Colors::default_color_vector, const string ddashed = "", const string ppen = "", Picture &picture = current_picture) const;
```
§1424.

(Define Solid functions 1337 ) +

```cpp
void Solid::draw(const vector<const Color*>& v, const string ddashed, const string ppen, Picture &picture) const { bool DEBUG = false;  /* true */
    if (DEBUG) cout << "Entering Solid::draw():" << "\n" << flush;
    Solid *s = create_new < Solid > (0);
    *s = *this;

    const Color *color_ptr = Colors::default_color;
    vector<const Color *>::const_iterator c_iter = v.begin();
    for (vector<Path *>::const_iterator iter = s->paths.begin(); iter != s->paths.end(); ++iter) {
        (Process vectors for draw( ) 1422)
    }
    c_iter = v.begin();
    for (vector<Circle *>::const_iterator iter = s->circles.begin(); iter != s->circles.end(); ++iter) {
        (Process vectors for draw( ) 1422)
    }
    c_iter = v.begin();
    for (vector<Ellipse *>::const_iterator iter = s->ellipses.begin(); iter != s->ellipses.end(); ++iter) {
        (Process vectors for draw( ) 1422)
    }
    for (vector<Reg_Polygon *>::const_iterator iter = s->reg_polygons.begin();
         iter != s->reg_polygons.end(); ++iter) {
        (Process vectors for draw( ) 1422)
    }
    for (vector<Rectangle *>::const_iterator iter = s->rectangles.begin(); iter != s->rectangles.end();
         ++iter) {
        (Process vectors for draw( ) 1422)
    }

    picture += dynamic_cast<Shape*>(s);
    if (DEBUG) {
        cout << "Exiting Solid::draw():" << "\n" << flush;
    }
```

1425. Process vectors for `fill()`. [LDF 2002.10.09.] Added this section. The same things are done to each of the `Shape` * vectors `paths`, `ellipses`, `circles`, `ng_polygons`, and `rectangles`, so I've put the code in this named section. Each time it's used, `iter` is an iterator for a different vector.

```
{Process vectors for `fill()` 1425} =
{
    if (c_iter != v.end()) {
        color_ptr = *c_iter++;
    }
    (**iter).setFillDrawValue(FILL);
    /* LDF 2002.10.09. Added code for handling `draw_color` and `fill_color`. */
    if (DEBUG) {
        cout << "color_ptr->get_use_name() = " << color_ptr->get_use_name() << endl << flush;
    }
    if (color_ptr->get_use_name() == false) {
        if (DEBUG) cout << "Allocating memory for Color. \n" << flush;
        Color *c = create_new < Color > (0);
        *c = *color_ptr;
        (**iter).setFillColor(c);
    } else {
        if (DEBUG) cout << "color_ptr->get_name() = " << color_ptr->get_name() << endl << flush;
        (**iter).setFillColor(color_ptr);
    }
    (**iter).setDrawColor(static_cast<Color*>(0));
    (**iter).setDashPattern("\n");
    (**iter).setPen("\n"); }
```

This code is used in section 1427.

1426. `Fill`. 

```
{Declare `Solid` functions 1336} =
    virtual void `fill`(const vector<const Color*>& v = Colors::default_color_vector, Picture &picture = current_picture) const;
```

[LDF 2003.08.10.] Removed pen argument, since filling doesn't use a pen.
1427.

(Define Solid functions 1337 ) +

```cpp
void Solid::fill(const vector<const Color*> v, Picture &picture) const { bool DEBUG = false;
    /* true */
    if (DEBUG) cout << "Entering Solid::fill()" << endl << flush;
    Solid *s = create_new < Solid > (0);
    *s = *this;
    const Color *color_ptr = Colors::default_color;
    vector<const Color*> const_iterator e_iter = v.begin();
    for (vector<Path *>::const_iterator iter = s->paths.begin(); iter != s->paths.end(); ++iter) {
        (Process vectors for fill() 1425)
    }
    e_iter = v.begin();
    for (vector<Circle *>::const_iterator iter = s->circles.begin(); iter != s->circles.end(); ++iter) {
        (Process vectors for fill() 1425)
    }
    e_iter = v.begin();
    for (vector<Ellipse *>::const_iterator iter = s->ellipses.begin(); iter != s->ellipses.end(); ++iter) {
        (Process vectors for fill() 1425)
    }
    e_iter = v.begin();
    for (vector<Reg_Polygon *>::const_iterator iter = s->regular_polygons.begin();
        iter != s->regular_polygons.end(); ++iter) {
        (Process vectors for fill() 1425)
    }
    e_iter = v.begin();
    for (vector<Rectangle *>::const_iterator iter = s->rectangles.begin(); iter != s->rectangles.end();
        ++iter) {
        (Process vectors for fill() 1425)
    }
    picture += dynamic_cast<Shape*>(s);
    if (DEBUG) {
        cout << "Exiting Solid::fill(): " << endl << flush;
    }
}
```
1428. Process vectors for `filldraw()`. [LDF 2002.10.09.] Added this section. The same things are done to each of the Shape * vectors `paths`, `ellipses`, `circles`, `reg_polygons`, and `rectangles`, so I've put the code in this named section. Each time it's used, `iter` is an iterator for a different vector.

```cpp
(Process vectors for filldraw () 1428) ≡
{
    if (draw_color_iter != draw_colors.end()) {
        draw_color_ptr = &draw_color_iter++;
    }
    if (fill_color_iter != fill_colors.end()) {
        fill_color_ptr = &fill_color_iter++;
    }
    (**iter).set_fill_draw_value(FILLDRAW);
    if (DEBUG) {
      cout ≡ "draw_color_ptr->get_use_name() ≡ " draw_color_ptr->get_use_name() ≡ endl ≡ flush;
    }
    if (draw_color_ptr->get_use_name() ≡ false) {
      if (DEBUG) cout ≡ "Allocating memory for Color.\n" ≡ flush;
      Color *c = create_new < Color > (0);
      *c = &draw_color_ptr;
      (**iter).set_draw_color(c);
    } else {
      if (DEBUG) {
        cout ≡ "draw_color_ptr->get_name() ≡ " draw_color_ptr->get_name() ≡ endl ≡ flush;
        (**iter).set_draw_color(draw_color_ptr);
      }
    }
    if (DEBUG) {
      cout ≡ "fill_color_ptr->get_use_name() ≡ " fill_color_ptr->get_use_name() ≡ endl ≡ flush;
    }
    if (fill_color_ptr->get_use_name() ≡ false) {
      if (DEBUG) cout ≡ "Allocating memory for Color.\n" ≡ flush;
      Color *c = create_new < Color > (0);
      *c = &fill_color_ptr;
      (**iter).set_fill_color(c);
    } else {
      if (DEBUG) {
        cout ≡ "fill_color_ptr->get_name() ≡ " fill_color_ptr->get_name() ≡ endl ≡ flush;
        (**iter).set_fill_color(fill_color_ptr);
      }
    }
    (**iter).set_pen(ppen);
    (**iter).set_dash_pattern(ddashed);
}
```

This code is used in section 1430.

1429. Filldraw.

(Declare Solid functions 1336) ≡
```cpp
virtual void filldraw(const vector< const Color * > draw_colors = Colors::default_color_vector, const vector< const Color * > fill_colors = Colors::background_color_vector, const string ddashed = "", const string ppen = "", Picture &picture = current_picture ) const;
```
§1430

(Define Solid functions 1337 ) +≡

void Solid::filldraw (const vector<const Color *> draw_colors, const vector<const Color *> fill_colors, const string dashed, const string pen, Picture &picture) const { bool DEBUG = false; /* true */
if (DEBUG) cout << "Entering Solid::filldraw(): " << endl << flush;
Solid *s = create_new < Solid > (0);
*s = this;
const Color *draw_color_ptr;
const Color *fill_color_ptr;
vector<const Color *>::const_iterator draw_color_iter = draw_colors.begin();
vector<const Color *>::const_iterator fill_color_iter = fill_colors.begin();
for (vector<Path *>::const_iterator iter = s->paths.begin(); iter != s->paths.end(); ++iter) {
  (Process vectors for filldraw() 1428 )
}
draw_color_iter = draw_colors.begin();
fill_color_iter = fill_colors.begin();
for (vector<Circle *>::const_iterator iter = s->circles.begin(); iter != s->circles.end(); ++iter) {
  (Process vectors for filldraw() 1428 )
}
draw_color_iter = draw_colors.begin();
fill_color_iter = fill_colors.begin();
for (vector<Ellipse *>::const_iterator iter = s->ellipses.begin(); iter != s->ellipses.end(); ++iter) {
  (Process vectors for filldraw() 1428 )
}
draw_color_iter = draw_colors.begin();
fill_color_iter = fill_colors.begin();
for (vector<Reg_Polygon *>::const_iterator iter = s->reg_polygons.begin(); iter != s->reg_polygons.end(); ++iter) {
  (Process vectors for filldraw() 1428 )
}
for (vector<Rectangle *>::const_iterator iter = s->rectangles.begin(); iter != s->rectangles.end(); ++iter) {
  (Process vectors for filldraw() 1428 )
}picture += dynamic_cast<Shape *>(s);
if (DEBUG) {
  cout << "Exiting Solid::filldraw(): " << endl << flush;
}
1431. Process vectors for `undraw()`. [LDF 2002.10.09.] Added this section. The same things are done to each of the `Shape`* vectors `paths`, `ellipses`, `circles`, `reg_polygons`, and `rectangles`, so I've put the code in this named section. Each time it's used, `iter` is an iterator for a different vector.

```cpp
{ Process vectors for undraw() 1431) 
  {
    (**iter**).set_fill_draw_value(UNDRAW);
    (**iter**).set_fill_color(static_cast(Color*)(*0));
    (**iter**).set_fill_color(static_cast(Color*)(*0));
    (**iter**).set_dash_pattern(ddashed);
    (**iter**).set_pen(ppen);
  }

This code is used in section 1433.

1432. Undraw.

(Declare Solid functions 1336) +

```cpp
virtual void undraw(const string ddashed = "", const string ppen = ", Picture &picture = current_picture) const;
```n

1433. (Define Solid functions 1337) +

```cpp
void Solid::undraw(const string ddashed, const string ppen, Picture &picture) const{
  bool DEBUG = false;  //* true */
  if (DEBUG) cout << "Entering Solid::undraw():" << endl;
  Solid *s = create_new < Solid > (0);
  *s = *this;
  for (vector(Path *)::const_iterator iter = s-paths.begin(); iter != s-paths.end(); ++iter) {
    (Process vectors for undraw() 1431)
  }
  for (vector(Circle *)::const_iterator iter = s-circles.begin(); iter != s-circles.end(); ++iter) {
    (Process vectors for undraw() 1431)
  }
  for (vector(Ellipse *)::const_iterator iter = s-ellipses.begin(); iter != s-ellipses.end(); ++iter) {
    (Process vectors for undraw() 1431)
  }
  for (vector(Reg_Polygon *)::const_iterator iter = s-reg_polygons.begin(); iter != s-reg_polygons.end(); ++iter) {
    (Process vectors for undraw() 1431)
  }
  for (vector(Rectangle *)::const_iterator iter = s-rectangles.begin(); iter != s-rectangles.end(); ++iter) {
    (Process vectors for undraw() 1431)
  }
  picture += dynamic_cast(Shape *)(s);
  if (DEBUG) {
    cout << "Exiting Solid::undraw():" << endl;
  }
}
```

[Log]

[LDF 2002.10.09.] Added this section. The same things are done to each of the Shape * vectors paths, ellipses, circles, reg.polygons, and rectangles, so I've put the code in this named section. Each time it's used, iter is an iterator for a different vector.

[LDF 2003.08.10.] Now setting pen to "", since I've removed the pen argument to `unfill()`.

```cpp
// Process vectors for unfill() 1434 )
{
    (**iter).set_fill_draw_value (UNFILL);
    (**iter).set_draw_color (static_cast (Color *) (0));
    (**iter).set_fill_color (static_cast (Color *) (0));
    (**iter).set_dash_pattern ("" );
    (**iter).set_pen (""");
}
```

This code is used in section 1436.

1435. Unfill.

[Log]

[LDF 2003.08.10.] Removed the pen argument, since unfilling doesn't use a pen.

```cpp
// Declare Solid functions 1336 ) +
virtual void unfill (Picture &picture = current_picture) const;
```
1436.  
(Define Solid functions 1337 ) 

```cpp
void Solid::unfill(Picture &picture) const {
  if (DEBUG) cout << "Entering Solid::unfill()" << "\n" << flush;
  Solid s = create_new < Solid>(0);
  s = *this;
  for (vector(Path *)::const_iterator iter = s-paths.begin(); iter != s-paths.end(); ++iter) {
    (Process vectors for unfill() 1434)
  }
  for (vector(Circle *)::const_iterator iter = s-circles.begin(); iter != s-circles.end(); ++iter) {
    (Process vectors for unfill() 1434)
  }
  for (vector(Ellipse *)::const_iterator iter = s-ellipses.begin(); iter != s-ellipses.end(); ++iter) {
    (Process vectors for unfill() 1434)
  }
  for (vector(Reg_Polygon *)::const_iterator iter = s-reg_polygons.begin();
       iter != s-reg_polygons.end(); ++iter) {
    (Process vectors for unfill() 1434)
  }
  for (vector(Rectangle *)::const_iterator iter = s-rectangles.begin(); iter != s-rectangles.end();
       ++iter) {
    (Process vectors for unfill() 1434)
  }
  picture += dynamic_cast<Shape*>(s);
  if (DEBUG) {
    cout << "Exiting Solid::unfill()" << "\n" << flush;
  }
}
```

1437.  Process vectors for unfilldraw().  [LDF 2002.10.09]  Added this section.  The same things are done
to each of the Shape * vectors paths, ellipses, circles, reg polyg on s, and rectangles, so I've put the code in
this named section.  Each time it's used, iter is an iterator for a different vector.

```cpp
(Process vectors for unfilldraw() 1437) =
{
  (**iter).set_fill_draw_value(UNFILLDRAW);
  (**iter).set_draw_color(static_cast<Color*>(0));
  (**iter).set_fill_color(static_cast<Color*>(0));
  (**iter).set_dash_pattern(ddashed);
  (**iter).set_pen(ppen);
}
```

This code is used in section 1439.

1438.  Unfilldraw.  [LDF 2002.10.09]  Unlike Path::unfilldraw(), Solid::unfilldraw() behaves like
METAPOST's unfilldraw command, i.e., it unfills and undraws.  I intend to change Path::unfilldraw() so
that it also behaves this way.

[LF 2002.10.09] Check this: the correct code is written to out_stream, but after filldraw() and
unfilldraw(), the outline is visible, but no lines inside the outline.  TO DO: Check what filldraw and
unfilldraw mean in METAPOST and METAFONT.

(Declare Solid functions 1336 ) +=

```cpp
virtual void unfilldraw(const string ddashed = "", const string ppen = "", Picture &picture = current_picture) const;
```
1439. 
(Define Solid functions 1337) +=
void Solid::unfilldraw(const string ddashed, const string pen, Picture &picture) const { bool
  DEBUG = false; /* true */
  if (DEBUG) cout << "EnteringSolid::unfilldraw():" << "\n" << flush;
  Solid *s = create_new < Solid > (0);
  *s = this;
  for (vector(Path *)::const_iterator iter = s->paths.begin(); iter != s->paths.end(); ++iter) {
    (Process vectors for unfilldraw ( ) 1437)
  }
  for (vector(Circle *)::const_iterator iter = s->circles.begin(); iter != s->circles.end(); ++iter) {
    (Process vectors for unfilldraw ( ) 1437)
  }
  for (vector(Ellipse *)::const_iterator iter = s->ellipses.begin(); iter != s->ellipses.end(); ++iter) {
    (Process vectors for unfilldraw ( ) 1437)
  }
  for (vector(Reg_Polygon *)::const_iterator iter = s->reg_polygons.begin();
    iter != s->reg_polygons.end(); ++iter) {
    (Process vectors for unfilldraw ( ) 1437)
  }
  for (vector(Rectangle *)::const_iterator iter = s->rectangles.begin(); iter != s->rectangles.end();
    ++iter) {
    (Process vectors for unfilldraw ( ) 1437)
  }
  picture += dynamic_cast<Shape *>(s);
  if (DEBUG) {
    cout << "ExitingSolid::unfilldraw():" << "\n" << flush;
  }
}

1440. Putting Solid together.

1441. This is what's compiled.
  (Include files 6)
  (Version control identifier 5)
  (Define class Solid 1333)
  (Define static const Solid data members 1334)
  (Define Solid functions 1337)
  (Declare non-member template functions for Solid 1341)
1442. This is what's written to solids.h.

```cpp
(Define class Solid 1333)
(Declare non-member template functions for Solid 1341)
```

1443. **Solid_Faced** (solfaced.web).

```cpp
Log

[LDF 2002.09.26.] Created this file.
[LDF 2003.11.12.] Removed the version control identifiers from the CWEB files for the distribution of 3DLDF 1.1. They're still used in my development versions.
[LDF 2003.12.10.] Put the version control identifiers back into my release versions for 3DLDF 1.1.4. I've already put some of them back in, now I'm doing the rest of them. However, the release versions are now in their own RCS repository.

(Version control identifier 5) ±≡
static string rs_id = "$Id: solfaced.web,v.1.4.0,004/01/12,21:33:15,1finsto1,Exp0$";
```

1444. Include files.

```cpp
(Include files 6) ±≡
#include "loader.h"
#include "pspglib.h"
#include "io.h"
#include "colors.h"
#include "transfor.h"
#include "shapes.h"
#include "pictures.h"
#include "points.h"
#include "lines.h"
#include "planes.h"
#include "paths.h"
#include "curves.h"
#include "polygons.h"
#include "rectangs.h"
#include "ellipses.h"
#include "circles.h"
#include "solids.h"
```

1445. **Solid_Faced** class definition.

```cpp
(Define class Solid_Faced 1445) ±≡
class Solid_Faced : public Solid {
 protected: unsigned short faces;
         unsigned short vertices;
         unsigned short edges;
 public: (Declare Solid_Faced functions 1446)
};
```

This code is used in sections 1449 and 1450.

1446. (Declare Solid_Faced functions 1446) ±≡
This code is used in section 1445.
1447. (Define Solid_Faced functions 1447) Ξ
This code is used in section 1449.

1448. Putting Solid_Faced together.

1449. This is what’s compiled.
  (Include files 6)
  (Version control identifier 5)
  (Define class Solid_Faced 1445)
  (Define Solid_Faced functions 1447)
1450. This is what's written to solfaced.h.
{solfaced.h 1450} 
{Define class Solid_Faced 1445}

1451. Cuboid (cuboid.web).

Log

[LDF 2002.04.22.] Created this file. When I've found out what the English word is for "Quader", I'll change it globally.
[LDF 2002.04.22.] Cuboid is the first three-dimensional object I've defined. I've just quickly put it together for use in a drawing. Ultimately, I'd like to derive it from Shape, which will require defining versions of all the pure virtual functions in Shape.
[LDF 2002.04.23.] Changed Quader to Cuboid. Haven't changed name of file, because this is more complicated, because of RCS (the source code control system).
[LDF 2002.06.03.] Changed the name of this file from quader.web to cuboid.web. This means that if you need to compare this file with revisions earlier than the initial version of this file, you'll have to check revisions of quader.web.
[LDF 2003.11.12.] Removed the version control identifiers from the CWEB files for the distribution of 3DLDF 1.1. They're still used in my development versions.
[LDF 2003.12.10.] Put the version control identifiers back into my release versions for 3DLDF 1.1.4. I've already put some of them back in, now I'm doing the rest of them. However, the release versions are now in their own RCS repository.

format Cuboid Solid
{Version control identifier 5} +≡
static string rev_id = "$Id:\u005fcuboid.web,v\u001d1.6\u002c2004/01/12\u002c21:27:51\u002c1finsto1\u002cExp\u0095$";

1452. Include files.
{Include files 6} +≡
#include "loader.h"
#include "pspglb.h"
#include "createnew.h"
#include "io.h"
#include "colors.h"
#include "transfor.h"
#include "shapes.h"
#include "pictures.h"
#include "points.h"
#include "lines.h"
#include "planes.h"
#include "paths.h"
#include "curves.h"
#include "polygons.h"
#include "rectangs.h"
#include "ellipses.h"
#include "circles.h"
#include "solids.h"
#include "solfaced.h"

1453. Cuboid class definition,
Log

[LD 2002.04.22.] Added this class declaration.
[LD 2003.08.10.] Removed \textit{dihedral angle}.

\begin{verbatim}
(Define class Cuboid 1453) \equiv
class Cuboid : public Solid_Faced {
protected:
  real height;
  real width;
  real depth;
public:
  \{ Declare Cuboid functions 1455 \}
};
\end{verbatim}

This code is used in sections 1468 and 1469.

1454. Constructors and setting functions.

\begin{verbatim}
(Declare Cuboid functions 1455) \equiv
Cuboid();
\end{verbatim}

See also sections 1457, 1459, 1464, and 1466.
This code is used in section 1453.

1456.
\begin{verbatim}
(Define Cuboid functions 1456) \equiv
Cuboid::Cuboid() {
  on_free_store = false;
  do_output = true;
  projective_extremes.resize(6, 0);
  faces = 6;
  vertices = 8;
  edges = 12;
}
\end{verbatim}

See also sections 1458, 1460, 1465, and 1467.
This code is used in section 1468.

1457. Copy constructor. [LD 2002.05.03.] Added this function.
\begin{verbatim}
(Declare Cuboid functions 1455) +\equiv
Cuboid(const Cuboid &c);
\end{verbatim}
1458.  
(Define Cuboid functions 1456) +≡
  Cuboid::Cuboid(const Cuboid &c){ on_free_store = false;
    do_output = true;
    projective_extremes.resize(6,0);
    faces = 6;
    vertices = 8;
    edges = 12; for (vector(Rectangle *)::const_iterator iter = c.rectangles.begin();
    iter != c.rectangles.end(); iter++) { rectangles.push_back ( create_new <Rectangle> (0) );
    *(rectangles.back()) = **iter; } }

1459.  Center, height, width, depth, and angles.  [LDF 2002.10.06] Added this constructor.
(Declare Cuboid functions 1455) +≡
  Cuboid(const Point &c, const real h, const real w, const real d, const real x = 0, const real y = 0, const real z = 0);
1460.  
\{Define Cuboid functions 1456\} \equiv  
\begin{verbatim}
Cuboid::Cuboid(const Point &c, const real h, const real w, const real d, const real x, const real y, const real z):  
  height(h), width(w), depth(d) {  
    bool DEBUG = false;  
    /* true */
  
    on_free_store = false;
  
    do_output = true;
  
    projective_extremes.resize(6,0);
    center = c;
    
    faces = 6;
  
    vertices = 8;
  
    edges = 12;

  Point pts[9];
    
  pts[1].shift(-.5 * width, -.5 * height, -.5 * depth);
  pts[2].shift(.5 * width, -.5 * height, -.5 * depth);
  pts[3].shift(.5 * width, .5 * height, -.5 * depth);
  pts[4].shift(-.5 * width, .5 * height, -.5 * depth);
  pts[5].shift(-.5 * width, -.5 * height, .5 * depth);
  pts[6].shift(.5 * width, -.5 * height, .5 * depth);
  pts[7].shift(.5 * width, .5 * height, .5 * depth);
  pts[8].shift(-.5 * width, .5 * height, .5 * depth);
  
    for (int i = 0; i < 6; i++) {  
      rectangles.push_back (  
        create_new < Rectangle > (0) ;  
      }  
    rectangles[0]=set(pts[1],pts[2],pts[3],pts[4]);

    /* front */
    rectangles[1]=set(pts[5],pts[6],pts[7],pts[8]);  
      /* back */
    rectangles[2]=set(pts[1],pts[4],pts[8],pts[5]);  
      /* left */
    rectangles[3]=set(pts[2],pts[6],pts[7],pts[3]);  
      /* right */
    rectangles[4]=set(pts[3],pts[7],pts[8],pts[4]);  
      /* top */
    rectangles[5]=set(pts[1],pts[2],pts[6],pts[5]);  
      /* bottom */

    rotate(x,y,z);

    shift(c);
    if (DEBUG)
      for (int i = 1; i < 9; i++) pts[i].dotlabel(i);
  }
\end{verbatim}

1461.  Pseudo-constructor for dynamic allocation.

1462.  Pointer argument.

\begin{verbatim}
Log  
  [LDF 2002.04.22]  Added this function.
  [LDF 2003.12.30]  Replaced Cuboid::create_new_cuboid() with a specialization of template(class C)  
  C*create_new() for Cuboid. The argument is now const.
\end{verbatim}

\{Declare non-member template functions for Cuboid 1462\} \equiv  

\begin{verbatim}
Cuboid *create_new(const Cuboid &c);
\end{verbatim}

See also section 1463.

This code is used in sections 1468 and 1469.

1463.  Reference argument.

(Declare non-member template functions for Cuboid 1462) +≡
Cuboid *create_new(const Cuboid &c);

1464. Destructor.  !! Make sure to delete anything else that I allocate dynamically!
(Declare Cuboid functions 1455) +≡
~Cuboid();

1465.
(Define Cuboid functions 1456) +≡
Cuboid :: ~Cuboid()
{
    for (vector<Rectangle *>::iterator iter = rectangles.begin(); iter != rectangles.end(); iter++) {
        delete *iter;
    }
    rectangles.clear();
}

1466. Assignment.
(Declare Cuboid functions 1455) +≡
void operator=(const Cuboid &c);

1467.
(Define Cuboid functions 1456) +≡
void Cuboid::operator=(const Cuboid &c)
{
    this->Solid::operator=(c);
    height = c.height;
    width = c.width;
    depth = c.depth;
}

1468. Putting Cuboid together.  This is what's compiled.
(Include files 6 )
(Version control identifier 5 )
(Define class Cuboid 1453)
(Define Cuboid functions 1456)
(Declare non-member template functions for Cuboid 1462)
1469. This is what’s written to cuboid.h.

\{(cuboid.h 1469) \equiv
\{Define class Cuboid 1453\}
\{Declare non-member template functions for Cuboid 1462\}\}


[LDF 2002.11.12] TO DO: Add assignment operators for Polyhedra! The individual types will need their own, but they can call Polyhedron::operator=( ).

Log

[LDF 2003.11.12] Removed the version control identifiers from the CWEB files for the distribution of 3DLDF 1.1. They’re still used in my development versions.
[LDF 2003.12.10] Put the version control identifiers back into my release versions for 3DLDF 1.1.4. I’ve already put some of them back in, now I’m doing the rest of them. However, the release versions are now in their own RCS repository.

\{(Version control identifier 5) \equiv
\{static string rev_id = "$Id: polyhedra_web,CWEB,1.5,2004/01/12,21:32:19,1finslo1Exp,\$";\}

1471. Include files.

\{(Include files 6) \equiv
#include "loader.h"
#include "pspglb.h"
#include "createmesh.h"
#include "io.h"
#include "colors.h"
#include "transform.h"
#include "shapes.h"
#include "pictures.h"
#include "points.h"
#include "lines.h"
#include "planes.h"
#include "paths.h"
#include "curves.h"
#include "polygons.h"
#include "rectangles.h"
#include "ellipses.h"
#include "circles.h"
#include "solids.h"
#include "solfaced.h"

1472. Polyhedron class definition. [LDF 2002.10.06] Polyhedron is meant to be used only as a base class, so there’s no need for constructors or setting functions.

TO DO: [LDF 2003.08.15] If I add any functions, I should add an explanation to “@node Polyhedron Getstart” in DOCUMENTATION/gasolfigi.texi about abstract or non-abstract base classes.

Log

[LDF 2002.11.08] Got rid of pure virtual function Polyhedron::get_next( ). I’ve made it static in the classes derived from Polyhedron, which makes more sense. virtual functions must be non-static.
format Polyhedron Reg_Polygon
(Define class Polyhedron 1472) \equiv
class Polyhedron : public Solid_Faced {
  protected: unsigned short number_of_polygon_types;
  real face_radius;
  real edge_radius;
  real vertex_radius;
public: (Declare Polyhedron functions 1473)
};

This code is used in sections 1534 and 1535.

1473. Intersection. \n[ ] [LDF 2003.04.15.] START HERE. This function doesn’t work yet.

---

Log

[ ] [LDF 2003.04.09.] Added this section, and the function it contains.
[ ] [LDF 2003.04.15.] Commented-out this function, since I need to get Reg_Polygon::intersection_points() working first.

(Declare Polyhedron functions 1473) \equiv

#include <regphen.h>

virtual vector(Point) intersection_points(const Reg_Polygon &r) const;

} endif

This code is used in section 1472.
1474.  
\(\text{Define Polyhedron functions 1474} \equiv\)  
\#if 0  
vector\langle Point\rangle Polyhedron :: intersection_points(const Reg_Polygon &r) const  
{  
  vector\langle Point\rangle w;  
  vector\langle Point\rangle v;  
  for (vector\langle Reg_Polygon \rangle::const_iterator iter0 = reg_polygons.begin(); iter0 != reg_polygons.end(); ++iter0)  
  {  
    for (vector\langle Reg_Polygon \rangle::const_iterator iter1 = iter0 + 1; iter1 != reg_polygons.end(); ++iter1)  
    {  
      v = (**iter0).intersection_points(**iter1);  
      cout << "v.size() = " << v.size() << endl << flush;  
      for (vector\langle Point\rangle::iterator pt_iter = v.begin(); pt_iter != v.end(); ++pt_iter)  
        w.push_back(*pt_iter);  
    }  
  }  
  return w;  
\}  
\#endif  

This code is used in section 1534.

1475.  Regular Platonic Polyhedra.

1476.  Tetrahedron.


1477.  Tetrahedron class definition.


\text{format  Tetrahedron Polyhedron}  
\(\text{(Define class Tetrahedron 1477) \equiv}\)  
\{  
  class Tetrahedron : public Polyhedron  
  {  
    protected: static const real \text{dihedral}\_\text{angle};  
      /* In radians! */  
    real \text{triangle}\_\text{radius};  
  }  
  public:  
  {  
    \langle\text{Declare Tetrahedron functions 1480}\rangle  
  }  
\};  

This code is used in sections 1534 and 1535.

1478.  Define static const Tetrahedron data members.


\(\text{(Define static const Tetrahedron data members 1478) \equiv}\)  
\{  
  const real Tetrahedron::\text{dihedral}\_\text{angle} = PI * (70 + 32/60.0)/180.0;  
\}
1479. Constructors and setting functions.

1480. Default constructor.  (No arguments.)

[LDF 2002.11.12.] Added this function.

\[
\text{Declare Tetrahedron functions } 1480 \equiv \text{Tetrahedron();}
\]

See also sections 1483, 1487, 1489, and 1491.

This code is used in section 1477.

1481.

\[
\text{Declare Tetrahedron functions } 1481 \equiv \text{Tetrahedron::Tetrahedron()}
\]

\[
\begin{array}{l}
\text{on_free_store = false; } /* \text{from Solid. */}
\text{do_output = true; }
\text{faces = 4; } /* \text{from Solid_Faced. */}
\text{vertices = 4;}
\text{edges = 6;}
\text{center = INVALID_POINT; } /* \text{from Polyhedron. */}
\text{number_of_polygon_types = 1;}
\text{face_radius = edge_radius = vertex_radius = INVALID_REAL;}
\text{triangle_radius = INVALID_REAL; } /* \text{From Tetrahedron. */}
\end{array}
\]

See also sections 1484, 1485, 1486, 1488, 1490, and 1492.

This code is used in section 1534.

1482. Center, diameter of triangle, and angles.

1483. Constructor.

[LDF 2002.11.12.] Added this function.
[LDF 2003.04.27.] Got this function to work, at least in a rudimentary way.
[LDF 2002.08.12.] Rewrote this function. It now works properly.

\[
\text{Declare Tetrahedron functions } 1480 \equiv \text{Tetrahedron(const Point &p, const real diameter_of_triangle, real angle_x = 0, real angle_y = 0, real angle_z = 0);}
\]
1484.

(Define Tetrahedron functions 1481) \( \equiv \)

\begin{verbatim}
Tetrahedron::Tetrahedron(const Point &p, const real triangle_diameter, real angle_x, real angle_y, real angle_z)
{  bool DEBUG = true;  /* false */
  on_free_store = false;  /* from Solid. */
  do_output = true;
  faces = 4;  /* from Solid_Faced. */
  vertices = 4;
  edges = 6;
  number_of_polygon_types = 1;
  
  #if 0  /* START HERE, TO DO: Must calculate these! */
    face_radius = 0;
    edge_radius = 0;
    vertex_radius = 0;
  #endif
  triangle_radius = triangle_diameter / 2.0;
  reg_polygons = get_net(triangle_diameter);
  real angle = 180 - (dihedral_angle * 180 / PI);
  Point pts[11];
  int i;
  for (i = 0; i < 3; ++i) pts[i] = reg_polygons[0]->get_point(i);
  reg_polygons[1]->rotate(pts[0], pts[1], angle);
  reg_polygons[2]->rotate(pts[2], pts[0], angle);
  reg_polygons[3]->rotate(pts[1], pts[2], -angle);
  
  #if 0
    for (i = 0; i < 3; ++i) pts[i].label(i, "n");
  #endif
  for (i = 3; i < 7; ++i) {
    pts[i] = reg_polygons[i - 3]->get_center();
  }  
  #if 0
    pts[i].label(i, "n");
  #endif
  pts[7] = reg_polygons[3]->get_point(0);
  
  #if 0
    pts[7].label(7, "n");
  #endif
  
  pts[8] = pts[0].mediate(pts[1]);
  pts[10] = pts[2].mediate(pts[0]);
  
  #if 0
    for (i = 8; i < 11; i++) pts[i].label(i);
  #endif
  using namespace Colors;
  
  #if 0
    pts[0].draw(pts[6], blue);
    pts[1].draw(pts[5], red);
    pts[2].draw(pts[4], green);
    pts[3].draw(pts[7], orange);
  #endif
\end{verbatim}

\end{verbatim}
1485. *center* is the intersection point of the line segments from the vertices of *(reg_polygons[0]) to the centers of the opposite faces. *distance* is the distance along one of these line segments to the intersection point divided by the length of the entire line segment. [LDF 2002.08.12.]

Since this ratio should be the same for all *Tetrahedron*, there's no need to recalculate it each time a *Tetrahedron* is constructed. In addition, intersections can't always be found, because of inaccuracies caused by rotating the triangles. [LDF 2002.08.12.]

Therefore, I've calculated distance using the commented-out code below, and now simply use the value I found. [LDF 2002.08.12.]

```cpp
(Define *Tetrahedron* functions 1481) \+\=
real distance = 0.74997889995574951171875;
#endif
Point P0 = Point : intersection_point(pts[0], pts[6], pts[1], pts[5]).pt;
P0.show("P0=\(0, 0, 0, 0\)\);
Point P1 = pts[5] - pts[1];
Point P2 = P0 - pts[1];
P1.show("P1");
cout << "\|P1\| = \|P1\|" << P1.magnitude() << endl << flush;
P2.show("P2");
cout << "\|P2\| = \|P2\|" << P2.magnitude() << endl << flush;
distance = (P2.magnitude() / P1.magnitude());
cout.precision(25);
cout << "distance = \|distance\|" << distance << endl << flush;
cout.precision(6);
#endif

1486.
(Define *Tetrahedron* functions 1481) \+\=
center = pts[1].mediate(pts[5], distance);
for (i = 0; i < 4; ++i) reg_polygons[i].shift(-center);
center.shift(-center);
if (angle.x \neq 0 \lor angle.y \neq 0 \lor angle.z \neq 0) {
  for (i = 0; i < 4; ++i) reg_polygons[i].rotate(angle.x, angle.y, angle.z);
}
if (p \neq origin) {
  center = p;
  for (i = 0; i < 4; ++i) reg_polygons[i].shift(p);
}
return; }
```
1487. Setting function. [LDF 2002.11.12] This work, but it fails to assign to the data members of Tetrahedron that are defined in its own class declaration. That's because neither Tetrahedron nor Polyhedron has an assignment operator yet. TO DO: Write assignment operators for Polyhedra!

LDF 2002.11.12.] Added this function.

(Declare Tetrahedron functions 1480) +

void set(const Point &p, const real diameter_of_triangle, real angle_x = 0, real angle_y = 0, real angle_z = 0);

1488.

(Define Tetrahedron functions 1481) +

void Tetrahedron::set(const Point &p, const real triangle_diameter, real angle_x, real angle_y, real angle_z)
{
    Tetrahedron t(p, triangle_diameter, angle_x, angle_y, angle_z);
    *this = t;
    return;
}

1489. Get net. [LDF 2002.11.12.] Unlike the get_net() functions for some of the other Polyhedra, this function has no "bool do_half" argument. It doesn't pay for a Tetrahedron.

LDF 2002.11.12.] Added this function.
LDF 2002.08.12.] Removed center_0 argument.

(Declare Tetrahedron functions 1480) +

static vector (Reg_Polygon *) get_net(const real triangle_diameter);
1490. Define Tetrahedron functions 1481 +

vector (Reg_Polygon +) Tetrahedron::get_net (const real triangle_diameter)
{
    vector (Reg_Polygon +) triangles;
    int i; for (i = 0; i < 4; ++i) triangles.push_back (create_new < Reg_Polygon > (0));
    triangles[0].set (origin, 3, triangle_diameter, 0, 180);
    triangles[1].set (origin, 3, triangle_diameter);
    Point pts[6];
    for (i = 0; i < 3; ++i) {
        pts[i] = triangles[0].get_point (i);
    }
    for (i = 3; i < 6; ++i) {
        pts[i] = triangles[1].get_point (i - 3);
    }
    triangles[1].shift (pts[0] - pts[4]);
    triangles[2].shift (pts[0] - pts[5]);
    return triangles;
}

1491. Draw net. [LDF 2002.11.12.] As of this date it’s necessary to rotate the triangles into the x-y plane, because Point::intersection_point() has a bug that I discovered when I tried to call it on Points in the x-z plane. It’s not so terrible, because as of this date it’s necessary to call the Picture in the x-y plane in order to use the parallel projection. The latter currently only works for the x-y plane. TO DO: Fix the bug and get parallel projection onto other major planes to work!


(Declare Tetrahedron functions 1480 ) +

static void draw_net (const real triangle_diameter, bool make_tabs = true);
1492.

(Define Tetrahedron functions 1481 )

```cpp
void Tetrahedron : draw_net(const real triangle_diameter, bool make_tabs )
{
    vector<Reg_Polygon*> v = get_net(triangle_diameter);
    int i;
    for (i = 0; i < 4; i++) {
        v[i] = rotate(90);
        v[i] = draw();
        v[i] = get_center().label(i, "n");
    }
    if ( ! make_tabs ) return;
    Point pts[32];
    pts[0] = v[2] = get_point(1);
    pts[1] = v[2] = get_point(2);
    pts[2] = v[1] = get_point(2);
    pts[3] = v[1] = get_point(0);
    pts[4] = v[3] = get_point(0);
    pts[5] = v[3] = get_point(1);
    pts[6] = pts[0].mediate(pts[5], 0.75);
    pts[7] = pts[5].mediate(pts[0], 0.75);
    pts[8] = pts[6];
    pts[9] = pts[7];
    pts[8] = pts[9].shift(0, 0, 1);
    pts[10] = pts[0];
    pts[10] = rotate(pts[6], pts[8], -110);
    pts[11] = rotate(pts[7], pts[9], 110);
    pts[10] = pts[6].mediate(pts[10], 1.5);
    pts[11] = pts[7].mediate(pts[11], 1.5);
    #if 0
        for (i = 0; i < 8; i++) pts[i].dotlabel(i);
        pts[10].dotlabel(10);
        pts[11].dotlabel(11);
    #endif
    Path p[6];
    p[0].set("--", true, & pts[6], & pts[10], & pts[11], & pts[7], 0);
    p[0].draw();
    pts[12] = pts[6].mediate(pts[7]);
    #if 0
        pts[12].dotlabel(12);
        pts[13].dotlabel(13);
    #endif
    p[1].set(pts[12], pts[13]);
    #if 0
        p[1].draw_help(*Colors::help_color, "n");
    #endif
    pts[14] = pts[12].mediate(pts[13]);
    #if 0
        pts[14].dotlabel(14);
    #endif
```
#endif
pts[15] = pts[6].mediate(pts[7], .25);
#endif
pts[16] = dotlabel(15);
#endif
#endif
pts[16].dotlabel(16);
#endif
bool_point bp = Point::intersection_point(pts[14], pts[16], pts[6], pts[10]);
pts[17] = bp.pt;
#endif
pts[17].dotlabel(17);
#endif bp = Point::intersection_point(pts[14], pts[16], pts[7], pts[11]);
pts[18] = bp.pt;
#endif pts[18].dotlabel(18);
#endif p[2].set(pts[17], pts[18]);
#endif p[2].draw_help(*Colors::help_color, "\n");
#endif
for (i = 1; i < 16; ++i) {
    pts[19] = pts[17].mediate(pts[18], i/16.0);
    pts[19].drawdot(*Colors::default_color, "pencircle_scaled_1.5mm");
}
pts[20] = pts[17];
pts[21] = pts[18];
p[3] = p[0];

Transform t;
t.shift(pts[4] - pts[5]);
t.rotate(pts[4], pts[5]);
p[3].draw();
for (i = 1; i < 16; ++i) {
    pts[19] = pts[20].mediate(pts[21], i/16.0);
    pts[19].drawdot(*Colors::default_color, "pencircle_scaled_1.5mm");
}
t.reset();
t.rotate(pts[4], pts[1]);
p[3].draw();
for (i = 1; i < 16; ++i) {
    pts[19] = pts[20].mediate(pts[21], i/16.0);
    pts[19].drawdot(*Colors::default_color, "pencircle_scaled_1.5mm");
}
pts[20] = pts[17];
pts[21] = pts[18];
p[3] = p[0];
\[\] 

\[
\text{pts}[20] \ast= \text{pts}[21] \ast= p[3] \ast= t; \\
p[3].\text{draw}(); \\
\text{for} (i = 1; i < 16; ++i) \{ \\
\quad \text{pts}[19] = \text{pts}[20].\text{mediate}(\text{pts}[21], i/16.0); \\
\quad \text{pts}[19].\text{drawdot}(\text{Colors::default_color}, \text{"pencircle_scaled_1.5mm"}); \\
\} \\
\text{t.reset}(); \\
\text{pts}[22] = \text{v[2]-get_center}(); \\
\text{pts}[23] = \text{pts}[22]; \\
\text{pts}[23].\text{shift}(0, 0, 1); \\
\text{t.rotate}([\text{pts}[22], \text{pts}[23], 120]); \\
\text{pts}[20] = \text{pts}[17]; \\
\text{pts}[21] = \text{pts}[18]; \\
\text{p}[3] = \text{p[0]}; \\
\text{pts}[20] \ast= \text{pts}[21] \ast= p[3] \ast= t; \\
\text{p}[3].\text{draw}(); \\
\text{for} (i = 1; i < 16; ++i) \{ \\
\quad \text{pts}[19] = \text{pts}[20].\text{mediate}(\text{pts}[21], i/16.0); \\
\quad \text{pts}[19].\text{drawdot}(\text{Colors::default_color}, \text{"pencircle_scaled_1.5mm"}); \\
\} \\
\text{t.rotate}([\text{pts}[4], \text{pts}[1]]); \\
\text{t.rotate}([\text{pts}[0], \text{pts}[2]]); \\
\text{pts}[20] = \text{pts}[17]; \\
\text{pts}[21] = \text{pts}[18]; \\
\text{p}[3] = \text{p[0]}; \\
\text{pts}[20] \ast= \text{pts}[21] \ast= p[3] \ast= t; \\
\text{p}[3].\text{draw}(); \\
\text{for} (i = 1; i < 16; ++i) \{ \\
\quad \text{pts}[19] = \text{pts}[20].\text{mediate}(\text{pts}[21], i/16.0); \\
\quad \text{pts}[19].\text{drawdot}(\text{Colors::default_color}, \text{"pencircle_scaled_1.5mm"}); \\
\} \\
\}
\]

1493. Dodecahedron.

1494. Dodecahedron class definition.

```cpp
format Dodecahedron Polyhedron
(Define class Dodecahedron 1494) \equiv
\begin{verbatim}
class Dodecahedron : public Polyhedron {
    protected: static const real dihedralAngle; /* In radians! */
    real pentagon_radius;
    public: (Declare Dodecahedron functions 1497)
};
\end{verbatim}
```

This code is used in sections 1534 and 1535.

1495. Define static const Dodecahedron data members.

---

[LDF 2003-07-18.] Now passing "2.0" instead of "2.0" as the argument to \textit{atan}(). GCC 3.3 couldn't compile this file, the way it was before.
1496. Constructors and setting functions.

1497. Default constructor. (No arguments.) [LDF 2002.09.29] TO DO: I should set the data members of other classes to INVALID_POINT, INVALID_REAL, etc., in the default constructors, too.

(Declare Dodecahedron functions 1497) ≡

Dodecahedron();

See also sections 1500, 1503, and 1505.

This code is used in section 1494.

1498.

(Define Dodecahedron functions 1498) ≡

Dodecahedron::Dodecahedron()
{
  on_free_store = false; /* from Solid, */
  do_output = true;
  faces = 12; /* from Solid_Faced, */
  vertices = 20;
  edges = 30;
  center = INVALID_POINT; /* from Polyhedron, */
  number_of_polygon_types = 1;
  face_radius = edge_radius = vertex_radius = INVALID_REAL;
  pentagon_radius = INVALID_REAL; /* From Dodecahedron. */
}

See also sections 1501, 1502, 1504, and 1506.

This code is used in section 1534.

1499. Center, diameter of pentagon, and angles.

1500. Constructor. [LDF 2003.08.10] TO DO: Check the way I specify the rotations. If it's not the way I think it should be, check Transform::align_with_axis() and any other functions that are involved. This may be a long-term project.

Log

[LDF 2002.10.16] Added shift to center and rotation.
[LDF 2003.08.10] Rewrote this function. It had suddenly stopped working properly, probably because of changes I made to Transform::align_with_axis(). I'm still not entirely happy with the way I've had to specify the rotations, see the "TO DO" note of this date, above.

(Declare Dodecahedron functions 1497) ≡

Dodecahedron(const Point &p, const real pentagon_diameter, real angle_x = 0, real angle_y = 0, real angle_z = 0);
1501.

(Define **Dodecahedron** functions 1498 ) +

**Dodecahedron::Dodecahedron**(const **Point &p, const real pentagon_diameter, real angle_x, real angle_y, real angle_z){
    bool DEBUG = false;    /* true */
    if (DEBUG) cout << "Entering Dodecahedron::Dodecahedron() \n" << flush;
    on_free_store = false;    /* from Solid */
    do_output = true;
    faces = 12;    /* from Solid_Faced */
    vertices = 20;
    edges = 30;
    number_of_polygon_types = 1;

    #if 0    /* START HERE, TO DO: Must calculate these! */
    face_radius = 0;
    edge_radius = 0;
    vertex_radius = 0;
    #endif

    pentagon_radius = pentagon_diameter / 2.0;
    reg_polygons = get_net(pentagon_diameter, true);
    **Point** pts[8];
    int i = 0;
    for (i = 0; i < 5; ++i) {
        pts[i] = reg_polygons[0].get_point(i);
    }

    real angle = 180 - (dihedral_angle * 180.0 / 31);
1502. [LDF 2003.08.10] Check this, as noted above.

(Define **Dodecahedron** functions 198) +

```cpp
reg_polygons[1].rotate (pts[0], pts [1], angle);
reg_polygons[2].rotate (pts[1], pts [0], angle);
reg_polygons[3].rotate (pts [2], pts [3], angle);
reg_polygons[4].rotate (pts[3], pts [2], angle);
reg_polygons[5].rotate (pts [1], pts [2], angle);

#if 0
using namespace Colors;
vector< const Color > col_vec;
col_vec.push_back (& black);
col_vec.push_back (& red);
col_vec.push_back (& green);
col_vec.push_back (& blue);
col_vec.push_back (& cyan);
col_vec.push_back (& magenta);
col_vec.push_back (& orange);
#endif
if (DEBUG) {
    i = 0;
    for (vector< Reg_Polygon >:: iterator iter = reg_polygons.begin (); iter != reg_polygons.end (); ++iter) {
        **iter . draw (*col_vec[i]);
    }
    ++i;
}
}
for (i = 0; i < 6; ++i) { reg_polygons.push_back ( create_new < Reg_Polygon > (0) );
*reg_polygons.back() = *reg_polygons[1];
reg_polygons.back() . rotate (180); } pts[5] = reg_polygons[1] . get_point (4);
pts[6] = reg_polygons[10] . get_point (2);
if (DEBUG) {
    pts[5] . dotlabel("$p_{5}$");
    pts [6] . dotlabel("$p_{6}$");
}
for (i = 6; i < 12; ++i) reg_polygons[i] . shift (pts [7]);
#if 0
if (DEBUG) {
    for (i = 6; i < 12; ++i) reg_polygons[i] . draw (*col_vec[i - 6]);
}
#endif
Point center_O = reg_polygons[0] . get_center ();
Point center_0 = reg_polygons[6] . get_center ();
center = center_O . mediate (center_O);
Transform t = center . shift ( - center );
if (angle_x != 0 || angle_y != 0) t . rotate (angle_x, angle_y, angle_z);
if (p != origin) center += t . shift (p);
```
for (vector<Reg_Polygon *>::iterator iter = reg_polygons.begin(); iter != reg_polygons.end(); ++iter)
    (**iter) *= t;
if (DEBUG) cout << "Exiting Dodecahedron::Dodecahedron() \n" << flush;
return; }

1503. Get net. [LDF 2002.09.29.] Changed this function. TO DO. I've removed rotate() and now rotate
the pentagons in the x-z plane only in order to avoid having the y-coordinates be off by small amounts. This
might not happen with rotate() once I implement the routine in Salomon ( !! Get reference!) for calculating
sine and cosine using integers.

    const Point & center 0 is only ever used when do_half == false, that is, when the net isn't just being
generated for use by a constructor. If get_net() is called by a constructor, the net is always made with the
center of pentagon 0 at the origin. Even if center 0 is used, the net is always generated in a plane parallel
to the x-z plane.

[Log]

[Declares Dodecahedron functions 1497] +≡

static vector<Reg_Polygon *> get_net(const real pentagon_diameter, bool do_half = false);
1504. Define Dodecahedron functions 1498 +edef
vector<Reg_Polygon *> Dodecahedron::get_net(const real pentagon_diameter, bool do_half){ int i;
vector<Reg_Polygon *> pents; for (i = 0; i < 6; ++i) pents.push_back ( create_new <
Reg_Polygon > (0) );
pents[0]=set(origin,5,pentagon_diameter,0,180); /* The middle pentagon. */
Point pts[10];
for (i = 0; i < 5; i++) {
    pts[i] = pents[0]->get_point(i);
}
*pents[1] = *pents[0];
pents[1]->rotate(0,36);
for (i = 5; i < 10; i++) pts[i] = pents[1]->get_point(i - 5);
pents[1]->shift(pts[0] - pts[8]);
pents[2]->shift(pts[0] - pts[6]);
pents[5]->shift(pts[1] - pts[9]);
if (do_half == true)
/* [LDF 2002.09.29.] We only need half of the net, if we’re using it to make a polyhedron. In that
case, we copy and transform the half we’ve already got and the copy on top of the first half. */
{
    for (i = 0; i < 6; ++i) {
        for (int j = 0; j < 5; j++)
            if (pents[i]->get_point(j).get_y() != 0) {
                cerr << "ERROR! In Dodecahedron::get_net():
                "y-coordinate != 0!\n" << "You’d better fix this!\n" << flush;

            }
        return pents;
    }
}
for (i = 6; i < 12; i++) { pents.push_back ( create_new < Reg_Polygon > (0) );
*pents[i] = *pents[i - 6];
pents[i]->rotate(0,180); } pts[0] = pents[11]->get_point(0);
pts[1] = pents[5]->get_point(1);
for (i = 6; i < 12; ++i) pents[i]->shift(pts[1] - pts[0]);
for (i = 0; i < 12; ++i) {
    for (int j = 0; j < 5; j++)
        if (pents[i]->get_point(j).get_y() != 0) {
            cerr << "ERROR! In Dodecahedron::get_net():
            "y-coordinate != 0!\n" << "You’d better fix this!\n" << flush;

        }
    return pents;
}

1505. Draw net. [LDF 2002.11.10.] This function is for drawing the net of a Dodecahedron. Normally,
this will be done in order to make a cardboard model, which will require tabs for gluing the pentagons
together. If no tabs are desired, passing false as the make_tabs argument will suppress the tabs.
[LDF 2002.11.10.] TO DO: The arrays Point *pts* and Path *p* have too many members. In working on this function, I ended up getting rid of some of the members of these arrays after I’d already used members following them. I should go through and reassign the numbers, so that no members are skipped.

[LDF 2002.11.10.] TO DO: Add the usual arguments for drawing and filling commands.

[LDF 2002.11.10.] TO DO: *portrait* doesn’t work right. Fix it! !! KLUDGE: *portrait* is set to false at the beginning of this function and a warning is issued.

---

Log

---

[LDF 2002.11.10.] Added this function.

[LDF 2002.11.10.] Tried to get output in portrait format to work, but it doesn’t yet.

(Declare Dodecahedron functions 1497) +≡

```c
static void draw_net(const real pentagon_diameter, bool portrait = true, bool make_tabs = true);
```
1506. Define Dodecahedron functions 1498 +

```cpp
void Dodecahedron::draw_net(const real pentagon_diameter, bool portrait, bool makeTabs)
{
    vector<Reg_Polygon *> v = get_net(pentagon_diameter);
    for (vector<Reg_Polygon *>::iterator iter = v.begin(); iter != v.end(); ++iter) {
        if (portrait) (**iter).rotate(0, 90);
    }
    Point p[32];
    int i;
    for (i = 0; i < 5; i++) {
        p[i] = v[i] - getPoint(i);
    }
    /* START HERE. */
    i = 0;
    for (vector<Reg_Polygon *>::iterator iter = v.begin(); iter != v.end(); ++iter) {
        (**iter).draw();
        (**iter).getCenter().label(i++, "n");
    }
    if (!makeTabs) return;
    return;
}
```

1507. Icosahedron.

1508. Icosahedron class definition.

```cpp
class class Icosahedron 1508 =
    class Icosahedron : public Polyhedron {
        protected: static const real dihedralAngle; /* In radians! */
        real triangleRadius;
        public: (Declare Icosahedron functions 1511)
    }
```

This code is used in sections 1534 and 1535.

1509. Define static const Icosahedron data members.

```cpp
const static const Icosahedron data members 1509 =
    const real Icosahedron::dihedralAngle = PI - asin(2.0/3.0);
```

This code is used in section 1534.

1510. Constructors and setting functions.

1511. Default constructor. (No arguments.)

```cpp
(Declare Icosahedron functions 1511 =
    Icosahedron());
```

See also sections 1514, 1516, and 1518.

This code is used in section 1508.
1512.  
(Define Icosahedron functions 1512) ≡  
Icosahedron::Icosahedron()  
{  
on_free_store = false;  /* from Solid. */  
do_output = true;  
faces = 20;  /* from Solid_Faced. */  
vertices = 12;  
edges = 30;  
center = INVALID_POINT;  /* from Polyhedron. */  
number_of_polygon_types = 1;  
face_radius = edge_radius = vertex_radius = INVALID_REAL;  
triangle_radius = INVALID_REAL;  
}  
See also sections 1515, 1517, and 1519.  
This code is used in section 1534.  

1513. Center, diameter of triangle, and angles.  


--- Log ---  
[LDF 2002.10.16] Defined this function.  

(Declare Icosahedron functions 1511) ≡  
Icosahedron(const Point &p, const real diameter_of_triangle, real angle_x = 0, real angle_y = 0, real angle_z = 0);
1515.

(Define Icosahedron functions 1512) 

Icosahedron::Icosahedron(const Point &p, const real triangle_diameter, real angle_x, real angle_y, real angle_z)

{ bool DEBUG = false;  /* true */ if (DEBUG) cout << "Entering Icosahedron::Icosahedron().\n" << flush;
on_free_store = false;  /* from Solid */
do_output = true;  
faces = 20;  /* from Solid_Faced */
vertices = 12;
edges = 30;
number_of_polygon_types = 1;
#endif

triangle_radius = triangle_diameter / 2.0;
int i;  reg_polygons.push_back ( create_new < Reg_Polygon > (0) ) ;
reg_polygons.front().set(origin, 3, triangle_diameter);
Point pts[7];
for (i = 0; i < 3; ++i)  pts[i] = reg_polygons.front().get_point(i);
for (i = 1; i < 6; ++i)  { reg_polygons.push_back ( create_new <
   Reg_Polygon > (reg_polygons.front()) ); }  reg_polygons[1].shift(pts[2] - pts[1]);
reg_polygons[3].shift(pts[0] - pts[2]);
reg_polygons[4].shift(pts[0] - pts[1]);
if (DEBUG) origin.label("0", "n");
reg_polygons[5].rotate(0, 180);
reg_polygons[5].shift(pts[1] - reg_polygons[5].get_point(0));  for (i = 6; i < 10; ++i)  {
   reg_polygons.push_back ( create_new < Reg_Polygon > (reg_polygons[5]) );  }
reg_polygons[7].shift(pts[1] - pts[0]);
reg_polygons[8].shift(pts[2] - pts[0]);
pts[3] = reg_polygons[2].get_point(1);  pts[4] = reg_polygons[2].get_point(0);
pts[5] = reg_polygons[1].get_point(2);
pts[6] = reg_polygons[1].get_point(0);
real angle = 180.0 - (dihedraAngle * 180.0/PI);
*reg_polygons[9] = *reg_polygons[8] = reg_polygons[7].rotate (pts[3], pts[5], -angle);
*reg_polygons[3] = *reg_polygons[4] = rotate (pts[4], pts[6], angle);
*reg_polygons[2] = reg_polygons[7].rotate (pts[1], pts[4], -angle);
   -angle);
*reg_polygons[1] = reg_polygons[9].rotate (pts[2], pts[6], angle);
   pts[0], angle);  for (i = 10; i < 20; ++i)  { reg_polygons.push_back ( create_new <
   Reg_Polygon > (reg_polygons[i - 10]) );
   reg_polygons.back().rotate(180);
reg_polygons.back()\-shift(0,12); }
if (DEBUG) {
    reg_polygons[4]\-dotlabel();
    reg_polygons[19]\-dotlabel();
}

real y\_shift = reg_polygons[4]\-get\_point(0).y() - reg_polygons[19]\-get\_point(0).y();
for (i = 10; i < 20; ++i) reg_polygons[i]\-shift(0,y\_shift);
center = reg_polygons[0]\-get\_center().mediate(reg_polygons[10]\-get\_center());
for (i = 0; i < 20; ++i) reg_polygons[i]\-shift(-center);
center = origin;
if (angle.x \neq 0 \lor angle.y \neq 0 \lor angle.z \neq 0)
    for (vector<Reg\_Polygon\&>\::iterator iter = reg_polygons.begin(); iter \neq reg_polygons.end();
        ++iter) (**iter).rotate(angle.x, angle.y, angle.z);
if (p \neq origin) {
    for (vector<Reg\_Polygon\&>\::iterator iter = reg_polygons.begin(); iter \neq reg_polygons.end();
        ++iter) {
        (**iter).shift(p - center);
    }
    center = p;
} if (DEBUG) {
    cout << "Exiting Icosahedron::Icosahedron() \n" << flush;
} return;

1516. Get net.

Log

[LDf 2002.08.12] Removed center\_0 argument.
[LDf 2003.08.27] Added size\_t \_triangles\_size and added size\_t \_i to a for loop, where it's compared to \_triangles\_size. This occurs in debugging code.

(Declare Icosahedron functions 1511) \+\=

static vector<Reg\_Polygon\&> get\_net(const real \_triangle\_diameter, bool do\_half = false);
1517.
(Define Icosahedron functions 1512 ) +
vector ( Reg_Polygon ) Icosahedron::get_net ( const real triangle_diameter , bool do_half ) { bool 
  DEBUG = false ; /* true */
  vector ( Reg_Polygon ) triangles ;
  int i ; for ( i = 0 ; i < 10 ; ++ i ) triangles . push_back ( new < Reg_Polygon > ( 0 ) ) ;
  /* The bottom left triangle. */
  triangles [ 0 ] = set ( origin , 3 , triangle_diameter , 0 , 180 ) ;
  Point pts [ 4 ] ;
  pts [ 0 ] = triangles [ 0 ] . get_point ( 0 ) ;
  pts [ 1 ] = triangles [ 0 ] . get_point ( 1 ) ;
  pts [ 2 ] = triangles [ 0 ] . get_point ( 2 ) ;
  * triangles [ 1 ] = * triangles [ 0 ] ;
  triangles [ 1 ] = rotate ( 0 , 180 ) ;
  triangles [ 3 ] = triangles [ 1 ] . get_point ( 0 ) ;
  * triangles [ 2 ] = * triangles [ 0 ] ;
  for ( i = 0 ; i < 4 ; ++ i ) {
    * triangles [ i + 1 ] = * triangles [ i ] ;
  }
  if ( DEBUG )
    for ( i = 0 ; i < 4 ; ++ i ) pts [ i ] . dotlabel ( i ) ;
  if ( do_half != true )
    for ( i = 0 ; i < 10 ; ++ i ) {
      for ( int j = 0 ; j < 3 ; j ++ )
        if ( triangles [ i ] . get_point ( j ) . get_y () != 0 )
          cerr << "ERROR! I have Icosahedron::get net () \n" << "y-coordinate! = 0! \n" << 
            "You'd better fix this! \n" << flush ;
    }
  return triangles ;
} / * Do the second half. */
if ( DEBUG )
  cout << "Doing the second half. \n" ;
  for ( i = 0 ; i < 10 ; ++ i ) triangles . push_back ( new < Reg_Polygon > ( 0 ) ) ;
  for ( int j = 8 ; j < 16 ; j ++ = 4 )
    for ( i = 0 ; i < 4 ; ++ i ) {
      * triangles [ j + i ] = * triangles [ j - 4 + i ] ;
    }
  if ( DEBUG )
    size_t triangles . size = triangles . size () ;
  for ( size_t i = 0 ; i < triangles . size ; ++ i )
    if ( triangles [ i ] . size () > 0 ) triangles [ i ] . get_center () . label ( i , "") ;
}
for (i = 0; i < 20; ++i) {
    for (int j = 0; j < 3; j++)
        if (triangles[i].get_point(j).get_y() != 0)
            cerr << "ERROR! Icosahedron::get_net(): \n" << "y-coordinate != 0! \n" << "You'd better fix this! \n" << flush;
}

return triangles; }

1518. Draw net.  TO DO: Add parallel projections onto planes other than the x-y plane.

Log

[LDF 2002.11.10.] Added this function.  portrait works, unlike *Dodecahedron::draw_net()polyhed.web* (as of this date).
[LDF 2002.08.12.] Changed, so that net is drawn in x-z plane.
[LDF 2002.08.12.] This function now returns before the code for making the tabs can be executed, because it doesn’t work yet.  TO DO: Write code for tabs.

(Declare *Icosahedron* functions 1511) ➞

  static void draw_net(const real triangle_diameter, bool portrait = true, bool make-tabs = true);
1519.

(Define Icosahedron functions 1512 ) +

void Icosahedron::draw_net (const real triangle_diameter, bool portrait, bool make_tabs)
{
    vector <Reg_Polygon *> v = get_net (triangle_diameter);
    int i = 0;
    for (vector <Reg_Polygon *> :: iterator iter = v.begin(); iter != v.end(); ++iter) {
        if (portrait) (**iter).rotate (0, 90);
        (**iter).get_center().label(i++, "n");
        (**iter).draw ();
    }
    return; /* Delete, when I start writing code for tabs. [LDF 2002.08.12. */
    if (~make_tabs) return;
    Path p[11];
    Point pts[11];
    #if 0
    v[0]=dotlabel ();
    #endif
    pts[0] = v[0]->get_point (0);
    pts[1] = v[0]->get_point (1);
    pts[2] = v[0]->get_point (2);
    pts[3] = pts[0].mediate (pts[1], 1);
    pts[4] = pts[1].mediate (pts[0], 1);
    #if 0
    pts[3].dotlabel (2);
    pts[4].dotlabel (3);
    #endif
    pts[5] = pts[0].mediate (pts[3], 1);
    pts[6] = pts[1].mediate (pts[4], 1);
    pts[7] = pts[3];
    pts[7].shift (0, 0, 1);
    pts[8] = pts[4];
    pts[8].shift (0, 0, 1);
    pts[5].rotate (pts[3], pts[7], 90);
    pts[6].rotate (pts[8], pts[4], 90);
    #if 0
    pts[5].dotlabel (4);
    pts[6].dotlabel (5);
    #endif
    pts[9] = pts[5].mediate (pts[6], 1);
    pts[10] = pts[6].mediate (pts[5], 1);
    #if 0
    pts[9].dotlabel (8);
    pts[10].dotlabel (9);
    #endif
    p[0].set ("") + true, &pts[3], &pts[9], &pts[10], &pts[4], 0);
    p[1].shift (v[4] - get_point (1) - pts[1]);
    p[2].shift (v[8] - get_point (1) - pts[1]);
    p[3].shift (v[12] - get_point (1) - pts[1]);
    p[4].shift (v[16] - get_point (1) - pts[1]);
\( p[5].\ shift( v[18].\ get\_point(1) - \text{pts}[1]); \)
\( p[6] = p[0]; \)
\( p[5].\ rotate( 0, 0, 240); \)
\( p[6].\ rotate( \text{pts}[0], \text{pts}[2]); \)
\#if 0
\( v[3].\ dot\_label(); \)
\#endif
\( p[6].\ shift( v[3].\ get\_point(0) - \text{pts}[2]); \)
\( p[7].\ shift( v[7].\ get\_point(0) - \text{pts}[2]); \)
\( p[8].\ shift( v[11].\ get\_point(0) - \text{pts}[2]); \)
\( p[9].\ shift( v[15].\ get\_point(0) - \text{pts}[2]); \)
\( p[10].\ shift( v[19].\ get\_point(0) - \text{pts}[2]); \)
\( \text{for} \ (i = 0; i < 11; i++) \ p[i].\ draw(); \)
\( \text{return}; \)
}

### 1520. Semi-Regular Archimedean Polyhedra.

#### 1521. Truncated Octahedron.

(Define class Truncated Octahedron 1522) \equiv

```cpp
class Truncated Octahedron : public Polyhedron {
  protected:
    static const real angle hex_square; /* In radians! */
    static const real angle hex_hex; /* In radians! */
    real hexagon radius;
  public:
    // Declare Truncated Octahedron functions 1525
};
```

This code is used in sections 1534 and 1535.

#### 1522. Define static const Truncated Octahedron data members.

(Define static const Truncated Octahedron data members 1523) \equiv

```cpp
const real Truncated Octahedron::angle hex_square = (125 + (16.0/60.0)) * (PI/180.0);
const real Truncated Octahedron::angle hex_hex = (109 + (28.0/60.0)) * (PI/180.0);
```

This code is used in section 1534.

#### 1523. Constructors and setting functions.

Log

[LDF 2003.04.15.] Commented-out the constructors and \textit{get\_net()}. They made use of the fact, which is no longer true, that \texttt{Rectangle} was formerly derived from \texttt{Reg\_Polygon}. Now that \texttt{Rectangle} is derived from \texttt{Path}, some of the code in these functions doesn't work. TO DO: Fix these functions!

---

#### 1525. Default constructor. (No arguments.)

(Declare Truncated Octahedron functions 1525) \equiv

```
Truncated Octahedron();
```

See also sections 1528 and 1530.

This code is used in section 1522.
1526.
\begin{verbatim}
(Define Trunc_Octahedron functions 1526) \equiv

Trunc_Octahedron::Trunc_Octahedron ()
{
    on_free_store = false;  /* from Solid. */
    do_output = true;
    faces = 14;  /* from Solid_Faced. [LDF 2002.10.29.] Truncated octahedrons consist of 6 squares
    and 8 hexagons. */
    vertices = 24;
    edges = 36;
    center = INVALID_POINT;  /* from Polyhedron. */
    number_of_polygon_types = 2;
    face_radius = edge_radius = vertex_radius = INVALID_REAL;
    hexagon_radius = INVALID_REAL;
}
\end{verbatim}
See also sections 1529, 1531, and 1532.
This code is used in section 1534.

1527.  Center, diameter of hexagon, and angles.

1528.  Constructor.

\begin{verbatim}
[LDf 2002.11.08.] Added this function.
\end{verbatim}

\begin{verbatim}
(Declare Trunc_Octahedron functions 1525) \equiv

#if 0
Trunc_Octahedron(const Point &p, const real diameter_of_hexagon, real angle_x = 0, real
angle_y = 0, real angle_z = 0);
#endif
\end{verbatim}
1529.
(Define Trunc_Octahedron functions 1526) +*

#if 0
Trunc_Octahedron::Trunc_Octahedron(const Point &p, const real hexagon_diameter, real angle_x, real angle_y, real angle_z) { bool DEBUG = false; /* true */
if (DEBUG) cout << "Entering Trunc_Octahedron::Trunc_Octahedron().\n" << flush;
center = p;
on_free_store = false; /* from Solid. */
do_output = true; /* START HERE. TO DO: Must calculate these! */
face_radius = 0;
edge_radius = 0;
vertex_radius = 0;
faces = 14; /* from Solid_Faced. [LDF 2002.10.29.] Truncated octahedrons consist of 6 squares
and 8 hexagons. */
vertices = 24;
edges = 36;
number_of_polygon_types = 2;
hexagon_radius = hexagon_diameter/2.0;
reg_polygons = get_net(hexagon_diameter, true);
Point pts[24];
int i;
for (i = 0; i < 6; i++) pts[i] = reg_polygons[0].get_point(i);
#endif

real angle_h = 180.0 - (angle_hex * 180.0/PI);
reg_polygons[1].rotate(pts[4], pts[5], angle_h);
reg_polygons[2].rotate(pts[3], pts[2], -angle_h);
reg_polygons[3].rotate(pts[0], pts[1], angle_h);
real angle_h = 180.0 - (angle_hex_square * 180.0/PI);
reg_polygons[4].rotate(pts[0], pts[5], angle_h);
reg_polygons[5].rotate(pts[3], pts[2], angle_h);
reg_polygons[6].rotate(pts[3], pts[4], angle_h);

/* [LDF 2002.11.07.] Do something about these comments! */ /* 7 */
reg_polygons.push_back ( create_new < Reg_Polygon > (reg_polygons[0]) ); /* 8 */
reg_polygons.push_back ( create_new < Reg_Polygon > (reg_polygons[1]) ); /* 9 */
reg_polygons.push_back ( create_new < Reg_Polygon > (reg_polygons[2]) ); /* 10 */
reg_polygons.push_back ( create_new < Reg_Polygon > (reg_polygons[3]) ); /* 11 */
reg_polygons.push_back ( create_new < Reg_Polygon > (reg_polygons[4]) ); /* 12 */
reg_polygons.push_back ( create_new < Reg_Polygon > (reg_polygons[5]) ); /* 13 */
reg_polygons.push_back ( create_new < Reg_Polygon > (reg_polygons[6]) );

Transform t;
t.rotate(180);
t.shift(0, 15);
for (i = 7; i < 14; i++) *reg_polygons[i] = t;
t.reset();
for (i = 7; i < 14; i++) *reg_polygons[i] += t;
i = 0;
for (vector<Reg_Polygon *>::iterator iter = reg_polygons.begin(); iter != reg_polygons.end();
    ++iter) {
    if (center != origin) (**iter).shift(center);
}
if (DEBUG) {
    cout << "Exiting Trunc_Octahedron::Trunc_Octahedron." << endl << endl << flush;
    return;
}

1530. Get net.

[Log] [LDF 2002.11.08.] Added this function.
[Log] [LDF 2002.08.12.] Removed center argument.

(Declare Trunc_Octahedron functions 1525 ) +=

# if 0
static vector<Reg_Polygon *> get_net(const real hexagon_diameter, bool do_half = false);
# endif
1531.
(Define Trunc_Octahedron functions 1526) +≡
#if 0

vector<Reg_Polygon *> Trunc_Octahedron::get_net(const real hexagon_diameter, bool do_half){
  bool DEBUG = false;    /* true */
  vector<Reg_Polygon *> reg_polygons;
  int i, reg_polygons.push_back ( create_new < Reg_Polygon > (0) ) ;
  reg_polygons[0]-set (origin, 6, hexagon_diameter);
  Point pts[24];
  for ( i = 0; i < 6; i++) pts[i] = reg_polygons[0]-get_point(i);
  reg_polygons.push_back ( create_new < Reg_Polygon > (reg_polygons[0]) ) ;
  reg_polygons[1]-shift (pts[4] − pts[2]); reg_polygons.push_back ( create_new < Reg_Polygon > (reg_polygons[0]) ) ;
  reg_polygons[2]-shift (pts[3] − pts[5]); reg_polygons.push_back ( create_new < Reg_Polygon > (reg_polygons[0]) ) ;
  reg_polygons[3]-shift (pts[1] − pts[3]);
  real side_length = pts[5].get_x() − pts[0].get_x();
  pts[6] = pts[0];
  pts[6].shift(0,0, side_length);
  pts[7] = pts[5];
  pts[7].shift(0,0, side_length);
  Rectangle r(pts[0],pts[5],pts[7],pts[6]); reg_polygons.push_back ( create_new < Reg_Polygon > (r) ) ;
  pts[8] = pts[2];
  pts[9] = pts[1];
  pts[9].shift(0,1);
  pts[8].rotate (pts[9],pts[1],−90);
  pts[10] = pts[8];
  r.set (pts[10],pts[2],pts[1],pts[8]); reg_polygons.push_back ( create_new < Reg_Polygon > (r) ) ;
  pts[12] = pts[4];
  pts[12].shift (0,1);
  pts[11].rotate (pts[4],pts[12],90);
  r.set (pts[11],pts[4],pts[3],pts[13]); reg_polygons.push_back ( create_new < Reg_Polygon > (r) ) ;
  if (do_half) return reg_polygons;
#endif
1532. [LDF 2002.11.08.] If we just want the net, \texttt{reg\_polygons[5]} and \texttt{reg\_polygons[6]} must be changed, because I've made the net a bit differently from the way it's done in Cundy. Get reference!! Page 104. I made two of the squares slanted, in order to avoid having to rotate them twice.

\begin{verbatim}
(Define Trunc\_Octahedron functions 1526) \equiv
#if 0
pts[14] = reg\_polygons[4]-get\_point(2);
pts[15] = reg\_polygons[4]-get\_point(3);
reg\_polygons[5]-shift(pts[1] - pts[14]);
reg\_polygons[6]-shift(pts[4] - pts[15]); reg\_polygons.push\_back ( create\_new < Reg\_Polygon > (reg\_polygons[0]) ); /* 7 */
reg\_polygons.push\_back ( create\_new < Reg\_Polygon > (reg\_polygons[4]) ); /* 8 */
reg\_polygons[9]-shift(pts[4] + pts[0] - (2 + pts[2])); reg\_polygons.push\_back ( create\_new < Reg\_Polygon > (reg\_polygons[0]) ); /* 10 */
reg\_polygons.push\_back ( create\_new < Reg\_Polygon > (reg\_polygons[0]) ); /* 11 */
reg\_polygons.push\_back ( create\_new < Reg\_Polygon > (reg\_polygons[0]) ); /* 12 */
reg\_polygons.push\_back ( create\_new < Reg\_Polygon > (reg\_polygons[0]) ); /* 13 */
return reg\_polygons;
#endif

1533. Putting polyhedra together.

1534. This is what's compiled.

\begin{verbatim}
(Include files 6)
(Version control identifier 5)
(Define class Polyhedron 1472)
(Define class Tetrahedron 1477)
(Define static const Tetrahedron data members 1478)
(Define class Dodecahedron 1494)
(Define static const Dodecahedron data members 1495)
(Define class Icosahedron 1508)
(Define static const Icosahedron data members 1509)
(Define class Trunc\_Octahedron 1522)
(Define static const Trunc\_Octahedron data members 1523)
(Define Polyhedron functions 1474)
(Define Tetrahedron functions 1481)
(Define Dodecahedron functions 1498)
(Define Icosahedron functions 1512)
(Define Trunc\_Octahedron functions 1526)
\end{verbatim}
§1535. This is what’s written to polyhed.h,

(polyhed.h §1535) ≡
(Define class Polyhedron 1472)
(Define class Tetrahedron 1477)
(Define class Dodecahedron 1494)
(Define class Icosahedron 1508)
(Define class Trunc_Octahedron 1522)

1536. Parsing (parser.web).

Log

Removed the code from this file. I plan to use Bison for making the parser. [LDF 2003.08.25.]
[LDF 2003.11.12.] Removed the version control identifiers from the CWEB files for the distribution of
3DLDF 1.1. They’re still used in my development versions.
[LDF 2003.12.10.] Put the version control identifiers back into my release versions for 3DLDF 1.1.4. I’ve
already put some of them back in, now I’m doing the rest of them. However, the release versions are now in
their own RCS repository.

(Version control identifier 5) +≡
  static string res_id = "$Id: parser_web, v.1.4, 2004/01/12, 21:30:44, ifinsto1, Exp1$";

1537. Include files. map.h is currently not needed, but I plan to use it for the input routine. [LDF 2004.01.06]

(Include files 6) +≡
#include "loader.h"
#include "pspglb.h"
#include "io.h"
#include "shapes.h"
#include "pictures.h"
#include "points.h"
#include "lines.h"
#include "planes.h"
#include "paths.h"
#include "curves.h"
#include "polygons.h"
#include "rectangs.h"
#include "ellipses.h"
#include "circles.h"
#include "patterns.h"
#include "solids.h"
#include "solfaced.h"
#include "cuboid.h"
#include "polyhed.h"
#include "utility.h"

1538. Parse.

(Declare parser functions 1538) ≡
This code is used in section 1542.
1539.  
Define parser functions 1539 \equiv
This code is used in section 1541.

1540.  Putting the parser together.

1541.  This is what’s compiled.
\begin{verbatim}
(Include files 6)
(Version control identifier 5)
(Define parser functions 1539)
\end{verbatim}
1542. This is what’s written to parser.h.

Declaring parser functions 1538

1543. Main (main.web).

Log

[LDF 2002.11.18.] Changed name of this file from persp.web to main.web. It now has fewer than 8 letters and can be used under DOS.
[LDF 2003.08.29.] Moved getopt.h from loader.web to here, because it’s only used here. TO DO: Move the rest of the include commands to the files where they’re needed, and get rid of loader.web.
[LDF 2003.11.12.] Removed the version control identifiers from the CWEB files for the distribution of 3DLDF 1.1. They’re still used in my development versions.
[LDF 2003.12.01.] Put the version control identifiers back into the release versions, because I’ve put them in their own RCS repository.

Version control identifier 5

static string res_id = "$Id: main.web,v 1.42 2004/01/12 21:30:34.1 fsinsto1 Exp "$;

1544. Include files. getopt.h is included for processing the command line options. [LDF 2003.08.14.]

Include files 6

#include "loader.h"
#include <bitset>
#ifdef __GNUC__
#include <getopt.h>
#endif
#include "pspglb.h"
#include "creativew.h"
#include "gslmp.h"
#include "io.h"
#include "colors.h"
#include "transfo.h"
#include "shapes.h"
#include "pictures.h"
#include "points.h"
#include "lines.h"
#include "planes.h"
#include "paths.h"
#include "curves.h"
#include "polygons.h"
#include "rectangls.h"
#include "ellipses.h"
#include "circles.h"
#include "patterns.h"
#include "solids.h"
#include "solfaced.h"
#include "cuboid.h"
#include "polyhed.h"
#include "utility.h"
#include "parser.h"
#include "examples.h"
1545. Get input. I plan to use Flex and Bison to handle scanning and parsing input. I haven’t started work on this yet. [LDF 2003.08.20.]

1546. Actions in main.

```cpp
using namespace Colors;
using namespace Projections; if (ldf_real<float> MAX_REAL = System::get_second_largest <
  float > FLT_MAX, false); else if (ldf_real<double> MAX_REAL = System::get_second_largest <
  double > (DBL_MAX, false);

MAX_REAL_SQRT = sqrt(MAX_REAL);

vector(const Color *) v;
  v.push_back(&red);  v.push_back(&green);
  v.push_back(&blue);  v.push_back(&cyan);  v.push_back(&yellow);
#include
  v.push_back(&magenta);  v.push_back(&orange);
  v.push_back(&violet);  v.push_back(&yellow_green);
  v.push_back(&green_yellow);  v.push_back(&blue_violet);
  v.push_back(&red);
  v.push_back(&green);
  v.push_back(&blue);  v.push_back(&cyan);  v.push_back(&yellow);
  v.push_back(&magenta);  v.push_back(&orange);
  v.push_back(&violet);
  v.push_back(&yellow_green);
  v.push_back(&green_yellow);  v.push_back(&blue_violet);
  v.push_back(&violet_red);
#endif
```

See also section 1547.

This code is used in section 1557.
§1547. Your code here!

\[\begin{aligned}
\text{Actions in main 1546} & \equiv \\
\text{beginfig}(1); \\
\text{Point } & p; \\
\text{Point } & q(1,1,1); \\
\text{Circle } & c(p,2,90); \\
\text{c.filldraw } & (); \\
p.\text{draw } & (q); \\
\text{current\_picture.\_output } & (); \\
\text{endfig } & (); \\
\end{aligned}\]

1548. **Process command line options.** This section includes one of (currently) two other sections, one for the GCC/Linux version and one for the DEC version. The section to be included is chosen by testing whether preprocessor macros are defined or not. Put another way, the command line option processing code is conditionally compiled. [LDF 2003.08.14]

The problem is that, unlike GCC, the DEC C++ compiler doesn’t support long command line options, so I have to implement the command line option processing code separately for each version. [LDF 2003.08.14]

---

[LDF 2003.08.14] Added this section.

\[\begin{aligned}
\text{Process command line options 1548} & \equiv \\
\text{if} & \text{defined } \_\_\text{GNU} \_\_ \\
\text{GCC command line option processing 1549} & \text{else} \\
\text{if} & \text{defined } \_\_\text{DECCXX} \\
\text{DEC command line option processing 1551} & \text{endif} \\
\text{endif} \\
\text{This code is used in section 1555.} \\
\end{aligned}\]
GCC version of command line processing.

```c
1549. bool DEBUG = false; /* true */
int option_ctr;
int digit_optind = 0;
const unsigned short HELP_INDEX = 0;
const unsigned short SILENT_INDEX = 1;
const unsigned short VERBOSE_INDEX = 2;
const unsigned short VERSION_INDEX = 3;
static struct option long_options[] = {
    "help", 0, 0, 0,
    "silent", 0, 0, 0,
    "verbose", 0, 0, 0,
    "version", 0, 0, 0,
};
int option_index = 0;
int this_option_optind = optind ? optind : 1;
while (1) {
    option_ctr = getopt_long_only(argc, argv, "hv", long_options, &option_index);
    if (DEBUG) {
        cout << "option_ctr=" << option_ctr << endl << flush;
        cout << "option_index=" << option_index << endl << flush;
    }
    if (option_ctr == -1) {
        if (DEBUG) cout << "No more options." << endl << endl << flush;
        break;
    }
    if (option_ctr == 0) {
        if (DEBUG) {
            cout << "long_options[option_index] name:
            if (optary) cout << "\u with arg" << optary << endl;
        }
    }
    if (option_index == HELP_INDEX) {
        cout << "3DLDF\u2013Version\u2013" << VERSION_3DLDF << "-u" << COPYRIGHT_3DLDF << endl << "Valid options for 3DLDF are:"
        "-help: prints this message and exits"
        "with return value 0."
        "-version: prints the version number of 3DLDF"
        "-silent: suppress some output to standard output"
        "-verbose: causes status information to be printed"
        "-standard output: output to standard output and end"
        "exits with return value 0."
        if (DEBUG) cout << "Exiting with return value 0." << endl << flush;
        exit(0);
```
SILENT_GLOBAL = true;
}
else if (option_index == VERBOSITY_INDEX) {
    if (DEBUG) cout << "Setting VERBOSITY_INDEX to true.\n" << endl;
    VERBOSITY_INDEX = true;
}
else if (option_index == VERSION_INDEX) {
    cout << "3DLDF Version:\n" << VERSION_3DLDF << ".\n" << COPYRIGHT_3DLDF << endl << flush;
    if (DEBUG) cout << "Exiting with return value 0.\n" << endl << flush;
    exit(0);
}
else {
    cerr << "This can't happen!\n" << "option_index has invalid value: \n" << option_index << \
        "Will try to continue.\n" << endl << endl << flush;
}
}
else if (option_str == '?' ) {
    cerr << "getopt_long() returned ambiguous match. Breaking.\n" << endl << endl << flush;
    break;
}
else {
    cerr << "getopt_long() returned invalid option.\n" << endl << flush;
}
if (DEBUG) cout << "***************\n\n";
} /* while */
if (optind < argc) {
    cout << "non-option ARGV-elements: \n";
    while (optind < argc) cout << argv[optind++] << endl;
    cout << endl << flush;
}
if (DEBUG) {
    cout << "Exiting (Debugging command line option processing.)\n" << endl << endl << flush;
    exit(0);
}
} /* End of group */

This code is used in section 1548.

1550. DEC version,
1551. This section doesn’t contain any code yet.

[LDF 2003.08.14.] Added this section.

{DEC command line option processing 1551} \equiv /* Do nothing. */

This code is used in section 1548.

1552. Print version, copyright, and license information. The version, copyright, and license information is printed to standard output when 3DLDF is run, unless the "--silent" option was used. The code for this differs for the GCC 2.95/Linux version on the one hand, and the other versions (currently, GCC 3.3/Linux and DEC) on the other. The reason for this is, that GCC 2.95 doesn’t handle stream formatting in the same way as the others. I assume that the others adhere to the standard and that GCC 2.95 doesn’t, but I haven’t checked this. At any rate, the non-GCC 2.95 version corresponds to what Stroustrup describes in The C++ Programming Language.

[LDF 2003.08.14.] Added this section.

{Print version, copyright, and license information 1552} \equiv
if (¬SILENT_GLOBAL) {
  #ifdef LDF_GCC_2.95
  (GCC 2.95 print version, copyright, and license information 1553)
  #else
  (GCC 3.3 and DEC print version, copyright, and license information 1554)
  #endif
}

This code is used in section 1556.

1553. GCC 2.95 version.

[LDF 2003.08.14.] Added this section.

{GCC 2.95 print version, copyright, and license information 1553} \equiv
  cout.setf (ios::fixed, ios::floatfield);
  cout.precision(1);
  cout << "3DLDF_VERSION" << VERSION_3DLDF << ": " << COPYRIGHT_3DLDF << endl << DISCLAIMER_3DLDF << endl << endl << flush;
  cout.setf (ios::fmtflags(0), ios::floatfield); /* Reset to defaults. [LDF 2003.08.14.] */
  cout.precision(6);

This code is used in section 1552.
1554. GCC 3.3 and DEC version.

[Log]

GCC 3.3 and DEC print version, copyright, and license information 1554) ≡

cout << "3DLDF\Version\n" << VERSION_3DLDF << "." << endl << COPYRIGHT_3DLDF << endl <<
DISCLAIMER_3DLDF << endl << endl << flush;
cout.precision(6);

This code is used in section 1552.

1555. Main itself.

[Main 1555) ≡

int main(int argc, char *argv[])
{
    (Process command line options 1548);
See also sections 1556 and 1557.
This code is used in section 1558.

[Log]

[Log]

Main 1555) +⇒

(Print version, copyright, and license information 1552);
1557. (Main 1555) +
    initialize_io("3DLDF.in.1df", "3DLDFput.mp", "3DLDFput.tex", argv[0]);
    Color::initialize_colors();
    (Actions in main 1546);
    write_footers();
    #if 0
    in_stream.close();
    #endif
    out_stream.close();
    tex_stream.close();
    if (~SILENT_GLOBAL) {
        cout << "Exiting\n3DLDF\nVersion\n" << VERSION_3DLDF << ".\n\n" << flush;
    }
    return (0); }

1558. Putting Main together. This is what's compiled.

   (Include files 6)
   (Version control identifier 5)
   (Main 1555)

1559. Appendices.

1560. References.


Knuth, Donald Ervin. The METAFONTbook. Computers and Typesetting; C. Addison Wesley Publishing

Knuth, Donald Ervin. TeX: The Program. Computers and Typesetting; B. Addison Wesley Publishing

Knuth, Donald E. The TEXbook. Computers and Typesetting A. Addison Wesley Publishing Company,
Inc. Reading, Massachusetts 1986.
Knuth, Donald E. and Silvio Levy. *The CWEB System of Structured Documentation.*
Version 3.64—February 2002.


Addison-Wesley.

Publishing Company.

1561. GNU Free Documentation License. The GNU Free Documentation License, which follows,
applies to this document.

GNU Free Documentation License
Version 1.2, November 2002

59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondary, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.
A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, TeXinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the
If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement. C. State on the Title page the name of the publisher of the Modified Version, as the publisher. D. Preserve all the copyright notices of the Document. E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy
of the Document, and likewise the network locations given in the Document for previous versions it was
based on. These may be placed in the “History” section. You may omit a network location for a work
that was published at least four years before the Document itself, or if the original publisher of the version
it refers to gives permission. K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve
the Title of the section, and preserve in the section all the substance and tone of each of the contributor
acknowledgements and/or dedications given therein. L. Preserve all the Invariant Sections of the Document,
unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of
the section titles. M. Delete any section Entitled “Endorsements”. Such a section may not be included in the
Modified Version. N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title
with any Invariant Section. O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections
and contain no material copied from the Document, you may at your option designate some or all of these
sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s
license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your
Modified Version by various parties—for example, statements of peer review or that the text has been approved
by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as
a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of
Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any
one entity. If the Document already includes a cover text for the same cover, previously added by you or by
arrangement made by the same entity you are acting on behalf of, you may not add another; but you may
replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names
for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms
defined in section 4 above for modified versions, provided that you include in the combination all of the
Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of
your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections
may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different
contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of
the original author or publisher of that section if known, or else a unique number. Make the same adjustment
to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents,
forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and
any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License,
and replace the individual copies of this License in the various documents with a single copy that is included
in the collection, provided that you follow the rules of this License for verbatim copying of each of the
documents in all other respects.
You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Deductions”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents
To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

(GNU Free Documentation License 1561) ≡ /* This section contains no C++ code. */

This code is cited in section 1.

1562. GNU General Public License. The GNU General Public License, which follows, applies to the program 3DLDF described in this document.

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.
Also, for each author’s protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors’ reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone’s free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The “Program”, below, refers to any such program or work, and a “work based on the Program” means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term “modification”.) Each licensee is addressed as “you”.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from a Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program’s source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice, this Public License, and an announcement stating that you do not accept the limitations of this Public License. If you distribute the modified program, consider giving them a choice to run from either the unmodified or modified version. The source code for the modified program shall be made available to them according to subclause a).

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the
distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
   
a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   
b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   
c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License.
If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and “any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO}
LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

(one line to give the program’s name and a brief idea of what it does.)
Copyright (C) (year) (name of author)

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program ‘Gnomovision’ (which makes passes at compilers) written by James Hacker.

(signature of Ty Coon), 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

(GNU General Public License 1562) Ξ /* This section contains no C++ code. */
This code is cited in section 1.
1563. Index. [LDF 2002.10.09.] The way CWEAVE handles indexing is not ideal for C++. It doesn’t index identifiers that include non-alphanumeric characters, so that neither Path :: draw () nor operator = ( ) are indexed automatically. Nor is there any indication of whether an identifier refers to a variable or a function.

I have added indexing commands in the source files for operators and class member functions. However, the alphabetization routine is naïve and doesn’t ignore the characters of the \TeX macros that I use for formatting the index entries, so the order of the entries is a bit peculiar. For example, \_func \z ( \_func is for class functions, and \_func is for operators that aren’t members of a class).

First come the index entries which start with “??” and “!!!”. These are followed by the non-operator class member functions for all of the classes. Then come the class member operators for all of the classes, followed by the non-class operators. Currently, I’m only putting index entries in by hand where the class member functions and operators are declared.

[LDF 2002.10.11.] Another problem is that “operator?” and “operator?=” use the italic ampersand. It would be possible to fix this, but slightly tricky. TO DO: Fix this!

BUG FIX: 18, 195, 218, 448, 593, 700, 702, 871, 1388, 1516.

!!: 95, 99, 155, 156, 158, 166, 174, 193, 207, 208, 226, 254, 297, 502, 504, 546, 547, 559, 572, 727, 834, 843, 906, 918, 940, 970, 1120, 1158, 1273, 1320, 1369, 1438, 1464, 1473, 1487, 1503, 1532.

!! BUG: 423, 581, 721.

!! KLUDGE: 19, 310, 317, 445, 475, 1505.

!! LOOK UP: 495.

!! NOTE: 1317.


!! URGENT: 700.

??: 158, 244, 353, 354, 530, 538, 590, 688, 700, 711, 721, 728, 835, 882, 1080, 1081.

Circle :: Circle: 1278, 1281.

Circle :: create_new_circle: 1286.

Circle :: get diameter: 1298.

Circle :: get radius: 1297.

Circle :: intersection points: 1300, 1302, 1304.

Circle :: is circular: 1295.

Circle :: set: 1283.

Color :: Color: 97, 99, 102, 107.

Color :: define_color_mp: 149.

Color :: get blue part: 143.

Color :: get green part: 142.

Color :: get name: 146.

Color :: get red part: 141.

Color :: get use name: 144.

Color :: initialize colors: 151.

Color :: is on free store: 138.

Color :: modify: 127.

Color :: set blue part: 133.

Color :: set green part: 131.

Color :: set name: 123.

Color :: set on free store: 121.

Color :: set red part: 129.

Color :: set use name: 125.


Color :: show: 135.

Cuboid: "Cuboid": 1464.

Cuboid: "Cuboid": 1455, 1457, 1459.

Dodecahedron: Dodecahedron: 1497, 1500.

Dodecahedron: draw net: 1505.


Ellipse :: 'Ellipse': 1156.

Ellipse :: angle point: 1201.

Ellipse :: dlabel: 1162.

Ellipse :: draw on rectangle: 1261.

Ellipse :: draw axis: 1146, 1149.

Ellipse :: get axis_h: 1197, 1199.

Ellipse :: get_axis_v: 1192, 1194.

Ellipse :: get center: 1177, 1179.

Ellipse :: get coefficients: 1174.

Ellipse :: get focus: 1182, 1184.

Ellipse :: in rectangle: 1259.

Ellipse :: intersection points: 1208, 1210, 1212.

Ellipse :: is cubic: 1167.

Ellipse :: is elliptical: 1164.

Ellipse :: is quadratic: 1166.

Ellipse :: is quartic: 1169.

Ellipse :: label: 1160.

Ellipse :: location: 1205.

Ellipse :: out rectangle: 1257.

Ellipse :: rotate: 1235, 1252, 1254.

Ellipse :: scale: 1237.

Ellipse :: set: 1151.

Ellipse :: shear: 1239.
Ellipse::shift_times: 1247, 1249.
Ellipse::shift: 1242, 1244.
Ellipse::sobe: 1171.
Focus::Focus: 602, 604, 609.
Focus::get_direction: 621.
Focus::get_distance: 622.
Focus::get_persp_element: 628.
Focus::get_persp: 627.
Focus::get_position: 620.
Focus::get_transform_element: 625.
Focus::get_transform: 624.
Focus::get_up: 623.
Focus::reset_angle: 615.
Focus::set: 606, 611.
Focus::show: 617.
Icosahedron::draw_net: 1518.
Icosahedron::get_net: 1516.
Icosahedron::Icosahedron: 1511, 1514.
Label::get_copy: 515.
Label::output: 516.
Line::get_distance: 648.
Line::get_path: 646, 978.
Line::Line: 639, 641.
Line::show: 652.
Path::Path: 726.
Path::align_with_axis: 790, 792, 794.
Path::append: 812.
Path::apply_transform: 783.
Path::clear: 728.
Path::dolebel: 877, 879.
Path::draw_help: 836, 838.
Path::draw: 818, 820.
Path::drawarrow: 827, 829.
Path::extract: 882.
Path::fill: 844, 846.
Path::filldraw: 849, 851.
Path::get_copy: 730.
Path::get_extremes: 888.
Path::get_last_point: 936.
Path::get_line_switch: 920.
Path::get_line: 976.
Path::get_maximum: 891.
Path::get_mean: 893.
Path::get_minimum: 889.
Path::get_normal: 940.
Path::get_plane: 946.
Path::get_point: 932, 934.
Path::get_size: 938.
Path::intersection_point: 964.
Path::is_cycle: 921.
Path::is_linear: 918.
Path::is_on_free_store: 914.
Path::is_planar: 916.
Path::label: 872, 874.
Path::output: 899.
Path::Path: 704, 707, 712, 717, 721.
Path::project: 785.
Path::reverse: 955, 959.
Path::rotate: 756, 762, 764.
Path::scale: 766.
Path::set_connectors: 751.
Path::set_cycle: 952.
Path::set_dash_pattern: 747.
Path::set_draw_color: 738, 740.
Path::set_extremes: 884.
Path::set_fill_color: 743, 745.
Path::set_fill_draw_value: 735.
Path::set_long_free_store: 732.
Path::set_pen: 749.
Path::set: 709, 714, 719.
Path::shear: 768.
Path::shift_times: 776, 778.
Path::shift: 771, 773.
Path::show_colors: 911.
Path::show: 909.
Path::size: 922.
Path::slope: 923.
Path::subpath: 925.
Path::suppress_output: 895.
Path::undraw: 855, 857.
Path::unfill: 863.
Path::unfilldraw: 866, 868.
Path::unsupport_output: 897.
Picture::clear: 300, 590.
Picture::kill_labels: 276.
Picture::output: 298, 299, 592, 598.
Picture::Picture: 263, 265.
Picture::reset_transform: 301.
Picture::rotate: 286, 288, 440.
Picture::scale: 280.
Picture::set_transform: 289.
Picture::shift: 283, 417.
Picture::show_transform: 295.
Picture::show: 293.
Picture::suppress_labels: 274.
Picture::unsupport_labels: 275.
Plane::get_distance: 677, 679.
Plane::intersection_line: 687.
Plane::intersection_point: 966.
Plane::Plane: 663, 665, 667.
Plane::show: 689.
Point::Point: 338.
Point::angle: 548.
Point::apply_transform: 448.
Polygon::get_center: 1022, 1024.
Polygon::intersection_points: 1028, 1037, 1039.
Polygon::rotate: 1047, 1050, 1052.
Polygon::scale: 1054.
Polygon::shear: 1056.
Polygon::shift_times: 1064, 1066.
Polygon::shift: 1059, 1061.
Polyhedron::intersection_points: 1473.
Rectangle::Rectangle: 1118.
Rectangle::corner: 1125.
Rectangle::is_rectangular: 1122.
Rectangle::mid_point: 1127.
Rectangle::Rectangle: 1103, 1106, 1111.
Rectangle::set: 1108, 1113.
Reg_C1 Plane_Curve::angle_point: 994.
Reg_C1 Plane_Curve::get_coefficients: 990.
Reg_C1 Plane_Curve::half: 1013.
Reg_C1 Plane_Curve::intersection_points: 997, 1008.
Reg_C1 Plane_Curve::is_cubic: 988.
Reg_C1 Plane_Curve::is_quadratic: 987.
Reg_C1 Plane_Curve::is_quartic: 989.
Reg_C1 Plane_Curve::location: 992.
Reg_C1 Plane_Curve::quarter: 1014.
Reg_C1 Plane_Curve::segment: 1011.
Reg_C1 Plane_Curve::solve: 991.
Reg_Polygon::draw_in_circle: 1091, 1092, 1309, 1310.
Reg_Polygon::draw_out_circle: 1095, 1096, 1312, 1313, 1314.
Reg_Polygon::get_radius: 1087.
Reg_Polygon::in_circle: 1089, 1307.
Reg_Polygon::out_circle: 1093, 1311.
Reg_Polygon::Reg_Polygon: 1073, 1076.
Reg_Polygon::set: 1079.
Solid::Solid: 1343.
Solid::apply_transform: 1388.
Solid::clear: 1383.
Solid::draw: 1423.
Solid::extract: 1404.
Solid::fill: 1426.
Solid::filldraw: 1429.
Solid::get_center: 1353.
Solid::get_circle_center: 1371.
Solid::get_circle_ptr: 1358.
Solid::get_copy: 1348.
Solid::get_ellipse_center: 1373.
Solid::get_ellipse_ptr: 1360.
Solid::get_extremes: 1408.
Solid::get_maximum_z: 1411.
Solid::get_mean_z: 1413.
Solid::get_minimum_z: 1409.
Solid::get_path_ptr: 1362.
Solid::get_rectangle_center: 1375.
Solid::get_rectangle_ptr: 1364.
Solid::get_reg_polygon_center: 1377.
Solid::get_reg_polygon_ptr: 1366.
Solid::get_shape_center: 1369.
Solid::get_shape_ptr: 1356.
Solid::is_on_free_store: 1379.
Solid::output: 1419.
Solid::rotate: 1399, 1401.
Solid::scale: 1390.
Solid::set_extremes: 1406.
Solid::set_on_free_store: 1350.
Solid::shear: 1392.
Solid::shift: 1395, 1397.
Solid::show: 1381.
Solid::Solid: 1336, 1338.
Solid::supress_output: 1415.
Solid::undraw: 1432.
Solid::unfill: 1435.
Solid::unfilldraw: 1438.
Solid::unsupress_output: 1417.
Tetrahedron::draw_net: 1491.
Tetrahedron::get_net: 1489.
Tetrahedron::set: 1487.
Transform::align_with_axis: 213, 423.
Transform::center: 180.
Transform::epison: 182.
Transform::get_element: 190.
Transform::inverse: 226, 232.
Transform::is_identity: 185, 187.
Transform::reset: 176.
Transform::rotate: 205, 211, 212, 439, 759.
Transform::scale: 195.
Transform::set_element: 178.
Transform::shear: 197.
Transform::shift: 200, 202, 203, 416.
Transform::show: 192.
Transform::Transform: 168, 170, 172.
Trunc_Octahedron::get_net: 1530.
Trunc_Octahedron::Trunc_Octahedron: 1525, 1528.
Circle::operator==: 1290, 1292.
Color::operator!=: 118.
Color::operator==: 116.
Color::operator=: 114.
Cuboid::operator==: 1466.
Ellipse::operator==: 1233.
Ellipse::operator!=: 1157.
Ellipse::operator+=(transform: 1231.
Line::operator==: 643.
Path::operator*=: 781.
Path::operator++: 781.
Path::operator+=: 781.
Path::operator=+: 799.
Path::operator==: 961.
Path::operator==: 700.
Path::operator+=: 805.
Path::operator+=: 810.
Picture::operator*=: 291.
Picture::operator+=: 269, 270, 272, 589.
Picture::operator==: 267.
Point::operator==: 569.
Point::operator==: 518, 529.
Point::operator*: 532.
Point::operator+=: 523.
Point::operator+: 521.
Point::operator-=: 527.
Point::operator-: 525, 536.
Point::operator=/: 538.
Point::operator/: 540.
Point::operator==: 559, 560, 567.
Point::operator:= 340.
Polygon::operator*: 1045.
Rectangle::operator*: 1119.
Reg_Polygon::operator*: 1070.
Solid::operator*: 1386.
Solid::operator+=: 1345.
Transform::operator*: 216, 218.
Transform::operator*: 221, 223.
Transform::operator+=: 174.
operator<<: 147, 480.
operator*: 534.
__CXX_PI: 52.
__DECCX: 6, 7, 11, 14, 19, 26, 52, 83, 323, 443, 1548.
__GNU__: 7, 24, 83, 323, 443, 1544, 1548.
__USE_STD_IIOSTREAM: 7.
__GNU_SOURCE: 7.
__ptr: 1352.
A: 1352.
am_coord: 390, 391.
an coord: 390, 391.
a.x: 650.
a.y: 650.
a.z: 650.
aa: 1222.
aaxis: 1106, 1107, 1108, 1109, 1149, 1150, 1151, 1152.
1427, 1430, 1433, 1436, 1439, 1547.

circles: 1333, 1339, 1346, 1347, 1356, 1357, 1359, 1369, 1370, 1382, 1384, 1387, 1389, 1405, 1407, 1419, 1420, 1422, 1424, 1425, 1427, 1428, 1430, 1431, 1433, 1434, 1436, 1437, 1439.

Circles: 1217, 1295, 1323, 1326.

classes: 700.
clear: 245, 266, 267, 277, 300, 344, 345, 514, 587, 590, 703, 727, 728, 729, 752, 1043, 1071, 1085, 1118, 1120, 1156, 1288, 1324, 1327, 1346, 1383, 1384, 1465.
close: 1557.
closed: 1011, 1012, 1013, 1014.
corrs: 178, 179, 190, 191.
col: 1044.
col_vec: 1502.


col_iter: 1327.
col_ptr: 1422, 1424, 1425, 1427.


cols: 1323, 1324, 1326, 1327.
column: 625, 626, 628, 629.

Compare_maximum_z: 497, 499, 596, 1420.

Compare_mean_z: 497, 500, 596.

Compare_minimum_z: 497, 498, 596.

Compiling: 1317.

congruent_flag: 1218.


contector_iter: 902, 903, 905, 907.
fixed: 83, 907, 1553.
floatfield: 83, 1553.
floats: 15, 27, 182, 546.
floor: 1224, 1293.
FLT_EXP: 70.
FLT_MAX: 7, 19, 66, 1546.
FLT_SIZE: 68.
fill_color: 844, 845, 846, 847, 848, 849, 850, 885, 852.
fill : 78, 79, 87, 89.
fill: 597, 844, 845, 846, 847, 848, 1013, 1014, 1321, 1425, 1426, 1427.
FILL: 244, 246, 845, 902, 1425.
fill_color : inner: 1320, 1321.
fill_color : inner : 1320, 1321.
fill_color : outer: 1428, 1430.
fill_color : outer: 1320, 1321.
fill_value: 1428, 1430.
fill_value: 1428, 1430.
fill_value: 1428, 1430, 1430.
fillout: 1265.
filldraw: 597, 848, 849, 850, 851, 852, 1013, 1014, 1321, 1429, 1430, 1438, 1547.
filldraw: 244, 246, 850, 902, 1428.
first_point: 711, 716, 717, 718, 719, 720.
first_point: 1320, 1321.
_%;V

   

S"+VD 1BL +V9 ( _m C;D  _
S"+VDLh.=E"PD 7P0 .1,/+YL +V9 (

C_]
_% DVD  _m D _  _m D   _% D;E 
_% D   _m D  ]
S"+VDLh.=E"PD 9!.87103B ; ( _ EV³@E  _ EV³ ²  _m D _V]
S"+VDL ; ( _% D@E ]
S"+V9<;TS (  D@D   D    D   >_ E  z_   >_ ²   C   C;³ ]
) H   )( C ;  C V  C@³ ²  C;³V³  VV  VV  V D  V C 
VV @= = D 9 C =; 9V @² D V² C E@E=C
EVE   E @   ³VD@³   _ V²z _@]     
S"+V9<;TS LU9!.PD&; TY19, ( EVE _V]
S$0 ( _  C)E  _m C ]
)F-( C  C ²  C;³   C  C;D  C;D ²  C@D@³   EV³  _@_% D  _m D  
_ E )²  _ E ² E  _ EV³   _m D _  _m D@³  _%_m  _% C   _m C@³ ]
S0 W2L B$9!+ ( E @  E  E  E @  E @  E V  E @²  E  ³  E  D 
E z_  E V  E  E  E V  E 9  E  ³  E D  E C  E )   D  
 D   D  D ² _ D z_@_ _ C z_  ²z_m² ²>_ ³ ² C@D
² C _  ² C   ² C)E  ² C  ² C ²  ² C@³  ²V D  ²@ C  ²V@  ²@V 
²V9  ²@V²  ²V ³  ² E _  ² E9C  ² EVE  ² E   ² E   ² E  ² E9³ 
²9 D  ²Vz_  ²9 C  ²VV  ²9;  ²V9  ²9;²  ²@ D  ²Vz_  ²@V 
²V E  ²@V  ²V9  ²@V²  ²V ³  ²9 C  ²=)  ²9 E  ²=;  ²9V 
²=)²  ²= ³  ²@² D  ³ @²  ³  ³  ³ C  ³ )  ³ E  _ D@³ _ 
_ D@³VC  _ D@³   _ D@³   _@_m9  _@_m@²  _V_  D  _V_ >_  _@_  C 
_ C z_  _ C  C  _ C V  _ C  E  _ C  ³  _ C D  _m DV³  _ z_ D 
_mz_   _mz_ E  _  C@D  _  C _  _ E=C   _ E9C)E  _ E9C   _ E9C 
_ E9C;³  _ E  D  _ E  C  _ E @  _ E V  _ E @  _ E V²  _ E  ³ ]
¶lnk@c ije) ( EV³  C)E _  C)E   C ;  C  E  C @²  C >_  C @ 
C  E  C 9  C @  C =  C  ³  C _  C )  C E  C @  C ;² 
C ²z_ C ² E C ²9 C;³VD C@³VC C;³@E C@³  C;³  DVC  D 
 D@³  E _)  EVEVD  E@E _  EVE@E  E @  E  E  E V  E ;  E 9 
E ;²  E  ³  E  D  E >_  E @  E  E  E @  E =  E  ³  E D 
E C  E ;  E ² C  E ² E  E ²@  EV³   D _   D    D    D 
 D ²  _ D  z_@_  _ C  z_   ;²=  @²@²  ;² ³   ³@D   ³9C 
 ³   ³   ³  ³ ²  ³ ² C ² ²>_ ² ²z_ ³ ² C;D ² C _
² C   ² C;E   ² C   ² C ²   ² C;³   ²@  D  ²@  C  ²V@   ²@V   
²V9  ²@V²  ²V ³  ² E _  ² E9C  ² EVE  ² E   ² E   ² E 
² EV³  ²V D  ²9_  ²V C  ²9@  ²V@  ²VV  ²9;²  ²@ D 
²Vz_  ²@V  ²V E  ²@V  ²V9  ²@V²  ²@ ³  ²= C  ²9; 
²= E  ²9@  ²=@  ²9;²  ²= ³  ²@² D  ²@² C  ²V² E  ²@²= 
² ³   ² ³  ² ³@³  ³@D _  ³ V²  ³  ³  ³ )  ³ E  _ D@³ _ 
_ D@³VC  _ D@³   _ D@³   _@_m9  _@_m@²  _V_  D  _V_ >_  _@_  C 
_ C z_  _ C  C  _ C V  _ C  E  _ C V  _ C  ³  _ C D  _  DV³ 
_mz_ D  _mz_m  _ z_ E  _  C;D  _  C _  _m E ²  _m@²V²  _ E _ ³ 
_ E9C   _ E9C;E  _ E9C   _ E=C  _ E=C;³  _ E  D  _ E  C  _ E V 
_ E V _ E @ _ E @² _ E  ³ _ E9³ _V]
¶lnk@c ije)9  ( _m@   C  C   C  E  C  >_  C ;²   DV³  C ²  _ @Vz]
S03DO ( _%;  _m9 ]
S2 ( @²9  @²@²  ³@³   ³@³ ²  ³@³@³  _ D@DVD  _ DVD  _ DVC;³ 
_ D  D _ D z_ _ D  E  _ D@EVD  _ D@E >]
S2 D P9,G.P2  ( @²V²  ]
¶n>bj ( _%  E9³  EVDVD  V@²  V ³  @z_  V@  @ E  VV 
V@  @=  V@²  @ ³  = D  9 C  9)  = E  9@ 
=@  9;²  = ³  @² D  V²z_  @²9  @²9  V²@²   ³VD 

  

D

 ³ _   ³VC  ³;E   ³;E  ³;E ²  ³ @  ³V³VC  ³@³   ³V³ 
@³ ³ ²  _ DVC@³  _ D;E9D  _ C _%  _  D ]
¶nzb0= ( 9 ³  @²V²  _ C _ E ]
 folnbdc ( C  _ ³  EV³  _ V  _ ³;E  C@DVC  C _@_  C _   C)E _ 
C)EVE C;E  C @ C @ C  ³ C ²V C ²V²  D  DV³ >_ D
z_ C  >_   z_)  >_ ³   C@D   C    C;E   C    C   C ² 
 C@³  V D  @>_  V C  @V  V E  @9  V@  @=  V@² 
@ ³   EVD   E _   E    E    E  Vz_  9 C  V@  9;² 
@>_  V@  @V  V@²  9_  =)  9;  =)²  @²>_  V²@ 
@²V  V²@²  @² ³   ³@D   ³    ³;E   ³    ³    ³   ³ ² 
 ³V³ E9D _ EVD  E9D EVDV³ E _  E _ E E _m E _m E _%
E9C@D  E=C _  E9CVC  E=C   E9C;E  E=C   E9C   E=C  E  D  E @ 
E =  E  ³  E@E9D  EVE   E@E   EVE  E@E ²  EVEV³  E z_  E @ 
E 9  E  ³  E >_  E @  E  E  E @  E =  E  ³  E D  E C 
E ;  E )  E ;²  E ² D  E ²>_  E ²@  E ²9  E ²@  E ²=  E9³@D 
EV³9C  E9³;E  EV³    DVC   D    D    D   D ²   DV³  z_@_ 
_m  z_m  _m  z_ ³   C _   C@C   C    C)E   C    C  
 C   C ²   C@³  @ D  ; C  @@  ; E  @V  ;V  @9 
; ³   EVD   E _   E9C   E    E@E   E    E    E   E ² 
 E9³  V_  @ C  V;  @ E  V@  @@  VV  ; D  @z_ 
;=  @@²  ; ³  9 D  V_  9 C  V;  9 E  ;²9   ³  
 ³  D@D  D _  D@³ z_ D >_@_ z_ C  C;D  C _  C 
@9  V9  @V²  V ³   E9D   E    E   E ²   E9³  9 D 
V ³  Vz_  @=  V@²  9V  =)²  9 ³  V²z_  @² E  V²V 
@²V²   ³ ²  D _  D   D  D ²  DV³  _ D  =_V_  _  
=_%  _   =_)  _ ²  =_ ³  C;D  C  ; D  ) C  ;@ 
@;  V E  @@  V@  @;²  V ³  )² C  ;² E  )²V  ³ _ 
³@E ³  ³ ³ ² ³V³ ² D@D ² D ² ²z_m ² C 
² C;E  ²V C  ²@V  ²@ E  ² E _  ² E9C  ²V D  ²@>_  ²==_ 
²9;  ²V² C  ²@²V  ²@²@  ²@²=  ² ³@³  ³VDVC  ³ _ D  ³ _% 
³ _ ³  ³9C;³  ³  D  ³  C  ³ V  ³  E  ³ V  ³ V  ³ 9 
³;E9D  ³@E _  ³;EVE  ³;E   ³;E  ³;E ²  ³ ;²  ³  E  ³ V 
³ =  ³ @²  ³  ³  ³ C  ³ ;  ³ E  ³ )²  ³ ²9  ³V³VC 
³@³   ³V³;E  ³V³   ³@³   ³@³  ³V³ ²  _ DVD _  _ D@D ²  _ DVD@³ 
_ D _@_  _ D _ C  _ D _ ³  _ DVCVC  _ DVC   _ D9C)E  _ D9C   _ DVC ² 
_ D9C;³  _ D  C  _ D  E  _ D  ³  _ D;E9D  _ D@E _  _ D@E   _ D  D 
_ D _  _ D z_  _ D  C  _ D V  _ D =  _ D )  _ D @  _ D )² 
_ D ³  _ D ² D  _V_ D   _V_ D  _@_ D ²  _@_ D@³  _@_V_ D  _V_@_@_ 
_@_V_ C  _@_V_   _V_@_ E  _V_ CVC  _@_ C   _@_ C   _@_ C   _V_ C 
_@_ C ²  _@_ E   _V_ EV³  _V_m D  _@_mz_  _@_% C  _@_mz_  _V_  E 
_@_mV  _@_)@  _V_%)²  _V_% ³  _@_ ² D  _@_m² C  _@_m²@  _V_ ² E 
_@_m²V  _ C@D _  _ C;DVC  _ C;D   _ C;D   _ C@D ²  _ C@D@³  _ C _ D 
_ C _   _ C _%  _ C _ ²  _ C@C _  _ C@CVC  _ CVC)E  _ CVC  _ C  C 
_ C;E@E  _ C;E   _ C)EV³  _ C  D  _ C  C  _ C ;  _ C ;²  _ C 9 
_ C @²  _ C ²z_  _ C ² C  _ C ²V  _ C ² E  _ C@³   _ C@³   _ C;³  
_ C@³@³  _  D@D  _m D _  _m D   _  D  _ >_@_  _  C _  _m C)E 
_  C   _  C  _m@@  _mV@  _ V E  _ V ³  _ = D  _m9=_ 
_ = C  _ =)  _m9 E  _m9@  _ 9;  _ =@  _ =)²  _m ³ 
_  ³ ²  _ E9D _  _ EVDVC  _ E  ³  _ E  D  _ E )  _ E E  _ E ²@ 
_ E ² E  _ E ²V  _ E ²9  _ E ²V²  _ EV³VD  _ E9³ _  _ E9³VC  _% D@D 
_m D _  _m DVC  _% D   _% D@E  _m D   _m D   _mz_ E  _%_m 
_mz_%  _%_ ³  _m C ²  _m C;³  _%;>_  _m E =]


760, 762, 763, 764, 765, 1001, 1012, 1047, 1048, 1050, 1051, 1052, 1053, 1078, 1107, 1150, 1165, 1202, 1212, 1218, 1235, 1236, 1252, 1253, 1254, 1255, 1267, 1268, 1305, 1324, 1327, 1399, 1400, 1401, 1428, 1460, 1484, 1496, 1492, 1502, 1503, 1504, 1506, 1515, 1517, 1519, 1529, 1531.

rotate around: 221, 212, 288, 423, 436, 438, 439, 440, 759, 760, 762, 764, 1050, 1052, 1252, 1254, 1401.

row: 173, 179, 190, 191, 227, 228, 229, 625, 626, 628, 629.

column: 1320, 1321.

cycle: 68, 69, 70.

cycle: 317, 993, 1001, 1002, 1003, 1004, 1005.

Run State: 502, 906.

r0: 992, 993, 1034, 1035, 1260.

r0_0: 72, 173.

r0_1: 72, 173.

r0_2: 72, 173.

r0_3: 72, 173.

r1: 1034, 1035, 1260.

r1_0: 72, 173.

r1_1: 72, 173.

r1_2: 72, 173.

r1_3: 72, 173.

r2: 1260.

r2_0: 72, 173.

r2_1: 72, 173.

r2_2: 72, 173.

r2_3: 72, 173.

r3_0: 72, 173.

r3_1: 72, 173.

r3_2: 72, 173.

r3_3: 72, 173.

s: 123, 124, 270, 271, 508, 678, 680, 735, 736, 747, 748, 749, 750, 751, 752, 801, 802, 930, 1007, 1161, 1182, 1183, 1184, 1185, 1338, 1339, 1345, 1346, 1349, 1356, 1357, 1358, 1359, 1360, 1361, 1362, 1363, 1364, 1365, 1366, 1367, 1369, 1370, 1371, 1372, 1373, 1374, 1375, 1376, 1377, 1378, 1419, 1420, 1424, 1427, 1430, 1433, 1436, 1439.

s.e.: 1227, 1228.

s.t.: 1227, 1228.

save as: 1419.

save as: 1424.

save as: 1445.


scale value: 1267.

second: 16, 32, 313, 316, 399, 647, 677, 681, 993, 996, 1002, 1003, 1004, 1005, 1029, 1032, 1043, 1173, 1174, 1175, 1201, 1202, 1218, 1223, 1225, 1227, 1260, 1267, 1305.

shape_type: 1356, 1357, 1369, 1370.

y0: 972.
y1: 972.
z_axis: 993.
z_axis.pt: 1001.
z_j: 579, 580, 582, 584, 585.
z_int_p: 582, 583, 584.
z_int_q: 582, 583, 584.
Z_Y: 993, 1221, 1223.
xx: 197, 198, 245, 408, 409, 768, 769, 1056, 1057, 1239, 1240, 1392, 1393.
zy: 197, 198, 245, 408, 409, 768, 769, 1056, 1057, 1239, 1240, 1392, 1393.
zz: 1242, 1243, 1390, 1391, 1395, 1396, 1399, 1400.
z0: 972.
z1: 972.
(Actions in main 1546, 1547) Used in section 1557.
(All Colors 0) Cited in sections 153 and 158.
(Calculate second-largest Real 69) Used in section 68.
(Check intersection point locations 1228) Used in section 1227.
(DEC command line option processing 1551) Used in section 1548.
(Declare I/O functions 81, 84, 86, 88) Used in section 92.
(Declare Pattern functions 1320, 1323, 1326) Used in sections 1329 and 1330.
(Declare namespace Projections 256) Used in section 305.
(Declare namespace Sorting 258) Used in section 305.
(Declare non-member non-template functions for Color 147) Used in section 163.
(Declare non-member non-template functions for Point 480, 534) Used in section 634.
(Declare non-member template functions for Circle 1286, 1287) Used in sections 1315 and 1316.
(Declare non-member template functions for Color 112, 113) Used in sections 162 and 163.
(Declare non-member template functions for Cuboid 1462, 1463) Used in sections 1468 and 1469.
(Declare non-member template functions for Ellipse 1154, 1155) Used in sections 1271 and 1272.
(Declare non-member template functions for Path 724, 725) Used in sections 980 and 981.
(Declare non-member template functions for Point 356, 337) Used in sections 633 and 634.
(Declare non-member template functions for Rectangle 1116, 1117) Used in sections 1139 and 1140.
(Declare non-member template functions for Reg_Polygon 1083, 1084) Used in sections 1097 and 1098.
(Declare non-member template functions for Solid 1341, 1342) Used in sections 1441 and 1442.
(Declare parser functions 1538) Used in section 1542.
(Declare utility functions 24, 31) Cited in section 24. Used in section 52.
(Declare Circle functions 1278, 1281, 1283, 1290, 1292, 1295, 1297, 1298, 1300, 1302, 1304) Used in section 1276.
(Declare Color functions 97, 99, 102, 104, 107, 109, 114, 116, 118, 121, 123, 125, 127, 129, 131, 133, 135, 138, 141, 142, 143, 144, 146, 149, 151) Used in section 95.
(Declare Cuboid functions 1455, 1457, 1459, 1464, 1466) Used in section 1453.
(Declare Dodecahedron functions 1497, 1500, 1503, 1505) Used in section 1494.
(Declare Ellipse functions 1146, 1149, 1151, 1157, 1160, 1162, 1164, 1166, 1167, 1169, 1171, 1174, 1177, 1179, 1182, 1184, 1186, 1188, 1192, 1194, 1197, 1199, 1201, 1203, 1205, 1208, 1210, 1214, 1231, 1233, 1235, 1237, 1239, 1242, 1244, 1247, 1249, 1252, 1254, 1257, 1259, 1261, 1263) Used in section 1143.
(Declare Focus functions 602, 604, 606, 609, 611, 613, 615, 617, 620, 621, 622, 623, 624, 625, 627, 628) Used in section 600.
(Declare Icosahedron functions 1511, 1514, 1516, 1518) Used in section 1508.
(Declare Label functions 255) Used in section 253.
(Declare Line constructors 639, 641) Used in section 637.
(Declare Line functions 643, 646, 648, 652) Used in section 637.
(Declare Point constructors 324, 327, 331) Used in section 309.
Declare Polygon functions 1022, 1024, 1028, 1037, 1039, 1045, 1047, 1050, 1052, 1054, 1056, 1059, 1061, 1064, 1066} Used in section 1019.

Declare Polyhedron functions 1473} Used in section 1472.

Declare Rectangle functions 1103, 1106, 1108, 1111, 1113, 1119, 1122, 1125, 1127, 1130, 1132, 1135, 1136, 1137, 1138} Used in section 1101.

Declare Reg_CiPlane_Curve functions 987, 988, 989, 990, 991, 992, 994, 997, 1008, 1011, 1013, 1014} Used in section 985.

Declare Reg_Polygon functions 1070, 1073, 1076, 1079, 1087, 1089, 1091, 1092, 1093, 1095, 1096} Used in section 1069.

Declare Solid_Faced functions 1446} Used in section 1445.

Declare Solid functions 1336, 1338, 1343, 1345, 1348, 1350, 1353, 1356, 1358, 1360, 1362, 1366, 1366, 1369, 1371, 1373, 1375, 1377, 1379, 1381, 1383, 1386, 1388, 1390, 1392, 1395, 1397, 1399, 1401, 1404, 1406, 1408, 1409, 1411, 1413, 1415, 1417, 1419, 1423, 1426, 1429, 1432, 1435, 1438} Used in section 1333.

Declare System functions 36, 38, 40, 43, 45, 47, 66, 71, 72} Used in sections 34, 52, 65, and 75.

Declare Tetrahedron functions 1480, 1483, 1487, 1489, 1491} Used in section 1477.


Declare Trunc_Octahedron functions 1525, 1528, 1530} Used in section 1522.

Declare create_new() 56, 58

Declare draw_axes() 968, 973} Used in section 981.

Declare namespace Colors 153} Cited in section 153. Used in section 162.

Declare namespace System 34, 65} Used in sections 51, 52, 74, and 75.

Define I/O functions 82, 83, 85, 87, 89} Used in section 91.

Define Pattern functions 1321, 1324, 1327} Used in section 1329.

Define classes 253, 261} Used in sections 305 and 306.

Define comparison classes 498, 499, 500} Cited in section 596. Used in sections 633 and 634.

Define non-member non-template functions for Color 148} Used in section 162.

Define non-member non-template functions for Point 481, 535} Used in section 633.

Define parser functions 1539} Used in section 1541.

Define utility functions 25, 32} Cited in section 25. Used in section 51.

Define Circle functions 1279, 1282, 1284, 1291, 1293, 1296, 1301, 1303, 1305} Used in section 1315.


Define Cuboid functions 1456, 1458, 1460, 1465, 1467} Used in section 1468.

Define Dodecahedron functions 1498, 1501, 1502, 1504, 1506} Used in section 1534.

Define Ellipse functions 1147, 1150, 1152, 1158, 1161, 1165, 1168, 1170, 1172, 1173, 1175, 1178, 1180, 1183, 1185, 1187, 1189, 1193, 1195, 1198, 1200, 1202, 1204, 1206, 1209, 1211, 1215, 1216, 1217, 1218, 1219, 1220, 1221, 1222, 1223, 1224, 1226, 1227, 1232, 1234, 1236, 1238, 1240, 1245, 1248, 1250, 1253, 1255, 1258, 1260, 1262, 1264} Used in section 1271.

Define Focus functions 605, 607, 610, 612, 614, 616, 618, 626, 629} Used in section 633.

Define Icosahedron functions 1512, 1515, 1517, 1519} Used in section 1534.

Define Label functions 514, 515, 516} Used in section 633.

Define Line constructors 640, 642} Used in section 657.

Define Line functions 644, 649, 650, 651, 653, 978} Used in sections 657 and 980.


Define Plane functions 664, 666, 668, 670, 673, 675, 678, 680, 685, 688, 690, 966} Used in sections 694 and 980.
(Define Point constructors 325, 328, 332)  Used in section 633.
(Define Polygon functions 1023, 1025, 1029, 1030, 1031, 1032, 1033, 1034, 1035, 1036, 1038, 1040, 1041, 1042, 1043, 1046, 1048, 1051, 1053, 1055, 1057, 1060, 1062, 1065, 1067)  Used in section 1097.
(Define Polyhedron functions 1474)  Used in section 1534.
(Define Rectangle functions 1104, 1107, 1109, 1112, 1114, 1120, 1123, 1126, 1128, 1131, 1133, 1267, 1268, 1269, 1270)  Used in sections 1139 and 1271.
(Define Reg_Cl_Plane_Curve functions 993, 995, 998, 999, 1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1009, 1012)  Used in section 1015.
(Define Reg_Polygon functions 1071, 1074, 1077, 1078, 1080, 1081, 1307, 1309, 1310, 1311, 1313, 1314)  Used in sections 1097 and 1315.
(Define Shape class 244, 245)  Used in sections 248 and 249.
(Define Solid_Faced functions 1447)  Used in section 1449.
(Define Solid functions 1337, 1339, 1344, 1346, 1347, 1349, 1351, 1354, 1357, 1359, 1361, 1363, 1365, 1367, 1370, 1372, 1374, 1376, 1378, 1380, 1382, 1384, 1387, 1389, 1391, 1393, 1396, 1398, 1400, 1402, 1405, 1407, 1410, 1412, 1414, 1416, 1418, 1420, 1424, 1427, 1430, 1433, 1436, 1439)  Used in section 1441.
(Define System functions 37, 39, 41, 44, 46, 48, 67, 68)  Used in sections 51, 74, and 75.
(Define Tetrahedron functions 1481, 1484, 1485, 1486, 1488, 1490, 1492)  Used in section 1534.
(Define Trunc_Octahedron functions 1526, 1529, 1531, 1532)  Used in section 1534.
(Define bool_point_quadruple functions 316)  Used in section 633.
(Define bool_point functions 314)  Cited in section 313.  Used in section 633.
(Define bool_real_point functions 318)  Used in section 633.
(Define class Circle 1276)  Used in sections 1315 and 1316.
(Define class Color 95)  Used in sections 162 and 163.
(Define class Cuboid 1453)  Used in sections 1468 and 1469.
(Define class Dodecahedron 1494)  Used in sections 1534 and 1535.
(Define class Ellipse 1143)  Used in sections 1271 and 1272.
(Define class Focus 600)  Used in sections 633 and 634.
(Define class Icosahedron 1508)  Used in sections 1534 and 1535.
(Define class Path 698)  Used in sections 980 and 981.
(Define class Point 309)  Used in sections 633 and 634.
(Define class Polygon 1019)  Used in sections 1097 and 1098.
(Define class Polyhedron 1472)  Used in sections 1534 and 1535.
(Define class Rectangle 1101)  Used in sections 1139 and 1140.
(Define class Reg_Cl_Plane_Curve 985)  Used in sections 1015 and 1016.
(Define class Reg_Polygon 1069)  Used in sections 1097 and 1098.
(Define class Solid_Faced 1445)  Used in sections 1449 and 1450.
(Define class Solid 1333)  Used in sections 1441 and 1442.
(Define class Tetrahedron 1477)  Used in sections 1534 and 1535.
(Define class Transform 166)  Used in sections 239 and 240.
(Define class Trunc_Octahedron 1522)  Used in sections 1534 and 1535.
(Define create_new( ) 57, 59)  Used in sections 61 and 62.
(Define draw_axes( ) 969, 970, 971, 972, 974)  Used in section 980.
(Define static Point data members 310)  Used in section 633.
(Define static const Dodecahedron data members 1495) Used in section 1534.
(Define static const Icosahedron data members 1509) Used in section 1534.
(Define static const Solid data members 1334) Used in section 1441.
(Define static const Tetrahedron data members 1478) Used in section 1534.
(Define static const Trunc_Octahedron data members 1523) Used in section 1534.
(Define static Ellipse data members 1144) Used in section 1271.
(Define static Shape member variables 246) Used in section 248.
(Define static class Path data members 699) Used in section 980.
(Define struct Line 637) Used in sections 657 and 658.
(Define struct Plane 661) Used in sections 694 and 695.
(Discard points and connectors 703) Used in sections 701, 710, 715, 720, and 729.
(Forward declarations 49) Used in sections 51 and 52.
(GCC 2.95 print version, copyright, and license information 1553) Used in section 1552.
(GCC 3.3 and DEC print version, copyright, and license information 1554) Used in section 1552.
(GCC command line option processing 1549) Used in section 1548.
(GNU Free Documentation License 1561) Cited in section 1.
(GNU General Public License 1562) Cited in section 1.
(Get input 1545)
(Global constants 22, 28, 159, 236, 319) Used in sections 51, 162, 239, and 633.
(Global variables 18, 19, 20, 78, 234, 302, 630) Cited in section 25. Used in sections 51, 91, 239, 305, and 633.
(Handle intersection point 1225) Used in section 1224.
(Include files 6, 7, 8, 9, 14, 54, 64, 77, 94, 165, 242, 251, 308, 636, 660, 697, 983, 1018, 1100, 1142, 1274, 1318, 1332, 1444, 1452, 1471, 1537, 1544) Used in sections 12, 51, 61, 74, 91, 102, 239, 248, 305, 633, 657, 694, 980, 1015, 1097, 1139, 1271, 1315, 1329, 1441, 1449, 1468, 1534, 1541, and 1558.
(Initialize coordinates and limits 323) Used in sections 325, 328, and 332.
(Initialize static Label data members 254) Used in section 305.
(Loop for testing bits 70) Cited in section 66.
(Main 1555, 1556, 1557) Used in section 1558.
(Major Colors 156) Cited in sections 153 and 158. Used in section 153.
(Normalize point 433) Used in section 425.
(Output Path 907) Used in sections 902, 904, 905, and 906.
(Print version, copyright, and license information 1552) Used in section 1556.
(Process command line options 1548) Used in section 1555.
(Process vectors for draw() 1422) Used in section 1424.
(Process vectors for fill() 1425) Used in section 1427.
(Process vectors for filldraw() 1428) Used in section 1430.
(Process vectors for undraw() 1431) Used in section 1433.
(Process vectors for unfill() 1434) Used in section 1436.
(Process vectors for unfilldraw() 1437) Used in section 1439.
(Type definitions 15, 312, 313, 315, 317) Cited in section 15. Used in sections 51, 52, 633, and 634.
(Utility classes 16) Used in sections 51 and 52.
(circles.h 1316)
(colors.h 163)
(creatnew.h 62)
cuboid.h 1469
curves.h 1016
ellipses.h 1272
(gsltmplt.h 75
(io.h 92)
{lines.h  658}
{loader.h   12}
{parser.h  1542}
{paths.h  981}
{patterns.h  1330}
{pictures.h  306}
{planes.h  696}
{points.h  634}
{polygons.h  1098}
{polyhedr.h  1535}
{pspplib.h  52}
{rectangls.h  1140}
{shapes.h  249}
{solfaced.h  1450}
{solids.h  1442}
{transfor.h  240}

(Line global constants  654) Used in section 657.
(Plane global constants  691) Used in section 694.
{extern All Colors  0} Cited in section 154.
{extern Major Colors  157} Cited in section 154. Used in section 154.
{extern declaration of namespace Projections  237} Used in section 306.
{extern declaration of namespace Sorting  259} Used in section 306.
{extern global constant declarations  160} Used in section 163.
{extern variable declarations  79} Used in section 92.
{extern namespace Colors declaration  154} Cited in sections 154 and 158. Used in section 163.