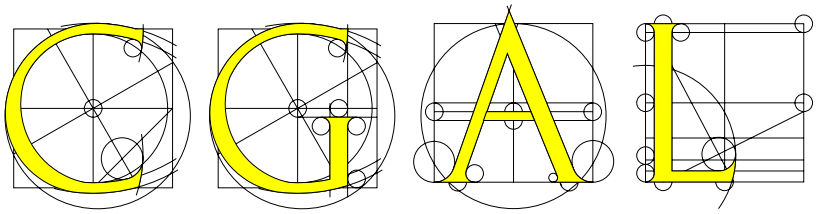

CGAL Installation Guide

Release 1.0, April 1998

Preface

Authors

Michael Hoffmann, ETH Zürich.
Wieger Wesselink, Utrecht University.

Acknowledgement

This work was supported by the ESPRIT IV Long Term Research Project No. 21957 (CGAL).

Contents

1 Introduction

CGAL is a *Computational Geometry Algorithms Library* written in C++, which is developed in an ESPRIT LTR project. The goal 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 your system. Besides that, you will find some information about the makefile structure of CGAL and the support for simultaneously using CGAL and LEDA, the Library of Efficient Datatypes and Algorithms[1], and/or the GNU Multiple Precision library GMP.

2 Prerequisites

In order to build the CGAL libraries you need a C++ compiler. Currently only a small number of recent compilers on Unix platforms are supported. The reason is that most compilers do not conform to the upcoming ANSI/ISO standard[2] and some of them have so many limitations/bugs that we could not work around all of them.

More precisely, `CGAL-1.0` supports the following Unix compilers/operating systems:

compiler	operating system
SGI Mips(Pro) `CC` 7.2	IRIX 6.2
SUNPRO `CC` 4.2	Solaris 2.5
GNU `g++` 2.8.0 / 2.8.1	IRIX 6.2 / SUN Solaris 2.5 / Linux 2.0
CYGNUS `egcs` 1.0.1 / 1.0.2	IRIX 5.3 / IRIX 6.2 / Solaris 2.5 / Linux 2.0

If your compiler/operating system is not on this list, this not necessarily implies that CGAL does not work on your system[3], it just means we did not test it so far. For some compilers we did the tests, but not all parts of CGAL work with them. This includes SGI Mips(Pro) `CC` 4.0, SUNPRO `CC` 4.1 and GNU `g++` 2.7.2. An overview of known problems is given at the following URL:

`http://www.cs.uu.nl/CGAL/Compilers/`

N.B. It is highly recommended to read this information, since some of these older compilers may cause a lot of problems! Especially the support for templates is sometimes very poor.

At the same location you can find some hints about CGAL support for PC compilers (BORLAND `C++` 5.02 and MICROSOFT Visual `C++` 5.0), but this is in a premature state at the moment.

In case you want to use LEDA together with CGAL (needed e.g. for number types supporting exact computation and for the graphical demo programs), you should have a look at section 12 now.

3 Getting CGAL

The CGAL library can be downloaded in two different ways: using ftp or using WWW. If you have a WWW connection, the easiest way to download CGAL is via the CGAL homepage:

`http://www.cs.uu.nl/CGAL`

[1] see `http://www.mpi-sb.mpg.de/LEDA/index.html` for information.
[2] see e.g. `http://reality.sgi.com/austern/std-c++/faq.html#PartB` for information
[3] e.g. we would expect CYGNUS `egcs` 1.0.* to work on IRIX 6.4 as well

and go to the 'Software' section.

Just follow the instructions on this page to obtain your copy of the library. The CGAL library can also be downloaded using FTP. The library can be found at the following location:

```
ftp.cs.uu.nl
```

in the directory `/pub/CGAL`. This directory contains release 1.0 of the CGAL library. There is also a `README` file that contains descriptions of the files in this directory. An example of an FTP-session is given below.

```
$ ftp ftp.cs.uu.nl
Name (ftp.cs.uu.nl:<your username>): anonymous
Password: <type your email address here>
ftp> cd pub/CGAL
ftp> get README
ftp> binary
ftp> get CGAL-1.0.tar.gz
ftp> quit
```

After you have downloaded the file containing the CGAL library, you have to decompress it. For the zipfile use the command

```
unzip <filename>.zip
```

and for the gzipped file use the commands

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

N.B. On a PC you should use an unzip utility that can deal with long filenames (like WinZip or InfoZip)!

In both cases the directory `CGAL-1.0` 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
`doc_html`	documentation (HTML)
`doc_ps`	documentation (Postscript)
`doc_pdf`	documentation (PDF)
`examples`	example programs
`include`	header files
`lib`	(shared) object libraries
`make`	files with platform dependent makefile settings
`src`	source files

4 Installing CGAL

The directory `CGAL-1.0` 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 don't 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.

Warning. If you have an old compiler like GNU `g++` 2.7.2, or if you want to use LEDA in combination with CGAL, there are some compatibility issues that need to be addressed first. See section 12 and `http://www.cs.uu.nl/CGAL/Compilers/` for more information about this.

5 A Sample Installation

In this section we sketch an example installation on a SUN running Solaris 2.5 with the SUNPRO `CC` 4.2 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-1.0` 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[4], but see some dots written to the screen indicating progress.

```
-------------------------------------------------------
  This is the install script for CGAL 1.0
-------------------------------------------------------

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

  Choosing compiler SUNPRO 4.2.
  Testing for builtin STL ... no.
.
```

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 and it is checked whether it has a 'built-in' STL, i.e. if STL works without adding any specific compiler flags. Here the chosen compiler is SUNPRO `CC` 4.2 that has no built-in STL. [5] If there is more than one compiler installed on your system (and supported by CGAL), you may later choose to use a different compiler from the compiler menu.

[4] If you are that curious what happens exactly, have a look at the file `CGAL-1.0/install.log`.

[5] If your compiler has a built-in STL, you will have to wait for a moment while the script tests for a number of language features, which can only be done with STL.

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

```
*************************************************************
**              CGAL 1.0 Installation Main Menu            **
**              -------------------------------            **
**                                                         **
**   OS:                  sparc_SunOS-5.5                  **
**   Compiler:            SUNPRO CC 4.2                    **
**   STL_DIR:             please configure!!!              **
**   LEDA:                not supported.                   **
**   GMP:                 not supported.                   **
**                                                         **
**   The setup has not been tested.                        **
**                                                         **
**   There are no libs for this os/compiler.               **
**                                                         **
**   <C>  Compiler Menu                                    **
**   <L>  LEDA Menu                                        **
**   <G>  GMP 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, are LEDA and GMP supported etc., in this case it also tells you that STL still has to be configured.

Moreover you can see that the current setup has not been tested yet and that there do not exist CGAL libraries for this OS/compiler combination in the CGAL lib directory by now. It is always a good idea to test your configuration before you start building the CGAL libraries, but before that we first have to setup STL.[6]

5.2 Setting up STL

Please type "`c`" to go to the compiler menu where all compiler specific options can be configured. Then type "`i`" and you will be prompted to enter the include directory where STL header files reside on your system. The script only accepts directories that exist and pass a confidence test.[7] Let's assume, you have an HP STL on your system in `/pub/local/STL/HP`.

After you set up the STL directory, the script tries to compile some small example programs to check whether STL works principally and if it does, which language features are supported by your compiler and your STL version. This is necessary, since there are many different STL implementations and CGAL (partly) depends on which implementation is used. A message `ok` always indicates that your compiler works as it should i.e. a feature is supported resp. a bug is *not* present. On the other hand `no` resp. `unfortunately` indicate a lack of support resp. bug.

[6] This is not necessary for compilers with built-in STL. So you might want to skip the next section and go directly to section 5.3.

[7] In this case it checks for a file `iterator.h` in the directory. This file should be present in all STL implementations.

```
************************************************************
**            CGAL 1.0 Installation Compiler Menu         **
**            ----------------------------------          **
**                                                        **
**   OS:                  sparc_SunOS-5.5                 **
**   Compiler:            SUNPRO CC 4.2                   **
**   STL_DIR:             please configure!!!             **
**   LEDA:                not supported.                  **
**   GMP:                 not supported.                  **
**                                                        **
**   The setup has not been tested.                       **
**                                                        **
**   <C>  Choose compiler                                 **
**   <I>  STL  include directory                          **
**        <undefined>                                     **
**   <F>  Set custom compiler flags                       **
**       <none>                                           **
**   <L>  Set custom linker flags                         **
**       <none>                                           **
**   <T>  Test (and save) setup                           **
**                                                        **
**   <Q>  Back to Main Menu                               **
**                                                        **
**   New STL_DIR: /pub/local/STL/HP                       **
**                                                        **
************************************************************

Testing for STL ... ok.
Testing for SGI STL 3.0 ... no.
Testing for SGI 6/97 STL ... no.
Testing for SGI 1996 STL ... no.
Testing for SGI CC STL ... no.
Testing for HP STL ... ok.
...
<tests for several compiler features>
...
Saving current setup ... done.
```

If the STL test succeeds, the current setup is marked as tested and the settings are saved to a file in the directory `CGAL-1.0/config/install`. Thus, if you run the install script a second time for this OS/compiler, you won't have to enter the STL directory again, but it is retrieved from the corresponding config file instead.

5.3 Building the CGAL libraries

Now we are ready to build the CGAL libraries. First go back to the main menu with "`q`" and then 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 (if not, you should have a look at the error messages from compiler or linker).

```
************************************************************
**                                                        **
**                    Compiling CGAL 1.0                  **
**                    ------------------                  **
**                                                        **
************************************************************

OS:           sparc_SunOS-5.5
COMPILER:     SUNPRO CC 4.2
STL:          CGAL_STL_HP
LEDA:         not supported
GMP:          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 LEDA or GMP support (go to the LEDA resp. GMP menu) or simply quit the install script by typing "`q`". When leaving the script, you get a list of successful builds during the session.

Now it would be a good idea to print and read the 'Getting Started with CGAL' document that can be found in the `doc_ps` directory.

6 The interactive mode

To run the install script in the interactive mode, go to the `CGAL-1.0` directory and enter the command

```
./install_cgal -i
```

First, certain utility programs are located and your system is searched for compilers supported by CGAL. After that, 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 setup STL. The compiler menu is described in section 6.2.

If you want to use LEDA or GNU GMP with CGAL, you will have to go to the *leda menu* described in section 6.3 resp. to the *gmp menu* described in section 6.4.

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 LEDA test and a GMP test, but not all tests are performed always; e.g. the LEDA test is only done, if you enabled LEDA 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 LEDA or GMP installation in system directories which otherwise is only done the first time you enable LEDA/GMP 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-1.0/config/install/`*<CGAL-OS-description>*

where *<CGAL-OS-description>* identifies your OS/compiler combination in a way specified in section 9[8]. 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 terminates. Here follows a list of these files (all are plain ASCII and reside in `CGAL-1.0`).

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

[8] 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 setup the compiler specific options, such as the compiler to use (if more than one has been detected) and the location of the STL files.

Some compilers come with their own STL adaptation to which we refer as *built-in* and some compilers are not shipped with STL. In the latter case you have to supply an `STL_DIR`, i.e. the path to a directory where the STL header files are stored.[9] In this case we speak of an external STL. Even if your compiler has built-in STL, you can still choose to use an external STL by setting `STL_DIR` appropriately.

Compiler Menu

<**C**> Choose the compiler to be used from the list of detected compilers.

<**S**> (*present if a built-in* STL *was detected for the current compiler)* Determine, if compiler built-in or an external STL is used.

<**I**> (*present if an external* STL *is used)* Set the include directory for an external STL.

<**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 LEDA Menu

This is the place to setup LEDA specific options, if you plan to use LEDA together with CGAL (see also section 12). 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.

6.4 The GMP Menu

This menu is to setup GMP (GNU Muptiple Precision Library) specific options, if you plan to use GMP together with CGAL. (In the `auxiliary` directory you can find a GMP distribution.) This menu

[9] See section 13 on where you can obtain an STL implementation and what problems you might have to consider.

contains an option to install GMP in your CGAL directory tree, but of course you can also install it independently from CGAL.

If GMP support is enabled 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.

GMP Menu

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

<**G**> Enable/Disable GMP 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.

7 The non-interactive mode

To run the install script in the non-interactive mode, go to the `CGAL-1.0` 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 -64"`. 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:

- your compiler does not have a 'built-in' STL (section 7.1),
- you want to use LEDA together with CGAL (section 7.2) *or*
- you want to use GNU GMP together with CGAL (section 7.3).

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, displaying the settings you gave via command line and classifying (see also section 13) the STL version used (which – of course – won't work, if you didn't set up STL correctly, see section 7.1 below). 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 i.e. a feature is supported resp. a bug is *not* present. On the other hand `no` resp. `unfortunately` indicate a lack of support resp. 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, a backup is made, 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.

When this is finished it would be a good idea to print and read the 'Getting Started with CGAL' document that can be found in the `doc_ps` directory.

7.1 Setting up STL

The install script and the makefiles use the variable `STL_DIR` to indicate the STL that shall be used with CGAL. This variable should point to the directory where STL header files are stored. It can be set from command line with option "`--STL_DIR` <*dir*>". If you do not set `STL_DIR` this way, it is assumed that the compiler has 'built-in' STL and this is used. If your compiler does not have built-in STL, you *have to* supply an `STL_DIR` to get things running.

7.2 Setting up LEDA support

See also section 12. 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*>").

7.3 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*>").

7.4 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 `cgal_install` 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-2.8.1/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 Troubleshooting

In case you run into problems with CGAL, you should first have a look at the CGAL homepage:

`http://www.cs.uu.nl/CGAL`

If you can't find a solution to your problem there, please send email to `cgal@cs.uu.nl`.

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-1.0/lib`) (and analogously its own include makefile in `CGAL-1.0/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. This distinction is necessary, because the object libraries look (at least possibly) different.

We call the resulting string CGAL-OS description.

Examples are `mips_IRIX-6.2_CC-7.2` or `sparc_SunOS-5.5_g++-2.8.1_LEDA`. You can use the install script to get your CGAL-OS description, see section 7.4.

10 The CGAL makefile structure

The CGAL distribution contains the following makefiles:

- `CGAL-1.0/src/makefile_lib` for compiling the CGAL object library `libCGAL.a`,
- `CGAL-1.0/src/makefile_sharedlib` for compiling the CGAL shared object library `libCGAL.so`,
- `CGAL-1.0/src/makefile_geomview` for compiling a library for geomview support,
- `CGAL-1.0/examples/makefile` as sample makefile *and*
- `CGAL-1.0/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-1.0/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_IRIX-6.2_CC-7.2 contains makefile settings for the IRIX 6.2 operating system and the SGI Mips(Pro) CC 7.2 compiler. These include files are automatically generated by the install_cgal script and they are all located in the CGAL-1.0/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 7.4). E.g. if your compiler is g++, you would type

```
CGAL_MAKEFILE=‘<insert your CGAL-1.0 dir>/install_cgal -os g++‘
```

in bourne shell resp.

```
setenv CGAL_MAKEFILE ‘<insert your CGAL-1.0 dir>/install_cgal -os g++‘
```

in csh derivatives.

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-1.0/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-1.0/make directory. After the installation of CGAL this sample makefile is ready for use. Just type 'make example' to compile the program example.C. You may use this makefile as a blueprint for your own makefiles.

Furthermore the directories CGAL-1.0/examples and CGAL-1.0/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 Using CGAL and LEDA

This section describes how to use CGAL and LEDA simultaneously.

12.1 Support for LEDA

CGAL supports LEDA in the following ways:

1. There are support functions defined for the LEDA number types leda_big_float, leda_integer, leda_rational and leda_real (see the files in the include directory <CGAL/leda_*>).

2. CGAL defines the following LEDA-related compiler flags:

   - When LEDA is used, the flags `CGAL_USE_LEDA` and `LEDA_PREFIX` will be set.
   - When LEDA is used, the LEDA memory management will be used for geometric primitives in CGAL. This can be turned of by setting the flag `CGAL_NO_LEDA_HANDLE`. In that case no memory management will be used (see `<CGAL/Handle.h>`).

The include makefiles in the `CGAL-1.0/make` directory corresponding to LEDA can be recognized by the suffix "`_LEDA`".

12.2 LEDA and STL conflicts

When you are using an old compiler or an old version of STL, the combination of LEDA and STL (which is required for CGAL) may give some problems.

12.2.1 Definition of type bool

If a compiler doesn't support the keyword bool, libraries like LEDA and STL provide their own definition of bool. Unfortunately the LEDA definition of bool is incompatible with the definition in most STL implementations. One of them needs to be changed. The easiest solution is to modify STL as follows:

1. locate the definition of bool; most of the time it is defined as

   ```
   typedef int bool;
   ```

   in a file called bool.h or stl_config.h

2. replace this definition by the following:

   ```
   #ifdef CGAL_USE_LEDA
   #include <LEDA/bool.h>
   #else
      typedef int bool;
   #endif
   ```

The flag `CGAL_USE_LEDA` ensures that you can continue to use STL without depending on LEDA.

REMARK 1) If you don't like to modify centrally managed STL files, it is possible to fake it. This can be done by making changes to a copy of the STL file bool.h or stl_config.h. After this you have to make sure that this copy is found earlier on the include path than the original file. It is also possible to modify LEDA instead of STL. However, then you have to modify and recompile (if you have the source code) LEDA.

REMARK 2) If you are using LEDA 3.5 (and not 3.4 or 3.6), the LEDA definition of bool can be disabled by defining the flag `LEDA_BOOL_H`. So if you add `-DLEDA_BOOL_H` to the custom compiler flags during the installation of CGAL, you can use LEDA and STL simultaneously without changing either of them.

12.2.2 'red' and 'black' conflict

When compiling the graphical demo programs that use LEDA windows with LEDA versions prior to 3.6.1, you may encounter a name clash. This is because LEDA defines the colors 'red' and 'black' in the file `<LEDA/impl/x_window.h>`, whereas some STL implementations use 'red' and 'black' in the implementation of trees. One of the libraries needs to be modified. In this case the easiest solution seems to prefix the LEDA color constants with something like `leda_`.

13 The Standard Template Library

For using the CGAL library, the Standard Template Library is required. Recent compilers provide their own implementation of STL. There are also free implementations. Most notable are the original Hewlett Packard implementation (`http://www.cs.rpi.edu/~musser/stl.html`), the implementation from Silicon Graphics (`http://www.sgi.com/Technology/STL/`, also contains documentation and links to other STL resources) and a port of this SGI version to a lot of other compilers (`http://www.metabyte.com/~fbp/stl/`).

The `install_cgal` script tries to figure out which STL is used. This is necessary for the circulator package, and also to work around some bugs. In case the right version is not recognized, you could try to use the non-interactive mode of the `install_cgal` script (see section 7). The compiler flag `CGAL_STL_VERSION` can take the following values:

CGAL_STL_GCC	STL version that comes with gcc-2.7.2
CGAL_STL_HP	Free HP version
CGAL_STL_SGI_3_0	SGI STL 3.0
CGAL_STL_SGI_WWW	Free new SGI implementation
CGAL_STL_SGI_WWW_OLD	Free SGI implementation for older compilers
CGAL_STL_SGI_CC	SGI implementation that comes with their 7.0/7.1 compilers
CGAL_STL_UNKNOWN	unknown STL version (this will probably not work)

Most STL libraries consist of only header files. An exception is HP's STL, which also has source files `random.cpp` and `tempbuf.cpp`. The CGAL makefiles do *not* take care of this. You have to compile these files yourself and add them to the custom linker flags during the installation of CGAL, or enter them manually in the makefiles.

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_BUILTIN_BOOL` denotes that the compiler doesn't know the keyword bool. If this flag is defined, the type bool will be borrowed from the STL library (currently by including the file `<pair.h>`).

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. For this the test programs in the directory `CGAL-1.0/config/testfiles` are used. The file `<CGAL/compiler_config.h>` and the test programs contain a description of the problem, so in case of trouble with a `CGAL_CFG` flag it is a good idea to take a look at it.

The file `<CGAL/config.h>` manages all configuration problems of the compiler. This file includes the file `CGAL/compiler_config.h`. It is therefore important that the file `<CGAL/config.h>` is always included before any other CGAL source file that depends on workaround flags. In most cases you don't have to do anything special for this, because many CGAL files already take care of including `<CGAL/config.h>`. Nevertheless it would be a good idea to always start your CGAL programs with including `<CGAL/config.h>` (or `<CGAL/basic.h>`, which contains some more basic CGAL definitions).