The LEDA Platform

for

Combinatorial and Geometric

Computing

KURT MEHLHORN
STEFAN NÄHER

Contents

Preface

LEDA (Library of Efficient Data Types and Algorithms) is a C++ library of combinatorial and geometric data types and algorithms. It offers

Data Types, such as random sources, stacks, queues, maps, lists, sets, partitions, dictionaries, sorted sequences, point sets, interval sets, ...,

Number Types, such as integers, rationals, bigfloats, algebraic numbers, and linear algebra.

Graphs and Supporting Data Structures, such as node- and edge-arrays, node- and edge-maps, node priority queues and node partitions, iteration statements for nodes and edges, ...,

Graph Algorithms, such as shortest paths, spanning trees, flows, matchings, components, planarity, planar embedding, ...,

Geometric Objects, such as points, lines, segments, rays, planes, circles, polygons, ...,

Geometric Algorithms, such as convex hulls, triangulations, Delaunay diagrams, Voronoi diagrams, segment intersection, ..., and

Graphical Input and Output.

The modules just mentioned cover a considerable part of combinatorial and geometric computing as treated in courses and textbooks on data structures and algorithms [AHU83, dBvKOS97, BY98, CLR90, Kin90, Kle97, NH93, Meh84b, O'R94, OW96, PS85, Sed91, Tar83a, van88, Woo93].

From a user's point of view, LEDA is a platform for combinatorial and geometric computing. It provides *algorithmic intelligence* for a wide range of applications. It eases a programmer's life by providing powerful and easy-to-use data types and algorithms which can be used as building blocks in larger programs. It has been used in such diverse areas as code optimization, VLSI design, robot motion planning, traffic scheduling, machine learning and computational biology. The LEDA system is installed at more than 1500 sites.

We started the LEDA project in the fall of 1988. The project grew out of several considerations.

- We had always felt that a significant fraction of the research done in the algorithms area was eminently practical. However, only a small part of it was actually used. We frequently heard from our former students that the intellectual and programming effort needed to implement an advanced data structure or algorithm is too large to be cost-effective. We concluded that *algorithms research must include implementation if the field wants to have maximum impact*.
- We surveyed the amount of code reuse in our own small and tightly connected research group. We found several implementations of the same balanced tree data structure. Thus there was constant reinvention of the wheel even within our own small group.
- Many of our students had implemented algorithms for their master's thesis. Work invested by these students was usually lost after the students graduated. We had no depository for implementations.
- The specifications of advanced data types which we gave in class and which we found in text books, including the one written by one of the authors, were incomplete and not sufficiently abstract to allow to combine implementations easily. They contained phrases of the form: "Given a pointer to a node in the heap its priority can be decreased in constant amortized time". Phrases of this kind imply that a user of a data structure has to know its implementation. As a consequence combining implementations is a non-trivial task. We performed the following experiment. We asked two groups of students to read the chapters on priority queues and shortest path algorithms in a standard text book, respectively, and to implement the part they had read. The two parts would not fit, because the specifications were incomplete and not sufficiently abstract.

We started the LEDA project to overcome these shortcomings by creating a platform for combinatorial and geometric computing. *LEDA should contain the major findings of the algorithms community in a form that makes them directly accessible to non-experts having only limited knowledge of the area*. In this way we hoped to reduce the gap between research and application.

The LEDA system is available from the LEDA web-site and also through anonymous ftp.

`http://www.mpi-sb.mpg.de/LEDA/leda.html`

or

`ftp.mpi-sb.mpg.de, directory pub/LEDA`

A commercial version of LEDA is available from Algorithmic Solutions Software GmbH.

`http://www.algorithmic-solutions.de`

LEDA can be used with almost any C++ compiler and is available for UNIX and WINDOWS systems. The LEDA mailing list (see the LEDA web page) facilitates the exchange of information between LEDA users.

This book provides a comprehensive treatment of the LEDA system and its use. We treat the architecture of the system, we discuss the functionality of the data types and algorithms available in the system, we discuss the implementation of many modules of the system, and we give many examples for the use of LEDA. We believe that the book is useful to five types of readers: readers with a general interest in combinatorial and geometric computing, casual users of LEDA, intensive users of LEDA, library designers and software engineers, and students taking an algorithms course.

The book is structured into fourteen chapters.

Chapter 1, Introduction, introduces into the use of LEDA and gives an overview of the system and our design goals.

Chapter 2, Foundations, discusses the basic concepts of the LEDA system. It defines key concepts, such as type, object, variable, value, item, copy, linear order, and running time, and it relates these concepts to C++. We recommend to read this chapter quickly and to come back to it as needed. The detailed knowledge of this chapter is a prerequisite for the intensive use of LEDA. The casual user should be able to satisfy his needs by simply modifying example programs given in the book. The chapter draws upon several sources: object-oriented programming, abstract data types, and efficient algorithms. It lays out many of our major design decisions which we call LEDA axioms.

Chapters 2 to 12 form the bulk of the book. They constitute a guided tour of LEDA. We discuss numbers, basic data types, advanced data types, graphs, graph algorithms, embedded graphs, geometry kernels, geometry algorithms, windows, and graphwins. In each chapter we introduce the functionality of the available data types and algorithms, illustrate their use, and give the implementation of some of them.

Chapter 13, Implementation, discusses the core part of LEDA, e.g., the implementation of parameterized data types, implementation parameters, memory management, and iteration.

Chapter 14, Documentation, discusses the principles underlying the documentation of LEDA and the tools supporting it.

The book can be read without having the LEDA system installed. However, access to the LEDA system will greatly increase the *joy of reading*. The demo directory of the LEDA system contains numerous programs that allow the reader to exercise the algorithms discussed in the book. The demos give a feeling for the functionality and the efficiency of the algorithms, and in a few cases even animate them.

The book can be read from cover to cover, but we expect few readers to do it. We wrote the book such that, although the chapters depend on each other as shown in Figure 0.1, most chapters can be read independently of each other. We sometimes even repeat material in order to allow for independent reading.

All Readers should start with the chapters Introduction and Foundations. In these chapters we give an overview of LEDA and introduce the basic concepts of LEDA. We suggest to read the chapter on foundations quickly and to come back to it as needed.

The chapter on basic data types (list, stacks, queues, array, random number generators, and strings) should also be read by every reader. The basic data types are ubiquitous in the book.

Having read the chapters Introduction, Foundations and Basic Data Types, the reader may take different paths depending on interest.

Casual users of LEDA should read the chapters treating their domain of interest, and *intensive users of LEDA* should also read the chapter on implementation.

Readers interested in Data Structures should read the chapters on advanced data types, on implementation, and some of the sections of the chapter on geometric algorithms. The chapter on advanced data types treats dictionaries, search trees and hashing, priority queues, partitions, and sorted sequences, and the chapter on implementation discusses, among other things, the realization of parameterized data types. The different sections in the chapter on advanced data types can be read independently. In the chapter on geometric algorithms we recommend the section on dynamic Delaunay triangulations; some knowledge of graphs and computational geometry is required to read it.

Readers interested in Graphs and Graph Algorithms should continue with the chapter on graphs. From there one can proceed to either the chapter on graph algorithms or the chapter on embedded graphs. Within the chapter on graph algorithms the sections can be read independently. However, the chapter on embedded graphs must be read from front to rear. Some knowledge of priority queues and partitions is required for some of the sections on graph algorithms.

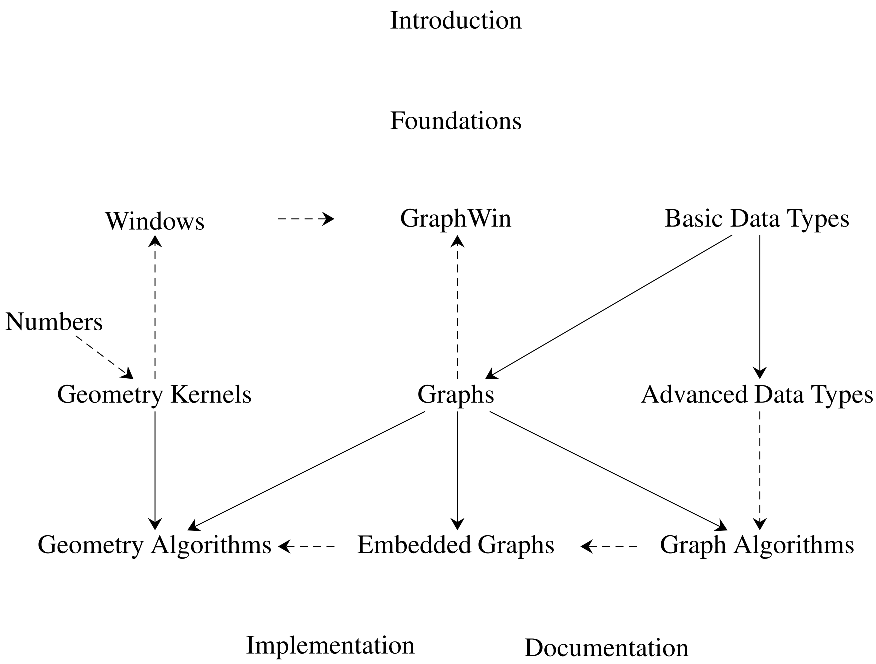

Figure 0.1 The dependency graph between the chapters. A dashed arrow means that partial knowledge is required and a solid arrow means that extensive knowledge is required. Introduction and Foundations should be read before all other chapters and Implementation and Documententation can be read independently from the other chapters.

Readers interested in Computational Geometry can continue with either the chapter on graphs or the chapter on geometry kernels. Both chapter are a prerequisite for the chapter on geometric algorithms. The chapter on geometry kernels requires partial knowledge of the chapter on numbers. The chapter on geometric algorithms splits into two parts that can be read independently. The first part is on convex hulls, Delaunay triangulations, and Voronoi diagrams, and the second part is on line segment intersection and polygons.

Geometric algorithms are dull without graphical input and output. The required knowledge is provided by the chapter on windows. The section on the Voronoi demo in the chapter on geometric algorithms gives a comprehensive example for the interplay between geometric data types and algorithms and the window class.

Readers interested in Algorithm Animation should read the chapter on windows and graphwin, the section on animating strongly connected components in the chapter on graph algorithms, the section on the Voronoi demo in the geometric algorithms chapter, and study the many programs in the xlman subdirectory of the demo directory.

Readers interested in Software Libraries should read the chapters on foundations, on implementation, and on documentation. They should also study some other chapters at their own choice.

Readers interested in developing a LEDA Extension Package should read the chapters on implementation and documentation in addition to the chapters related to their domain of algorithmic interest.

For all the algorithms discussed in the book, we also derive the required theory and give the proof of correctness. However, sometimes our theoretical treatment is quite compact and tailored to our specific needs. We refer the reader to the textbooks [AHU83, Meh84b, Tar83a, CLR90, O'R94, Woo93, Sed91, Kin90, van88, NH93, PS85, BY98, dBvKOS97] for a more comprehensive view.

LEDA is implemented in C++ and we expect our readers to have some knowledge of it. We are quite conservative in our use of C++ and hence a basic knowledge of the language suffices for most parts of the book. The required concepts include classes, objects, templates, member functions, and non-member functions and are typically introduced in the first fifty pages of a C++ book [LL98, Mur93, Str91]. Only the chapter on implementation requires to know more advanced concepts like inheritance and virtual functions.

The book contains many tables showing *running times*. All running times were determined on an ULTRA-SPARC with 300 MHz CPU and 256 MByte main memory. LEDA and all programs were compiled with CC (optimization flags -O and -DLEDA_CHECKING_OFF).

We welcome *feedback* from our readers. A book of this length is certain to contain errors. If you find any errors or have other constructive suggestions, we would appreciate hearing from you. Please send any comments concerning the book to

`LEDAbook@mpi-sb.mpg.de`

For comments concerning the system use

`LEDAresearch@mpi-sb.mpg.de`

or sign up for the LEDA discussion group. We will maintain a list of corrections on the web.

We received financial support from a number of sources. Of course, our home institutions deserve to be mentioned first. We started LEDA at the Universität des Saarlandes in Saarbrücken, in the winter 1990/1991 we both moved to the Max-Planck-Institut für Informatik, also in Saarbrücken, and in the fall of 1994 Stefan Näher moved to the Martin-Luther-Universität in Halle. Our work was also supported by the Deutsche Forschungsgemeinschaft (Sonderforschungsbereich SFB

124 VLSI-Entwurf und Parallelität und Schwerpunktprogramm Effiziente Algorithmen und ihre Anwendungen), by the Bundesministerium für Forschung und Technologie (project SOFTI), and by the European Community (projects ALCOM, ALCOM II, ALCOM-IT, and CGAL).

Discussions with many colleagues, bug reports, experience reports (positive and negative), suggestions for changes and extensions, and code contributions helped to shape the project. Of course, we could not have built LEDA without the help of many other persons. We want to thank David Alberts, Ulrike Bartuschka, Christoph Burnikel, Ulrich Finkler, Stefan Funke, Evelyn Haak, Jochen Könemann, Ulrich Lauther, Andreas Luleich, Mathias Metzler, Michael Müller, Michael Muth, Markus Neukirch, Markus Paul, Thomas Papanikolaou, Stefan Schirra, Christian Schwarz, Michael Seel, Jack Snoeyink, Ken Thornton, Christian Uhrig Michael Wenzel, Joachim Ziegler, Thomas Ziegler, and many others for their contributions.

Special thanks go to Christian Uhrig, the chief officer of Algorithmic Solutions GmbH, to Michael Seel, who is head of the LEDA-group at the MPI, and to Ulrich Lauther from Siemens AG, our first industrial user.

Evelyn Haak typeset the book. Actually, she did a lot more. She made numerous suggestions concerning the layout, she commented on the content, and she suggested changes. Michael Seel, Joachim Ziegler, Volker Priebe, Stefan Funke, Mark Ziegelmann, René Weißkircher, and Gunnar Klau proof-read parts of the book. We want to thank them for their many constructive comments. Of course, all the remaining errors are ours.

Finally, we want to thank David Tranah from Cambridge Univerisity Press for his support and patience.

We hope that you enjoy reading this book and that LEDA eases your life as a programmer.

Stefan Näher	Kurt Mehlhorn
Halle, Germany	Saarbrücken, Germany
December, 1998	December, 1998