1. Copyright and License.

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

The program GNU 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 3 of the License, or (at your option) any later version. See the section "(GNU General Public License 1.562)" in this document.

GNU 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 GNU 3DLDF; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

GNU 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. GNU 3DLDF is available for downloading from http://www.gnu.org/software/3dldf/LDF.html. It is also available from http://www.dante.de/software/ctan/, the Dante www-server and from http://wwwuser.gwdg.de/~1finsto1, the author’s website.

Please send bug reports to:
1finsto1@gwdg.de

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
c/o Free Software Foundation, Inc.
51 Franklin St, Fifth Floor
Boston, MA 02110-1301
USA
1finsto1@gwdg.de
s246794@student.uni-goettingen.de

Web site: http://wwwuser.gwdg.de/~1finsto1

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 CWB and it outputs MetaPost code.

In the text sections of the CWB 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 on 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.

Log

[LDF 2002.11.18.] Changed name of this file from cweavedriver.web to cdriver.web. It now has fewer than 8 letters and can be used under DOS.

[LDF 2003.08.16.] Changed name of this file from cdriver.web to 3DLDF.web sometime since 2002.11.18. Forgot to note 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.

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

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

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 cweave 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 cweaved 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 plfmvar.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 2003.08.25.] 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 plfmvar.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> ≡
static string rev_id = "$Id:, loader.web,v,1.4,2004/01/12,21:30:27,1finsto1,Exp,8$";

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 1553.

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, 1508, 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. However, 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
#define LDF_GCC_2_95
#include <new.h>
#endif
#include <sstream>
#include <stdarg.h>
#include <stdexcept>
#if defined __GNUC__
#include <stdio.h>
#endif
#if defined __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 files 6) +
#include <string>
#include <valarray>
#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 files 6) +
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 3dl df if loader.c is compiled.

#define __DECCXX
#include <stdlib.h>
#else HAVE_STDLIB_H
#include <stdlib.h>
#endif
#include <string>
using namespace std;

(Version control identifier 5)
12. This is what's written to the loader.h.

[Log]

[LDL 2004.01.06.] Added this section.

(Loader.h 12) +=

(Include files 6)

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

[Log]

[LDL 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.

[LDL 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 ver_id = "$id:psglb.web,v.1.14,2004/01/16,16:30:41,lfinsto1,Exp,$";


(Include files 6) +=

#include "loader.h"
#ifdef __DECCXX
#include <limits>
#else
#ifdef LDF_GCC_3_3
#include <limits>
#else
#ifdef LDF_GCC_2_95
#if HAVE_LIMTS_H
#include <limits.h>
#endif
#endif
#endif
#endif
#include <bitset>

15. Type definitions. [LDL 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]

[LDL 2002.04.10.] Added formatting commands.
[LDL 2002.04.10.] Added declaration of bool_real.
[LDL 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).
[LDL 2003.06.03.] Added real_short. It's the return type of Plane::get_distance().
format real float
format real_pair real
format bool_pair real_pair
format real_pair real_short
format Matrix int

(Type definitions 15) ≡
#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) ≡
   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, MAX_REAL = INVALID_REAL –
real_limits.epsilon() caused MAX_REAL and INVALID_REAL to be equal! I didn’t notice the problem until
I started to port 3DLDV to GNU/Linux. It also doesn’t work to use `MAX_REAL = INVALID_REAL - realLimits.min`.

(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. `MAX_REAL` is the second largest real value. `MAX_REAL_SQRT` is convenient to have for testing when computing distances.

!! 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]

[Log 2003.12.08.] Changed the definition of `MAX_REAL`. Previously, it was calculated using \(0.0000003 \times \text{FLT\_MAX}\), which was a kludge.

[Log 2003.12.29.] Changed the way `MAX_REAL` and `MAX_REAL_SQRT` are declared. They can no longer be `const`, because the value of `MAX_REAL` is set at the beginning of `main()` using `get\_second\_largest < Real > ()`. The value of `MAX_REAL_SQRT` is set after this. Their values should never change after this!
`MAX_REAL_SQRT` must be initialized here, because it’s used in `Point::magnitude()`. [LDF 2003.12.29]

(Global variables 18) \equiv

```
#ifndef __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.] `VERBOSE\_GLOBAL` is `false` by default. It is set to `true` by the command line option `"--verbose"`. If `VERBOSE\_GLOBAL` is `true`, the local `verbose` variables in functions are set to `true`.

[Log 2003.08.14.] Added `VERBOSE\_GLOBAL` and `SILENT\_GLOBAL`.

```
(Global variables 18) \equiv
bool VERBOSE\_GLOBAL = false;
bool SILENT\_GLOBAL = false;
```
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

[LDF 2003.08.14.] Added VERSION_3DLDF and COPYRIGHT_3DLDF.

(Global constants 22) ≡
#if LDF_REAL_FLOAT
  extern const bool ldf_realfloat = 1;
  extern const bool ldf_realdouble = 0;
#elif 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 = "3DLDF comes with ABSOLUTELY NO WARRANTY;
\nfor details, see the file COPYING, which you should have received in the distribution of 3DLDF.1.1.5.1\nThis is free software, and you are welcome into redistributing it under certain conditions; for details, again, see the file COPYING.\n\nPlease send bug reports to the author at:n\nemail: lfinstoi@gwdg.de
www: http://wwwuser.gwdg.de/~lfinstoi"

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

23.

Log

[LDF 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 DISCLAIMER_3DLDF;
extern const bool ldf_realfloat;
extern const bool ldf_realdouble;
24. TO DO: Find out why the library version of \texttt{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]

---

[LDF 2003.08.14.]. Put this function declaration in \texttt{<Declare utility functions 24>}. Formerly, it was in \texttt{<Declarations for the header file 21>}. Changed the conditional from \texttt{#ifdef _GNU_} to \texttt{#ifdef LDF\_GCC\_2\_95}, because the library version of \texttt{trunc()} is found when compiling with GCC 3.3 under Linux.

\texttt{(Declare utility functions 24) \equiv}\n\texttt{#ifdef LDF\_GCC\_2\_95}\n\texttt{double trunc(double d);}\n\texttt{#endif}

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

[LD 2003.08.14.] Put this function definition into (Define utility functions 25). Formerly, it was in (Global variables 18).

(Define utility functions 25) \equiv
\#ifdef LDF_GCC_2_95 /* KLUDGE!! [LD 2002.12.1.] \texttt{trunc()} isn't available on the Linux machine gwdg-wb02.gwdg.de! Find out why not! */
double \texttt{trunc(double d)}
{
    int i;
    i = static_cast\langle int\rangle(d);
    return static_cast\langle double\rangle(i);
}
\#endif
See also section 32.
This code is cited in section 25.
This code is used in section 51.

26. For the header file.
(Declarations for the header file 21) \equiv
\hspace{1cm} \textbf{extern valarray\langle real\rangle null\_coordinates;}
\#ifdef __DECCXX
\hspace{1cm} \textbf{extern numeric\_limits\langle real\rangle real\_Limits;}
\#endif

27. Global constants. 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 INVALID\_REAL or functions return it when something has gone wrong. INVALID\_REAL is also used for the real values in INVALID\_TRANSFORM and INVALID\_POINT. Another possibility would be to use exception handling, but so far I've found it convenient to use 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. [LD 2002.10.16.] Modified [LD 2002.10.20.]

28. For compilation.
(Global constants 22) \equiv
\hspace{1cm} \textbf{extern const real PI = 4.0 * \texttt{atan(1.0);}}
\hspace{1cm} \textbf{extern const real pair INVALID\_REAL\_PAIR(INVALID\_REAL, INVALID\_REAL);}
\hspace{1cm} \textbf{extern const real short INVALID\_REAL\_SHORT(INVALID\_REAL, 0);}
29. For the header file.
(Declarations for the header file 21) +≡
  extern const real P1;
  extern const real INVALID_REAL;
  extern const real_pair INVALID_REAL_PAIR;
  extern const real_short INVALID_REAL_SHORT;

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.

Log

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

(Declare utility functions 24) +≡
  real_pair solve_quadratic(real a, real b, real c);
32.  
(Define utility functions 25) \( \equiv \)

```c
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) \( \equiv \)

```c
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.  \texttt{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 \texttt{is\_little\_endiand()} and \texttt{is\_big\_endiand()} [LDF 2003.12.21.]

\begin{quote}
\texttt{[LDF 2003.12.29.] Added this function.}
\end{quote}

\begin{verbatim}
(Declare System functions 36) \equiv
  \textcolor{red}{\begin{align*}
  \textbf{signed short} & \texttt{get\_endianness(const bool verbose = false);} \end{align*}}
\end{verbatim}

See also sections 38, 40, 43, 45, 47, 66, 71, and 72.
This code is used in sections 34, 52, 65, and 75.

37. \begin{verbatim}
(Define System functions 37) \equiv
  \textcolor{red}{\begin{align*}
  \textbf{signed short} & \texttt{System::get\_endianness(const bool verbose)} \\
  \{ & \text{union \{} \\
    & \text{long } \texttt{Long;} \\
    & \text{char } \texttt{Char}\text{sizeof(long)}; \\
  \} \text{;} \\
  \text{u.\texttt{Long} = 1;} \\
  \text{if } (\text{u.\texttt{Char}[0] \equiv 1}) \{ \\
    \text{if } (\text{verbose}) \text{ cout \ll "Processor \texttt{.is\_little\_endiand.}\ll \text{endl \ll endl \ll flush;} \ll \text{return 0;} \} \\
  \text{else if } (\text{u.\texttt{Char[\text{sizeof(long)} - 1]} \equiv 1}) \{ \\
    \text{if } (\text{verbose}) \text{ cout \ll "Processor \texttt{.is\_big\_endiand.}\ll \text{endl \ll endl \ll flush;} \ll \text{return 1;} \} \\
  \text{else } \{ \\
    \text{cerr \ll "ERROR! } \text{.In}\texttt{.System::get\_endianness()}:\ln\ll " \text{Can\’t\texttt{.determine\_endiand.}}\ll \text{Returning\texttt{-1}}\ll \text{endl \ll endl \ll flush;} \ll \text{return -1;} \} \\
  \}
\end{align*}}
\end{verbatim}

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.

\begin{quote}
\texttt{[LDF 2003.12.29.] Added this function.}
\end{quote}

\begin{verbatim}
(Declare System functions 36) \equiv
  \textcolor{red}{\begin{align*}
  \textbf{bool} & \texttt{is\_big\_endiand(const bool verbose = false);} \end{align*}}
\end{verbatim}
39.  
{Define System functions 37} +\equiv
    bool System::is_big_endian(const bool verbose)
    {
        return (get_endianness(verbose) \equiv 1);
    }

40.  Is little endian.  

[LDF 2003.12.29.] Added this function.

{Declare System functions 36} +\equiv
    bool is_little_endian(const bool verbose = false);

41.  
{Define System functions 37} +\equiv
    bool System::is_little_endian(const bool verbose)
    {
        return (get_endianness(verbose) \equiv 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
    unsigned short get_register_width();

44.  
{Define System functions 37} +\equiv
    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
    bool is_32bit();
46. (Define System functions 37) +≡
   bool System::is_32_bit()
   {
      return (get_register_width() ≡ 32);
   }

47. Is 64 bit.
   Log
   [LDF 2003.12.29.] Added this function.

   (Declare System functions 36) +≡
   bool is_64_bit();

48. (Define System functions 37) +≡
   bool System::is_64_bit()
   {
      return (get_register_width() ≡ 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.
    Log
    [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) ≡
   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 psgl.h 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
   #ifdef __DECCXX    /* Using the DEC C++ Compiler. */
   const real PI = __CXXL_PI;
   #endif
   #endif
   (Declare utility functions 24)
   (Forward declarations 49)
   (Declare System functions 36)

53. Dynamic allocation for Shapes.

Log

[LDF 2003.12.29.] Added this template function.
[LDF 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 \texttt{create\_new} () \refsect\ 
  \texttt{template\langle class C\rangle C*create\_new(const C*ary)}\n  \{\n    C * obj = \texttt{new} (C);\n    obj->set\_on\_free\_store();\n    if (ary \neq 0) *obj = *ary;\n    \textbf{return} obj;\n  \}\n\}

See also section 59.
This code is used in sections 61 and 62.

58. \textbf{Reference argument.}\n\{(Declare \texttt{create\_new} () \refsect\ +\refsect\ 
  \texttt{template\langle class C\rangle C*create\_new(const C& ary)};\n\}

59.  
\{(Define \texttt{create\_new} () \refsect\ +\refsect\ \ 
  \texttt{template\langle class C\rangle C*create\_new(const C& ary)}\n  \{\n    C * obj = \texttt{new} (C);\n    obj->set\_on\_free\_store();\n    *obj = ary;\n    \textbf{return} obj;\n  \}\n\}

60. \textbf{Putting \texttt{create\_new} together.}\n
61.  
This is what’s compiled. I don’t really need to compile the definition of \texttt{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.]
\{(Include files 6\}
\{(Version control identifier 5\}
\{(Define \texttt{create\_new} () \refsect\}

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]

\{ creatnew.h 62 \} \equiv
\{ Define create_new() 57 \}

63. Get second-largest real value.
\{ Version control identifier 5 \} \equiv
\{ static string res_id = "$Id:\_/gspltmpl.web,v1.4,2004/01/12,29:56,1finsito1,Exp$n$"; \}

64. Include files.
\{ Include files 6 \} \equiv
\#include "loader.h"
\#include "pspglb.h"

65. Declare namespace System.

\[ LDF 2004.1.2. \] Added this section.

\{ Declare namespace System 34 \} \equiv
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 > (DBL_MAX). FLT_MAX or DBL_MAX must be passed as an argument. On systems with the
numeric_limits template, realLimits.max() could be used instead. [LDF 2003.12.29]

\{ 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 low
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]

\[ LDF 2003.12.29. \] Added this function.
\[ LDF 2004.1.2. \] Moved this section from pspglb.web to creatnew.web.

\{ Declare System functions 36 \} \equiv
\{ template(class Real) Real get_secondLargest(RealMAX_VAL, bool verbose = false); \}
\{ float get_secondLargest(float, bool); \}
\{ double get_secondLargest(double, bool); \}

67.
\{ Define System functions 37 \} \equiv
68.

(Define System functions 37) \(\equiv\)

```cpp
template (class Real) Real System::get_second_largest(Real MAX_VAL, bool verbose) {
  const unsigned short USHORT_SIZE = sizeof(unsigned short);
  const unsigned short UINT_SIZE = sizeof(unsigned int);
  const unsigned long ULONG_SIZE = sizeof(unsigned long);
  const unsigned long 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 RealEQUSHORT = (REAL_SIZE == USHORT_SIZE);
  const bool RealEQUINT = (REAL_SIZE == UINT_SIZE);
  const bool RealEQULONG = (REAL_SIZE == ULONG_SIZE);
  const bool RealEQULONG_LONG = (REAL_SIZE == ULONG_LONG_SIZE);

  if (verbose) {
    cout << "USHORT_SIZE='\u\"c'" << USHORT_SIZE << endl << flush;
    cout << "UINT_SIZE='\u\"c'" << UINT_SIZE << endl << flush;
    cout << "ULONG_SIZE='\u\"c'" << ULONG_SIZE << endl << flush;
    cout << "ULONG_LONG_SIZE='\u\"c'" << ULONG_LONG_SIZE << endl << flush;
    cout << "FLT_SIZE='\u\"c'" << FLT_SIZE << endl << flush;
    cout << "DBL_SIZE='\u\"c'" << DBL_SIZE << endl << flush;
    cout << "LONG_DBL_SIZE='\u\"c'" << LONG_DBL_SIZE << endl << flush;
    cout << "Real_SIZE='\u\"c'" << REAL_SIZE << endl << flush;
  }

  Real *rp;

  if (RealEQUSHORT) {
    if (verbose) cout << "Real_EQ_USHORT\n";
    typedef unsigned short i_type;
    (Calculate second-largest Real 69)
  }

  else if (RealEQUINT) {
    if (verbose) cout << "Real_EQ_UINT\n";
    typedef unsigned int i_type;
    (Calculate second-largest Real 69)
  }

  else if (RealEQULONG) {
    if (verbose) cout << "Real_EQ ULONG\n";
    typedef unsigned long i_type;
    (Calculate second-largest Real 69)
  }

  else if (RealEQULONG_LONG) {
    if (verbose) cout << "Real_EQ ULONG_LONG\n";
    typedef unsigned long long i_type;
    (Calculate second-largest Real 69)
  }

  else {
  }
  
```

3DLDF-1.1.5.1  §8
§68  3DLDF-1.1.5.1  GET SECOND LARGEST  21

cerr ≜ "ERROR!\nIn_main():\n" "Apparently in_real doesn’t have the same size,\n" "as any unsigned integral type.\n" "There must be some mistake.\n" "Exiting with return value -1.\n\n" ≜ flush;
return -1;
}
if (verbose) {
cout ≜ "MAX_VAL_u-\n" ≜ MAX_VAL ≜ endl ≜ "*rp_u-\n" ≜ *rp ≜ endl ≜
"(MAX_VAL_u-*rp_u-\n" ≜ (MAX_VAL ≡ *rp) ≜ endl ≜ "(MAX_VAL_u-*rp_u-\n"
(MAX_VAL - *rp) ≜ endl ≜ "MAX_VAL_u-\n*rp_u-\n" ≜ (MAX_VAL > *rp) ≜ endl ≜
"MAX_VAL_u\n*rp_u-\n" ≜ (MAX_VAL > *rp) ≜ endl ≜ flush;
}
if (MAX_VAL ≡ *rp) {
cerr ≜ "ERROR!\nIn_System::get_second_largest<Real>():\n" ≜
"MAX_VAL_u-\n*rp_u-\nExiting with return value 1" ≜ endl ≜ endl ≜ flush;
exit(1);
}
else if (MAX_VAL < *rp) {
cerr ≜ "ERROR!\nIn_System::get_second_largest<Real>():\n" ≜
"MAX_VAL_u\n*rp_u-\nExiting 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 *\n&MAX_VAL);
if (verbose) cout ≜ "\n*ip_u-\n" ≜ *ip ≜ endl ≜ flush;
\i\_type bit_pattern_i_type;
\i\_type result;
bit_set<\i\_type *\ CHAR_BIT\> b;
b = *ip;
if (verbose) cout ≜ "b_u(\nMAX_VAL_u-\n" ≜ b ≜ endl ≜ flush;
b = bit_pattern_i_type;
if (verbose) cout ≜ "b_u(\bit_pattern_i_type\_u-\n" ≜ b ≜ endl ≜ flush;
result = bit_pattern_i_type ⊕ *ip;
if (verbose) cout ≜ "result_u-\n" ≜ result ≜ endl ≜ flush;
b = result;
if (verbose) cout ≜ "b_u(result)\_u-\n" ≜ b ≜ endl ≜ flush;
\rp = reinterpret_cast\<\ Real *\\>(\&result);
if (verbose) cout ≜ "\star\rp_u-\n" ≜ \star\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. */ /* This is the case that the highest-order bit of the */ /* mantissa is 1, and all of the other bits (in particular, */ /* all the bits of the exponent) are 0. In this case, *rp */ /* is not a number (NAN). The GNU compiler copes with this, */ /* the DEC compiler signals a floating point error and dumps */ /* core (I believe). I wasn’t able to catch the error with */ /* try and catch. */ /* START HERE. Change (8 + 1) to (FLT_EXP + 1), except */ /* isn’t the right name. Find it, and put here. This */ /* assumes the exponent is at the right, which may not be */ /* true. Skipping this bit pattern is necessary on the DEC */ /* ALPHA, because the float is not a number. */ /* It must be 23 for float, and 52 for double. */
  if (i == (sizeof (Real) * CHAR_BIT) - (12)) {
    if (verbose) cout << "i - u" << i << "This produces NaN, Continuing. \n\n" << flush;
    continue;
  }
  bit_pattern_i_type = 1;
  bit_pattern_i_type <<= i;
  if (verbose)
    cout << "bit_pattern_i_type" << (i << "") << "\n" << bit_pattern_i_type << endl << flush;
  b = bit_pattern_i_type;
  if (verbose) cout << "b = (bit_pattern_i_type)\n" << b << endl << flush;
  result = bit_pattern_i_type | *rp;
  if (verbose) cout << "result\n" << result << endl << flush;
  b = result;
  if (verbose) cout << "b = (result)\n" << b << endl << flush;
  *rp = reinterpret_cast < Real * > (&result);
  if (verbose) cout << "*rp\n" << *rp << endl << flush;
  if (*rp + 0)
    if (verbose) cout << "*rp, Continuing. \n\n" << flush;
    continue;
  else if (*rp >= MAX_VAL)
    if (verbose) cout << "*rp, MAX_VAL\nContinuing.\n\n" << flush;
    continue;
  else if (secondLargest_real + *rp)
    if (verbose) cout << "secondLargest_real, Continuing.\n\n" << flush;
    continue;
  else if (*rp > secondLargest_real && *rp + MAX_VAL)
    if (verbose) cout << "*rp, secondLargest_real, MAX_VAL\nContinuing.\n\n" << flush;
    secondLargest_real = *rp;
    counter = i;
  else
    if (verbose) cout << "Some other condition, Continuing.\n\n";
    continue;
}
```
} /* for */
#endif

This code is cited in section 66.

71. Template function instantiations.

[Log] [LDF 2003.12.29.] Added this section.

(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 gaultpl 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 gsltmplt.h. The file gsltmplt.h must be included by all files that
define specializations of get_second_largest(). [LDF 2003.12.29.]

(gsltmplt.h 75) ≡
(Declare namespace System 34)
(Declare System functions 36)
(Define System functions 37)

76. I/O (io.web).

[Log]

[Version control identifier 5] +≡
static string res_id = "$Id: io.web,v.1.4,2004/01/12,21:30:03,lfinsto1,Exp,$"

77. Include files.

(Include files 6) +≡
#include "loader.h"
#include "pspglb.h"
#include <time.h>

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.

[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.
[LDF 2003.07.16.] Added fig_num.

format ifstream int
format ofstream int

(Global variables 18) +≡
ifstream in_stream;
ofstream out_stream;
ofstream tex_stream;
unsigned short fig_num;
79. **extern** declarations for the global variables.

\[
\{\text{extern variable declarations 79} \equiv \\
\text{extern ifstream } \text{in\_stream}; \\
\text{extern ofstream } \text{out\_stream}; \\
\text{extern ofstream } \text{tex\_stream}; \\
\text{extern unsigned short } \text{fig\_num}; \\
\}\]

This code is used in section 92.

80. **I/O functions.**

81. **Initialize I/O.**

\[
\begin{array}{l}
\text{Log }
\end{array}
\]

\[
\{\text{Declare I/O functions 81} \equiv \\
\text{void } \text{initialize\_io} (\text{string } in\_stream\_name, \text{string } out\_stream\_name, \text{string } tex\_stream\_name, \text{char } *\text{program\_name}); \\
\}
\]

See also sections 84, 86, and 88.

This code is used in section 92.

82.

\[
\{\text{Define I/O functions 82} \equiv \\
\text{void } \text{initialize\_io} (\text{string } in\_stream\_name, \text{string } out\_stream\_name, \text{string } tex\_stream\_name, \text{char } *\text{program\_name})\
\text{\{ time\_t } tt; \\
\text{tm } *lt; \\
\text{tt } = \text{time}(0); \\
\text{lt } = \text{localtime} (&tt); \\
\text{string } \text{datestamp}(\text{asctime}(lt)); \\
\text{datestamp}.\text{erase} (\text{datestamp}.\text{size} () - 1); \quad /* \text{Remove terminal line-feed. */}
\}
\]

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) +=

```c
#define in_stream
    in_stream.open(in_stream_name.c_str());
#endif
    out_stream.open(out_stream_name.c_str());
#else...
    out_stream.setf(ios_base::fixed, ios_base::floatfield);
#endif
	
tex_stream.open(tex_stream_name.c_str()); /* Write datestamp to out_stream. */
    out_stream << "\%\%\%This is\%\%\%" << out_stream_name << "." << endl << "\%\%\%Generated on\%\%\%" <<
        datestamp << ",\%\%\%from\%\%\%" << program_name << "\%\%\%\n"; /* Write datestamp to tex_stream. */
    tex_stream << "\%\%\%This is\%\%\%" << tex_stream_name << "." << endl << "\%\%\%Generated on\%\%\%" <<
        datestamp << ",\%\%\%from\%\%\%" << program_name << "\%\%\%\n"; }
```

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) +=

```c
    void write_footers();
```
§85  3DLDF-1.1.5.1

85.  (Define I/O functions 82) +≡

void write_footers()
{
#if 0
  out_stream ≡ endl ≡ "\%:\n" ≡ "\%\_Variables:\n" ≡ \%mode:Metafont ≡ \%eval:(if \_metapost\_keymap,\nil,(load,"\metapost\_keymap")) ≡ \%eval:(\_use\_local\_map,\_metapost\_mode\_map) ≡ \%eval:(local\_set\_key, [f9], \_mp\_file) ≡ \%run\_mp\_on\_file:"\persp\_mp\" ≡ \%run\_cweb\_on\_file:"\main\_web\" ≡ \%run\_tex\_on\_file:"\persp\_text\" ≡ \%run\_dvips\_on\_file:"\persp\_ps\" ≡ \%\_End:\n",
#endif
return;
}

86.  Begin figure.

Log

[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 2003.07.16.] 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 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
[LD 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.
[LD 2003.08.17.] Made silent non-const. Setting it to true, if SILENT_GLOBAL is true.

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 << 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`.

\[
\begin{array}{l}
\text{(io.h \ 92)} \\
\text{(\{extern variable declarations 79\})} \\
\text{(Declare I/O functions 81)}
\end{array}
\]

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.

\[
\begin{array}{l}
\text{(Version control identifier 5) \ +\} \\
\text{static string res_id = "\$Id:\colors.web,v.1.7.2004/01/12,21:27:38,1$";} \\
\end{array}
\]

94. Include files.

\[
\begin{array}{l}
\text{(Include files 6) \ +\} \\
\text{#include "loader.h"} \\
\text{#include "pspg1b.h"} \\
\text{#include "creatnew.h"} \\
\text{#include "io.h"}
\end{array}
\]

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`.

\[
\begin{array}{l}
\text{(Define class Color 95) \ +\} \\
\text{class Color \{} \\
\text{\quad string name;} \\
\text{\quad bool use_name;} \\
\text{\quad bool on_free_store;} \\
\text{\quad real red_part;} \\
\text{\quad real green_part;} \\
\text{\quad real blue_part;} \\
\text{\quad public: (Declare Color functions 97)} \\
\text{\quad \}}
\end{array}
\]

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.

\[
\begin{array}{l}
\text{(Declare Color functions 97) \ +\} \\
\text{Color();}
\end{array}
\]

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) \equiv
   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, 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!!

\begin{tabular}{ll}
\hline
\textbf{Log} & \textbf{Log} \\
\hline
[LDF 2002.09.25.] Added this function. & \\
\hline
\end{tabular}

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

100. (Define Color functions 98) \equiv
   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 \neq "" \land u \equiv true) {
         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) \equiv
   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 2002.10.09] 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 out_stream 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 98) +≡
    Color::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 < 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 > 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 < 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 > 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 < 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 > 1. Setting blue part to 1.\n";
            blue_part = 1;
        } else blue_part = b;
        return;
    }

109. Setting function. (Declare Color functions 97) +≡
    void set(const real r, const real g, const real b);
\[\text{110.} \]
\begin{verbatim}
Define Color functions 98} 

\begin{verbatim}
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! In Color: Color() with three real arguments:\n" << 
      "Red part argument < 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 > 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 < 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 > 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 < 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 > 1. Setting blue part to 1.\n"
    blue_part = 1;
  } else blue_part = b;
\end{verbatim}
\end{verbatim}

\[\text{111. Pseudo-constructor for dynamic allocation.} \]

\[
\text{Log} \[\text{LDF 2003.12.30.}\] Replaced \text{Color::create_new_color()} with specializations of template} (\text{class C})*create_new()]

\[\text{112. Pointer argument.} \]
\begin{verbatim}
\begin{verbatim}
(Declare non-member template functions for Color 112} 

\begin{verbatim}
Color *create_new(const Color *c);
\end{verbatim}
\end{verbatim}
\end{verbatim}
See also section 113.
This code is used in sections 162 and 163.

113. Reference argument.

[LDF 2004.1.2.] Added this declaration.

(Declare non-member template functions for Color 112) +≡
Color *create_new(const Color &c);


(Declare Color functions 97) +≡
void operator=(const Color &c);

115.

(Define Color functions 98) +≡
void Color::operator=(const Color &c)
{
    name = "n";
    use_name = false;
    red_part = c.red_part;
    green_part = c.green_part;
    blue_part = c.blue_part;
}

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.

(Declare Color functions 97) +≡
bool operator==(const Color &c) const;

117.

(Define Color functions 98) +≡
bool Color::operator==(const Color &c) const
{
    return ((red_part ≡ c.red_part) ∧ (green_part ≡ c.green_part) ∧ (blue_part ≡ c.blue_part));
}

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

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

120. Modifying.

121. Set on free store.

[Log

[LDF 2003.12.30.] Added this function.

(Declare Color functions 97) \[\equiv\]
\[
\text{bool set\_on\_free\_store (bool } b = \text{true);} \]

122.
(Define Color functions 98) \[\equiv\]
\[
\text{bool Color::set\_on\_free\_store (bool } b)
\]
\[
\{ \on\_free\_store = b; \return b; \}
\]

123. Set name.
(Declare Color functions 97) \[\equiv\]
\[
\text{void set\_name (const string } s); \]

124.
(Define Color functions 98) \[\equiv\]
\[
\text{void Color::set\_name (const string } s)
\]
\[
\{ \name = s; \}
\]

125. Set use name.
(Declare Color functions 97) \[\equiv\]
\[
\text{void set\_use\_name (const bool } b); \]

126.
(Define Color functions 98) \[\equiv\]
\[
\text{void Color::set\_use\_name (const bool } b)
\]
\[
\{ \use\_name = b; \}
\]

127. Modify.
(Declare Color functions 97) \[\equiv\]
\[
\text{void modify (const real } r, \text{const real } g = 0, \text{const real } b = 0); \]
128. (Define Color functions 98) +≡
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:" << red_part << endl << "Setting red_part to 1.\n\n";
        red_part = 1;
    } else if (red_part < 0) {
        cerr << "WARNING! In Color::modify():\n" << "red_part is less than 0:" << red_part << endl << "Setting red_part to 0.\n\n";
        red_part = 0;
    }
    if (green_part > 1) {
        cerr << "WARNING! In Color::modify():\n" << "green_part is greater than 1:" << green_part << endl << "Setting green_part to 1.\n\n";
        green_part = 1;
    } else if (green_part < 0) {
        cerr << "WARNING! In Color::modify():\n" << "green_part is less than 0:" << green_part << endl << "Setting green_part to 0.\n\n";
        green_part = 0;
    }
    if (blue_part > 1) {
        cerr << "WARNING! In Color::modify():\n" << "blue_part is greater than 1:" << blue_part << endl << "Setting blue_part to 1.\n\n";
        blue_part = 1;
    } else if (blue_part < 0) {
        cerr << "WARNING! In Color::modify():\n" << "blue_part is less than 0:" << blue_part << endl << "Setting blue_part to 0.\n\n";
        blue_part = 0;
    }
    return;
}

129. Set red part.
(Declare Color functions 97) +≡
void set_red_part(const real r);
130.  
(Define Color functions 98) \equiv
void Color::set_red_part(const real r) 
{
    if (r > 1) {
        cerr \ll \text{"WARNING! In Color::set_red_part():\n\text{\"r, is, greater, than, 1:\"} \ll r \ll \text{endl} \ll \text{Setting, red, part, to, 1.}\n\n"};
        red_part = 1;
    } else if (r < 0) {
        cerr \ll \text{"WARNING! In Color::set_red_part():\n\text{\"r, is, less, than, 0:\"} \ll r \ll \text{endl} \ll \text{Setting, red, part, to, 0.}\n\n"};
        red_part = 0;
    } else red_part = r;
    return;
}

131. Set green part.
(Declare Color functions 97) \equiv
void set_green_part(const real g);

132.  
(Define Color functions 98) \equiv
void Color::set_green_part(const real g) 
{
    if (g > 1) {
        cerr \ll \text{"WARNING! In Color::set_green_part():\n\text{\"g, is, greater, than, 1:\"} \ll g \ll \text{endl} \ll \text{Setting, green, part, to, 1.}\n\n"};
        green_part = 1;
    } else if (g < 0) {
        cerr \ll \text{"WARNING! In Color::set_green_part():\n\text{\"g, is, less, than, 0:\"} \ll g \ll \text{endl} \ll \text{Setting, green, part, to, 0.}\n\n"};
        green_part = 0;
    } else green_part = g;
    return;
}

133. Set blue part.
(Declare Color functions 97) \equiv
void set_blue_part(const real b);
134. Define Color functions 98) \(\equiv\)

```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" << "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" << "Setting blue_part to 0.\n\n";
        blue_part = 0;
    } else blue_part = b;
    return;
}
```

135. Show.

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

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

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

```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;
    return;
}
```

137. Returning elements and information.

138. Is on free store.

```cpp
Log

[LDF 2004.01.06.] Made non-inline.
```

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

```cpp
bool is_on_free_store() const;
```
§139. 3DLDF-1.1.5.1

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  
\begin{verbatim}
bool Color::get_use_name() const  
{  
  return use_name;
}
\end{verbatim}

146.  Get name.  
\{Declare Color functions 97\} +\equiv  
\begin{verbatim}
inline string get_name() const  
{  
  return name;
}
\end{verbatim}

147.  Output operator.  
\{Declare non-member non-template functions for Color 147\} +\equiv  
\begin{verbatim}
ostream & operator<<(ostream & o, const Color &c);  
This code is used in section 163.
\end{verbatim}

148.  
\{Declare non-member non-template functions for Color 148\} +\equiv  
\begin{verbatim}
ostream & operator<<(ostream & o, const Color &c)  
{  
  if (c.get_use_name() +\equiv true)  
  {  
    o +\equiv c.get_name();
  }  
  else  
  {  
    o +\equiv "( " +\equiv c.get_red_part() +\equiv +\equiv " , " +\equiv c.get_green_part() +\equiv +\equiv " , " +\equiv c.get_blue_part() +\equiv +\equiv " )";
  }
  return o;
}
\end{verbatim}
This code is used in section 162.

149.  Define Colors in METAPOST.  
\{Declare Color functions 97\} +\equiv  
\begin{verbatim}
void define_color_mp() const;
\end{verbatim}
150. Define Color functions 98) +≡

\[
\text{void Color::define_color_mp (const }
\]

\[
\text{if (!out_stream.is_open()) }
\]

\[
\text{cerr << "ERROR! In Color::define_color_mp():
\text{n" \n} \n\text{return;}
\]

\[
\text{if (name == "") }
\]

\[
\text{cerr << "ERROR! In Color::define_colors_mp():
\text{n" \n"name is empty. Not doing anything and returning. \n" \n} \n\text{flush;}
\]

\[
\text{out_stream << color: " name << "; " << "\n\text{get_red_part() << "; " \n\text{get_green_part() \n" << "; \n\text{get_blue_part() \n"; \n} \n\text{flush;}
\]

\[
\text{return;}
\]

151. Initialize Colors. [LDF 2002.09.25.] This function presupposes the existence of namespace Colors.

\[
\text{Declare Color functions 97) +≡}
\]

\[
\text{static void initialize_colors();}
\]
152.  
{Define Color functions 98} +≡
void Color::initialize_colors()
{
    using namespace Colors;
    if (!out_stream.is_open()) {
        cerr ≡ "ERROR! In Color::initialize_colors():
            "out_stream is closed! Returning.
        "flush;
        return;
    }
    out_stream ≡ "%% Color definition definitions.

#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_mp();
green.define_color_mp();
blue.define_color_mp();
black.define_color_mp();
white.define_color_mp();
background.define_color_mp();
#endif
yellow.define_color_mp();
cyan.define_color_mp();
magenta.define_color_mp();
orange.define_color_mp();
violet.define_color_mp();
purple.define_color_mp();
yellow_green.define_color_mp();
green_yellow.define_color_mp();
blue_violet.define_color_mp();
gray.define_color_mp();
lit_gray.define_color_mp();
violet_red.define_color_mp();
default_background.define_color_mp();
/* [LDF 2002.09.25] Currently, this function does nothing if I’m using all of the colors. */
out_stream ≡ "\nEnd of Color definitions.\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
unnecessary 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} ≡
{Major Colors 156} /* {All Colors 0} */
This code is cited in section 153.
This code is used in section 162.
The colors "red", "green", "blue", "black", and "white" are already defined in METAPOST, however, we need them here in order to access the Color functions for them.

154. [LDF 2002.09.25.] Here I can put either \{extern Major Colors 157\} or \{extern All Colors 0\} into \{extern namespace Colors declaration 154\}.

\{extern namespace Colors declaration 154\} \equiv
\{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 Color functions for them.

156. Internal (with initialization). [LDF 2002.09.25.] If I add Colors here, remember to add them in the "External" section below, and in the definition of Color::initialize_colors() below.

[LDF 2002.10.26.] Added 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, 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. */
\{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); \} /* Orange. */
\{extern const Color orange("orange", 255, 165, 0, true); \} /* Violet. */
\{extern const Color violet("violet", 238, 130, 238, true);\}
\{extern const Color purple("purple", 160, 32, 240, true);\}
\{extern const Color 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, 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);\}
\{extern const Color default_background("default_background", 255, 255, 255, true);\}
\{extern const Color *default_color = &black;\}
\{extern const Color *background_color = &default_background;\}
\{extern const Color *help_color = &green; \} /* LDF 2002.10.26. Added. */

/* [LDF 2002.09.25.] !! TO DO: default_background is a convenience, in case I change "background" in the METAPOST code. Check METAPOST documentation!! I believe it has something similar. */
157. External.

```cpp
namespace Colors { /* [LDF 2002.09.27.] The ordering should be as above for the internal declarations. */
    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. */
    extern const Color green_yellow;    /* Yellow. */
    extern const Color orange;    /* Orange. */
    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 *default_color;
    extern Color *help_color;    /* LDF 2002.10.26. Added. */
    extern Color *background_color;
}
```

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

158. All Colors. !! If colall.web is changed, I will have to make sure that this file is recompiled! cmp1 does not check the state of colall.web. ?? Should I do something about this, or is it not worth it? If I work on colall.web, I could just put the code back in here.

[LDF 2002.09.26.] If I want all of the colors declared in colall.web, I can uncomment the following line and use (All Colors 0) instead of (Major Colors 156) in (extern namespace Colors declaration 154) above.

@ colall.web (Commented out).

159. Global constants.

<table>
<thead>
<tr>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>[LDF 2002.10.26.] Added help_color_vector.</td>
</tr>
</tbody>
</table>

```
(global constants 22) +≡
namespace Colors {
    extern const vector(const Color *) default_color_vector (1, 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) \equiv
(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) \equiv
  static string rs_id = "$Id: transform.web,v,1.5,04/01/12,21:33:44,ldinstoi,Exp,$";

165. Include files.
(Include files 6) \equiv
#include "loader.h"
#include "pspglb.h"
#include "io.h"
#include "colors.h"

166. Transform class definition. The Transform class has a 4 \times 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) \equiv
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  DEFAULT CONSTRUCTOR  47

(Declare Transform functions 168) ≡
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, 222, 223, 226, and 232.
This code is used in section 166.

169.
(Define Transform functions 169) ≡
Transform::Transform()
{
    reset();
}
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) +≡
Transform(real r);

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

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) +≡
Transform(real r0_0, real r0_1, real r0_2, real r0_3, real r1_0, real r1_1, real r1_2, real r1_3, real r2_0, real r2_1, real r2_2, real r2_3, real r3_0, real r3_1, real r3_2, real r3_3);
173.  

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

Transform : Transform(real r0_0, real r0_1, real r0_2, real r0_3, real r1_0, real r1_1, real r1_2, real r1_3, real r2_0, real r2_1, real r2_2, real r2_3, real r3_0, real r3_1, real r3_2, real r3_3)

\{
  matrix[0][0] = r0_0;
  matrix[0][1] = r0_1;
  matrix[0][2] = r0_2;
  matrix[0][3] = r0_3;
  matrix[1][0] = r1_0;
  matrix[1][1] = r1_1;
  matrix[1][2] = r1_2;
  matrix[1][3] = r1_3;
  matrix[2][0] = r2_0;
  matrix[2][1] = r2_1;
  matrix[2][2] = r2_2;
  matrix[2][3] = r2_3;
  matrix[3][0] = r3_0;
  matrix[3][1] = r3_1;
  matrix[3][2] = r3_2;
  matrix[3][3] = r3_3;
\}

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) \(+\equiv\)

Transform operator=(const Transform &t);

175.  

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

Transform Transform::operator=(const Transform &t)

\{
  for (int i = 0; i < 4; i++)
    for (int j = 0; j < 4; j++) matrix[i][j] = t.matrix[i][j];
  return t;
\}

176.  Reset to identity matrix.

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

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

§178. Setting values.  

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

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

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

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

```cpp
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 Transform 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) +≡

```cpp
static real epsilon();
```

183. (Define Transform functions 169) +≡

```cpp
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 <`epsilon()`>. If so, they should be set to 1.

185. Non-const version.

(Declare Transform functions 168) +≡

```cpp
bool is_identity();
```
186.  
(Define Transform functions 169) \(\equiv\)

```c
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) || ((k != j) && matrix[i][j] != 0)) return false;
    return true;
}
```

187.  const version.

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

```c
bool is_identity() const;
```

188.  
(Define Transform functions 169) \(\equiv\)

```c
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) || ((k != j) && matrix[i][j] != 0)) return false;
    return true;
}
```

189.  Querying.

190.  Get element.

```markdown
| Log | [LDF 2003.07.04.] Added this function. |
```

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

```c
real get_element(const unsigned short row, const unsigned short col) const;
```

191.  
(Define Transform functions 169) \(\equiv\)

```c
real Transform::get_element(const unsigned short row, const unsigned short col) const
{
    return matrix[row][col];
}
```

192.  Show.

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

```c
void show(string text = ") const;
```
193. Define \texttt{Transform} functions 169 \equiv
\begin{verbatim}
void Transform::show (string text) const 
{ 
  if (text == "") text = "Transform:";
  cout \ll text \ll "\n";
  for (int i = 0; i < 4; i++) 
    for (int j = 0; j < 4; j++) 
      \{ \*!! [LDF 2002.02.07.] Can’t use \texttt{left} here, and I can’t include \texttt{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 \ll setw(7) \ll setprecision(3) \ll matrix[i][j] \ll "\n";
    \}
  cout \ll endl;
} 
cout \ll endl \ll flush;
\}
\end{verbatim}

The functions for the affine transformations all return a \texttt{Transform} representing the transformation, not \texttt{*this}. This makes it possible to chain expressions using \texttt{operator*=()}, e.g.,

\begin{verbatim}
Point pt0;
Point pt1 (1, 2, 3);
Point pt2 (3, 4, 5);
Transform t;
t.rotate(90, 90, 90);
pt0 *= pt1 *= pt2 *= t;
\end{verbatim}

\texttt{pt0}, \texttt{pt1}, and \texttt{pt2} are all rotated 90° around the x, y, and z-axes.

195. Scale.

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

[\texttt{LDF 2002.10.15.}] BUG FIX: If the absolute value of an argument is < \texttt{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.

[\texttt{LDF 2003.03.25.}] BUG FIX: Fixed conditional that tests whether all the arguments are 1.

\begin{verbatim}
(Declare \texttt{Transform} functions 168) \equiv
Transform scale(real x, real y = 1, real z = 1);
\end{verbatim}
196. Define Transform functions 169} +≡

Transform Transform : scale(real x, real y, real z)
{
    Transform t;
    if (x ≡ 1 ∧ y ≡ 1 ∧ z ≡ 1) {
        cerr << "WARNING! In Transform::scale():\n" <<
            "All arguments==1.0. Returning identity transformation." << 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(real xy, real zz = 0, real yx = 0, real yz = 0, real zx = 0, real y = 0);
198.  
(Define Transform functions 169) +≡

Transform Transform := shear(real xy, real xz, real yz, real xz, real yz)
{
    Transform t;
    real eps = epsilon();
    if (fabs(xy) < eps) xy = 0;
    if (fabs(xz) < eps) xz = 0;
    if (fabs(yz) < eps) yz = 0;
    if (fabs(xz) < eps) xz = 0;
    if (fabs(yz) < eps) yz = 0;
    if (xy == 0 && xz == 0 && yz == 0) {
        cerr « "WARNING! In Transform::shear():\n        "All arguments are zero. Returning identity transformation." « endl << endl << flush;
        return t;
    }
    t.matrix[1][0] = xy;
    t.matrix[2][0] = xz;
    t.matrix[0][1] = yx;
    t.matrix[2][1] = yz;
    t.matrix[0][2] = xz;
    t.matrix[1][2] = yz;
    (*this) *= t;
    clean();
    return t;
}

199. Shift.  (Translation.)

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

Transform shift(real x, real y = 0, 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 ≡ 0 ∨ y ≡ 0 ∨ z ≡ 0) {
    if (DEBUG) cerr ≪ "In rotate(real,real,real)\n" ≪ "0_rotation_about_all_axes..Returning_identity_matrix.\n" ≪ 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 != 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) ≥ eps ? temp2 : 0);
    t_x.matrix[i][1] = (fabs(temp1) ≥ eps ? temp1 : 0);
  }
} /* if */

208. Rotation around the y-axis.

Define Transform functions 169 +≡

if (y != 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) ≥ eps ? temp2 : 0);
    t_y.matrix[i][0] = (fabs(temp1) ≥ eps ? temp1 : 0);
  }
} /* if */
209. Rotation around the z-axis.

(Define Transform functions 169) +≡

if (z ≠ 0) {
    z *= P1 / 180.0;
    s = sin(z);
    c = cos(z);
    for (int i = 0; i < 4; i++) {
        temp1 = (t.x.m[i][0] * c) - (t.y.m[i][1] * s);
        temp2 = (t.x.m[i][0] * s) + (t.y.m[i][1] * c);
        t.m[i][1] = (fabs(temp2) ≥ eps) ? temp2 : 0;
        t.m[i][0] = (fabs(temp1) ≥ eps) ? temp1 : 0;
    }
} /∗ if ∗/

All * = t.x;
All * = t.y;
All .clean();
(*this) * = All;
clean();
return All; } /∗ End of rotate(). ∗/

210. Rotation around an arbitrary axis.

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

LDF 2002.4.7. Added default value for angle ≡ 180.
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(Point p0, Point p1, const real angle = 180);
212. Path argument. [LDF 2002.05.03.] Defined in paths.web because Path is an incomplete type here. 

[LDF 2002.06.03.] 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. 

[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. 

\{ Define 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; 
}


[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\=(-). 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.  
(Define Transform functions 169) +≡  
Transform Transform ::operator*=(const Transform &t) 
{
    bool DEBUG = false;  /* true */
    if (DEBUG) cout << "Entering Transform::operator*=.\n" << flush;
    if (t.is_identity()) {
        if (DEBUG) cout << "t.is_identity.transform\n" << "Returning t.\n" << flush;
        return t;
    } else if (is_identity()) {
        if (DEBUG) cout << "Setting this.to.t. and returning t.\n" << flush;
        return (*this = t);
    }  
    Matrix temp_matrix = {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}};
    for (int i = 0; i < 4; i++)
       for (int k = 0; k < 4; k++)
           for (int j = 0; j < 4; j++)
               temp_matrix[i][k] += matrix[i][j] * t.matrix[j][k];
    for (int i = 0; i < 4; i++)
       for (int j = 0; j < 4; j++)
           matrix[i][j] = temp_matrix[i][j];
    clean();
    if (DEBUG) cout << "Exiting Transform::operator*=.\n" << flush;
    return t;
}

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

222.  
(Define Transform functions 169) +≡  
Transform Transform ::operator*(const real r) const 
{
    Transform t = *this;
    t *= r;
    t.clean();
    return t;
}

223. Transform argument.
(Declare Transform functions 168) +≡  
Transform operator*(const Transform t) const;
224. Define Transform functions 169 +≡

```cpp
Transform Transform :: operator*(const Transform t) const
{
    Transform a = *this;
    a *= t;
    a.clean();
    return a;
}
```

225. Matrix inversion.

226. const version (no assignment). It would be easy to generate the inverses of the transformations that I call explicitly using `rotate()`, `shift()`, etc., as I go along. However, it is not possible to do this for the ones produced using `operator*()` and `operator*==()`. So, since a matrix inversion routine is needed anyway, I don’t bother to generate the inverses as I go along.

TO DO: Get format for references! `inverse()` uses the Gauff-Jordan algorithm with column pivot search. I’ve taken the algorithm from Stoer, Josef. *Numerische Mathematik 1* and adapted it to C++.

Log

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

(Declare Transform functions 168) +≡

```cpp
Transform inverse() const;
```
227.  

\{Define Transform functions 169\} +≡

```c++
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 hw[n];
  Transform t;
  if (DEBUG) {
    cout << "matrix_" << '\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';
    }
  }
  if (DEBUG) {
    cout << '\n';
    t.show("t");
    cout << "Enter \<RETURN> to continue. \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.

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

229. Row exchange.

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


\(\text{Define Transform functions 169} \equiv\)
\[\text{if (DEBUG) cout << "t.matrix["} << j << "]["} << j << "] = t.\text{matrix}[j][j] << endl << flush;\]
\[\quad hr = 1/t.\text{matrix}[j][j];\]
\[\text{for } (i = 0; i < n; i++) t.\text{matrix}[i][j] = hr * t.\text{matrix}[i][j];\]
\[\quad t.\text{matrix}[j][j] = hr;\]
\[\text{for } (k = 0; k < n; k++)\]
\[\quad \text{if (k != j)} \{\]
\[\quad\quad \text{for } (i = 0; i < n; i++)\]
\[\quad\quad\quad \text{if (i != j) t.\text{matrix}[i][k] -= t.\text{matrix}[i][j] * t.\text{matrix}[j][k];}\]
\[\quad\quad\quad t.\text{matrix}[j][k] *= -hr;\]
\[\quad \}\]\n\(\text{/* outer for */}\)
\[\text{if (DEBUG) cout << "Finished Transformation.\n" << flush;};\]
231. Column exchange.

(Define Transform functions 169) +≡

```
for (i = 0; i < n; i++) {
    for (k = 0; k < n; k++) 
        hv[i][k] = t.matrix[i][k];
    for (k = 0; k < n; k++) 
        t.matrix[i][k] = hv[k];
}  /* for */
if (DEBUG) {
    cout << "Finished, column, exchange.\n" << flush;
    cout << "Exiting, Transform::inverse()." << endl << flush;
}
if (DEBUG) cout << "Exiting, Transform::inverse()." << endl << flush;
return 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 ≡ false, since it is equivalent to the const version above with no argument. If it is called with assign ≡ false, a warning is issued, the const version is invoked, and its return value is returned.

(Declare Transform functions 168) +≡

```
Transform inverse(bool assign);
```

233.

(Define Transform functions 169) +≡

```
Transform Transform::inverse(bool assign)
{
    bool DEBUG = false;  /* true */
    if (DEBUG) cout << "Entering, Transform::inverse(bool, assign).\n" << flush;
    if (assign ≡ false) {
        cerr << "WARNING: In, Transform::inverse(bool, assign):\n" <<
            "assign, ≡, false, There’s, no, reason, to, error,\n" <<
            "call, this, function, with, a, false, argument, \n" <<
            "but, it, doesn’t, do, any, harm., Continuing.\n" << flush;
        return inverse();
    }
    if (DEBUG) cout << "Exiting, Transform::inverse(bool, assign).\n" << flush;
    return (*this = inverse());
}
```

234. Global variables.

(Global variables 18) +≡

```
Transform user_transform;
```
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`.

```c
#include "transfor.h"
#include "loader.h"
#include "pspglb.h"
#include "io.h"
#include "colors.h"
#include "transfor.h"
```

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`.

---

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.

```c
static string res_id = "$id:shapes.web,v,1.4,2004/01/12,21:32:30,1finsto1,Exp$";
```

242. Include files.

```c
#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.

**Note:** 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.

---

Log

[LDF 2003.05.16.] Added declarations of `get_minimum_z()` and `get_mean_z()`.

```c
format Shape int
```

```c
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 UNDRAWDOT;
static const signed short UNDRAW;
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

```cpp
(Define Shape class 244) +≡
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_x() const = 0;
  virtual real get_maximum_x() const = 0;
  virtual real get_mean_x() const = 0;
  virtual const valarray<real> get_extremes() const = 0;
  virtual void suppress_output() = 0;
  virtual void unsuppress_output() = 0;
} ;
```

246. Static data members.

```cpp
(Define static Shape member variables 246) +≡
  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.

(Define Shape class 244)

250. Picture and Label (pictures.web).

[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.

(Version control identifier 5) +1

static string vcs_id = "$Id::pictures.web,v.1.4,,2004/01/12,21:31:45.11f1st01,Exp.,$";

251. Include files.

(Include files 6) +1

#include "loader.h"
#include "psglbl.h"
#include "io.h"
#include "colors.h"
#include "transform.h"
#include "shapes.h"

252. Label. [LD 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", "ltf", "rt", "ltf", "lft", "lft", or "urt".

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

format Label int

(Define classes 253) +1

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. D0_LABELS is used for globally enabling or suppressing putting Labels onto Pictures. If D0_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. D0_LABELS was formerly a global variable defined in pspglb.web.

(Initialize static Label data members 254) \equiv
bool Label : D0_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.

(Declare Label functions 255) \equiv
void output(const Focus &f, const unsigned short proj, real factor, const Transform &t);
Label *getCopy () const;
This code is used in section 253.

256. namespace Projections.

$log$

[LDf 2003.06.11.] Added AXON.

(Declare namespace Projections 256) \equiv
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.

\[
\text{extern declaration of namespace Projections 257 } \equiv \\
\text{namespace Projections } \\
\text{\{ } \\
\text{\quad extern const unsigned short PERSP; } \\
\text{\quad extern const unsigned short PARALLEL_X_Y; } \\
\text{\quad extern const unsigned short PARALLEL_X_Z; } \\
\text{\quad extern const unsigned short PARALLEL_Z_Y; } \\
\text{\quad extern const unsigned short AXON; } \\
\text{\quad extern const unsigned short ISO; } \\
\text{\}; } \\
\]

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.

\[
\text{Log } \quad \text{Log } \\
\]

\[
\text{[LDF 2003.06.16.] Added this namespace.} \\
\]

\[
\text{Declare namespace Sorting 258 } \equiv \\
\text{namespace Sorting } \\
\text{\{ } \\
\text{\quad extern const unsigned short NO_SORT = 0; } \\
\text{\quad extern const unsigned short MAX_Z = 1; } \\
\text{\quad extern const unsigned short MIN_Z = 2; } \\
\text{\quad extern const unsigned short MEAN_Z = 3; } \\
\text{\}; } \\
\]

This code is used in section 305.
259. **External.**

\[
\text{(extern declaration of namespace Sorting 259)} \equiv \\
\text{namespace Sorting} \\
\text{\{ \\
\text{\hspace{1em} extern const unsigned short NO\_SORT; \\
\text{\hspace{1em} extern const unsigned short MAX\_Z; \\
\text{\hspace{1em} extern const unsigned short MIN\_Z; \\
\text{\hspace{1em} extern const unsigned short MEAN\_Z; \\
\text{\}}} \\
\text{\}; }
\]

This code is used in section 306.

260. **Picture.**

```cpp
\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().

\[
\text{(Define classes 253)} + \equiv \\
\text{class Picture} \\
\text{\{ \\
\text{\hspace{1em} Transform transform; \\
\text{\hspace{1em} vector\langle \text{Shape } \ast \rangle shapes; \\
\text{\hspace{1em} vector\langle \text{Label } \ast \rangle labels; \\
\text{\hspace{1em} bool do\_labels; \\
\text{\}}} \\
\text{\public: } \langle \text{Declare Picture functions 263} \rangle \\
\text{\}; }
\]

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) \equiv
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 undefined 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) \equiv
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 \TeX macro for “PORTING”. !! PORTING. [LDF 2002.12.05.] Moved to points.web because Picture::{clear()} and Label::{get_copy()} are undefined in this file. This didn't cause a problem with the DEC compiler, but it does with the GNU Compiler.
(Declare Picture functions 263) \equiv
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.

270. Add Shape.
(Declare Picture functions 263) \equiv
void operator+=(Shape *s);
271.  
(Define Picture functions 264) \(\equiv\)

\[
\begin{align*}
\text{void Picture}\_\text{::}\text{operator}\_\text{+=}(\text{Shape} \ast s) \\
\{ \\
\quad \text{shapes}\.\text{push}\_\text{back}(s); \\
\}
\end{align*}
\]

272.  
(Declare Picture functions 263) \(\equiv\)

\[
\begin{align*}
\text{void operator}\_\text{+=(Label} \ast \text{label});
\end{align*}
\]

273.  
(Define Picture functions 264) \(\equiv\)

\[
\begin{align*}
\text{void Picture}\_\text{::}\text{operator}\_\text{+=}(\text{Label} \ast \text{label}) \\
\{ \\
\quad \text{labels}\.\text{push}\_\text{back}(\text{label});
\}
\end{align*}
\]

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) \(\equiv\)

\[
\begin{align*}
\text{inline void suppress_labels}() \\
\{ \\
\quad \text{do_labels} = \text{false};
\}
\end{align*}
\]

275.  
Unsuppress Labels.

[Log]

[LDF 2002.12.20.] Added this function.

(Declare Picture functions 263) \(\equiv\)

\[
\begin{align*}
\text{inline void unsuppress_labels}() \\
\{ \\
\quad \text{do_labels} = \text{true};
\}
\end{align*}
\]

276.  
Kill Labels.

[Log]

[LDF 2003.05.07.] Added this function.

(Declare Picture functions 263) \(\equiv\)

\[
\begin{align*}
\text{void kill_labels}();
\end{align*}
\]
277.  
\begin{verbatim}
\{Define Picture functions 264 \} \equiv
  void Picture::killLabels()
  {
    labels.clear();
  }
\end{verbatim}

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

279.  Affine transformations.

280.  Scale.
\begin{verbatim}
\{Define Picture functions 264 \} \equiv
  Transform scale(real x, real y = 1, real z = 1);
\end{verbatim}

281.
\begin{verbatim}
\{Define Picture functions 264 \} \equiv
  Transform Picture::scale(real x, real y, real z)
  {
    return transform.scale(x, y, z);
  }
\end{verbatim}

282.  Shift. (Translation.)

283.  real version.
\begin{verbatim}
\{Declare Picture functions 263 \} \equiv
  Transform shift(real x, real y = 0, real z = 0);
\end{verbatim}

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

285.  Point version. This function must defined in points.web, because Point is an incompletely
defined type here.
\begin{verbatim}
\{Declare Picture functions 263 \} \equiv
  Transform shift(const Point &p);
\end{verbatim}

286.  Rotation around the main axes. \texttt{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 \texttt{rotate()} “knows” about which axis (or axes) to rotate.
\begin{verbatim}
\{Declare Picture functions 263 \} \equiv
  Transform rotate(const real x, const real y = 0, const real z = 0);
\end{verbatim}
§287  3DLDF-1.1.5.1  ROTATION AROUND THE MAIN AXES  75

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.

   [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 add shear! */

289.  Set transform.

   [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.

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

(Declare Picture functions 263) +≡
   Transform operator*=(const Transform &t);
(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: " ≡ text ≡ "\n" ≡ flush;
      transform.show("transform:");
      cout ≡ "shapes.size() = " ≡ shapes.size() ≡ endl ≡ flush;
      cout ≡ "labels.size() = " ≡ 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 \texttt{min_x_proj}, \texttt{max_x_proj}, etc. The purpose of these is to suppress output of \texttt{Shapes} whose \texttt{projective_extremes} fall outside of these limits, whereby the "z" values are not currently checked. They are not set for a particular \texttt{Focus} or \texttt{Picture}, but for a particular invocation of \texttt{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 \texttt{Picture} or the \texttt{Focus}, if that turns out be more convenient. They are checked in \texttt{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);

300. Clear. Defined in \texttt{points.web}. 
(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.

(pictures.h 306) ≡

(extern declaration of namespace Projections 257)
(extern declaration of namespace Sorting 259)
(Define classes 253)
(Declarations for the header file 21)

307. Point (points.web).
[LD 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 3DLDF and that Point has the most functions of any class in 3DLDF. Many of the functions in the other classes do little more than apply the Point version of the function to their Points.

Log
[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.

(Version control identifier s) ≡

static string res_id = "$Id: points.web,v.1.6,2004/01/12,21:32:01,lfinsto1,Exp,\$";

308. Include files.

(Include files 6) ≡

#include "loader.h"
#include "pspg1b.h"
#include "creatinew.h"
#include "io.h"
#include "colors.h"
#include "transfor.h"
#include "shapes.h"
#include "pictures.h"

309. Point class definition. [LD 2002.10.20]
- worldCoordinates contains the coordinates of the Point in the global coordinate system.
- userCoordinates and viewCoordinates are not currently used.
- userCoordinates 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 worldCoordinates can be derived from the userCoordinates by using the appropriate transformation.
- projectiveCoordinates 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 (axiometric, etc.) soon.
- transform is used for storing the transformations that are applied the Point. It is not necessary to update worldCoordinates (or userCoordinates, 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 `ostream`. 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) \equiv

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) ≡

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 cwwdriver.web.

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

312. point_pair and bool_point_pair.

(Define definitions 15) ≡

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.

```cpp
// Type definitions +
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.

```c
{ Define bool_point functions 314 } ⇔
bool_point::bool_point()
{  
  b = false;
  pt = INVALID_POINT;
}
This code is cited in section 313.
This code is used in section 633.
```

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.

```c
{ Type definitions 15 } ⇔
struct bool_point_quadruple {
  bool_point first;
  bool_point second;
  bool_point third;
  bool_point fourth;
  bool_point_quadruple();
  bool_point_quadruple(bool_point a, bool_point b, bool_point c, bool_point d) : first(a), second(b), third(c), fourth(d) { }
  void operator=(const bool_point_quadruple &arg)
  {
    first.b = arg.first.b;
    first.pt = arg.first.pt;
    second.b = arg.second.b;
    second.pt = arg.second.pt;
    third.b = arg.third.b;
    third.pt = arg.third.pt;
    fourth.b = arg.fourth.b;
    fourth.pt = arg.fourth.pt;
  }
};
```
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.
(Type 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.

(Define `bool_real_point` functions 318) \equiv
```
bool_real_point::bool_real_point()
  : b(false), r(INVALID_REAL), pt(INVALID_POINT) {
```
This code is used in section 633.

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

(Global constants 22) \equiv
```
extern const Point INVALID_POINT(INVALID_REAL, INVALID_REAL, INVALID_REAL);
extern const Point origin(0, 0, 0);
extern const bool_point INVALID_BOOL_POINT(false, INVALID_POINT);
extern const bool_point_pair INVALID_BOOL_POINT_PAIR(INVALID_BOOL_POINT,
  INVALID_BOOL_POINT);
extern const bool_real_point INVALID_BOOL_REAL_POINT(false, INVALID_REAL, INVALID_POINT);
extern const bool_point_quadruple INVALID_BOOL_POINT_QUADRUiple(INVALID_BOOL_POINT,
  INVALID_BOOL_POINT, INVALID_BOOL_POINT, INVALID_BOOL_POINT);
```

320. (Declarations for the header file 21) \equiv
```
extern const Point INVALID_POINT;
extern const Point origin;
extern const bool_point INVALID_BOOL_POINT;
extern const bool_point_pair INVALID_BOOL_POINT_PAIR;
extern const bool_real_point INVALID_BOOL_REAL_POINT;
extern const bool_point_quadruple INVALID_BOOL_POINT_QUADRUiple;
```

321. Constructors and setting functions.
322. The valarrays I use for the various sets of coordinates can be declared in the class declaration, but neither can their size be set nor can they be initialized. null_coordinates is defined in pspglb.web and is a valarray of reals with 4 elements = 0. Setting world_coordinates, etc. to null_coordinates makes them the right size.

323. Initialize coordinates and limits.

[LDF 2002 4.3] Now setting world_coordinates[3], user_coordinates[3], and view_coordinates[3] = 1. It fixes a bug that showed up when I tried to shift a Point with coordinates ≡ 0.

```c
#include "__DECCXX"

world_coordinates = null_coordinates;
user_coordinates = null_coordinates;
view_coordinates = null_coordinates;
projective_coordinates = null_coordinates;
#endif
#ifdef __GNUC__
world_coordinates.resize(4, 0);
user_coordinates.resize(4, 0);
view_coordinates.resize(4, 0);
projective_coordinates.resize(4, 0);
#endif

pen = "n";
transform.reset();
#endif

world_coordinates[3] = 1;
user_coordinates[3] = 1;
view_coordinates[3] = 1;
projective_extremes.resize(6, 0);
```

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


(Declare Point constructors 324) ≡

```c
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.

[Log] [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.

[Log] [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);
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.

[Log]

[Log]

[Declade 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 ()
  {}
343. (Define Point functions 330) +≡
   
   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.

   Log

   [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 transform. It doesn’t seem worthwhile to set draw.dot.value, draw.dot.color, or pen to any particular values.

(Declare Point functions 329) +≡
   void clear();

345. (Define Point functions 330) +≡
   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) +≡
   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 .0000000001;
      #else
         return .00001;
      #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 ??

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.

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

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);
\end{verbatim}

356. **Define Point functions 330** \+\+

\begin{verbatim}
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 == 0) f = &default_focus;
    project(f, proj, factor);
  }
  return projective_coordinates;
}
else {
  cerr << "ERROR! In Point::get_all_coords():\n" ##
      "Argument coords has invalid value: " << coords <<
      "Returning world_coordinates.\n" << flush;
  return world_coordinates;
}
\end{verbatim}

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

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

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;
\end{verbatim}

358. **Define Point functions 330** \+\+

\begin{verbatim}
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;
}
\end{verbatim}

359. **Get coord.** [LDF 2002.09.14] Added get_coord(). Fixing a bug that caused get_z('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. 

\[
\begin{align*}
\text{real get\_coord}\,(\text{char } c, \text{char coords} = \text{'w'}, \text{const bool do\_persp = true, const bool do\_apply = true, Focus *f = 0, const unsigned short proj = Projections::PERSP, real factor = 1})
\end{align*}
\]

361. 

\[
\begin{align*}
\text{real Point} &::\text{get\_coord}\,(\text{char } c, \text{char coords, const bool do\_persp, const bool do\_apply, Focus *f, const unsigned short proj, real factor})
\end{align*}
\]

\[
\begin{align*}
&\begin{cases}
&\text{if (do\_apply) apply\_transform();}
&\text{if (f == 0) } f = &\text{default\_focus;}
&\text{unsigned short ctr;}
&\text{c = tolower(c);}
&\text{if (c == 'x') } ctr = 0;
&\text{else if (c == 'y') } ctr = 1;
&\text{else if (c == 'z') } ctr = 2;
&\text{else if (c == 'w') } ctr = 3;
&\text{else }
&\text{cerr << "ERROR! In Point::get\_coord(): " }"\text{Invalid c, argument: }c" \text{ " }c \text{ " }\text{\n" }"\text{Using x, n}" \text{ }\text{flush;}
&\text{ctr = 0;}
\end{cases}
\end{align*}
\]

\[
\begin{align*}
&\text{coords = tolower(coords);}
&\text{if (coords == 'w') return world\_coordinates[ctr];}
&\text{else if (coords == 'u') return user\_coordinates[ctr];}
&\text{else if (coords == 'p') }
&\text{if (do\_persp) project(*f, proj, factor);}
&\text{return projective\_coordinates[ctr];}
\end{align*}
\]

\[
\begin{align*}
&\text{else if (coords == 'v') return view\_coordinates[ctr];}
&\text{else }
&\text{cerr << "ERROR! In Point::get\_coord(): " }"\text{Invalid coord, arg: }coords" \text{ " }coords \text{ " endl }"\text{Returning INVALID\_REAL, \n\n" }\text{flush;}
&\text{return INVALID\_REAL;}
\end{align*}
\]

362. const version. 

\[
\begin{align*}
\text{real get\_coord}\,(\text{char } c, \text{char coords = 'w'}, \text{const bool do\_persp = true, const bool do\_apply = true, Focus *f = 0, const unsigned short proj = Projections::PERSP, real factor = 1})\text{ const;}
\end{align*}
\]
363. (Define Point functions 330) \[+\]
real Point :: get_coord(char c, char coords, const bool do_persp, const bool do_apply, Focus *f, const unsigned short proj, real factor) const
{
  Point p(*this);
  return p.get_coord(c, coords, do_persp, do_apply, f, proj, factor);
}

364. Get x.

(Declare Point functions 329) \[+\]
real get_x(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;

366. (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)
{
  return get_coord('x', coords, do_persp, do_apply, f, proj, factor);
}

367. const version.
(Declare Point functions 329) \[+\]
real get_x(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;

368. (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);
}

369. Get y.

(Declare Point functions 329) \[+\]
real get_y(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);
371. 
(Define Point functions 330) +
real Point \( \rightarrow \) 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 = 'w', 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 \( \rightarrow \) get_y(char coords, const bool do_persp, const bool do_apply, Focus *f, const unsigned short proj, real factor) const
{
    return get_coord('y', 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 \( \rightarrow \) 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 \( \rightarrow \) 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.
(Declare Point functions 329) \( + \equiv \)
\[
\text{real get\_w(char coords = 'w', const bool do\_persp = true, const bool do\_apply = true, Focus *f = 0, const unsigned short proj = Projections::PERSP, real factor = 1);}\\
\]

381. (Define Point functions 330) \( + \equiv \)
\[
\text{real Point::get\_w(char coords, const bool do\_persp, const bool do\_apply, Focus *f, const unsigned short proj, real factor)}\\
\{
return get\_coord('w', coords, do\_persp, do\_apply, f, proj, factor);\\
\}
\]

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

383. (Define Point functions 330) \( + \equiv \)
\[
\text{real Point::get\_w(char coords, const bool do\_persp, const bool do\_apply, Focus *f, const unsigned short proj, real factor) const}\\
\{
return get\_coord('w', coords, do\_persp, do\_apply, f, proj, factor);\\
\}
\]

384. Get transform.

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

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

385. Get copy.

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

(Declare Point functions 329) \( + \equiv \)
\[
\text{Shape *get\_copy() const;}\\
\]
386.  
\begin{verbatim}
  (Define Point functions 330) +≡
  Shape *Point::get_copy() const { Point *p = create_new < Point > (0);
    *p = *this;
    return static_cast< Shape * >(p); }
\end{verbatim}

387.  Is on free store.  
\begin{verbatim}
[ LDF 2004.01.06. ] Made non-inline.
\end{verbatim}

\begin{verbatim}
  (Declare Point functions 329) +≡
  bool is_on_free_store() const;
\end{verbatim}

388.  
\begin{verbatim}
  (Define Point functions 330) +≡
  bool Point::is_on_free_store() const
  {
    return on_free_store;
  }
\end{verbatim}

389.  Slope.  \begin{verbatim}
[ LDF 2002.10.27. ] slope() returns the slope of the trace of the line
from \texttt{this} to \texttt{p} on the plane indicated by the \texttt{char}
arguments \texttt{m} and \texttt{n}. These should be \texttt{'x'}, \texttt{'y'}, \texttt{'z'}, \texttt{'X'}, \texttt{'Y'}, or \texttt{'Z'}.
\end{verbatim}
\begin{verbatim}
[ LDF 2004.01.06. ] Made non-inline.
\end{verbatim}

\begin{verbatim}
[ LDF 2002.10.27. ] Now using \texttt{worldCoordinates} directly instead of
"get" functions.
[ LDF 2002.10.27. ] Changed argument \texttt{p} from \texttt{const Point & Point}.
\end{verbatim}

\begin{verbatim}
  (Declare Point functions 329) +≡
  real slope(Point p, char m = 'x', char n = 'y') const;
\end{verbatim}
390. Define Point functions 330) +:

```cpp
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 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];
    return (a_m_coord - p_m_coord) / (a_m_coord - p_m_coord);
```}

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.

```cpp
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.] \( \text{is\_on\_segment}() \) returns a \texttt{bool\_real} with the \texttt{bool} indicating whether \texttt{*this} lies on the line segment between \( p0 \) and \( p1 \), and a \texttt{real} value \( t \) representing the distance of \texttt{*this} on the way from \( p0 \) to \( p1 \). If the \texttt{bool} is \texttt{true}, then \( 0 \leq t \leq 1 \). If \( t < 0 \) or \( t > 1 \), then \texttt{*this} lies on the line passing through \( p0 \) and \( p1 \), but not on the segment. If \texttt{*this} doesn't lie on the line, \( t \) will be \texttt{INVAL\_ID\_REAL}.

[LDF 2002.10.29.] To check whether \texttt{*this} lies on the line, use \texttt{is\_on\_line}().

---

Log

[LDF 2002.10.29.] 

BUG FIX: Added code to check whether the unit vectors \( \texttt{*this} - p0 \) and \( p1 - \texttt{*this} \) are equal before calculating \( r \). Before I did this, \texttt{true} was returned for \texttt{Points} that weren't on the line segment.

[LDF 2002.10.29.] Now using \textit{worldcoordinates} directly instead of \texttt{get\_x()}, \texttt{get\_y()}, and \texttt{get\_z()}.  

(Declare \texttt{Point} functions 329) +

\begin{verbatim}
bool\_real is\_on\_segment(Point p0, Point p1);
\end{verbatim}
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 = (world_coordinates[0] - p0.world_coordinates[0])/(p1.world_coordinates[0] - p0.world_coordinates[0]);
else if (p1.world_coordinates[1] != p0.world_coordinates[1])
  r = (world_coordinates[1] - p0.world_coordinates[1])/(p1.world_coordinates[1] - p0.world_coordinates[1]);
else if (p1.world_coordinates[2] != p0.world_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) \(\equiv\)
   \[
   \begin{align*}
   \text{bool\_real } & \text{is\_on\_segment}(\text{const Point } &p0, \text{const Point } &p1) \text{ const} \\
   & \quad \{ \\
   & \quad \text{Point } a(\ast\text{this}); \\
   & \quad \text{return } a, \text{is\_on\_segment}(p0, p1); \\
   & \} \\
   \end{align*}
   \]

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.

399. (Define Point functions 330) \(\equiv\)
   \[
   \begin{align*}
   \text{bool\_real } & \text{is\_on\_line}(\text{const Point } &p0, \text{const Point } &p1) \text{ const}; \\
   \end{align*}
   \]

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

   [LDF 2003.06.04.] Added this function.

401. Is in triangle. [LDF 2003.06.11.] This function returns \textit{true}, if \textit{*this} lies within the triangle defined by the three Point arguments, otherwise \textit{false}. Defined in \texttt{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 \texttt{test\_points}.

402. Transformations.

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

[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.

\begin{verbatim}
(Declare Point functions 329) +≡
 Transform rotate(const real x, const real y = 0, const real z = 0);
\end{verbatim}

405.

(Define Point functions 330) +≡
\begin{verbatim}
 Transform Point :: rotate(const real x, const real y, const real z)
 { return transform.rotate(x, y, z);
 }
\end{verbatim}

406. Scale.

[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.

\begin{verbatim}
(Declare Point functions 329) +≡
 Transform scale(real x, real y = 1, real z = 1);
\end{verbatim}

407.

(Define Point functions 330) +≡
\begin{verbatim}
 Transform Point :: scale(real x, real y, real z)
 { return transform.scale(x, y, z);
 }
\end{verbatim}

408. Shear.

[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.

\begin{verbatim}
(Declare Point functions 329) +≡
 Transform shear(real xy, real zz = 0, real yx = 0, real yz = 0, real zx = 0, real zy = 0);
\end{verbatim}

409.

(Define Point functions 330) +≡
\begin{verbatim}
 Transform Point :: shear(real xy, real zz, real yx, real yz, real zx, real zy)
 { return transform.shear(xy, zz, yx, yz, zx, zy);
 }
\end{verbatim}

410. Shift.

411. Point versions.
412. Three real arguments.
(Declare Point functions 329) \(\equiv\)
  Transform \(\text{shift}(\text{real } x, \text{real } y = 0, \text{real } z = 0)\);

413.
(Define Point functions 330) \(\equiv\)
  Transform \(\text{Point::shift(\text{real } x, \text{real } y, \text{real } z)}\)
  {
    Transform \(t\);
    if \((x \neq 0 \lor y \neq 0 \lor z \neq 0)\) transform \(\equiv t.\text{shift}(x, y, z)\);
    return \(t\);
  }

414. Point argument.
(Declare Point functions 329) \(\equiv\)
  Transform \(\text{shift(\text{const } \text{Point } \& p)}\);

415.
(Define Point functions 330) \(\equiv\)
  Transform \(\text{Point::shift(\text{const } \text{Point } \& p)}\)
  {
    return \(\text{shift}(p.\text{get}_x(), p.\text{get}_y(), p.\text{get}_z())\);
  }

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) \(\equiv\)
  Transform \(\text{Transform::shift(\text{const } \text{Point } \& p)}\)
  {
    return \(\text{shift}(p.\text{get}_x(), p.\text{get}_y(), p.\text{get}_z())\);
  }

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) \(\equiv\)
  Transform \(\text{Picture::shift(\text{const } \text{Point } \& p)}\)
  {
    return \(\text{shift}(p.\text{get}_x(), p.\text{get}_y(), p.\text{get}_z())\);
  }

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

---

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

---

Log

[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.

---

Log

[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.

---

Log

[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) +≡

Transform Transform::align_with_axis(Point p0, Point p1, char axis)
/* 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! Transform::align_with_axis() Invalid\"axis\" argument: \"\n    ≡ 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) +≡
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, and z coordinates
are all positive. See below for the explanation. */
if (DEBUG) {
  t.show("t, outside, of, normalization");
}
Point proj_on_\_plane (p1); /* [LDF 2002.10.23.] Get the projection of p1 on the x-z plane. */
proj_on_\_plane.shift (0, -p1.world_coordinates[1]);
if (DEBUG) proj_on_\_plane.show("proj_on_\_plane");
Define Transform functions 169 \{ 
if (axis \equiv 'x' \lor axis \equiv 'y') { 
  Point pt_on_x_axis; 
  pt_on_x_axis.set(1); 
  angle = proj_on_x_z_plane.angle(pt_on_x_axis); 
  if (DEBUG) cout \ll \"\text{angle}_{x\text{of projection:}u} \ll angle \ll endl \ll flush; 
  if (angle \neq 0 \land angle \neq \text{INVALID\_REAL}) t \ast= pt1.rotate(0,-angle); 
  if (DEBUG) pt1.show("pl\text{\_after\_rotation\_to\_x-y\_plane}"); 
  angle = pt1.angle(pt_on_x_axis); 
  if (DEBUG) cout \ll \"\text{angle}_{x\text{-axis:}u} \ll angle \ll endl \ll flush; 
  if (angle \neq 0 \land angle \neq \text{INVALID\_REAL}) t \ast= pt1.rotate(0,0,-angle); 
  if (DEBUG) pt1.show("pl\text{\_after\_rotation\_to\_x-axis}"); 
} 
else if (axis \equiv 'z') { 
  Point pt_on_z_axis; 
  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. */ 
  if (proj_on_x_z_plane.get_z() < 0) { 
    cerr \ll \"\text{ERROR! in: Transform::align\_with\_axis():\n" \ll 
    \"proj_on_x_z_plane.get_z()\ll<0\ll\n" \ll \"\text{Normalize point should ensure that, this value, is }>0\ll\n" \ll \"Handling the error, but find out why it happened!\ll\n" \ll endl \ll endl \ll flush; 
    pt_on_z_axis.set(0,0,-1); 
  } 
}

[LDF 2002.10.23.] If we're aligning with the x or y-axis, rotate pt1 onto the x-y plane and then to the x-axis.

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

Define Transform functions 169 \{ 
if (proj_on_x_z_plane.world_coordinates[0] \equiv 0 \land proj_on_x_z_plane.world_coordinates[1] \equiv 0) angle = 0; 
else angle = proj_on_x_z_plane.angle(pt_on_z_axis); 

[LDF 2002.10.23.] If we're aligning with the z-axis, rotate pt1 onto the y-z plane and then to the z-axis.

(Define Transform functions 169) \{ 
if (DEBUG) cout \ll \"\text{angle}_{z\text{of projection:}u} \ll angle \ll endl \ll flush; 
if (angle \neq 0 \land angle \neq \text{INVALID\_REAL}) t \ast= pt1.rotate(0,angle); 
if (DEBUG) pt1.show("pl\text{\_after\_rotation\_to\_z-y\_plane}"); 
pt1.apply_transform();

[Define Transform functions 169] \{ 

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 \textit{proj}_\textit{on}_\textit{z\textunderscore axis} to \textit{pl} in the “if” part of the following conditional. The \textit{y\textunderscore coordinate} of \textit{proj\textunderscore on\textunderscore z\textunderscore axis} is always 0, so \textit{angle} was always set to 0. I discovered this bug when I tried rotating a \textit{Point} in the plane of a \textit{Reg\textunderscore Polygon} about a line from the center of the \textit{Reg\textunderscore Polygon} in the direction of its normal, and the resulting \textit{Point} was not in the same plane.

\begin{verbatim}
(Define \textbf{Transform} functions 169) +\equiv
  if (pl.worldcoordinates[1] \equiv 0) angle = 0;
  else angle = pl.angle(pl.on_z_axis);
\end{verbatim}

431.

(Define \textbf{Transform} functions 169) +\equiv
\begin{verbatim}
  if (DEBUG) cout \textless{} "angle, to, z\textunderscore axis::" \textless{} angle \textless{} endl \textless{} flush;
  if (angle \neq 0 \land angle \neq INVALID\_REAL) t *= pl.rotate(-angle);
  if (DEBUG) pl.show("pl,after,rotation, to, z\textunderscore axis");
\end{verbatim}

432. [LDF 2002.10.23.] If we’re aligning with the y-axis, \textit{pl} 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.

(Define \textbf{Transform} functions 169) +\equiv
\begin{verbatim}
  if (axis \equiv 'y') {
    t *= pl.rotate(0,0,90);
    if (DEBUG) pl.show("pl,after,rotation, to, y\textunderscore axis");
  }
  if (DEBUG) {
    cout \textless{} "pl.magnitude(),\textunderscore t\textsuperscript{--}t\textsuperscript{--}" \textless{} pl.magnitude() \textless{} endl \textless{} flush;
    t.show("t,at,end,of,align\textunderscore with\textunderscore axis");
  }
  *this *= t;
  if (DEBUG) cout \textless{} "Exiting,Transform::align\textunderscore with\textunderscore axis." \textless{} endl \textless{} flush;
  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 p1’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 \(x_{p1}, y_{p1}, \) and \(z_{p1}\) 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.

\[
\text{ Normalize point 433 } \equiv
\]

```cpp
if (DEBUG) p1.show("p1\_before\_normalization");
if (p1.world_coordinates[0] < 0 \&\& p1.world_coordinates[1] >= 0 \&\& p1.world_coordinates[2] >= 0)
  /* x negative, y and z positive. */
  t *= p1.rotate(0, -90);
else if (p1.world_coordinates[0] >= 0 \&\& p1.world_coordinates[1] < 0 \&\& p1.world_coordinates[2] >= 0)
  /* x positive, y negative, z positive. */
  t *= p1.rotate(90);
else if (p1.world_coordinates[0] >= 0 \&\& p1.world_coordinates[1] >= 0 \&\& p1.world_coordinates[2] < 0)
  /* x positive, y positive, z negative. */
  t *= p1.rotate(-90);
else if (p1.world_coordinates[0] < 0 \&\& p1.world_coordinates[1] < 0 \&\& p1.world_coordinates[2] >= 0)
  /* x negative, y negative, z positive. */
  t *= p1.rotate(0, 0, 180);
else if (p1.world_coordinates[0] < 0 \&\& p1.world_coordinates[1] >= 0 \&\& p1.world_coordinates[2] < 0)
  /* x negative, y positive, z negative. */
  t *= p1.rotate(0, 180);
else if (p1.world_coordinates[0] >= 0 \&\& p1.world_coordinates[1] < 0 \&\& p1.world_coordinates[2] < 0)
  /* x positive, y negative, z negative. */
  t *= p1.rotate(180);
else if (p1.world_coordinates[0] < 0 \&\& p1.world_coordinates[1] < 0 \&\& p1.world_coordinates[2] < 0)
  /* All negative. */
  {
    real a = p1.world_coordinates[0];
    t *= p1.rotate(180, 180);
    t *= p1.shift(a);
    t *= p1.rotate(0, 180);
    t *= p1.shift(-a);
  }
}
```

This code is used in section 425.

434. Rotation around an arbitrary axis.

435. Point versions. [LDF 2002.4.7] Added default value for angle \(\equiv 180\).
436. **Point arguments.** This function first checks to see if *this* lies on the axis. It does this by creating unit vectors in the directions of \( p1 - p0 \) and \( *this - p0 \). 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 **Transform**: \( \.rotate() \) (defined below).

[LDF 2002.4.7.] Added default value for `angle \equiv 180`.

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

(Declare Point functions 329) +≡

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

437.

(Define Point functions 330) +≡

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

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

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

[LDF 2002.04.07.] Added default value for `angle \equiv 180`.

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

(Declare Point functions 329) +≡

**Transform** \( \text{rotate}(\text{const Path } &p, \text{const real } angle = 180) \);
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 $\overrightarrow{PQ}$ 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 2003.06.02.] 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() ≡ 1 V a.get_x() ≡ -1) axis = 'x';
    else if (a.get_y() ≡ 1 V a.get_y() ≡ -1) axis = 'y';
    else axis = 'z';
    Transform t;
    t.align_with_axis(p0, p1, axis);
    Transform i = t.inverse();
    if (axis ≡ 'x') t.rotate(angle);
    else if (axis ≡ 'y') t.rotate(0, angle);
    else t.rotate(0, 0, angle);
    t *= i;
    t.clean();
    *this *= t;
    clean();
    return t;
}

[Log]
[LDF 2003.06.02.] Changed name of this function from rotateAround() 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)
{
    Transform t;
    t.rotate(p0, p1, angle);
    transform *= t;
    return t;
}

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.

[Log]
[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(). 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 value 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.

[Log]
[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) +

```cpp
bool Point::project(const Focus &f, const unsigned short proj, real factor){ bool DEBUG = false;
  /* true */
  if (DEBUG) cout << "Entering,.project().\n" << flush;
  if (compose.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
#endif // ifdef __GNUC__
  valarray (real) temp_coordinates = projective_coordinates = null_coordinates;
#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.

[LD 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.

---

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\equiv0,\n    Multiplying coordinates:\b\n    y,0,\n    doesn’t make sense."
    \ll "Using 1 instead.\n    \m\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:\" \ll projective_coordinates[0] \ll ",\n    \" \ll projective_coordinates[1] \ll ",\n    \" \ll projective_coordinates[2] \ll ",\n    \" \ll 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() == 0) {
  cerr << "ERROR! In Point::project():n" << temp_coordinates[2] << ", " << f.distance() << f.get_distance() << endl << "Sum == 0. Can't perform division." << endl << "Setting,projective_coordinates,to,INVALID_REAL" << "and.returning." << flush;
  for (i = 0; i < 4; i++) projective_coordinates[i] = INVALID_REAL;
  return false;
}

if (DEBUG) {
  cout << "f.get_distance() = " << f.get_distance() << endl << flush;
  cout << "save_z = save_z = " << save_z << endl << flush;
}

for (i = 0; i < 4; i++) {
  for (j = 0; j < 4; j++)
  projective_coordinates[i] += temp_coordinates[j] * f.persp_element(j, i);
}

real eps = epsilon();
if (projective_coordinates[3] == 0) {
  cerr << "ERROR! In Point::project():n" << projective_coordinates[3] << ", " << 0.0 << ".n" << "This,will,cause,a,floatin_error." << "n" << "Setting,projective_coordinates,to,INVALID_REAL" << "and.returning." << flush;
  for (i = 0; i < 4; i++) projective_coordinates[i] = INVALID_REAL;
  return false;
}

for (i = 0; i < 4; i++) {
  projective_coordinates[i] /= projective_coordinates[3];
  if (fabs(projective_coordinates[i]) < eps) projective_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. */

projective_coordinates[2] = (fabs(save_z) > eps) ? save_z : 0;
if (DEBUG) {
  cout << "Perspective,coordinates:\n";
  for (i = 0; i < 4; i++)
  cout << projective_coordinates[i];
  if (i < 3) cout << ", ";
}
  cout << ").n" << 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 Trunc_Octahedron. eps was too small.
*/
    if (fabs(projective_coordinates[i]) <= 10.0 * 10^-8) 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_focus to project ( const Focus & f . . . ). This is necessary because it's impossible make the argument
f optional with default_focus 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_focus inside of project(), as long as default_focus 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_focus, 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 apply_transform();
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_0=" << world_coordinates[0] << endl << flush;
       cout << "y_0=" << world_coordinates[1] << endl << flush;
       cout << "z_0=" << world_coordinates[2] << endl << flush;
       cout << "w_0=" << 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 << "]=" new_coordinates[i] << endl << flush;
           }
         }
       }
     }
     real eps = epsilon();
     for (i = 0; i < 4; i++) {
       if (DEBUG) {
         cout << "new_coordinates[" << i << "]=" 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.  
(Define Point functions 330) +
   void Point::reset_transform()
   
   transform.reset();

452.  Drawing.
453.  Drawdot.  
   [LDF 2002.10.26.] drawdot() copies *this and puts the copy onto the vector shapes of the Picture argument picture. The data members drawdot_value, drawdot_color, and pen are only set on the copy, not on *this. All of the drawing and filling functions behave similarly.  
   [LDF 2003.05.30.] TO DO: Add code for allocating new Color, if ddrawdot_color.use_name is false, as in the drawing and filling functions for Path and Solid.

454.  Normal version.  

---

[LDF 2003.07.11.] Made ppen and drawdot() itself const.

(Declare Point functions 329) +
   void drawdot(const Color &ddrawdot_color = *Colors::default_color, const string ppen = "", Picture &picture = current_picture) const;

455.  
(Define Point functions 330) +
   void Point::drawdot(const Color &ddrawdot_color, const string ppen, Picture &picture) const{
      bool DEBUG = false;  /* true */
      if (DEBUG) cout << "Entering drawdot()" << endl;  
      Point *pt = create_new < Point > (0);
      *pt = *this;
      pt->drawdot_value = DRAWDOT;
      pt->drawdot_color = &ddrawdot_color;
      
      if 1
         pt->pen = ppen;
      
      picture += static_cast <Shape*>(pt);
      if (DEBUG) cout << "Exiting drawdot()" << endl;
   }
456. Picture argument first.

(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. [LDF 2002.10.26.] undraw() does not remove a dot from picture, but causes the METAPOST command undrawdot to be written to out_stream 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.

(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.


        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.

        Log

    [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.


        Log

    [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;

[LDF 2003.06.03.] Made drawarrow() const.

Declare Point functions 329) +
Path drawarrow(Picture &picture, const Point &p,
    const Color &drawable_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.

[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.

[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.] draw help() is like draw(), except that the Path is only drawn if
the static Path data member do help lines ≡ 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.

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

Declare Point functions 329) +
Path draw help(const Point &pt, const Color &drawable_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 dpen = "") 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 in a couple of places, in order to get the hyphenation to work.

[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 2003.04.30.] Changed, so that a newline is not following text.

(Declare Point functions 329) +≡
  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;

476.
(Define Point functions 330) +≡
  void Point#: show(string text, char coords, const bool do_persp, const bool do_apply, Focus *f, const unsigned short proj, const real factor) const
{
  bool DEBUG = false;       /* true */
  if (text ≡ "") text = "Point:";
  cout ≪ text ≪ "_";
  coords = tolower(coords);
  if (coords ≡ ’w’) { /* Do nothing. */
    else if (coords ≡ ’p’) cout ≪ "Projective_coordinates.
" ≪ flush;
    else if (coords ≡ ’u’) cout ≪ "User_coordinates.
" ≪ flush;
    else if (coords ≡ ’v’) cout ≪ "View_coordinates.
" ≪ flush;
    else {
      cerr ≪ "WARNING!_In_point(): _Invalid_character_for_coordinates._argument._n" ≪ "_S
" ≪ flush;
      coords = ’w’;
    }
  }
  if (*this ≡ INVALID_POINT) {
    cerr ≪ "Point: _Invalid_POINT._nCan’t show._nReturning._n" ≪ flush;
    return;
  }
  if (DEBUG) transform.show("Transform_before_apply_transform");
  valarray<real> v = get_all_coords(coords, do_persp, do_apply, f, proj, factor);
  cout ≪ "(\(x ≪ v[0] ≪ \", _w " ≪ v[1] ≪ \", w[2] ≪ ") _n" ≪ flush;
  if (DEBUG) {
    transform.show("Transform_after_apply_transform");
    cout ≪ "on_free_store._n->" ≪ on_free_store ≪ "_n;\n"
  }
}

477. Show transform.
(Declare Point functions 329) +≡
  void show_transform(string text = "");
§478.  
(Define Point functions 330) +≡
   void Point::show_transform(string text)
   {
      if (text == "") text = "transform:";
      cout << text << endl;
      transform.show();
   }

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 output stream. All code using this function must ensure that apply_transform() and project() are called first.

------------------------------------------- Log
[LDF 2002.09.16.] Removed calls to apply_transform() and project().

(Declare non-member non-template functions for Point 480) ≡
   ostream & operator<< (ostream & o, Point &p);
See also section 534.
This code is used in section 634.

481.  
(Define non-member non-template functions for Point 481) ≡
   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;
   }
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) +≡
   virtual void suppress_output();

483.  
(Define Point functions 330) +≡
   void Point::suppress_output()
   {
      do_output = false;
   }

484.  Uns 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) +≡
   virtual void unsuppress_output();
485. Define Point functions 330) +≡
   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.

   Log

   [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) +≡
   vector(Shape *) extract(const Focus &f, const unsigned short proj, real factor);
487. Define Point functions 330) \[ 3 \times \]

```cpp
vector(Shape *) Point::extract(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! In Path::extract():\n"    << "Point on Path cannot be projected.\n"    << "Returning empty vector of Shape*.\n"    << flush;

        }
        if (DEBUG) {

        }
        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.

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

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

```cpp
virtual real get_minimum_z() const;
```
(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_" << 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_" << endl << flush;
    if (DEBUG)  cout << "Exiting.Point::get_maximum_z()" << endl << flush;
    return projective_extremes[5];
}

493. Get mean z. [LDF 2003.05.16.] 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 set_extremes().

[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 \(330\)) \(\equiv\)

\[
\text{bool Point::set_extremes()}
\]

\[
\text{bool DEBUG = false; /* true */}
\]

\[
\text{if (DEBUG) cout \ll "EnteringPoint::set_extremes()" \ll endl \ll flush;}
\]

\[
\text{for (int } i = 0; i < 4; i++) 
\]

\[
\text{if (projective_coordinates[i] \equiv INVALID_REAL) }
\]

\[
\text{cerr \ll "ERROR! InPoint::set_extremes()\n" \ll "projective_coordinates[" \ll i \ll "]" \ll 
"\,i\_invalid_REAL\n" \ll "Setting every element in projective_extremes,\" \ll 
"to INVALID_REAL and returning false.\n" \ll flush;}
\]

\[
\text{for (int } j = 0; j < 6; j++) \text{ projective_extremes[j] = INVALID_REAL;}
\]

\[
\text{return false;}
\]

\[
\]
498. Compare minimum z.

Log

[LD 2003.06.16.] Added this class.

(Define comparison classes 498) \equiv

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

See also sections 499 and 500.

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

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

[LDG 2002.09.17.] Added this class.

(Define comparison classes) +≡

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. [LDG 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
out_stream.

[LDG 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.

[LDG 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().

[LDG 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 out_stream with the
proper METAPOST instructions.

(Declare Point functions) +≡

void output();
502.
(Define Point functions 330) +

```c
void Point::output()
{
    bool DEBUG = false;  /* true */
    if (DEBUG) cout << "Entering Point::output()\n" << flush;
    if (do_output == false) {
        if (DEBUG) cout << "In Point::output(): do_output==false: Returning.\n" << flush;
        return;
    }
    if (drawdot_value == DRAWDOT) out_stream << "drawdot_u";
    else if (drawdot_value == UNDRAWDOT) out_stream << "drawdot_u";
    else     /* DRAWDOT */
    {
        cerr << "WARNING! Invalid drawdot_value:u" << drawdot_value << ". Using " < drawdot_value < "\n" << flush;
        #if 0     /* !! Define a class for information on the run state. */
        if (-Run_State::non_stop) getchar();
        #endif
        out_stream << "drawdot_u";
    }
    out_stream << "(" < projective_coordinates[0] < "x" < measurement_units << ",u" <
               projective_coordinates[1] < measurement_units << ")";
    if (drawdot_color != Colors::default_color) out_stream << "withcolor_u" << drawdot_color;
    #if 1
    if (pen != "") out_stream << ";withpen_u" < pen;
    #endif
    out_stream << ";\n";
    if (DEBUG) cout << 
"Exiting Point::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", "lt", "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 `dottedlabel` is written to `out_stream` rather than `label`. This argument is mainly for use by the function `dottedlabel()`, 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 `dottedlabel()`!!

---

**Log**

[LDF 2002.05.14.] 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 “\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 2003.07.09.] Made `text_str`, `position_str`, and `dot` arguments const.

(Declare **Point** functions 329) +

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

---

**STRING ARGUMENT**

506.
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 2003.06.06.] 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 2003.06.22.] BUG FIX: The “WORLD_VALUES_Z” case started with if instead of else if. This caused s.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) {
        if (DEBUG) cout << "It's a little WORLD_VALUES.\n";
        s << "\"x\" \gets get_x();\n        s << "\"y\" \gets get_y();\n    } else if (text_short == WORLD_VALUES_Z) {
        if (DEBUG) cout << "It's a little WORLD_VALUES.\n";
        s << get_z();
    } else if (text_short == PROJ_VALUES) {
        if (DEBUG) cout << "It's a little PROJ_VALUES.\n";
        cerr << "WARNING! In Point::label():\" endl << "text_short==e\"PROJ_VALUES.\n" <<
            "Haven't programmed this case yet.\n" << "Returning.\n\n" << flush;
        return;
    } else if (text_short == USER_VALUES) {
        if (DEBUG) cout << "It's a little USER_VALUE.\n";
        cerr << "WARNING! In Point::label():\" endl << "text_short==e\"USER_VALUES.\n" <<
            "Haven't programmed this case yet.\n" << "Returning.\n\n" << flush;
        return;
    } else if (text_short == VIEW_VALUES) {
        if (DEBUG) cout << "It's a little VIEW_VALUE.\n";
        cerr << "WARNING! In Point::label():\" endl << "text_short==e\"VIEW_VALUES.\n" <<
            "Haven't programmed this case yet.\n" << "Returning.\n\n" << flush;
        return;
    } else {
        if (DEBUG) cout << "It's a little something.\n";
        s << text_short;
        if (DEBUG) cout << "s.\"str()\".\n" << 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.

\[
\begin{array}{l}
\text{Log} \\
\end{array}
\]

\[\text{[LDF 2003.07.09.] Made } \text{text_str} \text{ and } \text{position_str} \text{ arguments const.}\]

(Declare Point functions 329) +≡
void \textit{dotlabel} (const string \textit{text_str}, const string \textit{position_str} = "top", Picture &\textit{picture} = \textit{current_picture}) const;

511.

(Define Point functions 330) +≡
void \textit{Point}::\textit{dotlabel} (const string \textit{text_str}, const string \textit{position_str}, Picture &\textit{picture}) const
\{
label (\textit{text_str}, \textit{position_str}, true, \textit{picture});
\}

512. short argument.

\[
\begin{array}{l}
\text{Log} \\
\end{array}
\]

\[\text{[LDF 2003.07.09.] Made } \text{text_short} \text{ and } \text{position_str} \text{ arguments const.}\]

(Declare Point functions 329) +≡
void \textit{dotlabel} (const short \textit{text_short}, const string \textit{position_str} = "top", Picture &\textit{picture} = \textit{current_picture}) const;

513.

(Define Point functions 330) +≡
void \textit{Point}::\textit{dotlabel} (const short \textit{text_short}, const string \textit{position_str}, Picture &\textit{picture}) const
\{
label (\textit{text_short}, \textit{position_str}, true, \textit{picture});
\}
514. [LDF 2002.09.06.] Commented out `~Label()`. This was the cause of a bug that caused a memory fault when I tried to use a label in figure 2 (`beginfig(2)`) after having used it in figure 1 and then invoking `currentpicture.clear()` in between.

```c
(Define Label functions 514) +≡
#endif
Label::~Label()
{
    if (pt != 0) delete pt;
}
#endif
```

See also sections 515 and 516.

This code is used in section 633.

515. Get copy of Label.

```c
(Define Label functions 514) +≡
Label * Label::get_copy() const { Label *bl = new (Label); bl->pt = create_new < Point > (0);
    *(bl->pt) = *pt;
    bl->dot = dot;
    bl->text = text;
    bl->position = position;
    return bl; }
```

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

```c
void Label::output(const Focus &f, const unsigned short proj, real factor, const Transform &t)
{
    if (!is_identity()) *pt += t;
    pt->apply_transform();
    if (!pt->project(f,proj,factor)) {
        cerr << "WARNING! In Label::output():\n" << "Point in Label cannot be projected!\n" << "Not printing Label\n" << flush;
    return;
    }
    if (dot == true) out_stream << "dot";
    out_stream << "label";
    if (position != "") out_stream << "," << position;
    out_stream << "(btext," << text << ",etex,.(" << pt->get_x(’p’,
        false) << Point::measurement_units << ","," << pt->get_y(’p’,
        false) << Point::measurement_units << ")\n";
    return;
}
```

517. Matrix operations.

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

\[
\text{Declare Point functions 329} \implies
\quad \text{Transform operator}**(\text{const Transform } \& t);
\]

519.

\[
\text{Define Point functions 330} \implies
\quad \text{Transform Point} \mathbf{:: \text{operator}+}(\text{const Transform } \& t)
\{
\quad \text{return (transform } \ *= \ t); \}
\]

520. Vector operations. [LDF 2002.10.27.] Note that the vector operations don’t affect the \( w \) coordinate.

\[
\text{[LDF 2002.10.27.]} \text{ In the functions } \text{operator}+() , \text{operator}\ -=() , \text{operator}--() , \text{and operator}--() : \text{It doesn’t seem worth it to write non-const versions, although I could. Now using the elements of } p_0.\text{world_coordinates} \text{ directly instead of using get}_x() , \text{get}_y() , \text{and get}_z(). \text{This is safe, as is calling } \text{apply_transform()} \text{ on } p , \text{and saves the cost of three function calls.}
\]

521. Vector addition.

\[
\text{[Declare Point functions 329]} \implies
\quad \text{Point operator}+(\text{Point } p) \ \text{const} ;
\]

522.

\[
\text{[Define Point functions 330]} \implies
\quad \text{Point Point} \mathbf{:: \text{operator}+}(\text{Point } p) \ \text{const}
\{
\quad \text{Point } a ;
\quad a = \text{this} ;
\quad p.\text{apply_transform}() ;
\quad a.\text{shift} (p.\text{world_coordinates}[0], p.\text{world_coordinates}[1], p.\text{world_coordinates}[2]) ;
\quad \text{return } a ;
\}
\]

523. Vector addition with assignment.

\[
\text{[Declare Point functions 329]} \implies
\quad \text{void operator}+=(\text{Point } p);
\]
(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.
[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.

```
(Define Point functions 330) +≡
real Point :: operator*(const real r)
{
    apply_transform();
    for (int i = 0; i < 3; i++) world_coordinates[i] *= r;
    return r;
}
```

531. Vector-scalar multiplication.

532. Member version (Point first).

```
[Log] [LDF 2002 10.27.] Made this function and the argument r const.

(Declare Point functions 329) +≡
Point operator*(const real r) const;
```

533.

```
(Define Point functions 330) +≡
Point Point :: operator*(const real r) const
{
    Point a(*this);
    a.apply_transform();
    a *= r;
    return a;
}
```

534. Non-member version (scalar first).

```
(Declare non-member non-template functions for Point 480) +≡
Point operator*(const real, const Point &p);
```

535.

```
(Define non-member non-template functions for Point 481) +≡
Point operator*(const real r, const Point &p)
{
    return p * r;
}
```

536. Unary minus.

```
[Log] [LDF 2002 10.27.] Made this function const.

(Declare Point functions 329) +≡
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 2002.10.27.] Made the argument r const.

(Declare Point functions 329) +≡ 
    void operator/(const real r);

539.  
{Define Point functions 330} +≡ 
    void Point::operator/=(const real r)
    {
        apply_transform();
        for (int i = 0; i < 3; i++) world_coordinates[i] /= r;
    }

540.  Vector-scalar division.

[Log 2002.10.27.] Made this function and the argument r const.

(Declare Point functions 329) +≡ 
    Point operator/(const real r) const;
541. \(\text{Define Point functions 330 } \equiv \text{ Point Point::operator/(const real r) const}\)

\{ 
  Point a(*this);
  a.apply_transform();
  a /= r;
  return a;
\}

542. Dot product.

\[\text{Log}\]

[LDF 2002.10.27.] Changed this function and argument \(p\) to \texttt{const}. Now using \texttt{world_coordinates} directly instead of \texttt{getx()}, \texttt{gety()}, and \texttt{getz()}.

[LDF 2003.07.11.] Changed, so that if the dot product is less than \texttt{Point::epsilon()}, 0 will be returned.

\(\text{Declare Point functions 329 } \equiv \text{ real dot\_product(Point p) const;}\)

543. \(\text{Define Point functions 330 } \equiv \text{ real Point::dot\_product(Point p) const}\)

\{ 
  Point a(*this);
  a.apply_transform();
  p.apply_transform();
  if (fabs(r) < \text{Point::epsilon}()) \(r = 0;\)
  return r;
\}

544. Cross product.

\[\text{Log}\]

[LDF 2002.10.27.] Changed this function and argument \(p\) to \texttt{const}. Now using \texttt{world_coordinates} directly instead of \texttt{getx()}, \texttt{gety()}, and \texttt{getz()}.

\(\text{Declare Point functions 329 } \equiv \text{ Point cross\_product(Point p) const;}\)
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

[LDF 2002.10.27.] 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);  
  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: in Point::magnitude() \n" << 
      "Point has a coordinate too large for squaring! \n" << "Returning INVALID_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 magnitude() \n";  
    cerr << "Point has too great a magnitude! \n";  /* !! This show() outputs to stdout. It  
      would be nice to output it to stderr instead. Must write function for this. */  
    cerr << "Returning INVALID_REAL.\n";  
    return INVALID_REAL;  
  }  
  r += temp;  
  return sqrt(r);  
}  

548.  Angle between two vectors.  

Log

[LDF 2002.10.27.] Made this function const.
[LDF 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"  << "Returning INVALID_REAL."
                return INVALID_REAL;
    }
    else if (p_mag == INVALID_REAL)
    {
        cerr << "WARNING! In angle().p.magnitude().failed."  << "\nReturning INVALID_REAL."
                return INVALID_REAL;
    }
    else if (p_mag == 0)
    {
        if (DEBUG) cerr << "WARNING! In angle().\n"  << "p.hasmagnitude.0.\n"  << "Returning INVALID_REAL."
                return INVALID_REAL;
    }
    else return (180/PI * acos(dot_product(p)/(mag * p_mag)));

550. Unit vector.

[LD 2002.10.27.] Added a second version. If assign is not used, unitvector() 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 an compilation error. [LD 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.

[LD 2002.10.27.] If magnitude() fails, unitvector() now returns INVALID_POINT instead of origin.
[LD 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 unitvector (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():\n          "Don’t call this function with false as its argument.\n" <<
      "Use unit_vector() without an argument, instead.\n"
      "Calling unit_vector() without an argument.\n"
      << 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():\n          "Point::vector() has no magnitude. Returning INVALID_POINT.\n"
        << 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.

INGER

[Log]

[LDF 2003.12.09.] Changed from a non-member to a const member function.
556. \( \text{Define Point functions 330} \) +
\[\begin{align*}
\text{Point} & \quad \text{Point::mediate(Point } p, \text{const real } r) \text{ const} \\
& \{ \\
& \quad \text{Point } t(*this); \\
& \quad t *= (1 - r); \\
& \quad p *= r; \\
& \quad \text{return } (t + p); \\
& \}
\end{align*}\]

557. \textbf{Get normal.} \( \text{get\_normal()} \) must be defined in \texttt{paths\_web}, because it uses a \texttt{Path} in its definition, which is an incompletely defined type in this file.

\[\begin{align*}
[\text{LDF 2003.07.11.}] \quad & \text{Added this declaration.} \\
\end{align*}\]

\[\begin{align*}
\text{(Declare Point functions 329) } \equiv \\
\text{Point get\_normal(const Point } & p, \text{const Point } & q) \text{ const};
\end{align*}\]

558. \textbf{Comparison.}

559. \textbf{Equality.} \( \text{!! I may have to adjust to value of } \text{eps}. \) It would be nice to be able to use \texttt{epsilon()}, but for other purposes \texttt{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.

560. \textbf{Non-const version.}

\[\begin{align*}
[\text{LDF 2002.10.27.}] \quad & \text{Revised this function. Now using } \texttt{Point } a \text{ and } \texttt{Point } q. \text{ Added } \texttt{factor} \text{ and using it as the argument to } \texttt{clean()} \text{ and for calculating } \texttt{eps}. \text{ Since this function is an operator, it's not possible to pass } \texttt{factor} \text{ as an argument, unfortunately. Using } \texttt{clean(factor)} \text{ makes it possible to compare the coordinates with 0 directly rather than using } \texttt{fabs()} \text{ and } \texttt{eps}. \text{ Also, } \texttt{operator\equiv()} \text{ now uses } \texttt{world\_coordinates} \text{ directly rather than } \texttt{get\_x()}, \texttt{get\_y()}, \text{ and } \texttt{get\_z}(). \\
[\text{LDF 2003.07.09.}] \quad & \text{Made this function } \texttt{non-const}, \text{ and added } \texttt{const} \text{ version below.}
\end{align*}\]

\[\begin{align*}
\text{(Declare Point functions 329) } \equiv \\
\text{bool operator\equiv(Point } & p);
\end{align*}\]
561. 
(Define Point functions 330) +≡
bool Point::operator==(Point p) { bool DEBUG = false; /* true */
    if (DEBUG) cout << "Entering Point::operator==()" << endl << 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 t_x = world_coordinates[0];
    real t_y = world_coordinates[1];
    real t_z = world_coordinates[2];
    real p_x = p.world_coordinates[0];
    real p_y = p.world_coordinates[1];
    real p_z = 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) +≡
if (t_x ≡ INVALID_REAL ∨ t_y ≡ INVALID_REAL ∨ t_z ≡ INVALID_REAL ∨ p_x ≡ INVALID_REAL ∨ p_y ≡ INVALID_REAL ∨ p_z ≡ INVALID_REAL) DEBUG = false;

563. Debugging output.
(Define Point functions 330) +≡
if (DEBUG) {
    cout << "t_x=a=∞" << t_x << endl << flush;
    cout << "t_y=a=∞" << t_y << endl << flush;
    cout << "t_z=a=∞" << t_z << endl << flush;
    cout << "p_x=a=∞" << p_x << endl << flush;
    cout << "p_y=a=∞" << p_y << endl << flush;
    cout << "p_z=a=∞" << p_z << endl << flush;
}

564. Check whether the coordinates of both Points are all 0.

[Log]
LDF 2002 10.27.] 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) +≡
if (t_x ≡ 0 ∧ p_x ≡ 0 ∧ t_y ≡ 0 ∧ p_y ≡ 0 ∧ t_z ≡ 0 ∧ p_z ≡ 0) {
    if (DEBUG) cout << "All coordinates are 0, returning true." << endl;
    return true;
}
Get the signs of the coordinates.

[LDF 2002 10.27.] As in the previous section, changed so that the coordinates are compared with 0, instead of using `fabs()` and `eps`.

```cpp
<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 (\( t_x \equiv 0 \)) \( t_x \) sign = 0;
else if (\( t_x < 0 \)) \( t_x \) sign = -1;
else \( t_x \) sign = 1;
if (\( t_y \equiv 0 \)) \( t_y \) sign = 0;
else if (\( t_y < 0 \)) \( t_y \) sign = -1;
else \( t_y \) sign = 1;
if (\( t_z \equiv 0 \)) \( t_z \) sign = 0;
else if (\( t_z < 0 \)) \( t_z \) sign = -1;
else \( t_z \) sign = 1;
if (\( p_x \equiv 0 \)) \( p_x \) sign = 0;
else if (\( p_x < 0 \)) \( p_x \) sign = -1;
else \( p_x \) sign = 1;
if (\( p_y \equiv 0 \)) \( p_y \) sign = 0;
else if (\( p_y < 0 \)) \( p_y \) sign = -1;
else \( p_y \) sign = 1;
if (\( p_z \equiv 0 \)) \( p_z \) sign = 0;
else if (\( p_z < 0 \)) \( p_z \) sign = -1;
else \( p_z \) sign = 1;
if (DEBUG) {
    cout "t_x sign\n" \( t_x \) sign endl << flush;
    cout "t_y sign\n" \( t_y \) sign endl << flush;
    cout "t_z sign\n" \( t_z \) sign endl << flush;
    cout "p_x sign\n" \( p_x \) sign endl << flush;
    cout "p_y sign\n" \( p_y \) sign endl << flush;
    cout "p_z sign\n" \( p_z \) sign endl << flush;
}
if ((\( t_x \) sign \( \neq \) \( p_x \) sign) \( \lor \) (\( t_y \) sign \( \neq \) \( p_y \) sign) \( \lor \) (\( t_z \) sign \( \neq \) \( p_z \) sign)) {
    if (DEBUG)
        cout "At least one coordinate pair has signs that differ.\n" << "Returning false.\n";
    return false;
}
566. Get the difference between each pair of \( x, y, \) and \( z \)-coordinates.
\[
\{ \text{Define } \textbf{Point} \text{ functions } 330 \} + \equiv \\
\begin{align*}
\text{t}_x &= \text{fabs}(\text{t}_x) ; \\
\text{t}_y &= \text{fabs}(\text{t}_y) ; \\
\text{t}_z &= \text{fabs}(\text{t}_z) ; \\
\text{p}_x &= \text{fabs}(\text{p}_x) ; \\
\text{p}_y &= \text{fabs}(\text{p}_y) ; \\
\text{p}_z &= \text{fabs}(\text{p}_z) ; \\
\text{real } \text{delta}_x &= \text{fabs}(\text{t}_x - \text{p}_x) ; \\
\text{real } \text{delta}_y &= \text{fabs}(\text{t}_y - \text{p}_y) ; \\
\text{real } \text{delta}_z &= \text{fabs}(\text{t}_z - \text{p}_z) ; \\
\text{if } (\text{DEBUG} ) \{ \\
\text{cout } \ll \text{"delta}_x = " \ll \text{delta}_x \ll \text{endl } \ll \text{flush} ; \\
\text{cout } \ll \text{"delta}_y = " \ll \text{delta}_y \ll \text{endl } \ll \text{flush} ; \\
\text{cout } \ll \text{"delta}_z = " \ll \text{delta}_z \ll \text{endl } \ll \text{flush} ; \\
\} \\
\text{bool } r; \quad /* \text{LDF } 2002.10.27. \text{ The return value. It's only needed for the sake of the debugging code. } \\
\text{ Otherwise, this function could just return the result of the following expression. */} \\
r &= (\text{delta}_x < \text{eps } \wedge \text{delta}_y < \text{eps } \wedge \text{delta}_z < \text{eps} ) ; \\
\text{if } (\text{DEBUG} ) \text{ cout } \ll \text{"I" } \ll r \ll \text{endl } \ll \text{flush} ; \\
\text{if } (\text{DEBUG}) \text{ cout } \ll \text{"Exiting \textbf{Point}::operator-" } \ll \text{"\n" } \ll \text{flush} ; \\
\text{return } r; \} \\
\]

567. \textbf{const} version. \text{[LDF } 2003.07.09.] \text{ This function merely copies } *\textbf{this} \text{ and calls the non-\textbf{const} version on the copy. Here, } p \text{ can be a } \textbf{const} \textbf{Point} \&, \text{ because this function does nothing but pass it to non-\textbf{const} version, where it is passed by value.}

\[ \text{Log} \]

\text{[LDF } 2003.07.09.] \text{ Added this version. Made the original version non-\textbf{const}.}

\[
\{ \text{Declare } \textbf{Point} \text{ functions } 329 \} + \equiv \\
\text{bool operator}(\textbf{const} \textbf{Point} &p) \textbf{ const} ;
\]

568.
\[
\{ \text{Define } \textbf{Point} \text{ functions } 330 \} + \equiv \\
\text{bool } \textbf{Point}::\text{operator}(\textbf{const} \textbf{Point} &p) \textbf{ const} \\
\{ \\
\text{Point } \text{copy}(*\textbf{this}) ; \\
\text{return } (\text{copy } \equiv p) ; \\
\}
\]

569. \textbf{Inequality.}
\[
\{ \text{Declare } \textbf{Point} \text{ functions } 329 \} + \equiv \\
\text{bool operator}(\textbf{const} \textbf{Point} &p) \textbf{ const} ;
\]
570. 
\langle Define \textbf{Point} \text{ functions 330} \rangle \equiv

\begin{verbatim}
enum Point;
operator\$(\text{const Point \&p}) \text{ const }
{
    return \$\text{this \equiv p};
}
\end{verbatim}

571. \textbf{Intersection.}
[LDF 2002.10.27.] \textit{intersection\_point()} takes four \textbf{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 \textbf{bool\_point}. If the intersection point exists and is on both line segments, the \textbf{bool} is \textit{true} and the \textbf{Point} is the intersection point. If the intersection point exists, but is not on both line segments, the \textbf{bool} is \textit{false} and the \textbf{Point} is the intersection point. If no intersection point exists, i.e., the line segments are congruent or parallel, then the \textbf{bool} is \textit{false} and the \textbf{Point} is \textit{INVALID\_POINT}.

LDF Date? Note that \textit{intersection\_point()} had to be defined as a \textbf{static} member function, because \textbf{Path::intersection\_point()} was not able to resolve the call of this version, when it was not callable as \textit{"Point::intersection\_point()"}. I got a compiler error, because the call of \textit{intersection\_point()} inside \textbf{Path::intersection\_point()} with four references to \textbf{Points} as arguments was deemed to have "too many arguments" and the references to \textbf{Point} couldn’t be converted to \textbf{const} references to \textbf{Path}.

572. \textbf{Vector version.} [LDF 2003.06.29.] Defined in \textit{lines\_web}, because \textbf{Line} is an incomplete type here.


?? It may be necessary or desirable to add \texttt{try...catch} blocks where calculations are performed below, just in case overflow occurs.

?? Under Linux, both this function and the version of \textit{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 \textbf{Polyhedra}. [LDF 2003.06.29.]

\begin{verbatim}
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 \textit{lines\_web}, because it requires the use of \textbf{Lines}, and \textbf{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.
\end{verbatim}

\langle Declare \textbf{Point} \text{ functions 329} \rangle \equiv

\begin{verbatim}
static bool\_point intersection\_point(const Point \&pp0, const Point \&pp1, const Point \&qq0, const Point \&qq1);
\end{verbatim}

573. \textbf{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 \textbf{bool} argument \textit{trace} serves only to distinguish this function from the vector version. It doesn’t matter whether it’s \textit{true} or \textit{false}. [LDF 2003.06.29.]

\begin{verbatim}
Log

[LDF 2002.10.27.] Changed the \textbf{const Point \&} arguments to \textbf{Point}, because I had to copy them anyway in order to call \textit{apply\_transform()} on them.
\end{verbatim}
§573  3DLDF-1.1.5.1  TRACE VERSION  153

[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()." << flush;
        p0 . show("p0");
        p1 . show("p1");
        q0 . show("q0");
        q1 . show("q1");
    }
    bool_point bp;   /* Return value. */
    if (DEBUG) cout << "Error, after here0. " << flush;
```
576. Get deltas.

\begin{verbatim}
(Define Point functions 330) +≡
  real delta_x_p = p1_x - p0_x;
  real delta_y_p = p1_y - p0_y;
  real delta_x_q = p1_x - p0_x;
  real delta_y_q = q1_y - q0_y;
  real delta_x_r = q1_x - q0_x;
  real delta_y_r = q1_y - q0_y;
  if (DEBUG) cout << "Error, after, here, 2.\n" << flush;
\end{verbatim}

577. Slopes for line \(\overline{PQ}\).

\begin{verbatim}
(Define Point functions 330) +≡
  real slope_p_r_x = (delta_x_p \neq 0) ? delta_y_p / delta_x_p : INVALID_REAL;
  real slope_p_r_y = (delta_y_p \neq 0) ? delta_y_p / delta_x_p : INVALID_REAL;
  real slope_p_r_z = (delta_z_p \neq 0) ? delta_y_p / delta_z_p : INVALID_REAL;
  if (DEBUG) cout << "Error, after, here, 3.\n" << flush;
\end{verbatim}

578. Slopes for line \(\overline{QR}\).

\begin{verbatim}
(Define Point functions 330) +≡
  real slope_q_r_x = (delta_r_q \neq 0) ? delta_y_q / delta_x_q : INVALID_REAL;
  real slope_q_r_y = (delta_y_q \neq 0) ? delta_y_q / delta_x_q : INVALID_REAL;
  real slope_q_r_z = (delta_z_q \neq 0) ? delta_y_q / delta_z_q : INVALID_REAL;
  if (DEBUG) cout << "Error, after, here, 4.\n" << flush;
\end{verbatim}

579. The traces on the x-y plane. \(x_1\), \(y_1\), \(z_1\), \(y_{int_p}\), 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_1 = INVALID_REAL; /* x-coordinate of the intersection point. */
  real y_1 = INVALID_REAL; /* y-coordinate of the intersection point. */
  real z_1 = INVALID_REAL; /* z-coordinate of the intersection point. */
  real y_{int_p} = INVALID_REAL; /* y-intercept of \(\overline{PQ}\). */
  real y_{int_q} = INVALID_REAL; /* y-intercept of \(\overline{QR}\). */
  if (DEBUG) cout << "Error, after, here, 5.\n" << flush;
  if (slope_p_r_y \neq INVALID_REAL) /* \(\Delta x_p \neq 0\). */
  {
    y_{int_p} = p0_y - (slope_p_r_y \times p0_x);

    if (slope_q_r_y \neq INVALID_REAL) /* \(\Delta x_q \neq 0\). */
    {
      y_{int_q} = q0_y - (slope_q_r_y \times q0_x);

      if (DEBUG) cout << "Error, after, here, 6.\n" << flush;
    }
  }
\end{verbatim}
580. If both of the traces of $p'$ and $q'$ in the x-y plane are parallel to the y-axis (i.e., $\Delta x = 0$), we test whether $p_z = q_z$. If they are, then we set $x \perp$ to that value. If they're not, the lines don't intersect, so we return INVALID_POINT.

- If the trace of $p'$ or the trace of $q'$ in the x-y plane is parallel to the y-axis, we set $x \perp$ 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 $x \perp$ using the slope and y-intercept of the lines.

```
{ Define Point functions 330 } +
if (y_int_p \equiv INVALID_REAL \land y_int_q \equiv INVALID_REAL)
    /* $p_{zz}$ and $q_{zz}$ are both parallel to the z-axis. */
    
    if (DEBUG) cout << "Error, after here, 7. \n" << flush;
    if (p0_x \equiv q0_x) /* 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_p = p0_x;
        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 = p0_y - slope_p-x-y * p0_z;
        if (slope_q-x-y \neq INVALID_REAL) y_int_q_x = q0_y - slope_q-x-y * q0_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, z-y, plane, are coincident. \n" << flush;
            if (DEBUG) cout << "Error, after here, 10. \n" << flush;
            if (p0_z \equiv q0_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_p = p0_z;
            y_p = y_int_q_x * slope_q-z-y + y_int_q_z;
            if (DEBUG) {
                cout << "x_{ i \_u} - u \" << x_p << endl << flush;
                cout << "y_{ i \_u} - u \" << y_p << endl << flush;
                cout << "z_{ i \_u} - u \" << z_p << endl << flush;
                cout << "slope_q-z-y \" << slope_q-z-y << endl << flush;
                cout << "y_int_q_z \" << y_int_q_z << 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_q = q0_z;
            y_q = y_int_p_x * slope_p-z-x + y_int_p_z;
            if (DEBUG) {
                cout << "x_{ i \_u} - u \" << x_q << endl << flush;
                cout << "y_{ i \_u} - u \" << y_q << endl << flush;
            }
        }
    }
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, 13. \n" << flush;
    z_p = p0_z;
    y_p = y_int_q_x * slope_q-z-y + y_int_q_z;
    if (DEBUG) {
        cout << "x_{ i \_u} - u \" << x_p << endl << flush;
        cout << "y_{ i \_u} - u \" << y_p << 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, 14. \n" << flush;
    z_q = q0_z;
    y_q = y_int_p_x * slope_p-z-x + y_int_p_z;
    if (DEBUG) {
        cout << "x_{ i \_u} - u \" << x_q << endl << flush;
        cout << "y_{ i \_u} - u \" << y_q << endl << flush;
    }
```
cout << "z_int_y" << endl << flush;
cout << "slope_p_z_y" << slope_p_z_y << endl << flush;
}  
}  
}  
}  
}  
else  
if (DEBUG) cout << "Neither,trace,is,vertical\n" << flush;
if (DEBUG) cout << "Error,after,here,13.\n" << flush;
z_i = (y_int_p_y - y_int_p_z) / (slope_p_x - slope_q_x);
y_i = slope_p_x * x_i + y_int_p_x;
if (DEBUG)  
}  
}  
}  
}  
else    /* They don’t have the same value for x. */  
{  
if (DEBUG) cout << "Error,after,here,14.\n" << flush;
cerr << "Lines do not intersect: \n";
cerr << "p0_x \ p0_y \ p1_x \ p1_y \ p0_x \ p0_y \ p1_x \ p1_y \ ";
cerr << "p0_y \ p1_y \ q0_x \ q0_y \ q1_x \ q1_y \ ";
cerr << "Returning,INVALID_BOOL_POINT.\n" << flush;
return INVALID_BOOL_POINT;
}  
}  
else if (y_int_p == INVALID_REAL)  /* p_y is parallel to the y-axis. */  
{  
if (DEBUG) cout << "Error,after,here,15.\n" << flush;
x_i = p0_x;
y_i = slope_q_x * x_i + y_int_q;
}  
else if (y_int_q == INVALID_REAL)  /* q_y is parallel to the y-axis. */  
{  
if (DEBUG) cout << "Error,after,here,16.\n" << flush;
x_i = q0_x;
y_i = slope_p_x * 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 \( \vec{q}_{xy} \) nor \( \vec{q}_{xy} \) is parallel to the y-axis. */
  {
    if (DEBUG)    /* [LDF 2002.11.12.] Start working on finding bug here. */
    {
      cout ≡ "Error, after, here, 17, \n" ≡ flush;
      cout ≡ "slope_p_x_y, --, u" ≡ slope_p_x_y ≡ endl ≡ flush;
      cout ≡ "slope_q_x_y, --, u" ≡ slope_q_x_y ≡ endl ≡ flush;
    }
    if (slope_p_x_y \neq slope_q_x_y) {
      x, i = (y, int, q - y, int, p) / (slope_p_x_y - slope_q_x_y);
      y, i = slope_p_x_y * x, i + y, int, p;
    }
  }

582. The trace on the x-z plane. We don’t need to do this if we’ve calculated \( z, i \) above, in the case that the traces on the x-y plane are coincident.

(Define Point functions 330) +≡
  if (\( z, i \equiv INVALID, REAL \)) {
    if (DEBUG) cout ≡ "Error, after, here, 18, \n" ≡ flush;
    real z, int, p = INVALID, REAL;    /* z-intercept of \( \frac{p_0, z}{\theta, x, z} \). */
    real z, int, q = INVALID, REAL;    /* z-intercept of \( \frac{q_0, z}{\theta, x, z} \). */
    if (slope_p_x_z \neq INVALID, REAL)    /* \( \Delta x, p \neq 0 \). */
    {
      z, int, p = p_0, z - (slope_p_x_z * p_0, x);
    }
    if (slope_q_x_z \neq INVALID, REAL)    /* \( \Delta x, q \neq 0 \). */
    {
      z, int, q = q_0, z - (slope_q_x_z * q_0, x);
    }
    if (DEBUG) {
      cout ≡ "z, int, p, --, u" ≡ z, int, p ≡ endl ≡ flush;
      cout ≡ "z, int, q, --, u" ≡ z, int, q ≡ endl ≡ flush;
    }
  }

583. [LDF 2003.06.24.] \( z, i \) 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 (\( (z, i \equiv INVALID, REAL \land \neg (z, int, p \equiv INVALID, REAL \lor z, int, q \equiv INVALID, REAL) \)) {
    x, i = (z, int, q - z, int, p) / (slope_p_x_z - slope_q_x_z);
    y, i = p_0, y;
  }
In the following case, \( \vec{p}_{xz} \) and \( \vec{q}_{xz} \) are both parallel to the z-axis. They have the same value for \( x \). We’ve set \( x_i \) above, so there’s no need to do so here again.

\[
\text{Define Point functions 330) + =}
\]

\[
\text{if (z_int_p \equiv INVALID\_REAL \& z_int_q \equiv INVALID\_REAL) }
\]

\[
\text{if (DEBUG) cout \lless \"Error\_after\_here\_19.\n\lless flush;}
\]

\[
\text{if (p0.z \neq q0.z) }
\]

\[
\text{if (DEBUG) cout \lless \"Error\_after\_here\_20.\n\lless flush;}
\]

\[
\text{cerr \lless \"Lines do not intersect:\n\lless \"(\lless p0.x \lless \",\lless w \lless p0.z \lless \",\lless u \lless p0.z \lless \",\lless w \lless p0.z \lless \",\lless u \lless p0.z \lless \",\lless w \lless p0.z \lless \",\lless u \lless p0.z \lless \",\lless w \lless p0.z \lless \",\lless u \lless p0.z \lless \",\lless w \lless p0.z \lless \"\}\n\lless \nReturning_INVALID\_BOOL\_POINT.\n\lless flush;
\]

\[
\text{return INVALID\_BOOL\_POINT;}
\]

\[
\text{else if (z_int_p \equiv INVALID\_REAL) }
\]

\[
\text{if (DEBUG) cout \lless \"Error\_after\_here\_21.\n\lless flush;}
\]

\[
\text{z_i = p0.z;}
\]

\[
\text{else if (z_int_q \equiv INVALID\_REAL) }
\]

\[
\text{if (DEBUG) cout \lless \"Error\_after\_here\_22.\n\lless flush;}
\]

\[
\text{z_i = q0.z;}
\]

\[
\text{else }
\]

\[
\text{if (DEBUG) cout \lless \"Error\_after\_here\_23.\n\lless flush;}
\]

\[
\text{z_i = slope_p.x \times x_i + z_int_p;}
\]

\]
585. [LDF 2002.10.27.] If \( x_i, y_i, \) and \( z_i \) are all valid, set \( bp.pt \) using those values. Otherwise, set it to INVALID_POINT. If this Point is on both of the line segments \( \overline{p_0p} \) and \( \overline{q_0q} \); set \( bp.b \) to true, otherwise set it to false.

(Define Point functions 330) +≡

```cpp
eval {if (DEBUG) {
    cout "Error\nafter\nhere.24.\n" \n    cout "x_i\n\016\n\016\n\016\n" << x_i << endl \n    cout "y_i\n\016\n\016\n\016\n" << y_i << endl \n    cout "z_i\n\016\n\016\n\016\n" << z_i << endl \n} 
if (x_i \equiv INVALID_REAL \lor y_i \equiv INVALID_REAL \lor z_i \equiv INVALID_REAL) {
    return INVALID_BOOL_POINT;
    cout "Returning\nINVALID_BOOL_POINT.\n" \n} else {
    bp.pt.set(x_i, y_i, z_i);
    if (DEBUG) bp.pt.show("bp.pt");
} 
if (bp.pt.is_on_segment(p0, p1),first \equiv false \lor bp.pt.is_on_segment(q0, q1),first \equiv false) bp.b = false;
else bp.b = true;
if (DEBUG) {
    cout "bp.b\n\016\n\016\n\016\n" << bp.b << endl;
    cout "Exiting\n\016\n\016\n\016\npoint::intersection_point().\n" << flush;
} 
return bp;
```

586. Picture functions. These functions must be defined here, because they use types which are incompletely defined in pictures.web.

587. Assignment operator. ⌥ PORTING [LDF 2002.12.05.] Moved here from pictures.web. See that file for explanation.

(Define Picture functions 264) +≡

```cpp
void Picture::operator=(const Picture &p) {
    clear();
    transform = p.transform;
    for (vector<Shape *>::const_iterator iter = p.shapes.begin(); iter \!\! p.shapes.end(); iter++) {
        shapes.push_back((*iter)->getCopy());
    }
   Label *lbl;
   for (vector<Label *>::const_iterator iter = p.labels.begin(); iter \!\! p.labels.end(); iter++) {
        lbl = (*iter)->getCopy();
        labels.push_back(lbl);
    }
}
```


(Define Picture functions 264) +≡

```cpp
Picture::Picture(const Picture &p) : do_labels(true) {
```
*this = pt
}


[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.

(Define Picture functions 264) +

void Picture::operator+=(const Picture &p)
{
    for (vector<Shape>::const_iterator iter = p.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 pt the Shapes don’t “know” about it. */
    *(shapes.back()) += p.transform;
}
for (vector<Label>::const_iterator iter = p.labels.begin(); iter != p.labels.end(); iter++) {
    labels.push_back(iter->get_copy());
    *(labels.back()->pt) += p.transform;
}
}

590. Clear Picture.

(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.

#include <Picture functions 264> +

void 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=-\n" << min_x_proj << endl << flush;
        cout << "max_x_proj=-\n" << max_x_proj << endl << flush;
        cout << "min_y_proj=-\n" << min_y_proj << endl << flush;
        cout << "max_y_proj=-\n" << max_y_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.

---

[LDF 2003.01.05.] Modified the \TeX text above to account for the fact that I’ve added Solid.

[LDF 2003.01.05.] Bug fix: Moved the code that causes transform to be applied to the elements of shapes. This is now done before the extremes are set. The way it was before didn’t work properly, because extract() used the untransformed values to decide whether a Shape was projectable. In order to do this, I had to add apply_transform() a Shape function.

(Define Picture functions 264) $\equiv$

```cpp
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 transform.\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() returned an empty vector.\n" << "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, `set_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.

(Define Picture functions 264) +≡

```cpp
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).

(Define Picture functions 264) +≡

```cpp
extremes = (**iter), get_extremes();
if (DEBUG) /* [LDF 2002.09.21.] Show the extremes for this Shape. */
{
    for (int i = 0; i < 4; i++) {
        cout << "extremes[" << i << "]", << extremes[i] << ",", << extremes[i] << "\n";
    }
    cout << "extremes[0],", << min_x_proj, << "\" << (extremes[0] < min_x_proj) << endl << flush;
    cout << "extremes[1],", << max_x_proj, << "\" << (extremes[1] > max_x_proj) << endl << flush;
    cout << "extremes[2],", << min_y_proj, << "\" << (extremes[2] < min_y_proj) << endl << flush;
    cout << "extremes[3],", << max_y_proj, << "\" << (extremes[3] > max_y_proj) << endl << flush;
}
    if (dowarnings ≡ true) {
        cerr << "WARNING! In Picture::output():\n" "Shape lies outside the limits for this invocation of output().\n" "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) +≡

if (elements.size(0) > 0) {
  if (sort_value == MIN_Z) sort(elements.begin(), elements.end(), Compare_minimum_Z());
  else if (sort_value == MAX_Z) sort(elements.begin(), elements.end(), Compare_maximum_Z());
  else if (sort_value == MEAN_Z) sort(elements.begin(), elements.end(), Compare_mean_Z());
  if (DEBUG) {
    if (sort_value == MIN_Z) cout ≡ "_****.MIN_Z.sort_.***\n"
    else if (sort_value == MAX_Z) cout ≡ "_****.MAX_Z.sort_.***\n"
    else if (sort_value == MEAN_Z) cout ≡ "_****.MEAN_Z.sort_.***\n"
    for (iter = elements.begin(); iter != elements.end(); ++iter) {
      cout ≡ "Min:.z:.u:" ≡ (**iter).get_minimum_z() ≡ endl ≡ "Max:.z:.u:" ≡ (**iter).get_maximum_z() ≡ endl ≡ "Mean:.z:.u:" ≡ (**iter).get_mean_z() ≡ endl ≡ endl;
    }
    cout ≡ "****.End.of.result.of.sort:\n" ≡ flush;
  }
  for (iter = elements.begin(); iter != elements.end(); ++iter) {
    (**iter).output();
    (**iter).unsuppress_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) +≡

if (do_labels == true) {
  for (vector(Label*) i = labels.begin(); i != labels.end(); ++i) {
    /* [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 & const Transform &). [LDF 2002.04.25.] This applies transform to the Point Label::pt. */
    (**i).output(f, proj, factor, transform);
  }
  /* [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::Picture::output(const Focus&...)\n" ≡ flush;
}
598. No Focus argument.

\begin{verbatim}
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 max_z_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, max_z_proj, min_z_proj, max_z_proj);
}
\end{verbatim}

599. Focus.

600. Focus class definition. [LDF 2002.09.18.] Made Focus a class (it was formerly a struct). Added char axis data member. It indicates the axis to which position should be transformed to when it's put in standard position. This determines which plane the image is projected onto. If axis \( \equiv 'z' \) (the default), the image is projected onto the x-y plane, if axis \( \equiv 'x' \), the y-z plane, if \( 'y' \), 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 not 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.

\begin{verbatim}
class Focus { int

    Point position;
    Point direction;
    Point up;
    real distance;
    real angle;
    char axis;
    Transform transform;
    Transform persp;

    public: { Declare Focus functions 602}
};
\end{verbatim}

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 \( \neq 0 \) 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 \( \equiv 'x' \) or axis \( \equiv 'z' \), or (1,0,0) if axis \( \equiv 'y' \). If angle \( \neq 0 \), 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, 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 up 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.

(Define Focus functions 605)≡

Focus::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\) function\) \(7 real\) arguments\n":
    axis = tolower(axis);
  if (axis \(!\equiv\ x\) \&\& axis \(!\equiv\ y\) \&\& axis \(!\equiv\ z\) ) {
    cerr << "WARNING!\) In Focus\) function\) \(7 real\) arguments\":
      "axis\) argument\) has invalid value:\" axis \(!\equiv\ " Using\) \"z\) \":
    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. */
  #endif
  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":
  }
  Transform unalign_up = transform.inverse(); /* Use the positive y-axis for the \"up\" direction,
    if axis \(!\equiv\ x\) \&\& axis \(!\equiv\ z\), and the positive x-axis if axis \(!\equiv\ y\). */
  if (axis \(!\equiv\ \"z\" \&\& axis \(!\equiv\ \"x\") ) up.set(0,1,0);
  else up.set(1,0,0);
  if (angle \(!\equiv 0\) ) {
    if (axis \(!\equiv\ \"z\") up \(\leftrightarrow\) transform.rotate(0,0,-angle);
    else if (axis \(!\equiv\ \"x\") up \(\leftrightarrow\) transform.rotate(-angle);
    else if (axis \(!\equiv\ \"y\") up \(\leftrightarrow\) transform.rotate(0,-angle);
    else {
      cerr << "ERROR!\) In Focus::Focus\) \n\)axis\) has invalid value:\" axis \(!\equiv\ " endl << "Rotating around z-axis\n":
        "Enter \<RETURN>\) to try to continue\n":
      up \(\leftrightarrow\) transform.rotate(0,0,-angle);
    }
  }
  if (DEBUG) up.show("up after rotation");
}
up \(\leftrightarrow\) 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\) function\) \(7 real\) arguments\n":
}

See also sections 607, 610, 612, 614, 616, 618, 626, and 629.

\{Declare Focus functions 602\} +≡
\[\text{void set(const real pos.x, const real pos.y, const real pos.z, const real dir.x, const real dir.y, const real dir.z, const real dist, const real ang = 0.0, char ax = 'z');}\]

607.

\{Define Focus functions 605\} +≡
\[\text{void Focus::set(const real pos.x, const real pos.y, const real pos.z, const real dir.x, const real dir.y, const real dir.z, const real dist, const real ang, char ax)}\]
\[
\{\text{Focus f(pos.x, pos.y, pos.z, dir.x, dir.y, dir.z, dist, ang, ax);}\]
\[
\text{*this = f;}\]
\[
\}
\]

608. Point arguments.

609. Constructor.

\{Declare Focus functions 602\} +≡
\[\text{Focus(const Point &pos, const Point &dir, const real dist, const real ang = 0.0, char ax = 'z');}\]

610.

\{Define Focus functions 605\} +≡
\[\text{Focus::Focus(const Point &pos, const Point &dir, const real dist, const real ang, char ax)}\]
\[
\{\text{Focus f(pos.get.x(), pos.get.y(), pos.get.z(), dir.get.x(), dir.get.y(), dir.get.z(), dist, ang, ax);}\]
\[
\text{*this = f;}\]
\[
\}
\]

611. Setting function. [LDF 2002.09.17.] Added this function.

\{Declare Focus functions 602\} +≡
\[\text{void set(const Point &pos, const Point &dir, const real dist, const real ang = 0.0, char ax = 'z');}\]

612.

\{Define Focus functions 605\} +≡
\[\text{void Focus::set(const Point &pos, const Point &dir, const real dist, const real ang, char ax)}\]
\[
\{\text{Focus f(pos.get.x(), pos.get.y(), pos.get.z(), dir.get.x(), dir.get.y(), dir.get.z(), dist, ang, ax);}\]
\[
\text{*this = f;}\]
\[
\}
\]

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.

\{Declare Focus functions 602\} +≡
\[\text{const Focus &operator=(const Focus &);}\]
614. Define Focus functions 605 \[\equiv\]
\[
\text{const Focus &Focus::operator=}(\text{const Focus &f})
\{
    \text{if } (\text{this} \equiv \&f) \quad /* [LDF 2002.10.13] Prevent self-assignment. */
    \begin{align*}
    \text{return } & \text{this}; \\
    \text{position} & = f.\text{position}; \\
    \text{direction} & = f.\text{direction}; \\
    \text{up} & = f.\text{up}; \\
    \text{distance} & = f.\text{distance}; \\
    \text{angle} & = f.\text{angle}; \\
    \text{axis} & = f.\text{axis}; \\
    \text{transform} & = f.\text{transform}; \\
    \text{persp} & = f.\text{persp}; \\
    \text{return } & \text{this};
    \end{align*}
\}
\]

\[\text{Declare Focus functions 602} \equiv\]
\[\text{void reset_angle} (\text{const real ang});\]
616. Define Focus functions 605 \( \equiv \)

```cpp
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 \( \equiv \) \( \cdot X \) or \( \cdot Z \), and the positive x-axis if axis \( \equiv \) \( \cdot Y \). */
    if (axis \( \equiv \) \( \cdot Z \) \( \lor \) axis \( \equiv \) \( \cdot X \)) up.set(0, 1, 0);
    else if (angle \( \neq \) 0) {
        if (axis \( \equiv \) \( \cdot Z \)) up *= transform.rotate(0, 0, -angle);
        else if (axis \( \equiv \) \( \cdot X \)) up *= transform.rotate(-angle);
        else if (axis \( \equiv \) \( \cdot Y \)) up *= transform.rotate(0, -angle);
        else {
            cerr \( \ll \) "ERROR! In Focus::Focus():\n                \( \cdot Y \) axis has invalid value: \( \cdot Y \) \( \ll \) axis \( \ll \) endl \( \ll \) "Rotating around z-axis. \n                Enter \< RETURN > to try to continue.\n            \ll \) flush;
            up *= transform.rotate(0, 0, -angle);
        }
    }
    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]


(Declare Focus functions 602) \( \equiv \)

```cpp
void show(const string text_str = "Focus:", const bool show_transforms = false) const;
```
618.  
(Define *Focus* functions 605) +≡

```cpp
void Focus::show(const string text_str, const bool show_transforms) const
{
    cout « text_str « endl;
    position.show("position:");
    direction.show("direction:");
    up.show("up:");
    cout « "distance,--" « distance « "axis,--" « axis « endl « flush;
    cout « "angle,--" « angle « endl « flush;
    if (show_transforms == 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) +≡

```cpp
inline const Point &get_position() const
{
    return position;
}
```

621. Get direction.

---

Log
---

[**LDF 2003.07.09.**] Added this function.

(Declare *Focus* functions 602) +≡

```cpp
inline const Point &get_direction() const
{
    return direction;
}
```


(Declare *Focus* functions 602) +≡

```cpp
inline const real &get_distance() const
{
    return distance;
}
```

623. Get up.  [LDF 2002.09.18.] Added this function.
(Declare Focus functions 602) +\equiv
inline const Point &get_up() const
{
    return up;
}

624. Get transform.

[LDF 2002.09.18.] Added this function.

(Declare Focus functions 602) +\equiv
inline const Transform &get_transform() const
{
    return transform;
}

625. Get transform element.

[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) +\equiv
real get_transform_element(const unsigned short row, const unsigned short column) const;
626.
\[\text{(Define Focus functions 605) +}\]
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.

\[\text{(Declare Focus functions 602) +}\]
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.

\[\text{(Declare Focus functions 602) +}\]
real get\_persp\_element(const unsigned short row, const unsigned short column) const;

629.
\[\text{(Define Focus functions 605) +}\]
real Focus::get\_persp\_element(const unsigned short row, const unsigned short column) const
{
    return persp.get\_element(row, column);
}

630. Global variables. [LDF 2002.12.08.] BUG. Commented-out, because of a bug involving trans-
formations.
Default value. Can be changed.
\[\text{(Global variables 18) +}\]
Focus default\_focus(0,10,-10,0,10,0,10);
631.
(Declarations for the header file 21) +≡

extern Focus default_focus;

632. Putting Point and Focus together.

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.
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()`.

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()`.

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.
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.

declared 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.

   (Define Point functions 330) \equiv
   Line Point::get_line(const Point &pt) const
   {
      Line l(*this,(pt – *this));
      return l;
   }

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.

   (Declare Line functions 643) \equiv
   Path get_path(void) const;

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.
(Define Point functions 330) +≡

#if 1
 */ 0 */
bool_point Point::intersection_point(const Point &pp0, const Point &pp1, const Point &qq0, const Point &qq1)
{
  /* START HERE DEBUGGING. [LDF 2003.06.24.] */
  bool DEBUG = false; /* true */
  if (DEBUG) cout << "Entering Point::intersection_point() \n";
  Line Lp = pp0.get_line(pp1);
  Line Lq = qq0.get_line(qq1);
  if (DEBUG) {
    pp0.show("pp0");
    pp1.show("pp1");
    qq0.show("qq0");
    qq1.show("qq1");
    Lp.show("L_p");
    Lq.show("L_q");
  }
  bool_real_point brp = Lp.get_distance(Lq);
  if (DEBUG) {
    cout << "brp.b,--u," << brp.b << endl << flush;
    cout << "brp.r,--u," << brp.r << endl << flush;
    brp.pt.show("brp.pt");
  }
  if (brp.r ≠ 0 ∨ brp.b ≡ false ∨ brp.pt ≡ INVALID_POINT) return INVALID_BOOL_POINT;
  bool_point bp;
  bp.pt = brp.pt;
  bool_real brp = brp.pt.is_on_segment(pp0, pp1);
  bool_real brq = brp.pt.is_on_segment(qq0, qq1);
  if (DEBUG) {
    cout << "br.p.first,--u," << brp.first << endl << flush;
    cout << "br.p.second,--u," << brp.second << endl << flush;
    cout << "br.q.first,--u," << brq.first << endl << flush;
    cout << "br.q.second,--u," << brq.second << endl << flush;
  }
  bp.b = (brp.first ∧ brq.first);
  if (DEBUG) cout << "Exiting Point::intersection_point() \n";
  return bp;
} #endif

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) 

\[
\text{bool real point get_distance(const Line &l) const;}
\]

649.

(Define Line functions 644) 

\[
\text{bool real point Line::get_distance(const Line &l) const \{ bool DEBUG = false; /* true */}
\]

\[
\text{if (DEBUG) cout << "Entering Line::get_distance() \n";}\]

\[
\text{bool real point brp;}
\]

\[
\text{Point normal = direction.cross_product(l.direction);}\]

\[
\text{if (DEBUG) normal.show("normal_after_cross_product.");}
\]

\[
\text{Point normal_unit = normal.unit_vector();}
\]

\[
\text{if (DEBUG) normal_unit.show("normal_unit.");}
\]

\[
\text{if (normal_unit == origin) \{}
\]

\[
\text{if (DEBUG) cout << "Lines are parallel. \n" << flush;}
\]

\[
\text{brp.b = false; /* No intersection. */}
\]

\[
\text{brp.pt = INVALID_POINT;}
\]

\[
\text{Point temp.pt(l.position);}\]

\[
\text{temp.pt -= position;}
\]

\[
\text{temp.pt = temp.pt.cross_product(direction);}\]

\[
\text{brp.r = temp.pt.magnitude() / direction.magnitude(); /* [LDF 2002.10.29] Distance. */}
\]

\[
\text{if (DEBUG) \{}
\]

\[
\text{cout << "distance in \n" << brp.r << endl << flush;}
\]

\[
\text{\} if (DEBUG) cout << "Exiting Line::get_distance() \n";}
\]

\[
\text{return brp;}
\]

\[
\text{else \{}
\]

\[
\text{if (DEBUG) cout << "Lines are not parallel. \n" << flush;}
\]

\[
\text{brp.r = fabs(l.position - position).dot_product(normal_unit);}\]

\[
\text{if (DEBUG) cout << "distance in \n" << brp.r << endl << flush;}
\]
650. Lines have an intersection.

[LDf 2003.08.27] Commented-out the declarations of \( v_x, v_y, \) and \( v_z, \) since they are not used. I haven't deleted them, in case I need them someday.

(Define Line functions 644)

```cpp
if (brp.r == Point::epsilon()) {
  if (DEBUG) cout << "Lines have an intersection. \n" << flush;
  brp.r = 0;
  brp.b = true;
  real a_x = position.get_x();
  real a_y = position.get_y();
  real a_z = position.get_z();
  real b_x = l.position.get_x();
  real b_y = l.position.get_y();
  real b_z = l.position.get_z();
  #if 0
  real u_x = direction.get_x();
  real u_y = direction.get_y();
  real u_z = direction.get_z();
  #endif
  real w_x = l.direction.get_x();
  real w_y = l.direction.get_y();
  real w_z = l.direction.get_z();
  real u_w = normal.get_x();
  real u_y = normal.get_y();
  real u_z = normal.get_z();
  real t;
  if (u_w != 0) {  
    if (DEBUG) cout << "u_z is normalized, u_w != 0\n";
    t = ((b_y - a_y) * w_y) - ((b_z - a_z) * w_z)) / u_w;
  } else if (u_x != 0) {  
    if (DEBUG) cout << "u_x is normalized, u_w != 0\n";
    t = ((b_y - a_y) * w_x) - ((b_z - a_z) * w_y)) / u_x;
  } else if (u_y != 0) {  
    if (DEBUG) cout << "u_y is normalized, u_w != 0\n";
    t = ((b_x - a_x) * w_y) - ((b_z - a_z) * w_z)) / u_y;
  } else {
    cerr << "This case can't happen\n" << "In Line::get_distance()\n" << "normalized normal, origin, this case should have been\n" << "caught above, so something is really wrong.\n" << "Returning \nINVALID_BOOL_REAL_POINT.\n\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: ");
}
if (DEBUG) cout << "Exiting Line: \text{get_distance}(\n"
return brp;
}

651. Lines are not parallel, but do not intersect.
(Define Line functions 644) +≡
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: \text{get_distance}(\n"
        return brp;
}
}

652. Show.
(Declare Line functions 643) +≡
void show(string text = "");

653. (Define Line functions 644) +≡
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) ≡
    extern const Line INVALID_LINE(INVALID_POINT, INVALID_POINT);
This code is used in section 657.

655. (Declarations for the header file 21) +≡
    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)
658. This is what's written to lines.h.

��态 658) ndef
  (Define struct Line 637)
  (Declarations for the header file 21)

659. Plane (planes.web). [LDF 2002.10.29] 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 = "$Id:\planes.web,v.1.4,2004/01/12,21:31:54,lfinstoi,Exp,$";

660. Include files.

(Scene 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"

661. Plane struct definition.

  format Plane Line
  (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.

662. Constructors.

663. 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.
[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.

\begin{verbatim}
{ Define Plane functions 664 } +=
    Plane :: Plane(const Plane &p)
    {
      *this = p;
      return;
    }
\end{verbatim}

667. Point arguments.

\begin{verbatim}
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);
\end{verbatim}
§668.  
(Define Plane functions 664) +≡

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():\nnormal=origin.\n" <<= "Plane.is.INVALID_PLANE.\n" <<=  
        point = INVALID_POINT;  
        distance = INVALID_REAL;  
        return;  
    }  
    normal.unit_vector(true);  
    distance = -point.dot_product(normal);  
    if (fabs(distance) < Point::epsilon()) distance = 0;  
    return;  
}

§669. Assignment.  

Log  

[LD 2003-06-06] Added this function.

(Declare Plane functions 663) +≡

const Plane &operator=(const Plane &p);
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.

[Log]

        [LDF 2003.06.06.] Added this section.

672.  Equality.

[Log]

        [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.

[Log]

        [LDF 2003.06.06.] Added this function.

\{Declare Plane functions 663\} +≡
bool operator!=(const Plane &p) const;
675.  
\[ \text{Define Plane functions 664} \implies \]
\[
\begin{align*}
\text{bool Plane::operator\textbar(const Plane \& p) const} \\
\{ \\
\text{return } \neg(\text{operator\textbar(p)}); \\
\}
\end{align*}
\]

676.  Get distance.

677.  Point argument.  [LDF 2003.06.03]  This function returns a \texttt{real\_short} \( r \), where \( r_{, \text{first}} \) is the distance of the \texttt{Point} from the \texttt{Plane}.  \( r_{, \text{first}} \) is always positive.  \( r_{, \text{second}} \) can take on the following values:
- 0 If the \texttt{Point} lies in the \texttt{Plane}.
- 1 If it lies on the side of the \texttt{Plane} pointed at by the normal to the \texttt{Plane}, considered to be the "outside".
- -1 If it lies on the side of the \texttt{Plane} \textit{not} pointed at by the normal to the \texttt{Plane}, considered to be the "inside".

[LDF 2003.06.03.]  Changed the definition of this function.  The old definition was incorrect.  Also changed return type from \texttt{real} to \texttt{real\_short}.

[LDF 2003.06.04.]  BUG FIX:  In the case that \( r_{, \text{fabs}} < \texttt{Point::epsilon()} \), 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.

\[
\begin{align*}
\text{Declare Plane functions 663} \implies \texttt{real\_short get\_distance(const Point \& p) const} ;
\end{align*}
\]

678.  
\[ \text{Define Plane functions 664} \implies \]
\[
\begin{align*}
\text{real\_short Plane::get\_distance(const Point \& p) const} \\
\{ \\
r_{, \text{real}} = (p - \text{point}).dot\_product(normal); \\
r_{, \text{fabs}} = \text{fabs}(r); \\
s_{, \text{signed short}} = \text{static\_cast(signed short)}(r / r_{, \text{fabs}}); \\
return \text{real\_short}(r_{, \text{fabs}}, s); \\
\}
\end{align*}
\]

679.  No argument.  [LDF 2003.06.03.]  This version of \texttt{get\_distance()} returns the data member \texttt{distance} and its sign, i.e., the distance of \texttt{origin} to the \texttt{Plane}, and which side of the \texttt{Plane} it lies on.  I’m not using \texttt{origin} as the default for an optional \texttt{Point} argument, because of problems that may arise, when I implement \texttt{user\_coordinates} and \texttt{view\_coordinates}.

[LDF 2003.06.03.]  Added this function.

\[
\begin{align*}
\text{Declare Plane functions 663} \implies \texttt{real\_short get\_distance(void) const} ;
\end{align*}
\]
680.  
{Define Plane functions 664 } +≡
    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 true, if the Point lies on the Plane 
    p, otherwise false. Declared in points.web. Must be defined here, because Plane is an incomplete type in 
    that file.

<table>
<thead>
<tr>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>[LDF 2003.06.04.] Added this function</td>
</tr>
</tbody>
</table>

682.  Intersection.

683.  Intersection with a line.

684.  Point arguments.

<table>
<thead>
<tr>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>[LDF 2003.06.03.] Added this function</td>
</tr>
</tbody>
</table>

{Declare Plane functions 663 } +≡
    bool_point intersection_point(const Point &p0,const Point &p1) const;
685. 〈Define Plane functions 664〉 \equiv
\begin{verbatim}
bool_point Plane::intersection_point(const Point &p0, const Point &p1) const
{
  bool_point bp;
  real denominator = (p0 - p1).dot_product(normal);
  if (denominator \equiv 0) /* !! TO DO: Handle cases: Path is in Plane, and Path is in a parallel Plane. [LDF 2003.06.03] */
  {
    cerr << "ERROR! In Plane::intersection_point(const Point\&, const Point\&):"
    << "denominator is 0...Can’t divide..."
    << "Path is either in Plane, or parallel to\n    Plane."
    << endl
    << "Returning INVALID_BOOL_POINT."
    << endl
    << flush;
    return INVALID_BOOL_POINT;
  }
  real numerator = p0.dot_product(normal) + distance;
  bp.pt = p0 + ((numerator/denominator) * (p1 - p0));
  bp.b = (bp.pt \equiv INVALID_POINT) ? true : false;
  return bp;
}
\end{verbatim}

686. Path argument. [LDF 2003.06.03] Defined in paths.web, because Path is an incomplete type in this file.

\begin{verbatim}
[LDF 2003.06.03]  \hspace{1cm} Added this function.
\end{verbatim}

\begin{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\begin{verbatim}
\end{verbatim}

687. Intersection of two Planes. [LDF 2002.10.29] TO DO: Look up and explain!

\begin{verbatim}
[LDF 2003.06.04]  \hspace{1cm} Changed to const.
\end{verbatim}

\begin{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
688.

(Define Plane functions 664) +

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 not coincident. \n" << "Returning INVALID_LINE. \n\n" << flush;
            return INVALID_LINE;
        }
    } else {
        cerr << "Planes are not coincident. \n" << "Returning INVALID_LINE. \n\n" << flush;
        return INVALID_LINE;
    }
    /* 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.get_x();
    real ny = normal.get_y();
    real nz = normal.get_z();
    real mx = pl.normal.get_x();
    real my = pl.normal.get_y();
    real mz = pl.normal.get_z();
    real vx = direction_vector.get_x();
    real vy = direction_vector.get_y();
    real vz = direction_vector.get_z();
    if (direction_vector.get_x() != 0) {
        x = 0;
        y = -1 * ((d * mz - e * nz) / vx);
        z = (d * my - e * ny) / vx;
    } else if (direction_vector.get_y() != 0) {
        x = (d * mz - e * nz) / vy;
        y = 0;
        z = -1 * ((d * mx - e * nx) / vy);
    } else {
        x = -1 * ((d * my - e * ny) / vz);
    }
\[ y = (d \cdot mx - e \cdot nx)/uz; \]
\[ z = 0; \]

```cpp
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.

(Declare Plane functions 663) +≡
void show(string text = "") const;

690.

(Define Plane functions 664) +≡
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, -" << distance << endl << endl << flush;
}

691. Global constants for Plane.

(Plane global constants 691) ≡
extern const Plane INVALID_PLANE(INVALID_POINT, INVALID_POINT);
This code is used in section 694.

692.

(Declarations for the header file 21) +≡
extern const Plane INVALID_PLANE;

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 = "$Id:\dpaths.web,v\d1.7,v2004/01/12,21:30:51\d,lfinsto1\dExp,$";

697. Include files.

(Include files 6) \equiv
  #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"

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 2002.4.8.] Added static variables for help lines and curves: help_color, help_dash_pattern, do_highlight. 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  3DLDF-1.15.1

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.

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) +≡

    Path();

705.

(Define Path functions 701) +≡

    Path::Path()
    {
        bool DEBUG = false; /* true */
        if (DEBUG) cout << "Entering Path().\n" << flush;
        on_free_store = false;
        line_switch = false;
        cycle_switch = false;
        fill_draw_value = 0;
        dashed = "n"; /* LDF 2003.04.06. Added these three lines. */
        pen = "n";
        arrow = false;
        draw_color = 0;
        fill_color = 0;
        do_output = true; /* LDF 2002.09.18. Added this line. */
        projective_extremes.resize(6, 0); /* LDF 2002.09.18. Added this line. */
        if (DEBUG) cout << "Exiting Path().\n" << flush;
        return;
    }

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) +≡

    Path(const Point &p0, const Point &p1);
708.  
(Define Path functions 701) +≡

Path::Path(const Point &p0, const Point &p1) { bool DEBUG = false;  /* true */
    if (DEBUG) cout « "Entering Path() at line", version)." « flush;
    line_switch = true;
    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 > (p0) ) ;
    points.push_back ( create_new < Point > (p1) ) ;
    connectors.push_back("-n");
    if (DEBUG) cout « "Exiting Path() at line", version)." « flush;
    return; }

709. Setting function.
(Declare Path functions 700) +≡

void set(const Point &p0, const Point &p1);

710.  
(Define Path functions 701) +≡

void Path::set(const Point &p0, const Point &p1) { line_switch = true;
    cycle_switch = 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("--n");
    line_switch = 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 "...".

§708  3DLDF-1.1.5.1  CONSTRUCTOR  197
712. Constructor.

(Declare Path functions 700) +≡

Path(string connector, bool cycle ...);

713.

(Define Path functions 701) +≡

Path::Path(string connector, bool cycle ...){
  bool DEBUG = false; /* true */
  if (DEBUG) cout << "Entering Path() with connector, cycle...\n" << flush;
  line_switch = false;
  cycle_switch = cycle;
  connectors.push_back(connector);
  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;
  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);
  if (DEBUG) cout << "Exiting Path() with connector, cycle...\n" << flush;
  return;
}

714. Setting function.

(Declare Path functions 700) +≡

void set(string connector, bool cycle ...);
715.  
(Define Path functions 701) +≡

```cpp
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)
fill_draw_value = 0;
draw_color = 0;
fill_color = 0;
dashed = "n"; /* 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.

---

![Log]

[LDF 2002.10.29.] BUG FIX: No longer pushing first_point_ptr onto points. Copying it instead.

717. Constructor.
(Declare Path functions 700) +≡

```cpp
Path ( Point *first_point_ptr ... );
```
718. 

(Define Path functions 701) \( \equiv \)

```cpp
Path::Path (Point *first_point_ptr ...) { bool DEBUG = false; /* true */
    if (DEBUG) cout << "Entering Path()\n(Point*...).\n" << 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.argv(ap, char *)) ) ) 
        direct_cast (char *) (0) )
    
    (connector_string = connector_ptr);
    if ( ((point_ptr = va.argv(ap, Point *)) ) ) static_cast (Point *) (0) )
        cycle_switch = true;
        break;
    }
    points.push_back (create_new < Point > (point_ptr)); } va_end(ap);
    if (DEBUG) cout << "Exiting Path()\n(Point*...).\n" << flush;
}
```

719. Setting function.

(Declare Path functions 700) \( \equiv \)

```cpp
void set (Point *first_point_ptr ... );
```
§720.
(Define Path functions 701) +≡

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_arg(ap,char *)) ≠ (char *) 0) {
        connectors.push_back((connector_string = connector_ptr));
        if ((point_ptr = va_arg(ap,Point *)) ≡ 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 “*this = p” in a copy constructor.

Log

[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.

[LDF 2002.11.03.] Rewrote this function. Now just using the assignment operator.

(Declare Path functions 706) +≡

Path(const Path &p);
722.  
(Define Path functions 701 ) +≡  
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 ≠ 0, the new Path is assigned to using the values from p.  
[Log]  [LDF 2003.12.30.] Replaced this version of Path::create_new_path() with a specialization of template<class  
C> C*create_new().  
[Log]  [LDF 2003.12.30.] Changed the argument. It’s now a const Path *.  
[Log]  [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 ) +≡  
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.  
[Log]  [LDF 2003.12.30.] Replaced this version of Path::create_new_path() with a specialization of template<class  
C> C*create_new().  
[Log]  [LDF 2003.12.30.] Changed argument from Path to const Path &.

(Declare non-member template functions for Path 724 ) +≡  
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 ) +≡  
virtual ~Path();
727. !! Make sure to delete anything else that I allocate dynamically!

\( \text{Define Path functions 701} \) \( \implies \)

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. Shape +t Path *Path() const
   *Path = create_new < Path > (0);
   return static_cast<Shape*>(p);

§732. Set on free store.


(Declare Path functions 700) +t
   virtual bool set_on_free_store(bool b = true);

§733. bool Path::set_on_free_store(bool b)
   {
      on_free_store = b;
      return b;
   }

§734. Setting drawing and filling data.

§735. Set fill_draw_value.
(Declare Path functions 700) +t
   virtual void set_fill_draw_value(const signed short s);

§736. void Path::set_fill_draw_value(const signed short s)
   {
      fill_draw_value = s;
      return;
   }

§737. Set draw color.

§738. Color version.
(Declare Path functions 700) +t
   virtual void set_draw_color(const Color& c);

§739. void Path::set_draw_color(const Color& c)
   {
      draw_color = &c;
      return;
   }
740. Color pointer version.

(Declare Path functions 700) \(\equiv\)

\[
\text{virtual void set\_draw\_color(const Color \&c);}\]

741.

(Define Path functions 701) \(\equiv\)

\[
\text{void Path::set\_draw\_color(const Color \&c)}\]

\[
\{
\text{if } \left( \text{draw\_color} \neq 0 \land \text{draw\_color\_is\_on\_free\_store()} \equiv \text{true} \right) \{ \\
\text{delete draw\_color;} \\
\text{draw\_color} = c; \\
\text{return;} \\
\}
\]

742. Set fill color.

743. Color version.

(Declare Path functions 700) \(\equiv\)

\[
\text{virtual void set\_fill\_color(const Color \&c);}\]

744.

(Define Path functions 701) \(\equiv\)

\[
\text{void Path::set\_fill\_color(const Color \&c)}\]

\[
\{
\text{fill\_color} = \&c; \\
\text{return;} \\
\}
\]

745. Color pointer version.

(Declare Path functions 700) \(\equiv\)

\[
\text{virtual void set\_fill\_color(const Color \&c);}\]

746.

(Define Path functions 701) \(\equiv\)

\[
\text{void Path::set\_fill\_color(const Color \&c)}\]

\[
\{
\text{if } \left( \text{fill\_color} \neq 0 \land \text{fill\_color\_is\_on\_free\_store()} \equiv \text{true} \right) \{ \\
\text{delete fill\_color;} \\
\text{fill\_color} = c; \\
\text{return;} \\
\}
\]

747. Set dash pattern.

(Declare Path functions 700) \(\equiv\)

\[
\text{virtual void set\_dash\_pattern(const string s = \"\);}\]
748. (Define Path functions 701) +≡

void Path::set_dash_pattern(const string s)
{
    dashed = s;
    return;
}

749. Set pen.
(Declare Path functions 700) +≡

virtual void set_pen(const string s = "");

750. (Define Path functions 701) +≡

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.

[Log] Added this function.

(Declare Path functions 700) +≡

virtual void set_connectors(const string s = ". .");

752. (Define Path functions 701) +≡

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) +≡

virtual Transform rotate(const real x, const real y = 0, const real z = 0);
757. (Define Path functions 701) +≡
    Transform Path :: rotate(const real x, const real y, const real z)
    {
        Transform t;
        t.rotate(x, y, z);
        return *this == t;
    }

758. Rotation around an arbitrary axis.

759. Transform version. Declared in transfor.web. Must be defined here, because Path is an
    incomplete type there.

---

[Log]

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

[LDF 2003.07.06.] Changed, so that is_linear() is used, instead of get_line_switch().

---

(Define Transform functions 169) +≡
    Transform Transform :: rotate(const Path &p, const real angle)
    {
        if (!p.is_linear()) {
            cerr ≡ "ERROR! In Transform::rotate(Path, real).\n" ≡ "Path! is! no! to! linear! \nReturning! INVALID_TRANSFORM.\n\n";
            return INVALID_TRANSFORM;
        }
        Transform t;
        t.rotate(p.get_start_point(), 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.

```c
(Define Point functions 330) +=
  Transform Point::rotate(const Path &p, const real angle)
  {
    if (!p.get_line_switch()) {
      cerr << "ERROR! In Point::rotate(Path, real).\n"      << "Path is not a line segment. Returning INVALID_TRANSFORM.\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 2002.04.7.] 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.

```c
(Declare Path functions 700) +=
  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.

```c
(Define Path functions 701) +=
  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.

(Declare Path functions 700) +≡
    Transform rotate(const Path &p, const real angle = 180);

765.

(Define Path functions 701) +≡
    Transform Path::rotate(const Path &p, const real angle)
    {
        if (!p.isLinear()) {
            cerr << "ERROR! Path::rotate(Path, real).\n";
            return INVALID_TRANSFORM;
        }
        return rotate(p.getPoint(0), p.getLastPoint(), angle);
    }

766. Scale.  [LDF 2002.12.20.] TO DO: Make all of the transformations virtual!

(Declare Path functions 700) +≡
    Transform scale(real x, real y = 1, real z = 1);

767.

(Define Path functions 701) +≡
    Transform Path::scale(real x, real y, real z)
    {
        Transform t;
        t.scale(x, y, z);
        return (*this += t);
    }

768. Shear.

(Declare Path functions 700) +≡
    Transform shear(real xy, real xz = 0, real yx = 0, real yz = 0, real zx = 0, real zy = 0);
769.
\[\text{Define Path functions 701} \equiv\]
\text{Transform Path :: shear(real xy, real xz, real yx, real yz, real xx, real xy)}
\{
    \text{bool DEBUG = false; /* true */}
    \text{if (DEBUG) cout << "Entering Path::shear().\n";}
    \text{Transform t;}
    \text{t.shear(xy, xz, yx, yz, xx, xy);}
    \text{if (DEBUG) cout << "Exiting Path::shear().\n";}
    \text{return (*this += t);}
\}

770. Shift.
\[\text{real arguments.}\]
\[\text{Declare Path functions 700} \equiv\]
\text{Transform shift(real x, real y = 0, real z = 0)};

772.
\[\text{Define Path functions 701} \equiv\]
\text{Transform Path :: shift(real x, real y, real z)}
\{
    \text{Transform t;}
    \text{t.shift(x, y, z);}
    \text{return (*this += t);}
\}

773. Point argument.
\[\text{Declare Path functions 700} \equiv\]
\text{Transform shift(const Point &p)};

774.
\[\text{Define Path functions 701} \equiv\]
\text{Transform Path :: shift(const Point &p)}
\{
    \text{return shift(p.get_x(), p.get_y(), p.get_z());}
\}

775. Shift times.
\[\text{LDF 2003.01.19.} \text{ shift times() returns void, because Path doesn’t have a Transform data member,}
\text{and there’s no guarantee that all of the Points on points will have identical transforms.}\]
\[\text{LDF 2003.01.19.} \text{ Note that shift times() will only have an effect on the Points on a Path if it’s called}
\text{after a call to shift() and before an operation is applied that causes Point :: apply_transform() to be called.}\]
\[\text{LDF 2003.01.19.} \text{ Added this section.}\]
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!
(Declare Path functions 700) +≡
   bool project(const Focus &f, const unsigned short proj, real factor);

786.  (Define Path functions 701) +≡
   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! In Path::project():\n" << "Point::project() returned false.\n" <<
               "Returning false.\n\n" << 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.

   Log  
   [LDF 2002.11.03.]  Changed char argument to const char.

(Declare Path functions 700) +≡
   Transform align_axis(const char axis = 'z') const;
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.is_not_linear() 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.is_not_linear() 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.\nIt won’t cause any harm, though.\n\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 \( \overrightarrow{p_0p_1} \) 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 &p0, const Point &p1, const char axis);

795. (Define Path functions 701) +≡
 Transform Path::align_with_axis(const Point &p0, const Point &p1, 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\} +≡
\hspace{1em} 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.

\hline
\multicolumn{3}{l}{[LDF 2003.02.09.] Added this function.}
\hline
\{Declare Path functions 700\} +≡
\hspace{1em} void operator+=(const string s);

802.  
\{Define Path functions 701\} +≡
\hspace{1em} void Path::operator+=(const string s)
\hspace{3em} {
\hspace{5em} connectors.push_back (s);
\hspace{5em} return;
\hspace{3em} }

803. Concatenating Paths.

804. Versions using “&”.

805. With assignment. This function appends the Path argument pa to *this.

\hline
\multicolumn{3}{l}{[LDF 2002.4.6.] Added this function.}
\multicolumn{3}{l}{[LDF 2002.11.03.] Made non-inline.}
\hline
\{Declare Path functions 700\} +≡
\hspace{1em} void operator&=(const Path &pa);

806.  
\{Define Path functions 701\} +≡
\hspace{1em} void Path::operator&=(const Path &pa){
\hspace{3em} if (is_cycle() \lor pa.is_cycle()) /* Return if either one of the Paths is a cycle. */
\hspace{5em} {
\hspace{7em} cerr \ll \"ERROR! In Path::operator&(Path&).\n\" \ll 
\hspace{9em} \"One of the Paths is a cycle, can’t concatenate.\n\" \ll \"Returning *this.\n\";
\hspace{7em} return;
\hspace{5em} }
\hspace{1em} 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.

---

[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! In Path::operator&(Path&)." << "Paths don’t touch.\n" << "Using\n"..\"to join them instead of\"\n\".\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.

---

[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 &amp; = 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. METAFOR 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_connector = r.connectors.back();
    while (r.connectors.size() < r.points.size() - 1) r.connectors.push_back(last_connector);
    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 iter = 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 iter = 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*.

```cpp
(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 &ddmw_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! In Path::draw():\n" << "Path doesn't contain any Points.\n" <<
        "Not doing anything.\n\n" << flush;
        return;
      }
  Path *p = create_new < Path > (*this);
  p->fill_draw_value = DRAW;
  p->arrow = arrow;
  if (DEBUG)
    cout << "ddmw_color.get_use_name()=\n" << ddmw_color.get_use_name() << endl << flush;
  if (ddmw_color.get_use_name() == false)
    if (DEBUG) cout << "Allocating memory for Color.\n" << flush;
    Color *c = create_new < Color > (0);
    *c = ddmw_color;
    p->draw_color = c; }
  else {
    if (DEBUG) cout << "ddmw_color.get_name()=\n" << ddmw_color.get_name() << endl << flush;
    p->draw_color = &ddmw_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.

```cpp
(Declare Path functions 700) +=
virtual void draw(Picture &picture, const Color &ddraw_color = *Colors::default_color, string ddashed = "", string ppen = "", bool aarrow = false) const;
```

821.

```cpp
(Define Path functions 701) +=
void Path::draw(Picture &picture, const Color &ddraw_color, string ddashed, string ppen, bool aarrow) const
{
    draw(ddraw_color, ddashed, ppen, picture, aarrow);
}
```

822. **Point versions.** Declared in points.web, but must be defined here, because Path is an incomplete type here.

[LDF 2002.11.03.] Unlike the Path versions of draw(), this function returns a Path. This can be useful, if you want to use the Path that it creates again for something else.

823. **Normal version.**

```cpp
(Define Point functions 330) +=
Path Point::draw(const Point &p, const Color &ddraw_color, string ddashed, string ppen, Picture &picture, bool aarrow) const
{
    Path pa(*this, p);
    pa.draw(ddraw_color, ddashed, ppen, picture, aarrow);
    return pa;
}
```

824. **Picture argument first.** [LDF 2002.09.17.] Added this function. It’s convenient for when I want to pass a Picture argument.

```cpp
(Define Point functions 330) +=
Path Point::draw(Picture &picture, const Point &p, const Color &ddraw_color, string ddashed, string ppen, bool aarrow)
{
    return draw(p, ddraw_color, ddashed, ppen, picture, aarrow);
}
```

825. **Draw arrow.**

826. **Path versions.**
§827. Normal version.

\[
\begin{array}{lll}
\text{Log} & \text{Log} & \text{Log} \\
\hline
\end{array}
\]

[\text{LDF 2003.01.15.}] \text{Added this function.}

\begin{verbatim}
(Declare Path functions 700) +\equiv
  virtual void drawarrow (const Color &ddraw_color = *Colors::default_color, string ddashed = "", string ppen = "", Picture &picture = current_picture) const;
\end{verbatim}

828.

(Define Path functions 701) +\equiv

\begin{verbatim}
void Path :: drawarrow (const Color &ddraw_color, string ddashed, string ppen, Picture &picture) const
{
  draw (ddraw_color, ddashed, ppen, picture, true);
}
\end{verbatim}

829. Picture argument first.

\[
\begin{array}{lll}
\text{Log} & \text{Log} & \text{Log} \\
\hline
\end{array}
\]

[\text{LDF 2003.01.15.}] \text{Added this function.}

\begin{verbatim}
(Declare Path functions 700) +\equiv
  virtual void drawarrow (Picture &picture, const Color &ddraw_color = *Colors::default_color, string ddashed = "", string ppen = "") const;
\end{verbatim}

830.

(Define Path functions 701) +\equiv

\begin{verbatim}
void Path :: drawarrow (Picture &picture, const Color &ddraw_color, string ddashed, string ppen) const
{
  draw (picture, ddraw_color, ddashed, ppen, true);
}
\end{verbatim}

831. Point versions.

832. Normal version.

\[
\begin{array}{lll}
\text{Log} & \text{Log} & \text{Log} \\
\hline
\end{array}
\]

[\text{LDF 2003.01.15.}] \text{Added this function.}

[\text{LDF 2003.06.03.}] \text{Made drawarrow \( ) \) const.

\begin{verbatim}
(Define Point functions 330) +\equiv
  Path Point :: drawarrow (const Point &p, const Color &ddraw_color, string ddashed, string ppen, Picture &picture) const
  {
    Path pa (*this, p);
    pa.drawarrow (ddraw_color, ddashed, ppen, picture);
    return pa;
  }
\end{verbatim}
833. Picture argument first.

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

Define Point functions 330) +
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.

[LD 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.
[LD 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(). [LD 2002.11.03.] I could have help lines outputted
last, if I had them on a vector of their own.

835. Path versions. [LD 2002.12.20.] ?? Could these functions be const?

836. Normal version. [LD 2002.4.8.] Added this function.
Declare Path functions 700) +
void drawHelp(const Color &ddraw_color = *help_color, string dashed = help_dash_pattern, string
ppen = "", Picture &picture = current_picture) const;

837.
Define Path functions 701) +
void Path::drawHelp(const Color &ddraw_color, string dashed, string ppen, Picture &picture)
    const
{
    if (doHelp_lines = false) return;
    draw (ddraw_color, dashed, ppen, picture);
}

838. Picture argument first.
Declare Path functions 700) +
void drawHelp(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 ddashed, string ppen)
const
{
    draw_help(ddraw_color, ddashed, 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 ddashed, string
ppen, Picture &picture) const
{
    Path pa(*this, pt);
    pa.draw_help(ddraw_color, ddashed, 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
ddashed, string ppen) const
{
    return draw_help(pt, ddraw_color, ddashed, ppen, picture);
}

843. Fill.
!! [LDF 2003.02.02.] Filling doesn’t use a pen!! Change everywhere!!.


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

[LP 2002.10.07.] Added code for handling and fill_color.

(Declare Path functions 799) +≡

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! In Path::fill():\n" << "Path doesn’t contain any Points.\n" << "Not doing anything.\n\n" << flush;
        return;
     }
Path *p = create_new < Path > (0);
   *p = *this;
   p->fill.draw_value = FILL;
   if (DEBUG)
      cout << "ffill_color.get_use_name()<<\n" << ffill_color.get_use_name() << endl << flush;
   if (ffill_color.get_use_name() == false) {
      if (DEBUG) cout << "Allocating memory for Color.\n" << flush;
      Color *c = create_new < Color > (0);
      *c = ffill_color;
      p->fillcolor = c; }
else {
      if (DEBUG) cout << "ffill_color.get_name()<<\n" << ffill_color.get_name() << endl << flush;
      p->fill_color = &ffill_color;
   }
   p->pen = "\n";
   p->dashed = "\n";
   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 &fill_color)
{
    fill(fill_color, 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.


[Log]

[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 dashed = "", string ppen = "", Picture &picture = current_picture) const;
850. (Define Path functions 701) +≡

```cpp
void Path::filldraw(const Color &ddraw_color, const Color &fill_color, string ddashed, string ppen, Picture &picture) const { bool DEBUG = false; /* true */
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. */
    
    { err << "WARNING! In Path::filldraw():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 = FILLDRAW;
if (DEBUG)
    cout << "ddraw_color.get_use_name().n--n" << ddraw_color.get_use_name() << endl << flush;
if (DEBUG)
    cout << "fill_color.get_use_name().n--n" << fill_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 << "ddraw_color.get_name().n--n" << ddraw_color.get_name() << endl << flush;
p->draw_color = &ddraw_color;
}
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--n" << fill_color.get_name() << endl << flush;
p->fill_color = &fill_color;
}
p->dashed = ddashed;
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) +≡

```cpp
void filldraw(Picture &picture, const Color &ddraw_color = *Colors::default_color, const Color &fill_color = *Colors::background_color, string ddashed = "", string ppen = ") const;
```
§852 3DLDF-1.1.5.1  PICTURE ARGUMENT FIRST  227

852.
(Define Path functions 701) +≡
void Path::filldraw(Picture &picture, const Color &draw_color, const Color &fill_color, string
dashed, string ppen) const
{
    filldraw(draw_color, fill_color, dashed, ppen, picture);
}

853. Undraw.

854. Path versions.

855. Normal version.
(Declare Path functions 700) +≡
void undraw(string dashed = "", string ppen = "", Picture &picture = current_picture);

856. (Define Path functions 701) +≡
void Path::undraw(string dashed, string ppen, 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! In Path::undraw():" << " Path doesn’t contain any points. 
" <<
                "Not doing anything. \n" << flush;
            return;
        }
    Path *p = create_new < Path > (this);
    p->draw_value = UNDRAW;
    p->draw_color = 0;
    p->fill_color = 0;
    p->dashed = dashed;
    p->pen = ppen;
    picture += static_cast<Shape *>(p); }

(Declare Path functions 700) +≡
void undraw(Picture &picture, string dashed = "", string ppen = "");

858. (Define Path functions 701) +≡
void Path::undraw(Picture &picture, string dashed, string ppen)
{
    undraw(dashed, ppen, 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 2002.04.08.] Added this function.
[LDF 2002.11.03.] Changed this function, so that it returns the `Path pa`, instead of `void`.

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

```
(Define Point functions 330) \equiv
Path Point::undraw(Picture &picture, const Point &pt, string ddashed, string ppen)
{
    return undraw(pt, ddashed, ppen, picture);
}
```

862. Unfill.

863. Normal version.

```
(Declare Path functions 700) \equiv
void unfill(Picture &picture = current_picture);
```
864. (Define Path functions 701) +≡
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\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) +≡
void unfilldraw(const Color &ddraw_color = *Colors::background_color, string ddashed = "n", string ppen = "n", Picture &picture = current_picture);
867. Define Path functions 701 +→

```c
void Path::unfilldraw(const Color &draw_color, string ddashed, 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. */
        {
            cerr << "WARNING! In Path::unfilldraw():\n" << "Path doesn't contain any Points.\n" << "Not doing anything.\n\n" << flush;
            return;
        }
    Path *p = create_new < Path > (this);
    p->fill_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) {
        if (DEBUG) cout << "Allocating memory for Color.\n" << flush;
        Color *c = create_new < Color > (0);
        *c = draw_color;
        p->draw_color = c;
    } else {
        if (DEBUG) cout << "ddraw_color.get_name() == false.\n" << ddraw_color.get_name() << \n            "end\n" << flush;
        p->draw_color = &ddraw_color;
    }
    p->fill_color = 0;
    p->dashed = ddashed;
    p->pen = 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.

```c
(Declare Path functions 700) +→

void unfilldraw(Picture &picture, const Color &draw_color = Colors::background_color, string ddashed = "", string ppen = ");
```
869. 
(Define Path functions 701) +≡
void Path::unfilldraw(Picture &picture, const Color &ddraw_color, string ddashed, string ppen)
{
    unfilldraw(ddraw_color, ddashed, ppen, picture);
}

870. Labelling.

871. Label.

[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.

[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, etc. Now, the conditional tests for VIEW_VALUES_X, Y ≤ text_short ≤ 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) +≡
void label(short text_short = 0, const string position_string = "top", const bool dot = false, Picture &picture = current_picture) const;
873.  \(\text{Define \textbf{Path} functions 701}\) \(\implies\)

\[
\begin{align*}
\text{void Path::label(short text_short, const string &position_string, const bool &dot, Picture &picture)} & \quad \text{const} \\
\text{const} & \\
\{ \\
\text{bool DEBUG = false; /* true */} \\
\text{if (DEBUG) cout } \ll \text{"Entering Path::label() \"n \ll flush;} \\
\text{if (Label::DO_LABELS } \equiv \text{false) return;} \\
\text{if (points.size() } \equiv \text{0)} \\
\text{/* LDF 2002.09.27. Added this error handling code. If the Path is empty, don’t label it. */} \\
\{ \\
\text{cerr } \ll \text{"WARNING! In Path::label():n ”} \ll \text{"Path does’t contain any Points. \"n } \ll \\
\text{"Not doing anything. \"n \ll flush;} \\
\text{return;} \\
\} \\
\text{for (vector(Point *)::const_iterator iter = points.begin(); iter } \neq \text{points.end(); iter ++)} \\
\{ \\
\text{(*iter).label(text_short, position_string, dot, picture);} \\
\text{if (text_short } \leq \text{Point::WORLD_VALUES } \land \text{text_short } \geq \text{Point::VIEW_VALUES)} \\
\text{/* LDF 2003.05.06. Changed this conditional. */} \\
\text{; }  \\
\text{/* Do nothing. */} \\
\text{else ++text_short;} \\
\} \\
\text{if (DEBUG) cout } \ll \text{"Exiting Path::label():n ”} \ll \text{"n } \ll \text{flush;} \\
\}
\end{align*}
\]

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}.

\[
\begin{align*}
\text{Log} \\
\end{align*}
\]

875.  \(\text{Declare \textbf{Path} functions 700}\) \(\implies\)

\[
\begin{align*}
\text{void label(Picture &picture, short text_short = 0, const string position_string } = \text{"top", const bool &dot } = \text{false) const;}
\end{align*}
\]

876.  \textbf{Dotlabel.}

[LDF 2003.07.09.] Made `text_short` and `position_string` arguments `const`.

```cpp
(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) +≡

```cpp
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.

[LDF 2002.09.17.] Added this function. It’s convenient for when I want to pass a `Picture` argument.

[LDF 2003.07.09.] Made `text_short` and `position_string` arguments `const`.

```cpp
(Declare Path functions 700) +≡
  void dotlabel(Picture &picture, const short text_short = 0, const string position_string = "top")
  const;
```

880.

(Define Path functions 701) +≡

```cpp
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`.

[LDF 2003.01.31.] ?? Do I need to call `Point::project()` on the `Points` here and in `Path::project()`?

[LDF 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()`.

[LDF 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`.

```cpp
(Declare Path functions 700) +≡
  vector<Shape *> extract(const Focus &f, const unsigned short proj, real factor);
```
883.

\( \text{(Define Path functions 701)} \) \( \equiv \)

\begin{verbatim}
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) /* Path::extract() failed. LDF 2003.05.09. */
      return v;
  }
  vector(Shape *) r;
  r.push_back(this);
  return r;
}
\end{verbatim}

884. Set extremes. [LDF 2002.09.18] \( \text{set_extremes()} \) doesn’t check that the projective_coordinates in all of the Points on the Path are valid. This is done already in \( \text{project()} \) and \( \text{extract()} \), so I don’t think it’s necessary to repeat it here, since \( \text{extract()} \) (which invokes \( \text{project()} \)), is called before \( \text{set_extremes()} \) in Picture::output(). The latter is the only place where \( \text{set_extremes()} \) is invoked.


\( \text{(Declare Path functions 700)} \) \( \equiv \)

\begin{verbatim}
virtual bool set_extremes();
\end{verbatim}

885.

\( \text{(Define Path functions 701)} \) \( \equiv \)

\begin{verbatim}
bool Path::set_extremes()
{
  bool DEBUG = false; /* true */
  if (DEBUG)
    cout << "Entering Path::set_extremes()" << endl << flush; /* If there are no Points on
    the Path, set all the elements of projective_extremes to INVALID_REAL and return false. */
  if (points.size() == 0) {
    cerr << "ERROR! No Points on Path::set_extremes():
    Setting projective_extremes to INVALID_REAL and returning."
    << flush;
    projective_extremes = INVALID_REAL;
    return false;
  }
\end{verbatim}
§886. [LDF 2002.09.18.] Added this routine. Set the minimum values to \texttt{MAX\_REAL} and the maximum values to \texttt{\textemdash 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 \texttt{\textemdash MAX\_REAL}, I defined and used \texttt{\text{MIN\_REAL} = numeric\_limits(\texttt{real}) :: min()}. However, this isn’t the negative \texttt{real} with the largest magnitude, but the smallest positive \texttt{real}.

\begin{verbatim}
Define Path functions 701) \equiv 
projective_extremes[0] = MAX\_REAL; /* Minima. */
projective_extremes[2] = MAX\_REAL;
projective_extremes[4] = MAX\_REAL;
projective_extremes[1] = \texttt{\textemdash MAX\_REAL}; /* Maxima. */
projective_extremes[3] = \texttt{\textemdash MAX\_REAL};
projective_extremes[5] = MAX\_REAL;
for (vector\(\text{Point }*):: iterator iter = points\_begin(); iter \neq points\_end(); ++iter) {
  if (DEBUG) {
    cout << "min\_x\textunderscore u" << projective_extremes[0] << endl;
    cout << "max\_x\textunderscore u" << projective_extremes[1] << endl;
    cout << "x\textunderscore world\textunderscore u" << (**iter).\text{get\_x()} << endl;
    cout << "x\textunderscore persp\textunderscore u" << (**iter).\text{get\_x('p', false, false)} << endl;
    cout << "min\_y\textunderscore u" << projective_extremes[2] << endl;
    cout << "max\_y\textunderscore u" << projective_extremes[3] << endl;
    cout << "y\textunderscore world\textunderscore u" << (**iter).\text{get\_y()} << endl;
    cout << "y\textunderscore persp\textunderscore u" << (**iter).\text{get\_y('p', false, false)} << endl;
    cout << "min\_z\textunderscore u" << projective_extremes[4] << endl;
    cout << "max\_z\textunderscore u" << projective_extremes[5] << endl;
    cout << "z\textunderscore world\textunderscore u" << (**iter).\text{get\_z()} << endl;
    cout << "z\textunderscore persp\textunderscore u" << (**iter).\text{get\_z('p', false, false)} << endl;
  }
  projective_extremes[0] = min(projective_extremes[0], (**iter).\text{get\_x('p', false, false)}); /* Min x */
  projective_extremes[2] = min(projective_extremes[2], (**iter).\text{get\_y('p', false, false)}); /* Min y */
  projective_extremes[4] = min(projective_extremes[4], (**iter).\text{get\_z('p', false, false)}); /* Min z */
  projective_extremes[1] = max(projective_extremes[1], (**iter).\text{get\_x('p', false, false)}); /* Max x */
  projective_extremes[3] = max(projective_extremes[3], (**iter).\text{get\_y('p', false, false)}); /* Max y */
  projective_extremes[5] = max(projective_extremes[5], (**iter).\text{get\_z('p', false, false)}); /* Max z */
}
if (DEBUG) {
  for (int i = 0; i < 6; i++)
    cout << "projective\_extremes[" << i << "]\textunderscore u" << projective_extremes[i] << endl << flush;
}
\end{verbatim}
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 \( \equiv \) 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 \( z_p = z/(z + p) \).

\[
\text{Define Path functions 701}) + \equiv \\
\text{for (int i = 0; i < 6; i += 2) \{}
\text{if (projective_extremes[i] \equiv MAX_REAL} /* Minima */
\text{\lor projective_extremes[i + 1] \equiv -MAX_REAL) /* Maxima */}
\text{\{}
\text{if (DEBUG) \{}
\text{cout \ll "1_--_\n" \ll i \ll endl \ll flush;}
\text{cout \ll "projective_extremes[" \ll i \ll "]\-->_\n" \ll projective_extremes[i] \ll endl \ll flush;}
\text{\}}
\text{cerr \ll "ERROR! In Path::set_extremes();\n" \ll "maxima_and_minima_could_not_be_set_prop\nearly\n" \ll "Setting them all to INVALID_REAL and returning false.\n" \ll flush;}
\text{projective_extremes = INVALID_REAL;}
\text{return false;}
\text{break;}
\text{\}}
\text{if (DEBUG) cout \ll "Exiting Path::set_extremes()" \ll endl \ll flush;}
\text{return true; }
\]

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

\[
\text{Declare Path functions 700}) + \equiv \\
\text{virtual inline const valarray\( \langle \text{real}\rangle \) get_extremes() const}
\text{\{}
\text{return projective_extremes;}
\text{\}}
\]

889. Get minimum z. [LDF 2003.05.16.] Added this function.

\[
\text{Declare Path functions 700}) + \equiv \\
\text{virtual real get_minimum_z() const;}
\]
§890  3DLDF-1.1.5.1  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 << "Exiting Path::get_minimum_z()" << endl << flush;
  }
  return projective_extremes[4];
}

(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 << "Exiting Path::get_maximum_z()" << 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.

[Log]

[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 ≡ false) {
      if (DEBUG) cout ≡ "In Path::output()::do_output==false\Returning.\n" ≡ 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 761) +

if (points.size() == 0) {
    cerr << "THIS CAN’T HAPPEN! In Path::output():n"
         << "This code should never be reached.\n"
         << "However, I may be able to recover.\n"
         << "Type <RETURN> to continue.\n"
         << flush;
    getchar();
    cerr << "WARNING! In Path::output():n"
         << "Path doesn’t contain any Points.\n"
         << "Not doing anything.\n"
         << flush;
    return;
}
902.

Define Path functions 791) +

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 == DRAW) {
    if (DEBUG) cout << "Drawing.\n" << flush;
    if (arrow == true) out_stream << "drawarrow\n" << **point_iter++;
    else out_stream << "draw\n" << **point_iter++;
    (Output Path 907) /* [LD 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 != Colors::default_color) {
        out_stream << ".withcolor\n" << *draw_color;
    }

    if (dashed != "") out_stream << ".dashed\n" << dashed;
    if (pen != "") out_stream << ".withpen\n" << pen;
    out_stream << ";\n" << flush;
}

else if (fill_draw_value == FILL) {
    if (DEBUG) cout << "Filling.\n" << flush;
    out_stream << ".\n" << **point_iter++;
    (Output Path 907)

    if (fill_color != Colors::default_color) out_stream << ".withcolor\n" << *fill_color;
    out_stream << ";\n" << flush;
}

else if (fill_draw_value == FILLDRAW) {
    if (DEBUG) cout << "Filldrawing.\n" << flush;

    if (draw_color == fill_color) {
        out_stream << ".filldraw\n" << **point_iter++;
        (Output Path 907)

        if (draw_color != Colors::default_color) out_stream << ".withcolor\n" << *draw_color;
        if (dashed != "") out_stream << ".dashed\n" << dashed;
        if (pen != "") out_stream << ".withpen\n" << pen;
        out_stream << ";\n" << flush;
    }

    else /* We have two different colors, so we have to fill once and draw once. */
    {
        out_stream << "fill\n" << **point_iter++;
        (Output Path 907)

        out_stream << ".withcolor\n" << *fill_color;
        out_stream << ";\n" << flush;
        point_iter = points.begin();
    }

}
903.

[Log]

[Added the code in this section. It fixes a bug. If it’s not done, then the correct connectors are not used when the Path is output the second time.]

(Define *Path* functions 701) +⇒

```cpp
connector_iter = connectors.begin();
if (connectors.size() > 0) connector_string = *connector_iter++;
else connector_string = "--";
```

904.

(Define *Path* functions 701) +⇒

```cpp
if (arrow == true) out_stream << "drawarrow" << **point_iter++;
else out_stream << "draw" << **point_iter++;
```

```cpp
(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 (full_draw_value == UNDRAW)
   if (DEBUG) cout << "Undrawing.\n" << flush;
out_stream << "undraw", << **point_iter++;
```

```cpp
(Output *Path* 907)
if (dashed != ") out_stream << ",dashed," << dashed;
if (pen != ") out_stream << ",withpen," << pen;
out_stream << ";\n" << flush;
```

```cpp
else if (full_draw_value == UNFILL) {
if (DEBUG) cout << "Unfilling.\n" << flush;
out_stream << "unfill", << **point_iter++;
```

```cpp
(Output *Path* 907)
out_stream << ";\n" << flush;
}
905. Filldraw case.

Log

[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 ≪ "unfill\n" ≪ **point_iter ++ ;
         {Output Path 907}
         if (dashed ≠ "") out_stream ≪ ".dashed\n" ≪ dashed ;
         if (pen ≠ "") out_stream ≪ ".withpen\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 ≪ ".withcolor\n" ≪ *draw_color ;
   if (dashed ≠ "") out_stream ≪ ".dashed\n" ≪ dashed ;
   if (pen ≠ "") out_stream ≪ ".withpen\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} +≡
else /* Use DRAW as default. [LDF 2003.03.25.] */
{
    cerr << "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_Status::non_stop) getchar();
    #endif
    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;
    }
if (DEBUG) cout << "Exiting Path: output (Focus)" \"\n" \* flush;
    return;
}
907. When fillcolor and draw_color are different, this will have to be performed twice, so I’ve made it a named section.

Log

[LDF 2002.11.03] 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 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.

{Output Path 907} ≡

if (DEBUG) cout ≡ "Entering Output |Path| |n" ≡ flush;
for (unsigned short counter = 2; point_iter ≠ points.end(); ) {
  out_stream ≡ "," ≡ connector_string ≡ "," ≡ ++point_iter++;
  /* This breaks the line and indents after two points */
  if (counter ≡ 2 ∧ point_iter ≠ points.end()) {
    out_stream ≡ ",";
    counter = 1;
  }
  else {
    ++counter;
  }
  if (connector_iter ≠ connectors.end()) connector_string = *connector_iter++;
}
if (is_cycle()) out_stream ≡ "," ≡ connector_string ≡ ",cycle";
if (DEBUG) cout ≡ "Exiting Output |Path| |n" ≡ flush;
This code is used in sections 902, 904, 905, and 906.

908. Showing.

909. Show.

Log

[LDF 2003.07.13] Commented-out the line that prints fill_draw_value to stdout.
[LDF 2003.08.20] Now printing points.size () and connectors.size () to stdout. If the latter is 0, a message is printed, that "--" will be used as the connector.

(Declare Path functions 790) +=

void show(string text = "", char coords = 'y', const bool do_persp = true, const bool
do_apply = true, Focus *f = 0, const unsigned short proj = Projections::PERSP, const real
dpi_factor = 1) const;
910.
(Define Path functions 701) +v
  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;
#if 0
    cout << "fill_draw_value,--", 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 {
        cerr << "WARNING! In |show():" << "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 << "connectors.size(),--", connectors.size() << endl << flush;
    cout << "connectors.size(),--", connectors.size() << endl << flush;
    if (connectors.size() == 0) {
      cout << "Using,--", connector_string << 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 << "\n" << connector_string << "\n";
      }
      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 << "\n" << connector_string << "\n" 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 = "draw_color";
    else cout << "draw_color == 0.\n";
    if (fillcolor != 0) fillcolor = "fill_color";
    else cout << "fill_color == 0.\n";
    if (stop) getchar();
}

913. Returning elements and information.

914. Is on free store.

915. (Declare Path functions 701) +≡
bool is_on_free_store() const;

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.

Log
[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.

917. (Declare Path functions 700) +≡
virtual bool is_planar(bool verbose = false, string text = "") const;
917. (Define Path functions 701) +≡

```cpp
bool Path::is_planar(bool verbose, string text) const
{
    bool DEBUG = false;  /* true */
    if (VERBOSE_GLOBAL) verbose = true;
    if (DEBUG || verbose) cout ≈ "Entering Path::is_planar() \n";
    Point p(get_normal());
    if (p == INVALID_POINT) {
        if (DEBUG) cout ≈ "Exiting Path::is_planar() \nReturning false. \n" ≈ flush;
        if (verbose) {
            if (text == "") text = "Path";
            cout ≈ text ≈ " is non-planar. \n";
        }
        return false;
    } else if (p == origin) {
        cerr ≈ "WARNING! In Path::is_planar(): \n" ≈ "Path is linear. Returning false. \n" ≈ flush;
        if (verbose) {
            if (text == "") text = "Path";
            cout ≈ text ≈ " is planar. \n";
        }
        return true;
    } else {
        if (verbose) {
            if (text == "") text = "Path";
            cout ≈ text ≈ " is planar. \n";
        }
        if (DEBUG) cout ≈ "Exiting Path::is_planar() \nReturning true. \n" ≈ flush;
        return true;
    }
}
```

**Log**

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_normal() 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_normal(). !! 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 2003.06.14] Made verbose non-const. Setting it to true if VERBOSE_GLOBAL is true. Added VERBOSE_GLOBAL to pspglb.web today.

(Declare Path functions 700) +≡

```cpp
bool is_linear(bool verbose = false, string text = "") const;
```
919.  
(Define Path functions 701) +≡

```cpp
bool Path::is_linear(bool verbose, string text) const
{
    bool DEBUG = false;      // true *
    if (VERBOSE_GLOBAL)    verbose = true;
    if (DEBUG || verbose) cout << "Entering Path::is_linear() \n";
    if (line_switch)       /* LDF 2003.04.09. Added this conditional. */
        return true;
    Point p(get_normal());
    if (p == origin) {
        if (verbose) {
            if (text == "") text = "Path";
            cout << text << " is_linear \n";
        }
        if (DEBUG) cout << "Exiting Path::is_linear() \n" << "Returning true \n" << flush;
        return true;
    }
    else {
        if (verbose) {
            if (text == "") text = "Path";
            cout << text << " is_non-linear \n";
        }
        if (DEBUG) cout << "Exiting Path::is_linear() \n" << "Returning false \n" << flush;
        return false;
    }
}
```

920. Get line switch. [LDF 2002.11.03] This function returns true for Paths that are created or set using two Points only, and no connectors, as arguments.

```cpp
inline bool get_line_switch() const
{
    return line_switch;
}
```

921. Test for cycles.

(Declare Path functions 700) +≡

```cpp
inline bool is_cycle() const
{
    return cycle_switch;
}
```

922. Size (number of points).

(Declare Path functions 700) +≡
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 \( P \) is a Path \( p \) and \( q \) 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 \( q \) and \( x \) and \( y \) are the x and y-coordinates of points on \( q \).

[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.

\[\text{Declare Path functions 790}\] +\=
real slope(char \( a = \)'x', char \( b = \)'y');

924.

\[\text{Define Path functions 791}\] +\=
real Path::slope(char \( a, \) char \( b)\)
{
    if (\!(is_linear)) {
        cerr \ll \"ERROR! Path::slope().Path.is_linear!\n" \ll
            "Returning INVALID_REAL\n" \ll flush;
        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.

\[\text{Declare Path functions 790}\] +\=
Path subpath(size_t \( \text{start}, \text{end} \), const bool cycle = false, const string connector = "") const;
926. (Define Path functions 701) +≡
 Path Path : subclass(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! In Path::subpath():\n" << "The\"start\"argument is\<\, the\"end\"\n argument.\n" << "Returning an 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! In Path::subpath():\n" << "Will try to recover by setting start=0.\n\n" << flush;
     start = 0;
 }
 if (end > points.size()) {
     cerr << "ERROR! In Path::subpath():\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++) {
         cerr << "ERROR! In Path::subpath():\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 2002.11.06.] If no <em>connector</em> argument is specified, then we have to get the appropriate connectors from *<em>this</em>. This is slightly tricky, because connectors doesn't have to contain a <em>connector</em> for each pair of Points that is joined in a Path. So, first we must fill up <em>p</em>.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) +≡

<em>p</em> = *<em>this</em>;

<tt>unsigned short a = points.size();<br>if (~cycle) a -= 1;</tt>

<tt>int i;<br>string s = connectors.back();</tt>

<tt>for (i = connectors.size(); i &lt; a; i++) {<br>    p.connectors.push_back(s);<br>}</tt>

<tt>if (DEBUG) cout &lt;&lt; "p.connectors.size() = " &lt;&lt; p.connectors.size() &lt;&lt; endl &lt;&lt; flush;</tt>

<tt>Path q; for (size_t i = start; i &lt; end; i++) { q.points.push_back (create_new &lt; Point &gt; (p.points[i])<br>    );</tt>

<tt>if (i &lt; p.connectors.size()) q.connectors.push_back(p.connectors[i]);<br>    q.set_cycle(cycle);<br>return q;}</tt>

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) +≡

<tt>const Point &get_point(const unsigned short a);</tt>
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 is > size of Path.\n" << "Returning INVALID_POINT.\n" << flush;
            return INVALID_POINT;
        }
    }

934.  const version.

---------------------------------

[Log]
[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 is > size of Path.\n" << "Returning INVALID_POINT.\n" << flush;
            return INVALID_POINT;
        }
    }

936. Get last point.

---------------------------------

[Log]
[LDF 2002.05.10] 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) +≡
   \newcommand{\Path}{\text{Path}}
   \begin{verbatim}
   const Point &\Path::get_last_point() const
   {
     if (points.size() \neq 0) return *points[points.size() - 1];
     else {
       \cerr << "\text{ERROR! In }\Path::get_last_point():\n" << 
        "\text{Path is empty.}\n" << 
        "Returning \text{INVALID\_POINT.}\n\" << \flush;
       return \text{INVALID\_POINT};
     }
   }
\end{verbatim}

938. Get size.
(Declare Path functions 700) +≡
   \begin{verbatim}
   virtual inline size_t \Path::get_size() const
   {
     return points.size();
   }
\end{verbatim}

939. Get normal.
940. Path version. [LDF 2002.11.05.] \texttt{get\_normal()} returns a unit vector representing the normal to the plane of the Path \textit{*this}, if \textit{*this} is planar. If the Points on \textit{*this} are collinear and there are no connectors that could make the Path non-planar, then \textit{origin} \((0, 0, 0)\) is returned. If the Path is neither planar nor linear, \texttt{get\_normal()} returns \texttt{INVALID\_POINT}.

[LDF 2002.11.05.]
- \texttt{get\_normal()} first checks whether a Path contains no Points or only one Point. If so, \texttt{get\_normal()} returns \texttt{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 \texttt{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, \texttt{get\_normal()} returns the Point \((0, 0, 0)\), because the Path will be linear.
- Then it gets the cross product \(b_0 \) of \(p_1^2\) and \(p_0^2\), where \(p_0\) and \(p_1\) 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 collinear, so \texttt{get\_normal()} returns \texttt{origin}.
- If, however, \(b_0 \neq (0, 0, 0)\) exists, then cross products \(b_x\) are calculated using \(p_0^2\) and the direction vectors \(p_{p_2}\) for the rest of the Points \(p_x\) on the Path. If and only if \(b_x = b_0\) or \(b_x = -b_0\) or \(b_x = (0, 0, 0)\) for all \(b_x\), then the Path is planar, and \texttt{get\_normal()} returns \(-b_0\) (see explanation of sign below). Otherwise, the Path is non-planar, and \texttt{get\_normal()} returns \texttt{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 Huw Jones, \textit{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

[254] Path version. [LDF 2002.11.05.] Rewrote this function.
[254] Path version. [LDF 2003.06.04.] Changed sign of the normal, when it's returned, in the cases where a proper normal is found (not \texttt{INVALID\_POINT} or \texttt{origin}). See explanation above.

---

(Declare \texttt{Path} functions 700) \(\equiv\)
\texttt{virtual Point get\_normal()} \texttt{const;}

941.
(Define \texttt{Path} functions 701) \(\equiv\)
\begin{verbatim}
Point Path::get_normal() const { bool DEBUG = false; /* true */
    if (DEBUG) cout << "Entering Path::get_normal().\n" << flush;
    if (points.size() \(\leq\) 0) {
        cerr << "WARNING! In Path::get_normal():\n" "Path is empty or contains only one Point.\n" << "Returning INVALID\_POINT.\n\n" << flush;
        return INVALID\_POINT;
    }
\end{verbatim}
§942. [LDF 2002.11.06.] 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 in a plane.

```cpp
for (vector<string>::const_iterator iter = connectors.begin(); iter != connectors.end(); ++iter) {
    if (!(iter == ".." || iter == "--" || iter == "..." || iter == "----")) {
        cerr << "WARNING! In Path: get_normal():n" << "Connector may make Path non-linear, or non-planar:"
        endl << "Returning INVALID_POINT.\n\n" << flush;
        return INVALID_POINT;
    }
}
```

§943. [LDF 2002.11.06.] Two points determine a line.

```cpp
if (points.size() == 2) {
    cerr << "WARNING! In Path: get_normal():n" << "Path has 2 Points. Returning origin.\n\n" << flush;
    return origin;
}
```
944.

{Define Path functions 701} +≡
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 ≡ origin ∧ iter ≠ points.end()) {
    p2 = **iter ++;
    a1 = p2 - p0;
    b0 = a0.cross_product(a1);
}
if (iter ≡ points.end() ∧ b0 ≡ origin) {
    if (DEBUG) cout ≡ "Exiting, Path::get_normal().u" ≡ "Points are all colinear,\n" ≡ "Returning, origin.\n\n" ≡ flush;
    return origin;
}
if (DEBUG) b0.show("b0");
b0.unit_vector(true);
if (iter ≡ points.end() ∧ b0 ≠ origin) {
    if (DEBUG) {
        cout ≡ "Exiting, Path::get_normal().u" ≡ "Points are all colinear, except for one.\n" ≡ "Returning, normal.\n\n" ≡ flush;
    }
    return -b0;
}
Point b1;
if (DEBUG) cout ≡ "Entering, second while.\n\n" ≡ flush;
while (iter ≠ points.end()) {
    p2 = **iter ++;
    a1 = p2 - p0;
    b1 = a0.cross_product(a1);
    if (b1 ≡ origin)
        /* [LDF 2002.11.03] This if merely prevents a warning from being issued by unit_vector(). */
        b1.unit_vector(true);
    if (DEBUG) b1.show("b1");
    if (¬(b1 ≡ origin ∨ b1 ≡ b0 ∨ b1 ≡ -b0)) {
        if (DEBUG)
            cout ≡ "Exiting, Path::get_normal().u" ≡ "Returning, INVALID_POINT.\n\n" ≡ flush;
            return INVALID_POINT;
    }
}
if (DEBUG) cout ≡ "Exiting, Path::get_normal().u" ≡ "Returning, normal.\n\n" ≡ flush;
return -b0; }
945. **Point version.** Point::get_normal() is declared points.web, but it must be defined here, because it calls Path::get_normal(). [LDF 2003.07.11.]

(Log) Added this function.

(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():\n" <<
            "The Points do not determine a plane.\n" << "Returning INVALID_POINT.\n" << flush;
        return INVALID_POINT;
    }
    else return r.get_normal();
}

946. **Get plane.**

(Log) Rewrote this function to correspond to the new definition of get_normal().

(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! Path::get_plane().\n" << "Path::is_not_a Plane.\nReturning INVALID_PLANE.\n\n" << flush;
       return INVALID_PLANE;
     }
     Point point(get_point(0));
     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.

   Log

   [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 pspglb.web today.

(Define Point functions 330) +≡
   bool Point::is_in_triangle(const Point &p0, const Point &p1, const Point &p2, bool verbose)
   {
     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! Point::is_in_triangle():\n" << "The arguments do not determine a plane.\n" << "Returning false.\n\n" << flush;
       }
       return false;
     }
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 t_x = t.world_coordinates[0];
real t_y = t.world_coordinates[1];
real t_z = t.world_coordinates[2];
real c_x = c.world_coordinates[0];
real c_y = c.world_coordinates[1];
real c_z = c.world_coordinates[2];
real d_x = d.world_coordinates[0];
real d_y = d.world_coordinates[1];
real d_z = d.world_coordinates[2];
real e_x = e.world_coordinates[0];
real e_y = e.world_coordinates[1];
real e_z = e.world_coordinates[2];
real lambda_denominator = (((e_x - c_x) * (d_y - c_y)) - ((e_y - c_y) * (d_x - c_x)));
real mu_denominator = (((e_x - c_x) * (d_y - c_y)) - ((e_y - c_y) * (d_x - c_x)));
bool exchange_yz = false;
bool exchange_xz = false; if (lambda_denominator \equiv 0 \lor mu_denominator \equiv 0) {
    if (DEBUG) cout << "lambda_denominator or mu_denominator \equiv 0.\n" <<
    "Exchanging y and z-coordinates. \n" << flush;
    real temp;
temp = t_y;
t_y = t_x;
t_x = temp;
temp = c_y;
c_y = c_z;
c_z = temp;
temp = d_y;
d_y = d_z;
d_z = temp;
temp = e_y;
e_y = e_z;
e_z = 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 == 0 \lor mu_denominator == 0) {
    if (DEBUG) cout << "Exchanging \( y \) 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 \( y \) and \( z \)-coordinates didn't work.\n" << "Exchanging \( x \) and \( z \)-coordinates.\n" << flush;

949. First, put things back the way they were. It's wasteful, but less confusing. [LDF 2003.06.24]
\end{verbatim}

\begin{verbatim}
(Define \textbf{Point} functions 330) +\!
\begin{align*}
temp &= t_y; \\
t_y &= t_z; \\
t_z &= temp; \\
temp &= c_y; \\
c_y &= c_z; \\
c_z &= temp; \\
temp &= d_y; \\
d_y &= d_z; \\
d_z &= temp; \\
temp &= e_y; \\
e_y &= e_z; \\
e_z &= temp;
\end{align*}
\end{verbatim}
950. Now, exchange the x and z-coordinates. [LDF 2003.06.21.]

(Define \textbf{Point} functions 330) \equiv

\begin{verbatim}
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;

\textbf{lambda} \_\text{denominator} = \left( (e_x - c_x) * (d_y - c_y) - (e_y - c_y) * (d_x - c_x) \right) /
\textbf{mu} \_\text{denominator} = \left( (e_x - c_x) * (d_y - c_y) - (e_y - c_y) * (d_x - c_x) \right);
\end{verbatim}

if \((-\text{lambda} \_\text{denominator} = 0 \lor \text{mu} \_\text{denominator} = 0)) \{
  if (DEBUG) cout \ll "Exchanging \textit{x} and \textit{z}-coordinates worked. \n" \ll "\textbf{lambda} \_\text{denominator}, and \textbf{mu} \_\text{denominator}, are no longer 0. \n" \ll flush;
  exchange \_\textit{w} = \textit{true};
\}
else \{
  if (verbose \lor DEBUG) {
    cerr \ll "\textbf{WARNING!}, \textbf{InPoint::is_in_triangle()}\n" \ll "\textbf{lambda} \_\text{denominator}, or \textbf{mu} \_\text{denominator}, is 0. \n" \ll "Returning, false. \n\n" \ll flush;
  }
  return false;
\}

\textbf{real} lambda = \left( (t_x - c_x) * (d_y - c_y) - (t_y - c_y) * (d_x - c_x) \right) /
\textbf{real} mu = \left( (t_x - c_x) * (e_y - c_y) - (t_y - c_y) * (e_x - c_x) \right) /
\textbf{lambda} \_\text{denominator};
\}

if (DEBUG) {
  cout \ll "\textbf{lambda}, \textit{\rightarrow}, \textit{\n}" \ll lambda \ll endl \ll flush;
  cout \ll "\textbf{mu}, \textit{\rightarrow}, \textit{\n}" \ll mu \ll endl \ll flush;
  cout \ll "\textbf{lambda}, \textbf{mu}, \textit{\rightarrow}, \textit{\n}" \ll lambda + mu \ll endl \ll flush;
  cout \ll "\textit{\n}\textbf{(lambda}, >, 0, \&\&\textbf{mu}, >, 0, \&\&\textbf{(lambda}, +, \textbf{mu}, \rightarrow, 1)\textit{\rightarrow}, \textit{\n}}\ll (lambda \geq 0 \land mu \geq 0 \land ((lambda + mu) \leq 1)) \ll endl \ll flush;
\}

\textbf{bool} b = (lambda \geq 0 \land mu \geq 0 \land ((lambda + mu) \leq 1));
if (verbose) {
  cout \ll "\textbf{InPoint::is_in_triangle:}\n";
  if (b) cout \ll "The, \textit{\n}, \textbf{Point}, \textit{\n}, is, within, the, \textbf{triangle}\n." \ll "Returning, true.\n";
  else cout \ll "The, \textbf{Point}, \textbf{\n}, doesn't, lie, within, the, \textbf{triangle}\n." \ll "Returning, false.\n";
  cout \ll endl \ll flush;
}
return b; }

951. Manipulating Paths.

952. Set cycle.

---

[LDF 2002.4.7.] Changed, so that the argument \textbf{bool} c is \textit{true} by default.
Made bool c argument const.

```cpp
(Declare Path functions 700) \equiv
void set_cycle(const bool c = true);
```

**953.**

(Define Path functions 701) \equiv

```cpp
void Path::set_cycle(const bool c)
{
    cycle_switch = c;
}
```

**954. Reverse.**

**955. With assignment.**

[LDF 2002.11.05.] Added this function.
[LDF 2003.07.16.] Added error handling code for the case that this function is called with assign \equiv false. I've now added a const version, so there's no need to call this version with assign \equiv false. If assign is false, the const version is called, so I could simplify the code in this version.

```cpp
(Declare Path functions 700) \equiv
Path reverse(bool assign);
```

**956.**

(Define Path functions 701) \equiv

```cpp
Path Path::reverse(bool assign){ bool DEBUG = false; /* true */
    if (is_cycle()) /* Return *this if *this is a cycle. */
    {
        cerr \ll "ERROR\nIn Path::reverse()\n\n" \ll "*this is a cycle\nCan’t reverse\n\n" \ll 
    "Returning *this\n\n";
    return *this;
    }
    if (!assign) {
        cerr \ll "WARNING\nIn Path::reverse(bool)\n\n" \ll "assign \equiv false\n\n" \ll 
    "Use reverse() without an argument instead\n\n" \ll "Calling reverse(void)\n\n" \ll 
    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 () \neq points.size () - 1) {
    string lastConnector;
    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 \ll \"Reversing, connectors, and, points.\n\" \ll flush;
std::reverse (connectors.begin (), connectors.end ());
std::reverse (points.begin (), points.end ());
if (DEBUG) {
    cout \ll \"Showing, connectors:\n\";
    for (vector(string)::iterator iter = connectors.begin (); iter \neq connectors.end (); iter ++)
        cout \ll \*iter \ll endl;
    cout \ll \"Showing, points:\n\";
    for (vector(Point \*:::iterator iter = points.begin (); iter \neq 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
[Log 2003.07.16.] Added this function.

(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_first_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} +≡

\begin{verbatim}
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." << endl << "flush;" << flush;
    return INVALID_BOOL_POINT;
  }
}
\end{verbatim}

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.

\begin{verbatim}
bool_point Plane :: intersection_point(const Path &p) const
{
  if (p.is_linear()) return intersection_point(p.get_point(0), p.get_last_point());
  else {
    cerr << "ERROR! Plane::intersection_point(const Path&):" << endl << "Path is not linear! Returning INVALID_BOOL_POINT." << endl << "flush;" << flush;
    return INVALID_BOOL_POINT;
  }
}
\end{verbatim}

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 \(x \leq 0\), 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.

\begin{verbatim}
\end{verbatim}

\[LDF 2003.02.05\] 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 (dashed) and pen (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 shift_x, shift_y, and 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 dashed and pen const in both versions.

---

968. Length argument first.

```c
void draw_axes(real dist = 2.5, string pos_x = "bot", string pos_y = "lft", string pos_z = "bot", const Color &ddraw_color = *Colors::default_color, const string dashed = ",", const string pen = ",", const Point &shift_x = origin, const Point &shift_y = origin, const Point &shift_z = origin, Picture &picture = current_picture);
```

See also section 973.
This code is used in section 981.

969. 

```c
void draw_axes(real dist, string pos_x, string pos_y, string pos_z, const Color &ddraw_color, const string dashed, const string pen, const Point &shift_x, const Point &shift_y, const Point &shift_z, Picture &picture)
```

See also sections 970, 971, 972, and 974.
This code is used in section 986.

970. Remember to change this if you change any of the defaults!

---

[LDF 2003.04.01.] Added this section.

(Define draw_axes() 969) +
```c
if (dist <= 0) dist = 2.5;
if (pos_x == "d") pos_x = "bot";
if (pos_y == "d") pos_y = "lft";
if (pos_z == "d") pos_z = "bot";
```

971. 

---

[LDF 2003.04.01.] Added this error handling code.

(Define draw_axes() 969) +
```c
if (pos_x == "n" && pos_y == "n" && pos_z == "n") {
  cerr << "WARNING! In draw_axes(): " << endl << "All axes are suppressed. Returning." << endl << endl << flush;
  return;
}
```
972.

\[ \text{Define } \texttt{draw_axes()} \text{ 969} \] +
\[
\text{if } (\texttt{pos}_x \neq "\text{n}") \{ \\
\quad \texttt{Point } x0(0,-\text{dist}); \\
\quad \texttt{Point } x1(\text{dist}); \\
\quad x0.\texttt{drawarrow}(x1, \texttt{ddraw_color}, \texttt{ddashed}, \texttt{ppen}, \texttt{picture}); \\
\quad x1 += \texttt{shift}_x; \\
\quad x1.\texttt{label("x", pos}_x, \text{false}, \text{picture}); \\
\}
\]
\[
\text{if } (\texttt{pos}_y \neq "\text{n}") \{ \\
\quad \texttt{Point } y0(0,0,-\text{dist}); \\
\quad \texttt{Point } y1(\text{dist}); \\
\quad y0.\texttt{drawarrow}(y1, \texttt{ddraw_color}, \texttt{ddashed}, \texttt{ppen}, \texttt{picture}); \\
\quad y1 += \texttt{shift}_y; \\
\quad y1.\texttt{label("y", pos}_y, \text{false}, \text{picture}); \\
\}
\]
\[
\text{if } (\texttt{pos}_z \neq "\text{n}") \{ \\
\quad \texttt{Point } z0(0,0,0,-\text{dist}); \\
\quad \texttt{Point } z1(\text{dist}); \\
\quad z0.\texttt{drawarrow}(z1, \texttt{ddraw_color}, \texttt{ddashed}, \texttt{ppen}, \texttt{picture}); \\
\quad z1 += \texttt{shift}_z; \\
\quad z1.\texttt{label("z", pos}_z, \text{false}, \text{picture}); \\
\}
\]
\text{return;} 

973. Color argument first.

\[ \text{Log} \]

[\text{LDF 2003.06.02.} \text{ Added this function.} ]

\[ \text{Declare } \texttt{draw_axes()} \text{ 968} \] +
\[
\text{void } \texttt{draw_axes(const Color &\texttt{ddraw_color}, \text{real dist} = 2.5, \text{string pos}_x = "\text{bot}", \text{string pos}_y = "\text{ft}", \text{string pos}_z = "\text{bot}", \text{const string dashed} = "\text{"}, \text{const string ppen} = "\text{"}, \text{const Point &\texttt{shift}_x = \texttt{origin}, \text{const Point &\texttt{shift}_y = \texttt{origin}, \text{const Point &\texttt{shift}_z = \texttt{origin, Picture \&picture = \texttt{current, picture};} } }\]
974.
(Define \texttt{draw\_axes} (969) \equiv
\begin{verbatim}
void draw_axes(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)
{
    draw_axes(dist, pos_x, pos_y, pos_z, draw_color, dashed, pen, shift_x, shift_y, shift_z, picture);
}
\end{verbatim}
)

975. Paths and Lines. \[\text{Log}\]

\[\text{Add this heading.}\]

976. Get Line. Returns a Line corresponding to \texttt{*this}, if \texttt{*this} is linear. Otherwise, \texttt{get\_line()} returns INVALID\_LINE.

\[\text{Log}\]

\[\text{Added this function.}\]

977.
(Declare \texttt{Path} functions 700) \equiv
\begin{verbatim}
Line get_line(void) const;
\end{verbatim}

978. Get Path. Declared in \texttt{lines.web}. Must be defined here, because \texttt{Path} is an incomplete type there.

\[\text{Log}\]

\[\text{Added this function.}\]

979. Putting Path together.
§980. 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).

[LDI 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.
[LDI 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

(Version control identifier 5) +≡
static string res_id = "$Id: curves.web,v.1.5,2004/01/12,21:27:59,ifinsto1,Exp$";

983. Include files.

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

[LDI 2002.11.12] Changed the name "Regular_Closed_Plane_Curve" to "Reg_C1_Plane_Curve", because the former caused too many "Overfull boxes" when running cweave.

985. Reg_C1_Plane_Curve class definition. A Reg_C1_Plane_Curve is assumed to be closed, planar, convex, and have at least 3 points. The functions that create and modify Reg_C1_Plane_Curves must ensure that these assumptions are correct!
[LDI 2002.11.05] Reg_C1_Plane_Curve is intended to be used as a base class. No objects of type Reg_C1_Plane_Curve should be defined, however, it is not an abstract class, so it is possible to do so.

format Reg_C1_Plane_Curve Curve

(Define class Reg_C1_Plane_Curve 985) ≡
class Reg_C1_Plane_Curve : public Path {
p
protected: Point center;
    unsigned short number_of_points;
p
public: (Declare Reg_C1_Plane_Curve functions 987)
\section*{986. Returning elements and information.} [LDF 2002.11.05.] The \texttt{virtual} functions in this section are meant to be overloaded by member functions of types derived from \texttt{Reg_Cl_Plane_Curve}.

\begin{verbatim}
[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.}
(Declare \texttt{Reg_Cl_Plane_Curve} functions 987) \equiv
\begin{verbatim}
inline virtual bool is_quadratic() const
{
    return false;
}
\end{verbatim}
See also sections 988, 989, 990, 991, 992, 994, 997, 1008, 1011, 1013, and 1014.
This code is used in section 985.

\section*{988. Is cubic.}
(Declare \texttt{Reg_Cl_Plane_Curve} functions 987) \equiv
\begin{verbatim}
inline virtual bool is_cubic() const
{
    return false;
}
\end{verbatim}

\section*{989. Is quartic.}
(Declare \texttt{Reg_Cl_Plane_Curve} functions 987) \equiv
\begin{verbatim}
inline virtual bool is_quartic() const
{
    return false;
}
\end{verbatim}

\section*{990. Get coefficients.}
(Declare \texttt{Reg_Cl_Plane_Curve} functions 987) \equiv
\begin{verbatim}
inline virtual real_triple get_coefficients(real, real) const
{
    return real_triple(INVALID_REAL, INVALID_REAL, INVALID_REAL);
}
\end{verbatim}

\section*{991. Solve.} [LDF 2002.11.05.] This \texttt{virtual} function is meant to be overloaded by member functions of types derived from \texttt{Reg_Cl_Plane_Curve}.
(Declare \texttt{Reg_Cl_Plane_Curve} functions 987) \equiv
\begin{verbatim}
inline virtual pair(real, real) solve(char, real) const
{
    return pair(real, real)(INVALID_REAL, INVALID_REAL);
}
\end{verbatim}
992. **Location of a point.** `location()` returns a signed short indicating the location of its Point argument with respect to the `Reg_Cl_Plane_Curve`.

[LDF 2002.11.05.] TO DO: Currently, the programmer must ensure that a `Reg_Cl_Plane_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_Cl_Plane_Curve` to become non-planar.

[LDF 2002.11.05.] The number of Points in a `Reg_Cl_Plane_Curve` must be a multiple of 4, and that the Point number of Points / 4 must be at 90° to Point 0. Also, refpt can’t be Point 0.

[LDF 2003.07.16.] `Reg_Cl_Plane_Curve` now has a data member named `center`. However, a `Reg_Cl_Plane_Curve` need not have a meaningful center. Usually, when an object of a class derived from `Reg_Cl_Plane_Curve` calls this function, its `center` will be passed as the `refpt` argument. However, this need not be the case.

TO DO: Check whether it will work if pt0.x < 0. I think it should.

[LDF 2003.06.14.] !! CHECK. Bug, when `Reg_Cl_Plane_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_Cl_Plane_Curve`.
0 The `Point` lies on the perimeter of the `Reg_Cl_Plane_Curve`.
1 The `Point` lies inside the perimeter of the `Reg_Cl_Plane_Curve`.

These values are returned in cases where errors have occurred:

-2 The `Point` is not in the same plane as the `Reg_Cl_Plane_Curve`.
-3 Something has gone terribly wrong.
-4 The normal to the `Reg_Cl_Plane_Curve` has 0 magnitude, i.e., the Points on the `Reg_Cl_Plane_Curve` are collinear.
-5 An error occurred in putting the `Reg_Cl_Plane_Curve` in one of the major planes.
-6 The `Reg_Cl_Plane_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 “\let” 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 rθ 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 `supress_warnings`.

[LDF 2003.07.16.] Changed name of `center` argument to `refpt`, because I’ve made `center` a data member of `Reg_Cl_Plane_Curve`.

(Declare `Reg_Cl_Plane_Curve` functions 987) +

```cpp
   virtual signed short location(Point refpt, Point pt0, const bool suppress_warnings = false) const;
```
993.
(Define Reg_Cl_Plane_Curve functions 993)
signed short Reg_Cl_Plane_Curve::location(Point ref_pt, Point pt0, const bool suppress_warnings)
{
  bool DEBUG = false; /* true */
  if (DEBUG) {
    cout << "Entering Reg_Cl_Plane_Curve::location()\n";
    ref_pt.show("ref_pt");
    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(getPlane());
  if (pl == INVALID_PLANE) /* LDF 2003.06.14. Added this error handling code. */
  {
    cerr << "ERROR! In Reg_Cl_Plane_Curve::location():\n" <<
      "The Reg_Cl_Plane_Curve is non-planar.\n" << "Returning -6\n" << flush;
    return -6;
  }

  real r0 = pl.getDistance(pt0).first;
  if (r0 > Point::epsilon())
  /* LDF 2003.06.14. Changed r0 will always be positive, so I now longer take its absolute value. */
  {
    if (!suppress_warnings) cerr << "WARNING! In Reg_Cl_Plane_Curve::location():\n" <<
      "Point u is not in plane of regular closed plane curve.\n" <<
      "Returning -2\n" << flush;
    return -2;
  }

  Reg_Cl_Plane_Curve copy(*this);
  if (ref_pt != origin) pt0 == ref_pt == copy.shift(-ref_pt); /* LDF 2002.11.05. Simplified. */

  Point copy_normal = copy.get_normal();
  if (DEBUG) copy_normal.show("copy_normal");
  if (copy_normal.magnitude() == 0) {
    cerr << "ERROR! In Reg_Cl_Plane_Curve::location():\n" <<
      "Normal has no magnitude.\nReturning -4\n" << flush;
    return -4;
  }

  else if (fabs(copy_normal.getx()) < Point::epsilon() &&
    fabs(copy_normal.gety()) < Point::epsilon())
  {
    if (DEBUG) cout << "Regular closed plane curve is already in x-y plane.\n";
    orientation = X_Y;
  }
  else if (fabs(copy_normal.getz()) < Point::epsilon())
  {
    if (DEBUG) cout << "Regular closed plane curve is already in x-z plane.\n";
    orientation = X_Z;
  }
else if (fabs(copy_normal.get_z()) < Point::epsilon() \&\& fabs(copy_normal.get_y()) < Point::epsilon())
{
    if (DEBUG) cout << "Regular, closed plane, curve is already in z-y 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 pt14 (copy.get_point(number_of_points/4));
    Transform t2;
    t2.align_with_axis(ref_pt, pt14, 'z');
    copy *= ref_pt *= pt0 *= t2;
    orientation = X_Z;
}
if (DEBUG) {
    cout << "orientation=-\n" << orientation << endl << flush;
    copy.draw(Colors::blue);
    copy.show("copy");
    ref_pt.dotlabel("C");
    pt0.dotlabel("pt0");
    pt0.show("pt0");
}
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 \equiv X_Y \lor orientation \equiv X_Z) {
    if (pt1 \neq x_axis \&\& pt1 \neq -x_axis) {
        t3.align_with_axis(ref_pt, pt1, 'x');
        copy *= ref_pt *= pt0 *= t3;
    }
}
else if (orientation \equiv Z_Y) {
    if (pt1 \neq z_axis \&\& pt1 \neq -z_axis) {
        t3.align_with_axis(ref_pt, pt1, 'z');
        copy *= t3;
        ref_pt *= t3;
        pt0 *= t3;
    }
}
real_pair rr;
real pt0_v; /* [LDF 2002.11.05.] Vertical. */
real pt0_h; /* [LDF 2002.11.05.] Horizontal. */
if (orientation \equiv X_Y) {
    pt0_h = pt0.get_x();
    pt0_v = pt0.get_y();
}
```c
if (orientation \equiv \mathbf{X}_Z) {
    pt0_x = pt0 \cdot \text{get}_x();
    pt0_z = pt0 \cdot \text{get}_z();
}
else if (orientation \equiv \mathbf{Z}_Y) {
    pt0_x = pt0 \cdot \text{get}_x();
    pt0_z = pt0 \cdot \text{get}_y();
}
else {
    \text{cerr} \ll "ERROR! In Reg Cl Plane Curve::location() \n" \ll "orientation has invalid value: \n" \ll "Returning -5 \n" \ll \text{flush};
    return -5;
}
rr = solve('y', pt0_y); /* LDF 2002.11.05. TO DO: Explain. */
if (rr.first \equiv \text{INVALID-REAL} \land rr.second \equiv \text{INVALID-REAL}) {
    if (DEBUG) {
        cout \ll "Point is outside regular closed plane curve. \n" \ll "Returning -1 \n";
        cout \ll "Exiting Reg Cl Plane Curve::location() \n";
    }
    return -1;
}
else if ((\text{fabs}(\text{fabs}(pt0_y) - \text{fabs}(rr.first)) < \text{Point \cdot epsilon()} \lor (\text{fabs}(\text{fabs}(pt0_y) - \text{fabs}(rr.second)) < \text{Point \cdot epsilon()})) {
    if (DEBUG) {
        cout \ll "Point is on regular closed plane curve. \n" \ll "Returning 0 \n" \ll "Exiting Reg Cl Plane Curve::location() \n";
        getchar();
    }
    return 0;
}
else if (\text{fabs}(pt0_y) < \text{fabs}(rr.first)) {
    if (DEBUG) {
        cout \ll "Point is inside regular closed plane curve. \n" \ll "Returning 1 \n";
        cout \ll "Exiting Reg Cl Plane Curve::location() \n";
        getchar();
    }
    return 1;
}
else if (\text{fabs}(pt0_y) > \text{fabs}(rr.first))
    /* This case should never occur, I believe. LDF 2002.11.05. Why not?? */
    if (DEBUG) {
        cout \ll "Point is outside regular closed plane curve. \n" \ll "Returning -1 \n" \ll "Exiting Reg Cl Plane Curve::location() \n";
        getchar();
    }
    return -1;
}
else {
    \text{cerr} \ll "ERROR! In Reg Cl Plane Curve::location() \n" \ll "This can't happen! Returning -3 \n" \ll \text{flush};
```
```cpp
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 2003.01.05.] TO DO: Find out why this function isn’t `const`!

(Declare `Reg_Cl_Plane_Curve` functions 987) + ≡

```cpp
virtual Point angle_point(real angle);
```
995.  
(Define Reg_Cl_Plane_Curve functions 993) +≡  
Point Reg_Cl_Plane_Curve :: angle_point(real angle)  
{
  cerr << "ERROR! In Reg_Cl_Plane_Curve::angle_point().\n" <<
    "This virtual function doesn't have a real definition for\n" <<
    "ordinary Reg_Cl_Plane_Curves.\nReturning INVALID_POINT.\n\n" << flush;
  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.

---

[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 rotations reduce 
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 collinear. 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 ref_pt, Point p0, Point p1) const;

998. (Define Reg_Ci_Plane_Curve functions 993) +≡

bool_point_pair Reg_Ci_Plane_Curve::intersection_points(Point ref_pt, Point p0, Point p1)
    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 = get_plane();
        if (DEBUG) pl.normal.show("pl.normal");
        Point surface_vector = (get_point(0) - ref_pt);
        surface_vector.unit_vector(true);
        Point pt_vector(p1 - p0);
        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(p0).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!::In_Reg_Ci_Plane_Curve::intersection_points():\n" << "Something is wrong with the normals:\n";
    pt_vector.show("pt_vector");
    pl.normal.show("pl.normal");
    cerr << "Returning INVALID_BOOL_POINT_PAIR.\n" << flush;
    if (DEBUG) cout << "ExitingPolygon::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 \texttt{Reg\_Cl\_Plane\_Curve} functions 993) \begin{verbatim}
if (distance \equiv 0) {
    if (DEBUG) cout \ll "Line, and Reg\_Cl\_Plane\_Curve\_are, coplanar.\n"
    Transform \( t0 \);
    \texttt{Reg\_Cl\_Plane\_Curve} \texttt{copy(*this)};
    \texttt{Point curve\_0} = \texttt{copy.get\_point(0)};
    if (DEBUG) {
        cout \ll "After, copying:" \ll endl;
        show("this:");
        \texttt{copy.show("copy:"};
    curve\_0.show("curve\_0\n");
    }
    \( t0 == curve\_0 \) == pt0 == pt1 == \texttt{copy.shift(-ref_pt)};
    if (DEBUG) {
        cout \ll "After, shift:" \ll endl;
        \texttt{copy.show("copy:"};
        curve\_0.show("curve\_0\n");
    pt0.show("pt0\n");
    pt1.show("pt1\n");
    }
    if (\( curve\_0.\texttt{get\_x()} < 0 \)) {
        \( t0 == curve\_0 \) == pt0 == pt1 == \texttt{copy.rotate(0,0,180)};
    }
    if (DEBUG) {
        cout \ll "After, rotating, so, curve\_0, has, positive, x:" \ll endl;
        \texttt{copy.show("copy:"};
        curve\_0.show("curve\_0\n");
    pt0.show("pt0\n");
    pt1.show("pt1\n");
    }
    \texttt{Point trace.x.x.0} = \texttt{curve\_0};
    \texttt{trace.x.x.0.shift(0,-(curve\_0.\texttt{get\_y()}));}
    \texttt{Point x.axis_pt(1)};
    \texttt{real ang = trace.x.x.0.\texttt{angle(x.axis_pt)};}
    if (DEBUG) {
        cout \ll "ang==--", \ll ang \ll endl \ll flush;
    }
    if (ang \neq 0) {
        if (\( curve\_0.\texttt{get\_x()} > 0 \)) ang \equiv -1;
        \( t0 == curve\_0 \) == pt0 == pt1 == \texttt{copy.rotate(0,ang)};
    }
    if (DEBUG) {
        cout \ll "After, rotating, so, the, trace, of, curve\_0, is, on, x-axis:" \ll endl;
        \texttt{copy.show("copy:"};
        curve\_0.show("curve\_0\n");
    pt0.show("pt0\n");
    pt1.show("pt1\n");
    }
    ang = \texttt{curve\_0.\texttt{angle(x.axis_pt)};}
\end{verbatim}
if (ang \neq 0) {
    if (curve._.get_y() > 0) ang *= -1;
    t0 *= curve._ *= pt0 *= pt1 *= copy.rotate(0, 0, ang);
}

if (DEBUG) {
    cout << "After rotating, so curve_0 is on x-axis:" << endl;
    copy.show("copy");
    pt0.show("pt0");
    pt1.show("pt1");
}

Point curve_4 = copy.get_point(number_of_points / 4);
Point z_axis pt(0, 0, 1);
ang = curve_4.angle(z_axis.pt);
if (DEBUG) {
    curve_4.show("curve_4");
    cout << "ang_4 =" << ang << " endl << flush;
}

if (ang \neq 0) {
    if (curve._.get_y() > 0) ang *= -1;
    t0 *= curve._ *= pt0 *= pt1 *= copy.rotate(ang);
}

if (DEBUG) {
    cout << "After rotating, so curve_4 is on z-axis:" << endl;
    copy.show("copy");
    curve_4.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);
}
#endif
    pt0.dotlabel("pt0");
    pt1.dotlabel("pt1");
    copy.get_center().dotlabel("copy_center");
}

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_z(); /* pt1_y isn't used. Leaving it here, just in case. [LDF 2003.08.27] */
real Slope = pt1_.slope(pt0, 'x', 'z');
if (DEBUG) cout << "Slope_ =" << Slope << " endl << flush;

real pair rr; /* BEGIN */
1002. Slope is 0 (line is horizontal).
(Define `Reg_Cl_Plane_Curve` functions 993) +
if (Slope ≡ 0) /* v is known, h is unknown. */
{
    if (DEBUG) {
        cout << "Slope\n-0\n" << endl << flush;
    }
    rr = solve('h' , pt0_v);
    if (rr.first ≠ INVALID_REAL) {
        bpp.first.pt.set(rr.first , 0 , pt0_v);
    }
    else bpp.first.pt = INVALID_POINT;
    if (rr.second ≠ INVALID_REAL) {
        bpp.second.pt.set(rr.second , 0 , pt0_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 ≡ 0. */

1003. Slope is undefined (line is vertical).
(Define `Reg_Cl_Plane_Curve` functions 993) +
else
    if (Slope ≡ INVALID_REAL) {
        if (DEBUG) {
            cout << "Line\nis\nvertical.\n";
        }
        rr = solve('v' , pt1_h);
        if (rr.first ≠ INVALID_REAL) {
            bpp.first.pt.set(pt1_h , 0 , rr.first);
        }
        else bpp.first.pt = INVALID_POINT;
        if (rr.second ≠ INVALID_REAL) {
            bpp.second.pt.set(pt1_h , 0 , rr.second);
        }
        else bpp.second.pt = INVALID_POINT;
    } /* End Slope ≡ INVALID_REAL. */
1004. Slope ∈ real is defined and ≠ 0.

(Define Reg_Ci_Plane_Curve functions 993) +⇒

else {
    real v_intercept;
    v_intercept = pt0.v - (Slope * pt0.h);
    if (DEBUG) cout << "v_intercept= " << v_intercept << endl << flush;
    real triple coeffs = get_coefficients(Slope, v_intercept); /* New h-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;
    }
    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.Cl_Plane_Curve functions 993) +≡

```cpp
define bool_real on_segment;
if (bpp.first.pt == 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 << "on_segment.first = " << on_segment.first << endl << flush;
    cout << "on_segment.second = " << on_segment.second << endl << flush;
}
if (on_segment.first) bpp.first.b = true;
else bpp.first.b = false;
Transform t.inverse; 
t.inverse = t0.inverse();
if (bpp.first.pt != INVALID_POINT) {
    if (DEBUG) cout << "Transforming bpp.first.pt" << flush;
    bpp.first.pt *= t.inverse;
}
else {
    if (DEBUG) cout << "bpp.first.pt is invalid" << flush;
}
if (bpp.second.pt == INVALID_POINT) {
    on_segment.second = false;
    on_segment.second = INVALID_REAL;
}
else on_segment = bpp.second.pt.is_on_segment(pt0, pt1);
if (on_segment.first) bpp.second.b = true;
else bpp.second.b = false;
if (DEBUG) {
    cout << "on_segment.first = " << on_segment.first << endl << flush;
    cout << "on_segment.second = " << on_segment.second << endl << flush;
}
if (bpp.second.pt != INVALID_POINT) {
    if (DEBUG) cout << "Transforming bpp.second.pt" << flush;
    bpp.second.pt *= t.inverse;
}
else {
    bpp.second.pt = INVALID_POINT;
    if (DEBUG) cout << "bpp.second.pt is invalid" << flush;
}
if (DEBUG) {
    cout << "rr.first = " << rr.first << endl << flush;
    cout << "rr.second = " << rr.second << endl << flush;
    cout << "bpp.first.b = " << bpp.first.b << endl << flush;
    cout << "bpp.second.b = " << bpp.second.b << endl << 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_1 and Reg_Cl_Plane_Curve are in parallel planes.\n" <<
    "No intersections...Returning INVALID_BOOL_POINT_PAIR." << endl << endl << flush;
  return INVALID_BOOL_POINT_PAIR;
}

/* 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\n" << "Reg_Cl_Plane_Curve.\n" << flush;
  } else {
    if (DEBUG)
      cout << "The line and the Reg_Cl_Plane_Curve\n" << "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: s = \n" << 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 << "On segment: s = \n" << bpp.first.b << endl << flush;
    return bpp;
  }
  else return INVALID_BOOL_POINT_PAIR;
}

/* 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_Segment functions 993) +≡

```
bool_point_pair Reg_Cl_Plane_Curve::intersection_points (const Point &ref_pt, const Path &p)
const
{
  if (!(is_linear())) {
    cerr << "ERROR! In Reg_Cl_Plane_Curve::intersection_points():\n" <<
     "Path::point::p.is_non-linear.\n" << "Returning INVALID_BOOL_POINTPAIR.\n\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 2002.11.10.] `segment()` returns a subpath of the Reg_Cl_Plane_Curve representing 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     Optional, with 0 as the default. If angle is ≠ 0, a Point is found in the direction of
                the normal to the Reg_Cl_Plane_Curve from the center of the Reg_Cl_Plane_Curve, and
                the segment is rotated around the center and this Point.
```

```
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 ≡
number_of_points, return *this, cast to a Path, with warning.

[LDF 2002.11.12.] Added “\relax” after the arguments to “\ARG” in the \TeX code above in order to
supress 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 ≡ 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_Cl_Plane_Curve functions 993) +≡

Path Reg_Cl_Plane_Curve::segment(unsigned int factor, real angle, bool closed) const
{
    Path p;
    if (factor <= 1 V factor > number_of_points)
    {
        cerr << "ERROR! In Reg_Cl_Plane_Curve::segment():\n        "The argument factor has an invalid value:\n        "\nReturning empty Path.\n        "\n        return p;
    }
    if (fabs(angle) > 360)
    {
        cerr << "WARNING! In Reg_Cl_Plane_Curve::segment():\n        "The argument angle is greater than 360\n        "It will be reduced.\n        "\n        return 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 += ";
        p += p.get_last_point();
        p += ";
        p += p.get_point(0);
        p += ";
        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_Cl_Plane_Curve. If the argument closed is true, then the segment is closed and can be filled using fill() or filldraw().

(Declare Reg_Cl_Plane_Curve functions 987) +≡

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_Cl_Plane_Curve. If the argument closed is true, then the segment is closed and can be filled using fill() or filldraw().
\section*{1014 \ 3DLD-1.15.1}

\begin{verbatim}
(Declare Reg_Cl_Plane_Curve functions 987) +
  inline Path quarter(real angle = 0, bool closed = true) const
  {
    return segment(4, angle, closed);
  }

1015. Putting Reg_Cl_Plane_Curve together.
  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)
\end{verbatim}
1016. This is what's written to curves.h.

```c
#include <curves.h>  // 1016
#include <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 `rectangs.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.

```c
// Version control identifier 5
static char revision[512] = "Id: polygons.web,v.1.4,2004/01/12,21:32:08,lfinstoi,Exp,1016"
```

1018. Include files.

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

[LDF 2003.06.06.] `Polygon` is meant to be used primarily as a base class for more specialized types of polygons. Currently, `Reg_Polygon` and `Rectangle` are defined. I've added `Polygon` so that I can define intersection functions that will work for both `Reg_Polygon` and `Rectangle`.

Log

[LDF 2003.06.06.] Added class `Polygon`.

```c
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.

[LDF 2003.07.18.] Moved these functions from `Reg_Polygon` to `Polygon`. Also removed the `Rect` versions, since `Rect` inherits the `Polygon` versions.

1022. non-const version.

[LDF 2002.04.24.] Added this function.
[LDF 2003.06.09.] Changed return value from `Point` & to `const Point` &.

(Declare `Polygon` functions 1022) ≡

```cpp
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) ≡

```cpp
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" << "Polygon_ doesn't contain any Points, so its presumably doesn't have a center.\n" << "Returning INVALID_POINT.\n" << flush;
        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 1007.

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) ≡

```cpp
Point get_center() const;
```
1025.  
\{ Define Polygon functions 1023 \} \equiv
\begin{verbatim}
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!:\ln, Polygon::get_center():\n\ll \"Polygon does not contain any Points, it presumably\n\ll \"does not 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;
}
\end{verbatim}

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 &pt0, const Point &pt1) const;


1029. (Define Polygon functions 1023) +≡

```cpp
bool point_pair::intersection_points(const Point &pt0, const Point &pt1) const { bool
  DEBUG = false; /* true */
  if (DEBUG) cout << "Entering::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_point.show("pl.point");
    pl_normal.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_normal.show("pl.normal");
  }
  short distance = pl.get_distance(pt0).second;
```

1030. Degenerate cases, error handling.

(Define Polygon functions 1023) +≡

```cpp
if (pt_vector == INVALID_POINT || pl_normal == INVALID_POINT || pt_vector == origin || pl_normal == origin) {
  cerr << "ERROR! in Polygon::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_POINT_PAIR;
}
```

1031. Parallel and coplanar cases.

```
[Log LDF 2003.06.20] Now checking surface_vector against pt_vector in the following conditional.
```

```cpp
else if (surface_vector == pt_vector || surface_vector == -pt_vector || cross == pl_normal || cross == -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 thebool_point will be true, otherwise false. If a Point lies on a line segment belonging to the Polygon, but not to the line segment $p_0p_1$, the Point will be put into thebool_point, but thebool 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) \(\equiv\)

```c++
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;
                if (DEBUG) cout << "Found, first, intersection.\n";
            } else {
                bpp.second.b = true;
                bpp.second.pt = bp.pt;
                if (DEBUG) cout << "Found, second, intersection. Returning, \n";
                return bpp;
            }
    }
    else if (bp.pt != INVALID_POINT && bp.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;
```
§1032 3DLDF-1.15.1  COPLANAR CASE  293

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.

```cpp
// Define Polygon functions 1023) +\imath
else {
    if (pl.normal == pt_vector || pl.normal == -pt_vector) {
        if (DEBUG) cout << "The line is perpendicular to the polygon.\n" << flush;
    } else {
        if (DEBUG) cout << "The line and the polygon are non-coplanar.\n" << flush;
    }
    bool_point bp = pl.intersection_point(pt0, pt1);
    if (DEBUG) {
        bp.pt.show("bp.pt");
    }
    if (bp.pt == center) {
        if (DEBUG) {
            cout << "bp.pt==center.\n" << endl << 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 << "points.size()\n" << points.size() << endl << flush;
    }
    for (vector<Point>::const_iterator iter = points.begin(); iter != points.end(); ++iter) {
        if ((iter + 1) == 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 == r0 || bp.pt == r1) {
            if (DEBUG) {
                if (bp.pt == r0) cout << "bp.pt==r0.\n" << endl << flush;
                else if (bp.pt == r1) cout << "bp.pt==r1.\n" << endl << flush;
            }
            bpp.first.pt = bp.pt;
            bpp.first.b = bp.pt.is_on_segment(pt0, pt1).first;
            return bpp;
        }
    }
}
```
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 << "Intersection point 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      " << 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 &p) const;
1040.  
(Define Polygon functions 1023)  
\[ \text{Define Polygon functions 1023} \]  
\[ \text{vector (Point) Polygon::intersection_points(const Polygon &r) const \{ bool DEBUG = false; \}} \]  
\[ \text{*/ true */} \]  
\[ \text{vector (Point) v;} \]  
\[ \text{Plane pl = getplane();} \]  
\[ \text{Plane r_pl = r.getplane();} \]  
\[ \text{if (DEBUG) \{} \]  
\[ \text{pl.normal.show("pl.normal");} \]  
\[ \text{r_pl.normal.show("r_pl.normal");} \]  
\[ \text{cout << "pl.distance,-,,-" << pl.distance << endl << flush;} \]  
\[ \text{cout << "r_pl.distance,-,,-" << r_pl.distance << endl << flush;} \]  
\[ \text{\}} \]  
\[ \text{real distance = fabs(fabs(pl.distance) - fabs(r_pl.distance));} \]  
\[ \text{if (distance < \text{Point::epsilon()}) distance = 0;} \]  
\[ \text{if (DEBUG) cout << "distance,-,,-" << distance << endl << flush;} \]  
\[ \text{if (pl.normal == r_pl.normal)} \{ \]  
\[ \text{1041.  Coplanar case.} \]  
(Define Polygon functions 1023)  
\[ \text{Define Polygon functions 1023} \]  
\[ \text{if (distance == 0) \{} \]  
\[ \text{if (DEBUG) cout << "Coplanar.\n";} \]  
\[ \text{bool_point bp;} \]  
\[ \text{Point *ptr;} \]  
\[ \text{Point *r_ptr;} \]  
\[ \text{for (vector (Point *):const_iterator iter = points.begin(); iter != points.end(); ++iter) \{} \]  
\[ \text{if ((iter + 1) == points.end()) ptr = points.front();} \]  
\[ \text{else ptr = *iter + 1;} \]  
\[ \text{for (vector (Point *):const_iterator r_iter = r.points.begin(); r_iter != r.points.end(); ++r_iter) \{} \]  
\[ \text{if ((r_iter + 1) == r.points.end()) r_ptr = r.points.front();} \]  
\[ \text{else r_ptr = *(r_iter + 1);} \]  
\[ \text{bp = Point::intersection_point(**iter, *ptr, **r_iter, *r_ptr);} \]  
\[ \text{if (bp.b) v.push_back(bp.pt);} \]  
\[ \text{\}} \]  
\[ \text{/* Inner for */} \]  
\[ \text{\}} \]  
\[ \text{/* Outer for */} \]  
\[ \text{return v;} \]  
\[ \} \]  
1042.  Parallel case.  
(Define Polygon functions 1023)  
\[ \text{Define Polygon functions 1023} \]  
\[ \text{else /* Parallel. */} \]  
\[ \text{cerr << "WARNING! \text{in Polygon::intersection_points():}\n" \ll \} \]  
\[ \text{"The Polygons lie in parallel planes.\" \ll "Returning empty vector<Point>.
\n\n" \ll flush; \} \]  
\[ \text{return v;} \]  
\[ \} \]
1043. 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;
    }
} else {
    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 l lies within n" <<
        "the intersection of r with l.\n";
}
else if (br[2].first && br[3].first) {
    if (DEBUG) cout << "The intersection of r with l lies within n" <<
        "the intersection of this with l.\n";
}
else if (br[0].first || br[1].first || br[2].first || br[3].first) {
    if (DEBUG) cout << "The intersections of this and r with l overlap partially.\n";
}
else {
    if (DEBUG) cout << "The intersections of this and r with l don’t overlap at all.\n";
    return v;
} /* else. End of non-parallel, non-coplanar case. */

1044. Transformations.

[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.] [LDF 2003.07.18.] Moved “Transformations” section, including operators==(const Transform &) from Reg_Polygon to Polygon. Also removed the Rectangle versions in rectangles.web. The Polygon versions are now inherited by Reg_Polygon and Rectangle.

1045. Applying a transformation.

[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.  
(Define Polygon functions 1023) \(+\equiv\)

```cpp
Transform Polygon::operator*(const Transform &t)  
{
    Path::operator*(t);
    return (center *= t);
}
```

1047.  Rotation around the main axes.  
(Declare Polygon functions 1022) \(+\equiv\)

```cpp
virtual Transform rotate(const real x, const real y = 0, const real z = 0);
```

1048.  
(Define Polygon functions 1023) \(+\equiv\)

```cpp
Transform Polygon::rotate(const real x, const real y, const real z)  
{
    Transform t;
    t.rotate(x, y, z);
    return (*this *= t);
}
```

1049.  Rotate around an arbitrary axis.

1050.  Point arguments.

---

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

```cpp
virtual Transform rotate(const Point &p0, const Point &p1, const real angle = 180);
```

1051.  
(Define Polygon functions 1023) \(+\equiv\)

```cpp
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.

---

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

```cpp
virtual Transform rotate(const Path &p, const real angle = 180);
```
1053.  
(Define Polygon functions 1023) \(\equiv\)  
Transform Polygon \(:: \text{rotate}(\text{const Path } \& p, \text{const real angle})\)  
\{  
  if (!p.is_linear()) {  
    cerr \(\ll\) "ERROR!\(\text{in Ellipse::rotate(Path, real)}\).
\n    return \text{INVALID_TRANSFORM};  
  }  
  return \text{rotate}(p.get_point(0), p.get_last_point(), angle);  
\}

1054.  Scale.  
(Declare Polygon functions 1022) \(\equiv\)  
virtual Transform \text{scale}(\text{real } x, \text{real } y = 1, \text{real } z = 1);

1055.  (Define Polygon functions 1023) \(\equiv\)  
Transform Polygon \(:: \text{scale}(\text{real } x, \text{real } y, \text{real } z)\)  
\{  
  Transform t;  
  t.scale(x, y, z);  
  return (*this *= t);  
\}

1056.  Shear.  
(Declare Polygon functions 1022) \(\equiv\)  
virtual Transform \text{shear}(\text{real } xy, \text{real } xz = 0, \text{real } yx = 0, \text{real } yz = 0, \text{real } xx = 0, \text{real } yy = 0);

1057.  (Define Polygon functions 1023) \(\equiv\)  
Transform Polygon \(:: \text{shear}(\text{real } xy, \text{real } xz, \text{real } yx, \text{real } yz, \text{real } xx, \text{real } yy)\)  
\{  
  Transform t;  
  t.shear(xy, xz, yx, yz, xx, yy);  
  return (*this *= t);  
\}

1058.  Shift.  

1059.  real arguments.  
(Declare Polygon functions 1022) \(\equiv\)  
virtual Transform \text{shift}(\text{real } x, \text{real } y = 0, \text{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.  
(Define Polygon functions 1023) +≡
    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 center will be
        transformed along with the Points pointed to by the pointers on points. Some are present already, but not all.
        [LDF 2003.04.15.]  TO DO: Add in_circle(), out_circle(). Align a line from center in the direction of a
        normal with the y-axis. Use the inverse of the Transform to transform the Circle.

        format Reg_Polygon Polygon

1069.  Reg_Polygon class definition. Reg_Polygon is derived from Polygon.

        Log

        [LDF 2003.04.15.]  Changed, so that Reg_Polygon is derived from Path. Previously, it was derived from
        Reg_CL_Plane_Curve.
        [LDF 2003.04.27.]  Changed protected data members to private. They no longer need to be protected,
        because Rectangle is no longer derived from Reg_Polygon.
        [LDF 2003.06.06.]  Changed, so that Reg_Polygon is derived from Polygon, which I’ve just added above.

        (Define class Reg_Polygon 1069) ≡
        class Reg_Polygon : public Polygon {
            real internal_angle;
            real radius;
            unsigned short sides;
            bool on_free_store;
            public:
                (Declare Reg_Polygon functions 1070 )
        };

This code is used in sections 1097 and 1098.

1070.  Assignment.

        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 Path::operator=( ) in paths.web for more
        information.

        (Declare Reg_Polygon functions 1070) ≡
        const Reg_Polygon &operator=(const Reg_Polygon &p);

See also sections 1073, 1076, 1079, 1087, 1089, 1091, 1092, 1093, 1095, and 1096.

This code is used in section 1069.
1071.  
(Define `Reg_Polygon` functions 1071) \(\equiv\)
\[
\text{const } \text{Reg}_\text{Polygon} &\text{Reg}_\text{Polygon}::\text{operator}=(\text{const } \text{Reg}_\text{Polygon} &p) \\
\{ \\
\quad \text{clear}(); \\
\quad \text{Path}::\text{operator}=(p); \\
\quad \text{internalAngle} = p.\text{internalAngle}; \\
\quad \text{radius} = p.\text{radius}; \\
\quad \text{sides} = p.\text{sides}; \\
\quad \text{center} = p.\text{center}; \\
\quad \text{return } *\text{this}; \\
\}
\]
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) \(\equiv\)
\[
\text{Reg}_\text{Polygon}();
\]

1074.  (Define `Reg_Polygon` functions 1071) \(\equiv\)
\[
\text{Reg}_\text{Polygon}();
\]
\[
\{ \\
\quad \text{on_free_store} = \text{false}; \\
\quad \text{lineSwitch} = \text{false}; \\
\quad \text{cycleSwitch} = \text{true}; \\
\quad \text{projectiveExtremes}.\text{resize}(6, 0); \quad /* \text{LDF 2003.04.09. Added this line. */}
\}
\]

1075.  Center, sides, diameter, and angles.

1076.  Constructor.
(Declare `Reg_Polygon` functions 1070) \(\equiv\)
\[
\text{Reg}_\text{Polygon}(\text{const } \text{Point} &\text{center}, \text{const unsigned short } ssides, \text{const real } ddiameter, \text{const real } angle_x = 0, \text{const real } angle_y = 0, \text{const real } angle_z = 0);
\]
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) +1

Reg_Polygon::Reg_Polygon(const Point &center, const unsigned short ssides, const real
diameter, const real angle_x, const real angle_y, const real angle_z): radius(diameter/2),
sides(ssides) { bool DEBUG = false; /* true */
    if (DEBUG) cout << "Entering Reg_Polygon::Reg_Polygon()", <<
"(center, ssides, diameter, angles).\n" << flush;
    center = center, on_free_store = false;
    internalAngle = 360.0/sides;
    cycle_switch = true;
    projective_extremes.resize(6, 0); /* LDF 2003.04.09. Added this line. */
    center.apply_transform();
}

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) +1

for (int i = 0; i < sides; i++) { Point *vertex = create_new < Point > (0);
    vertex->set(0, 0, radius);
    if (i % 2 == 0) vertex->rotate(0, internalAngle/2.0);
    if (i > 0) /+ LDF 2002.11.06. Only rotate if the angle != 0, i.e., don't rotate the first time. */
        vertex->rotate(0, i * internalAngle, 0);
    if (angle_x != 0 && angle_y != 0 && angle_z != 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, diameter, angles).\n" << flush;
    return;
}

Setting function.

(Declare Reg_Polygon functions 1070) +1

void set(const Point &center, const unsigned short ssides, const real
diameter, const real angle_x = 0, const real angle_y = 0, const real angle_z = 0);

?? [LDF 2002.10.07.] See below.

(Define Reg_Polygon functions 1071) +1

void Reg_Polygon::set(const Point &center, const unsigned short ssides, const real
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, 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, diameter, angle_x, angle_y, angle_z);
*this = p;
if (DEBUG)
    cout << "Exiting Reg_Polygon::set()." << "(center, ssides, diameter, angles).\n" << flush;
return;
}

1082. Pseudo-constructor for dynamic allocation.

1083. Pointer argument.

(Declare non-member template functions for Reg_Polygon 1083) +≡

Reg_Polygon *create_new(const Reg_Polygon *); See also section 1084.

This code is used in sections 1097 and 1098.

1084. Reference argument.

(Declare non-member template functions for Reg_Polygon 1083) +≡

Reg_Polygon *create_new(const Reg_Polygon &); 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.

[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.

[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.

[LDF 2003.06.13] Added this section.

1089. Enclosed circle.

[LDF 2003.06.13] Added this function.

1090. Draw enclosed circle.

1091. Normal version.

1092. Picture argument first.

[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.

1093. Surrounding circle.

1094. Draw surrounding circle.

1095. Normal version.
1096. Picture argument first.

[Log]

[LDF 2003.07.04.] Removed default argument for picture. Having one made it impossible for the compiler to resolve calls to `draw_out_circle()` with no arguments.

(Declare `Reg_Polygon` functions 1070) +

```
Circle draw_out_circle(Picture &picture, const Color &draw_color = *Colors::default_color, const string dashed = "", const string ppens = "") 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)
(Define `Polygon` functions 1023)
(Declare non-member template functions for `Reg_Polygon` 1083)
This is what’s written to polygons.h.

\[ \text{polygons.h 1098} \equiv \]

\begin{verbatim}
(Define class Polygon 1019)
(Define class Reg_Polygon 1069)
(Declare non-member template functions for Reg_Polygon 1083)
\end{verbatim}

Rectangle (rectangles.web).

Log

- [LDF 2003.07.18.] Removed the “Transformations” section, including \texttt{operator\*\*=(const Transform \&)}. Also moved the \texttt{Reg_Polygon} versions to \texttt{Polygon} in \texttt{polygons.web}. These are now inherited by \texttt{Rectangle}.
- [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.

\texttt{format Rectangle Reg_Polygon}

(Version control identifier 5) \equiv

\begin{verbatim}
static string res_id = "$Id::rectangl.web,v,1.5,2004/01/12,21:32:24,1fnsrtool,Exp_8$";
\end{verbatim}

Include files.

(Include files 6) \equiv

\begin{verbatim}
#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"
\end{verbatim}

Rectangle class definition. [LDF 2003.07.18.] TO DO: \textit{axis}_{\mathbf{h}} and \textit{axis}_{\mathbf{v}} are not recalculated when a Rectangle is transformed. I should do something about this.

\begin{verbatim}
(Define class Rectangle 1101) \equiv

class Rectangle : public Polygon {
  real \textit{axis}_{\mathbf{h}};
  real \textit{axis}_{\mathbf{v}};
  bool \texttt{on_free_store};
public:  \{ Declare Rectangle functions 1103 \}
};
\end{verbatim}

This code is used in sections 1139 and 1140.

Constructors and setting functions.
§1103. Default constructor. No arguments.
(Declare Rectangle functions 1103) \equiv
\textbf{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
\textbf{Rectangle}();
\{
  \textit{on\_free\_store} = \textit{false};
  \textit{line\_switch} = \textit{false};
  \textit{cycle\_switch} = \textit{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 \textit{angle\_x}, \textit{angle\_y}, and \textit{angle\_z}, if at least one of them is non-zero.

§1106. Constructor.

\begin{verbatim}
| Log |
\hline
| [LDF 2002.11.06] Made real arguments const. |
| [LDF 2003.07.18] BUG FIX: Now, \textit{axis\_h} and \textit{axis\_v} are no longer divided by 2, when I initialize \textit{axis\_h\_half} and \textit{axis\_v\_half}. I mistakenly used /= instead of /.
\hline
\end{verbatim}

(Declare Rectangle functions 1103) \texttt{=} \equiv
\textbf{Rectangle}(\texttt{const Point &ccenter, const real aaxis\_h, const real aaxis\_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 aaxis_h, const real aaxis_v, const real angle_x, const real angle_y, const real angle_z): axis_h(aaxis_h), axis_v(aaxis_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 bot_lft(-axis_h_half,0,-axis_v_half);
  Point bot_rt(axis_h_half,0,-axis_v_half);
  Point top_lft(-axis_h_half,0,axis_v_half);
  Point top_rt(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. */
  {
    bot_lft.rotate(angle_x, angle_y, angle_z);
    bot_rt.rotate(angle_x, angle_y, angle_z);
    top_lft.rotate(angle_x, angle_y, angle_z);
    top_rt.rotate(angle_x, angle_y, angle_z);
  } /* Put around center. */
  bot_lft.shift(ccenter);
  bot_rt.shift(ccenter);
  top_lft.shift(ccenter);
  top_rt.shift(ccenter);
  top_rt.shift(ccenter); for (int i = 0; i < 4; i++) points.push_back ( create_new < Point > (0) ) ;
  *points[0] = bot_lft;
  *points[1] = bot_rt;
  *points[2] = top_rt;
  *points[3] = top_lft;
}

1108. Setting function.

Log

[LDF 2002.11.06.] Made real arguments const.

(Declare Rectangle functions 1103) +≡
void set(const Point &ccenter, const real aaxis_h, const real aaxis_v, const real aaxis_w, 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 aaxis_h, const real aaxis_v, const real aaxis_w, const real angle_x, const real angle_y, const real 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 pt4 = pt1 - pt0;  /* Check that the Point arguments are coplanar. */
      Point pt5 = pt2 - pt0;
      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_h = (pt1 - pt0).magnitude();
          axis_w = (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::(Rectangle) with four Point arguments.\n          Points are not coplanar. Returning.\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 Rectangle functions 1104 ] [≡

```cpp
void Rectangle::set(const Point &pt0, const Point &pt1, const Point &pt2, const Point &pt3)
{
    Rectangle r(pt0, pt1, pt2, pt3);
    *this = r;
}
```

1115. Pseudo-constructor for dynamic allocation.

1116. Pointer argument.

[LDF 2003.12.30.] Replaced Rectangle::create_new_rectangle() with a specialization of template (class C) C*create_new() for Rectangle.

(Declare non-member template functions for Rectangle 1116 ] ≡

```cpp
Rectangle *create_new(const Rectangle &c);
```

See also section 1117.

This code is used in sections 1139 and 1140.

1117. Reference argument.

[LDF 2003.12.30.] Replaced Rectangle::create_new_rectangle() with a specialization of template (class C) C*create_new() for Rectangle.

(Declare non-member template functions for Rectangle 1116 ] ≡

```cpp
Rectangle *create_new(const Rectangle &c);
```

1118. 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 Rectangle (rather than Reg_Polygon or Path).

1119. Assignment.

[LDF 2002.11.06.] Changed return value from void to const Rectangle &.

(Declare Rectangle functions 1103 ] ≡

```cpp
const Rectangle &operator=(const Rectangle &c);
```
1120.  !! Remember to put anything specific to Rectangle in here!

    (Define Rectangle functions 1104) +≡
    const Rectangle &Rectangle :: operator=(const Rectangle &c)
    {
        clear();
        Path::operator=(c);
        center = c.center;
        axis_h = c.axis_h;
        axis_v = c.axis_v;
        return *this;
    }

1121.  Returning Elements and information.

[Log] [LDF 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. [LDF 2003.12.02]

[Log] [LDF 2003.11.28.] Added this function.
[LDF 2003.12.02.] Added test of planarity at beginning of function.
[LDF 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) +≡

```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) ≡ origin ∧ c.cross_product(d) ≡ origin ∧ fabs(a, angle (d)) - 90 < 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 <= `c` <= 3.

(Declare `Rectangle` functions 1103) +≡

```cpp
Point corner(unsigned short c);
```

1126.  
(Define `Rectangle` functions 1104) +≡

```cpp
Point Rectangle::corner(unsigned short c)
{
  if (c > 3) {
    cerr ≡ "ERROR: Rectangles have 4 corners."
        ≡ "numbered 0 through 3. Returning INVALID_POINT."
    return INVALID_POINT;
  }
  return *points[c];
}
```

1127.  Get Mid-point.  The argument `c` should be in the range 0 <= `c` <= 3.

Log

[LDF 2002.11.06.] Changed this function so that it uses `medlate()`.

[LDF 2003.06.09.] Renamed this function `get_mid_point()`. Formerly, it was called `mid_point()`.

[LDF 2003.07.18.] Made `const`.

(Declare `Rectangle` functions 1103) +≡

```cpp
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 mid_points, " <<
               "numbered 0 through 3. Returning INVALID_POINT."
               << 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 } \equiv
real Rectangle \::: \text{get\textunderscore axis\textunderscore v}( )\ const
\{
  \text{return axis\textunderscore v};
\}

1134.  Ellipses.

\[LDF\ 2003.07.18.\]  Added this section. These functions must be defined in \texttt{ellipses.web}, because \texttt{Ellipse} is an incomplete type in this file.

1135.  Surrounding Ellipse.

\[LDF\ 2003.07.18.\]  Added this function.

\{Declare Rectangle functions 1103 } \equiv
Ellipse \texttt{out\textunderscore ellipse}( )\ const;

1136.  Enclosed Ellipse.

\[LDF\ 2003.07.18.\]  Added this function.

\{Declare Rectangle functions 1103 } \equiv
Ellipse \texttt{in\textunderscore ellipse}( )\ const;

1137.  Draw surrounding Ellipse.

\[LDF\ 2003.07.18.\]  Added this function.

\{Declare Rectangle functions 1103 } \equiv
Ellipse \texttt{draw\textunderscore out\textunderscore ellipse}(\texttt{const Color }&\texttt{draw\textunderscore color} = \texttt{*Colors::default\textunderscore color},\texttt{string ddashed} = \\"",\texttt{string ppen} = \\",\texttt{Picture }&\texttt{picture} = \texttt{current\textunderscore picture})\ const;

1138.  Draw enclosed Ellipse.

\[LDF\ 2003.07.18.\]  Added this function.

\{Declare Rectangle functions 1103 } \equiv
Ellipse \texttt{draw\textunderscore in\textunderscore ellipse}(\texttt{const Color }&\texttt{draw\textunderscore color} = \texttt{*Colors::default\textunderscore color},\texttt{string ddashed} = \\"",\texttt{string ppen} = \\",\texttt{Picture }&\texttt{picture} = \texttt{current\textunderscore picture})\ const;

1139.  Putting Rectangle together.  This is what’s compiled.

\{Include files 6\}
\{Version control identifier 5\}
§1 139  3DLDF-1.1.5.1  PUTTING RECTANGLE TOGETHER  317

{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

\langle Version control identifier 5 \rangle ≡
static string res_id = "$Id:\ellipses.web,v,1.7,2004/01/14,17:56:19,1lfinsto1,Exp,$";

1142. Include files.

\langle Include files 6 \rangle ≡
#include "loader.h"
#include "pspglb.h"
#include "creatinew.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"

1143. Ellipse class definition.

Log

\[LDF\ 2003.07.25.\] Added focus0, focus1, and linear_eccentricity.
\[LDF\ 2003.07.27.\] Added numerical_eccentricity.

\langle Define class Ellipse 1143 \rangle ≡
class Ellipse : public Reg_Cl_Plane_Curve {
  protected: Point focus0;
  Point focus1;
  real linear_eccentricity;
  real numerical_eccentricity;
  real axis_J;
  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.


(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 axis_h, const real axis_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 (const Point &center, const real aaxis_h, const real aaxis_v, const real angle_x, const real angle_y, const real angle_z, const unsigned short number_of_points):

  axis_h(aaxis_h), axis_v(aaxis_v) { center = center;
  center.apply_transform();
  focus0 = origin;
  focus1 = origin;
  real axis_h_half = aaxis_h / 2;
  real axis_v_half = aaxis_v / 2;
  if (aaxis_h >= aaxis_v) {
    linear_eccentricity = sqrt((aaxis_h_half * aaxis_h_half) - (aaxis_v_half * aaxis_v_half));
    focus0.shift(-linear_eccentricity);
    focus1.shift(linear_eccentricity);
    numericalEccentricity = linear_eccentricity / axis_h_half;
  }
  else {
    linear_eccentricity = sqrt((aaxis_v_half * aaxis_v_half) - (aaxis_h_half * aaxis_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" << ", invalid" << "Using default, instead:" << 
    DEFAULT_NUMBER_OF_POINTS << endl << 
    flush;
    number_of_points = DEFAULT_NUMBER_OF_POINTS;
  }
  else number_of_points = number_of_points;
  on_vote_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_y;
for (int i = 0; i < number_of_points; i++)
  /* LDF 2002.11.06. Modified this code. */
  { curr_angle = 2 * PI * i / number_of_points;
    curr_x = aaxis_h / 2 * cos(curr_angle);
    curr_y = aaxis_v / 2 * sin(curr_angle); points.push_back (create_new <= Point > (0));
    points.back () . set(curr_x,0, curr_y);
    *(points.back ()) *= t;
  } /* Rotate *(points.back ()) around the x, y, and z-axes and shift it to center. */}
1151. Setting function.

\[ \text{Log} \]

[\text{LDF 2003.03.01.}] Added this function.

\( \langle \text{Declare Ellipse functions 1146} \rangle \) \[ \equiv \]

\text{void \textit{set} (const \textit{Point} &\textit{center}, \textit{const real} \textit{axis}_h, \textit{const real} \textit{axis}_v, \textit{const real} \textit{angle}_x = 0, \textit{const real} \textit{angle}_y = 0, \textit{const real} \textit{angle}_z = 0, \textit{const unsigned short} \textit{number_of_points} = \text{DEFAULT\_NUMBER\_OF\_POINTS}); \]

1152. \( \langle \text{Define Ellipse functions 1147} \rangle \) \[ \equiv \]

\text{void \textit{Ellipse::set} (const \textit{Point} &\textit{center}, \textit{const real} \textit{axis}_h, \textit{const real} \textit{axis}_v, \textit{const real} \textit{angle}_x, \textit{const real} \textit{angle}_y, \textit{const real} \textit{angle}_z, \textit{const unsigned short} \textit{number_of_points}); \}

\}

1153. Pseudo-constructor for dynamic allocation.

1154. Pointer argument.

\[ \text{Log} \]

[\text{LDF 2002.11.06.}] Added optional \textit{const \textit{Ellipse} *} \textit{argument}.  
[\text{LDF 2003.12.30.}] Replaced \textit{Ellipse::create\_new\_ellipse} () with a specialization of \textit{template\langle\text{class} \textit{C}\rangle \textit{C}\_create\_new\_()} for \textit{Ellipse}.  

\( \langle \text{Declare non-member template functions for Ellipse 1154} \rangle \) \[ \equiv \]

\text{Ellipse *\textit{create\_new} (const \textit{Ellipse} *\textit{e});} 

See also section 1155.  
This code is used in sections 1271 and 1272.
1155. **Reference argument.**

---

[LDF 2002 11.06.] Added this function.
[LDF 2003.12.30.] Replaced Ellipse::create_new_ellipse() with a specialization of template<class C>
\[\text{C*create_new()}\] for Ellipse.

---

(Declare non-member template functions for Ellipse 1154) \texttt{+\texttt{+}}

\[
\text{Ellipse *create_new(const Ellipse \\&e);}
\]

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.**

---

[LDF 2002 11.06.] Added error handling code to prevent self-assignment.
[LDF 2002 11.10.] Changed return value to Ellipse \\&.

---

(Declare Ellipse functions 1146) \texttt{+\texttt{+}}

\[
\text{Ellipse \\&operator=(const Ellipse \\&e);}
\]

1158. !! Remember to put anything specific to Ellipses in here!

(Define Ellipse functions 1147) \texttt{+\texttt{+}}

\[
\text{Ellipse \\&Ellipse \::\operator=(const Ellipse \\&e)}
\]

\[
\begin{verbatim}
\{
  if (this \texttt{\equiv \&e}) \texttt{/* LDF 2002.11.06. */ Make sure it’s not self-assignment. */}
  return \texttt{\&this};
  Path::operator=(e);
  center = e.center; \texttt{/* Ellipse members. */}
  axis_h = e.axis_h;
  axis_v = e.axis_v;
  focus0 = e.focus0;
  focus1 = e.focus1;
  number_of_points = e.number_of_points; \texttt{/* Reg_Cl_Plane_Curve members. */}
  return \texttt{\&this};
\}
\end{verbatim}
\]

1159. **Labelling.**

1160. **Label.**

(Declare Ellipse functions 1146) \texttt{+\texttt{+}}

\[
\text{void label(const string pos = "top", const bool dot = false, Picture \\&picture = current_picture)}
\]

\[
\text{const;}
\]
§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.

[LDF 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.  [LDF 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.

[LDF 2003.07.20.] Added this function.
[LDF 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."
      << endl << "This is non-planar. Returning false."
      << endl << flush;
    return false;
  }
  
  Ellipse e = *this;
  normal = shift(center);
  Transform t;
  t.align_with_axis(center, normal, 'y');
  e = do_transform(t);
  if (DEBUG) e.show("e.after_ellipse:");
  Point x_axis_pt(1);
  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 Elliptical():\n"     << "Rotation failed. Returning false\n"
      << flush;
    return false;
  }
  if (DEBUG) e.show("e.after_ellipse.in.x-y-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_x();
    if (DEBUG) {
        cout << "Pointi" << i << ": " << endl << flush;
        (**iter).show("iter:");
        cout << "x_i=" << x << endl << flush;
        cout << "z_i=" << z << endl << flush;
    }
    r = ((x * x)/(a * a)) + ((z * z)/(b * b));
    if (DEBUG) cout << "x_i-" << r << endl << flush;
    if (fabs(r - 1) > Point::epsilon()) {
        if (DEBUG) cout << "Pointi" << i << ", does not satisfy ellipse\equation.\n" << endl << flush;
        return false;
    }
    ++i;
}

1166. Is quadratic.
(Declare Ellipse functions 1146) +≡
inline bool is_quadratic() const
{
    return true;
}

1167. Is cubic.

---

Log


(Declare Ellipse functions 1146) +≡
virtual bool is_cubic() const;

---
1168.  

\begin{verbatim}
(Define Ellipse functions 1147) +≡
  bool Ellipse::is_cubic() const
  {
    return false;
  }
\end{verbatim}

1169.  Is quartic.

Log

[LDf 2003.07.27.] Made virtual and non-inline.

\begin{verbatim}
(Declare Ellipse functions 1146) +≡
  virtual bool is_quartic() const;
\end{verbatim}

1170.  

\begin{verbatim}
(Define Ellipse functions 1147) +≡
  bool Ellipse::is_quartic() const
  {
    return false;
  }
\end{verbatim}

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.

Log

[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.

\begin{verbatim}
(Declare Ellipse functions 1146) +≡
  real_pair solve(char axis_unknown, real known) const;
\end{verbatim}
1172.

(Define Ellipse functions 1147) +≡

```cpp
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

\[ \frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \]

where a is half the horizontal axis and b is half the vertical axis. Therefore,

\[ y = \sqrt{(1 - \frac{x^2}{a^2}) \times b^2} \]

and

\[ x = \sqrt{(1 - \frac{y^2}{b^2}) \times a^2}. \]

(Define Ellipse functions 1147) +≡

\[
\begin{align*}
r.\text{first} &= \sqrt{((1 - ((\text{known}*\text{known})/(\text{radius}\_\text{known} + \text{radius}\_\text{known})))) \times (\text{radius}\_\text{unknown} + \text{radius}\_\text{unknown})}); \\
r.\text{second} &= -r.\text{first} ; \\
\text{return} \ r; \ }
\end{align*}
\]

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

\[ \frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \]

namely !! START HERE. Check this. I think the x in the coefficient for \( x^2 \) is wrong.

\[ \frac{x^2}{a^2} + (mx + b)^2 / b^2 - 1 = 0 \equiv (\beta^2 x + \alpha^2 m^2)x^2 + 2\alpha^2 b mx + (\alpha^2 b^2 - \alpha^2 \beta^2) = 0. \]

The coefficients are returned in the struct real_triple in the order one would expect: \( r.\text{first} \) is the coefficient of \( x^2 \), \( r.\text{second} \) of x and \( r.\text{third} \) 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) +≡

\[
\text{real_triple getCoefficients(real Slope, real v_intercept) const ;}
\]

1175.

(Define Ellipse functions 1147) +≡

\[
\text{real_triple Ellipse :: getCoefficients(real Slope, real v_intercept) const} \\
\{
\quad \text{real_triple r; }
\quad \text{real ax_h = get_axis_h();}
\quad \text{real ax_v = get_axis_v();}
\quad \text{r.\text{first} = ((ax_v/2) * (ax_v/2)) + ((ax_h/2) * (ax_h/2)) * (Slope * Slope));} \quad /* \ a */
\quad \text{r.\text{second} = 2 * Slope * v_intercept * ((ax_h/2) * (ax_h/2));} \quad /* b */
\quad \text{r.\text{third} = (((ax_h/2) * (ax_h/2)) * v_intercept * v_intercept) - (((ax_v/2) * (ax_v/2)) * (ax_h/2) * (ax_h/2))}; \quad /* c */
\quad \text{return} \ r; \}
\]

1176. Get center.

{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.

{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.

{Declare Ellipse functions 1146} +≡
  const Point &get_focus(const unsigned short s);
1183.  
(Define Ellipse functions 1147) \(\equiv\)
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():\n" "Invalid argument: s \n" "Valid arguments are 0 and 1. \n" "Returning INVALID_POINT.\n" << flush;
        return INVALID_POINT;
    }
}

1184.  const version.

------------- Log -------------

[LDF 2003.07.25.] Added this function.

(Declare Ellipse functions 1146) \(\equiv\)
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!\n|Ellipse::get_focus()|:\nInvalid argument: \"s \" \"" << s << endl <<
    "Valid arguments are 0 and 1.\nReturning,INVALID_POINT.\n\n" << flush;
    return INVALID_POINT;
  }
}

1186. Get linear eccentricity.

[LD 2003.07.25.] Added this function.

(Declare Ellipse functions 1146) +≡
real get_linear_eccentricity() const;

1187.
(Define Ellipse functions 1147) +≡
real Ellipse::get_linear_eccentricity() const
{
  return linear_eccentricity;
}

1188. Get numerical eccentricity.

[LD 2003.07.27.] Added this function.

(Declare Ellipse functions 1146) +≡
real get_numerical_eccentricity() const;
1189.  
(Define Ellipse functions 1147) \(\varepsilon\)  
\[
\text{real Ellipse::get\_numerical\_eccentricity() const} \\
\quad \text{return numerical\_eccentricity;} 
\]

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 \(a_{\perp}\) or \(a_{\parallel}\) should have is recalculated, and this value is returned. In the non-const versions, \(a_{\perp}\) or \(a_{\parallel}\) is reset to the new value. If *this is no longer elliptical, the function returns INVALID\_REAL, and \(a_{\perp}\) or \(a_{\parallel}\) is set to INVALID\_REAL in the non-const versions.  

[LDF 2003.07.20.] \(a_{\perp}\) and \(a_{\parallel}\) 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: \(a_{\perp}\) and \(a_{\parallel}\) were too small by half. Now multiplying by 2 in all versions of get\_axis\_\(a_{\parallel}\)( ) and get\_axis\_\(a_{\perp}\)( ).

1191.  Get vertical axis.  

1192.  const version.  
(Declare Ellipse functions 1146) \(\varepsilon\)  
\[
\text{real get\_axis\_\(a_{\perp}\)( ) const;} 
\]

1193.  
(Define Ellipse functions 1147) \(\varepsilon\)  
\[
\text{real Ellipse::get\_axis\_\(a_{\perp}\)( ) const} \\
\quad \text{if (is\_elliptical()) return (2 * (get\_point(number\_of\_points/4) - get\_center()).magnitude());} \\
\quad \text{else return INVALID\_REAL;} 
\]

1194.  Non-const version.  
(Declare Ellipse functions 1146) \(\varepsilon\)  
\[
\text{real get\_axis\_\(a_{\perp}\)();} 
\]

1195.  
(Define Ellipse functions 1147) \(\varepsilon\)  
\[
\text{real Ellipse::get\_axis\_\(a_{\perp}\)()} \\
\quad \text{if (is\_elliptical()) \(a_{\perp} = ((get\_point(number\_of\_points/4) - get\_center()).magnitude() \ast 2);} \\
\quad \text{else \(a_{\perp} = INVALID\_REAL;\)} \\
\quad \text{return \(a_{\perp};\)} 
\]

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.

---

[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 pt0.

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_u() and get_axis_v(), 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 &c);
#endif
1204. (Define Ellipse functions 1117) +

```cpp
#include <iostream>

class Ellipse {
public:
    Ellipse() = default;
    Ellipse(const Ellipse& e) : center(e.center), a(e.a), b(e.b), c(e.c) {
        // Initialize ellipse properties
    }
    virtual ~Ellipse() = default;

private:
    double center[2]; // Center coordinates
    double a, b, c; // A, B, C coordinates

    // Accessor methods (getters)
    double getA() const { return a; }
    double getB() const { return b; }
    double getC() const { return c; }

    // Mutator methods (setters)
    void setA(double a) { this->a = a; }
    void setB(double b) { this->b = b; }
    void setC(double c) { this->c = c; }

    // Other methods
    double area() const {
        return M_PI * a * b;
    }
    double perimeter() const {
        return M_PI * (3 * (a + b) - sqrt((3 * a + b) * (a + 3 * b)));
    }

    virtual bool operator==(const Ellipse& e) {
        return center[0] == e.center[0] && center[1] == e.center[1] && a == e.a && b == e.b && c == e.c;
    }

};
```

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 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 \( (P - foci ).magnitude() + (P - foci ).magnitude() \), d for \( 2 \times \max(\text{axis}_h, \text{axis}_v) \), and \( \epsilon \) for Point::epsilon(). The return values are as follows:

- **0** \( |m - d| < \epsilon \). 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 1116) +

```cpp
virtual signed short location(const Point &p) const;
```
1206.  
(Define Ellipse functions 1147) \(\equiv\)

\[
\text{signed short Ellipse::location(const Point \\ &p) const}
\]

\[
\begin{align*}
\text{bool DEBUG = false; } & \quad / \!* \text{ true } */ \\
\text{real } ar_h = \text{get_axis}_h(); & \quad \\
\text{real } ar_v = \text{get_axis}_v(); & \\
\text{if } (ar_h \equiv \text{INVALID-REAL } \lor ar_v \equiv \text{INVALID-REAL}) \{ & \\
& \quad \text{cerr } \ll \text{"ERROR! In Ellipse::location():
\n" } \ll \\
& \quad \text{"Ellipse is non-elliptical...
\nReturning -3.\n
" } \ll \text{flush;} \\
& \quad \text{return -3;} \\
\}
\]

\[
\begin{align*}
\text{if } (-p.\text{is_on_plane (get客气()) }) \{ & \\
& \quad \text{cerr } \ll \text{"WARNING! In Ellipse::location():
\n" } \ll \\
& \quad \text{"Point doesn't lie on plane of Ellipse.
\n" } \ll \text{"Returning -2.\n
" } \ll \text{flush;} \\
& \quad \text{return -2;} \\
\}
\]

\[
\begin{align*}
\text{real } max_ax = \text{max}(ar_h, ar_v); & \quad \\
\text{Point } q = p - \text{get_focus}(0); & \\
\text{real } mag = q.\text{magnitude}(); & \\
q = p - \text{get_focus}(1); & \\
mag += q.\text{magnitude}(); & \\
\text{if } (\text{fabs}(mag - max_ax) \lt \text{Point::epsilon()}) \{ & \\
& \quad \text{if } (\text{DEBUG}) \quad \text{cout } \ll \text{"Point lies on
\nperimeter of Ellipse.\n\n" } \ll \\
& \quad \text{return 0;} & \\
\}
\]

\[
\begin{align*}
\text{else if } (mag > max_ax) \{ & \\
& \quad \text{if } (\text{DEBUG}) \quad \text{cout } \ll \text{"Point lies outside
\nperimeter of Ellipse.\n\n" } \ll \\
& \quad \text{return -1;} & \\
\}
\]

\[
\begin{align*}
\text{else } \{ & \\
& \quad \text{if } (\text{DEBUG}) \quad \text{cout } \ll \text{"Point lies inside
\nperimeter of Ellipse.\n\n" } \ll \\
& \quad \text{return 1;} & \\
\}
\end{align*}
\]

1207.  Intersection points.

1208.  Point arguments.

\[
\begin{array}{cccc}
\text{Log} & \text{1206. \(\equiv\)} & \text{1207.} & \text{1208.}
\end{array}
\]

\[
\begin{align*}
\text{LDF 2003.07.27. Made the arguments const Point \\ 
\&.}
\end{align*}
\]

(Declare Ellipse functions 1146) \(\equiv\)

\[
\begin{align*}
\text{virtual bool_point_pair intersection_points(const Point \\ &p0, const Point \\ &p1) const;}
\end{align*}
\]
1200.  
(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.

1211. (Define Ellipse functions 1147) +≡
  bool_point_pair Ellipse::intersection_points(const Path &p) const
  {
    if (!p.is_linear()) {
      cerr << "ERROR! UnEllipse::intersection_points(const, Path)\n"
           "Path argument is non-linear.\n"
           "Returning INVALID_BOOL_POINT_PAIR.\n"
           "endl endl flush; return INVALID_BOOL_POINT_PAIR;
    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 the 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 \( 15^\circ \) or \( 45^\circ \), \textit{intersection_points()} found both intersections. Try to find out why! I want to write a \texttt{Circle} version with an \texttt{Ellipse} argument, and vice versa. If I do, I may not have to worry about this problem.

1214. [LDF 2003.07.20.]

\begin{verbatim}
[Log]

[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!

[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 \texttt{Planes} and \texttt{Lines}, which is not a bad thing, and it checks whether the intersection \texttt{Points} are on the \texttt{Ellipse}, so it wasn’t a total loss.

[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.

[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.

[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.

[LDF 2002.04.14.] Changed argument from \texttt{const Ellipse \&} to \texttt{Ellipse}, since I was having to copy it, anyway.

[LDF 2003.07.01.] Added \texttt{verbose} argument.

[LDF 2003.07.01.] Changed “perpendicular and skew case”. Debugged coplanar case, where the \texttt{Ellipses} don’t have the same centers and/or axis orientation.

[LDF 2003.07.06.] Made a minor change to the conditional that determines whether the \texttt{Ellipses} are coplanar or not.

[LDF 2003.08.14.] Setting \texttt{verbose} to \texttt{true} if \texttt{VERBOUSE\_GLOBAL} is \texttt{true}. Added \texttt{VERBOUSE\_GLOBAL} to \texttt{pspgib_web} today.

[LDF 2004.01.12.] Added \texttt{real step} argument, with 3 as its default value.
\end{verbatim}
§1215.
(Define Ellipse functions \(1147\) \(\equiv\)

```cpp
bool_point_quadruple Ellipse::intersection_points(Ellipse e, const real step, bool verbose) const {
    bool DEBUG = false;  /* true */
    if (VERBOSE_GLOBAL) verbose = true;
    if (DEBUG || verbose)
        cout << "Entering Ellipse::intersection_points(Ellipse, e, const, bool, verbose)" << end;
    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 \(\equiv\) e_plane.normal \(\lor\) this_plane.normal \(\equiv\) -e_plane.normal) {
```

1216. Coplanar case.

```
[LDf 2003.07.20.] Now using `get_axis_h()` and `get_axis_v()`, instead of accessing `axis_h` and `axis_v` directly.

(Define Ellipse functions \(1147\) \(\equiv\)

```cpp
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(point[0]);
    this_axis_orientation = get_center();
    this_axis_orientation, unit_vector (true);

Point e_axis_orientation (e, 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);
1218. [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 }  +
\[
\text{if (e._axis_orientation \neq this._axis_orientation) e._axis_orientation_rotated.rotate(origin, normal_point, 90);}
\text{if (DEBUG) e._axis_orientation_rotated.show("e._axis_orientation");}
\text{if (this.center \equiv e.center \land ((this._axis_orientation \equiv e._axis_orientation \lor this._axis_orientation \equiv
-e._axis_orientation) \lor (this._axis_orientation \equiv e._axis_orientation_rotated \lor this._axis_orientation \equiv
-e.axis_orientation_rotated)))}
\text{if (DEBUG) cout \ll "Centers, and axis, orientation, are, the, same." \ll flush;}
\text{/* [LDF 2002.04.14.] Pick the maximum of axis_h and axis_v and multiply it by 1.5. We’ll use a line segment of this length to find intersection points with the two Ellipses. Using this length guarantees we’ll find them. Actually, max(axis_h, axis_v) ought to do the trick. */}
\]

Point pt0;
\text{if (ax_h \geq ax_v) pt0 = get_point(0);}
\text{else pt0 = get_point(number_of_points / 4);}
\text{pt0 = center;}
\text{pt0 *= 1.5;}
\text{pt0.shift(center);}
\text{if (DEBUG) }
\text{this._plane.normal.show("normal");}
\text{center.show("center");}
\text{/* [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 pt0. */}

Point pt1 (this._plane.normal);
\text{pt1.shift(center);}
\text{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;
\text{bool point_pair bpp_this;}
\text{bool point_pair bpp_e;}
\text{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 the we Ellipses are not congruent and in the same location, so we break out of the loop. */}
\text{for (int i = 0; i < 5; i++) }
\text{if (DEBUG) cout \ll "1_{-}\ldots\ldots\ldots\ldots\ldots\ldots; i \ll endl \ll flush;}
\text{if (i \neq 0) pt0.rotate(center, pt1, 30);}
\text{if (DEBUG) pt0.dotlabel("00");}
\text{bpp_this = intersection_points(center, pt0);}
\text{bpp_e = e.intersection_points(center, pt0);}
\text{if (bpp_this.first.b \equiv true) }
\text{if (DEBUG) cout \ll "first_is, an, intersection, point,\ (this).\n\text{pt2 = bpp_this.first.pt;}
\text{}}
\text{else if (bpp_this.second.b \equiv true) }
\text{if (DEBUG) cout \ll "second, is, an, intersection, point,\ (this).\n\text{pt2 = bpp_this.second.pt;}
\text{}}
\text{else}
\text{if (DEBUG) cout \ll "no, intersection, point,\ (this).\n\text{pt2 = 0;}
\text{}}
cerr << "In Ellipse::intersection_points(Ellipse).\n" << "This can't happen (this)!\n" << "Will try to continue.\n\n" << flush;
pt2 = INVALID_POINT;
}
if (bpy.first.b == true) {
  if (DEBUG) cout << "first is an intersection point of (e).\n";
  pt3 = bpy.first.pt;
}
else if (bpy.second.b == true) {
  if (DEBUG) cout << "second is an intersection point of (e).\n";
  pt3 = bpy.second.pt;
}
else {
  cerr << "In Ellipse::intersection_points(Ellipse).\n" << "This can't happen (e)!\nWill try to continue.\n\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 --\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_QUADRUPLE.\n\n" << flush;
  return INVALID_BOOL_POINT_QUADRUPLE;
}
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. Programm this case!! */
else if (DEBUG) {
  cout << "The centers are different, or the axis orientation is\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 1147) +≡
if (copy_center ≠ 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 1147) +≡
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.

[Log] Commented-out the declaration of OTHER, because it’s never used. I haven’t deleted it, just in case.

(Define Ellipse functions 1147) +≡
unsigned short orientation;
const unsigned short X_Y = 0;
const unsigned short X_Z = 1;
const unsigned short Z_Y = 2;
#if 0
const unsigned short OTHER = 3;
#endif
if (normal_x ≡ 0 ∧ normal_y ≡ 0 ∧ normal_z ≡ 0) {
cerr « "ERROR! in Ellipse:intersection_points():\n" « "Normal_u=" "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 y-z plane. */ {
if (DEBUG) cout « "Ellipse lies in a plane parallel to y-z 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(“c”);
}
Point ellipse_p0 = copy.get_point(0);
Transform t0;
t0.align_with_axis(copy_center, ellipse_p0, ‘x’);
copy »= t0;
copy_center »= t0;
elipse_p0 »= t0;
e »= t0;
if (DEBUG) {

3DLDF-1.15.1 §1221
```cpp
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;
t *= t0;
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");
}
```
1222. 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(0);
e_axis_orientation -= e_center;
e_axis_orientation.unit_vector(true); if ((e_center == origin ∧ copy_center == origin) ∧ (copy_axis_orientation == e_axis_orientation ∨ copy_axis_orientation == − e_axis_orientation) ∨ (copy_axis_orientation == e_axis_orientation.oriented ∨ copy_axis_orientation == − e_axis_orientation.oriented)) { 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(1)/2.0;
real d = e.get_axis_v(1)/2.0;
if (DEBUG) {
    cout << "a, b = " << a << endl << flush;
    cout << "b, c = " << b << endl << flush;
    cout << "c, d = " << c << endl << flush;
    cout << "d, d = " << d << endl << flush;
}
real aa = (a * a);
real bb = (b * b);
real cc = (c * c);
real dd = (d * d);
if (DEBUG) {
    cout << "a*a, b*b, c*c, d*d = " << aa << endl << flush;
    cout << "b*b, c*c, d*d = " << bb << endl << flush;
    cout << "c*c, d*d = " << cc << endl << flush;
    cout << "d*d, d*d = " << dd << endl << flush;
}
real denominator;
real numerator;
if (DEBUG) cout << "x, coordinate.\n";
denominator = (aa - ((bb * cc)/dd)) * dd;
umererator = bb * (aa - cc);
if (DEBUG) {
    cout << "numerator, numerator = " << numerator << endl << flush;
    cout << "denominator, denominator = " << denominator << endl << flush;
}
if (denominator == 0) {
    if (DEBUG) cout << "x, INVALID_REAL.\n";
\begin{verbatim}
x = INVALID_REAL;
}
else {
    try {
        x = -(c * sqrt(1 - (numerator/denominator)));
    }
    catch(...) {
        x = INVALID_REAL;
        if (DEBUG) cout << "x = INVALID_REAL\n";
    }
}
if (DEBUG) {
    cout << "x = \n" << endl << flush;
    cout << "y coordinate: \n";
}
numerator = b * sqrt(fabs(aa - cc));
denominator = sqrt(fabs(aa - ((bb + cc)/dd)));
if (DEBUG) {
    cout << "numerator = \n" << numerator << endl << flush;
    cout << "denominator = \n" << denominator << endl << flush;
}
if (denominator == 0) {
    if (DEBUG) cout << "y = INVALID_REAL\n";
    y = INVALID_REAL;
} else {
    try {
        y = -(numerator/denominator);
    }
    catch(...) {
        y = INVALID_REAL;
        if (DEBUG) cout << "y = INVALID_REAL\n";
    }
}
if (DEBUG) {
    cout << "y = \n" << y << endl << flush;
}
\end{verbatim}
1223. 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.

(Define Ellipse functions 1147) +\+

```cpp
if (x \equiv INVALID_REAL \lor y \equiv INVALID_REAL) {
  cerr \ll "WARNING! In Ellipse::intersection_points(Ellipse).\n  "Ellipses don't intersect.\n  "Returning INVALID_BOOL_POINT_QUADRUPE.\n  \n  return INVALID_BOOL_POINT_QUADRUPE;
}
else if (orientation \equiv X_Y) {
  pt20.set(x, y, 0);
  pt21.set(-x, -y, 0);
  if (y \neq 0 \land x \neq 0) {
    pt22.set(x, -y, 0);
    pt23.set(-x, y, 0);
  }
  else {
    pt22 = INVALID_POINT;
    pt23 = INVALID_POINT;
  }
}
else if (orientation \equiv X_Z) {
  pt20.set(x, 0, y);
  pt21.set(-x, 0, -y);
  if (y \neq 0 \land x \neq 0) {
    pt22.set(x, 0, -y);
    pt23.set(-x, 0, y);
  }
  else {
    pt22 = INVALID_POINT;
    pt23 = INVALID_POINT;
  }
}
else if (orientation \equiv Z_Y) {
  pt20.set(0, y, x);
  pt21.set(0, -y, -x);
  if (x \neq 0 \land y \neq 0) {
    pt22.set(0, y, -x);
    pt23.set(0, -y, x);
  }
  else {
    pt22 = INVALID_POINT;
    pt23 = INVALID_POINT;
  }
}
else {
  cerr \ll "ERROR! In Ellipse::intersection_points(Ellipse).\n  "This can't happen! orientation has invalid value: \n  "orientation \ll orientation \ll endl \ll "Will try to continue.\n  \n  return INVALID_BOOL_POINT_QUADRUPE;
}
```

```cpp
if (DEBUG) {
```
pt20.show("pt20");
pt20.dotlabel("20");
pt21.show("pt21");
pt21.dotlabel("21");
pt22.show("pt22");
pt22.dotlabel("22");
pt23.show("pt23");
pt23.dotlabel("23");
}
signed short ss_copy;
signed short ss_e;
if (pt20 != INVALID_POINT) {
    ss_copy = copy.location(pt20);
    if (DEBUG) cout << "ss_copy=--" << ss_copy << endl << flush;
    ss_e = e.location(pt20);
    if (DEBUG) cout << "ss_e=--" << ss_e << endl << flush;
}
if (ss_copy == 0 && ss_e == 0) {
    pt20 == t_inverse;
    pt21 == t_inverse;
    bpq.first.b = true;
    bpq.first.pt = pt20;
    bpq.second.b = true;
    bpq.second.pt = pt21;
    if (pt22 != INVALID_POINT) {
        if (DEBUG) cout << "Ellipses intersect at four points.\n";
        pt22 == t_inverse;
        bpq.third.b = true;
        bpq.third.pt = pt22;
    }
    else {
        if (DEBUG) cout << "Ellipses intersect at two points.\n";
    }
    if (pt23 != INVALID_POINT) {
        pt23 == t_inverse;
        bpq.fourth.b = true;
        bpq.fourth.pt = pt23;
    }
    return bpq;
} /* if */
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 e.\n";
    else if (ss_e == -1) cerr << "e is inside of *this.\n";
    else {
        cerr << "This can’t happen! Invalid value for ss_e:--" << 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”. */
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
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.

Define Ellipse functions:

```cpp
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.

Define Ellipse functions:
```cpp
```
\texttt{cerr \ll \textquote{ERROR! In Ellipse::intersection_points():}\n\ll \\textquote{This can't happen! curr_location has bad value: u}\ll \text{curr_location \ll \\
\textquote{u}}\ll \text{"Returning INVALID_BOOL_POINT_QUADRUPLE\n
\ll flush; \\
return INVALID_BOOL_POINT_QUADRUPLE;}\}
}

\textbf{if} (\texttt{curr_location} \neq \texttt{location_switch}) \{
\textbf{if} (\texttt{DEBUG}) \{
\quad \texttt{cout \ll \textquote{Found a transition!}\n\ll \textquote{i}} \ll \text{endl \ll flush; \\
\quad \texttt{cout \ll \textquote{save_angle -- u}} \ll \text{save_angle \ll endl \ll flush; \\
\quad \texttt{cout \ll \textquote{curr_location -- u}} \ll \text{curr_location \ll endl \ll flush; \\
\quad \texttt{cout \ll \textquote{location_switch -- u}} \ll \text{location_switch \ll endl \ll flush; \\
\textbf{if} (\texttt{location_switch} \equiv 1) \{
\quad \texttt{in_angle} = \text{save_angle; \n\quad \texttt{out_angle} = \text{i;} \\
\textbf{else} \{
\quad \texttt{in_angle} = \text{i;} \n\quad \texttt{out_angle} = \text{save_angle; \\
\textbf{while} (\texttt{true}) \{
\quad \texttt{test_angle} = (\text{in_angle} + \text{out_angle})/2; \\
\quad \texttt{curr_point} = \text{copy_angle_point(test_angle);} \\
\quad \texttt{curr_location} = \text{e.location(curr_point); \n\quad \textbf{if} (\texttt{curr_location} \equiv 0) \{
\quad \textbf{if} (\texttt{DEBUG}) \{
\quad \quad \texttt{cout \ll \textquote{Found an intersection point!}\n\ll \textquote{curr_point:} ; \\
\quad \quad \texttt{curr_point.show("curr_point:"); \\
\quad \}\text{Handle intersection point 1225} \}
\quad \quad \texttt{i = floor(test_angle);} \\
\quad \quad \texttt{i += .5;} \\
\quad \textbf{if} (\texttt{i < test_angle}) \texttt{i += .5;} \\
\quad \texttt{save_angle} = \text{i;} \\
\quad \texttt{curr_point} = \text{copy_angle_point(i);} \\
\quad \texttt{location_switch} = \text{e.location(curr_point); \n\quad \textbf{break; \\
\textbf{else if} (\texttt{curr_location} \equiv 1) \{
\quad \texttt{in_angle} = \text{test_angle; \\
\textbf{continue; \\
\textbf{else if} (\texttt{curr_location} \equiv -1) \{
\quad \texttt{out_angle} = \text{test_angle; \\
\textbf{continue; \\
\textbf{else} \{
\quad \texttt{cerr \ll \textquote{ERROR! In Ellipse::intersection_points(Ellipse).}\n\ll \textquote{This can't happen! Invalid value for curr_location: u}\ll \text{curr_location \ll \\
\textquote{Returning INVALID_BOOL_POINT_QUADRUPLE.\n
\ll flush; \\
return INVALID_BOOL_POINT_QUADRUPLE; 
}
1225. Handle intersection point.

(Handle intersection point 1225) \[\]
{++intersection ctr;
 if (DEBUG) {
    cout << "intersection ctr: " << intersection ctr << endl << flush;
}
if (intersection ctr == 1) {
    bpq.first.b = true;
    bpq.first.pt = curr_point;
    bpq.first.pt *= \_inverse;
}
else if (intersection ctr == 2) {
    bpq.second.b = true;
    bpq.second.pt = curr_point;
    bpq.second.pt *= \_inverse;
}
else if (intersection ctr == 3) {
    bpq.third.b = true;
    bpq.third.pt = curr_point;
    bpq.third.pt *= \_inverse;
}
else if (intersection ctr == 4) {
    bpq.fourth.b = true;
    bpq.fourth.pt = curr_point;
    bpq.fourth.pt *= \_inverse;
    if (DEBUG \_verbose) {
        cout << "In Ellipse::intersection_points(Ellipse, const bool, verbose)" << 
            "Found fourth intersection point. \n" << "Returning bpq.\n\n";
    }
    return bpq;
}
else {
    cerr << "ERROR! in Ellipse::intersection_points(Ellipse).\n" << 
        "This can\t happen! Invalid value for\n" << "intersection ctr:\n" << 
        intersection ctr << ",\nReturning \n" << "INVALID_BOOL_POINT_QUADRUPLE.\n\n";
    return INVALID_BOOL_POINT_QUADRUPLE;
}
}

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.

{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 << "bbox.first.b = \n" << bpp.first.b << endl << flush;
        bpp.first.pt.show("bbox.first.point:");
        cout << "bbox.second.b = \n" << bpp.second.b << endl << flush;
        bpp.second.pt.show("bbox.second.point:");
    }
    bpp.first.pt = bpp.first.pt;
    bpp.first.b = (bpp.first.pt == INVALID_POINT) ? false : true;
    bpp.second.pt = bpp.second.pt;
    bpp.second.b = (bpp.second.pt == INVALID_POINT) ? false : true;
    bpp = e.intersection_points(isect_line.position, pt0);
    bpp.first.pt = bpp.first.pt;
    bpp.second.b = (bbox.first.pt == INVALID_POINT) ? false : true;
    bpp.second.pt = bpp.second.pt;
    bpp.second.b = (bbox.second.pt == INVALID_POINT) ? false : true;

    signed short s_x;
    signed short s_y;
    bool temp_bool;
    string temp_string;
    if (bbox.first.b) {
        s_x = location(bbox.first.pt);
        s_y = e.location(bbox.first.pt);
        temp_bool = bbox.first.b;
        temp_string = "First";
        (Check intersection point locations 1228)
    }
    else if (verbose) cout << "FirstIntersectionPointIsINVALID_POINT.\n";
    if (bbox.second.b) {
        s_x = location(bbox.second.pt);
        s_y = e.location(bbox.second.pt);
        temp_bool = bbox.second.b;
        temp_string = "Second";
        (Check intersection point locations 1228)
    }
    else if (verbose) cout << "SecondIntersectionPointIsINVALID_POINT.\n";
    if (bbox.third.b) {
        s_x = location(bbox.third.pt);
        s_y = e.location(bbox.third.pt);
        temp_bool = bbox.third.b;
        temp_string = "Third";
        (Check intersection point locations 1228)
§1227 3DLDF-1.15.1

ELLIPSE ARGUMENT

} 
else if (verbose) cout << "Third,intersection_point, is,INVALID_POINT.\n"; 
if (bpq_fourth.b) {
  s,l = location(bpq_fourth.pt);
  s,e = 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.]

[Log]

1228. Check intersection point locations 1228} =
if (temp_bool) {
  if (verbose) {
    if (s,l = 0 & s,e = 0)
      cout << temp_string << ".point, lies, on, the, perimeter, of, both, ellipses.\n";
    else if (s,l = 0) cout << temp_string << ".point, lies, on, the, perimeter, of, *this, \n";
    else if (s,l = -1) cout << temp_string << ".point, lies, outside, *this, \n";
    else if (s,l = 1) cout << temp_string << ".point, lies, inside, *this, \n";
    else if (s,e = -2) cout << temp_string << ".point, doesn’t, lie, in, the, plane, of, *this, \n";
    if (s,e = 0 & s,l ≠ 0) cout << temp_string << ".point, lies, on, the, perimeter, of, e, \n";
    else if (s,l = -1) cout << temp_string << ".point, lies, outside, e, \n";
    else if (s,e = 1) cout << temp_string << ".point, lies, inside, e, \n";
    else if (s,e = -2) cout << temp_string << ".point, doesn’t, lie, in, the, plane, of, e, \n";
  }
}

This code is used in section 1227.

1229. Transformations.

[Log]

[1229.04.27.] 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 \equiv 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 \equiv true, and the transformation does not cause *this to become non-elliptical, axis_h and axis_v are recalculated.

---

Log

[LDF 2003.07.20] Added this function. It makes it possible to perform a transformation on an Ellipse, optionally calling is_elliptical(). It is called in is_elliptical() with check \equiv false. This prevents operator*=( ) and is_elliptical() from calling each other ad infinitum.

[LDF 2003.07.25] Added code for recalculating linear_eccentricity, focus0 and focus1.

[LDF 2003.07.25] BUG FIX: axis_h and axis_v were too small by half. Now multiplying by 2.


---

(Declare Ellipse functions 1146) +≡

virtual Transform do_transform(const Transform &t, bool check = false);
1232.

<Define Ellipse functions 1147> \equiv

Transform Ellipse :: do_transform(const Transform &t, bool check)
{
    bool DEBUG = false; /* true */
    focus0 \equiv focus1 \equiv center \equiv 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() \times 2;
            axis_v = (get_point(number_of_points/4) - c).magnitude() \times 2;
            real axis_h_half = axis_h/2;
            real axis_v_half = axis_v/2;
            if (fabs(axis_h - old_axis_h) > Point::epsdon() \vee fabs(axis_v - old_axis_v) > Point::epsdon()) {
                if (DEBUG) cout \ll "Recalculating,linear_eccentricity,v" \ll "numerical_eccentricity,linear_eccentricity,center\n" \ll 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;
                    focus01 = 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;
                    focus01 = focus0 = (get_point(number_of_points/4) - get_center());
                }
                focus01.unit_vector(true);
                focus01.unit_vector(true);
                focus0 *= -linear_eccentricity;
                focus1 *= linear_eccentricity;
                focus1 *= focus0.shift(get_center());
            }
            else if (DEBUG)
                cout \ll "axis_h and axis_v are unchanged\n" \ll "Not recalculating,foci\n" \ll flush;
        } /* if (is_elliptical()) */
    } else {
        cerr \ll "WARNING! In Ellipse::do_transform(const Transform&)\n" \ll "This transformation has made," \ll *this, non-elliptical!" \ll endl \ll endl \ll flush;
        axis_h = axis_v = linear_eccentricity = INVALID_REAL;
        numerical_eccentricity = INVALID_REAL;
    }
    return t;
}
1233. Operator.

| Log |
| [LDF 2002.04.12.] Added this section. |
| [LDF 2003.07.20.] Changed, so that it calls $\text{do\_transform()}$ with $\text{check} \equiv \text{true}$, so that $\text{is\_elliptical()}$ is called. |

\[
\text{Declare Ellipse functions 1146} \equiv \text{virtual Transform operator} *= (\text{const Transform} \ & t);
\]

1234.

\[
\text{Define Ellipse functions 1147} \equiv \text{Transform Ellipse} :: \text{operator} *= (\text{const Transform} \ & t) \|
\]
\[
\left\{ \begin{array}{l}
\text{return } \text{do\_transform}(t, \text{true});
\end{array} \right.
\]

1235. Rotation around the main axes.

| Log |
| [LDF 2003.07.25.] Changed, so that $\text{do\_transform()}$ is called with $\text{check} \equiv \text{false}$. Rotation can neither change the lengths of $\text{axis}_h$ or $\text{axis}_v$, nor make an Ellipse non-elliptical, so there’s no need to check $*this$ after rotation. |

\[
\text{Declare Ellipse functions 1146} \equiv \text{virtual Transform rotate(const real } x, \text{const real } y = 0, \text{const real } z = 0);\]

1236.

\[
\text{Define Ellipse functions 1147} \equiv \text{Transform Ellipse} :: \text{rotate}(\text{const real } x, \text{const real } y, \text{const real } z)\]
\[
\left\{ \begin{array}{l}
\text{Transform } t; \\
\text{\quad } t.\text{rotate}(x, y, z); \\
\text{\quad return } \text{do\_transform}(t, \text{false});
\end{array} \right.
\]

1237. Scale.

| Log |
| [LDF 2003.07.20.] Added check for whether $*this$ is still elliptical after the scaling operation. |

\[
\text{Declare Ellipse functions 1146} \equiv \text{virtual Transform scale(real } x, \text{real } y = 1, \text{real } z = 1);\]
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] 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] 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.

(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] [LDF 2003.07.25.] 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.

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

[LDF 2003.05.02.] Changed name of this function from `rotate_around()` to `rotate()`. This function now overloads `rotate()` with three `real` arguments.

[LDF 2003.07.25.] Changed, so that `do_transform()` is called with `check == false`. Rotation 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 rotation.

```
<Declare Ellipse functions 1146> +≡

virtual Transform rotate(const Point &p0, const Point &p1, const real angle = 180);
```

1253.

```
<Define Ellipse functions 1147> +≡

Transform Ellipse::rotate(const Point &p0, const Point &p1, const real angle)
{
    Transform t;
    t.rotate(p0, p1, angle);
    return do_transform(t, false);
}
```

1254. Path arguments.

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

```
<Declare Ellipse functions 1146> +≡

virtual Transform rotate(const Path &p, const real angle = 180);
```
1255. Log

[LDF 2003.04.27.] Changed \texttt{get\_point(1)} to \texttt{get\_last\_point()}. 

\begin{verbatim}
(Define Ellipse functions 1147) \equiv
Transform Ellipse :: rotate(const Path & p, const real angle)
{
    if (!p.is\_linear()) {
        std::cerr \ll \text{"ERROR! In Ellipse::rotate(Path,\_real).\n\nReturning INVALID\_TRANSFORM.\n\n"};
        return INVALID\_TRANSFORM;
    }
    return rotate(p.get\_point(0), p.get\_last\_point(), angle);
}
\end{verbatim}

1256. Rectangles.

1257. Surrounding rectangle. Log

[LDF 2003.07.18.] Made const.

\begin{verbatim}
(Declare Ellipse functions 1146) \equiv
Rectangle out\_rectangle() const;
\end{verbatim}
1258.  
\{ Define Ellipse functions 1147 \} +≡  
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;  
\}  

1259. Inscribed rectangle.  

Log  
[ LDF 2003.07.18.] Made const.  

\{ Declare Ellipse functions 1146 \} +≡  
Rectangle `in_rectangle ()` const;
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 \equiv INVALID\_POINT \lor bpp0\_second\_pt \equiv INVALID\_POINT \lor bpp1\_first\_pt \equiv INVALID\_POINT \lor bpp1\_second\_pt \equiv INVALID\_POINT)
    {
        cerr << "Intersection\_didn’t\_work.\n\n" << "Returning\_empty\_rectangle.\n\n" << flush;
        Rectangle\ r2;
        return\ r2;
    }
    Rectangle\ r1(bpp0\_second\_pt, bpp1\_first\_pt, bpp0\_first\_pt, bpp1\_first\_pt, bpp1\_second\_pt);
    return\ r1;
}

1261. Draw\ surrounding\ rectangle.

Log

- [LDF 2003.07.01.] Changed the return value from void to\ Rectangle. Now the surrounding\ Rectangle is returned.
- [LDF 2003.07.18.] Made\ const.

Declare\ Ellipse\ functions\ 1146\ ) 

Rectangle\ draw\_out\_rectangle(const\ Color &draw\_color = *Colors::default\_color, string
dashed = "", string ppens = "", Picture &picture = current\_picture)\ const;

1262. Define\ Ellipse\ functions\ 1147\ ) 

Rectangle\ Ellipse::draw\_out\_rectangle(const\ Color &draw\_color, string\ dashed, string\ ppens, Picture &picture)\ const
{
    Rectangle\ r = out\_rectangle();
    r\_draw(draw\_color, dashed, ppens, picture);
    return\ r;
}

1263. Draw\ inscribed\ rectangle.

Log

- [LDF 2003.07.01.] Changed the return value from void to\ Rectangle. Now the inscribed\ Rectangle is returned.
- [LDF 2003.07.18.] Made\ const.

Declare\ Ellipse\ functions\ 1146\ ) 

Rectangle\ draw\_in\_rectangle(const\ Color &draw\_color = *Colors::default\_color, string
 dashed = "", string ppens = "", Picture &picture = current\_picture)\ const;
1264. Define Ellipse functions. 

```cpp
Rectangle Ellipse::draw_in_rectangle (const Color &draw_color, string dashed, string ppen, Picture &picture) const
{
    Rectangle r = in_rectangle();
    r.draw (draw_color, dashed, ppen, picture);
    return r;
}
```

1265. Rectangle functions. [LDF 2003.07.18.] TO DO: Add `undraw_in_ellipse()`, `fill_out_ellipse()`, etc. Also, I should add versions with the `Picture` argument first.

[LDF 2003.07.18.] Added this section. These functions are declared in `rectangs.web`, but must be defined here, because `Ellipse` is an incomplete type there.

1266. Ellipses.
The image contains a page from a document discussing the calculation of the surrounding ellipse of a rectangle. The code snippet is written in a language that appears to be C++, and it involves functions to calculate the distance and midpoint from the center of an ellipse to a point on its boundary. The code includes comments that explain the steps involved in the calculation, such as calculating the distance from the center to a point on the boundary, aligning the transformation with the ellipse's normal, and finding the intersection points between the ellipse and a line segment. The comments at the end of the code snippet note that this function was added in a specific version of the library on a specific date.

---

Here is the code snippet translated into plain text:

```plaintext
1267. Surrounding Ellipse.

LDF 2003.07.18.] Added this function.

{ 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);
    if (M.unit_vector() != x_axis.pt) {
        cerr << "WARNING!\nIn\nRectangle::in_ellipse() :\n" << "M.is_not_u(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. [LDF 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!\nIn\nRectangle::out_ellipse() \n" << "Couldnt\nfind\nintersection_point.\n" << "Returning\nempty 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]

[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 += C == 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! In |Rectangle::in_ellipse() |: \nl" \ll "M.is_not_u(1, 0, 0) !\nl"
        M.show("M: ");
        /* I'd rather output this to stderr, but I don't have a way to do this yet. [LDF 2003.07.18.] */
        cout \ll endl \ll flush;
    }
    Ellipse e(origin, h_length, v_length);
    e += t.inverse();
    return e;
}


[Log]

[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.

\begin{verbatim}
\{Define Rectangle functions 104 \} + \equiv
Ellipse Rectangle::draw_in_ellipse(const Color &ddraw_color, string ddashed, string ppen, Picture &picture) const
{
    Ellipse e = in_ellipse();
    e.draw(ddraw_color, ddashed, ppen, picture);
    return e;
}
\end{verbatim}

1271. Putting Ellipse together. This is what’s compiled.

\begin{verbatim}
\{Include files 6 \}
\{Version control identifier 5 \}
\{Define class Ellipse 1143 \}
\{Define static Ellipse data members 1144 \}
\{Define Ellipse functions 1147 \}
\{Define Rectangle functions 1104 \}
\{Declare non-member template functions for Ellipse 1154 \}
\end{verbatim}
1272. This is what’s written to ellipses.h.

\{ellipses.h 1272\} \equiv
\{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 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.

---

format Circle Shape

\{Version control identifier 5\} \equiv

\{Version control identifier 5\} \equiv

static string res_id = "$Id:\circles.web,v,1.8,2004/01/12,21:27:22,1\$";

1274. Include files.

\{Include files 6\} \equiv

#include "loader.h"
#include "pspglb.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"
#include "polygons.h"
#include "rectangs.h"
#include "ellipses.h"

1275. Circle class definition.
1276.  
(Define class Circle 1276) ≡
  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) ≡
  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) ≡
  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) +≡
  Circle(const Point &center, const real diameter, 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) +≡
  Circle : : Circle(const Point &center, const real diameter, const real angle_x, const real angle_y, const real angle_z, const unsigned short number_of_points)
  : radius(diameter/2) {
    on_free_store = false;
    line_switch = false;
    cycle_switch = true;
    center = center;
    center.apply_transform();
    axis_h = axis_v = diameter;
    number_of_points = number_of_points;
    Ellipse e(center, axis_h, axis_v, angle_x, angle_y, angle_z, number_of_points);
    *this = e;
  }
1283. Setting function.

[LD 2003.05.06.] Added the argument `number_of_points`. Without it, this setting function didn’t match the constructor above.

(Declare Circle functions 1278 ) ≡

```c
void set(const Point &center, const real diameter, 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);
```
1284. (Define Circle functions 1279) +≡

```c
void Circle::set(const Point &ccenter, const real diameter, const real angle_x, const real angle_y, const real angle_z, const unsigned short nnumber_of_points)
{
    Circle c(ccenter, diameter, angle_x, angle_y, angle_z, nnumber_of_points);
    *this = c;
    return;
}
```

1285. Pseudo-constructor for dynamic allocation.

1286. Pointer argument.

---

[LDF 2003.08.10] Removed redefinition of the default argument `const Circle *p = 0` from the definition.
The DEC compiler complained, but GCC didn't.

[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 Circle 1286) +≡

```c
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) +≡

```c
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=(...)`. 
§1290  3DLDF-1.1.5.1

\[
\begin{align*}
\text{(Declare Circle functions 1278)} & \equiv \\
& \text{Circle \& operator} = (\text{const Circle \& c})
\end{align*}
\]

1291.

\[
\begin{align*}
\text{(Define Circle functions 1279)} & \equiv \\
& \text{Circle \& Circle::operator} = (\text{const Circle \& c}) \\
& \{ \\
& \quad \text{radius} = c.\text{radius}; \\
& \quad \text{Ellipse::operator} = (c); \\
& \quad \text{return *this;}
& \}
\end{align*}
\]

1292. **Ellipse argument.** This function returns a reference to *this, which can be used for further assignment.

\[
\begin{align*}
\text{Log} & \quad \text{Log}
\end{align*}
\]

\text{[LDF 2002.11.10.] Changed and simplified this function. It now uses Ellipse::operator =().} \\
\text{[LDF 2003.08.14.] Added code for handling the case that e.axis\_v and e.axis\_h differ by a small amount, possible due to imprecision (see below).}

\[
\begin{align*}
\text{(Declare Circle functions 1278)} & \equiv \\
& \text{Circle \& operator} = (\text{const Ellipse \& e})
\end{align*}
\]
1293. If \( e \cdot \text{axis}_v \neq e \cdot \text{axis}_h \), 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 \texttt{Point::epsilon()} instead of checking whether they're equal. \[\text{LDF 2003.08.14.}\]

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 \textit{radius} to half of the former, because it's probably the correct value. So, if \( e \cdot \text{axis}_v \equiv \text{floor}(e \cdot \text{axis}_v) \), we set \textit{radius} to \( e \cdot \text{axis}_v/2 \). Otherwise, we set \textit{radius} to \( e \cdot \text{axis}_h \) without further ado. If \( e \cdot \text{axis}_h \equiv \text{floor}(e \cdot \text{axis}_h) \), so much the better, and if it isn't, neither it nor \( e \cdot \text{axis}_v \) has an integer value, so it doesn't matter which one we use to set \textit{radius}. \[\text{LDF 2003.08.14.}\]

(Define Circle functions 1279) ++

\begin{verbatim}
Circle &Circle::operator=(const Ellipse &e)
{
    real e_axis_v = e.get_axis_v();
    real e_axis_h = e.get_axis_h();
    if (e.axis_v \neq e_axis_h) {
        if (fabs(e.axis_v - e_axis_h) > Point::epsilon()) {
            cerr \ll "ERROR! in Circle::operator-(Ellipse).\n            "Elliptic has unequal axes.\n            "Can't perform assignment.\n            Returning.\n            
        return *this;
        }
        else if (e.axis_v \equiv \text{floor}(e.axis_v)) radius = e.axis_v/2.0;
        else radius = e.axis_h/2.0;
    }
    else radius = e.get_axis_v()/2.0;
    Ellipse::operator=(e);
    return *this;
}
\end{verbatim}

1294. Returning elements and information.

1295. Is circular. Tests whether *this is circular. Operations such as \texttt{scale()} and \texttt{shear()} can cause \texttt{Circles} to become non-circular. \[\text{LDF 2003.07.25.}\]

\texttt{is\_circular()} first tests whether *this is planar using \texttt{is\_planar()}. If it's not, \texttt{false} is returned. Otherwise, the \texttt{Point} \( p \) is set to *\texttt{points[0]}\), and \texttt{Point} \( c = get\_center() \) is is subtracted from \( p \). \texttt{p\_magnitude()} is then stored in \texttt{real mag0}. \[\text{LDF 2003.07.25.}\]

Then, \( p \) is set to each of the other \texttt{Points} on the \texttt{Circle} in turn, \( c \) is subtracted from \( p \), and \( p\_magnitude() \) is stored in \texttt{real mag}. If the absolute value of the difference \( mag - mag0 \) is greater than \texttt{Point::epsilon()}, \texttt{is\_circular()} immediately returns \texttt{false}. If \( \text{fabs}(mag - mag0) \leq \text{Point::epsilon()} \) for all of the \texttt{Points} *\texttt{points[n]}\) for \( n > 0 \), \texttt{is\_circular()} returns \texttt{true}. \[\text{LDF 2003.07.25.}\]

\[\text{LDF 2003.07.25.} \] Added this function.

(Declare Circle functions 1278) ++

\begin{verbatim}
bool is\_circular(void) const;
\end{verbatim}
1296.  
(Define Circle functions 1279) +≡

```cpp
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(): " << *this << non-planar.\nReturning false." << 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" << i << doesn't lie on Circle.\n Exiting Circle::is_circular().\nReturning false.\n" << flush;
      return false;
    }
  }
  ++i;
  if (DEBUG) cout << "Exiting Circle::is_circular().\nReturning true.\n" << flush;
  return true;
}
```

1297.  Get radius.  Log

[LD] 2002.05.10. Added this function.

```cpp
(Declare Circle functions 1278) +≡
inline real get_radius()
{
  return radius;
}
```

1298.  Get diameter.  [LD] 2002.05.10. Added this function.

```cpp
(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](image)

[Declare Circle functions 1278] +==

```cpp
virtual bool_point_pair intersection_points(const Point &pt0, const Point &pt1) const;
```

1301.

[Define Circle functions 1279] +==

```cpp
bool_point_pair Circle::`intersection_points(const Point &pt0, const Point &pt1) const
{
    return Ellipse::`intersection_points(pt0, pt1);
}
```

1302. **Path argument.**

![Log](image)

[Declare Circle functions 1278] +==

```cpp
virtual bool_point_pair intersection_points(const Path &p) const;
```
1303.  
(Define Circle functions 1279 ) \( \implies \)

```cpp
bool_point_pair Circle::intersection_points(const Path &p) const
{
    return Ellipse::intersection_points(p);
}
```

1304.  Circle argument.

\[ \text{Log} \]

[LDF 2003.07.20.] Wrote the definition of this function. Tested all cases. It should probably be tested more thoroughly.

[LDF 2003.08.14.] Made `verbose` argument non-`const`. Setting `verbose` to `true` if `VERBOSE.Globalization` is `true`. Added `VERBOSE.Globalization` to `psg1b.web` today.

[LDF 2003.08.27.] Removed the declaration `real c.radius = c.radius`, since `c.radius` was never used.

(Declare Circle functions 1278 ) \( \implies \)

```cpp
virtual bool_point_quadruple intersection_points(const Circle &c, bool verbose = false) const;
```
1305.

(Define Circle functions 1279 ) +−

\begin{verbatim}
bool Point::in_rectangle(const Circle &c, bool verbose) const
{
  bool DEBUG = false;  /* true */
  if (VERBOSE) verbose = true;
  if (DEBUG || verbose) cout << "************\n" <<
    "Entering::intersection_points(const Circle&)\n\n";

bool_point_quadruple bpq = INVALID_BOOL_POINT_QUADRUPLE;
Plane this_plane = get_plane();
Plane c_plane = c.get_plane();
if (!this_plane.normal || c_plane.normal) {
  if (verbose || DEBUG) cout << "Circles,are,non-coplanar..\n" <<
    "Calling::intersection_points()\n\n" << flush;
  bpq = Ellipse::intersection_points(c, verbose);
  if (DEBUG)
    cout << "Exiting::intersection_points(const Circle&)" << endl << endl << flush;
  return bpq;
}
else {
  if (verbose || DEBUG) cout << "Circles,are,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,\n";
    if (radius == c.radius) {
      if (verbose || DEBUG) {
        cout << "Circles,are,\n" << "Returning::INVALID_BOOL_POINT_QUADRUPLE.\n\n" << flush;
      }
      if (DEBUG) cout << "Exiting::intersection_points(const Circle&).\n" << endl << endl << flush;
      return INVALID_BOOL_POINT_QUADRUPLE;
    }
    else if (radius > c.radius) {
      if (verbose || DEBUG) {
        cout << "\n\n\n/*this,\n" << "Returning::INVALID_BOOL_POINT_QUADRUPLE.\n\n" << flush;
      }
else {
cerr << "ERROR! \"c\" in \"Circle::intersection_points(const Circle&)\" \"enl\" "
"This can't happen!" \"\"radius, and/or \"c\radius, have invalid values: \n" \n"radius, -c\" \"radius \"enl \"c\radius, \"c\radius, \"enl \n"Returning INVALID_BOOLEAN QUADRUPLE.\n\n" \n flush;
if (DEBUG) cout << "\nExiting Circle::intersection_points(const Circle&)." \n \n flush;
return INVALID_BOOLEAN QUADRUPLE;
}
/* if (dist \equiv 0) */
else if (dist > (radius + c.radius)) {
  if (verbose \verb+\+ DEBUG)
    cout << "\"This, and, c, lie\, outside, of, each\, other, \n" \n"Returning INVALID_BOOLEAN QUADRUPLE.\n\n" \n flush;
if (DEBUG) cout << "\nExiting Circle::intersection_points(const Circle&)." \n \n flush;
return INVALID_BOOLEAN QUADRUPLE;
}
else if (dist \equiv (radius + c.radius)) {
  if (verbose \verb+\+ DEBUG) cout << "\This, and, c, have a, tangent, on, the, outside,\n" \n"\n\nOne, intersection, point, .\n";
bgf.first.pt.set(c.center - center);
bgf.first.pt.unit_vector(true);
bgf.first.pt *= radius;
bgf.first.pt.shift(center);
bgf.first.b = true;
if (DEBUG) cout << "\nExiting Circle::intersection_points(const Circle&)." \n \n flush;
return bgf;
}
else if (dist \equiv max(radius, c.radius) - min(radius, c.radius)) {
  if (verbose \verb+\+ DEBUG) cout << "\This, and, c, have a, tangent, on, the, inside,\n" \n"\n\nOne, intersection, point, .\n";
if (radius > c.radius) {
    if (verbose \verb+\+ DEBUG) cout << "\c, lies, within, \This, .\n";
    bgf.first.pt.set(c.center - center);
  }
  else {
    if (verbose \verb+\+ DEBUG) cout << "\This, lies, within, c, \n";
    bgf.first.pt.set(center - c.center);
  }
bgf.first.pt.unit_vector(true);
bgf.first.pt *= radius;
bgf.first.pt.shift(center);
bgf.first.b = true;
if (DEBUG) cout << "\nExiting Circle::intersection_points(const Circle&)." \n \n flush;
return bgf;
}
else if (dist < (radius + c.radius)) {
  if (dist > max(radius, c.radius) - min(radius, c.radius)) {

}}
if (verbose || DEBUG) cout << "t<sub>c</sub> and<sub>c</sub> have<sub>c</sub> 2<sub>c</sub> intersections.\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_1 = \n" << 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("P");
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 << "\nExiting Circle::intersection_points(const Circle&)." << 
    endl << endl << flush;
return bpq;
else if (radius > c.radius) {
    if (verbose || DEBUG)
        cout << "c<sub>c</sub> lies within *this<sub>c</sub>, different<sub>c</sub> centers," << "\n(no<sub>c</sub> intersections).\n" << "Returning INVALID_BOOL_POINT_QUADRUPEL.\n";
    if (DEBUG) cout << "\nExiting Circle::intersection_points(const Circle&)." << 
        endl << endl << flush;
    return INVALID_BOOL_POINT_QUADRUPEL;
} else {
    if (verbose || DEBUG)
        cout << "*this<sub>c</sub> lies within c<sub>c</sub>, different<sub>c</sub> centers," << "\n(no<sub>c</sub> intersections).\n" << "Returning INVALID_BOOL_POINT_QUADRUPEL.\n";
    if (DEBUG) cout << "\nExiting Circle::intersection_points(const Circle&)." << 
        endl << endl << flush;
    return INVALID_BOOL_POINT_QUADRUPEL;
} /* else if (dist < (radius + c.radius)) */
else {
    cerr << "ERROR! In Circle::intersection_points(const Circle&):" << 
        "This<sub>c</sub> can't<sub>c</sub> happen! This<sub>c</sub> case isn't<sub>c</sub> accounted for.\n" << "Returning INVALID_BOOL_POINT_QUADRUPEL." << endl << flush;
if (DEBUG) cout << "\nExiting::intersection_points(const Circle&)." << endl << endl;
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.web. 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::medial() 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! In::Reg_Polygon::in_circle():\n        Returning, empty.Circle.\n    " << flush;
    return c;
  }
  if (points.size() < 3) {
    cerr << "ERROR! In::Reg_Polygon::in_circle():\n        Reg_Polygon has less than 3 Points.\n    " << flush;
    return c;
  }
  Point mid_pt = points[0].medial(*points[1]);
  mid_pt -= center;
  real r = mid_pt.magnitude();
  c.set(origin, 2*r);
  Point normal = get_normal();
  normal.shift(center);
  Transform t;
  t.align_with_axis(center, normal, 'y');
  c *= t.invert();
  return c;
}
```

1308. **Draw enclosed circle.**
1309. Normal version.

(Define Reg_Polygon functions 1071) +≡

Circle Reg_Polygon:: draw_in_circ (const Color &draw_color, const string dashed, const string ppen, Picture &picture) const
{
    Circle c = in_circ();
    c.draw (draw_color, dashed, ppen, picture);
    return c;
}

1310. Picture argument first.

(Define Reg_Polygon functions 1071) +≡

Circle Reg_Polygon:: draw_in_circ (Picture &picture, const Color &draw_color, const string dashed, const string ppen) const
{
    return draw_in_circ (draw_color, dashed, ppen, picture);
}

1311. Surrounding circle.

[Log - added this function.]

(Define Reg_Polygon functions 1071) +≡

Circle Reg_Polygon:: out_circ () const
{
    Circle c;
    if (¬is_planar()) {
        cerr << "ERROR! in Reg_Polygon::out_circ():\n" << "Reg_Polygon is non-planar.\n" << "Returning empty Circle.\n" << flush;
        return c;
    }
    if (points.size () < 3) {
        cerr << "ERROR! in Reg_Polygon::out_circ():\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 + \radius);
    Transform t;
    t.align_with_axis (center, normal, 'y');
    c == t.inverse ();
    return c;
}

1312. Draw surrounding circle.

Log

[LDF 2003.06.13] Added this function.

(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

[LDF 2003.06.13] Added this function.

(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.

\{circles.h 1316\} \equiv
\begin{verbatim}
(Define class Circle 1276)
(Declare non-member template functions for Circle 1286)
\end{verbatim}

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.

\begin{verbatim}
(Log)
(LDF 2002.09.21.) Started using this file again. Moved \texttt{hex\_pattern\_l}() here. Made the appropriate changes to cmplprsp.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 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.
\end{verbatim}

\begin{verbatim}
\{Version control identifier 5\} \equiv
static string \texttt{rcs\_id} = "$Id:\_patterns.web,v,1.4,2004/01/12,12:31:19,1finstead,Exp,$";
\end{verbatim}

1318. Include files.

\begin{verbatim}
(Include files 6 \equiv
#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"
#include "planes.h"
#include "paths.h"
#include "curves.h"
#include "polygons.h"
#include "rectangs.h"
#include "ellipses.h"
#include "circles.h"
\end{verbatim}

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.

\begin{verbatim}
(Log)
(LDF 2002.09.22.) Added arguments. \texttt{hex\_pattern\_l}() 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. \texttt{hex\_pattern\_l}() 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
\end{verbatim}
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.

(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) ≡

    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: \n" ≡ first_row ≡ "It must be strictly positive.\n" ≡ "Taking absolute value.\n" ≡ flush;
        first_row = abs(first_row);
        cerr ≡ "Now, first_row = \n" ∈ 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: \n" ≡ first_row ≡ "It can be at most 25.\n" ≡ "Setting first_row to 25.\n" ≡ flush;
        first_row = 25;
    }
    if (fill_color_outer ≡ *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 fill_color, and it’s better to be able to use “” than to have to type “background”. */
        fill_color_outer = *Colors:: background_color;
    if (fill_color_middle ≡ *Colors:: default_color) fill_color_middle = *Colors:: background_color;
    if (fill_color_inner ≡ *Colors:: default_color) fill_color_inner = *Colors:: background_color;

    bool do_middle;    /* [LDF 2002.09.22.] Having do_middle and do_inner 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 do_inner;

    do_middle = (diameter_middle ≡ 0) ? false : true;
    do_inner = (diameter_inner ≡ 0) ? false : true;

    Point pt0;    /* origin. */
    Reg_Polygon p_outer(pt0, 6, diameter_outer);
    Reg_Polygon p_middle;
    Reg_Polygon p_inner;    /* [LDF 2002.09.22.] The middle and inner hexagons are only set (and used) if do_middle and/or do_inner are true. */
    if (do_middle) p_middle.set(pt0, 6, diameter_middle);
    if (do_inner) p_inner.set(pt0, 6, diameter_inner);
    Reg_Polygon p_outer_copy;    /* These Reg_Polygons are used for copying. */
Reg_Polygon p_middle_copy;
Reg_Polygon p_inner_copy;
#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 lines. [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;
    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(drow_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(drow_color_outer, fill_color_outer, "", pen_outer, picture);
    if (do_middle) {
        if (fill_color_middle == fill_color_outer)
            p_middle_copy.fill(drow_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(drow_color_middle, fill_color_middle, "", pen_middle, picture);
    }
    if (do_inner) {
        if (fill_color_inner == fill_color_middle)
            p_inner_copy.fill(drow_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(drow_color_inner, fill_color_inner, "", pen_inner, picture);
    }
    ++hexagon_cotr;
    if (hexagon_cotr >= max_hexagons) {
        cerr << "ERROR! In hex_pattern_1():\n" << "Too many sets of hexagons:\n" << hexagon_cotr << "...Returning.\n" << endl << flush;
        return hexagon_cotr;
    }
    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_cotr;

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.

---

[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) +≡

```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 = Colors::default_color_vector, int arc_divisions = 72);
```
1324.
〈Define Pattern functions 1321〉 \[ \ni \]

\begin{verbatim}
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!\ndiameter_inner.<\ndiameter_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!\noffsets has invalid value:\n" << offsets << endl <<
    "offsets must be >= 0.\nSetting 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() \ni 0) colors.push_back(default_color);
    for (unsigned int i = 0; i < offsets; ++i) {
      if (i \i 0) outer_circle_center += normal += p0 += outer_circle.rotate(0, 360.0/offsets);
        else {
          
        
    }
  }

Note: The code snippet provided is a representation of the EPICYCLID_PATTERN_1 function. It includes the necessary checks, calculations, and loop structures to generate a pattern based on the given parameters. The code handles cases where the diameter inner is smaller than the diameter outer start, and it also handles the case where the offsets are less than 1. The function iterates through the given colors, adjusting them as necessary for the pattern generation. All necessary imports are assumed to be included in the context of the code snippet.
\end{verbatim}

3DLDF-1.1.5.1 §1324
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_center *= temp_circle.rotate(0, theta);
    p2.rotate(temp_circle_center, temp_circle_normal, phi);
    spiral += p2;
    if (DEBUG) {
        p2.dotlabel("p2", "1ft");
        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 \texttt{epicycloid\_pattern\_1()}, except that the offsets are not made by rotating the outer \texttt{Circle} around the center of the inner \texttt{Circle}, but by rotating the \texttt{Point} used for tracing the epicycloid about the center of the outer \texttt{Circle}.

1326. **Epicycloid pattern 3.** [LDF 2003.02.11.] This function works well for outer \texttt{Circles} with radii that are not even divisors of the radius of the inner \texttt{Circle}.

---

[LDF 2003.02.11.] Added this function.
[LDF 2003.08.27.] Removed the declaration \texttt{real radius\_ratio = radius\_outer/radius\_inner}, since \texttt{radius\_ratio} was never used.

(Declare Pattern functions 1320) +\equiv

\begin{verbatim}
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);
\end{verbatim}
1327.
(Define Pattern functions 1321) +≡
unsigned int epicycloid_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;
  Path start_pt (p0);
  unsigned int spiral_counter = 1;
  unsigned int iter_ctr = 0;
  while (true) {
    if (theta_total ≥ 360) {
      cout ≡ "theta_total -= \n" ≡ theta_total ≡ endl ≡ "Reducing, theta_total from \n" ≡
      (theta_total - 360) ≡ 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=-\n" ≡ iter_ctr ≡ endl ≡ "Breaking, \n\n" ≡ flush;
      break;
    }
    else if (iter_ctr > 0 ∧ (fmod((iter_ctr * radius_outer), radius_inner) ≡ 0)) {
      cout ≡ "came out, even.\n" ≡ "iter_ctr -= \n" ≡ iter_ctr ≡ endl ≡
      "fmod((iter_ctr * radius_outer), radius_inner) -= \n" ≡ fmod((iter_ctr * radius_outer),
      radius_inner) ≡ endl ≡ "Breaking, \n\n" ≡ 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)


---

[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.1.5.2004/01/12,21:33:23,1finsto1,Exp,6;";

1332. Include files.
(include files 6) +≡
#include "loader.h"
#include "pspglb.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"
#include "polygons.h"
#include "rectangs.h"
#include "ellipses.h"
#include "circles.h"

1333. Solid class definition.

---

[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 to 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  3DLD-1.1.5.1  SOLID CLASS DEFINITION  395

(Define class Solid 1333) ≡
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.

[LDF 2003.04.11.] Added this section.

(Define static const Solid data members 1334) ≡
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) ≡
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 ) ≡
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 ) +≡
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) );
  }
}

1340. Pseudo-constructor for dynamic allocation.

1341. Pointer argument.

[Log]
[LDF 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 *c);

See also section 1342.
This code is used in sections 1441 and 1442.

1342. Reference argument.

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

(Declare non-member template functions for Solid 1341) +≡
Solid *create_new(const Solid &c);

1343. Destructor.

[Log]
[LDF 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 \} +\equiv  
Solid::~Solid()  
\} 

1345.  Assignment.  
\{Declare Solid functions 1336 \} +\equiv  
virtual const Solid &operator=(const Solid &s); 

1346.  
\{Define Solid functions 1337 \} +\equiv  
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();  
  if (circles.size() > 0)  
    for (vector(Circle *): iterator = circles.begin(); iter != circles.end(); ++iter)  
      (**iter).clear();  
  if (ellipses.size() > 0)  
    for (vector(Ellipse *): iterator = ellipses.begin(); iter != ellipses.end(); ++iter)  
      (**iter).clear();  
  if (reg_polygons.size() > 0)  
    for (vector(Reg_Polygon *): iterator = reg_polygons.begin(); iter != reg_polygons.end(); ++iter)  
      (**iter).clear();  
  if (rectangles.size() > 0)  
    for (vector(Rectangle *): iterator = rectangles.begin(); iter != rectangles.end(); ++iter)  
      (**iter).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) \( \equiv \)

Path *\texttt{p} for (vector(Path *)::const_iterator iter = s.paths.begin(); iter \neq 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 \neq s.circles.end(); ++iter) { circles.push_back ( create_new < Circle > (0) );
*(circles.back()) = **iter; } for (vector(Ellipse *)::const_iterator iter = s.ellipses.begin();
  iter \neq 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 \neq 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 \neq 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 \text{"paths.size()"} \ll \text{endl} \ll \text{flush};
  cout \ll \text{"circles.size()"} \ll \text{endl} \ll \text{flush};
  cout \ll \text{"ellipses.size()"} \ll \text{endl} \ll \text{flush};
  cout \ll \text{"reg_polygons.size()"} \ll \text{endl} \ll \text{flush};
  cout \ll \text{"rectangles.size()"} \ll \text{endl} \ll \text{flush};
}
if (DEBUG) cout \ll \text{"Exiting.Solid::operator-() \n"};
return *this; }

1348. Copying.

[Log][LDF 2003.06.06] BUG FIX. Changed \( s \) from \texttt{Shape * to Solid *}. I noticed this bug when I tried to copy a \texttt{Picture} containing a \texttt{Cuboid}, and the copy contained a single empty \texttt{Shape * on shapes}.

(Declare Solid functions 1336) \( \equiv \)

virtual Shape *\texttt{get_copy()} const;

1349. (Define Solid functions 1337) \( \equiv \)

Shape *\texttt{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) \( \equiv \)

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.

[Log]

[LDF 2003.06.06.] 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.

[Log]

[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.
§1356. 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)

[Log]

[LDF 2003.04.11.] Added this function.
[LDF 2003.04.30.] Now using get_copy() 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;
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" << s << ") > circles.size()(" << circles.size() << ")\nReturning a null pointer\n\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" << s << ") > ellipses.size()(" << ellipses.size() << ")\nReturning a null pointer\n\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" << s << ") > paths.size()(" << paths.size() << ")\nReturning a null pointer\n\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" << s << ") > rectangles.size()(" << rectangles.size() << ")\nReturning a null pointer\n\n" << flush;
            return static_cast<Shape*>(0);
        }
    } else if (shape_type == REG_POLYGON) {
        if (s < reg_polygons.size()) {
return reg_polygons[s]->getCopy();
}
else {
    cerr << "ERROR! In Solid::get_shape_ptr():\n" << "s_" << s_ << ", reg_polygons.size()_" << reg_polygons.size() << "\nReturning a null pointer(0).\n\n" << flush;
    return static_cast(Shape *)(0);
}

1358. Get Circle pointer.

<table>
<thead>
<tr>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>[LDF 2003.04.11.] Added this function.</td>
</tr>
<tr>
<td>[LDF 2003.04.30.] Changed this function, so that it no longer uses get_shape_ptr().</td>
</tr>
</tbody>
</table>

(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.

<table>
<thead>
<tr>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>[LDF 2003.04.11.] Added this function.</td>
</tr>
<tr>
<td>[LDF 2003.04.30.] Changed this function, so that it no longer uses get_shape_ptr().</td>
</tr>
</tbody>
</table>

(Declare Solid functions 1336) +≡

virtual const Ellipse *get_ellipse_ptr(const unsigned short s) const;
1361. \[(\text{Define Solid functions 1337 }) \equiv\]
\begin{verbatim}
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);
    }
}
\end{verbatim}

1362. Get Path pointer.

\begin{verbatim}
<table>
<thead>
<tr>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>[LDF 2003.04.11.] Added this function.</td>
</tr>
<tr>
<td>[LDF 2003.04.30.] Changed this function, so that it no longer uses get_shape_ptr().</td>
</tr>
</tbody>
</table>
\end{verbatim}

\[(\text{Declare Solid functions 1336 }) \equiv\]
\begin{verbatim}
virtual const Path *get_path_ptr(const unsigned short s) const;
\end{verbatim}

1363. \[(\text{Define Solid functions 1337 }) \equiv\]
\begin{verbatim}
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);
    }
}
\end{verbatim}

1364. Get Rectangle pointer.

\begin{verbatim}
<table>
<thead>
<tr>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>[LDF 2003.04.11.] Added this function.</td>
</tr>
<tr>
<td>[LDF 2003.04.30.] Changed this function, so that it no longer uses get_shape_ptr().</td>
</tr>
</tbody>
</table>
\end{verbatim}

\[(\text{Declare Solid functions 1336 }) \equiv\]
\begin{verbatim}
virtual const Rectangle *get_rectangle_ptr(const unsigned short s) const;
\end{verbatim}
1365.  
<Define Solid functions 1337> \[=\]
  const Rectangle *Solid::get_rectangle_ptr(const unsigned short s) const
  {
    if (rectangles.size() > s) {
      return rectangles[s];
    } else {
      return static_cast(const Rectangle *)(0);
    }
  }

1366.  Get Reg_Polygon pointer.

---

[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 Reg_Polygon *get_reg_polygon_ptr(const unsigned short s) const;

1367.  
<Define Solid functions 1337> \[=\]
  const Reg_Polygon *Solid::get_reg_polygon_ptr(const unsigned short s) const
  {
    if (reg_polygons.size() > s) {
      return reg_polygons[s];
    } else {
      return static_cast(const Reg_Polygon *)(0);
    }
  }

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.

---

[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!

---

Log

[LD 2002.10.16.] Added this function.
[LD 2003.04.09.] Moved this function from Polyhedron to Solid.
[LD 2003.04.11.] Changed this function from get_polygon_center() to get_shape_center(). Added the char argument shape_type to indicate whether it should return the center of a Circle, Ellipse, Rectangle, or Reg_Polygon.
[LD 2003.04.11.] Changed the shape_type argument from char to const unsigned short.

(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() << ") Returning 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() << ") Returning 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() << ") Returning 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" << "Invalid argument for shape_type: \n" << shape_type << endl << "Returning INVALID_POINT.\n\n" << flush;
        return INVALID_POINT;
    }
}
1371. Get Circle center.

[Log]

[Declare Solid functions 1336] +≡
  virtual const Point &get_circle_center(const unsigned short s) const;

1372.

[Define Solid functions 1337] +≡
  const Point &Solid::get_circle_center(const unsigned short s) const
  {
    return get_shape_center(CIRCLE, s);
  }

1373. Get Ellipse center.

[Log]

[Declare Solid functions 1336] +≡
  virtual const Point &get_ellipse_center(const unsigned short s) const;

1374.

[Define Solid functions 1337] +≡
  const Point &Solid::get_ellipse_center(const unsigned short s) const
  {
    return get_shape_center(ELLIPSE, s);
  }

1375. Get Rectangle center.

[Log]

[Declare Solid functions 1336] +≡
  virtual const Point &get_rectangle_center(const unsigned short s) const;

1376.

[Define Solid functions 1337] +≡
  const Point &Solid::get_rectangle_center(const unsigned short s) const
  {
    return get_shape_center(RECTANGLE, s);
  }
1377. **Get Reg\_Polygon center.**

[Log]

[Declaret Solid functions 1336] +\equiv
\begin{verbatim}
    virtual const Point \& get_reg_polygon_center(const unsigned short s) const;
\end{verbatim}

1378.

[Define Solid functions 1337] +\equiv
\begin{verbatim}
    const Point \&Solid::get_reg_polygon_center (const unsigned short s) const
    {
        return get_shape_center (REG_POLYGON, s);
    }
\end{verbatim}

1379. **Is on free store.**

[Declaret Solid functions 1336] +\equiv
\begin{verbatim}
    virtual bool is_on_free_store() const;
\end{verbatim}

1380.

[Define Solid functions 1337] +\equiv
\begin{verbatim}
    bool Solid::is_on_free_store() const
    {
        bool b = true;
        return b;
    }
\end{verbatim}

1381. **Show.**

[Declaret Solid functions 1336] +\equiv
\begin{verbatim}
    virtual 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
dfactor = 1) const;
\end{verbatim}
1382.

\{Define Solid functions 1337\} +=

```cpp
void Solid::show(string text, char coors, 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.<<" << 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(), coors, do_persp, do_apply, f, proj, factor);
            g.str("\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(), coors, do_persp, do_apply, f, proj, factor);
            g.str("\n");
        }
    }
    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(), coors, do_persp, do_apply, f, proj, factor);
            g.str("\n");
        }
    }
    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(), coors, do_persp, do_apply, f, proj, factor);
            g.str("\n");
        }
    }
    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(Reg_Polygon *):::iterator iter = reg_polygons.begin(); iter != reg_polygons.end(); ++iter) (**iter).clear();
    reg_polygons.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
\[
\text{Transform Solid::operator\&\& (const Transform &t)}
\]
\[
\begin{array}{l}
\text{center \&\& t; } \\
\text{for (vector(Path \&): \&\& iter = paths.begin(); iter \neq paths.end(); ++iter)} \text{ \&\& iter \&\& t; } \\
\text{for (vector(Ellipse \&): \&\& iter = ellipses.begin(); iter \neq ellipses.end(); ++iter)} \text{ \&\& iter \&\& t; } \\
\text{for (vector(Circle \&): \&\& iter = circles.begin(); iter \neq circles.end(); ++iter)} \text{ \&\& iter \&\& t; } \\
\text{for (vector(Reg_Polygon \&): \&\& iter = reg_polygons.begin(); iter \neq reg_polygons.end(); } \\
\text{++iter)} \text{ \&\& iter \&\& t; } \\
\text{for (vector(Rectangle \&): \&\& iter = rectangles.begin(); iter \neq rectangles.end(); ++iter)} \text{ \&\& iter \&\& t; }
\end{array}
\]
\[
\text{return t;}
\]

1388.  Applying a transformation.

Log

[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".

1389.  Define Solid functions 1337
\[
\text{virtual void apply_transform (void);}
\]

1390.  Scale.
\[
\text{virtual Transform scale(real xx, real yy = 0, real zz = 0);}
\]
§1391. 3DLDF-1.1.5.1

1391. 
\{ Define Solid functions 1337 \} +≡
Transform Solid:: scale(real\ xx, real\ yy, real\ zz)
\{
    Transform t;
    t.scale(xx, yy, zz);
    \*this *= t;
    return t;
\}

1392. Shear.
\{ Declare Solid functions 1336 \} +≡
virtual Transform shear(real\ xy, real\ xx = 0, real\ yx = 0, real\ yz = 0, real\ zx = 0, real\ zy = 0);

1393. 
\{ Define Solid functions 1337 \} +≡
Transform Solid:: shear(real\ xy, real\ xx, real\ yx, real\ yz, real\ zx, real\ zy)
\{
    Transform t;
    t.shear(xy, xx, yx, yz, zx, zy);
    \*this *= t;
    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.shift(xx, yy, zz);
    \*this *= t;
    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.  Rotation 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.  Rotation around an arbitrary axis.

[LD/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 &p0, const Point &p1, const real angle = 180);

1402.  
(Define Solid functions 1337) +≡

Transform Solid:: rotate(const Point &p0, const Point &p1, const real angle)
{
    Transform t;
    t.rotate(p0, 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.
\( \text{Define Solid functions 1337} \) $$
\text{vector(Shape *) Solid::extract(const Focus &f, const unsigned short proj, real factor)} \n\{ \text{vector(Shape *) v;} \n\text{for (vector(Path *)::iterator iter = paths.begin(); iter \neq paths.end(); ++iter) \{ \} \text{apply_transform();} \n\text{if (-(*(*iter).project(f, proj, factor)) \{ \} \text{ERR \"WARNING! In Solid::extract():\n" \"Path cannot be projected.\u" \text{Returning empty vector<Shape*>::\n" \text{flush;}} \n\text{return v;} \n\text{break;} \n\} \text{for (vector(Ellipse *)::iterator iter = ellipses.begin(); iter \neq ellipses.end(); ++iter) \{ \} \text{apply_transform();} \n\text{if (-(*(*iter).project(f, proj, factor)) \{ \} \text{ERR \"WARNING! In Solid::extract():\n" \"Ellipse cannot be projected.\u" \text{Returning empty vector<Shape*>::\n" \text{flush;}} \n\text{return v;} \n\text{break;} \n\} \text{for (vector(Circle *)::iterator iter = circles.begin(); iter \neq circles.end(); ++iter) \{ \} \text{apply_transform();} \n\text{if (-(*(*iter).project(f, proj, factor)) \{ \} \text{ERR \"WARNING! In Solid::extract():\n" \"Circle cannot be projected.\u" \text{Returning empty vector<Shape*>::\n" \text{flush;}} \n\text{return v;} \n\text{break;} \n\} \text{for (vector(Reg_Polygon *)::iterator iter = reg_polygons.begin(); iter \neq reg_polygons.end(); ++iter) \{ \} \text{apply_transform();} \n\text{if (-(*(*iter).project(f, proj, factor)) \{ \} \text{ERR \"WARNING! In Solid::extract():\n" \"Polygon cannot be projected.\u" \text{Returning empty vector<Shape*>::\n" \text{flush;}} \n\text{return v;} \n\text{break;} \n\} \text{for (vector(Rectangle *)::iterator iter = rectangles.begin(); iter \neq rectangles.end(); ++iter) \{ \} \text{apply_transform();} \n\text{if (-(*(*iter).project(f, proj, factor)) \{ \} \text{ERR \"WARNING! In Solid::extract():\n" \"Rectangle cannot be projected.\u" \text{Returning empty vector<Shape*>::\n" \text{flush;}} \n\text{return v;} \n\text{break;} \n\} \text{v.push_back(this);} \n\}
1406. Set extremes.

(Declare \texttt{Solid} functions 1336) \equiv

\begin{verbatim}
virtual bool set_extremes();
\end{verbatim}
```cpp
1407.  
(Define Solid functions 1337) 

```}bool Solid::setExtremes()
```cpp
{  
    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 gwdu01.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]);
        }
```
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::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(Rectangle *):::iterator iter = rectangles.begin(); iter != rectangles.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]);
    }
}

if (DEBUG) cout << "Exiting Solid::set_extremes()" << endl << flush;
return true;
}

1408. Get extremes.
(Declare Solid functions 1336) +≡
inline virtual const varray(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 << "Entering Solid::get_minimum_z() \n"   << endl << flush;
        cout << "Exiting Solid::get_minimum_z() \n"    << 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 << "Entering Solid::get_maximum_z() \n"   << endl << flush;
        cout << "Exiting Solid::get_maximum_z() \n"    << endl << flush;
    }
    return projective_extremes[5];
}

1413.  Get mean z.  [LDF 2003.05.16] 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 \} +≡
\vspace{1mm}
\begin{verbatim}
void Solid::suppress_output()
{
    do_output = false;
    return;
}
\end{verbatim}

1417.  Unsuppress output.
\{ Declare Solid functions 1336 \} +≡
\vspace{1mm}
\begin{verbatim}
virtual void unsuppress_output();
\end{verbatim}

1418.  
\{ Define Solid functions 1337 \} +≡
\vspace{1mm}
\begin{verbatim}
void Solid::unsuppress_output()
{
    do_output = true;
    return;
}
\end{verbatim}

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(Solid *) 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 \} +≡
\vspace{1mm}
\begin{verbatim}
virtual void output();
\end{verbatim}
1420.
(Define Solid functions 1337 ) \( \equiv \)

```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());
    for (vector<Rectangle *>::iterator iter = rectangles.begin(); iter != rectangles.end(); ++iter)
        s.push_back((**iter).get_copy());
    sort(s.begin(), s.end(), Compare_maximum_x());
    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, *and* 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 `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().u-=u" << color_ptr->get_use_name() << endl << flush;
        }

        if (color_ptr->get_use_name() == false) {
            Color *c = create_new < Color>(0);
            *c = *color_ptr;
            (**iter).set_draw_color(c);
        } else {
            if (DEBUG) cout << "color_ptr->get_name().u-=u" << 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.

```cpp
(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 ) $\equiv$

```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 */
c_iter = v.begin();
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 it's used, `iter` is an iterator for a different vector.

---

[LDF 2003.08.10.] Now setting pen to "", because I've removed the pen argument from `fill()`. 

---

```cpp
(Process vectors for `fill()` 1425) ≡
{
  if (c_iter != v.end()) {
    color_ptr = *c_iter++;
  }
  (*iter).set_fill_draw_value(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).set_fill_color(c);
  } else {
    if (DEBUG) cout << "color_ptr->get_name()=" << color_ptr->get_name() << endl << flush;
    (*iter).set_fill_color(color_ptr);
  }
  (*iter).set_draw_color(static_cast<Color*>(0));
  (*iter).set_dash_pattern("");
  (*iter).set_pen("");
} 

This code is used in section 1427.

1426. Fill.

---

[LDF 2003.08.10.] Removed pen argument, since filling doesn't use a pen. 

---

(Declare `Solid` functions 1336) ≡

```cpp
virtual void fill(const vector(const Color*) v = Colors::default_color_vector, Picture &picture = current_picture) const;
```
1427.

\[ \text{Define Solid functions } 1337 \implies \]
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 c_iter = v.begin();
  for (vector<Path *>::const_iterator s_iter = s-paths.begin(); s_iter != s-paths.end(); ++s_iter) {
    (Process vectors for fill() 1425)
  }
  c_iter = v.begin();
  for (vector<Circle *>::const_iterator s_iter = s-circles.begin(); s_iter != s-circles.end(); ++s_iter) {
    (Process vectors for fill() 1425)
  }
  c_iter = v.begin();
  for (vectorEllipse *>::const_iterator s_iter = s-ellipses.begin(); s_iter != s-ellipses.end(); ++s_iter) {
    (Process vectors for fill() 1425)
  }
  c_iter = v.begin();
  for (vectorReg_Polygon *>::const_iterator s_iter = s-reg_polygons.begin(); s_iter != s-reg_polygons.end(); ++s_iter) {
    (Process vectors for fill() 1425)
  }
  c_iter = v.begin();
  for (vectorRectangle *>::const_iterator s_iter = s-rectangles.begin(); s_iter != s-rectangles.end(); ++s_iter) {
    (Process vectors for fill() 1425)
  }
  picture += dynamic_cast< Shape * >(s);
  if (DEBUG) {
    cout << "Exiting Solid::fill():" << endl << flush;
  }
}
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 {
    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 {
    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. 3DLDF-1.15.1  FILLDRAW  427

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():\n" << 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():\n" << 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.

\[
\begin{align*}
\text{Process vectors for undraw() 1431} & \equiv \\
& \{ \\
& \quad (\ast \text{iter}).\text{set_fill_draw_value(UNDRAW)}; \\
& \quad (\ast \text{iter}).\text{set_fill_color(static_cast(Color *)(0))}; \\
& \quad (\ast \text{iter}).\text{set_dash_pattern(ddashed)}; \\
& \quad (\ast \text{iter}).\text{set_pen(ppen)}; \\
& \} \\
\text{This code is used in section 1433.}
\end{align*}
\]

1432. Undraw.

\[
\begin{align*}
\text{Declare Solid functions 1336} & \equiv \\
& \text{virtual void undraw(const string ddashed = "", const string ppen = ", Picture &picture = current\_picture) const;}
\end{align*}
\]

1433.

\[
\begin{align*}
\text{Define Solid functions 1337} & \equiv \\
& \text{void Solid::undraw(const string ddashed, const string ppen, Picture &picture) const \{ bool DEBUG = false; /* true */}
& \quad \text{if (DEBUG) cout \"Entering\_Solid::undraw(): \n \\ll flush;}
& \quad Solid *s = create\_new < Solid > (0);
& \quad *s = \ast this;
& \quad \text{for (vector(Path *)::const\_iterator iter = s\_paths.begin(); iter \ne s\_paths.end(); ++iter) \{}
& \quad \quad \text{Process vectors for undraw() 1431}
& \quad \} \\
& \quad \text{for (vector(Circle *)::const\_iterator iter = s\_circles.begin(); iter \ne s\_circles.end(); ++iter) \{}
& \quad \quad \text{Process vectors for undraw() 1431}
& \quad \} \\
& \quad \text{for (vector(Ellipse *)::const\_iterator iter = s\_ellipses.begin(); iter \ne s\_ellipses.end(); ++iter) \{}
& \quad \quad \text{Process vectors for undraw() 1431}
& \quad \} \\
& \quad \text{for (vector(Reg\_Polygon *)::const\_iterator iter = s\_reg\_polygons.begin(); iter \ne s\_reg\_polygons.end(); ++iter) \{}
& \quad \quad \text{Process vectors for undraw() 1431}
& \quad \} \\
& \quad \text{for (vector(Rectangle *)::const\_iterator iter = s\_rectangles.begin(); iter \ne s\_rectangles.end(); ++iter) \{}
& \quad \quad \text{Process vectors for undraw() 1431}
& \quad \} \\
& \quad \text{picture += dynamic\_cast(Shape *)(s);}
& \quad \text{if (DEBUG) \{}\\
& \quad \quad \text{cout \"Exiting\_Solid::undraw(): \n \ll flush;}
& \quad \} \\
& \}
\end{align*}
\]
1434. Process vectors for unfill().

[Log: LDF 2002.10.02.] 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.

[Log: LDF 2003.08.10.] Now setting pen to "", since I've removed the pen argument to unfill().

```cpp
(Process vectors for unfill() 1434) \equiv
{
    (**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) +\equiv
    virtual void unfill(Picture &picture = current_picture) const;
```
1436.  
\texttt{(Define Solid functions 1337 )} \equiv
\begin{verbatim}
void Solid::unfill(Picture &picture) const { bool DEBUG = false;  /* true */
  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;
  }
}
\end{verbatim}

1437.  Process vectors for \texttt{unfilldraw()}.  \texttt{[LDF 2002.10.09.]}  Added this section. The same things are done to each of the \texttt{Shape *} vectors \texttt{paths, ellipses, circles, reg_polygons,} and \texttt{rectangles,} so I've put the code in this named section. Each time it's used, \texttt{iter} is an iterator for a different vector.

\texttt{(Process vectors for unfilldraw() 1437) } \equiv
\begin{verbatim}
  \{ (*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(dashed);
  (*iter).set_pen(ppen);
\}
\end{verbatim}

This code is used in section 1439.

1438.  Unfilldraw.  \texttt{[LDF 2002.10.09.]}  Unlike \texttt{Path::unfilldraw()}, \texttt{Solid::unfilldraw()} behaves like METAPOST's \texttt{unfilldraw} command, i.e., it unfills and undraws. I intend to change \texttt{Path::unfilldraw()} so that it also behaves this way.

\texttt{[LDF 2002.10.09.]}  Check this: the correct code is written to \texttt{out\_stream}, but after \texttt{filldraw()} and \texttt{unfilldraw()}, the outline is visible, but no lines inside the outline.  TO DO: Check what \texttt{filldraw} and \texttt{unfilldraw} mean in METAPOST and METAFONT.

\texttt{(Declare Solid functions 1336 )} \equiv
\begin{verbatim}
virtual void unfilldraw(const string dashed = "", const string ppen = "", Picture &picture = current\_picture) const;
\end{verbatim}
1439.  
\{Define Solid functions \}  
\begin{verbatim}
void Solid::unfilldraw(const string ddashed, const string pen, Picture &picture) const {
    bool DEBUG = false;  // true */
    if (DEBUG) cout << "Entering Solid::unfilldraw()" << endl;
    Solid *s = create_new < Solid > (0);
    for (vector(Path *): const_iterator iter = s->paths.begin(); iter \ne s->paths.end(); ++iter) {
        (Process vectors for unfilldraw())
    }  // for
    for (vector(Circle *): const_iterator iter = s->circles.begin(); iter \ne s->circles.end(); ++iter) {
        (Process vectors for unfilldraw())
    }  // for
    for (vector(Ellipse *): const_iterator iter = s->ellipses.begin(); iter \ne s->ellipses.end(); ++iter) {
        (Process vectors for unfilldraw())
    }  // for
    for (vector(Reg_Polygon *): const_iterator iter = s->reg_polygons.begin();
        iter \ne s->reg_polygons.end(); ++iter) {
        (Process vectors for unfilldraw())
    }  // for
    picture += dynamic_cast<Shape *>(s);
    if (DEBUG) {
        cout << "Exiting Solid::unfilldraw()" << endl;
    }
}
\end{verbatim}

1440. Putting Solid together.

1441. This is what’s compiled.
\begin{verbatim}
#include <files 6>
#include <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)
\end{verbatim}
1442. This is what’s written to solids.h.

```cpp
(solids.h 1442) ≡
(Define class Solid 1333)
(Declare non-member template functions for Solid 1341)
```

1443. Solid_Faced (solfaced.web).

<table>
<thead>
<tr>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>[LDF 2002.09.26.] Created this file.</td>
</tr>
<tr>
<td>[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.</td>
</tr>
<tr>
<td>[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.</td>
</tr>
</tbody>
</table>

```cpp
(Version control identifier 5) +≡
static string res_id = "$Id:solfaced.web,v,1.4,u,2004/01/12,21:33:15,u,lfinsto1,Exp,$";
```

1444. Include files.

```cpp
(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 "rectangls.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.

```cpp
(Declare Solid_Faced functions 1446) ≡
```

This code is used in section 1445.
§1447. 3DLDF-1.1.5.1

**SOLID_FACED CLASS DEFINITION**  433

1447.  Define `Solid_Faced` functions 1447

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 “Quadraer”, 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 Quadraer 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 quadraer.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 quadraer.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 rcs_id = "$/Id: cuboid.web,v,1.6,2004/01/12,21:27:51,lfinst01,Exp,1";

1452. Include files.

{Include files 6} ≡
#include "loader.h"
#include "psglb.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"
#include "polygons.h"
#include "rectangs.h"
#include "ellipses.h"
#include "circles.h"
#include "solids.h"
#include "solfaced.h"

1453. Cuboid class definition.
(Define class Cuboid 1453) ≡
    class Cuboid : public Solid_Faced {
    protected: real height;
    real width;
    real depth;
    public: {Declare Cuboid functions 1455}
    }

This code is used in sections 1468 and 1469.

1454. Constructors and setting functions.

(Declare Cuboid functions 1455) ≡
    Cuboid();

See also sections 1457, 1459, 1464, and 1466.
This code is used in section 1453.

1456.
(Define Cuboid functions 1456) ≡
    Cuboid::Cuboid() {
    on_free_store = false;
    do_output = true;
    projective_extremes.resize(6, 0);
    faces = 6;
    vertices = 8;
    edges = 12;
    }

See also sections 1458, 1460, 1465, and 1467.
This code is used in section 1468.

1457. Copy constructor. [LDF 2002.05.03.] Added this function.
(Declare Cuboid functions 1455) +≡
    Cuboid(const Cuboid &c);
1458.

{Define Cuboid functions 1456} \(1456\) \(1458\)

\[\text{Cuboid} :: \text{Cuboid}(\text{const Cuboid} \ &c)\{ \text{on_free\_store = false;} \]
\[\text{do_output = true;}
\text{projective\_extremes\_resize}(6, 0);
\text{faces = 6;}
\text{vertices = 8;}
\text{edges = 12; for (vector(\text{Rectangle} \ *)\:: \text{const\_iterator \ iter = c.rectangles\_begin();}
\text{iter \neq c.rectangles\_end(); \ iter++} \} \text{rectangles\_push\_back (create\_new < \text{Rectangle} > (0));}
\text{*(rectangles\_back()) = ***iter; } \} \]  

1459.  Center, height, width, depth, and angles.  [LDF 2002.10.06]  Added this constructor.

{Declare Cuboid functions 1455} \(1455\) \(1459\)

\[\text{Cuboid(\text{const Point} \ &c, \text{const real} \ h, \text{const real} \ w, \text{const real} \ d, \text{const real} \ x = 0, \text{const real} \ y = 0, \text{const real} \ z = 0); \]
§1460.  
(Define Cuboid functions 1456) +≡

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);
  }

§1461. Pseudo-constructor for dynamic allocation.

§1462. Pointer argument.

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.

(Declare non-member template functions for Cuboid 1462) ≡

Cuboid *create_new(const Cuboid *c);

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.

```c
#include "cuboid.h"
#include "loader.h"
#include "psglb.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"
#include "polygons.h"
#include "rectangs.h"
#include "ellipses.h"
#include "circles.h"
#include "solids.h"
#include "solfaced.h"
```


[LDF 2002.11.12] TO DO: Add assignment operators for Polyhedra! The individual types will need there 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.

```c
static string rs_id = "$Id: polyhedra.web,v1.5,2004/01/12,21:32:19,1finsto1,Exp,\$";
```

1471. Include files.

```c
#include "loader.h"
#include "psglb.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"
#include "polygons.h"
#include "rectangs.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/gssolfig.texti about abstract or non-abstract base classes.

Log

[LDF 2002.11.08] Got rid of pure virtual function Polyhedron::get_net( ). 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) ≡
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. !! [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) ≡
#if 0
  virtual vector(Point) intersection_points(const Reg_Polygon &r) const;
#endif

This code is used in section 1472.
1474.  
(Define Polyhedron functions 1474) \equiv
\#if 0
  vector<Point> Polyhedron::intersection_points(const Reg_Polygon &r) const
  {
    vector<Point> w;
    for (vector<Reg_Polygon *>::const_iterator iter0 = reg_polygons.begin();
         iter0 != reg_polygons.end();
         ++iter0)
      for (vector<Reg_Polygon *>::const_iterator iter1 = iter0 + 1;
           iter1 != reg_polygons.end();
           ++iter1)
        w = (**iter0).intersection_points(**iter1);
        cout << "v.size()=" << w.size() << endl << flush;
        for (vector<Point>::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.

[LDF 2002.11.12] Added this section.

1477.  Tetrahedron class definition.

[LDF 2002.11.12] Added this section.

format Tetrahedron Polyhedron

(Define class Tetrahedron 1477) \equiv
class Tetrahedron : public Polyhedron {
protected: static const real dihedral_angle;  /* In radians! */
    real triangle_radius;
public: (Declare Tetrahedron functions 1480)
};
This code is used in sections 1534 and 1535.

1478.  Define static const Tetrahedron data members.

[LDF 2002.11.12] Added this section.

(Define static const Tetrahedron data members 1478) \equiv
const real Tetrahedron::dihedral_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.

(Declare Tetrahedron functions 1480) ≡

Tetrahedron();

See also sections 1483, 1487, 1489, and 1491.

This code is used in section 1477.

1481.

(Define Tetrahedron functions 1481) ≡

Tetrahedron::Tetrahedron()
{
    on_free_store = false; // from Solid. */
    do_output = true;
    faces = 4; // from Solid_Faced. */
    vertices = 4;
    edges = 6;
    center = INVALID_POINT; // from Polyhedron. */
    number_of_polygon_types = 1;
    face_radius = edge_radius = vertex_radius = INVALID_REAL;
    triangle_radius = INVALID_REAL; // From Tetrahedron. */
}

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 last in a rudimentary way.

[LDF 2002.08.12.] Rewrote this function. It now works properly.

(Declare Tetrahedron functions 1480) ≡

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) +≡

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
1485. \textit{center} is the intersection point of the line segments from the vertices of \texttt{*(reg\_polygons[0])} to the centers of the opposite faces. \textit{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 \texttt{Tetrahedron}, there’s no need to recalculate it each time a \texttt{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.]

\begin{verbatim}
(Define Tetrahedron functions 1481) +=
real distance = 0.7499788995574951171875;
#endif
Point P0 = Point::intersection_point(pts[0], pts[6], pts[1], pts[5], pt);
P0.show("P0:0,6,1,5: ");
Point::intersection_point(pts[0], pts[6], pts[2], pts[4]).show("0,6,2,4: ");
Point::intersection_point(pts[0], pts[6], pts[3], pts[7]).show("0,6,3,7: ");
Point::intersection_point(pts[1], pts[5], pts[2], pts[4]).show("1,5,2,4: ");
Point::intersection_point(pts[1], pts[5], pts[3], pts[7]).show("1,5,3,7: ");
Point::intersection_point(pts[2], pts[4], pts[3], pts[7]).show("2,4,3,7: ");
Point P1 = pts[5] - pts[1];
Point P2 = P0 - pts[1];
P1.show("P1");
cout << "\nP1.magnitude().\n" << P1.magnitude() << endl << flush;
P2.show("P2");
cout << "\nP2.magnitude().\n" << P2.magnitude() << endl << flush;
distance = (P2.magnitude() / P1.magnitude());
cout.precision(25);
cout << "distance: \n" << 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 \texttt{origin}) {
    center = p;
    for (i = 0; i < 4; ++i) reg\_polygons[i]-shift(p);
}
return; }
\end{verbatim}
1487. Setting function. [LDF 2002.11.12.] !! This works, 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. \textbf{Define Tetrahedron functions 1481} \( \equiv \)

\begin{verbatim}
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;
}
\end{verbatim}

1491. \textbf{Draw net. [LDF 2002.11.12.]} As of this date it's necessary to rotate the triangles into the x-y plane, because \textbf{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 put 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!

\[ \text{LDF 2002.11.12.} \text{ Added this function.} \]

\begin{verbatim}
(Declare Tetrahedron functions 1480) \( \equiv \)
static void draw_net(const real triangle_diameter, bool make_fabs = true);
\end{verbatim}
1492.
(Define Tetrahedron functions 1481) +
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], .075);
    pts[7] = pts[5].mediate (pts[0], .075);
    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 ( *Colors::help_color, "n");
    #endif
    pts[14] = pts[12].mediate (pts[13]);
    #if 0
    pts[14], dotlabel (14);
#end
pts[15] = pts[6].mediate(pts[7], .25);
#else 0
pts[15].dotlabel(15);
#end
pts[16] = pts[14];
#else 0
pts[16].dotlabel(16);
#end
bool_point bp = Point :: intersection_point(pts[14], pts[16], pts[6], pts[10]);
pts[17] = bp.p;
#else 0
pts[17].dotlabel(17);
#end
bp = Point :: intersection_point(pts[14], pts[16], pts[7], pts[11]);
pts[18] = bp.p;
#else 0
pts[18].dotlabel(18);
#end
p[2].set(pts[17], pts[18]);
#else 0
p[2].draw_help (*Colors :: help_color, """);
#end
for (i = 1; i < 16; ++i) {
  pts[19] = pts[17].mediate(pts[18], i / 16.0);
  pts[19].drawdot (*Colors :: default_color, " pencircle_scaled_.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_.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_.5mm" );
}
pts[20] = pts[17];
pts[21] = pts[18];
p[3] = p[0];
\( \text{pts[20]} \Rightarrow \text{pts[21]} \Rightarrow \text{pts[3]} \Rightarrow t; \)
\( \text{pts[3].draw();} \)
\( \text{for} \ (i = 1; \ i < 16; \ ++i) \) {
  \( \text{pts[19]} \Rightarrow \text{pts[20].mediate(pts[21],i/16.0);} \)
  \( \text{pts[19].drawdot(*Colors::default_color,"pencircle_scaled_5mm");} \)
}
\( \text{t.reset();} \)
\( \text{pts[22]} \Rightarrow \text{v[2].get_center();} \)
\( \text{pts[23]} \Rightarrow \text{pts[22];} \)
\( \text{pts[23].shift(0.0,1);} \)
\( \text{t.rotate(pts[22],pts[23],120);} \)
\( \text{pts[20]} \Rightarrow \text{pts[17];} \)
\( \text{pts[21]} \Rightarrow \text{pts[18];} \)
\( \text{pts[3]} \Rightarrow \text{pts[0];} \)
\( \text{pts[20]} \Rightarrow \text{pts[21]} \Rightarrow \text{pts[3]} \Rightarrow t; \)
\( \text{pts[3].draw();} \)
\( \text{for} \ (i = 1; \ i < 16; \ ++i) \) {
  \( \text{pts[19]} \Rightarrow \text{pts[20].mediate(pts[21],i/16.0);} \)
  \( \text{pts[19].drawdot(*Colors::default_color,"pencircle_scaled_5mm");} \)
}
\( \text{t.rotate(pts[4],pts[1]);} \)
\( \text{t.rotate(pts[9],pts[2]);} \)
\( \text{pts[20]} \Rightarrow \text{pts[17];} \)
\( \text{pts[21]} \Rightarrow \text{pts[18];} \)
\( \text{pts[3]} \Rightarrow \text{pts[0];} \)
\( \text{pts[20]} \Rightarrow \text{pts[21]} \Rightarrow \text{pts[3]} \Rightarrow t; \)
\( \text{pts[3].draw();} \)
\( \text{for} \ (i = 1; \ i < 16; \ ++i) \) {
  \( \text{pts[19]} \Rightarrow \text{pts[20].mediate(pts[21],i/16.0);} \)
  \( \text{pts[19].drawdot(*Colors::default_color,"pencircle_scaled_5mm");} \)
}

1493. Dodecahedron.

1494. Dodecahedron class definition.

    format Dodecahedron Polyhedron
    { Define class Dodecahedron 1494 } \equiv
    class Dodecahedron : public Polyhedron {
    protected: static const real dihedral_angle; /* In radians */
        real pentagon_radius;
    public: { Declare Dodecahedron functions 1497 }
    };

This code is used in sections 1534 and 1535.

1495. Define static const Dodecahedron data members.

[Log 2003.07.18. Now passing "2.0" instead of "2.0" as the argument to atan(). GCC 3.3 couldn’t compile this file, the way it was before.]
Define static const Dodecahedron data members (1495) \equiv
const real Dodecahedron :: dihedral_angle = \pi - \tan(2.0);

This code is used in section 1534.

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) \equiv
Dodecahedron();

See also sections 1500, 1503, and 1505.

This code is used in section 1494.

1498.

(Define Dodecahedron functions 1498) \equiv
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) \equiv
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[i].get_point();
    }
    real angle = 180 - (dihedral_angle * 180.0 / P1);
1502. [LDF 2003.08.10] Check this, as noted above.

(Define Dodecahedron functions 198 ) +
reg_polygons[1]-rotate (pts[0], pts[1], angle);
reg_polygons[2]-rotate (pts[0], pts[1], angle);
reg_polygons[3]-rotate (pts[0], pts[1], angle);
reg_polygons[4]-rotate (pts[0], pts[1], angle);
reg_polygons[5]-rotate (pts[0], pts[1], angle);
#endif
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) {
    if (0)
      (**iter).draw (* (col_vec[i]));
    ++i;
    reg_polygons[0]-draw ();
  }
  for (i = 0; i < 6; ++i) { reg_polygons.push_back ( create_new < Reg_Polygon > (0) );
    *reg_polygons.back () = *reg_polygons[i];
    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_0 = reg_polygons[0]-get_center ();
Point center_6 = reg_polygons[6]-get_center ();
center = center_0.mediate (center_6);
Transform t = center.shift (-center);
if (angle_x = 0 \lor angle_y = 0 \lor angle_z = 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_\theta is only ever used when do_half \equiv 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_\theta is used, the net is always generated in a plane parallel
to the x-z plane.

[Log]

(Declare Dodecahedron functions 1497) +\equiv

    static vector<Reg_Polygon *> get_net(const real pentagon_diameter, bool do_half = false);

[Log]

[LDF 2002.08.12.] Removed center_\theta argument.
1504. 

Define Dodecahedron functions 1998 \( \equiv \)

```cpp
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():\nclairon<"y-coordinate\!-\!0!\n" << "You'd better fix this!\n\n" << flush;
    }
    return pents;
}
for (i = 6; i < 12; i++) { pents.push_back ( create_new < Reg_Polygon > (0) );
*pents[i] = *pents[i - 6];
pents[11].rotate (0, 180); } pts[0] = pents[11].get_point(0);
pents[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():\nclairon<"y-coordinate\!-\!0!\n" << "You'd better fix this!\n\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 \texttt{pts} and Path \texttt{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: \textit{portrait} doesn't work right. Fix it! !! KLUDGE: \textit{portrait} is set to \textit{false} at the beginning of this function and a warning is issued.

---

\textbf{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) +
\begin{verbatim}
static void draw_net(const real pentagon_diameter, bool portrait = true, bool make_tabs = true);
\end{verbatim}
1506.  
(Define Dodecahedron functions 1498 ) +

```cpp
void Dodecahedron::draw_net(const real &pentagon_diameter, bool portrait, bool make-tabs)
{
    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);
    
        #if 0
            (**iter).rotate(90);
        #endif

        Point p[32];
        int i;
        for (i = 0; i < 5; i++) {
            p[i] = v[3] -> get_point(i);
            #if 0
                p[i].dotlabel(i);
            #endif
        }
    }
}
```

/* START HERE. */

```cpp
i = 0;
    for (vector<Reg_Polygon *>::iterator iter = v.begin(); iter != v.end(); ++iter) {
        (**iter).draw();
        (**iter).get_center().label(i++, "n");
    }
    if (!make-tabs) return;
    return;
```

1507.  Icosahedron.

1508.  Icosahedron class definition.

(Define class Icosahedron 1508 ) +

```cpp
class Icosahedron : public Polyhedron {
    protected: static const real dihedral_angle;  /* In radians! */
    real triangle_radius;
    public: (Declare Icosahedron functions 1511)
};
```

This code is used in sections 1534 and 1535.

1509.  Define static const Icosahedron data members.

(Define static const Icosahedron data members 1509 ) +

```cpp
const real Icosahedron::dihedral_angle = PI - asin(2.0/3.0);
```

This code is used in section 1534.

1510.  Constructors and setting functions.

1511.  Default constructor.  (No arguments.)

(Declare Icosahedron functions 1511 ) +

```cpp
Icosahedron();
```

See also sections 1514, 1516, and 1518.

This code is used in section 1508.
1512.

(Define Icosahedron functions 1512) \equiv

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.


\[
\text{Log}
\]

[\text{LDF 2002.10.16.}] Defined this function.

(Declare Icosahedron functions 1511) \equiv

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) \[\equiv\]

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;
  #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;

  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", "0");
  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 - ( dihedrA_Langle * 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);
  *reg_polygons[7] *= *reg_polygons[2] *= *reg_polygons[3] *= reg_polygons[5].rotate (pts[1], pts[0], -angle);
  *reg_polygons[1] *= *reg_polygons[9].rotate (pts[2], pts[6], angle);
  *reg_polygons[9] *= *reg_polygons[1] *= *reg_polygons[6] = rotate (pts[2], 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).get_y() - reg_polygons[19].get_point(0).get_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 != 0 || angle_y != 0 || angle_z != 0)
    for (vector<Reg_Polygon *>::iterator iter = reg_polygons.begin(); iter != reg_polygons.end(); ++iter) (**iter).rotate(angle_x, angle_y, angle_z);
if (p != origin) {
    for (vector<Reg_Polygon *>::iterator iter = reg_polygons.begin(); iter != reg_polygons.end(); ++iter) {
        (**iter).shift(p - center);
    }
    center = p;
}
if (DEBUG) {
    cout << "Exiting Icosahedron::Icosahedron() .n" << flush;
}
return; }

1516. Get net.

Log

[LDF 2002.11.10] BUG FIX: Removed erroneous code that created 6 extra triangles.
[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 ) \( \equiv \)

vector(Rgg_polygon *) Icosahedron::get_net(const real Vtriangle_diameter, bool do_half) { bool
  DEBUG = false; /* true */
  vector(Rgg_polygon *) triangles;
  int i; for (i = 0; i < 10; ++i) triangles.push_back ( create_new < Rgg_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);
  pts[3] = triangles[1].get_point(0);
  *triangles[2] = *triangles[0];
  *triangles[3] = *triangles[1];
  for (i = 0; i < 4; ++i) {
    *triangles[4 + i] = *triangles[i];
  }
  if (DEBUG)
    for (i = 0; i < 4; ++i) pts[i].dotlabel(i);
  if (do_half \equiv true) {
    *triangles[8] = *triangles[4];
    *triangles[9] = *triangles[5];
    for (i = 0; i < 10; ++i) {
      for (int j = 0; j < 3; j++)
        if (triangles[i].get_point(j).get_y() \neq 0) {
          cerr << "ERROR! I.\n\nI. Icosahedron::get_net() \n" << "\ncoordinate \n" << "\nYou\'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 ( create_new < Rgg_polygon > (0) );
  for (int j = 5; j \leq 10; j += 4)
    for (i = 0; i < 4; ++i) {
      *triangles[j + i] = *triangles[j - 4 + i];
      triangles[j + i].shift(pts[1] - pts[2]);
    }
  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, "\n");
  }
for (i = 0; i < 20; ++i) {
    for (int j = 0; j < 3; j++)
        if (triangles[i].get_point(j).get_y() != 0)
            cerr << "ERROR! In Icosahedron::get_net():\n" << "y-coordinate != 0!\n" << 
            "You’d better fix this!\n\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);
void Icosahedron(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[1];
    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]);
\$1519\ 3DLDF-1.1.5.1\quad\text{DRAW NET}\quad463

\begin{verbatim}
p[5].shift(v[18]-get_point(1) - pts[1]);
p[6] = p[0];
p[6].rotate (0, 0, 240);
p[6].rotate (pts[0], pts[2]);
#if 0
v[3]~dotlabel( );
#endif
p[6].shift(v[3]-get_point(0) - pts[2]);
p[7].shift(v[7]-get_point(0) - pts[2]);
p[8].shift(v[11]-get_point(0) - pts[2]);
p[9].shift(v[15]-get_point(0) - pts[2]);
p[10].shift(v[19]-get_point(0) - pts[2]);
for (i = 0; i < 11; i++) p[i].draw ( );
return;
\end{verbatim}

1520. Semi-Regular Archimedean Polyhedra.

1521. Truncated Octahedron.

1522. \texttt{Trunc\_Octahedron} class definition.

(Define class \texttt{Trunc\_Octahedron} 1522 ) \equiv

\begin{verbatim}
class Trunc\_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 \texttt{Trunc\_Octahedron} functions 1525)
};
\end{verbatim}

This code is used in sections 1534 and 1535.

1523. Define static const \texttt{Trunc\_Octahedron} data members.

(Define static const \texttt{Trunc\_Octahedron} data members 1523 ) \equiv

\begin{verbatim}
const real Trunc\_Octahedron::angle\_hex\_square = (125 + (16.0/60.0)) * (PI/180.0);
const real Trunc\_Octahedron::angle\_hex\_hex = (109 + (28.0/60.0)) * (PI/180.0);
\end{verbatim}

This code is used in section 1534.

1524. Constructors and setting functions.

\begin{tabular}{ll}
\textbf{Log} & \\
\hline
\end{tabular}

[LDF 2003.04.15.] Commented-out the constructors and \texttt{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 \texttt{Trunc\_Octahedron} functions 1525 ) \equiv

\texttt{Trunc\_Octahedron();}

See also sections 1528 and 1530.

This code is used in section 1522.
1526.

Define Trunc_Octahedron functions 1526}

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;
}

See also sections 1529, 1531, and 1532.
This code is used in section 1534.

1527. Center, diameter of hexagon, and angles.

1528. Constructor.

[LDF 2002.11.08.] Added this function.

(Declare Trunc_Octahedron functions 1525 ) +≡
#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
1529.

(Define Trunc_Octahedron functions 1526) \[=\]

```c
# 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);
# if 0
  pts[0].dotlabel(0);
  pts[1].dotlabel(1, "lft");
  pts[2].dotlabel(2, "bot");
  pts[3].dotlabel(3, "bot");
  pts[4].dotlabel(4, "rt");
  pts[5].dotlabel(5);
# endif
  real angle_hh = 180.0 - (angle_hex_hex * 180.0/PI);
  reg_polygons[1].rotate(pts[4], pts[5], angle_hh);
  reg_polygons[2].rotate(pts[3], pts[2], -angle_hh);
  reg_polygons[3].rotate(pts[0], pts[1], angle_hh);
  real angle_hh = 180.0 - (angle_hex_square * 180.0/PI);
  reg_polygons[4].rotate(pts[0], pts[5], angle_hh);
  reg_polygons[5].rotate(pts[1], pts[2], angle_hh);
  reg_polygons[6].rotate(pts[3], pts[4], angle_hh);
  /* [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;

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

[LDL 2002.11.08.] Added this function.
[LDL 2002.08.12.] Removed center_0 argument.

(Declare Trunc_Octahedron functions 1529) +

# if 0
static vector<Reg_Polygon *> get_net(const real hexagon_diameter, bool do_half = false);
# endif
1531.  
(def 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_Polygon > (reg_polygons[0]) ) ;
    Reg_Polygon > (reg_polygons[0]) ) ;
    reg_polygons[3]->shift (pts[1] - pts[3]);
    real side_length = pts[5].getx() - pts[0].getx();
    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, `reg_polygons[5]` and `reg_polygons[6]` must be changed, because I've made the net a bit differently from the way it's done in Cundy. Get references! Page 104. I made two of the squares slanted, in order to avoid having to rotate them twice.

(Define `Trunc_Octahedron` functions 1526) +≡
  #if 0
  pts[14] = reg_polygons[4].get_point(2);
  pts[15] = reg_polygons[4].get_point(3);
  reg_polygons[5].shift(p1[0] - pts[14]);
  reg_polygons[6].shift(p1[1] - pts[14]);
  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.push_back( create_new < Reg_Polygon > (reg_polygons[0]) ); /* 9 */
  reg_polygons.push_back( create_new < Reg_Polygon > (reg_polygons[6]) ); /* 10 */
  return reg_polygons;
#endif

1533. Putting polyhedra together.

1534. This is what's compiled.

(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)
1535. This is what’s written to polyhed.h.

\[
\begin{align*}
\text{(polyhed.h 1535)} & \equiv \\
& \text{(Define class Polyhedron 1472)} \\
& \text{(Define class Tetrahedron 1477)} \\
& \text{(Define class Dodecahedron 1494)} \\
& \text{(Define class Icosahedron 1508)} \\
& \text{(Define class Trunc_Octahedron 1522)} \\
\end{align*}
\]

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.
```

\[
\begin{align*}
\text{(Version control identifier 5)} & +\equiv \\
& \text{static string res_id = "$Id: parser.web,v.1.4,2004/01/12,21:30:44,ifinsto1,Exp,$";} \\
\end{align*}
\]

1537. Include files. map.h is currently not needed, but I plan to use it for the input routine. [LDF 2004.01.06.]

\[
\begin{align*}
\text{(Include files 6)} & +\equiv \\
& \text{#include "loader.h"} \\
& \text{#include "pspglb.h"} \\
& \text{#include "io.h"} \\
& \text{#include "colors.h"} \\
& \text{#include "transfor.h"} \\
& \text{#include "shapes.h"} \\
& \text{#include "pictures.h"} \\
& \text{#include "points.h"} \\
& \text{#include "lines.h"} \\
& \text{#include "planes.h"} \\
& \text{#include "paths.h"} \\
& \text{#include "curves.h"} \\
& \text{#include "polygons.h"} \\
& \text{#include "rectangs.h"} \\
& \text{#include "ellipses.h"} \\
& \text{#include "circles.h"} \\
& \text{#include "patterns.h"} \\
& \text{#include "solids.h"} \\
& \text{#include "solfaced.h"} \\
& \text{#include "cuboid.h"} \\
& \text{#include "polyhed.h"} \\
& \text{#include "utility.h"} \\
\end{align*}
\]

1538. Parse.

\[
\begin{align*}
\text{(Declare parser functions 1538)} & \equiv \\
& \text{This code is used in section 1542.}
\end{align*}
\]
1539.  
(Define parser functions 1539) ≡
This code is used in section 1541.

1540. **Putting the parser together.**

1541. This is what’s compiled.
    (Include files 6)
    (Version control identifier 5)
    (Define parser functions 1539)
This is what’s written to parser.h.

```c
static string res_id = "$Id::main.web,v,1.12,2004/01/12,21:30:34_1finsto11_Exp,$"
```

getopt.h is included for processing the command line options. [LDF 2003.08.14.]
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.]

Get input 1545

1546. Actions in main.

Actions in main 1546

using namespace Colors;
using namespace Projections; if (ldf_real_float) MAX_REAL = System::get_secondLargest < float > (FLT_MAX, false); else if (ldf_real_double) MAX_REAL = System::get_secondLargest < 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);
#endif
See also section 1547.
This code is used in section 1557.
1547. Your code here!

BEGINFIG(1);
Point p;
Point q(1,1,1);
Circle c(p,2,90);
c.filldraw();
p.draw(g);
currentpicture.output();
ENDFIG;

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.]

[LDL 2003.08.14.] Added this section.

(Process command line options 1548) \equiv
#ifdef __GNU__
(GCC command line option processing 1549)
#else
#ifdef __DECCXX
(DEC command line option processing 1551)
#endif
#endif

This code is used in section 1555.
1549. GCC version of command line processing.

[Log]

Added this section.
[Log]

Added code for handling the "--silent" option, including the constant SILENT_INDEX.

GCC command line option processing 1549

```c
bool DEBUG = false; /* true */
int optionctr;
int digitoptind = 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) {
  optionctr = getopt_long_only (argc, argv, "hv", long_options, &option_index);
  if (DEBUG) {
    cout << "optionctr=\"" << optionctr << endl << flush;
    cout << "option_index=\"" << option_index << endl << flush;
    cout << "optarg=\"" << optarg << endl << flush;
  }
  if (optionctr == -1) {
    if (DEBUG) cout << "No more options." << endl << flush;
    break;
  }
  if (optionctr == 0) {
    if (DEBUG) {
      cout << "option=\"" << long_options[option_index].name;
      if (optarg) cout << "\"" << optarg << "\"";
      cout << endl;
    }
    if (option_index == HELP_INDEX) {
      cout << "3DLDF version\" " << VERSION_3DLDF << "\"" << COPYRIGHT_3DLDF << endl << "Valid options for 3DLDF are:\" " << endl << "-help\" Prints this message and exits.\" " << with_return_value << endl << "-version\" Prints the version number of 3DLDF\" " << endl << "-slient\" Suppresses some output to standard output\" " << endl << "--version\" Prints the version number of 3DLDF\" " << endl << "--slient\" Suppresses some output to standard output\" " << endl << "--exit\" Exits with return value\" " << endl << exit(0);
    }
  } else if (option_index == SILENT_INDEX) {
    if (DEBUG) cout << "Setting SILENT_GLOBAL to true.\" " << endl;
  }
  // other options
}
```

---

Process Command Line Options

3DLDF-1.1.5.1 §1549

---
SILENT_GLOBAL = true;
}
else if (option_index == VERbose_INDEX) {
    if (DEBUG) cout << "Setting VERbose_GLOBAL to true." << endl;
    VERbose_GLOBAL = true;
}
else if (option_index == VERSION_INDEX) {
    cout << "3DLDF_Version: " << VERSION_3DLDF << " COPYRIGHT_3DLDF " << endl << flush;
    if (DEBUG) cout << "Exiting with return value 0." << endl << flush;
    exit(0);
} else {
    cerr << "This can't happen!" << "option_index has invalid value:" << option_index << endl << "Will try to continue." << endl << flush;
}
else if (option_str == '?' ) {
    cerr << "getopt_long() returned ambiguous match. Breaking." << endl << endl << flush;
    break;
} else {
    cerr << "getopt_long() returned invalid option." << endl << flush;
}
    if (DEBUG) cout << "**************

*/ while */
if (optind < argc) {
    cout << "non-opt_ARGV-elements:";
    while (optind < argc) cout << argv[optind++] << endl;
    cout << endl << flush;
}
if (DEBUG) {
    cout << "Exiting (Debugging command line option processing.)" << 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⟩ ≡ /* 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⟩ ≡

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⟩ ≡

  cout.setf(ios::fixed, ios::floatfield);
  cout.precision(1);
  cout << "3DLDF_Version" << VERSION_3DLDF << "." << endl << 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.

[LDI 2003.08.14.] Added this section.

\[
\text{\{GCC 3.3 and DEC print version, copyright, and license information 1554\} \equiv}
\]
\[
cout \ll "3DLDF\_Version\_", \ll \text{VERSION\_3DLDF} \ll "." \ll \text{COPYRIGHT\_3DLDF} \ll \text{endl} \ll \\
\text{DISCLAIMER\_3DLDF} \ll \text{endl} \ll \text{flush};
\]
\[
cout\_\text{precision}(6);
\]
This code is used in section 1552.

1555. Main itself.

\[
\text{\{Main 1555\} \equiv}
\]
\[
\text{int } \text{main}(\text{int } \text{argc}, \text{char } *\text{argv}[]) \{ \text{Process command line options 1548} \};
\]
See also sections 1556 and 1557.

This code is used in section 1558.

1556. [LDI 2003.08.29.] Changed the string that’s passed to \text{initialize\_io()} as the name of the input file. It’s a dummy name, since I’ve changed \text{initialize\_io()} today, so that no \text{in\_stream} isn’t opened. Also, commented-out the line where \text{in\_stream} is closed.

[LDI 2003.11.28.] Removed \text{TeX} text above, that referred to the precision used when printing out \text{VERSION\_3DLDF}.

This is no longer relevant, because \text{VERSION\_3DLDF} is now a \text{string} rather than a \text{real}.

\[
\text{\{Main 1555\} \equiv}
\]
\[
\text{(Print version, copyright, and license information 1552)};
\]
1557.  
\langle Main 1555 \rangle + \equiv 
initialize_io("3DLDFin.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,3DLDF,Version=" << VERSION_3DLDF << ".\n\n" << flush;
}
return (0); }

1558. Putting Main together. This is what’s compiled.
\langle Include files 6 \rangle
\langle Version control identifier 5 \rangle
\langle Main 1555 \rangle

1559. Appendices.

1560. References.


Unfortunately out of print.


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.


### 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

51 Franklin St, Fifth Floor, Boston, MA 02110-1231 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. Secondarily, 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 worldwide, 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
copies you make or distribute. However, you may accept compensation in exchange for copies. If you
distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

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 such 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”, “Dedications”, 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 this:

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.

\[ \text{GNU Free Documentation License 1561} \equiv \] /* 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 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. (http://fsf.org/)
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. 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 them 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 prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. 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.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.
For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.


The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.
The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.


No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work’s users, your or third parties’ legal rights to forbid circumvention of technological measures.


You may convey 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; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to
the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to “keep intact all notices”.

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an “aggregate” if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation’s users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions...
next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publically documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.
Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.


You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission
to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s "contributor version".

A contributor’s "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then
the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.


If 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 convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU 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 that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

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.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS 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.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

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 state 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 3 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, see ⟨http://www.gnu.org/licenses/⟩.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

⟨program⟩ Copyright (C) (year) (name of author) This program 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, your program’s commands might be different; for a GUI interface, you would use an “about box”.
You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see (http://www.gnu.org/licenses/).

The GNU 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 Lesser General Public License instead of this License. But first, please read (http://www.gnu.org/philosophy/why-not-lgpl.html).

(GNU General Public License 1562)  * This section contains no C++ code. */

This code is cited in section 1.563.

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, \cfun(z) would come before \func(a). \cfun is for class functions, and \func is for operators that aren’t members of a class.

First come the index entries that 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, 354, 448, 480, 502, 546, 547, 559, 572, 727, 834, 813, 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: 441.

!! 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_map: 149.

Color::get_blue_part: 143.

Color::get_green_part: 142.

Color::get_name: 146.

Color::get_red_part: 141.

Color::set_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.

Dodecahedron::set_net: 1503.

Ellipse::Ellipse: 1156.
Ellipse::angle_point: 1201.
Ellipse::dotlabel: 1162.
Ellipse::draw_out_rectangle: 1261.
Ellipse::Ellipse: 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::intersection_rectangle: 1259.
Ellipse::intersection_points: 1208, 1210, 1212.
Ellipse::oriented: 1167.
Ellipse::is_elliptical: 1164.
Ellipse::is_quadratic: 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::solve: 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::dotlabel: 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_free_store: 732.
Path::set_pen: 749.
Path::set: 709, 714, 719.
Path::show: 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::unsuppress_output: 897.
Path::clear: 300, 590.
Path::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::unsuppress_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.
Point::clean: 346.
Point::cross_product: 544.
Point::dot_product: 542.
Point::dotlabel: 510, 512.
Point::draw_help: 472, 473, 841, 842.
Point::draw: 463, 464, 823, 824.
Point::drawarow: 466, 467, 832, 833.
Point::drawdot: 454, 456.
Point::epsdou: 350.
Point::extract: 486.
Point::get_all_coords: 355, 357.
Point::get_coord: 360, 362.
Point::get_extremes: 488.
Point::get_line: 352, 645.
Point::get_maximum: 491.
Point::get_mean: 493.
Point::get_minimum: 489.
Point::get_normal: 557, 945.
Point::get_transform: 384.
Point::get_x: 380, 382.
Point::get_y: 365, 367.
Point::getz: 370, 372.
Point::getz: 375, 377.
Point::intersection_point: 572, 573, 647.
Point::is_identity: 349.
Point::is_in_triangle: 401.
Point::is_on_line: 398.
Point::is_on_plane: 400.
Point::is_on_segment: 393, 396.
Point::label: 505, 507.
Point::magnitude: 546.
Point::medial: 555.
Point::output: 501.
Point::Point: 324, 327, 331.
Point::project: 442, 446.
Point::reset_transform: 450.
Point::rotate: 404, 436, 438, 760.
Point::scale: 406.
Point::set_extremes: 495.
Point::set_on_free_store: 342.
Point::set: 329, 333.
Point::shear: 408.
Point::shift_times: 419, 421.
Point::shift: 285, 412, 414.
Point::show_transform: 477.
Point::show: 475.
Point::slope: 389.
Point::suppress_output: 482.
Point::undraw: 469, 470, 860, 861.
Point::undrawdot: 458, 460.
Point::unit_vector: 551, 553.
Point::unsuppress_output: 484.
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_CI_Plane_Curve::angle_point: 994.
Reg_CI_Plane_Curve::get_coefficients: 990.
Reg_CI_Plane_Curve::half: 1013.
Reg_CI_Plane_Curve::intersection_points: 997, 1008.
Reg_CI_Plane_Curve::is_cubic: 988.
Reg_CI_Plane_Curve::is_quadratic: 988.
Reg_CI_Plane_Curve::is_quartic: 989.
Reg_CI_Plane_Curve::location: 992.
Reg_CI_Plane_Curve::quarter: 1014.
Reg_CI_Plane_Curve::segment: 1011.
Reg_CI_Plane_Curve::solve: 991.
Reg_Polygon::draw_inCircle: 1091, 1092, 1309, 1310.
Reg_Polygon::draw_outCircle: 1095, 1096, 1312, 1313, 1314.
Reg_Polygon::get_radius: 1087.
Reg_Polygon::incircle: 1089, 1307.
Reg_Polygon::outcircle: 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::fill_draw: 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_max: 1411.
Solid::get_mean: 1413.
Solid::get_min: 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::suppress_output: 1415.
Solid::undraw: 1432.
Solid::unfill: 1435.
Solid::unfill_draw: 1438.
Tetrahedron::draw_net: 1491.
Tetrahedron::get_net: 1489.
Tetrahedron::set: 1487.
Transform::align_with_axis: 213, 423.
Transform::clear: 180.
Transform::epsilon: 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.
Ellipsoid::operator*=: 1233.
Ellipsoid::operator+=: 1157.
Ellipsoid::operator*+=: 1231.
Line::operator+=: 643.
Path::operator*=: 781.
Path::operator+=: 797, 801.
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.
create: 245.
create.new.circle: 1286, 1287.
create.new.color: 111.
create.new.cuboid: 1462.
create.new.ellipse: 1154, 1155.
create.new.path: 724, 725.
create.new_point: 335.
cuboid: 1166, 1167.
cuboid: 1146, 1145, 1153, 1154, 1155, 1156, 1157, 1158, 1159, 1160, 1162, 1163, 1164, 1165, 1166, 1167.
cur_angle: 1150.
cur_color: 1323.
cur_location: 1224.
cur_point: 1224, 1225.
cur_x: 1150.
curr.y: 300, 310, 327, 329.
curr.z: 300, 310, 327, 329.
curr.z: 1150.
curve_0: 1001.
curve_4: 1001.
cyan: 152, 156, 157, 1502, 1546.
cycle: 711, 712, 713, 714, 715, 925, 926, 929, 930.
d: 24, 25, 315, 688, 948, 1123, 1222, 1352, 1459, 1460.
d.x: 948, 950.
d.y: 948, 949, 950.
d.z: 948, 949, 950.
date_stamp: 82, 83.
dbl_max: 19, 66, 1546.
dbl_size: 65.
dd: 1222.
ddiameter: 1076, 1077, 1079, 1080, 1081, 1281, 1282, 1283, 1284.
drawdot_color: 453, 454, 455, 456, 457.
DISCLAIMER. 3DLDF: 22, 23, 1553, 1554.
dist: 603, 604, 605, 606, 607, 609, 610, 611, 612, 967, 968, 969, 970, 972, 973, 974, 1305.
do_half: 1180, 1503, 1504, 1516, 1517, 1530, 1531.
do_help_lines: 471, 608, 699, 837.
do_inner: 1321.
DO_LABELS: 253, 254, 261, 506, 873, 1161.
do_middle: 1321.
do_sort: 591.
do_transform: 593, 1165, 1231, 1232, 1233, 1234, 1235, 1236, 1242, 1243, 1252, 1253.
do_warnings: 208, 299, 591, 592, 595, 598.
Dodecahedron: 241, 1352, 1470, 1494, 1495, 1497, 1498, 1500, 1501, 1504, 1505, 1506, 1514, 1518.
dot: 253, 505, 506, 507, 508, 515, 516, 871, 872, 873, 874, 875, 1160, 1161, 1162.
double_rows: 1320, 1321.
doubles: 15, 182.
DRAW: 244, 246, 819, 902, 1422.
draw_axes: 968, 969, 973, 974, 1001.
draw_color: 698, 701, 705, 708, 710, 713, 715, 718, 720, 727, 728, 729, 739, 741, 818, 819, 845, 849, 850, 856, 864, 907, 902, 904, 905,
is_32_bit: 45, 46
is_64_bit: 47, 48
insect_line: 1227
ISO: 236, 237
iter: 294, 587, 589, 590, 593, 594, 595, 596, 703,
727, 777, 782, 784, 786, 808, 809, 813, 814, 873,
883, 886, 942, 944, 958, 1032, 1034, 1041, 1161,
1165, 1296, 1324, 1339, 1346, 1347, 1382, 1384,
1387, 1389, 1405, 1407, 1420, 1422, 1424, 1425,
1427, 1428, 1430, 1431, 1433, 1434, 1436, 1437,
1439, 1458, 1465, 1502, 1506, 1515, 1519, 1529.
iter_cnt: 1327
iterator: 294, 590, 593, 597, 703, 727, 777, 782,
784, 786, 883, 886, 902, 958, 1324, 1327, 1346,
1384, 1387, 1405, 1407, 1420, 1465, 1474, 1502,
1506, 1515, 1519, 1529.
iter0: 1474.
iter1: 1474.
j: 171, 175, 177, 181, 186, 188, 193, 217, 219, 227,
443, 449, 496, 1321, 1324, 1501, 1517.
k: 219, 227, 1321.
killLabels: 276, 277.
known: 1171, 1172, 1173.
l: 641, 642, 643, 644, 645, 648, 649, 1043
Lp: 647.
Lq: 647.
label: 252, 254, 272, 273, 309, 310, 505, 506, 507,
508, 509, 511, 513, 871, 872, 873, 874, 875,
878, 967, 972, 1160, 1161, 1162, 1484, 1492,
1506, 1515, 1517, 1519.
Label: 253, 254, 255, 261, 267, 272, 273, 309,
505, 506, 514, 515, 516, 587, 589, 590, 597,
873, 967, 1161.
Labels: 241, 252, 253, 254, 261, 596.
labels: 261, 266, 273, 277, 294, 506, 587, 589,
590, 592, 597.
lambda: 950.
lambda_denumerator: 948, 950.
last Connector: 806, 807, 813, 957
ld: 506, 515, 587.
LDF_GCC_2_95: 5, 6, 7, 8, 14, 24, 25, 1552.
LDF_GCC_3_3: 5, 6, 7, 14.
LDF_PUBLIC: 5, 7.
ldf_real_double: 21, 22, 23, 1546.
LDF_REAL_DOUBLE: 15, 19, 22, 183, 351.
LDF_REAL_FLOAT: 15, 19, 22.
ldf_real_float: 21, 22, 23, 1546.
left: 193.
left_shift: 1321.
light_gray: 152, 156, 157.
limit: 1326, 1327.
Reg_Polygons: 702, 1321.

Reg_polygons: 1333, 1339, 1346, 1347, 1356, 1357, 1367, 1369, 1370, 1382, 1384, 1387, 1389, 1405, 1407, 1419, 1420, 1422, 1424, 1425, 1427, 1428, 1430, 1431, 1433, 1434, 1436, 1437, 1439, 1447, 1448, 1456, 1501, 1502, 1515, 1529, 1531, 1532.

Regular_Closed_Plane_Curve: 984.


reset_angle: 615, 616.

reset_transform: 269, 301, 450, 451.


result: 69, 70.


right_shift: 1321.


rotate_around: 211, 212, 228, 423, 436, 438, 439, 440, 759, 760, 762, 764, 1050, 1052, 1252, 1254, 1401.

row: 178, 179, 190, 191, 227, 228, 229, 625, 626, 628, 629.

row_shift: 1320, 1321.

rp: 68, 69, 70.

rr: 317, 993, 1001, 1002, 1003, 1004, 1005.

Run_State: 502, 906.

r0: 992, 993, 1034, 1035, 1260.

r0_0: 172, 173.

r0_1: 172, 173.

r0_2: 172, 173.

r0_3: 172, 173.

r1: 1034, 1035, 1260.

r1_0: 172, 173.

r1_1: 172, 173.

r1_2: 172, 173.

r1_3: 172, 173.

r2: 1260.

r2_0: 172, 173.

r2_1: 172, 173.

r2_2: 172, 173.

r2_3: 172, 173.

r3_0: 172, 173.

r3_1: 172, 173.

r3_2: 172, 173.

r3_3: 172, 173.

strings: 505, 809, 814, 1321.

strings: 508, 1382.

subpath: 925, 926, 1012.

subpath_size: 1011, 1012.

suppress_labels: 261, 274.

suppress_output: 215, 482, 483, 504, 505, 506, 895, 896, 1415, 1416.

suppress_earnings: 992, 993.

suppress_y: 967.

suppress_x: 967.

surface_hiding: 1043.

surface_vector: 997, 998, 1000, 1028, 1029, 1031.

System: 34, 37, 39, 41, 44, 46, 48, 65, 68, 1546.

sl: 498, 499, 500.

s2: 498, 499, 500.


t_all: 206, 209.

t_inverse: 1005, 1218, 1221, 1223, 1225.


t_x: 206, 207, 209, 561, 562, 563, 564, 565, 566, 948, 949, 950.

t_x: 206, 209, 561, 562, 563, 564, 565, 566, 948, 949, 950.

t_x: 206, 207, 209, 561, 562, 563, 564, 565, 566, 948, 949, 950.

t_x: 206, 209, 561, 562, 563, 564, 565, 566, 948, 949, 950.

t_x: 206, 207, 209, 561, 562, 563, 564, 565, 566, 948, 949, 950.

t_x: 206, 207, 209, 561, 562, 563, 564, 565, 566, 948, 949, 950.

t_x: 206, 207, 209, 561, 562, 563, 564, 565, 566, 948, 949, 950.

t_x: 206, 207, 209, 561, 562, 563, 564, 565, 566, 948, 949, 950.

t_x: 206, 207, 209, 561, 562, 563, 564, 565, 566, 948, 949, 950.

t_x: 206, 207, 209, 561, 562, 563, 564, 565, 566, 948, 949, 950.

t_x: 206, 207, 209, 561, 562, 563, 564, 565, 566, 948, 949, 950.

t_x: 206, 207, 209, 561, 562, 563, 564, 565, 566, 948, 949, 950.

t_x: 206, 207, 209, 561, 562, 563, 564, 565, 566, 948, 949, 950.

t_x: 206, 207, 209, 561, 562, 563, 564, 565, 566, 948, 949, 950.
649, 668, 944, 993, 997, 998, 1029, 1039, 1112, 1165, 1202, 1217, 1222, 1232, 1267, 1268, 1305.
unsupress_labels: 275.
unsupress_output: 245, 484, 485, 596, 897, 898, 1417, 1418.
up: 600, 601, 603, 605, 614, 616, 618, 623, 967.
user_transform: 234, 235, 328.
USER_VALUE: 309.
USER_VALUES: 309, 310, 507, 508.
USER_VALUES_X_Y: 309, 310, 507.
USHORT_SIZE: 68
v_coord: 1004.
v_intercept: 1004, 1174, 1175.
v_length: 1267, 1268.
v_x: 650.
v_y: 650.
v_z: 650.
va_ary: 713, 715, 718, 720.
va_end: 713, 715, 718, 720.
va_start: 713, 715, 718, 720.
valarrays: 322.

VERSION_3DLDF: 22, 23, 1549, 1553, 1554, 1556, 1557.
vertex: 1078.
vertex_radius: 1472, 1481, 1484, 1498, 1501, 1512, 1515, 1526, 1529.
vertical: 444.
vertices: 1145, 1146, 1148, 1160, 1181, 1484, 1498, 1501, 1512, 1515, 1526, 1529.
view_value: 309.
VIEW_VALUES: 309, 310, 507, 508, 873.
VIEW_VALUES_X_Y: 309, 310, 507, 872.
viok: 152, 156, 157, 1546.
viok_red: 152, 156, 157, 1546.
x: 688.
y: 688.
z: 688.
v0: 394.
v1: 394.
v2: 394.
w: 1459, 1460, 1474.
x: 650.
y: 650.
z: 650.
web: 1518.
white: 152, 156, 157.
width: 1453, 1460, 1467.
world_value: 309.
WORLD_VALUES: 309, 310, 507, 508, 872, 873.
WORLD_VALUES_X_Y: 309, 310, 507, 508, 872.
WORLD_VALUES_Z: 309, 310, 507, 508.
write_footers: 84, 85, 1557.
x: 193, 196, 200, 201, 203, 204, 205, 206, 280, 281.
axes: 993.
axes_pt: 1001, 1165, 1267, 1268.
x: 579, 580, 581, 583, 584, 585.
X_Y: 993, 1221, 1223.
X_Z: 993, 1221, 1223.
xz: 1242, 1243, 1390, 1391, 1395, 1396, 1399, 1400.
xy: 197, 198, 245, 408, 409, 768, 769, 1056, 1057, 1239, 1240, 1392, 1395.
zz:  197. 198. 245. 408. 409. 768. 769. 1056. 1057.  
     1239. 1240. 1392. 1393.
zd:  972.
x:   972.
     766. 767. 771. 772. 776. 777. 1047. 1048. 1054. 
     1055. 1059. 1060. 1061. 1065. 1222. 1235. 1236. 
     1237. 1238. 1247. 1248. 1459. 1460.
y_int_p:  579. 580. 581.
y_int_p_z:  580.
y_int_q:  579. 580. 581.
y_int_q_z:  580.
y_shift:  1515.
yellow:  152. 156. 157. 1546.
yellow_green:  152. 156. 157. 1546.
yz:  197. 198. 245. 408. 409. 768. 769. 1056. 1057. 
     1239. 1240. 1392. 1393.
yz:  1242. 1243. 1390. 1391. 1395. 1396. 1399. 1400.
yz:  197. 198. 245. 408. 409. 768. 769. 1056. 1057. 
     1239. 1240. 1392. 1393.
yd:  972.
yl:  972.
     766. 767. 771. 772. 776. 777. 1047. 1048. 1054. 
     1055. 1059. 1060. 1061. 1065. 1165. 1235. 1236. 
     1237. 1238. 1247. 1248. 1459. 1460.
lex:  993.
z_axis:  1001.
z:  579. 580. 582. 584. 585.
z_int_p:  582. 583. 584.
z_int_q:  582. 583. 584.
z_y:  993. 1221. 1223.
zz:  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.
zd:  972.
z:  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 336, 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 Regular 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, 1116, 1122, 1125, 1127, 1130, 1132, 1135, 1136, 1137, 1138) Used in section 1101.

(Declare Reg_Circle 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, 1368, 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, 1243, 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, 1146, 1124, 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.
(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)
(polyhed.h 1535)
(bspglb.h 52)
(rectangs.h 1140)
(shapes.h 249)
(solfaced.h 1450)
(solids.h 1442)
(transform.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 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.
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get second-largest real value</td>
<td>63</td>
<td>Include files</td>
<td>64</td>
</tr>
<tr>
<td>Declare namespace System</td>
<td>65</td>
<td>Get second largest</td>
<td>66</td>
</tr>
<tr>
<td>Calculate second-largest Real</td>
<td>69</td>
<td>Loop for testing bits</td>
<td>70</td>
</tr>
<tr>
<td>Template function instantiations</td>
<td>71</td>
<td>Putting gettemplate together</td>
<td>73</td>
</tr>
<tr>
<td>I/O (io.web)</td>
<td>76</td>
<td>Include files</td>
<td>77</td>
</tr>
<tr>
<td>Global variables</td>
<td>78</td>
<td>I/O functions</td>
<td>80</td>
</tr>
<tr>
<td>Initialize I/O</td>
<td>81</td>
<td>Write footers</td>
<td>84</td>
</tr>
<tr>
<td>Begin figure</td>
<td>86</td>
<td>End figure</td>
<td>88</td>
</tr>
<tr>
<td>Putting I/O together</td>
<td>90</td>
<td>Constructor and setting functions</td>
<td>96</td>
</tr>
<tr>
<td>Default constructor</td>
<td>97</td>
<td>Copy constructor</td>
<td>99</td>
</tr>
<tr>
<td>Name and unsigned short arguments</td>
<td>101</td>
<td>Constructor</td>
<td>102</td>
</tr>
<tr>
<td>Setting function</td>
<td>104</td>
<td>Three real arguments</td>
<td>106</td>
</tr>
<tr>
<td>Constructor</td>
<td>107</td>
<td>Setting function</td>
<td>109</td>
</tr>
<tr>
<td>Pointer argument</td>
<td>112</td>
<td>Pseudo-constructor for dynamic allocation</td>
<td>111</td>
</tr>
<tr>
<td>Reference argument</td>
<td>113</td>
<td>Assignment</td>
<td>114</td>
</tr>
<tr>
<td>Equality</td>
<td>116</td>
<td>Equality</td>
<td>116</td>
</tr>
<tr>
<td>Inequality</td>
<td>118</td>
<td>Inequality</td>
<td>118</td>
</tr>
<tr>
<td>Modifying</td>
<td>120</td>
<td>Modifying</td>
<td>120</td>
</tr>
<tr>
<td>Set on free store</td>
<td>121</td>
<td>Set on free store</td>
<td>121</td>
</tr>
<tr>
<td>Set name</td>
<td>123</td>
<td>Set name</td>
<td>123</td>
</tr>
<tr>
<td>Set use name</td>
<td>125</td>
<td>Set use name</td>
<td>125</td>
</tr>
<tr>
<td>Modify</td>
<td>127</td>
<td>Modify</td>
<td>127</td>
</tr>
<tr>
<td>Set red part</td>
<td>129</td>
<td>Set red part</td>
<td>129</td>
</tr>
<tr>
<td>Set green part</td>
<td>131</td>
<td>Set green part</td>
<td>131</td>
</tr>
<tr>
<td>Set blue part</td>
<td>133</td>
<td>Set blue part</td>
<td>133</td>
</tr>
<tr>
<td>Show</td>
<td>135</td>
<td>Show</td>
<td>135</td>
</tr>
<tr>
<td>Returning elements and information</td>
<td>137</td>
<td>Returning elements and information</td>
<td>137</td>
</tr>
<tr>
<td>Is on free store</td>
<td>138</td>
<td>Is on free store</td>
<td>138</td>
</tr>
<tr>
<td>Get Color parts</td>
<td>140</td>
<td>Get Color parts</td>
<td>140</td>
</tr>
<tr>
<td>Get red part</td>
<td>141</td>
<td>Get red part</td>
<td>141</td>
</tr>
<tr>
<td>Get green part</td>
<td>142</td>
<td>Get green part</td>
<td>142</td>
</tr>
<tr>
<td>Get blue part</td>
<td>143</td>
<td>Get blue part</td>
<td>143</td>
</tr>
<tr>
<td>Get use name</td>
<td>144</td>
<td>Get use name</td>
<td>144</td>
</tr>
<tr>
<td>Get name</td>
<td>146</td>
<td>Get name</td>
<td>146</td>
</tr>
</tbody>
</table>
# Table of Contents

Output operator .......................................................... 147 40
Define Colors in METAPost ........................................... 149 40
Initialize Colors ......................................................... 151 41
Namespace Colors ...................................................... 153 42
Major Colors .............................................................. 155 43
   Internal (with initialization) ....................................... 156 43
   External .................................................................. 157 44
All Colors ................................................................. 158 44
Global constants ......................................................... 159 44
Putting Color together .................................................. 161 45

Transformations (transform.web) ....................................... 164 46
Include files ............................................................... 165 46
Transform class definition ............................................. 166 46
Constructors ............................................................... 167 46
   Default constructor .................................................. 168 46
   Constructor with one real argument ............................. 170 47
   Constructor with 16 real arguments ............................. 172 47
Assignment .................................................................. 174 48
Reset to identity matrix ............................................... 175 48
Setting values .............................................................. 176 48
Clean ....................................................................... 178 49
Epsilon ..................................................................... 180 49
Test for identity matrix .................................................. 182 50
   Non-const version ..................................................... 185 50
   const version ................................................................ 187 51
Querying .................................................................... 189 51
   Get element ................................................................ 190 51
Show .................................................................... 192 51
Affine transformations ................................................... 194 52
   Scale ..................................................................... 195 52
   Shear ..................................................................... 197 53
   Shift ..................................................................... 199 54
   real arguments ......................................................... 200 54
   Point argument ........................................................ 202 55
   Shift with multiplication ............................................ 203 55
Rotation around the main axes ....................................... 205 55
Rotation around an arbitrary axis ................................. 210 57
   Point arguments ....................................................... 211 57
   Path argument ......................................................... 212 58
   Alignment with an axis .............................................. 213 58
Matrix multiplication .................................................... 214 58
   With assignment ....................................................... 215 58
   real argument ........................................................ 216 58
   Transform argument ............................................... 218 58
Plain multiplication ..................................................... 220 59
   real argument ........................................................ 221 59
   Transform argument ............................................... 223 59
Matrix inversion ........................................................... 225 60
   const version (no assignment) ................................. 226 60
   Non-const version (with assignment) .......................... 232 63
Global variables ......................................................... 234 63
Global constants .......................................................... 236 64
# 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> <em>(shapes.web)</em></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> <em>(pictures.web)</em></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> <strong>Projections</strong></td>
<td>256</td>
</tr>
<tr>
<td><strong>namespace</strong> <strong>Sorting</strong></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>Constructors</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>Destructor</td>
<td>266</td>
</tr>
<tr>
<td>Assignment</td>
<td>267</td>
</tr>
<tr>
<td>Adding elements</td>
<td>268</td>
</tr>
<tr>
<td>Add <strong>Picture</strong></td>
<td>269</td>
</tr>
<tr>
<td>Add <strong>Shape</strong></td>
<td>270</td>
</tr>
<tr>
<td>Add <strong>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>Affine transformations</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</strong> version</td>
<td>283</td>
</tr>
<tr>
<td><strong>Point</strong> version</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>Multiplying <strong>transform</strong></td>
<td>291</td>
</tr>
<tr>
<td><strong>Show</strong></td>
<td>293</td>
</tr>
<tr>
<td><strong>Show</strong> <strong>transform</strong></td>
<td>295</td>
</tr>
<tr>
<td>Output</td>
<td>297</td>
</tr>
<tr>
<td><strong>Focus</strong> argument</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> <em>(points.web)</em></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>
<tr>
<td>point_pair and bool_point_pair</td>
<td>312 81</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>bool_point</td>
<td>313 81</td>
</tr>
<tr>
<td>bool_point_quadruple</td>
<td>315 83</td>
</tr>
<tr>
<td>Default Constructor for bool_point_quadruple</td>
<td>316 84</td>
</tr>
<tr>
<td>bool_real_point</td>
<td>317 84</td>
</tr>
<tr>
<td>Default Constructor for bool_real_point</td>
<td>318 85</td>
</tr>
<tr>
<td>Global constants</td>
<td>319 85</td>
</tr>
<tr>
<td>Constructors and setting functions</td>
<td>321 85</td>
</tr>
<tr>
<td>Initialize coordinates and limits</td>
<td>323 86</td>
</tr>
<tr>
<td>Default version</td>
<td>324 86</td>
</tr>
<tr>
<td>Three real values</td>
<td>326 87</td>
</tr>
<tr>
<td>Constructor</td>
<td>327 87</td>
</tr>
<tr>
<td>Setting function</td>
<td>329 87</td>
</tr>
<tr>
<td>Copy constructor</td>
<td>331 88</td>
</tr>
<tr>
<td>Setting function</td>
<td>333 88</td>
</tr>
<tr>
<td>Pseudo-constructor for dynamic allocation</td>
<td>335 89</td>
</tr>
<tr>
<td>Pointer argument</td>
<td>336 89</td>
</tr>
<tr>
<td>Reference argument</td>
<td>337 89</td>
</tr>
<tr>
<td>Destructor</td>
<td>338 89</td>
</tr>
<tr>
<td>Assignment</td>
<td>340 90</td>
</tr>
<tr>
<td>Set on free store</td>
<td>342 90</td>
</tr>
<tr>
<td>Clear</td>
<td>344 91</td>
</tr>
<tr>
<td>Clean</td>
<td>346 91</td>
</tr>
<tr>
<td>Returning elements and information</td>
<td>348 92</td>
</tr>
<tr>
<td>Is identity</td>
<td>349 92</td>
</tr>
<tr>
<td>Epsilon</td>
<td>350 92</td>
</tr>
<tr>
<td>Get Line</td>
<td>352 92</td>
</tr>
<tr>
<td>Getting coordinates</td>
<td>353 92</td>
</tr>
<tr>
<td>Get all coordinates</td>
<td>354 92</td>
</tr>
<tr>
<td>Non-const version</td>
<td>355 93</td>
</tr>
<tr>
<td>const version</td>
<td>357 93</td>
</tr>
<tr>
<td>Get coord</td>
<td>359 93</td>
</tr>
<tr>
<td>Non-const version</td>
<td>360 94</td>
</tr>
<tr>
<td>const version</td>
<td>362 94</td>
</tr>
<tr>
<td>Get x</td>
<td>364 95</td>
</tr>
<tr>
<td>Non-const version</td>
<td>365 95</td>
</tr>
<tr>
<td>const version</td>
<td>367 95</td>
</tr>
<tr>
<td>Get y</td>
<td>369 95</td>
</tr>
<tr>
<td>Non-const version</td>
<td>370 95</td>
</tr>
<tr>
<td>const version</td>
<td>372 96</td>
</tr>
<tr>
<td>Get z</td>
<td>374 96</td>
</tr>
<tr>
<td>Non-const version</td>
<td>375 96</td>
</tr>
<tr>
<td>const version</td>
<td>377 96</td>
</tr>
<tr>
<td>Get w</td>
<td>379 96</td>
</tr>
<tr>
<td>Non-const version</td>
<td>380 97</td>
</tr>
<tr>
<td>const version</td>
<td>382 97</td>
</tr>
<tr>
<td>Get transform</td>
<td>384 97</td>
</tr>
<tr>
<td>Get copy</td>
<td>385 97</td>
</tr>
<tr>
<td>Is on free store</td>
<td>387 98</td>
</tr>
<tr>
<td>Slope</td>
<td>389 98</td>
</tr>
<tr>
<td>Is on segment</td>
<td>392 99</td>
</tr>
<tr>
<td>Non-const version</td>
<td>393 100</td>
</tr>
<tr>
<td>const version</td>
<td>396  102</td>
</tr>
<tr>
<td>---------------</td>
<td>---------</td>
</tr>
<tr>
<td>Is on line</td>
<td>398  103</td>
</tr>
<tr>
<td>Is on Plane</td>
<td>400  103</td>
</tr>
<tr>
<td>Is in triangle</td>
<td>401  103</td>
</tr>
<tr>
<td>Transformations</td>
<td>402  103</td>
</tr>
<tr>
<td>Affine transformations</td>
<td>403  103</td>
</tr>
<tr>
<td>Rotation around the main axes</td>
<td>404  104</td>
</tr>
<tr>
<td>Scale</td>
<td>406  104</td>
</tr>
<tr>
<td>Shear</td>
<td>408  104</td>
</tr>
<tr>
<td>Shift</td>
<td>410  104</td>
</tr>
<tr>
<td>Point versions</td>
<td>411  104</td>
</tr>
<tr>
<td>Three real arguments</td>
<td>412  105</td>
</tr>
<tr>
<td>Point argument</td>
<td>414  105</td>
</tr>
<tr>
<td>Transform version</td>
<td>416  105</td>
</tr>
<tr>
<td>Picture version</td>
<td>417  105</td>
</tr>
<tr>
<td>Shift times</td>
<td>418  105</td>
</tr>
<tr>
<td>Three real arguments</td>
<td>419  106</td>
</tr>
<tr>
<td>Point argument</td>
<td>421  106</td>
</tr>
<tr>
<td>Alignment with an axis</td>
<td>423  106</td>
</tr>
<tr>
<td>Normalize point</td>
<td>433  110</td>
</tr>
<tr>
<td>Rotation around an arbitrary axis</td>
<td>434  110</td>
</tr>
<tr>
<td>Point versions</td>
<td>435  110</td>
</tr>
<tr>
<td>Point arguments</td>
<td>436  111</td>
</tr>
<tr>
<td>Path argument</td>
<td>438  111</td>
</tr>
<tr>
<td>Transform version</td>
<td>439  112</td>
</tr>
<tr>
<td>Picture version</td>
<td>440  113</td>
</tr>
<tr>
<td>Projection</td>
<td>441  113</td>
</tr>
<tr>
<td>Focus argument</td>
<td>442  113</td>
</tr>
<tr>
<td>Parallel projection</td>
<td>444  115</td>
</tr>
<tr>
<td>Perspective projection</td>
<td>445  116</td>
</tr>
<tr>
<td>No Focus argument</td>
<td>446  117</td>
</tr>
<tr>
<td>Applying transformations</td>
<td>448  117</td>
</tr>
<tr>
<td>Set transform to identity</td>
<td>450  118</td>
</tr>
<tr>
<td>Drawing</td>
<td>452  119</td>
</tr>
<tr>
<td>Drawdot</td>
<td>453  119</td>
</tr>
<tr>
<td>Normal version</td>
<td>454  119</td>
</tr>
<tr>
<td>Picture argument first</td>
<td>456  120</td>
</tr>
<tr>
<td>Undrawdot</td>
<td>458  120</td>
</tr>
<tr>
<td>Picture argument first</td>
<td>460  120</td>
</tr>
<tr>
<td>Draw</td>
<td>462  121</td>
</tr>
<tr>
<td>Normal version</td>
<td>463  121</td>
</tr>
<tr>
<td>Picture argument first</td>
<td>464  121</td>
</tr>
<tr>
<td>Draw arrow</td>
<td>465  121</td>
</tr>
<tr>
<td>Normal version</td>
<td>466  121</td>
</tr>
<tr>
<td>Picture argument first</td>
<td>467  122</td>
</tr>
<tr>
<td>Undraw</td>
<td>468  122</td>
</tr>
<tr>
<td>Normal version</td>
<td>469  122</td>
</tr>
<tr>
<td>Picture argument first</td>
<td>470  122</td>
</tr>
<tr>
<td>Draw help</td>
<td>471  122</td>
</tr>
<tr>
<td>Normal version</td>
<td>472  122</td>
</tr>
<tr>
<td>Picture argument first</td>
<td>473  123</td>
</tr>
<tr>
<td>Showing</td>
<td>474  123</td>
</tr>
</tbody>
</table>
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page 1</th>
<th>Page 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show</td>
<td>475</td>
<td>123</td>
</tr>
<tr>
<td>Show transform</td>
<td>477</td>
<td>124</td>
</tr>
<tr>
<td>Outputting</td>
<td>479</td>
<td>125</td>
</tr>
<tr>
<td>Output operator</td>
<td>480</td>
<td>125</td>
</tr>
<tr>
<td>Suppress output</td>
<td>482</td>
<td>125</td>
</tr>
<tr>
<td>Unsuppress output</td>
<td>484</td>
<td>125</td>
</tr>
<tr>
<td>Extract</td>
<td>486</td>
<td>126</td>
</tr>
<tr>
<td>Get extremes</td>
<td>488</td>
<td>127</td>
</tr>
<tr>
<td>Get minimum z</td>
<td>489</td>
<td>127</td>
</tr>
<tr>
<td>Get maximum z</td>
<td>491</td>
<td>128</td>
</tr>
<tr>
<td>Get mean z</td>
<td>493</td>
<td>128</td>
</tr>
<tr>
<td>Set extremes</td>
<td>495</td>
<td>128</td>
</tr>
<tr>
<td>Comparison classes</td>
<td>497</td>
<td>129</td>
</tr>
<tr>
<td>Compare minimum z</td>
<td>498</td>
<td>130</td>
</tr>
<tr>
<td>Compare maximum z</td>
<td>499</td>
<td>130</td>
</tr>
<tr>
<td>Compare mean z</td>
<td>500</td>
<td>131</td>
</tr>
<tr>
<td>Output</td>
<td>501</td>
<td>131</td>
</tr>
<tr>
<td>Labelling</td>
<td>503</td>
<td>132</td>
</tr>
<tr>
<td>Label</td>
<td>504</td>
<td>132</td>
</tr>
<tr>
<td>string argument</td>
<td>505</td>
<td>133</td>
</tr>
<tr>
<td>short argument</td>
<td>507</td>
<td>135</td>
</tr>
<tr>
<td>Dotlabel</td>
<td>509</td>
<td>136</td>
</tr>
<tr>
<td>string argument</td>
<td>510</td>
<td>137</td>
</tr>
<tr>
<td>short argument</td>
<td>512</td>
<td>137</td>
</tr>
<tr>
<td>Get copy of Label</td>
<td>515</td>
<td>138</td>
</tr>
<tr>
<td>Output Labels</td>
<td>516</td>
<td>138</td>
</tr>
<tr>
<td>Matrix operations</td>
<td>517</td>
<td>138</td>
</tr>
<tr>
<td>Multiplication by a Transform with assignment</td>
<td>518</td>
<td>138</td>
</tr>
<tr>
<td>Vector operations</td>
<td>520</td>
<td>139</td>
</tr>
<tr>
<td>Vector addition</td>
<td>521</td>
<td>139</td>
</tr>
<tr>
<td>Vector addition with assignment</td>
<td>523</td>
<td>139</td>
</tr>
<tr>
<td>Vector subtraction</td>
<td>525</td>
<td>140</td>
</tr>
<tr>
<td>Vector subtraction with assignment</td>
<td>527</td>
<td>140</td>
</tr>
<tr>
<td>Vector-scalar multiplication with assignment</td>
<td>529</td>
<td>140</td>
</tr>
<tr>
<td>Vector-scalar multiplication</td>
<td>531</td>
<td>141</td>
</tr>
<tr>
<td>Member version (Point first)</td>
<td>532</td>
<td>141</td>
</tr>
<tr>
<td>Non-member version (scalar first)</td>
<td>534</td>
<td>141</td>
</tr>
<tr>
<td>Unary minus</td>
<td>536</td>
<td>141</td>
</tr>
<tr>
<td>Vector-scalar division with assignment</td>
<td>538</td>
<td>142</td>
</tr>
<tr>
<td>Vector-scalar division</td>
<td>540</td>
<td>142</td>
</tr>
<tr>
<td>Dot product</td>
<td>542</td>
<td>143</td>
</tr>
<tr>
<td>Cross product</td>
<td>544</td>
<td>143</td>
</tr>
<tr>
<td>Magnitude</td>
<td>546</td>
<td>144</td>
</tr>
<tr>
<td>Angle between two vectors</td>
<td>548</td>
<td>145</td>
</tr>
<tr>
<td>Unit vector</td>
<td>550</td>
<td>146</td>
</tr>
<tr>
<td>With assignment</td>
<td>551</td>
<td>146</td>
</tr>
<tr>
<td>const (no assignment)</td>
<td>553</td>
<td>147</td>
</tr>
<tr>
<td>Mediation</td>
<td>555</td>
<td>147</td>
</tr>
<tr>
<td>Get normal</td>
<td>557</td>
<td>148</td>
</tr>
<tr>
<td>Comparison</td>
<td>558</td>
<td>148</td>
</tr>
<tr>
<td>Equality</td>
<td>559</td>
<td>148</td>
</tr>
<tr>
<td>Non-const version</td>
<td>560</td>
<td>148</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>const version</td>
<td>567</td>
<td></td>
</tr>
<tr>
<td>Inequality</td>
<td>569</td>
<td></td>
</tr>
<tr>
<td>Intersection</td>
<td>571</td>
<td></td>
</tr>
<tr>
<td>Vector version</td>
<td>572</td>
<td></td>
</tr>
<tr>
<td>Trace version</td>
<td>573</td>
<td></td>
</tr>
<tr>
<td><strong>Picture functions</strong></td>
<td>586</td>
<td></td>
</tr>
<tr>
<td>Assignment operator</td>
<td>587</td>
<td></td>
</tr>
<tr>
<td>Copy constructor</td>
<td>588</td>
<td></td>
</tr>
<tr>
<td>Combining Pictures</td>
<td>589</td>
<td></td>
</tr>
<tr>
<td>Clear Picture</td>
<td>590</td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>591</td>
<td></td>
</tr>
<tr>
<td><strong>Focus</strong></td>
<td>592</td>
<td></td>
</tr>
<tr>
<td><strong>Focus class definition</strong></td>
<td>599</td>
<td></td>
</tr>
<tr>
<td>Constructors and setting functions</td>
<td>600</td>
<td></td>
</tr>
<tr>
<td>Default constructor</td>
<td>601</td>
<td></td>
</tr>
<tr>
<td>real arguments</td>
<td>602</td>
<td></td>
</tr>
<tr>
<td>Constructor</td>
<td>603</td>
<td></td>
</tr>
<tr>
<td>Setting function</td>
<td>604</td>
<td></td>
</tr>
<tr>
<td><strong>Point</strong></td>
<td>608</td>
<td></td>
</tr>
<tr>
<td>Constructor</td>
<td>609</td>
<td></td>
</tr>
<tr>
<td>Setting function</td>
<td>610</td>
<td></td>
</tr>
<tr>
<td>Assignment</td>
<td>611</td>
<td></td>
</tr>
<tr>
<td>Reset angle</td>
<td>612</td>
<td></td>
</tr>
<tr>
<td>Show</td>
<td>613</td>
<td></td>
</tr>
<tr>
<td>Returning elements and information</td>
<td>614</td>
<td></td>
</tr>
<tr>
<td>Get position</td>
<td>619</td>
<td></td>
</tr>
<tr>
<td>Get direction</td>
<td>620</td>
<td></td>
</tr>
<tr>
<td>Get distance</td>
<td>621</td>
<td></td>
</tr>
<tr>
<td>Get distance</td>
<td>622</td>
<td></td>
</tr>
<tr>
<td>Get up</td>
<td>623</td>
<td></td>
</tr>
<tr>
<td>Get transform</td>
<td>624</td>
<td></td>
</tr>
<tr>
<td>Get transform element</td>
<td>625</td>
<td></td>
</tr>
<tr>
<td>Get transform element</td>
<td>626</td>
<td></td>
</tr>
<tr>
<td>Get persp</td>
<td>627</td>
<td></td>
</tr>
<tr>
<td>Get persp element</td>
<td>628</td>
<td></td>
</tr>
<tr>
<td>Global variables</td>
<td>630</td>
<td></td>
</tr>
<tr>
<td><strong>Putting Point and Focus together</strong></td>
<td>632</td>
<td></td>
</tr>
<tr>
<td>This is what’s compiled</td>
<td>633</td>
<td></td>
</tr>
<tr>
<td>This is what’s written to points.h</td>
<td>634</td>
<td></td>
</tr>
<tr>
<td><strong>Line (lines.web)</strong></td>
<td>635</td>
<td></td>
</tr>
<tr>
<td>Include files</td>
<td>636</td>
<td></td>
</tr>
<tr>
<td><strong>Line struct definition</strong></td>
<td>637</td>
<td></td>
</tr>
<tr>
<td>Constructors</td>
<td>638</td>
<td></td>
</tr>
<tr>
<td>Default constructor</td>
<td>639</td>
<td></td>
</tr>
<tr>
<td>Copy constructor</td>
<td>641</td>
<td></td>
</tr>
<tr>
<td>Assignment</td>
<td>642</td>
<td></td>
</tr>
<tr>
<td>Get Line</td>
<td>643</td>
<td></td>
</tr>
<tr>
<td>Get Path</td>
<td>645</td>
<td></td>
</tr>
<tr>
<td>Intersection</td>
<td>646</td>
<td></td>
</tr>
<tr>
<td>Get distance</td>
<td>647</td>
<td></td>
</tr>
<tr>
<td>Show</td>
<td>648</td>
<td></td>
</tr>
<tr>
<td>Global constants for Line</td>
<td>652</td>
<td></td>
</tr>
<tr>
<td></td>
<td>654</td>
<td></td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>-----------------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>Putting Line together</td>
<td>656</td>
<td></td>
</tr>
<tr>
<td>Plane (planes.web)</td>
<td>659</td>
<td></td>
</tr>
<tr>
<td>Include files</td>
<td>660</td>
<td></td>
</tr>
<tr>
<td>Plane struct definition</td>
<td>661</td>
<td></td>
</tr>
<tr>
<td>Constructors</td>
<td>662</td>
<td></td>
</tr>
<tr>
<td>Default constructor</td>
<td>663</td>
<td></td>
</tr>
<tr>
<td>Copy constructor</td>
<td>665</td>
<td></td>
</tr>
<tr>
<td>Point arguments</td>
<td>667</td>
<td></td>
</tr>
<tr>
<td>Assignment</td>
<td>669</td>
<td></td>
</tr>
<tr>
<td>Comparing Planes</td>
<td>671</td>
<td></td>
</tr>
<tr>
<td>Equality</td>
<td>672</td>
<td></td>
</tr>
<tr>
<td>Inequality</td>
<td>674</td>
<td></td>
</tr>
<tr>
<td>Get distance</td>
<td>676</td>
<td></td>
</tr>
<tr>
<td>Point argument</td>
<td>677</td>
<td></td>
</tr>
<tr>
<td>No argument</td>
<td>679</td>
<td></td>
</tr>
<tr>
<td>Point is on Plane</td>
<td>681</td>
<td></td>
</tr>
<tr>
<td>Intersection</td>
<td>682</td>
<td></td>
</tr>
<tr>
<td>Intersection with a line</td>
<td>683</td>
<td></td>
</tr>
<tr>
<td>Point arguments</td>
<td>684</td>
<td></td>
</tr>
<tr>
<td>Path argument</td>
<td>686</td>
<td></td>
</tr>
<tr>
<td>Intersection of two Planes</td>
<td>687</td>
<td></td>
</tr>
<tr>
<td>Show</td>
<td>689</td>
<td></td>
</tr>
<tr>
<td>Global constants for Plane</td>
<td>691</td>
<td></td>
</tr>
<tr>
<td>Putting Plane together</td>
<td>693</td>
<td></td>
</tr>
<tr>
<td>Path (paths.web)</td>
<td>696</td>
<td></td>
</tr>
<tr>
<td>Include files</td>
<td>697</td>
<td></td>
</tr>
<tr>
<td>Path class definition</td>
<td>698</td>
<td></td>
</tr>
<tr>
<td>Static member variable definitions</td>
<td>699</td>
<td></td>
</tr>
<tr>
<td>Assignment</td>
<td>700</td>
<td></td>
</tr>
<tr>
<td>Constructors and setting functions</td>
<td>702</td>
<td></td>
</tr>
<tr>
<td>Discard points and connectors</td>
<td>703</td>
<td></td>
</tr>
<tr>
<td>Default constructor</td>
<td>704</td>
<td></td>
</tr>
<tr>
<td>Lines</td>
<td>706</td>
<td></td>
</tr>
<tr>
<td>Constructor</td>
<td>707</td>
<td></td>
</tr>
<tr>
<td>Setting function</td>
<td>709</td>
<td></td>
</tr>
<tr>
<td>Points and one type of connector</td>
<td>711</td>
<td></td>
</tr>
<tr>
<td>Constructor</td>
<td>712</td>
<td></td>
</tr>
<tr>
<td>Setting function</td>
<td>714</td>
<td></td>
</tr>
<tr>
<td>Variable number of Points and connectors</td>
<td>716</td>
<td></td>
</tr>
<tr>
<td>Constructor</td>
<td>717</td>
<td></td>
</tr>
<tr>
<td>Setting function</td>
<td>719</td>
<td></td>
</tr>
<tr>
<td>Copy constructor</td>
<td>721</td>
<td></td>
</tr>
<tr>
<td>Pseudo-constructor for dynamic allocation</td>
<td>723</td>
<td></td>
</tr>
<tr>
<td>Pointer argument</td>
<td>724</td>
<td></td>
</tr>
<tr>
<td>Reference argument</td>
<td>725</td>
<td></td>
</tr>
<tr>
<td>Destructor</td>
<td>726</td>
<td></td>
</tr>
<tr>
<td>Clear</td>
<td>728</td>
<td></td>
</tr>
<tr>
<td>Get copy</td>
<td>730</td>
<td></td>
</tr>
<tr>
<td>Set on free store</td>
<td>732</td>
<td></td>
</tr>
<tr>
<td>Setting drawing and filling data</td>
<td>734</td>
<td></td>
</tr>
<tr>
<td>Set fill, draw, value</td>
<td>735</td>
<td></td>
</tr>
<tr>
<td>Set draw color</td>
<td>737</td>
<td></td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>Color version</td>
<td>738</td>
<td></td>
</tr>
<tr>
<td>Color pointer version</td>
<td>740</td>
<td></td>
</tr>
<tr>
<td>Set fill color</td>
<td>742</td>
<td></td>
</tr>
<tr>
<td>Color version</td>
<td>743</td>
<td></td>
</tr>
<tr>
<td>Color pointer version</td>
<td>745</td>
<td></td>
</tr>
<tr>
<td>Set dash pattern</td>
<td>747</td>
<td></td>
</tr>
<tr>
<td>Set pen</td>
<td>749</td>
<td></td>
</tr>
<tr>
<td>Set connectors</td>
<td>751</td>
<td></td>
</tr>
<tr>
<td>Transformations</td>
<td>753</td>
<td></td>
</tr>
<tr>
<td>Affine transformations</td>
<td>754</td>
<td></td>
</tr>
<tr>
<td>Rotation</td>
<td>755</td>
<td></td>
</tr>
<tr>
<td>Rotation around the main axes</td>
<td>756</td>
<td></td>
</tr>
<tr>
<td>Rotation around an arbitrary axis</td>
<td>758</td>
<td></td>
</tr>
<tr>
<td>Transform version</td>
<td>759</td>
<td></td>
</tr>
<tr>
<td>Point version</td>
<td>760</td>
<td></td>
</tr>
<tr>
<td>Path versions</td>
<td>761</td>
<td></td>
</tr>
<tr>
<td>Point arguments</td>
<td>762</td>
<td></td>
</tr>
<tr>
<td>Path arguments</td>
<td>764</td>
<td></td>
</tr>
<tr>
<td>Scale</td>
<td>766</td>
<td></td>
</tr>
<tr>
<td>Shear</td>
<td>768</td>
<td></td>
</tr>
<tr>
<td>Shift</td>
<td>770</td>
<td></td>
</tr>
<tr>
<td>real arguments</td>
<td>771</td>
<td></td>
</tr>
<tr>
<td>Point argument</td>
<td>773</td>
<td></td>
</tr>
<tr>
<td>Shift times</td>
<td>775</td>
<td></td>
</tr>
<tr>
<td>real arguments</td>
<td>776</td>
<td></td>
</tr>
<tr>
<td>Point argument</td>
<td>778</td>
<td></td>
</tr>
<tr>
<td>Applying transformations</td>
<td>780</td>
<td></td>
</tr>
<tr>
<td>Multiplying by a Transform</td>
<td>781</td>
<td></td>
</tr>
<tr>
<td>Applying transform to points</td>
<td>783</td>
<td></td>
</tr>
<tr>
<td>Projection</td>
<td>785</td>
<td></td>
</tr>
<tr>
<td>Functions for lines</td>
<td>787</td>
<td></td>
</tr>
<tr>
<td>Alignment with an axis</td>
<td>788</td>
<td></td>
</tr>
<tr>
<td>For lines</td>
<td>789</td>
<td></td>
</tr>
<tr>
<td>No assignment</td>
<td>790</td>
<td></td>
</tr>
<tr>
<td>With assignment</td>
<td>792</td>
<td></td>
</tr>
<tr>
<td>For non-lines</td>
<td>794</td>
<td></td>
</tr>
<tr>
<td>Adding Points to Paths</td>
<td>796</td>
<td></td>
</tr>
<tr>
<td>With assignment</td>
<td>797</td>
<td></td>
</tr>
<tr>
<td>Without assignment</td>
<td>799</td>
<td></td>
</tr>
<tr>
<td>Adding connectors to Paths</td>
<td>801</td>
<td></td>
</tr>
<tr>
<td>Concatenating Paths</td>
<td>803</td>
<td></td>
</tr>
<tr>
<td>Versions using “&amp;”</td>
<td>804</td>
<td></td>
</tr>
<tr>
<td>With assignment</td>
<td>805</td>
<td></td>
</tr>
<tr>
<td>Without assignment</td>
<td>810</td>
<td></td>
</tr>
<tr>
<td>Appending with a connector argument</td>
<td>812</td>
<td></td>
</tr>
<tr>
<td>Drawing and filling</td>
<td>815</td>
<td></td>
</tr>
<tr>
<td>Draw</td>
<td>816</td>
<td></td>
</tr>
<tr>
<td>Path versions</td>
<td>817</td>
<td></td>
</tr>
<tr>
<td>Normal version</td>
<td>818</td>
<td></td>
</tr>
<tr>
<td>Picture argument first</td>
<td>820</td>
<td></td>
</tr>
<tr>
<td>Point versions</td>
<td>822</td>
<td></td>
</tr>
<tr>
<td>Normal version</td>
<td>823</td>
<td></td>
</tr>
<tr>
<td>Module</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>Picture argument first</td>
<td>824</td>
<td></td>
</tr>
<tr>
<td>Path versions</td>
<td>825</td>
<td></td>
</tr>
<tr>
<td>Normal version</td>
<td>826</td>
<td></td>
</tr>
<tr>
<td>Point versions</td>
<td>827</td>
<td></td>
</tr>
<tr>
<td>Normal version</td>
<td>828</td>
<td></td>
</tr>
<tr>
<td>Picture argument first</td>
<td>829</td>
<td></td>
</tr>
<tr>
<td>Fill</td>
<td>830</td>
<td></td>
</tr>
<tr>
<td>Normal version</td>
<td>831</td>
<td></td>
</tr>
<tr>
<td>Picture argument first</td>
<td>832</td>
<td></td>
</tr>
<tr>
<td>Filldraw</td>
<td>833</td>
<td></td>
</tr>
<tr>
<td>Normal version</td>
<td>834</td>
<td></td>
</tr>
<tr>
<td>Picture argument first</td>
<td>835</td>
<td></td>
</tr>
<tr>
<td>Undraw</td>
<td>836</td>
<td></td>
</tr>
<tr>
<td>Path versions</td>
<td>837</td>
<td></td>
</tr>
<tr>
<td>Normal version</td>
<td>838</td>
<td></td>
</tr>
<tr>
<td>Picture argument first</td>
<td>839</td>
<td></td>
</tr>
<tr>
<td>Point versions</td>
<td>840</td>
<td></td>
</tr>
<tr>
<td>Normal version</td>
<td>841</td>
<td></td>
</tr>
<tr>
<td>Picture argument first</td>
<td>842</td>
<td></td>
</tr>
<tr>
<td>Unfill</td>
<td>843</td>
<td></td>
</tr>
<tr>
<td>Normal version</td>
<td>844</td>
<td></td>
</tr>
<tr>
<td>Picture argument first</td>
<td>845</td>
<td></td>
</tr>
<tr>
<td>Labelling</td>
<td>846</td>
<td></td>
</tr>
<tr>
<td>Label</td>
<td>847</td>
<td></td>
</tr>
<tr>
<td>Normal version</td>
<td>848</td>
<td></td>
</tr>
<tr>
<td>Picture argument first</td>
<td>849</td>
<td></td>
</tr>
<tr>
<td>Outputting</td>
<td>850</td>
<td></td>
</tr>
<tr>
<td>Extract</td>
<td>851</td>
<td></td>
</tr>
<tr>
<td>Set extremes</td>
<td>852</td>
<td></td>
</tr>
<tr>
<td>Get extremes</td>
<td>853</td>
<td></td>
</tr>
<tr>
<td>Get minimum z</td>
<td>854</td>
<td></td>
</tr>
<tr>
<td>Get maximum z</td>
<td>855</td>
<td></td>
</tr>
<tr>
<td>Get mean z</td>
<td>856</td>
<td></td>
</tr>
<tr>
<td>Suppress output</td>
<td>857</td>
<td></td>
</tr>
<tr>
<td>Unsuppress output</td>
<td>858</td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>859</td>
<td></td>
</tr>
<tr>
<td>Showing</td>
<td>860</td>
<td></td>
</tr>
<tr>
<td>Show</td>
<td>861</td>
<td></td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Colors</td>
<td>911</td>
</tr>
<tr>
<td>Returning elements and information</td>
<td>912</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_C1Plane_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>
<tr>
<td>TABLE OF CONTENTS</td>
<td></td>
</tr>
<tr>
<td>--------------------</td>
<td></td>
</tr>
<tr>
<td>3DLDF-1.1.5.l</td>
<td></td>
</tr>
</tbody>
</table>

<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>Path arguments</td>
<td>1008</td>
</tr>
<tr>
<td>Reg_C1Plane_Curve segments</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 Reg_C1Plane_Curve together</td>
<td>1015</td>
</tr>
<tr>
<td>Polygon (polygons.web)</td>
<td>1017</td>
</tr>
<tr>
<td>Include files</td>
<td>1018</td>
</tr>
<tr>
<td>Polygon class definition</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>Intersections</td>
<td>1026</td>
</tr>
<tr>
<td>Intersection with a line</td>
<td>1027</td>
</tr>
<tr>
<td>Point version</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>Path version</td>
<td>1037</td>
</tr>
<tr>
<td>Intersection with another Polygon</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>Transformations</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>Point arguments</td>
<td>1050</td>
</tr>
<tr>
<td>Path argument</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>Point argument</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>Point argument</td>
<td>1066</td>
</tr>
<tr>
<td>Reg_Polygon (polygons.web)</td>
<td>1068</td>
</tr>
<tr>
<td>Reg_Polygon class definition</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>Pseudo-constructor for dynamic allocation</td>
<td>1082 305</td>
</tr>
<tr>
<td>Pointer argument</td>
<td>1083 305</td>
</tr>
<tr>
<td>Reference argument</td>
<td>1084 305</td>
</tr>
<tr>
<td>Destructor</td>
<td>1085 305</td>
</tr>
<tr>
<td>Returning elements and information</td>
<td>1086 305</td>
</tr>
<tr>
<td>Get radius</td>
<td>1087 305</td>
</tr>
<tr>
<td>Circles</td>
<td>1088 306</td>
</tr>
<tr>
<td>Enclosed circle</td>
<td>1089 306</td>
</tr>
<tr>
<td>Draw enclosed circle</td>
<td>1090 306</td>
</tr>
<tr>
<td>Normal version</td>
<td>1091 306</td>
</tr>
<tr>
<td>Picture argument first</td>
<td>1092 306</td>
</tr>
<tr>
<td>Surrounding circle</td>
<td>1093 306</td>
</tr>
<tr>
<td>Draw surrounding circle</td>
<td>1094 306</td>
</tr>
<tr>
<td>Normal version</td>
<td>1095 306</td>
</tr>
<tr>
<td>Picture argument first</td>
<td>1096 307</td>
</tr>
<tr>
<td>Putting polygons together</td>
<td>1097 307</td>
</tr>
<tr>
<td><strong>Rectangle (rectangles.web)</strong></td>
<td>1099 308</td>
</tr>
<tr>
<td>Include files</td>
<td>1100 308</td>
</tr>
<tr>
<td><strong>Rectangle class definition</strong></td>
<td>1101 308</td>
</tr>
<tr>
<td>Constructors and setting functions</td>
<td>1102 308</td>
</tr>
<tr>
<td>Default constructor</td>
<td>1103 309</td>
</tr>
<tr>
<td>Center, lengths, and angles</td>
<td>1105 309</td>
</tr>
<tr>
<td>Constructor</td>
<td>1106 309</td>
</tr>
<tr>
<td>Setting function</td>
<td>1108 310</td>
</tr>
<tr>
<td><strong>Four Points</strong></td>
<td>1110 310</td>
</tr>
<tr>
<td>Constructor</td>
<td>1111 311</td>
</tr>
<tr>
<td>Setting function</td>
<td>1113 311</td>
</tr>
<tr>
<td>Pseudo-constructor for dynamic allocation</td>
<td>1115 312</td>
</tr>
<tr>
<td>Pointer argument</td>
<td>1116 312</td>
</tr>
<tr>
<td>Reference argument</td>
<td>1117 312</td>
</tr>
<tr>
<td>Destructor</td>
<td>1118 312</td>
</tr>
<tr>
<td>Assignment</td>
<td>1119 312</td>
</tr>
<tr>
<td>Returning Elements and information</td>
<td>1121 313</td>
</tr>
<tr>
<td>Is rectangular</td>
<td>1122 313</td>
</tr>
<tr>
<td>Returning Points</td>
<td>1124 314</td>
</tr>
<tr>
<td>Corner</td>
<td>1125 314</td>
</tr>
<tr>
<td>Get Mid-point</td>
<td>1127 314</td>
</tr>
<tr>
<td>Getting axes</td>
<td>1129 315</td>
</tr>
<tr>
<td>Get axis.h</td>
<td>1130 315</td>
</tr>
<tr>
<td>Get axis.v</td>
<td>1132 315</td>
</tr>
<tr>
<td><strong>Ellipses</strong></td>
<td>1134 316</td>
</tr>
<tr>
<td>Surrounding Ellipse</td>
<td>1135 316</td>
</tr>
<tr>
<td>Enclosed Ellipse</td>
<td>1136 316</td>
</tr>
<tr>
<td>Draw surrounding Ellipse</td>
<td>1137 316</td>
</tr>
<tr>
<td>Draw enclosed Ellipse</td>
<td>1138 316</td>
</tr>
<tr>
<td>Putting Rectangle together</td>
<td>1139 316</td>
</tr>
<tr>
<td><strong>Ellipse (ellipses.web)</strong></td>
<td>1141 318</td>
</tr>
<tr>
<td>Include files</td>
<td>1142 318</td>
</tr>
<tr>
<td><strong>Ellipse class definition</strong></td>
<td>1143 318</td>
</tr>
<tr>
<td>Static data members</td>
<td>1144 319</td>
</tr>
<tr>
<td>Constructors</td>
<td>1145 319</td>
</tr>
<tr>
<td>Default constructor</td>
<td>1146 319</td>
</tr>
<tr>
<td>Center, lengths, and angles of rotation</td>
<td>1148 319</td>
</tr>
<tr>
<td>Constructor</td>
<td>1149 319</td>
</tr>
<tr>
<td>Setting function</td>
<td>1151 321</td>
</tr>
<tr>
<td>Pseudo-constructor for dynamic allocation</td>
<td>1153 321</td>
</tr>
<tr>
<td>Pointer argument</td>
<td>1154 321</td>
</tr>
<tr>
<td>Reference argument</td>
<td>1155 322</td>
</tr>
<tr>
<td>Destructor</td>
<td>1156 322</td>
</tr>
<tr>
<td>Assignment</td>
<td>1157 322</td>
</tr>
<tr>
<td>Labelling</td>
<td>1159 322</td>
</tr>
<tr>
<td>Label</td>
<td>1160 322</td>
</tr>
<tr>
<td>Dotlabel</td>
<td>1162 323</td>
</tr>
<tr>
<td>Returning elements and information</td>
<td>1163 323</td>
</tr>
<tr>
<td>Is elliptical</td>
<td>1164 323</td>
</tr>
<tr>
<td>Is quadratic</td>
<td>1166 325</td>
</tr>
<tr>
<td>Is cubic</td>
<td>1167 325</td>
</tr>
<tr>
<td>Is quartic</td>
<td>1169 326</td>
</tr>
<tr>
<td>Solve</td>
<td>1171 326</td>
</tr>
<tr>
<td>Get coefficients</td>
<td>1174 328</td>
</tr>
<tr>
<td>Get center</td>
<td>1176 328</td>
</tr>
<tr>
<td>Non-const version</td>
<td>1177 329</td>
</tr>
<tr>
<td>const version</td>
<td>1179 329</td>
</tr>
<tr>
<td>Get focus</td>
<td>1181 329</td>
</tr>
<tr>
<td>Non-const version</td>
<td>1182 329</td>
</tr>
<tr>
<td>const version</td>
<td>1184 330</td>
</tr>
<tr>
<td>Get linear eccentricity</td>
<td>1186 331</td>
</tr>
<tr>
<td>Get numerical eccentricity</td>
<td>1188 331</td>
</tr>
<tr>
<td>Get axes</td>
<td>1190 332</td>
</tr>
<tr>
<td>Get vertical axis</td>
<td>1191 332</td>
</tr>
<tr>
<td>const version</td>
<td>1192 332</td>
</tr>
<tr>
<td>Non-const version</td>
<td>1194 332</td>
</tr>
<tr>
<td>Get horizontal axis</td>
<td>1196 332</td>
</tr>
<tr>
<td>const version</td>
<td>1197 333</td>
</tr>
<tr>
<td>Non-const version</td>
<td>1199 333</td>
</tr>
<tr>
<td>Angle point</td>
<td>1201 333</td>
</tr>
<tr>
<td>Equality</td>
<td>1203 334</td>
</tr>
<tr>
<td>Location of a point</td>
<td>1205 335</td>
</tr>
<tr>
<td>Intersection points</td>
<td>1207 336</td>
</tr>
<tr>
<td>Point arguments</td>
<td>1208 336</td>
</tr>
<tr>
<td>Path argument</td>
<td>1210 337</td>
</tr>
<tr>
<td>Ellipse argument</td>
<td>1212 337</td>
</tr>
<tr>
<td>Check intersection point locations</td>
<td>1228 355</td>
</tr>
<tr>
<td>Transformations</td>
<td>1229 355</td>
</tr>
<tr>
<td>Performing a transformation</td>
<td>1230 355</td>
</tr>
<tr>
<td>Do transform</td>
<td>1231 356</td>
</tr>
<tr>
<td>Operator</td>
<td>1233 358</td>
</tr>
<tr>
<td>Rotation around the main axes</td>
<td>1235 358</td>
</tr>
<tr>
<td>Scale</td>
<td>1237 358</td>
</tr>
<tr>
<td>Shear</td>
<td>1239 359</td>
</tr>
<tr>
<td>Shift</td>
<td>1241 359</td>
</tr>
<tr>
<td>real arguments</td>
<td>1242 359</td>
</tr>
<tr>
<td>Point argument</td>
<td>1244 359</td>
</tr>
<tr>
<td>Shift times</td>
<td>1246 360</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>3DLDF-1.1.5.1</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>real arguments</td>
<td>1247 360</td>
</tr>
<tr>
<td>Point argument</td>
<td>1249 360</td>
</tr>
<tr>
<td>Rotation around an arbitrary axis</td>
<td>1251 360</td>
</tr>
<tr>
<td>Point arguments</td>
<td>1252 361</td>
</tr>
<tr>
<td>Path arguments</td>
<td>1254 361</td>
</tr>
<tr>
<td>Rectangles</td>
<td>1256 362</td>
</tr>
<tr>
<td>Surrounding rectangle</td>
<td>1257 362</td>
</tr>
<tr>
<td>Inscribed rectangle</td>
<td>1259 363</td>
</tr>
<tr>
<td>Draw surrounding rectangle</td>
<td>1261 364</td>
</tr>
<tr>
<td>Draw inscribed rectangle</td>
<td>1263 364</td>
</tr>
<tr>
<td>Rectangle functions</td>
<td>1265 365</td>
</tr>
<tr>
<td>Ellipses</td>
<td>1266 365</td>
</tr>
<tr>
<td>Surrounding Ellipse</td>
<td>1267 366</td>
</tr>
<tr>
<td>Enclosed Ellipse</td>
<td>1268 367</td>
</tr>
<tr>
<td>Draw surrounding Ellipse</td>
<td>1269 367</td>
</tr>
<tr>
<td>Draw enclosed Ellipse</td>
<td>1270 368</td>
</tr>
<tr>
<td>Putting Ellipse together</td>
<td>1271 368</td>
</tr>
<tr>
<td>Circle (circles.web)</td>
<td>1273 369</td>
</tr>
<tr>
<td>Include files</td>
<td>1274 369</td>
</tr>
<tr>
<td>Circle class definition</td>
<td>1275 369</td>
</tr>
<tr>
<td>Constructors and setting functions</td>
<td>1277 370</td>
</tr>
<tr>
<td>Default constructor</td>
<td>1278 370</td>
</tr>
<tr>
<td>Center, diameters and angles</td>
<td>1280 370</td>
</tr>
<tr>
<td>Constructor</td>
<td>1281 370</td>
</tr>
<tr>
<td>Setting function</td>
<td>1283 371</td>
</tr>
<tr>
<td>Pseudo-constructor for dynamic allocation</td>
<td>1285 372</td>
</tr>
<tr>
<td>Pointer argument</td>
<td>1286 372</td>
</tr>
<tr>
<td>Reference argument</td>
<td>1287 372</td>
</tr>
<tr>
<td>Destructor</td>
<td>1288 372</td>
</tr>
<tr>
<td>Assignment</td>
<td>1289 372</td>
</tr>
<tr>
<td>Circle argument</td>
<td>1290 372</td>
</tr>
<tr>
<td>Ellipse argument</td>
<td>1292 373</td>
</tr>
<tr>
<td>Returning elements and information</td>
<td>1294 374</td>
</tr>
<tr>
<td>Is circular</td>
<td>1295 374</td>
</tr>
<tr>
<td>Get radius</td>
<td>1297 375</td>
</tr>
<tr>
<td>Get diameter</td>
<td>1298 375</td>
</tr>
<tr>
<td>Intersections</td>
<td>1299 375</td>
</tr>
<tr>
<td>Point argument</td>
<td>1300 376</td>
</tr>
<tr>
<td>Path argument</td>
<td>1302 376</td>
</tr>
<tr>
<td>Circle argument</td>
<td>1304 377</td>
</tr>
<tr>
<td>Reg_Polygon functions</td>
<td>1306 381</td>
</tr>
<tr>
<td>Enclosed circle</td>
<td>1307 381</td>
</tr>
<tr>
<td>Draw enclosed circle</td>
<td>1308 381</td>
</tr>
<tr>
<td>Normal version</td>
<td>1309 382</td>
</tr>
<tr>
<td>Picture argument first</td>
<td>1310 382</td>
</tr>
<tr>
<td>Surrounding circle</td>
<td>1311 382</td>
</tr>
<tr>
<td>Draw surrounding circle</td>
<td>1312 382</td>
</tr>
<tr>
<td>Normal version</td>
<td>1313 383</td>
</tr>
<tr>
<td>Picture argument first</td>
<td>1314 383</td>
</tr>
<tr>
<td>Putting Circle together</td>
<td>1315 383</td>
</tr>
<tr>
<td>Patterns (patterns.web)</td>
<td>1317 384</td>
</tr>
<tr>
<td>Include files</td>
<td>1318 384</td>
</tr>
<tr>
<td>Topic</td>
<td>Page 1</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Plane tessellations</td>
<td>1319</td>
</tr>
<tr>
<td>Hexagonal tessellation</td>
<td>1320</td>
</tr>
<tr>
<td>patterns</td>
<td>1322</td>
</tr>
<tr>
<td>Epicycloid pattern 1</td>
<td>1323</td>
</tr>
<tr>
<td>Epicycloid pattern 2</td>
<td>1325</td>
</tr>
<tr>
<td>Epicycloid pattern 3</td>
<td>1326</td>
</tr>
<tr>
<td>Putting patterns together</td>
<td>1328</td>
</tr>
<tr>
<td>Solid (solids.web)</td>
<td>1331</td>
</tr>
<tr>
<td>Include files</td>
<td>1332</td>
</tr>
<tr>
<td>Solid class definition</td>
<td>1333</td>
</tr>
<tr>
<td>Define static const Solid data members</td>
<td>1334</td>
</tr>
<tr>
<td>Constructors</td>
<td>1335</td>
</tr>
<tr>
<td>Default constructor</td>
<td>1336</td>
</tr>
<tr>
<td>Copy constructor</td>
<td>1338</td>
</tr>
<tr>
<td>Pseudo-constructor for dynamic allocation</td>
<td>1340</td>
</tr>
<tr>
<td>Pointer argument</td>
<td>1341</td>
</tr>
<tr>
<td>Reference argument</td>
<td>1342</td>
</tr>
<tr>
<td>Destructor</td>
<td>1343</td>
</tr>
<tr>
<td>Assignment</td>
<td>1345</td>
</tr>
<tr>
<td>Copying</td>
<td>1348</td>
</tr>
<tr>
<td>Set on free store</td>
<td>1350</td>
</tr>
<tr>
<td>Returning elements and information</td>
<td>1352</td>
</tr>
<tr>
<td>Get center</td>
<td>1353</td>
</tr>
<tr>
<td>Getting Shapes</td>
<td>1355</td>
</tr>
<tr>
<td>Get Shape pointer</td>
<td>1356</td>
</tr>
<tr>
<td>Get Circle pointer</td>
<td>1358</td>
</tr>
<tr>
<td>Get Ellipse pointer</td>
<td>1360</td>
</tr>
<tr>
<td>Get Path pointer</td>
<td>1362</td>
</tr>
<tr>
<td>Get Rectangle pointer</td>
<td>1364</td>
</tr>
<tr>
<td>Get Reg_Polygon pointer</td>
<td>1366</td>
</tr>
<tr>
<td>Getting Shape centers</td>
<td>1368</td>
</tr>
<tr>
<td>Get Shape center</td>
<td>1369</td>
</tr>
<tr>
<td>Get Circle center</td>
<td>1371</td>
</tr>
<tr>
<td>Get Ellipse center</td>
<td>1373</td>
</tr>
<tr>
<td>Get Rectangle center</td>
<td>1375</td>
</tr>
<tr>
<td>Get Reg_Polygon center</td>
<td>1377</td>
</tr>
<tr>
<td>Is on free store</td>
<td>1379</td>
</tr>
<tr>
<td>Show</td>
<td>1381</td>
</tr>
<tr>
<td>Clear</td>
<td>1383</td>
</tr>
<tr>
<td>Transformations</td>
<td>1385</td>
</tr>
<tr>
<td>Multiplying by a Transform</td>
<td>1386</td>
</tr>
<tr>
<td>Applying a transformation</td>
<td>1388</td>
</tr>
<tr>
<td>Scale</td>
<td>1390</td>
</tr>
<tr>
<td>Shear</td>
<td>1392</td>
</tr>
<tr>
<td>Shift</td>
<td>1394</td>
</tr>
<tr>
<td>real arguments</td>
<td>1395</td>
</tr>
<tr>
<td>Point argument</td>
<td>1397</td>
</tr>
<tr>
<td>Rotation around the main axes</td>
<td>1399</td>
</tr>
<tr>
<td>Rotation around an arbitrary axis</td>
<td>1401</td>
</tr>
<tr>
<td>Outputting</td>
<td>1403</td>
</tr>
<tr>
<td>Extract</td>
<td>1404</td>
</tr>
<tr>
<td>Set extremes</td>
<td>1406</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get extremes</td>
<td>1408</td>
</tr>
<tr>
<td>Get minimum z</td>
<td>1409</td>
</tr>
<tr>
<td>Get maximum z</td>
<td>1411</td>
</tr>
<tr>
<td>Get mean z</td>
<td>1413</td>
</tr>
<tr>
<td>Suppress output</td>
<td>1415</td>
</tr>
<tr>
<td>Unsuppress output</td>
<td>1417</td>
</tr>
<tr>
<td>Output</td>
<td>1419</td>
</tr>
<tr>
<td>Drawing and filling</td>
<td>1421</td>
</tr>
<tr>
<td>Draw</td>
<td>1423</td>
</tr>
<tr>
<td>Fill</td>
<td>1426</td>
</tr>
<tr>
<td>Filldraw</td>
<td>1429</td>
</tr>
<tr>
<td>Undraw</td>
<td>1432</td>
</tr>
<tr>
<td>Unfill</td>
<td>1435</td>
</tr>
<tr>
<td>Unfilldraw</td>
<td>1438</td>
</tr>
<tr>
<td>Putting Solid together</td>
<td>1440</td>
</tr>
<tr>
<td>Solid_Faced (solidface.web)</td>
<td>1443</td>
</tr>
<tr>
<td>Include files</td>
<td>1444</td>
</tr>
<tr>
<td>Solid_Faced class definition</td>
<td>1445</td>
</tr>
<tr>
<td>Putting Solid_Faced together</td>
<td>1448</td>
</tr>
<tr>
<td>Cuboid (cuboid.web)</td>
<td>1451</td>
</tr>
<tr>
<td>Include files</td>
<td>1452</td>
</tr>
<tr>
<td>Cuboid class definition</td>
<td>1453</td>
</tr>
<tr>
<td>Constructors and setting functions</td>
<td>1454</td>
</tr>
<tr>
<td>Default constructor</td>
<td>1455</td>
</tr>
<tr>
<td>Copy constructor</td>
<td>1457</td>
</tr>
<tr>
<td>Center, height, width, depth, and angles</td>
<td>1459</td>
</tr>
<tr>
<td>Pseudo-constructor for dynamic allocation</td>
<td>1461</td>
</tr>
<tr>
<td>Pointer argument</td>
<td>1462</td>
</tr>
<tr>
<td>Reference argument</td>
<td>1463</td>
</tr>
<tr>
<td>Destructor</td>
<td>1464</td>
</tr>
<tr>
<td>Assignment</td>
<td>1466</td>
</tr>
<tr>
<td>Putting Cuboid together</td>
<td>1468</td>
</tr>
<tr>
<td>Polyhedra (polyhedra.web)</td>
<td>1470</td>
</tr>
<tr>
<td>Include files</td>
<td>1471</td>
</tr>
<tr>
<td>Polyhedron class definition</td>
<td>1472</td>
</tr>
<tr>
<td>Intersection</td>
<td>1473</td>
</tr>
<tr>
<td>Regular Platonic Polyhedra</td>
<td>1475</td>
</tr>
<tr>
<td>Tetrahedron</td>
<td>1476</td>
</tr>
<tr>
<td>Tetrahedron class definition</td>
<td>1477</td>
</tr>
<tr>
<td>Define static const Tetrahedron data members</td>
<td>1478</td>
</tr>
<tr>
<td>Constructors and setting functions</td>
<td>1479</td>
</tr>
<tr>
<td>Default constructor</td>
<td>1480</td>
</tr>
<tr>
<td>Center, diameter of triangle, and angles</td>
<td>1482</td>
</tr>
<tr>
<td>Constructor</td>
<td>1483</td>
</tr>
<tr>
<td>Setting function</td>
<td>1487</td>
</tr>
<tr>
<td>Get net</td>
<td>1489</td>
</tr>
<tr>
<td>Draw net</td>
<td>1491</td>
</tr>
<tr>
<td>Dodecahedron</td>
<td>1493</td>
</tr>
<tr>
<td>Dodecahedron class definition</td>
<td>1494</td>
</tr>
<tr>
<td>Define static const Dodecahedron data members</td>
<td>1495</td>
</tr>
<tr>
<td>Constructors and setting functions</td>
<td>1496</td>
</tr>
</tbody>
</table>

3DLDF-1.1.5.1
TABLE OF CONTENTS

| Default constructor | 1497 | 450 |
| Center, diameter of pentagon, and angles | 1499 | 450 |
| Constructor | 1500 | 450 |
| Get net | 1503 | 453 |
| Draw net | 1505 | 454 |
| Icosahedron | 1507 | 456 |
| Icosahedron class definition | 1508 | 456 |
| Define static const Icosahedron data members | 1509 | 456 |
| Constructors and setting functions | 1510 | 456 |
| Default constructor | 1511 | 456 |
| Center, diameter of triangle, and angles | 1513 | 457 |
| Constructor | 1514 | 457 |
| Get net | 1516 | 459 |
| Draw net | 1518 | 461 |
| Semi-Regular Archimedean Polyhedra | 1520 | 463 |
| Truncated Octahedron | 1521 | 463 |
| Trunc_Octahedron class definition | 1522 | 463 |
| Define static const Trunc_Octahedron data members | 1523 | 463 |
| Constructors and setting functions | 1524 | 463 |
| Default constructor | 1525 | 463 |
| Center, diameter of hexagon, and angles | 1527 | 464 |
| Constructor | 1528 | 464 |
| Get net | 1530 | 466 |
| Putting polyhedra together | 1532 | 468 |
| Parsing (parser.web) | 1536 | 469 |
| Parse | 1538 | 469 |
| Putting the parser together | 1540 | 470 |
| Main (main.web) | 1543 | 471 |
| Include files | 1544 | 471 |
| Get input | 1545 | 472 |
| Actions in main | 1546 | 472 |
| Process command line options | 1548 | 473 |
| Print version, copyright, and license information | 1552 | 476 |
| Main itself | 1555 | 477 |
| Putting Main together | 1558 | 478 |
| Appendices | 1559 | 478 |
| References | 1560 | 478 |
| GNU Free Documentation License | 1561 | 479 |
| GNU General Public License | 1562 | 484 |
| Index | 1563 | 493 |