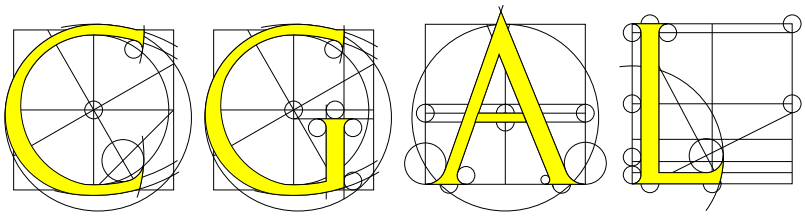

Installation Guide

Release 3.0.1, February 2004

Authors

Michael Hoffmann, ETH Zürich.
Dmitrii Pasechnik, Delft University of Technology.
Wieger Wesselink, Utrecht University.

Acknowledgement

This work was supported by the ESPRIT IV LTR Projects No. 21957 (CGAL) and No. 28155 (GALIA).

Contents

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. 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 GNU Multiple Precision library GMP[1], the CORE library[2] for robust numerical and geometric computation, LEDA, the Library of Efficient Datatypes and Algorithms[3], or Trolltech's[4] QT toolkit.

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 reasonnably conform to the ISO 14882 standard for C++.

`CGAL-3.0.1` supports the following compilers/operating systems:

compiler	**operating system**
SGI Mips(Pro) `CC` 7.3 (n32 and 64) [7]	IRIX 6.5
GNU `g++` 2.95.3, 3.0, 3.1, 3.2, 3.3 [9]	IRIX 6.5 / Solaris 2.6+ / Linux 2.x / MS Windows 95/98/2000/XP/NT4[10]
SUNPRO `CC` 5.3, 5.4, 5.5 [11]	Solaris 2.6+
MS Visual `C++` 7.0, 7.1 (.NET) [12]	MS Windows 95/98/2000/XP/NT4[10]
INTEL `C++` 7.1 [13]	MS Windows 95/98/2000/XP/NT4[10]

If you are going to install CGAL using Cygwin[12], please read Section 12 first.

For the SUNPRO `CC` 5.3 compiler, the latest patch set, in particular Patch 111685-05, is required. You can check with `CC -V` which compiler version you have installed. It should read

```
CC: Sun WorkShop 6 update 2 C++ 5.3 Patch 111685-05 2002/02/03
```

or later.

Note that GNU `g++` 2.96/97 are not official `gcc` releases that are neither supported by the `GCC` team[13] nor by CGAL. Please upgrade to GNU `g++` 3, if you happen to have this compiler.

3 Getting CGAL

The CGAL library can be downloaded from the CGAL homepage:

[1] `http://www.gnu.org/software/gmp`
[2] `http://www.cs.nyu.edu/exact/core/`
[3] `http://www.mpi-sb.mpg.de/LEDA`
[4] `http://www.trolltech.com`
[7] `http://www.sgi.com/developers/devtools/languages/mipspro.html`
[9] `http://gcc.gnu.org/`
[10] with Cygwin (`http://www.cygwin.com`)
[11] `http://www.sun.com/software/Developer-products/cplusplus/`
[12] `http://msdn.microsoft.com/visualc/`
[13] `http://developer.intel.com/software/products/compilers/`
[12] `http://www.cygwin.com`
[13] `http://gcc.gnu.org/gcc-2.96.html`

```
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-3.0.1` will be created. This directory contains the following subdirectories:

directory	contents
`auxiliary`	packages that can optionally be used with CGAL
`config`	configuration files for install script
`demo`	demo programs (some of them need LEDA, 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 directory `auxiliary` contains a distribution of the GNU Multiple Precision library GMP[14]. The directory `src/Core` contains a distribution of the CORE library[15] for robust numerical and geometric computation. Both libraries are not part of CGAL and have their own licenses.

3.1 Visualization

The programs in 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 based applications.
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 need do nothing more than compile CGAL with LEDA support.
Some demo programs for 3D structures require the geomview program for visualisation. This is available from `http://www.geomview.org` (note that it does not run on MS Windows).

4 Installing CGAL

The directory `CGAL-3.0.1` 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 5. This provides you with an overview on how the interactive installation works. If you want more detailed information about specific menus and their options, take a look at section 6. Finally, for the non-interactive mode refer to section 7.
If you want to use LEDA together with CGAL, have a look at section A.

[14] `http://www.gnu.org/software/gmp`
[15] `http://www.cs.nyu.edu/exact/core/`

5 A Sample Installation

In this section we sketch an example installation on a SUN running Solaris 2.9 with the GNU g++ 3.3 compiler. For a complete description of the different menus and their options refer to section 6.

5.1 Starting the script

Go to the CGAL-3.0.1 directory and enter the command

```
./install_cgal -i
```

You get a message indicating the CGAL version you are going to install and that you are running the interactive mode. Then it takes some time while the script locates a number of utility programs. You will not get informed about this[16], but see some dots written to the screen indicating progress.

```
-------------------------------------------------------
  This is the install script for CGAL 3.0.1
-------------------------------------------------------

starting interactive mode - one moment, please
.......

  Choosing compiler GNU 3.3.2.
```

If there is any compiler installed on your system and accessible through your PATH environment variable that is supported by CGAL, one of these compilers is chosen. If there is more than one compiler installed on your system (and supported by CGAL), you may choose to use a different compiler from the compiler menu (cf. Section 6.2).

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

```
  ************************************************************
  **              CGAL 3.0 Installation Main Menu           **
  **              ------------------------------            **
  **                                                        **
  **   OS:                  sparc_SunOS-5.9                 **
  **   Compiler:            GNU 3.3.2                       **
  **   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                                   **
  **   <S>  Support Menu                                    **
  **   <T>  Test (and save) setup                           **
  **   <A>  Run all setup tests (no cache)                  **
  **                                                        **
  **   <B>  Build 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 (such as GMP, CORE, LEDA, or QT). 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.

[16] If you are that curious what happens exactly, have a look at the file CGAL-3.0.1/install.log.

5.2 Building the CGAL libraries

In a first step, you should test the current setup by typing "`t`". Then a number of tests are done to check whether your compiler supports certain language constructs or has specific bugs. There is quite a number of these tests, so this step may take a while. For each test you should get a message what particularly is tested at the moment and what the result is.

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

<many lines omitted>

Testing for VC7_PRIVATE_TYPE_BUG ... ok.
Saving current setup ... done.
```

If all these tests have been completed successfully, the current settings are saved into a file that resides in the directory `CGAL-3.0.1/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.

5.3 Building the CGAL libraries

We are now ready to build the CGAL libraries. Just type "`b`" to start compilation. Building consists of three steps:

1. writing the include makefile,
2. compiling the static libraries *and*
3. compiling the shared libraries.

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 3.0                   **
**                    ------------------                   **
**                                                         **
*************************************************************

OS:           sparc_SunOS-5.9
COMPILER:     GNU 3.3.2
GMP:          not supported
GMPXX:        not supported
CORE:         not supported
LEDA:         not supported
Qt:           not supported

Generating Makefiles ... done.
Building CGAL_lib ... done.
Building CGAL_sharedlib ... done.

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

That's all, it's done. Press "<ENTER>" to return to the main menu and proceed by installing for a different compiler (go to the compiler menu and choose "c" to get a list of supported compilers detected on your system), or with GMP, CORE, LEDA, or QT support (go to the GMP, CORE, LEDA, or QT menu, respectively). 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-3.0.1/make/makefile_sparc_SunOS-5.6_g++-2.95.3
```

while for csh descendants the syntax is

```
setenv CGAL_MAKEFILE CGAL-3.0.1/make/makefile_sparc_SunOS-5.6_g++-2.95.3
```

In Section 10 you can find more information on the CGAL makefile structure, and how to set CGAL_MAKEFILE when using CGAL on several platforms.

6 The interactive mode

To run the install script in the interactive mode, go to the CGAL-3.0.1 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 3 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 6.2.
If you want to use GMP, CORE, LEDA, or QT with CGAL, you will have to go to the *gmp menu* (cf. Section 6.4), *core menu* (cf. Section 6.5), *leda menu* (cf. Section 6.6), or *qt menu* (cf. Section 6.7), respectively.
Finally you can 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 an STL test, a GMP test, a CORE test, a LEDA test, and a QT test. But not all tests are performed always; e.g., the GMP test is only done, if you enabled GMP support. The install script keeps track of the tests passed and only tests again, if you change the setup in a way that might affect the test result. If you want to redo *all* tests, you have to choose option "a" from the main menu. This also retests for GMP/LEDA/QT installations in system directories. Otherwise, this is only done the first time you enable gmp/LEDA/QT support for an OS/compiler combination.

6.1 Files created during installation

The install script stores all relevant settings for an OS/compiler combination in the directory

CGAL-3.0.1/config/install/<*CGAL-OS-description*>

where <*CGAL-OS-description*> identifies your OS/compiler combination in a way specified in section 9. [17] This saves you typing everything again, if you upgrade CGAL or another package that makes recompiling the CGAL libraries necessary.
Besides the config files, install_cgal 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-3.0.1) 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

[17] Note that these files are only OS/compiler specific, i.e. there are no different files for with and without LEDA support.

6.2 The Compiler Menu

Here is the place to set the compiler specific options, such as the compiler to use (if more than one has been detected) and custom compiler or linker flags.

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.

6.3 The Support Menu

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

Support Menu

<**G**> Setup support for the GNU Multiple Precision library GMP.

<**C**> Setup support for the CORE library for robust numerical and geometric computation.

<**L**> Setup support for LEDA, the Library of Efficient Datatypes and Algorithms.

<**K**> Setup support for Trolltech's QT toolkit.

6.4 The GMP Menu

This menu is to set GMP (GNU Multiple Precision Library) specific options, if you plan to use GMP together with CGAL. In case you do not already have GMP installed on your system, a GMP distribution is shipped with CGAL. You can find it in the `auxiliary` directory. The menu provides an option to install GMP in your CGAL directory tree[18], but – of course – you can also install GMP independently from CGAL.

If GMP support is enabled for the first time, the script tests whether GMP is installed in standard system directories or in the CGAL tree. If this test does not succeed, you have to supply directories containing the GMP header files (`GMP_INCL_DIR`) and GMP libraries (`GMP_LIB_DIR`). Even if the tests are passed, you still have the option to set these directories differently.

If you decide to install the GMP distribution shipped with CGAL from the install script, GMP will be configured by calling its *configure* script. GMP being a C library, it requires a C compiler. Its *configure* script usually does a good job at finding a C compiler on the system, but it also gives the possibility to specify it by setting the environment variable *CC* before calling *configure*. If you want the CGAL install script to build GMP and specify the C compiler to be used, you can also to the same, by setting the *CC* environment variable (see the GMP installation documentation for details). If you need a more complex configuration of GMP, we recommend that you install GMP yourself separately.

If GMP support is enabled, you may additionnally enable support for GMPXX, GMP's built-in C++ interface, from the GMP menu. Note that this is not supported by all C++ compilers, you should check the GMP manual

[18] This option on MS Visual `C++` just unpacks a pre-compiled library that comes with CGAL.

for more information. Once you choose to have GMPXX support, then if you install GMP from the CGAL tree, GMP will be configured with C++ support.

GMP Menu

<**C**> Install the GMP distribution shipped with CGAL in the CGAL directory tree.

<**G**> Enable/Disable GMP support in CGAL.

<**X**> Enable/Disable GMPXX support in CGAL.

<**I**> (*present if* GMP *support is enabled)* Set the include directory for GMP.

<**L**> (*present if* GMP *support is enabled)* Set the directory containing the GMP libraries.

<**M**> (*present if* GMP *support is enabled, there is a* GMP *installation in system directories or in the* CGAL *tree and* `GMP_INCL_DIR` *or* `GMP_LIB_DIR` *have been set)* Use GMP installation from system directories / CGAL tree.

6.5 The CORE Menu

This menu is to set CORE specific options, if you plan to use CORE together with CGAL. Since CORE requires GMP, you have to enable GMP support together with CORE support. Refer to Section 6.4 for how to setup GMP support for CGAL.

The current CORE release is shipped together with CGAL. If CORE support is enabled, the CORE library is built and installed together with the CGAL libraries. Hence, if you enable CORE support, you have to (re)build the CGAL libraries afterwards (cf. Section 5.3). Another option is to use a CORE installation that is independent from CGAL. For this you have to supply directories containing the CORE header files (`CORE_INCL_DIR`) and the CORE library (`CORE_LIB_DIR`).

CORE Menu

<**C**> Enable/Disable CORE support in CGAL.

<**I**> (*present if* CORE *support is enabled)* Set the include directory for CORE.

<**L**> (*present if* CORE *support is enabled)* Set the directory containing the CORE library.

<**Z**> (*present if* CORE *support is enabled and* `CORE_INCL_DIR` *or* `CORE_LIB_DIR` *have been set)* Use CORE installation from CGAL directories.

6.6 The LEDA Menu

This is the place to set LEDA specific options, if you plan to use LEDA together with CGAL (see also Section A). In order to enable LEDA support in CGAL, LEDA has to be installed on your system.

If LEDA support is enabled the first time, the script tests whether LEDA is installed in standard system directories. If this test does not succeed, you have to supply directories containing the LEDA header files (`LEDA_INCL_DIR`) and LEDA libraries (`LEDA_LIB_DIR`). Even if the tests are passed, you still have the option to set these directories differently.

LEDA Menu

<**E**> Enable/Disable LEDA support in CGAL.

<**I**> (*present if* LEDA *support is enabled)* Set the include directory for LEDA.

<**J**> (*present if* LEDA *support is enabled,* LEDA *headers have been found in a system include directory and* `LEDA_INCL_DIR` *has been set)* Use LEDA header from system include directory.

<**L**> (*present if* LEDA *support is enabled)* Set the directory containing the LEDA libraries.

<**M**> (*present if* LEDA *support is enabled,* LEDA *libs have been found in a system lib directory and* `LEDA_LIB_DIR` *has been set)* Use LEDA libraries from system lib directory.

6.7 The Qt Menu

This menu is to set QT specific options, if you plan to use QT together with CGAL. Note that for use with CGAL, QT version 2.2 (or later) if required. In order to enable QT support in CGAL, QT has to be installed on your system first. Unlike for GMP, there is no option to install QT from the CGAL installation script. For information on QT, please refer to

`http://doc.trolltech.com/`

advanced

The QT menu has a basic mode and an advanced one. If your QT installation is standard, you shouldn't have to go into the advanced mode.
The QT menu starts in basic mode, unless QT is installed in standard system directories (because in that case, the advanced mode is quicker).

advanced

6.7.1 Basic mode (to use with a standard Qt installation)

If QT support is enabled the first time, the script tests whether the `$QTDIR` environment variable points to a valid QT directory. is installed in standard system directories. If that is not the case, you have to supply the QT directory containing your QT installation. Even if the test passed, you still have the option to set this directory differently. If your QT installation is not standard, you would have to go into the advanced mode.

Qt basic Menu

<**K**> Enable/Disable Qt support in CGAL.

<**A**> (*present if Qt support is enabled)* Go to the advanced mode

<**D**> (*present if Qt support is enabled*) Set the Qt directory.

<**E**> (*present if Qt support is enabled,* `$QTDIR` *points to a valid Qt directory and* `QT_DIR` *has been set*) Use `$QTDIR` as Qt directory.

advanced

6.7.2 Advanced mode (to use with a non-standard Qt installation)

In that mode, you have to specify separately directories containing the QT header files (`QT_INCL_DIR`) and the QT library (`QT_LIB_DIR`), and the path to the MOC[19] executable (`QT_MOC`) unless they are in system directories.

[19] `http://doc.trolltech.com/moc.html`

Even in that case, you still have the option to set them differently.

Qt advanced Menu

<**K**> Enable/Disable Qt support in CGAL.

<**B**> (*present if Qt support is enabled)* Go to the basic mode

<**I**> (*present if Qt support is enabled*) Set the directory containing Qt headers.

<**J**> (*present if Qt support is enabled, Qt headers are in system directories and* `QT_INCL_DIR` *has been set*) Use Qt headers from system include directories.

<**L**> (*present if Qt support is enabled*) Set the directory containing Qt library.

<**M**> (*present if Qt support is enabled, Qt library is in system directories and* `QT_LIB_DIR` *has been set*) Use Qt library from system library directories.

<**O**> (*present if Qt support is enabled*) Set the path to MOC executable.

<**P**> (*present if Qt support is enabled, MOC is in path and* `QT_MOC` *has been set*) Use Qt MOC executable in path.

advanced

7 The non-interactive mode

To run the install script in the non-interactive mode, go to the `CGAL-3.0.1` directory and enter the command

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

where `<compiler>` is the C++ compiler executable.
You can either specify a full path, e.g. `/usr/local/bin/g++`, or just the basename, e.g. `g++`, which means the script searches your `PATH` for the compiler location. If your compiler call contains whitespaces it has to be quoted, e.g. `./install_cgal -ni "CC -n32"`. The options given this way become part of your CGAL-OS description (see section 9) which is useful e.g. to distinguish between different compilers using the same frontend (SGI Mips(Pro) `CC` on `IRIX6`).

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 one or more of the following conditions apply to you:

- you want to use GMP together with CGAL (Section 7.1),
- you want to use GMP's built-in C++ interface together with CGAL (Section 7.2),
- you want to use CORE together with CGAL (Section 7.3),
- you want to use LEDA together with CGAL (Section 7.4),
- you want to use QT together with CGAL (Section 7.5).

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.

Finally 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 and you should get a corresponding message.

To compile the CGAL libraries go now to the `src` directory. Then type `make -f makefile_lib` to compile the CGAL object library and `make -f makefile_sharedlib` to compile the CGAL shared object library. If you want to make changes to the makefiles first, see section 10 for an explanation of the makefile structure of CGAL.

If you enabled CORE support and want to use the CORE distribution shipped with CGAL, go to the `src/Core` directory and type `make -f makefile_Core` to compile the CORE library.

If you enabled QT support, go to the `src/CGALQt` directory and type `make -f makefile_Qt` to compile the CGAL QT support library.

7.1 Setting up support for GMP

By default there is no support for GMP, but you can change this easily by use of the command line option "`-gmp`". If GMP is installed in system directories on your system, you are already done now. If this is not the case, you have to supply the directories containing the GMP header files ("`--GMP_INCL_DIR` <*dir*>") and the GMP library ("`--GMP_LIB_DIR` <*dir*>").

7.2 Setting up support for GMPXX

By default there is no support for GMPXX, GMP's built-in C++ interface, but you can change this easily by use of the command line option "`-gmpxx`". The only requirement for this to work is that GMP support is enabled correctly.

7.3 Setting up support for CORE

By default there is no support for CORE, but you can change this easily by use of the command line option "`-core`". If you want to use the CORE distribution shipped with CGAL, this is all you have to do. Otherwise, you also have to supply the directories containing the CORE header files ("`--CORE_INCL_DIR` <*dir*>") and the CORE library ("`--CORE_LIB_DIR` <*dir*>").

7.4 Setting up LEDA support

See also section A. By default there is no support for LEDA, but you can change this easily by use of the command line option "`-leda`". If LEDA is installed in system directories on your system, you should indicate this by setting the flags "`--leda-sys-incl`" resp. "`--leda-sys-lib`". If this is not the case, you have to supply the directories containing the LEDA header files ("`--LEDA_INCL_DIR` <*dir*>") resp. the LEDA libraries for your compiler ("`--LEDA_LIB_DIR` <*dir*>").

7.5 Setting up support for Qt

By default there is no support for QT, but you can change this easily by use of the command line option "`-qt`". If QT is installed in system directories on your system or the `$QTDIR` environment variable points to the QT directory, you are already done now. If this is not the case, you have to supply either the QT directory ("`--QT_DIR` <*dir*>") if your QT installation is standard, either directories containing the QT header files ("`--QT_INCL_DIR` <*dir*>") and the QT library ("`--QT_LIB_DIR` <*dir*>"), and the path to the MOC[20] executable ("`--QT_MOC` <*exe*>") if your QT installation is not standard.

[20] `http://doc.trolltech.com/moc.html`

7.6 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:

```
./install_cgal -ni g++ --CUSTOM_CXXFLAGS "-I/my/include -O2"
```

7.7 Other Options

There are some less important features of the install script we will summarize here.

First of all, you can get the version number of `install_cgal` with option "`--version`". Note that 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 9). The compiler can either be given by an absolute path like

```
./install_cgal -os /usr/local/gcc-2.95.3/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 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.

8 Upgrading a previous CGAL installation

In case you already have a previous release of CGAL installed on your system, you might like to reuse your configuration files and GMP installations. Simply use the following command to copy them into the right place:

```
./install_cgal --upgrade <OLD_CGAL_DIR>
```

where `<OLD_CGAL_DIR>` is the root directory of your existing CGAL installation
(e.g. `/pub/local/CGAL-2.4`). You can then build all libraries for the actual operating system that existed in your previous CGAL installation with

```
./install_cgal --rebuild-all
```

If you want to install CGAL for more than one operating system in the same directory structure, you have to run the latter command (`rebuild-all`) once on each operating system.

Using `--build-all` instead of `--rebuild-all` will save you the time of the configuration tests, and will only rebuild the libraries.

If you want to install only one configuration on a given operating system, you can specify its name (the base name of a file in `CGAL-3.0.1/config/install`) with the option `--rebuild <config>` or `--build <config>`.

Note that some compilers that have been supported in previous CGAL releases might not be supported in CGAL-3.0.1 anymore, see section 2. Trying to build CGAL-3.0.1 with these compilers will most probably fail. You can solve this problem by deleting the obsolete config files (see section 6.1) from `CGAL-3.0.1/config/install` before issuing the `rebuild-all` command.

Similarly, you might want to use compilers with CGAL-3.0.1 that have not been supported in previous releases. For these compilers please follow the usual procedure as described in section 6 or 7.

9 Identifying 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-3.0.1/lib`) (and analogously its own include makefile in `CGAL-3.0.1/make`) named as determined by the following scheme.

<*arch*>_<*os*>-<*os-version*>_<*comp*>-<*comp-version*>[_LEDA]

<**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).

The suffix _LEDA is appended to indicate LEDA support.

We call the resulting string CGAL-OS description.
Examples are `mips_IRIX-6.2_CC-7.2` or `sparc_SunOS-5.5_g++-2.95.3_LEDA`.
You can use the install script to get your CGAL-OS description, see section 7.7.

10 The CGAL makefile structure

The CGAL distribution contains the following makefiles:

- `CGAL-3.0.1/src/makefile_lib` for compiling the CGAL object library `libCGAL.a`,
- `CGAL-3.0.1/src/makefile_sharedlib` for compiling the CGAL shared object library `libCGAL.so`,
- `CGAL-3.0.1/examples/makefile` as sample makefile *and*
- `CGAL-3.0.1/examples/*/makefile` for compiling the CGAL example programs.

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 = /users/jannes/CGAL-3.0/make/makefile_<CGAL-OS description>
include $(CGAL_MAKEFILE)
```

The file `CGAL_MAKEFILE` is an include file with platform dependent makefile settings. The abbreviation `<CGAL-OS description>` (see section 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-3.0.1/make` directory. For convenience, the `install_cgal` script will substitute the include makefile that was generated most recently.

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. An alternative way to do this is to create an environment variable `CGAL_MAKEFILE`. To pass the value of the environment variable to the makefile you can either comment out the `CGAL_MAKEFILE` line in the makefile or use an appropriate command line option for the make utility. A comfortable way to set `CGAL_MAKEFILE` is by using `install_cgal -os` (see section 7.7). E.g. if your compiler is `g++`, you would type

```
CGAL_MAKEFILE='<insert your CGAL-3.0.1 dir>/install_cgal -os g++'
```

in bourne shell resp.

```
setenv CGAL_MAKEFILE '<insert your CGAL-3.0.1 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`.

11 Compiling a CGAL application

The directory `CGAL-3.0.1/examples` contains a small program (example.C) and a sample makefile with some comments. The `CGAL_MAKEFILE` variable in this makefile (see section 10) is automatically substituted by the `install_cgal` script and equals the most recently generated include makefile in the `CGAL-3.0.1/make` directory. After the installation of CGAL this sample makefile is ready for use. Just type '`make example`' to compile the program `example.C`. There is a script for conveniently creating makefiles for CGAL applications, see section E.1.

Furthermore the directories `CGAL-3.0.1/examples` and `CGAL-3.0.1/demo` contain many subdirectories with non-graphical and graphical example programs. In all these directories you will find a makefile that is ready for use.

12 Installation on Cygwin

Cygwin is a free Unix-like environment for MS-Windows, distributed by Cygnus Solutions. For our tests we have used version 1.3.2 and *B*-20.1.

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
```

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 seperated 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`[21] 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 better be avoided.

Links, shortcuts, etc should be avoided as well.

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. The interface for doing this is different for NT and for Win9*.

MS Windows-NT4.0. One can set the corresponding environment variables using the usual NT interface[22]. Alternatively, they can be set in the `.bashrc` file for the particular user, or in the system-wide `bash` customization file (usually `/etc/bashrc`).

The result should look roughly as follows, assuming that `C:\PROGRA~1\MICROS~2\` is the location of the MS Visual `C++` installation.

```
LIB=C:\PROGRA~1\MICROS~2\VC98\LIB
INCLUDE=C:\PROGRA~1\MICROS~2\VC98\INCLUDE
```

and `PATH` should contain

```
/PROGRA~1/MICROS~2/Common/msdev98/BIN:
/PROGRA~1/MICROS~2/VC98/BIN:/PROGRA~1/MICROS~2/Common/TOOLS:
/PROGRA~1/MICROS~2/Common/TOOLS/WINNT
```

MS Windows-9*. 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` (or `cygnus.bat` in Cygwin 0.9), located in the cygwin main directory and add the line

```
call C:\PROGRA~1\MICROS~2\VC98\Bin\MSCVARS32.BAT
```

where `C:\PROGRA~1\MICROS~2\` has to be 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 `MSCVARS32.BAT` by `VCVARS32.BAT`.

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

[22] open MyComputer, press right mouse button, select Properties, select Environment, set the relevant variables

A 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.
5. CGAL defines the following LEDA-related compiler flags when LEDA is used: `CGAL_USE_LEDA` and `LEDA_PREFIX`.

The include makefiles in the `CGAL-3.0.1/make` directory corresponding to LEDA can be recognized by the suffix "`_LEDA`".

B 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 9).

The test programs used to generate the `compiler_config.h` file can be found in `CGAL-3.0/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.

Within CGAL, the file `<CGAL/basic.h>` manages all configuration problems. In particular, it includes the file `CGAL/compiler_config.h`. 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>`.

B.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 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*.

C Compiler Optimizations

You may have noticed that we do not set optimizer flags as `-O` by default in the include makefiles(see section 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
```

D 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 CGAL homepage:

`http://www.cgal.org`

or send an email to `contact@cgal.org`.

D.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 6.2).

D.2 Defect in the G++ 3.2 ABI

Some versions of `gcc`, for example GNU `g++` 3.3.0, have problems in their C++-ABI, that surface in error messages similar to the following.

```
error: due to a defect in the G++ 3.2 ABI, G++ has assigned
  the same mangled name to two different types.
```

If this occurs to you, please seriously consider upgrading your compiler. This issue is fixed starting from GNU `g++` 3.3.1. Alternatively, you can add `-fabi-version=0` to your custom compiler flags[23]. In interactive mode, this is done via the Compiler Menu, as described in Section 6.2. Afterwards rebuild the libraries. But note that changing the ABI might have side effects. Hence, a compiler upgrade is the recommended fix here.

[23] Thanks to Christopher Intemann for pointing this out.

D.3 The “Long-Name-Problem” on IRIX6

The system assembler and linker on IRIX6 cannot handle symbols with more than 4096 characters. But this number can be exceeded when one starts nesting templates into each other. So if you encounter strange assembler or linker errors like

```
as: Error: /var/tmp/ccPBl5vJ.s, line 41289: Truncating token:
<some ridiculously long token snipped>
```

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

In contrast to Solaris, using the GNU binutils does not work, since `gas` has not been ported to IRIX6 yet. The solution proposed in the GCC faq[24] is to compile with the (experimental) option `-fsquangle`, that enables compression of symbol names. This option was experimental and has disappeared in GCC 3.0, where the ABI has been improved. So this is only interesting for GCC 2.95.3.

Citing from the above FAQ:

```
Note that this option is still under development, and subject to
change. Since it modifies the name mangling mechanism, you'll need to
build libstdc++ and any other C++ libraries with this option enabled.
Furthermore, if this option changes its behavior in the future, you'll
have to rebuild them all again. :-(

This option can be enabled by default by initializing
'flag_do_squangling' with '1' in 'gcc/cp/decl2.c' (it is not
initialized by default), then rebuilding GCC and any C++ libraries.
```

D.4 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[25] and to tell the compiler that it shall use the GNU– instead of the native tools. From the compiler-menu (described in section 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 section10), 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.

[24] `http://gcc.gnu.org/cgi-bin/fom.cgi?file=41`

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

```
#------------------------------------------------------------------#
#                    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)

#------------------------------------------------------------------#
#                    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)
```

D.5 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 conformant headers[26], 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++ -specific problems. Also, the LEDA and CGAL libraries have to be compiled with the same options controlling the use of debugging and multithreading. [27]

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`.

D.6 MS Visual C++ -specific C++ problems

D.6.1 MS Visual C++ 6.0

This compiler version is no longer supported starting with CGAL 3.0.

D.6.2 MS Visual C++ 7.0 (.NET)

`std::iterator_traits` is only partially supported, due to absence of partial specialization support in MS Visual C++ . This means that `std::iterator_traits<T>` must be explicitly instantiated when `T` is a C pointer.

CGAL provides these instantiations for most commonly used primitive data types and CGAL kernel types[28]. For other types the user must do this himself. This can be done by using the macro `CGAL_DEFINE_ITERATOR_TRAITS_POINTER_SPEC(T)`. Note that the parameter of this macro cannot contain symbols < and >. Thus the macro cannot be used with types that contain < and > directly. However, one can be done by first define a new

[26] the ones that do not have `.h` suffix
[27] MS Visual C++ compilation/linking options `-ML, -MT, -MD, -MLD, -MTD, -MDD`
[28] CGAL `Point, Segment`, etc.

type using `typedef` and then applying the macro to this new type. See e.g. `examples/Getting_started/advanced_hull.C` for an example of using this macro.

Note that this macro does not have any effect when other than MS Visual `C++` 7.0 compiler is used, so it can be safely left in the source code.

Other problems.

Here goes an incomplete list of problems encountered, and CGAL-specific workarounds, if available. Compiler error messages are meant to be hints only, and do not pretend to be complete, as well.

1. Partial specialization of template parameters is not supported. Do not use it. Error messages say that "the class has already been declared", or "unrecognizable template declaration/definition". See `config/testfiles/CGAL_CFG_NO_PARTIAL_CLASS_TEMPLATE_SPECIALISATION.C` for a test program illustrating the problem.
2. Compiler does not always match the most specialized instance of a function template correctly. So you get "error C2667: none of 2 overload have a best conversion".
   See `config/testfiles/CGAL_CFG_MATCHING_BUG_2.C` for a test program illustrating the problem.
3. Compiler does not support the Koenig lookup. That is, it does not search in the namespace of the arguments for the function. See `config/testfiles/CGAL_CFG_NO_KOENIG_LOOKUP.C`.
4. Internal compiler errors can sometimes be avoided by increasing the amount of memory available to the compiler. Use `-Zm<number>` option. In CGAL makefiles it is set to `-Zm900`, meaning "using 900% out of the usual memory limit".
5. `[...]/VC98/INCLUDE/xlocnum(268) : error C2587: '_U' : illegal use of local variable as default parameter` can occur[29]. The only workaround we know is to redefine the macro `_VIRTUAL` in `<xlocnum>`[30] to be empty. Search for `#define _VIRTUAL virtual` there and replace it by `#define _VIRTUAL` .
6. Various matching failures for overloaded functions and ctors. Use dummy parameters.
7. Avoid multiple forward declarations.
8. If necessary, simplify template parameters by using extra `typedef`s.

E Scripts

E.1 `create_makefile`

The bourne-shell script `create_makefile` is contained in the `CGAL-3.0.1/scripts` directory. It can be used to create makefiles for compiling CGAL applications. Executing `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 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.

[29] For instance, in CGAL `Min_circle` package

[30] Yes, in the MS Visual `C++` header! You need not edit the actual file though. Copy it to a directory that is searched ahead of the other directories. DISCLAIMER: *We do not know if the actions described in this footnote are legal in your country. You are on your own here.*

E.2 `use_cgal_namespace`

The perl script `use_cgal_namespace` is contained in the `CGAL-3.0.1/scripts` directory. It can be used to convert CGAL-1.* application sourcecode to the CGAL-2.* format. Basically, it replaces `CGAL_` prefixes by `CGAL::` namespace qualifiers. You have to give the files to convert as arguments, e.g.

```
use_cgal_namespace my_great_file1.C *.h
```

The original files are kept with the suffix `.bck`.

In order to use it, you first have to set the perl path correctly, i.e. replace `/net/bin/perl5` in the first line by the path to perl on your system (try `which perl`(5), if you do not know). Alternatively, you can type

```
perl -wi.bck -- use_cgal_namespace <FILES>
```

Index

Pages on which definitions are given are presented in **boldface**.