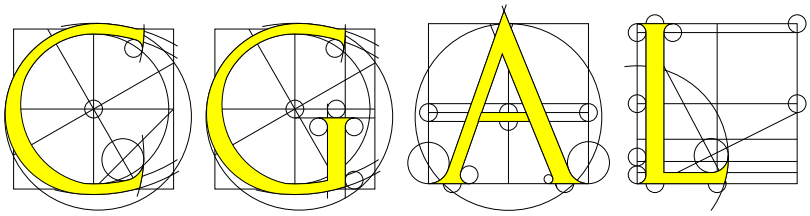

Installation

Release CGAL-3.2

22 May 2006

Contents

Chapter 1

Installation Guide

Michael Hoffmann, Dmitrii Pasechnik, and Wieger Wesselink

Contents

1.1 Introduction

CGAL stands for *Computational Geometry Algorithms Library*. It is a software library written in C++, whose development started in an ESPRIT LTR project. The goal of CGAL is to make the large body of geometric algorithms developed in the field of computational geometry available for industrial application.

This document describes how to install CGAL on Unix-like systems. If you install CGAL on Windows but not using cygwin, you should locate and load the Windows installer, read the file INSTALL.win32, and skip this manual as it doesn't apply for you.

Besides that, you will find some information about the makefile structure of CGAL and the support for using CGAL together with other software libraries, such as the BOOST libraries[1], the GNU Multiple Precision library GMP[2], the MPFR library[3] for multiple-precision floating-point computations with exact rounding, the CORE library[4] for robust numerical and geometric computation, LEDA, the Library of Efficient Datatypes and Algorithms[5], TAUCS, a library of sparse linear solvers[6], or Trolltech's[7] QT toolkit.

1.2 Prerequisites

In order to build the CGAL libraries you need a C++ compiler. Most recent compilers on Unix platforms and MS Windows are supported, provided that they reasonably conform to the ISO 14882 standard for C++.

CGAL-3.2 supports the following compilers/operating systems:

[1] http://www.boost.org/
[2] http://www.swox.com/gmp/
[3] http://www.mpfr.org/
[4] http://www.cs.nyu.edu/exact/core_pages/
[5] http://www.algorithmic-solutions.com/
[6] http://www.tau.ac.il/~stoledo/taucs/
[7] http://www.trolltech.com

compiler	operating system
GNU g++ 3.3.3, 3.4, 4.0, 4.1 [9]	IRIX 6.5 / Solaris 2.6+ / Linux 2.x / MacOS X / MS Windows 95/98/2000/XP/NT4[10]
MS Visual C++ 7.1 (.NET) [12]	MS Windows 95/98/2000/XP/NT4[10]
INTEL C++ 8.1 [13]	MS Windows 95/98/2000/XP/NT4[10]

Note that neither prerelease versions nor repository snapshots of GCC are supported.

Moreover, CGAL requires a working installation of the BOOST libraries (at least, the header files) Version 1.32 or later. In case the BOOST libraries are not installed on your system already, you can obtain them from http://www.boost.org/. For Windows you can download an installer from http://www.boost-consulting.com/download.html.

Having GMP version 4.1.4 or later and MPFR version 2.0.1 or later installed is highly recommended. These libraries can be obtained from http://www.swox.com/gmp/ and http://www.mpfr.org/, respectively, if they should not be present on your system. Note that MPFR used to be included with GMP up to GMP version 4.1.X, but it is not included anymore starting from GMP 4.2. Within CGAL, support for GMP requires also MPFR and vice versa.

If you are going to install CGAL using Cygwin[13], please read Section 1.12 first.

1.3 Getting CGAL

The CGAL library can be downloaded from the CGAL homepage:

```
http://www.cgal.org
```

and go to the 'Download' section. Just follow the instructions on this page to obtain your copy of the library.

After you have downloaded the file containing the CGAL library, you have to decompress it. Use the commands

```
gunzip <filename>.tar.gz
tar xvf <filename>.tar
```

Alternatively, your browser might be able to invoke the right decompression program by itself.

In both cases the directory CGAL-3.2 will be created. This directory contains the following subdirectories:

[9] http://gcc.gnu.org/
[10] with Cygwin (http://www.cygwin.com)
[12] http://msdn.microsoft.com/visualc/
[13] http://developer.intel.com/software/products/compilers/
[13] http://www.cygwin.com

directory	contents
auxiliary	packages that can optionally be used with CGAL
config	configuration files for install script
demo	demo programs (most of them need QT, geomview or other third-party products)
doc_html	documentation (HTML)
doc_pdf	documentation (PDF)
doc_ps	documentation (Postscript)
examples	example programs
include	header files
lib	(shared) object libraries
make	files with platform dependent makefile settings
scripts	some useful scripts (e.g. for creating makefiles)
src	source files

The directories include/CORE and src/Core contain a distribution of the CORE library[14] version 1.7 for robust numerical and geometric computation. CORE is not part of CGAL and has its own license.

1.3.1 Visualization

The programs below the demo directory provide visual output. Most of these use *CGAL::Qt_widget*, a widget and some helper classes that allow to interact with two dimensional CGAL objects in QT 3 based applications. There is no support for QT 4 in CGAL yet.

If you have LEDA installed, you might want to use *CGAL::Window_stream* as an interface between two dimensional CGAL objects and a *leda_window*. To be able to use the *Window_stream*, you simply have to compile CGAL with LEDA support.

Some demo programs for 3D structures require the geomview program for visualization. This is available from http://www.geomview.org (note that it does not run on MS Windows).

1.4 Installing CGAL

The directory CGAL-3.2 contains a Bourne shell script called install_cgal . The script can be run in two modes: a menu-driven interactive mode and a non-interactive mode. Normally you should use the interactive mode, but in case you run into problems with it or do not like it for some reason, you can still use the non-interactive mode.

We first describe a sample installation in Section 1.5. This section provides an overview on the interactive installation. If you want more detailed information about specific menus and their options, take a look at Section 1.6. Finally, for the non-interactive mode refer to Section 1.7.

[14] http://www.cs.nyu.edu/exact/core_pages/

1.5 A Sample Installation

In this section we sketch an example installation on a Linux machine with the GNU g++ 3.4.5 compiler. For a complete description of the different menus and their options refer to Section 1.6.

1.5.1 Start the Script

Go to the CGAL-3.2 directory and enter the command

```
./install_cgal -i
```

The script first parses support options for various third party libraries that can be use together with CGAL. Then you get a message indicating the CGAL version you are going to install and that you are running the interactive mode. It takes some time while the script locates a number of utility programs. You will not get informed about this[15], but you see some dots written to the screen indicating progress.

```
Parsing support specfiles for 3rd party libraries:
BOOST, BOOSTPROGRAMOPTIONS, X11, GMP, GMPXX, MPFR, CORE, CGALCORE, ZLIB, ZLIBMS, LEDA, LEDAWIN, LEDAMS, LEDAWINMS, QT3MT, QT3ST, QT3MSMT, QT3MSST, TAU

------------------------------------------------------------
  This is the install script for CGAL 3.2
------------------------------------------------------------

starting interactive mode - one moment, please
.
------------------------------------------------------------
Settings from Command Line/Environment:
------------------------------------------------------------

  Choosing compiler GNU 3.4.5.
```

The script searches for C++ compilers that are installed on your system and accessible through your PATH environment variable. If more than one compiler (that is supported by CGAL) is found on your system, you are presented a list of compilers to choose from. In this example, only one compiler, GNU g++ 3.4.5, was found on the system and the script selects this compiler without any user interaction. In general, the selection of which compiler to use can be made from the compiler menu (cf. Section 1.6.2).

The script interprets the environment variables CXX, CXXFLAGS, and LDFLAGS and accordingly sets the compiler, compiler flags, and linker flags, respectively. Therefore, another way to select the compiler is to set the environment variable CXX to the compiler to be selected.

A menu similar to the following will appear on your screen.

```
**************************************************************
**             CGAL 3.2 Installation Main Menu              **
**             ==============================               **
**                                                          **
**   OS:          i686_Linux-2.6                            **
**   Compiler:    GNU 3.4.5                                 **
**   Support for: no other library.                         **
**                                                          **
**   Compiler is supported by CGAL.                         **
**   The setup has not been tested.                         **
**                                                          **
**   There are no libs for this os/compiler.                **
**                                                          **
**   <C>  Compiler Menu                                     **
```

[15] If you are that curious what happens exactly, have a look at the file CGAL-3.2/install.log.

```
**   <S>  Support Menu                                          **
**   <T>  Test (and save) setup                                 **
**   <A>  Run all setup tests (no cache)                        **
**                                                              **
**   <B>  Build CGAL Libraries                                  **
**   <I>  Install CGAL Libraries                                **
**                                                              **
**   <Q>  Back to OS                                            **
**                                                              **
**   Your Choice:                                               **
**                                                              **
**************************************************************
```

The first lines below the headline contain some kind of status report: current OS and compiler, and which third-party software libraries are supported. Moreover you can see that the current setup has not yet been tested, and that there do not exist CGAL libraries for this OS/compiler combination in the CGAL lib directory by now.

1.5.2 Test the Setup

As a first step, you should test the current setup by typing "`t`". This starts a number of tests to check whether your compiler supports certain language constructs or has specific bugs. The script also tries to locate other libraries that are installed on your system and can be used together with CGAL. There is quite a number of these tests, so this step may take a while. For each test you should get a message what in particular is tested at the moment and what the result is. More details about why a certain test failed can be found in the installation logfile `install.log`.

```
**************************************************************
**   The following lines show results of configuration tests. **
**   Some of the tests might fail, since many compilers are   **
**   still not completely ISO standard compliant.             **
**   Since we worked around the arising problems,             **
**               *** CGAL will work fine ***                  **
**   regardless of the outcome of these tests.                **
**************************************************************
Checking for standard header files
All headers at once : ok.
Testing for CCTYPE_MACRO_BUG ... ok.

<many lines omitted>

Saving current setup ... done.
```

After all these tests are completed, the current settings are saved into a file that resides in the directory `CGAL-3.2/config/install`. Thus, if you run the install script a second time for this OS/compiler, you will not have to go through the whole config-/test cycle again, but the configuration will be retrieved from the corresponding config file instead. In case you want to re-run all tests, pick the option `<A> Run all setup tests (no cache)` from the main menu.

Looking at the menu header field "`Support for`", you can see whether the setup has been tested successfully and which other libraries have been found on your system. The install script automatically enables CGAL support for any library that has been found.

```
**************************************************************
**              CGAL 3.2 Installation Main Menu               **
**              ===============================               **
**                                                            **
**   OS:          i686_Linux-2.6                              **
**   Compiler:    GNU 3.4.5                                   **
**   Support for: BOOST, BOOST_PROGRAM_OPTIONS, X11, GMP,     **
**                GMPXX, MPFR, CORE, ZLIB, QT, and TAUCS.     **
**                                                            **
**   Compiler is supported by CGAL.                           **
```

```
**   The setup has been tested ok.                          **

<some lines omitted>

*************************************************************
```

In case you are not happy with the set of libraries for which support is configured, you can change the settings for each library individually from the support menu (cf. Section 1.6.3). In case you wonder why a certain library was not automatically configured correctly, you will find more details in the installation logfile `CGAL-3.2/install.log`.

1.5.3 Build the CGAL Libraries

We are now ready to build the CGAL libraries. From the main menu just type "`b`" to start compilation. Building consists of several steps:

1. write the include makefile,
2. compile the static libraries,
3. *(on some systems)* compile the shared libraries,
4. *(if* QT *support is configured)* compile the CGALQt library, *and*
5. *(if* CGALCORE *support is configured)* compile the CORE library.

The include makefile encapsulates the OS– and compiler-specific settings and should be included (hence the name) in all makefiles that compile CGAL applications. If everything went ok, the output should look as follows. (Otherwise, you should have a look at the error messages from compiler or linker.)

```
*************************************************************
**                                                         **
**                    Compiling CGAL 3.2                   **
**                    ==================                   **
**                                                         **
*************************************************************

OS:                           i686_Linux-2.6
COMPILER:                     GNU 3.4.5
BOOST: supported
BOOST_PROGRAM_OPTIONS: supported
X11: supported
GMP: supported
GMPXX: supported
MPFR: supported
CORE: supported
ZLIB: supported
LEDA: not supported
LEDAWIN: not supported
QT: supported
TAUCS: supported

Generating Makefiles ... done.
Building CGAL_lib ... done.
Building CGAL_sharedlib ... done.
Building CGAL_Qt ... done.
Building CGAL_Core ... done.

*************************************************************
**            Please press <ENTER> to continue.            **
*************************************************************
```

That's all, it's done. Press "`<ENTER>`" to return to the main menu. CGAL is now ready to use.

1.5.4 Install the CGAL Libraries

It is perfectly fine to install and use CGAL from its build-tree and, in fact, this has been the default setup for many years. But sometimes it is more convenient to install CGAL in a directory outside its build-tree such as, for instance, `/usr/local/cgal`. In case you prefer such a setup, go to the install menu by choosing the option "`i`" from the main menu.

```
*************************************************************
**             CGAL 3.2 Installation Install Menu          **
**             ==================================          **
**                                                         **
**   OS:          i686_Linux-2.6                           **
**   Compiler:    GNU 3.4.5                                **
**   Support for: BOOST, BOOST_PROGRAM_OPTIONS, X11, GMP,  **
**                GMPXX, MPFR, CORE, ZLIB, QT, and TAUCS.  **
**                                                         **
**   Compiler is supported by CGAL.                        **
**   The setup has been tested ok.                         **
**                                                         **
**   Libs built:          19 Apr 17:20                     **
**                                                         **
**   <D>  Change Installation root directory               **
**       /usr/local/CGAL-3.2                               **
**   <I>  Install CGAL                                     **
**                                                         **
**   <Q>  Back to Main Menu                                **
**                                                         **
**   Your Choice:                                          **
**                                                         **
*************************************************************
```

You can adjust the installation root directory using option "`d`"; it defaults to the current build root. Then start the installation by selecting "`i`".

```
*************************************************************
**                                                         **
**                   Installing CGAL 3.2                   **
**                   ===================                   **
**                                                         **
*************************************************************

OS:                          i686_Linux-2.6
<some lines omitted>

*************************************************************
**                                                         **
**   Target Directory: /usr/local/CGAL-3.2.                **
**   Do you want to install CGAL into this directory (y/N)?  **
**                                                         **
*************************************************************
```

If you confirm the settings and enter "`y`", CGAL is installed below the given root directory. It can there be used independently from its build-tree, that is, you may even delete the build-tree.

```
Install CGAL headers ... done.
Install CGAL libraries ... done.
Install CGAL programs ... done.
Install CGAL Makefile ... done.

*************************************************************
**             Please press <ENTER> to continue.           **
*************************************************************
```

Press "`<ENTER>`" and then select "`q`" to return to the main menu.

1.5.5 Further Steps

You may now proceed by installing CGAL for a different compiler (go to the compiler menu and choose "c" to get a list of supported compilers detected on your system).

Another option is to simply quit the install script by typing "q". When leaving the script, you get a list of successful builds during the session. Furthermore, the script prints the setting of CGAL_MAKEFILE for the last active configuration. Remember to set this environment variable before compiling CGAL applications. On Bourne shell derivatives, you would type in our example

```
export CGAL_MAKEFILE=CGAL-3.2/make/makefile_sparc_SunOS-5.6_g++-3.4.3
```

while for csh descendants the syntax is

```
setenv CGAL_MAKEFILE CGAL-3.2/make/makefile_sparc_SunOS-5.6_g++-3.4.3
```

In Section 1.10 you can find more information on the CGAL makefile structure, and how to set CGAL_MAKEFILE when using CGAL on several platforms.

1.6 The Interactive Mode

To run the install script in the interactive mode, go to the CGAL-3.2 directory and enter the command

```
./install_cgal -i
```

After initialization during which certain utility programs are located and your system is searched for compilers supported by CGAL, you get into the CGAL installation *main menu* (see page 5 for a picture).

From the main menu you can reach a number of different sub-menus, of which the most important maybe is the *compiler menu*. This is where you can choose the compiler you want to work with and set custom compiler or linker options. The compiler menu is described in Section 1.6.2.

If you want to use other libraries such as BOOST, TAUCS, GMP, MPFR, CORE, LEDA, or QT with CGAL, you can setup support for these libraries from the support menu that is described in Section 1.6.3.

At some point, you most probably want to build the CGAL libraries by typing b. However, it is recommended to run the *setup test* – which is available in all menus as option t – before. The setup test includes tests for certain compiler features or bugs as well as tests for the presence of other libraries that can be used in conjunction with CGAL. The install script tries to find and enable support for as many other libraries as possible.

As a default, CGAL is built and installed within its build-tree. In case you want to install CGAL in a directory separate from its build-tree, the *install menu* is the place to go to.

The install script keeps track of the tests passed and only tests again, if you change the setup in a way that may affect the test result. If you want to redo *all* tests, you have to choose option "a" from the main menu.

1.6.1 Files Created during Installation

The install script stores all relevant settings for an OS/compiler combination in a file

`CGAL-3.2/config/install/<CGAL-OS-description>`

where <*CGAL-OS-description*> identifies your OS/compiler combination in a way specified in Section 1.9. This saves you the work of configuring everything again, if you upgrade CGAL or another package that makes recompiling the CGAL libraries necessary.

Besides the config files, the install script uses several temporary files during interactive installation. Most of them are removed after use, but some are not, since it might be helpful to keep some information about the last run. You can keep or delete them as you like, as they are not needed anymore once the script terminated. A list of these files (all are plain ASCII and reside in `CGAL-3.2`) follows.

filename	**content**
`install.log`	detailed overall protocol
`install.completed`	list of systems for which CGAL libraries have been built
`compile.log`	output of the last compiler call

1.6.2 The Compiler Menu

Here is the place to set compiler specific options, such as the compiler to use (if more than one has been detected), custom compiler or linker flags, or to decide whether or not to build shared libraries.

Compiler Menu

<**C**> Choose the compiler to be used from the list of detected compilers. You can also register other compilers, if they have not been detected automatically.

<**F**> Set custom compiler flags. These are the first flags given to the compiler in every call. Under normal circumstances there should be no need to set any such flag.

<**L**> Set custom linker flags. These are the first flags given to the linker in every call. Under normal circumstances there should be no need to set any such flag.

<**S**> Toggle shared libraries building. By default, shared libraries are built (e.g. `libCGAL.so`), but it is possible to only build static libraries using this option.

<**O**> Set custom ostype tag. Set a tag that is mangled into the CGAL-OS-description. This is useful to distinguish CGAL installations with different settings for the same OS-compiler combination.

1.6.3 The Support Menu

This menu provides the starting point to setup support for third-party software libraries such as BOOST, TAUCS, GMP, MPFR, CORE, LEDA, or QT.

First, it provides an option "Auto-find all libraries" to find and enable support for as many of those third-party software libraries as possible. Note that this is also done automatically during the first setup test for a platform.

Then there is a sub-menu for each third-party library that can be used in conjunction with CGAL. Each of these sub-menus contains an autofind option that is similar to the global autofind in the support menu but restricted to the particular library under consideration.

In case a library is installed in a non-standard location, you may have to provide a path to its header files and/or libraries. Therefore, the menu for each library L offers an option to set an include directory L_INCL_DIR and a lib directory L_LIB_DIR for L. For instance, if on your system GMP is installed in /opt/sw/gmp/, you would set GMP_INCL_DIR to /opt/sw/gmp/include and GMP_LIB_DIR to /opt/sw/gmp/lib. Note that on Cygwin you have to provide Posix-style paths, e.g., /cygdrive/c/gmp instead of C:\gmp.

If you have trouble to setup support for one of the libraries, having a look at the install.log file may give you a hint.

More details about the different third-party libraries that are supported by CGAL can be found in sections 1.7.2–1.7.12.

1.7 The Non-Interactive Mode

To run the install script in the non-interactive mode, you have to specify the compiler to use by setting the environment variable CXX to the C++ compiler executable. You can either specify a full path, e.g. CXX=/usr/local/bin/g++, or just the basename, e.g. CXX=g++, which means the script searches your PATH for the compiler location. If your compiler call contains whitespaces then it has to be quoted, e.g., CXX="CC -n32". The options given this way become part of your CGAL-OS description (see Section 1.9) which is useful, e.g., to distinguish between different compilers using the same frontend (such as SGI Mips(Pro) CC on IRIX6).

Go to the CGAL-3.2 directory and enter the command

```
CXX=<compiler> ./install_cgal -ni
```

where <compiler> is the C++ compiler executable, as described above.

There are a number of additional command line options to customize your CGAL setup which are discussed below. You should read the corresponding paragraphs before you continue, especially if you want to enable support for a third party library that is installed in a non-standard location.

The script tries to enable support for as many third party libraries as possible, most of them should be found automatically if installed in a standard location. If you do not want to enable support for third party libraries automatically, use the command line switch --without-autofind.

Once you started the script, it should give you a message indicating the CGAL version you are going to install and that you are running the non-interactive mode. Then it proceeds by locating some utility programs, determining your OS and compiler version and displaying the settings you gave via command line. Your compiler is also checked for a number of bugs resp. support of certain language features; a message ok always indicates that your compiler works as it should, that is, a feature is supported or a bug is *not* present. On the other hand, no or unfortunately indicate a lack of support or the presence of a bug.

Then the current setup is summarized, system specific directories for makefiles and libraries are created (if they did not exist before) and a new include makefile is written into the makefile directory. If there already exists a makefile for the current OS/compiler combination, it is backed up.

Finally, the CGAL libraries are built and – if a separate installation root was specified – installed into the given installation directory.

1.7.1 Specify an Install Directory

As a default, CGAL is installed within its build-tree. The installation root directory can be changed using the command line option "`--prefix <dir>`". For example, to install CGAL below `/usr/local/cgal`, you would type

```
--prefix /usr/local/cgal
```

1.7.2 Setup Support for Boost

The BOOST headers are required by CGAL, you can get them from `http://www.boost.org/`. If installed in a standard location, BOOST should be found automatically. Otherwise, you have to specify where BOOST is installed on your system using the command line options "`--BOOST_INCL_DIR`" and/or "`--BOOST_LIB_DIR`". For instance, if you have installed BOOST in `/opt/boost/`, type

```
--BOOST_INCL_DIR /opt/boost/include --BOOST_LIB_DIR /opt/boost/lib
```

Even in case you disabled the automatic support for third party libraries, boost support is still enabled because it is required.

1.7.3 Setup Support for Boostprogramoptions

The BOOST library program_options is used by a few demo programs to parse command line options. You can get it from `http://www.boost.org/`. If installed in a standard location, program_options should be found automatically. Otherwise, you have to specify where program_options is installed on your system using the command line options "`--BOOSTPROGRAMOPTIONS_INCL_DIR`" and/or "`--BOOSTPROGRAMOPTIONS_LIB_DIR`".

In case you disabled the automatic support for third party libraries, support for program_options can be enabled with the command line switch "`--with-BOOSTPROGRAMOPTIONS`".

1.7.4 Setup Support for TAUCS

TAUCS is library of sparse linear solvers. In CGAL it is used to speedup the computations within the *Surface_mesh_parameterization* package (Planar Parameterization of Triangulated Surface Meshes) only.

You can download the official release from `http://www.tau.ac.il/~stoledo/taucs/`or download a version customized by CGAL team from the Download section of `http://www.cgal.org`.

See TAUCS documentation to compile it. In a nutshell, on Unix machines:

```
./configure [prefix=PREFIX]
make
make install  # only available in the version customized by CGAL team
```

If installed in a standard location, TAUCS should be found automatically. Otherwise, you have to specify where TAUCS is located on your system using the command line options "--TAUCS_INCL_DIR" and "--TAUCS_LIB_DIR". For instance, if you have uncompressed and compiled the official release of TAUCS (which has no "make install" option) in /opt/TAUCS/, type

```
--TAUCS_INCL_DIR "/opt/TAUCS/build/linux:/opt/TAUCS/src" --TAUCS_LIB_DIR "/opt/TAUCS/external/lib/linux
```

(replace "linux" by your platform).

In case you disabled the automatic support for third party libraries, support for TAUCS can be enabled with the command line switches "--with-TAUCSSOLARISCC", "--with-TAUCSWINMKL", "--with-TAUCSWINATLAS", "--with-TAUCSGCCATLAS", "--with-TAUCSGCCATLAS2", "--with-TAUCSINTELCCATLAS", "--with-TAUCSINTELCCMUMPS", "--with-TAUCSDARWIN", "--with-TAUCSGCCMKL", "--with-TAUCSIRIXCC32", and "--with-TAUCSIRIXCC64" (depending of your platform and BLAS library).

1.7.5 Setup Support for X11

As visualization in CGAL is done using QT mostly, the direct support for X11 is used in conjunction with the LEDA window library mostly. You can get X11 from http://www.x.org/.

If installed in a standard location, X11 should be found automatically. Otherwise, you have to specify where X11 is installed on your system using the command line options "--X11_INCL_DIR" and/or "--X11_LIB_DIR". For instance, if you have installed X11 in /opt/X11R8/, type

```
--X11_INCL_DIR /opt/X11R8/include --X11_LIB_DIR /opt/X11R8/lib
```

In case you disabled the automatic support for third party libraries, support for X11 can be enabled with the command line switch "--with-X11".

1.7.6 Setup Support for GMP

GMP is a multi-precision number type library that is available from http://www.swox.com/gmp/. In CGAL it is used in many example and demo programs to speedup the computation. It is highly recommended to configure CGAL with support for GMP. Note that GMP support also requires MPFR support and vice versa.

If installed in a standard location, GMP should be found automatically. Otherwise, you have to specify where GMP is installed on your system using the command line options "--GMP_INCL_DIR" and/or "--GMP_LIB_DIR". For instance, if you have installed GMP in /opt/gmp, type

```
--GMP_INCL_DIR /opt/gmp/include --GMP_LIB_DIR /opt/gmp/lib
```

In case you disabled the automatic support for third party libraries, support for GMP can be enabled with the command line switch "--with-GMP".

1.7.7 Setup Support for GMPXX

GMPXX denotes a C++ wrapper for GMP, that is an optional feature of GMP and, therefore, not present in all installations.

If installed in a standard location, GMPXX should be found automatically. As GMPXX depends on GMP, there is usually no need to specify include or lib directories for GMPXX once GMP support is configured correctly.

In case you disabled the automatic support for third party libraries, support for GMPXX can be enabled with the command line switch "`--with-GMPXX`".

1.7.8 Setup Support for MPFR

MPFR is a library for multi-precision floating-point computations with exact rounding that is available from `http://www.mpfr.org/`. In CGAL it is used in many example and demo programs to speedup the computation. It is highly recommended to configure CGAL with support for MPFR. Note that MPFR support also requires GMP support and vice versa.

If installed in a standard location, MPFR should be found automatically. As MPFR depends on GMP, there is no need to specify include or lib directories for MPFR when GMP version prior to 4.2 is used and configured correctly. Otherwise, you have to specify where MPFR is installed on your system using the command line options "`--MPFR_INCL_DIR`" and/or "`--MPFR_LIB_DIR`". For instance, if you have installed MPFR in `/opt/mpfr`, type

```
--MPFR_INCL_DIR /opt/mpfr/include --MPFR_LIB_DIR /opt/mpfr/lib
```

In case you disabled the automatic support for third party libraries, support for MPFR can be enabled with the command line switch "`--with-MPFR`".

1.7.9 Setup Support for CORE

CORE is a number type library for exact geometric computation that is available from `http://www.cs.nyu.edu/exact/core_pages/`. CORE version 1.7 is also shipped with CGAL 3.2. In CGAL it is used in many example and demo programs to speedup the computation. It is highly recommended to configure CGAL with support for CORE. Note that CORE support also requires GMP support.

If installed in a standard location, CORE should be found automatically. Otherwise, you have two options: Either use the version of CORE that is shipped with CGAL and, hence, found automatically, or specify where CORE is installed on your system using the command line options "`--CORE_INCL_DIR`" and/or "`--CORE_LIB_DIR`". For instance, if you have installed CORE in `/opt/core`, type

```
--CORE_INCL_DIR /opt/core/include --CORE_LIB_DIR /opt/core/lib
```

In case you disabled the automatic support for third party libraries, support for CORE can be enabled with the command line switch "`--with-CORE`". In order to enable support for the version of CORE that is shipped with CGAL, use the switch"`--with-CGALCORE`" instead.

1.7.10 Setup Support for ZLIB

ZLIB is a compression library that is available from `http://www.zlib.net/`. Within CGAL, it is used to compress images for the Surface Mesher only.

If installed in a standard location, ZLIB should be found automatically. Otherwise, you have to specify where ZLIB is installed on your system using the command line options "`--ZLIB_INCL_DIR`" and/or "`--ZLIB_LIB_DIR`". For instance, if you have installed ZLIB in `/opt/zlib`, type

```
--ZLIB_INCL_DIR /opt/zlib/include --ZLIB_LIB_DIR /opt/zlib/lib
```

In case you disabled the automatic support for third party libraries, support for ZLIB can be enabled with the command line switch "`--with-ZLIBMS`" (for Microsoft and Intel Compiler) or "`--with-ZLIB`" (all other compilers).

1.7.11 Setup Support for LEDA

LEDA is a library for efficient data types and algorithms that can be bought from `http://www.algorithmic-solutions.com/`. CGAL offers an interface to the LEDA window and geowin classes and support for LEDA number types. For more details, see Section 1.13.

If installed in a standard location, LEDA should be found automatically. Otherwise, you have to specify where LEDA is installed on your system using the command line options "`--LEDA_INCL_DIR`" and/or "`--LEDA_LIB_DIR`". For instance, if you have installed LEDA in `/opt/leda`, type

```
--LEDA_INCL_DIR /opt/leda/include --LEDA_LIB_DIR /opt/leda/lib
```

The LEDA window library is handled separately because it depends on X11. Obviously there is no need to specify include and lib directories for the LEDA window library once LEDA support is configured correctly.

In case you disabled the automatic support for third party libraries, support for LEDA can be enabled with the command line switch "`--with-LEDAMS`" (for Microsoft and Intel Compiler) or "`--with-LEDA`" (all other compilers). Support for the LEDA window library can be enabled with the command line switch "`--with-LEDAWINMS`" (for Microsoft and Intel Compiler) or "`--with-LEDAWIN`" (all other compilers).

1.7.12 Setup Support for Qt

QT is a GUI library that is available from `http://doc.trolltech.com/`. Most demo programs of CGAL for two-dimensional data structures and algorithms use QT for visualization. Note that CGAL supports QT 3 but not (yet) QT 4.

QT support requires the environment variable `QTDIR` to be set to the QT root directory. On most systems where QT is installed this should be set correctly, otherwise you have to set this variable manually. We assume the QT meta object compiler is available under `$QTDIR/bin/moc`.

Based on the setting of `QTDIR`, QT should be found automatically. Therefore, it should not be necessary to specify include or lib directories for QT on the command line.

There are two different ways to build QT, namely single-threaded or multi-threaded. The automatic support settings prefer the multi-threaded version where present.

In case you disabled the automatic support for third party libraries, support for the multi-threaded version of QT can be enabled with the command line switch "`--with-QT3MSMT`" (for Microsoft and Intel Compiler) or "`--with-QT3MT`" (all other compilers). Support for the single-threaded version of QT can be enabled with the command line switch "`--with-QT3MSST`" (for Microsoft and Intel Compiler) or "`--with-QT3ST`" (all other compilers).

1.7.13 Setting Custom Compiler/Linker Flags

You can supply custom compiler and linker flags using the options ("`--CUSTOM_CXXFLAGS` <*flags*>") and ("`--CUSTOM_LDFLAGS` <*flags*>"). These are the first flags given to the compiler/linker in every call.

Note: Do not forget to quote your options in case they contain spaces. Example:

```
--CUSTOM_CXXFLAGS "-I/my/include -O2"
```

1.7.14 Disable Shared Libraries

You can disable the building of shared libraries (e.g., `libCGAL.so`) using the option "`--disable-shared`". This way, only static libraries (object file archives) are built.

1.7.15 Other Options

Some less important features of the install script are summarized here.

First of all, you can get the version number of `install_cgal` using the option "`--version`". All other options are ignored in this case.

Second, there is an option "`-os` <*compiler*>" where <*compiler*> is your C++ compiler. This allows you to determine your CGAL-OS description (see Section 1.9). The compiler can either be given by an absolute path like

```
./install_cgal -os /usr/local/gcc-3.4.3/sun/bin/g++
```

or just by denoting its basename, as long as it is on your path:

```
./install_cgal -os CC
```

The option is intended for testing purposes and automatic detection of the correct include makefile (see also Section 1.10).

Finally, there exists an option "`--verbose`" that can be set in interactive mode as well as in non-interactive mode. When set you get a detailed summary of error messages occurring during *any* compiler test (determining STL version etc.). Normally you only get these messages, if a required test (such as the general STL test) fails, otherwise you are just informed, *if* it succeeded or not. This option is not recommended for general use, but it can be useful to check why a certain test fails that was expected to be passed. Most of the extra information it provides can also be extracted from the installation logfile.

1.8 Upgrade an Existing CGAL Installation

Due to a large number of changes in the configuration and installation process, we recommend to install CGAL from scratch, without re-using configuration files of possible previous CGAL installations.

1.9 Identify OS and Compiler

Since CGAL supports several different operating systems and compilers, this is also reflected in the structure of the CGAL directory tree. Each OS/compiler combination has its own lib directory under `CGAL-3.2/lib`) (and analogously its own include makefile in `CGAL-3.2/make`) named as determined by the following scheme.

<arch>_<os>-<os-version>_<comp>-<comp-version>

<**arch**> is the system architecture as defined by "`uname -p`" or "`uname -m`",

<**os**> is the operating system as defined by "`uname -s`",

<**os-version**> is the operating system version as defined by "`uname -r`",

<**comp**> is the basename of the compiler executable (if it contains spaces, these are replaced by "-") *and*

<**comp-version**> is the compiler's version number (which unfortunately can not be derived in a uniform manner, since it is quite compiler specific).

We call the resulting string CGAL-OS description.
Examples are `mips_IRIX-6.5_CC-7.2` or `sparc_SunOS-5.5_g++-3.4.3`.
You can use the install script to get your CGAL-OS description, see Section 1.7.15.

1.10 The CGAL Makefile Structure

There is a makefile in each directory below `CGAL-3.2/examples` and `CGAL-3.2/demo`. All these makefiles are generic: they can be used for more than one compiler. To achieve this, the first section of each makefile contains an include statement that looks as follows:

```
# CGAL_MAKEFILE = ENTER_YOUR_INCLUDE_MAKEFILE_HERE
include $(CGAL_MAKEFILE)
```

The file `CGAL_MAKEFILE` is an include file with platform dependent makefile settings. The abbreviation `<CGAL-OS description>` (see Section 1.9 for details) is used to identify the operating system and compiler for which the settings hold. For example, the file `makefile_mips_IRIX64-6.5_CC-n32-7.30` contains makefile settings for the IRIX 6.5 operating system and the SGI Mips(Pro) `CC` 7.3 compiler. These include files are automatically generated by the `install_cgal` script and they are all located in the `CGAL-3.2/make` directory.

If you want to compile an application or an object library with a different compiler, the only thing you need to do is to substitute another include makefile for the `CGAL_MAKEFILE` variable. A very convenient way to do this is to create an environment variable `CGAL_MAKEFILE`. In Bourne shell you type

```
export CGAL_MAKEFILE=<insert your CGAL makefile>
```

whereas in csh derivatives you use

```
setenv CGAL_MAKEFILE <insert your CGAL makefile>
```

For instance,

```
setenv CGAL_MAKEFILE /usr/CGAL-3.2/make/makefile_i686_Linux-2.6_g++-3.4.4
```

If you use CGAL with several different OS/compiler combination, a comfortable way to set `CGAL_MAKEFILE` is by using `install_cgal -os` (see Section 1.7.15). For example, if your compiler is `g++`, you would type

```
CGAL_MAKEFILE='<insert your CGAL-3.2 dir>/install_cgal -os g++'
```

in Bourne shell resp.

```
setenv CGAL_MAKEFILE '<insert your CGAL-3.2 dir>/install_cgal -os g++'
```

in csh derivatives.

Tip: Include the setting of `CGAL_MAKEFILE` into your shell startup script (e.g. `.(t)cshrc` for `(t)csh` or `.bashrc` for `bash`).

All makefiles contain sections with compiler and linker flags. You can add your own flags here. For example, you might want to add the flag `-DCGAL_NO_PRECONDITIONS` to turn off precondition checking. The flags `$(CGAL_CXXFLAGS)` and `$(CGAL_LDFLAGS)` should never be removed.

The default extension for CGAL source files is `.C`. The last section of the makefiles contains a suffix rule that tells the compiler how to create a `.o`-file from a `.C`-file. If you want to use the default rule that is defined by the make utility, you may want to remove this suffix rule. However, note that this may have consequences for the makefile variables `CGAL_CXX` and `CXXFLAGS`.

1.11 Compile a CGAL Application

There is a script for conveniently creating makefiles for CGAL applications, see Section 1.17.1.

The directories `CGAL-3.2/examples` and `CGAL-3.2/demo` contain many subdirectories with non-graphical and graphical example programs. In all these directories you will find a makefile that is ready to use.

You either need to substitute the `CGAL_MAKEFILE` variable in these makefiles (see Section 1.10), or set the environment variable, to point to the makefile in the `CGAL-3.2/make` directory. Then just type `make`.

Within CGAL, the file `<CGAL/basic.h>` manages all configuration problems. It is therefore **important that `<CGAL/basic.h>` is always included before any other file**. In most cases you do not have to do anything special for this, because many CGAL files (in particular, `<CGAL/Cartesian.h>` and `<CGAL/Homogeneous.h>`) already take care of including `<CGAL/basic.h>` first. Nevertheless it is a good idea to always start your CGAL programs with including `<CGAL/basic.h>`.

1.12 Installation on Cygwin

Cygwin is a free Unix-like environment for MS-Windows, distributed by Red Hat. It consists of a port of a large number of GNU tools, such as bash, make, gcc, gas, file utilities, etc, as well as tools ensuring an ability to emulate Unix-like access to resources, for instance mount. For a comprehensive introduction and details, see `http://www.cygwin.com/` .

Make sure that the link `/bin/sh.exe` exists. If not, create it:

```
cd /bin
ln -s bash.exe sh.exe
```

1.12.1 Pathnames

Cygwin has a UNIX-like way of navigating hard drives, NFS shares, etc. This is also the way in which directories and pathnames have to given to the installation script. They are automatically converted to Win32-style pathnames when given to the compiler or linker.

The main difference is that directories are separated by slash ("/") rather than by backslash ("\"). The other difference is concerned with specifying drives. One way is to use POSIX-style pathnames that map Win32-style drives (`A:`, `B:`) to `//a/...`, `//b/...` respectively. For instance, the path `D:\Mystuff\Mydir\LEDA` translates to `//d/Mystuff/Mydir/LEDA`.

Alternatively, it can be done using the mount utility, that can be used to establish a map between Win32-style drives and the Unix-like style. More precisely, it maps the forest of the directories/files on Win32-drives to a tree with the root that is usually located at the top level of the boot drive, say `C:`. The root location can be seen by typing `mount` command without parameters. For instance, if `D:` is mounted on `C:\ddrive`[16] then the path `D:\Mystuff\Mydir\LEDA` translates to `/ddrive/Mystuff/Mydir/LEDA`.

Upper/lower case and spaces in file names Behavior of Cygwin in this regard might be different from the MS Windows behavior. In particular, using spaces in filenames should be avoided.

Links, shortcuts, etc. should be avoided as well.

1.12.2 MS Visual C++ -Setup

A number of environment variables has to be set (or updated) in order to use the installation.

`PATH` should contain MS Visual `C++` command line tools locations. The environment variables `INCLUDE` and `LIB` should point to the location of MS Visual `C++` header files and to the location of the MS Visual `C++` libraries, respectively.

First, the memory for environment variables has to be increased. Select the Cygwin icon from the Start-menu, press the right mouse button and choose *Properties*. Go to *Memory*, select *Initial Environment*, set it to at least 2048 and *apply* the changes.

Second, edit the file `cygwin.bat`, located in the cygwin main directory and add the line

[16] by typing `mount D: /ddrive`

```
call C:\VisualStudio2003\Common7\Tools\vsvars32.bat
```

customized according to where MS Visual C++ is installed on your system. Depending on the version of MS Visual C++ you might have to replace VSVARS32.BAT by MSCVARS32.BAT or VCVARS32.BAT and the script may reside in a different sub-directory.

All libraries that are used by an application have to be compiled with the same options[17] controlling the use of debugging and multithreading.

1.13 Using CGAL and LEDA

CGAL supports LEDA in the following ways.

1. There are support functions defined for the LEDA number types big_float, integer, rational and real (see the files <CGAL/leda_*>).
2. For all two-dimensional geometric objects there are input/output operators for a leda_window.
3. For all two-dimensional geometric objects there are output operators to a leda_ps_file.
4. The registration functions needed to interact with a leda_geowin are defined for all geometric objects from the CGAL kernel.

1.14 Compiler Workarounds

In CGAL, a number of compiler flags is defined. All of them start with the prefix CGAL_CFG. These flags are used to work around compiler bugs and limitations. For example, the flag CGAL_CFG_NO_LONG_LONG denotes that the compiler does not know the type long long.

For each compiler a file <CGAL/compiler_config.h> is defined, with the correct settings of all flags. This file is generated automatically by the install_cgal script, and it is located in the compiler specific include directory. This directory can be found below include/CGAL/config/; it is named according to the compiler's CGAL-OS description (cf. Section 1.9).

The test programs used to generate the compiler_config.h file can be found in config/testfiles. Both compiler_config.h and the test programs contain a short description of the problem. In case of trouble with one of the CGAL_CFG flags, it is a good idea to take a look at it.

The file CGAL/compiler_config.h is included from <CGAL/basic.h>. As discussed in Section 1.11, <CGAL/basic.h> should always be the first file included in any application using CGAL.

1.14.1 Standard Header Replacements

Some compilers do still not provide a complete standard library. In particular they fail to provide the C++ wrappers for files from the standard *C* library, like cstddef for stddef.h. The CGAL install scripts checks for all standard header files and generates a simple wrapper file in the CGAL include directory for those that

[17] MS Visual C++ compilation/linking options -ML, -MT, -MD, -MLD, -MTD, -MDD

are missing. These wrapper files include the corresponding C header files and add all symbols required by the C++ standard into namespace *std*. You can turn off the additions to namespace *std* by defining the macro *CGAL_NO_STDC_NAMESPACE*.

1.15 Compiler Optimizations

You may have noticed that we do not set optimizer flags as -O by default in the include makefiles(see section 1.10 for a description of the makefile structure in CGAL). The main reason for not doing this is that compilers run much more stable without. On the other hand, most if not all CGAL programs will run considerably faster when compiled with optimizations! So if you are going for performance, you should/have to add -O, -O3 or maybe more specific optimizer flags (please refer to the compiler documentation for that) to the CXXFLAGS variable in your application makefile:

```
#--------------------------------------------------------------------#
#                    compiler flags
#--------------------------------------------------------------------#
# The flag CGAL_CXXFLAGS contains the path to the compiler and is defined
# in the file CGAL_MAKEFILE. You may add your own compiler flags to CXXFLAGS.

CXXFLAGS = $(CGAL_CXXFLAGS) -O
```

1.16 Troubleshooting

This section contains some remarks about known problems and the solutions we propose. If your problem is not listed here, please have a look at the bug reporting instructions from the CGAL homepage:

```
http://www.cgal.org
```

In case of problems or questions related to installation, please include a copy of the install.log file for reference.

1.16.1 Compiler Version Test Execution Failed

Possibly already during the startup of the install script, the execution of the compiler version test might fail with the following (or similar) error message.

```
ld.so.1: ./tmp_test: fatal: libstdc++.so.5:
  open failed: No such file or directory
```

This means that the standard C++ library for your compiler is installed in a directory that is not on your current runtime linker path. You can solve this problem by adding the directory containing libstdc++.so to your runtime linker path, usually represented by the environment variable LD_LIBRARY_PATH.

For example, if you have a standard gcc installation below /software/gcc-3.3.2/, you would type

```
export LD_LIBRARY_PATH=/software/gcc-3.3.2/lib:$LD_LIBRARY_PATH
```

for Bourne shell alikes, while for `csh` descendants the syntax is

```
setenv LD_LIBRARY_PATH /software/gcc-3.3.2/lib:$LD_LIBRARY_PATH
```

You might want to add this command to your shell startup file.

Alternatively, you can build the runtime linker path into the executables by setting corresponding custom linker flags (cf. Section 1.6.2).

1.16.2 The "Long-Name-Problem" on Solaris

The system assembler and linker on Solaris 2.5 and 2.6 cannot handle symbols with more than 1024 characters. But this number is quickly exceeded where one starts nesting templates into each other. So if you encounter strange assembler or linker errors like

```
/usr/ccs/bin/as: "/var/tmp/cc0B5iGc.s", line 24:
error: can't compute value of an expression involving an external symbol
```

there is a good chance that you suffer from this "long-name" problem.

A solution is to install the GNU-binutils[18] and to tell the compiler that it shall use the GNU– instead of the native tools. From the compiler-menu (described in section 1.6.2) you can set the corresponding option through the custom compiler flags, i.e. for `gcc` you would add

```
-B/my/path/to/gnu/binutils/bin
```

assuming you installed the GNU-binutils executables in `/my/path/to/gnu/binutils/bin`.

If you cannot (or do not want to) install GNU-binutils, there is a workaround that lets you compile, link and run your programs, but it prevents debugging, since the executables have to be stripped. In short the workaround is to compile with `-g` and to link with `-z nodefs -s` on Solaris, `-U -s` on IRIX, respectively.

In order to still have portable makefiles (see section1.10), we define flags `LONG_NAME_PROBLEM_CXXFLAGS` and `LONG_NAME_PROBLEM_LDFLAGS` in the include makefiles which are empty except for the Solaris platform where they are set as stated above. In order to use these flags, edit your application makefile and add the flags to `CXXFLAGS` resp. `LDFLAGS` as indicated below.

```
#--------------------------------------------------------------------#
#                    compiler flags
#--------------------------------------------------------------------#
# The flag CGAL_CXXFLAGS contains the path to the compiler and is defined
# in the file CGAL_MAKEFILE. You may add your own compiler flags to CXXFLAGS.

CXXFLAGS = $(LONG_NAME_PROBLEM_CXXFLAGS) $(CGAL_CXXFLAGS)
```

[18] see `http://www.gnu.org/software/binutils/`

```
#--------------------------------------------------------------------#
#                    linker flags
#--------------------------------------------------------------------#
# The flag CGAL_LDFLAGS contains common linker flags and is defined
# in the file CGAL_MAKEFILE. You may add your own linker flags to CXXFLAGS.

LDFLAGS = $(LONG_NAME_PROBLEM_LDFLAGS) $(CGAL_LDFLAGS)
```

1.16.3 LEDA and STL Conflicts

If you are using an old version of LEDA, the combination of LEDA and STL may give some problems. In order to avoid them, it is highly recommended to use the latest LEDA release, since this is what we test CGAL with.

With MS Visual C++ or BORLAND C++ , LEDA has to be compiled and used with the LEDA_STD_HEADERS flag set. CGAL uses C++ standard conforming headers[19], while LEDA can also work with the old-style header files; but mixing the styles is strictly forbidden. Before compiling LEDA edit the file $(LEDAROOT)/incl/LEDA/system.h and uncomment the #define in the following fragment there.

```
// use c++ std headers
//#define LEDA_STD_HEADERS
```

MS Visual C++ Specifics. If a binary release of LEDA is used, make sure that it is one of them that uses new-style headers. Namely, among the self-extracting executables, choose one of these that have the name ending with -std.exe.

1.17 Scripts

1.17.1 cgal_create_makefile

The Bourne-shell script cgal_create_makefile is contained in the CGAL-3.2/scripts directory. It can be used to create makefiles for compiling CGAL applications. Executing cgal_create_makefile in an application directory creates a makefile containing rules for every *.C file there.

In order to use this makefile, you have to specify the CGAL include makefile (see section 1.10) to be used. This can be done be either setting the environment variable CGAL_MAKEFILE or by editing the line

```
  # CGAL_MAKEFILE = ENTER_YOUR_INCLUDE_MAKEFILE_HERE
```

of the created makefile. First remove the "#" at the beginning of the line and then replace the text after "=" by the location of the include makefile.

Finally type make to compile the application programs.

[19] the ones that do not have .h suffix

Index