Installation Guide

Release 2.1, January 2000

Authors

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

Acknowledgement

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

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] Multiple Precision library GMP[2] and/or the Class Library for Numbers (CLN)[3].

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 and MS Windows are supported. The reason is that most compilers do not conform to the ISO 14882 standard for C++[4] and some of them have so many limitations/bugs that we could not work around all of them.

On MS Windows, one has two options for installing: one is specifically targeted for this OS and is described in section 13[5]; the other is using Cygwin[6] and the generic installation procedure for Unix-like environments described in section 5 and subsequent sections. If you are going to install CGAL using Cygwin, please read section 12 first.

More precisely, `CGAL-2.1` supports the following compilers/operating systems:

compiler	**operating system**
SGI Mips(Pro) `CC` 7.3 (n32)	IRIX 6.5
GNU `g++` 2.95 (and later)	IRIX 6.5 / Solaris 2.6 / Linux 2.x / MS Windows 95/98/NT4[7]
`Egcs` 1.1.2 (and later)	IRIX 6.5 / Solaris 2.6 / Linux 2.x / MS Windows 95/98/NT4[7]
MS Visual `C++` 6.0	MS Windows 95/98/NT4

There are plans to provide full support for BORLAND `C++` 5.4 in the future, but there is no such support at the moment. MS Visual `C++` 5.0 and prior is not supported and there are no plans to support it in future releases. The SUNPRO `CC` 4.2 compiler is not supported anymore, please stay with CGAL-1.2 if you have to use it. GNU `g++` 2.8.1 is not supported anymore, please stay with CGAL-2.0 if you have to use it. Support for SUNPRO `CC` 5 will be reconsidered as soon as an acceptable degree of standard-conformance is reached. Support for INTEL `C++` 4.0 will be considered.

3 Getting CGAL

This section applies to Unix-like environments, as well as to MS Windows.

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] `http://www.mpi-sb.mpg.de/LEDA`
[2] `http://www.gnu.org/software/gmp`
[3] `http://clisp.cons.org/~haible/packages-cln.html`
[4] see e.g. `http://reality.sgi.com/austern/std-c++/faq.html#PartB` for information
[5] see also the file `wininstall.txt` in the root directory of the installation
[6] `http://sourceware.cygnus.com/cygwin`
[7] with Cygwin β20.1

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 2.1 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-2.1.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[7].

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

and for the gzipped file 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. On MS Windows you should use an unzip utility that can deal with long filenames (like WinZip or InfoZip).

In both cases the directory `CGAL-2.1` 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
`stlport`	a customized version of `Stlport-3.2.1`[8] that is used with MS Visual C++ 6.0

In order to avoid problems with installation routines, please make sure that the full path to `CGAL-2.1` does not contain spaces (the latter, while in principle possible on MS Windows, in not supported in the present release).

It is planned that in the future on MS Windows the installation procedure will be automated via InstallShield.

[7] `unzip` for MS Windows can be downloaded, for instance, from `http://www.itribe.net/virtunix/`

[8] `http://www.stlport.org/`

4 Installing CGAL

The directory `CGAL-2.1` 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.

5 A Sample Installation

In this section we sketch an example installation on a SUN running Solaris 2.6 with the GNU `g++` 2.95 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-2.1` 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[8], but see some dots written to the screen indicating progress.

```
-------------------------------------------------------
  This is the install script for CGAL 2.1
-------------------------------------------------------

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

  Choosing compiler GNU gcc-2.95
  Testing for STL ... ok.
  ...
  <tests for several compiler features>
  ...
  Saving current setup ... done.
.
```

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 a number of tests are done to check whether your compiler supports certain language constructs respectively has specific bugs. There are quite a number of these tests, so this step may take some time. For each test you should get a message what particularly is tested at the moment and what the result is. 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.

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

```
  ****************************************************************
  **             CGAL 2.1 Installation Main Menu              **
  **             -------------------------------              **
  **                                                          **
  **   OS:                 sparc_SunOS-5.6                    **
  **   Compiler:           GNU gcc-2.95                       **
  **   LEDA:               not supported.                     **
  **   GMP:                not supported.                     **
```

[8] If you are that curious what happens exactly, have a look at the file `CGAL-2.1/install.log`.

```
**   CLN:                 not supported.                     **
**                                                           **
**   Compiler is supported by CGAL.                          **
**   The setup has been tested ok.                           **
**                                                           **
**   There are no libs for this os/compiler.                 **
**                                                           **
**   <C>  Compiler Menu                                      **
**   <L>  LEDA Menu                                          **
**   <G>  GMP Menu                                           **
**   <M>  CLN 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, GMP and CLN supported etc. Moreover you can see that the current setup has been tested and that there do not exist CGAL libraries for this OS/compiler combination in the CGAL lib directory by now.

The fact that your setup has been tested implies that the current settings have been saved to a file in the directory `CGAL-2.1/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.

5.2 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 (if not, you should have a look at the error messages from compiler or linker).

```
***************************************************************
**                                                           **
**                  Compiling CGAL 2.1                       **
**                  ------------------                       **
**                                                           **
***************************************************************

OS:           sparc_SunOS-5.6
COMPILER:     GNU gcc-2.95
LEDA:         not supported
GMP:          not supported
CLN:          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, GMP or CLN support (go to the LEDA, GMP, CLN menu resp.) 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 document "Getting Started with CGAL" that can be found in various formats in the `doc_html`, `doc_pdf` and `doc_ps` directories.

6 The interactive mode

To run the install script in the interactive mode, go to the `CGAL-2.1` 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 LEDA, GMP or CLN with CGAL, you will have to go to the *leda menu*, *gmp menu* or *cln menu*. These are described in section 6.3, 6.4, 6.5, 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 LEDA test, a GMP test and a CLN 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/GMP/CLN installation in system directories which otherwise is only done the first time you enable LEDA/GMP/CLN 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-2.1/config/install/<`*`CGAL-OS-description`*`>`

where <*CGAL-OS-description*> identifies your OS/compiler combination in a way specified in section 9. [9] 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-2.1`) 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

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.

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

Compiler Menu

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

<**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 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.

6.4 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 the `auxiliary` directory you can find a GMP distribution, if you do not already have it installed on your system. This menu contains an option to install GMP in your CGAL directory tree[10], 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.

[10]This option is on MS Visual `C++` just unpacks the corresponding pre-compiled library that comes with CGAL distribution.

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.

6.5 The CLN Menu

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

```
http://clisp.cons.org/~haible/packages-cln.html
```

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

CLN Menu

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

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

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

<**M**> (*present if CLN support is enabled, there is a CLN installation in system directories and* `CLN_INCL_DIR` *or* `CLN_LIB_DIR` *have been set)* Use CLN installation from system directories.

7 The non-interactive mode

To run the install script in the non-interactive mode, go to the `CGAL-2.1` 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 LEDA together with CGAL (section 7.1),
- you want to use GNU GMP together with CGAL (section 7.2) *or*
- you want to use CLN 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 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 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, 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.

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

7.1 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*>").

7.2 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*>").

7.3 Setting up support for CLN

By default there is no support for CLN, but you can change this easily by use of the command line option "`-cln`". If CLN 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 CLN header files ("`--CLN_INCL_DIR` <*dir*>") and the CLN library ("`--CLN_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-2.95/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 have CGAL 1.*/2.0 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-2.0`). You can then build all libraries for the actual operating system that existed in your CGAL 1.*/2.0 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.

Note that some compilers that have been supported in previous CGAL releases might not be supported in CGAL-2.1 anymore, see section 2. Trying to build CGAL-2.1 with these compilers will most probably fail. You can solve this problem by deleting the obsolete config files (see section 6.1) from `CGAL-2.1/config/install` before issuing the `rebuild-all` command.

Similarly you might want to use compilers with CGAL-2.1 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-2.1/lib`) (and analogously its own include makefile in `CGAL-2.1/make`) named as determined by the following scheme.

$$<arch>\_<os>-<os\text{-}version>\_<comp>-<comp\text{-}version>[\_\text{LEDA}]$$

<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_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-2.1/src/makefile_lib` for compiling the CGAL object library `libCGAL.a`,
- `CGAL-2.1/src/makefile_sharedlib` for compiling the CGAL shared object library `libCGAL.so`,
- `CGAL-2.1/src/makefile_geomview` for compiling a library for geomview support,
- `CGAL-2.1/examples/makefile` as sample makefile *and*
- `CGAL-2.1/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-2.1/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-2.1/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 7.4). E.g. if your compiler is `g++`, you would type

```
CGAL_MAKEFILE=‘<insert your CGAL-2.1 dir>/install_cgal -os g++‘
```

in bourne shell resp.

```
setenv CGAL_MAKEFILE ‘<insert your CGAL-2.1 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-2.1/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-2.1/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-2.1/examples and CGAL-2.1/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 β-20.1.

It consists of a port of a large number of GNU tools, such as bash, make, gcc (egcs), 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://sourceware.cygnus.com/cygwin/ .

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[11] then the path D:\Mystuff\Mydir\LEDA translates to /ddrive/Mystuff/Mydir/LEDA.

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

Links, shortcuts, etc should be avoided as well.

12.2 MS Visual C++ 6.0-setup

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

PATH should contain MS Visual C++ 6.0 command line tools locations. The environment variables INCLUDE and LIB should point to the location of MS Visual C++ 6.0 header files and to the location of the MS Visual C++ 6.0 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[12]. 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.

[11] by typing mount D: /ddrive

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

```
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 `cygnus.bat`, 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.

13 Installation on MS Windows with native tools

CGAL-2.1 provides a basic support for MS Visual `C++` 6.0 compiler. It can also be considered an early beta release for Inprise Borland C++ Builder 4, known also as BORLAND `C++` 5.4. Using the latter with CGAL is briefly described in file `wininstall.txt` located in the directory CGAL-2.1.

Downloading and contents of the distribution are described in section 3 above.

In order to install and use CGAL with MS Visual `C++` , you need MS Visual `C++` command line tools installed and working[13]. Certain familiarity with using Windows command prompt and these tools is assumed. A reasonable amount of system resources, that is enough memory to run large compilations (64 Mb on NT4.0) and about 15MB of disk space, is required. Note, however, that executables, object files and debug-related files in subdirectories `demo/` and `examples/` can occupy several hunderd megabytes all together.

It is highly recommended that you have a recent (at least 4.0) version of LEDA installed. Most of the CGAL demos use LEDA for visualization.

CGAL has interfaces to a number of rather important 3rd party packages, that provide, in particular, arbitrary precision arithmetic, and visualization. On Windows these packages are LEDA and GNU MP (also known as GMP). GMP is supported by default; see section 13.4 for details. LEDA configuration is described below.

In what follows `CGALROOT` will refer to the full path to the directory CGAL-2.1, where the downloaded source is unpacked.

13.1 Initial customization

Change to `CGALROOT` and run the batch file `cgal_config.bat` at the command prompt. This script will set the appropriate makefiles for subsequent compilation.[14]

`cgal_config.bat` accepts 4 parameters.

1. the compiler type (`msc` or `bcc`)

[13]To make sure that certain environment variables are set correctly, one should execute `VCVARS32.BAT` located in the same directory as the MS Visual `C++` executables. A possible MS Windows-9* - specific problem (and a solution) is discussed in section 13.1

[14]**MS Windows-9*.** The memory for environment variables might have to be increased. Select the MS-DOS 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. Then press *OK* and kill the window.

2. (optional) compiler options (to enable debugging and/or threads support)
3. (optional) `LEDAROOT` (the location where LEDA lib files are installed)
4. (optional) the location where LEDA header files are installed. If omitted, it is assumed to be `LEDAROOT\incl`.

Typically, assuming that LEDA is installed in `k:\LEDA-4.0`, one would run at the command prompt:

```
K:\cgal\> cgal_config msc k:\LEDA-4.0
```

Otherwise, if LEDA header files are not in `LEDAROOT\incl`, one would run, say,

```
K:\cgal\> cgal_config msc k:\LEDA-4.0\msvc k:\LEDA-4.0\incl
```

It is assumed that paths to `LEDAROOT` and LEDA include directories do not have names with spaces.

Running `cgal_config.bat` without parameters prints out the detailed options list (in particular, the compiler and linker options).

13.2 Building the library, examples and demos

Change (if necessary) to `CGALROOT` and run `make_lib.bat` at the command prompt. Namely,

```
K:\cgal\> make_lib
```

builds the library with the compiler and linker options specified by the last run of `cgal_config.bat`. For other settings, one would have to reconfigure and recompile (same as for current LEDA installation).

```
K:\cgal\> make_examples
```

compiles and links the examples and

```
K:\cgal\> make_demos
```

compiles and links the demos.

Examples and demos can also be built on per directory, or even on per executable basis. Each `examples/` and `demo/` subdirectory contains a `makefile.mak` file that is a MS Windows `nmake` makefile. For instance, change to CGAL`-2.1\demo\Point_set_2` and type `nmake -f makefile.mak all` to build all the examples there. Typing `nmake -f makefile.mak ps_test1` will build the particular demo program `ps_test1`.

The corresponding executable(s) then can be run as usual, from the command prompt, or via Explorer.

Note that demos requiring LEDA would not work if CGAL was configured without LEDA support. LEDA must be compiled and used with `LEDA_STD_HEADERS` flag on. See section D.2 for details.

13.3 Developing for CGAL with MS Visual C++ 6.0

Here we give some hints on the use of CGAL with native MS Visual `C++` tools; C++ language issues, troubleshooting and the use of CGAL with Cygwin are covered above.

The most important issue is the following. CGAL uses C++ STL (Standard Template Library) quite extensively. To resolve many C++ Standard compliance problems of the native MS Visual `C++` STL, we chose to use a drop-in replacement for it, `STLPort` (see `http://www.stlport.org/`). We have customized `STLPort-3.2.1` to partially support `std::iterator_traits` template of STL.

For this to work, the compiler should have `STLPort` subdirectory first in its search path. This issue is taken care of in the makefiles generated. Customization of IDE is described in Sect 4.2 below.

Using nmake makefiles. Simply adapt a makefile `makefile.mak` from an `examples/` or `demos/` subdirectory. This means changing the target `all` and changing the targets that create executables (look in the subsection “target entries” there). For instance, you would like to create an executable `demo_beta.exe` from a C++ source file `demo_beta.C`. Then your `makefile`, that would include the “main” makefile makefile.mak in `CGALROOT` would have

```
all: demo_beta

demo_beta: demo_beta.obj
        $(CGAL_CXX) $(LIBPATH) $(EXE_OPT)demo_beta demo_beta.obj \
        $(LDFLAGS)
```

and a “suffix rule” that describes creation of an object file from a .C - file, e.g.

```
.C.obj:
        $(CGAL_CXX) $(CXXFLAGS) $(OBJ_OPT) $<
```

Extra compiler and linker options are to be specified by changing macros `CXXFLAGS` and `LDFLAGS`, respectively.

One way to specify the “main” makefile is by using environment variable `CGAL_MAKEFILE` to specify the full path to `makefile.mak` in `CGALROOT`.

Probably such a makefile can also be used for a “custom build” via an IDE, although we never attempted this ourselves.

Using MS Visual `C++` IDE. The compiler/linker options used to build the CGAL library should match those used for the project. This can be done by selecting *Project/Settings...*, the *C/C++* section, *Code Generation* category, and choosing the right library in *Use run-time library*:

Single-Threaded	ml (or no option)
Multi-threaded	mt
Multi-threaded DLL	md
Debug Single-Threaded	mld
Debug Multi-threaded	mtd
Debug Multi-threaded DLL	mdd

Further necessary customization is as follows. We hope that the forthcoming automatic installation will simplify these elaborated procedures.

Customizing compilation. Additional include directories (in the right order!) and preprocessor definitions have to be specified. We assume that either `CGAL_ROOT` is an (already defined) environment variable (and then the include and library paths can be entered as shown), or `$(CGAL_ROOT)` must be replaced by the corresponding full path to the CGAL installation.

Select *Project/Settings...*, the *C/C++* section, *Preprocessor* category. Add (in the following order)

```
$(CGAL_ROOT)\stlport
$(CGAL_ROOT)\include\CGAL\config\msvc
$(CGAL_ROOT)\include
$(CGAL_ROOT)\include
$(CGAL_ROOT)\auxiliary\wingmp\gmp-2.0.2
```

to ”Additional include directories:”. It is very important that `$(CGAL_ROOT)\stlport` appears first in the list in *Additional include directories*. Otherwise a bunch of ugly compiler error messages is guaranteed.

If LEDA is used, don’t forget to add `$(LEDAROOT)\incl` to *Additional include directories*.

Add `CGAL_USE_GMP` to *Preprocessor definitions*.
If LEDA is used, also add `CGAL_USE_LEDA` and `LEDA_PREFIX` there.

Should your source file(s) have suffices that are not recognized by MS Visual `C++` 6.0 IDE as C++ source files, (like all the CGAL source files, by the way) add `/TP` to options *Project Settings* in the *C/C++* section, *Preprocessor* category.

Customizing linking. Additional library directories have to be specified. Select *Project/Settings...*, *Link* section, *Input* category. Add `CGAL.lib` and `gmp.lib` in *Object/library modules*: list. If LEDA is used, also add there `libP.lib`, `libG.lib`, and `libL.lib`.

If LEDA Window is used, also add there `libW.lib`. In addition, make sure that you link against the system libraries `user32.lib, gdi32.lib, comdlg32.lib, shell32.lib, advapi32.lib` (usually IDE has them already on the list).

If LEDA Geowin is used, one should as well add `libGeoW.lib` and `libD3.lib` to *Object/library modules*.

13.4 GMP support

A pre-compiled for MS Visual `C++` 6.0, using Cygwin GNU C compiler and GNU assembler, static object library `gmp.lib` is located in `CGALROOT\auxiliary\wingmp\gmp-2.0.2\msvc` and the corresponding header file `gmp.h` in `CGALROOT\auxiliary\wingmp\gmp-2.0.2`.

It was created using M.Khan's Cygwin-specific patches to GMP-2.0.2, that are available from `http://www.xraylith.wisc.edu/~khan/software/gnu-win32/` and then slightly modified (by adding a few routines, like `random`, and creating a separate MS Visual `C++` 6.0-compiled library that does I/O of the GMP numbers.) It was checked that all tests that come with GMP distribution pass. For details see `CGALROOT\auxiliary\wingmp\gmp-2.0.2\msvc\src`.

Subject to time constraints, we might provide a smoother GMP support for MS Visual `C++` in the future. A new major release of GMP is expected in March 2000, probably CGAL will be upgraded to use it then.

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, the flags `CGAL_USE_LEDA` and `LEDA_PREFIX` will be set.
    - When LEDA is used, the LEDA memory management (LEDA handles) will be used for geometric primitives in CGAL. This can be turned of by setting the flag `CGAL_NO_LEDA_HANDLE`. In that case CGAL memory management will be used (see `<CGAL/Handle.h>`).

The include makefiles in the `CGAL-2.1/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_MUTABLE` denotes that the compiler does not know the keyword `mutable`.

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 `\cgaldir/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 do not 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).

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.cs.uu.nl/CGAL`

or send an email to `cgal@cs.uu.nl`.

D.1 The "Long-Name-Problem" (Solaris only)

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[15] 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/egcs` you would add

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

```
-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`.

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.

```
#---------------------------------------------------------------------#
#                    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.2 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[16], since this is what we test CGAL with.

MS Visual C++ -specific problems. Using MS Visual C++ with LEDA requires[17] the latter to be compiled and used with `LEDA_STD_HEADERS` flag on. CGAL uses new-style C++ standard conformant headers[18], while LEDA can work with both styles. Mixing these styles is a strict no-no for MS Visual C++ . 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
```

Also, LEDA and CGAL libraries must be compiled with the same options controlling the use of debugging and multithreading. [19]

If a binary release of LEDA4.0 is used, make sure that it is one of them that uses new-style headers. Namely, among the self-extracting executables on `http://www.mpi-sb.mpg.de/LEDA/download/windows/` choose one of these that have the name ending with `-std.exe`.

D.3 MS Visual C++ 6.0-specific C++ problems

CGAL uses a modified drop-in replacement for STL, STLPort, version 3.2.1. Our modifications included

[16]At the moment this is LEDA 4.0.
[17]Also applies to BORLAND C++ .
[18]the ones that do not have `.h` suffix
[19]MS Visual C++ compilation/linking options `-ML, -MT, -MD, -MLD, -MTD, -MDD`

- modifying the code to enable the use of `std::iterator_traits` template, where possible;
- providing `std::vector` class that does not use a plain C pointer as `vector_iterator`.

The latter allows using the full power of STL algorithms on `std::vector` without the limitation given by the absense of the support for partial specialization of template parameters.

Always make sure that the subdirectory `$(CGAL_ROOT)/stlport/` comes first in the list of additional include directories for MS Visual `C++` . This is handled automatically if you use CGAL-supplied (or CGAL-generated) makefiles.

`std::iterator_traits` is only partially supported, due to absense 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[20]. 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 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++` 6.0 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. A scope operator in parameter types of a member function only works when the function is defined inline. See `config/testfiles/CGAL_CFG_NO_SCOPE_MEMBER_FUNCTION_PARAMETERS.C`.
5. Passing a dependant type as template parameter cannot be done with `typename` keyword on MS Visual `C++` . Replace `typename` by `CGAL_TYPENAME_MSVC_NULL` in such places.
   See `config/testfiles/CGAL_CFG_TYPENAME_BUG.C`.
6. Template `friend` declarations must not contain $<>$. Replace $<>$ by `CGAL_NULL_TMPL_ARGS`.
   See `config/testfiles/CGAL_CFG_NO_TEMPLATE_FRIEND_DISTINCTION`.
7. Using non-class template parameters often gives internal compiler errors.
8. 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".

[20]CGAL `Point, Segment`, etc.

9. `[...]/VC98/INCLUDE/xlocnum(268) : error C2587: '_U' : illegal use of local variable as default parameter` can occur[21]. The only workaround we know is to redefine the macro `_VIRTUAL` in `<xlocnum>`[22] to be empty. Search for `#define _VIRTUAL virtual` there and replace it by `#define _VIRTUAL` .

10. Various matching failures for overloaded functions and ctors. Use dummy parameters.

11. Avoid multiple forward declarations.

12. If necessary, simplify template parameters by using extra `typedef`s.

E Scripts

E.1 `create_cgal_makefile`

The bourne-shell script `create_cgal_makefile` is contained in the `CGAL-2.1/scripts` directory. It can be used to create makefiles for compiling CGAL applications. Executing `create_cgal_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.

E.2 `use_cgal_namespace`

The perl script `use_cgal_namespace` is contained in the `CGAL-2.1/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>
```

E.3 `replace_headers`

The perl script `redirect_headers` is contained in the `CGAL-2.1/scripts` directory. It can be used to replace oldstyle include directives for standard headers by their standard ISO counterparts, e.g.

```
#include <algo.h>
```

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

[22] Yes, in the MS Visual `C++` header! You need not edit the actual file though. Copy it to `CGAL-2.1/stlport/` and edit it there. As `CGAL-2.1/stlport/` is searched by the compiler ahead of the other directories, your updates will be used. DISCLAIMER: *We do not know if the actions described in this footnote are legal in your country. You are on your own here.*

is replaced by

```
#include <algorithm>
```

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 `.hrbck`.

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.hrbck -- replace_headers <FILES>
```