

Preface

What This Book Contains

This book is about how to use the CGAL *2D Arrangements* package to solve problems. It will teach you the functionality of this package and a few related CGAL packages. Every feature of the package is demonstrated by a small example program. Even the basic tools are sufficient to solve problems, as shown in Chapter 2, which includes the first *application*. We use the word application here to refer to a complete standalone program (written on top of CGAL arrangements) to solve a meaningful problem. The applications presented in the book include finding the minimum-area triangle defined by a set of points, planning the motion of a polygon translating amidst polygons in the plane, computing the offset polygon, constructing the farthest-point Voronoi diagram, coordinating the motion of two discs moving amidst obstacles in the plane, performing Boolean set operations on curved polygons, and more. The programs are designed such that you can use each of them as is, provided that the definition of your problem is the same as the definition of the problem the application program was set out to solve. If your problem is similar (but not exactly the same) it may require modification of the application program presented in the book. The book contains material that will help you in this modification process. As the book progresses, the applications become more involved and the range of problems solved by the applications widens. Moreover, the book was designed such that about halfway through, you will have accumulated sufficient knowledge to write your own application programs. You can download the code of the programs listed in the book along with appropriate input data-files from the book's website <http://acg.cs.tau.ac.il/cgal-arrangement-book>.

The book is about using CGAL arrangements; it is *not* about developing CGAL packages. The latter requires more in-depth knowledge of C++ and of generic programming than is required for reading this book, in addition to familiarity with the CGAL design rules and programming standards. If you are interested in developing basic CGAL code, you will find some of the necessary information on the CGAL website <http://www.cgal.org>. You will also find many pointers in the book to papers describing the design principles and implementation details of various related packages of CGAL. The CGAL code is open source. In particular, you can browse the entire code of the *2D Arrangements* package. In the book we barely discuss how the package was implemented.

One of the guiding principles in developing the CGAL *2D Arrangements* package is the separation of the combinatorial algorithms (e.g., plane sweep) on the one hand and the algebraic operations they use (e.g., intersecting two curves) on the other. This book is *not* about algebraic computation. We concentrate the algebra in so-called traits classes (see Chapter 5). The package comes with several ready-made traits-classes for the most common types of curves in use. We refer the interested reader to the book *Effective Computational Geometry of Curves and Surfaces* [29], which deals intensively with the algebra of the underlying algorithms for arrangements. More references to arrangement-related algebraic computation can be found in Section 5.7.

What You Should Know before Reading This Book

CGAL in general, and the *2D Arrangements* package in particular, rigorously adhere to the *generic programming* paradigm. The code of the library is written in C++, a programming language that

is well equipped for writing software according to the generic programming paradigm through the extensive use of class templates and function templates. We assume that the reader is familiar with the C++ programming language and with the basics of the generic programming paradigm. Some information on the required material is provided in the Introduction chapter.

Most of the underlying algorithmic material was developed within the field of computational geometry. We made an effort throughout the book to supply sufficient details for the problems we introduce and the techniques that we use to solve those problems, so that you can get started without first reading a computational geometry book or taking a computational geometry course. This was however a nontrivial task, and we may have not accomplished it in full. Computational geometry is a broad and deep field. Quite often we sketch an algorithm or a technique, without providing all the fine details, extensions, or variations. We attempted to give adequate references in the Bibliographic Notes and Remarks section at the end of each chapter, so that you can learn more about the topics if you wish.

While the theory of geometric algorithms has been studied for several decades and is very well developed, the practice of implementing geometric algorithms is far less advanced. Even if you master the theory and know what your options are to attack a problem, the theoretical measures of asymptotic time and space consumption may be misleading when an algorithm is put to the test of actual work. The budding area of *algorithm engineering* is about figuring out the better strategies in practice. At least for arrangements, this type of information is still scarce. If you wish to develop your own *efficient* applications, you will need to know more about algorithm engineering; see more in the Introduction chapter.

How to Obtain and Install CGAL

The Computational Geometry Algorithm Library (CGAL) comprises a large collection of packages. It has a small set of thin library objects (e.g., `.lib` files on WINDOWS) executables are linked with, and other components. In this book we only discuss the *2D Arrangements*, *2D Intersection of Curves*, *2D Regularized Boolean Set-Operations*, *2D Minkowski Sums*, *Envelopes of Curves in 2D*, and *Envelopes of Surfaces in 3D* packages. The data structures and operations provided by the *2D Arrangements* package serve as a foundation layer for the other packages.¹ You must obtain the entire collection and install the header files and at least the main CGAL library object in order to use any one of its packages.² The remaining library objects are optional, but three of them, i.e., **CGAL_Core**, **CGAL_Qt**, and **CGAL_Qt4**, are relevant in the context of this book. The **CGAL_Core** library object provides a custom version of the CORE library and depends on GMP and MPFR. (All three supply multi-precision number types.) The **CGAL_Qt** and **CGAL_Qt4** library objects provide convenient components for demonstration programs that have visual effects. They depend on QT Version 3 and Version 4, respectively; additional information about all these dependencies is provided in the next subsection.

When you develop an application based on the *2D Arrangements* package or related packages, you introduce `#include` statements in your code that refer to header files, e.g., **CGAL/Arrangement_2.h**. The compiled objects linked with the CGAL library object (and perhaps with other library objects) form the application executable. There is no trace of the *2D Arrangements* code in the library object, as the entire code resides only in header files. As a matter of fact, the library object itself is very thin. Thus, the advantage of creating and using a shared library object³ diminishes. Nevertheless, such an object is installed by default as part of the installation procedure.

The CGAL distribution consists of various packages. Each package comes with (i) header files

¹The *2D Intersection of Curves*, *2D Regularized Boolean Set-Operations*, *2D Minkowski Sums*, and *Envelopes of Surfaces in 3D* packages depend on the *2D Arrangements* package and in particular on its main data structure **Arrangement_2**. The *Envelopes of Curves in 2D* package does not depend on the **Arrangement_2** data structure but it does depend on other components of the *2D Arrangements* package.

²In the future CGAL may be divided into interdependent components with a clear set of dependencies between them. This will allow you to obtain and install only a portion of the entire collection, or upgrade only those components that you may find necessary.

³It is called a dynamic-link library (DLL) on WINDOWS platforms, and a shared object (SO) on UNIX platforms.

that consist not only of the interface but also of the generic implementation of the package code, (ii) comprehensive and didactic documentation divided into two parts, namely, the programming guide and the reference manual, (iii) a set of non-interactive standalone example programs, and optionally (iv) an interactive demonstration program with visual effects.⁴ The packages described in this book are no exception. The programming guide of the *2D Arrangements* package, for example, consists of roughly 100 pages. There are more than 50 (non-interactive) examples that exercise various operations on arrangements, and there is an interactive program included in the package that demonstrates its features. The *2D Regularized Boolean Set-Operations* and *Envelopes of Surfaces in 3D* packages also come with demonstration programs.

Libraries That CGAL Uses

BOOST: The CGAL library depends on other libraries, which must be installed a priori. First, it depends on BOOST. BOOST provides free peer-reviewed portable C++ source libraries that work well with, and are in the same spirit as, the C++ Standard Template Library (STL). Several BOOST libraries are implemented using only templates defined in header files; other libraries include source code that needs to be built and installed, such as the `Program_options` and `Thread` libraries. The latter is a prerequisite, and, as such, it must be installed before you commence the build process of CGAL. The former is required to build some of the CGAL demonstration programs. On WINDOWS, for example, you can download a WINDOWS installer for the current BOOST release (1.45.0 at the time this book was written), available at <http://www.boostpro.com/download>. Popular LINUX and UNIX distributions such as FEDORA, DEBIAN, and NETBSD include pre-built BOOST packages. Source code, documentation, and other related material can be obtained from <http://www.boost.org/>.

GMP and MPFR: GMP and MPFR are two libraries that provide types for multi-precision integers, rational numbers, and floating-point numbers. They are highly recommended but optional, as CGAL has built-in multi-precision number types used when none of the external libraries that provide multi-precision number types is installed on your system. (CORE and LEDA—see the next paragraph—also provide multi-precision rational-number types.) Do not use the CGAL built-in number types if you want to get optimal performance. LINUX distributions include the GMP and MPFR libraries by default. The MPFR library can be obtained from <http://www.mpfr.org/>. The page <http://gmplib.org/> provides useful information regarding the installation of GMP. The CGAL distribution includes precompiled versions of GMP and MPFR for Visual C++ users who find directly obtaining and installing these libraries less convenient. Notice that these libraries make use of processor-specific instructions when compiled with maximum optimization enabled. It is reasonable to assume that downloaded precompiled objects do not contain these instructions. If it turns out that this is a performance bottleneck for your application, you can always download the source package and compile it locally.

CORE and LEDA: If you want to handle arrangements of non-linear curves in a robust manner, you must use support for algebraic numbers to carry out some of the arithmetic operations involved. In some of those cases the CORE or LEDA libraries come to the rescue. In particular, constructing an arrangement of conic curves or of Bézier curves requires the CORE library. A custom version of the CORE library that has its own license is distributed with CGAL for your convenience. More information on the CORE library can be found at <http://www.cs.nyu.edu/exact/core/>. Information on LEDA can be found at <http://www.algorithmic-solutions.com/leda/>. There are free and commercial editions for this library.

⁴A noteworthy component every package includes (but not distributed as part of public releases) is a collection of functional and regression tests. The tests of all packages combined compose the CGAL test suite. All the tests in the test suite run daily, and their results are automatically assembled and analyzed.

QT: Most CGAL demos, including the ones provided by the *2D Arrangements*, *2D Regularized Boolean Set-Operations*, and *Envelopes of Surfaces in 3D* packages, are based on the QT cross-platform library, developed by Trolltech, and currently owned by Nokia. Some of the CGAL demos are still based on QT Version 3—an older version of QT, which was not licensed free of charge for WINDOWS. Unfortunately, some of the demos of the packages above are included in this subset. However, they were being ported at the time this book was written to Version 4, the latest version of QT. Editions with open-source licenses of QT Version 4 are available for WINDOWS as well as for LINUX. QT is included by default in LINUX distributions, and can be obtained from <http://qt.nokia.com/products>.

Supported Platforms⁵

Table 1: CGAL-supported platforms.

Compiler	Operating System
GNU g++	LINUX, Mac OS, Solaris, XP, XP64, Vista, WINDOWS 7
MS Visual C++ (.NET)	XP, XP64, Vista, WINDOWS 7
Intel Compiler	LINUX

CGAL Version 3.8, the latest version of CGAL at the time this book was written, is supported by the platforms listed in Table 1. If your platform is not on this list, it does not necessarily mean that CGAL does not work on your platform. Note that 64-bit variants of these platforms are also supported. For the specific versions of the supported operating systems and compilers consult the official Web page <http://www.cgal.org/platforms.html>.

Visual C++-Related Issues

Choosing the Runtime Library: Do not mix static and dynamic versions of the C runtime (CRT) libraries, and make sure that all static and dynamic link libraries are built with the same version of the CRT libraries. More than one copy of a (CRT) library in a process can cause problems because static data in one copy is not shared with other copies. The linker prevents you from linking with both static and dynamic versions within one executable file (**.exe**), but you can still end up with two (or more) copies of CRT libraries. For example, a dynamic-link library linked with the static (non-DLL) versions of the CRT libraries can cause problems when used with an executable file that was linked with the dynamic (DLL) version of the CRT libraries. (You should also avoid mixing the debug and non-debug versions of the libraries in one process, because the debug version might have different definitions of objects, as is the case, for example, with the STL containers and iterators in VC9.)

Automatic Library Selection: Starting from CGAL Version 3.3, BOOST-style automatic library selection (auto-linking) is used for all CGAL library objects (including **CGAL**, **CGAL_Qt**, **CGAL_Qt4**, and **CGAL_Core**) and the third-party libraries GMP and MPFR installed via the WINDOWS installer. This feature allows you to choose any build configuration for your application without worrying about a matching precompiled library (**.lib**) to link against. Auto-linking also prevents mixing different runtime libraries.

Runtime Type Information: You must enable runtime type information. This is forced with the **/GR** compiler option.

Compiling: When compiling some of the source code listed in this book (and also included on the book's website <http://acg.cs.tau.ac.il/cgal-arrangement-book>) using Visual C++, the following warning is issued by the compiler:

⁵The term “platform” refers to the combination of an operating system and a compiler.

warning C4503: ... decorated name length exceeded, name was truncated.

This implies that the decorated name was longer than the compiler limit (4,096 according to MSDN), and thus was truncated. You must reduce the number of arguments or name length of identifiers used to avoid the truncation. For example, instead of

```
typedef CGAL::Cartesian<Number_type>           Kernel;
define
```

```
struct Kernel : public CGAL::Cartesian<Number_type> {};
```

In many cases, though, the truncation is harmless. You can suppress only these warning messages by adding the following pragma statement at the beginning of the file that contains the source code:

```
#pragma warning( disable : 4503 )
```

Alternatively, you may use the `/w` or the `/W` compiler options to suppress all warnings or to set the appropriate warning-level, respectively.⁶

Installing CGAL

There are ongoing efforts to make the CGAL installation process as easy as possible. In spite of these efforts, and mainly due to the dependence of CGAL on other libraries, you may encounter difficulties while trying to install CGAL on your computer for the first time. We next give some tips on the installation and refer you below to sources containing more information. Answers to frequently asked questions appear at <http://www.cgal.org/FAQ.html> and an archive of a discussion forum can be found at <https://lists-sop.inria.fr/sympa/arc/cgal-discuss>.

All versions of CGAL, including the latest version of the library, that is, Version 3.8 at the time this book was written, can be obtained online. Precompiled versions of CGAL library-objects, source files required for development, demonstration programs, and documentation are available as **deb** packages for LINUX distributions based on DEBIAN, such as DEBIAN and UBUNTU.

There are no precompiled versions for WINDOWS. A self-extracting executable that installs the CGAL sources, and that allows you to select and download some precompiled third-party libraries is available at <http://gforge.inria.fr/projects/cgal/>. It obtains the source files required for the configuration and compilation of the various CGAL library-objects, and all other components required for the development of applications. Alternatively, you can find an archive (a UNIX “tar ball”), e.g., **CGAL-3.7.tar.gz**, that contains all the code (but not necessarily the manuals) at <http://www.cgal.org/download.html>. Download the archive, and follow the instructions described at http://www.cgal.org/Manual/last/doc_html/installation_manual/Chapter_installation_manual.html, which are summarized below, to configure, build, and install CGAL. This process, commonly referred to as the *build* process, is carried out by CMAKE—a cross-platform build system generator. If CMAKE is not installed already on your system, you can obtain it from <http://www.cmake.org/>.

The archive can be expanded using the following command:

```
tar zxvf tar-ball
```

As a result, the CGAL root directory will be created, e.g., **CGAL-3.8**.

Ideally, building CGAL and then building an example that depends on CGAL amounts to

```
cd CGAL-3.7           # go to CGAL directory
cmake .               # configure CGAL
make                 # build the CGAL library objects
cd examples/Arrangement_on_surface_2 # go to an example directory
cmake -DCGAL_DIR=$HOME/CGAL-3.7 . # configure the examples
make io              # build the io example
```

⁶Suppressing all warnings is a risky practice, as warnings may convey crucial information about real problems.

When you build CGAL, you are free to choose which generator to use. The generator is determined by a specific compiler, a development environment, or other choices. On WINDOWS, for example, you may choose to generate **project** and **solution** files for a specific version of the common Visual C++ development environment. You may choose to generate **makefile** files by a specific version of Visual C++, or you may choose to generate **makefile** files for the CYGWIN port of g++. The CGAL code is written in ISO/IEC C++, and compiles with most C++ compilers; consult the website http://www.cgal.org/platforms_frame.html for an up-to-date list of supported compilers and operating systems.

License

The CGAL library is released under open-source licenses. Common parts of CGAL (i.e., the *kernel* and the *support library*; see Section 1.4.2) are distributed under the terms of the GNU Lesser General Public License (or GNU LGPL for short) and the remaining part (i.e., the *basic library*, which contains most of the software described in the book) is distributed under the terms of the Q Public License (QPL), a non-copyleft free software license created by Trolltech for its free edition of the QT toolkit. The QPL captures the general meaning of the GNU General Public License (GPL), but is incompatible with it, meaning that you cannot legally distribute products derived from both GPL'ed and QPL'ed code. Trolltech changed its policy, and released Version 4.0 of QT under the terms of GPL Version 2, abandoning QPL. As a consequence, the use of QPL became rare and is decreasing. Most probably, GPL will replace QPL for all relevant CGAL components. (This may have already taken place by the time you are reading this.) A copy of each license used by CGAL components is included in the distribution in the **LICENSE** file and its derivatives.

The CGAL library may be used freely in noncommercial applications. Commercial licenses for selected components of CGAL are provided by GEOMETRY FACTORY.⁷ There are licenses for industrial and academic development and for industrial research.

Style Conventions

These style conventions are used in this book:

- **Bold** — command names.
- **SMALL CAPS** — special terms and names, e.g., CGAL.
- **Sans Serif** — generic programming concepts, e.g., CopyConstructible.

Code excerpts are set off from the text in a monospace font, for example:


```
#include <iostream>
int main() { std::cout << "Hello_World" << std::endl; return 0; }
```

Implicit association between identifiers in the code set off in a monospace font and the corresponding entity denoted in math font is assumed. For example, the identifier **p**, which explicitly refers to a point object (or a pointer to a point object) also implicitly refers to the point denoted by *p*.

Topics that are particularly complicated—and that you can skip, if, for example, you are new to CGAL—are surrounded by the *advanced* marks:

————— *advanced* —————
 |
 | This can apply to a single paragraph or to an entire section.
 |
 |————— *advanced* —————


⁷See <http://www.geometryfactory.com/>.

Simple tasks that are left for the reader to practice with are marked with the  icon. For example, assuming that the listing of an imaginary program coded in `ex_compute_union.cpp` is given:



Try: Modify the program coded in `ex_compute_union.cpp` such that it computes the union of discs rather than the union of squares.



Example programs are marked with the  icon.



Example: Typically, this consists of a description of the example followed by the program code.

Exercises

At the end of every chapter, with the exception of the Introduction chapter, you will find several exercises. These are meant to recite the material covered in that chapter, and altogether strengthen the practical direction this book has taken. The exercises are divided into two categories according to their difficulty levels. Solving an exercise marked as **Project** typically requires considerably more resources than solving a non-marked exercise. Several programming exercises (typically marked as **Project**) ask for the implementation of a useful feature that is currently not implemented, but could nicely fit into CGAL. If you work out such an exercise, and you believe that your solution meets the high standards of CGAL code, kindly send it to us. We will consider your contribution as a potential candidate for inclusion in a future release.

The Cover

The illustration on the cover of the book depicts an arrangement of Fibonacci spirals, which govern the layout of sunflower seeds. This is explained in detail in Exercise 5.7.

Errata

Errors and corrections will appear on <http://acg.cs.tau.ac.il/cgal-arrangement-book>, the book's website, as soon as errors are detected.

Acknowledgments

The CGAL *2D Arrangements* package is an integrated part of CGAL. The initial development of CGAL was funded by two European Union projects, CGAL and GALIA, over three years (1996–1999). Several sites kept on working on CGAL after the European Union funding stopped. The European Union projects ECG⁸ (Effective Computational Geometry for curves and surfaces) and ACS⁹ (Algorithms for Complex Shapes with certified topology and numerics) provided further partial support for new research and development in CGAL.

The work on the *2D Arrangements* package and its related packages took place mostly at Tel Aviv University, and has been supported in part by the Israel Science Foundation through several grants to Dan Halperin.

The code of the *2D Arrangements* and *2D Intersection of Curves* packages is the result of a long development process. Initially (and until Version 3.1), the code was spread among several components, namely, `Topological_map`, `Planar_map_2`, `Planar_map_with_intersections_2`,

⁸See <http://www-sop.inria.fr/prisme/ECG/>.

⁹See <http://acs.cs.rug.nl/>.

and **Arrangement_2**, that were developed by Ester Ezra, Eyal Flato, Efi Fogel, Dan Halperin, Iddo Hanniel, Idit Haran, Shai Hirsch, Eugene Lipovetsky, Oren Nechushtan, Sigal Raab, Ron Wein, Baruch Zukerman, and Tali Zvi.

In Version 3.2, as part of the ACS project, the packages have gone through a major redesign, resulting in improved and unified *2D Arrangements* and *2D Intersection of Curves* packages. The code of the two new packages was restructured and developed by Efi Fogel, Idit Haran, Ron Wein, and Baruch Zukerman. This version included for the first time a new geometry-traits class that handles circular and linear curves, and is based on the circular kernel. The circular kernel was developed by Monique Teillaud, Sylvain Pion, and Julien Hazebrouck.

A significant outcome of the redesign was the reduction of the geometry-traits concept, which was assisted by Monique Teillaud. Version 3.2 also exploited an optimized multi-set data structure implemented as a red-black tree, which was developed by Ron Wein. This feature was reviewed by Remco Velthkamp.

Version 3.2.1 featured arrangements of unbounded curves for the first time. The design and development of this feature required yet another restructuring of the entire package. All this was done by Eric Berberich, Efi Fogel, Dan Halperin, Ophir Setter, and Ron Wein. Michael Hemmer helped in tuning up parts of the geometry-traits concept related to unbounded curves. Sylvain Pion reviewed Version 3.2.1.

Version 3.7 of CGAL introduced a geometry-traits class that handles planar algebraic curves of arbitrary degree. It was developed by Michael Kerber and Eric Berberich and reviewed by Efi Fogel.

At the time this book was written an internal version of CGAL introduced a new geometry-traits class that handles rational arcs. It was developed by Oren Salzman and Michael Hemmer, reviewed by Eric Berberich, and was expected to replace in version 3.9 an old traits, which handles the same family of curves, developed by Ron Wein.

The code of the *2D Regularized Boolean Set-Operations* package was developed by Efi Fogel, Dan Halperin, Ophir Setter, Guy Zucker, Ron Wein, and Baruch Zukerman. Andreas Fabri reviewed this package. The code of the *2D Minkowski Sums* package was developed by Ron Wein. Michael Hoffmann reviewed this package and the two packages that we mention next. The code of the *Envelopes of Curves in 2D* package was developed by Ron Wein. The code of the *Envelopes of Surfaces in 3D* package was developed by Dan Halperin, Michal Meyerovitch, Ron Wein, and Baruch Zukerman.

The ongoing work on an extension of the *2D Arrangements* package to handle arrangements on surfaces (which is still at a prototype stage, and hence not described in the book) had a major influence on the quality of the planar-arrangement code. This work is carried out by the authors of this book together with Eric Berberich, Michael Hemmer, Michael Kerber, Kurt Mehlhorn, and Ophir Setter.

The quality of the book in general and the example programs and applications listed in it in particular were immensely improved following a diligent review and code revisiting and rewriting conducted by Ophir Setter.

The *2D Arrangements* package depends on other components of the library, which have been developed, and are still being developed, by many people. The package benefits from many services provided collectively to all parts of CGAL by groups and individuals, many of whom are not explicitly acknowledged here. Nonetheless, we would like to thank the CGAL Board (formerly the CGAL Editorial Board) and its current and past members, Pierre Alliez, Eric Berberich, Andreas Fabri, Bernd Gärtner, Michael Hemmer, Susan Hert, Michael Hoffmann, Menelaos Karavelas, Lutz Kettner, Sylvain Pion, Marc Pouget, Laurent Rineau, Monique Teillaud, Mariette Yvinec, and Remco Velthkamp, GEOMETRY FACTORY and its members, Andreas Fabri, Fernando Cacciola, Sébastien Lorient, and Laurent Rineau, and all members of the CGAL developers' community for providing those vital components and services and for sharing their wisdom through many rich discussions.

Over the years the *2D Arrangements* package benefitted from intensive discussions in private meetings, workshops, conferences, and over electronic means, with many folks, including Ioannis Z. Emiris, Peter Hechenberger, Eli Packer, and Stefan Schirra. We are indebted to them for their

invaluable insights and critiques.

Last but not least we owe a special debt to the creators of CGAL and the people who nursed and nurtured CGAL during its early stages. This book would not have been written if it had not been for these visionaries.

Tel Aviv, 2011

*Efi Fogel
Dan Halperin
Ron Wein*

CGAL Arrangements and Their Applications
A Step-by-Step Guide

Fogel, E.; Halperin, D.; Wein, R.

2012, XIX, 293 p., Hardcover

ISBN: 978-3-642-17282-3