

To

Ena and Ulli,

Uli, Steffi, Tim, and Tim

LEDA
A Platform for
Combinatorial and Geometric
Computing

KURT MEHLHORN
STEFAN NÄHER

Contents

Preface	<i>page xi</i>
1 Introduction	1
1.1 Some Programs	1
1.2 The LEDA System	8
1.3 The LEDA Web-Site	10
1.4 Systems that Go Well with LEDA	11
1.5 Design Goals and Approach	11
1.6 History	13
2 Foundations	16
2.1 Data Types	16
2.2 Item Types	26
2.3 Copy, Assignment, and Value Parameters	32
2.4 More on Argument Passing and Function Value Return	36
2.5 Iteration	39
2.6 STL Style Iterators	41
2.7 Data Types and C++	41
2.8 Type Parameters	45
2.9 Memory Management	47
2.10 Linearly Ordered Types, Equality and Hashed Types	47
2.11 Implementation Parameters	51
2.12 Helpful Small Functions	52
2.13 Error Handling	54
2.14 Program Checking	54

2.15	Header Files, Implementation Files, and Libraries	56
2.16	Compilation Flags	57
3	Basic Data Types	58
3.1	Stacks and Queues	58
3.2	Lists	61
3.3	Arrays	73
3.4	Compressed Boolean Arrays (Type <code>int_set</code>)	77
3.5	Random Sources	79
3.6	Pairs, Triples, and such	94
3.7	Strings	95
3.8	Making Simple Demos and Tables	96
4	Numbers and Matrices	99
4.1	Integers	99
4.2	Rational Numbers	103
4.3	Floating Point Numbers	104
4.4	Algebraic Numbers	108
4.5	Vectors and Matrices	117
5	Advanced Data Types	121
5.1	Sparse Arrays: Dictionary Arrays, Hashing Arrays, and Maps	121
5.2	The Implementation of the Data Type Map	133
5.3	Dictionaries and Sets	146
5.4	Priority Queues	147
5.5	Partition	158
5.6	Sorted Sequences	180
5.7	The Implementation of Sorted Sequences by Skiplists	196
5.8	An Application of Sorted Sequences: Jordan Sorting	228
6	Graphs and their Data Structures	240
6.1	Getting Started	240
6.2	A First Example of a Graph Algorithm: Topological Ordering	244
6.3	Node and Edge Arrays and Matrices	245
6.4	Node and Edge Maps	249
6.5	Node Lists	251
6.6	Node Priority Queues and Shortest Paths	253
6.7	Undirected Graphs	257
6.8	Node Partitions and Minimum Spanning Trees	259
6.9	Graph Generators	263
6.10	Input and Output	269
6.11	Iteration Statements	271

6.12	Basic Graph Properties and their Algorithms	274
6.13	Parameterized Graphs	280
6.14	Space and Time Complexity	281
7	Graph Algorithms	283
7.1	Templates for Network Algorithms	283
7.2	Algorithms on Weighted Graphs and Arithmetic Demand	286
7.3	Depth-First Search and Breadth-First Search	293
7.4	Reachability and Components	296
7.5	Shortest Paths	316
7.6	Bipartite Cardinality Matching	360
7.7	Maximum Cardinality Matchings in General Graphs	393
7.8	Maximum Weight Bipartite Matching and the Assignment Problem	413
7.9	Weighted Matchings in General Graphs	443
7.10	Maximum Flow	443
7.11	Minimum Cost Flows	489
7.12	Minimum Cuts in Undirected Graphs	491
8	Embedded Graphs	498
8.1	Drawings	499
8.2	Bidirected Graphs and Maps	501
8.3	Embeddings	506
8.4	Order-Preserving Embeddings of Maps and Plane Maps	511
8.5	The Face Cycles and the Genus of a Map	512
8.6	Faces, Face Cycles, and the Genus of Plane Maps	515
8.7	Planarity Testing, Planar Embeddings, and Kuratowski Subgraphs	519
8.8	Manipulating Maps and Constructing Triangulated Maps	564
8.9	Generating Plane Maps and Graphs	569
8.10	Faces as Objects	571
8.11	Embedded Graphs as Undirected Graphs	574
8.12	Order from Geometry	575
8.13	Miscellaneous Functions on Planar Graphs	577
9	The Geometry Kernels	581
9.1	Basics	583
9.2	Geometric Primitives	593
9.3	Affine Transformations	601
9.4	Generators for Geometric Objects	604
9.5	Writing Kernel Independent Code	606
9.6	The Dangers of Floating Point Arithmetic	609
9.7	Floating Point Filters	613
9.8	Safe Use of the Floating Point Kernel	632

9.9	A Glimpse at the Higher-Dimensional Kernel	634
9.10	History	634
9.11	LEDA and CGAL	635
10	Geometry Algorithms	637
10.1	Convex Hulls	637
10.2	Triangulations	656
10.3	Verification of Geometric Structures, Basics	664
10.4	Delaunay Triangulations and Diagrams	672
10.5	Voronoi Diagrams	686
10.6	Point Sets and Dynamic Delaunay Triangulations	708
10.7	Line Segment Intersection	731
10.8	Polygons	758
10.9	A Glimpse at Higher-Dimensional Geometric Algorithms	790
10.10	A Complete Program: The Voronoi Demo	795
11	Windows and Panels	813
11.1	Pixel and User Coordinates	814
11.2	Creation, Opening, and Closing of a Window	815
11.3	Colors	817
11.4	Window Parameters	818
11.5	Window Coordinates and Scaling	821
11.6	The Input and Output Operators \ll and \gg	821
11.7	Drawing Operations	822
11.8	Pixrects and Bitmaps	823
11.9	Clip Regions	828
11.10	Buffering	829
11.11	Mouse Input	831
11.12	Events	834
11.13	Timers	842
11.14	The Panel Section of a Window	844
11.15	Displaying Three-Dimensional Objects: <code>d3_window</code>	855
12	GraphWin	857
12.1	Overview	858
12.2	Attributes and Parameters	861
12.3	The Programming Interface	866
12.4	Edit and Run: A Simple Recipe for Interactive Demos	875
12.5	Customizing the Interactive Interface	879
12.6	Visualizing Geometric Structures	890
12.7	A Recipe for On-line Demos of Network Algorithms	892
12.8	A Binary Tree Animation	897

13	On the Implementation of LEDA	904
13.1	Parameterized Data Types	904
13.2	A Simple List Data Type	904
13.3	The Template Approach	906
13.4	The LEDA Solution	909
13.5	Optimizations	929
13.6	Implementation Parameters	934
13.7	Independent Item Types (Handle Types)	937
13.8	Memory Management	941
13.9	Iteration	943
13.10	Priority Queues by Fibonacci Heaps (A Complete Example)	946
14	Manual Pages and Documentation	963
14.1	Lman and Fman	963
14.2	Manual Pages	966
14.3	Making a Manual: The Mkman Command	984
14.4	The Manual Directory in the LEDA System	985
14.5	Literate Programming and Documentation	986
	Bibliography	992
	Index	1002

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, dBKOS97, BY98, CLR90, Kin90, Kle97, NH93, Meh84, O'R94, OW96, PS85, Sed91, Tar83, 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.

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

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 the reader to 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 that you read this chapter quickly and 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 3 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 A, 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 that you read the chapter on foundations quickly and come back to it as needed.

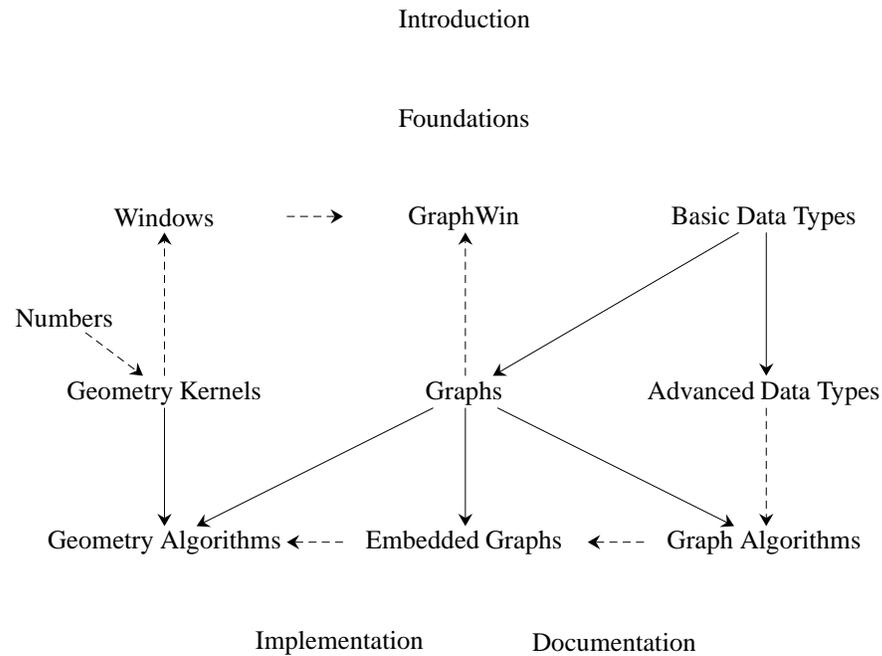


Figure A The dependency graph between the chapters. A dashed arrow means that partial knowledge is required and a solid arrow means that extensive knowledge is required. Introduction and Foundations should be read before all other chapters and Implementation and Documentation can be read independently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

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.

Readers interested in Computational Geometry can continue with either the chapter on graphs or the chapter on geometry kernels. Both chapters 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, Meh84, Tar83, CLR90, O'R94, Woo93, Sed91, Kin90, van88, NH93, PS85, BY98, dBKOS97] 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 the reader 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 `-DLEDA_CHECKING_OFF` and `-O`).

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

ledares@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. Holger Blaar, Stefan Funke, Gunnar Klau, Volker Priebe, Michael Seel, René Weißkircher, Mark Ziegelmann, and Joachim Ziegler 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 University 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
Halle, Germany
April, 1999

Kurt Mehlhorn
Saarbrücken, Germany
April, 1999

Bibliography

- [AHU83] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1983.
- [BY98] J.-D. Boissonnat and M. Yvinec. *Algorithmic Geometry*. Cambridge University Press, Cambridge, 1998.
- [CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press/McGraw-Hill Book Company, 1990.
- [dBKOS97] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, 1997.
- [Kin90] J.H. Kingston. *Algorithms and Data Structures*. Addison-Wesley, 1990.
- [Kle97] R. Klein. *Algorithmische Geometrie*. Addison-Wesley, 1997.
- [LL98] S.B. Lippmann and J. Lajoie. *C++ Primer*. Addison-Wesley, 1998.
- [Meh84] K. Mehlhorn. *Data Structures and Algorithms 1, 2, and 3*. Springer, 1984.
- [Mur93] R.B. Murray. *C++ Strategies and Tactics*. Addison-Wesley, 1993.
- [NH93] J. Nievergelt and K.H. Hinrichs. *Algorithms and Data Structures*. Prentice Hall, 1993.
- [O'R94] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1994.
- [OW96] T. Ottmann and P. Widmayer. *Algorithmen und Datenstrukturen*. Spektrum Akademischer Verlag, 1996.
- [PS85] F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction*. Springer, 1985.
- [Sed91] R. Sedgewick. *Algorithms*. Addison-Wesley, 1991.
- [Str91] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, 1991.
- [Tar83] R.E. Tarjan. Data structures and network algorithms. In *CBMS-NSF Regional Conference Series in Applied Mathematics*, volume 44, 1983.
- [van88] C.J. van Wyk. *Data Structures and C Programs*. Addison-Wesley, 1988.
- [Woo93] D. Wood. *Data Structures, Algorithms, and Performance*. Addison-Wesley, 1993.