# The Audio Programming Book Mit Press

The process of user-centered innovation: how it can benefit both users and manufacturers and how its emergence will bring changes in business models and in public policy. Innovation is rapidly becoming democratized. Users, aided by improvements in computer and communications technology, increasingly can develop their own new products and services. These innovating users—both individuals and firms—often freely share their innovations with others, creating user-innovation communities and a rich intellectual commons. In Democratizing Innovation, Eric von Hippel looks closely at this emerging system of user-centered innovation. He explains why and when users find it profitable to develop new products and services for themselves, and why it often pays users to reveal their innovations freely for the use of all.The trend toward democratized innovation can be seen in software and information products—most notably in the free and open-source software movement—but also in physical products. Von Hippel's many examples of user innovation in action range from surgical equipment to surfboards to software security features. He shows that product and service development is concentrated among "lead users," who are ahead on marketplace trends and whose innovations are often commercially attractive. Von Hippel argues that manufacturers should redesign their innovation processes and that they should systematically seek out innovations developed by users. He points to businesses—the custom semiconductor industry is one example—that have learned to assist user-innovators by providing them with toolkits for developing new products. User innovation has a positive impact on social welfare, and von Hippel proposes that government policies, including R&D subsidies and tax credits, should be realigned to eliminate biases against it. The goal of a democratized user-centered innovation system, says von Hippel, is well worth striving for. An electronic version of this book is available under a Creative Commons license.

Want to turn your mobile device into a musical instrument? Or equip your game with interactive audio, rather than canned samples? You can do it with Pure Data (Pd), an open source visual programming environment that lets you manipulate digital audio in real time. This concise book shows you how to use Pd—with help from the libpd library—as an easily embeddable and widely portable sound engine. Whether you're an audio developer looking to create musical apps with sophisticated audio capabilities, or an application developer ready to enhance mobile games with real-time procedural audio, Making Musical Apps introduces you to Pd and libpd, and provides hands-on instructions for creating musical apps for Android and iOS. Get a crash course in Pd, and discover how to generate and control sounds Learn how to create and deploy algorithmic compositions that react to a user's activity and environment Use Java or Objective-C to integrate Pd and libpd into mobile apps Learn the steps necessary to build libpd-based apps for Android and iOS

The essential reference to SuperCollider, a powerful, flexible, open-source, cross-platform audio programming language. SuperCollider is one of the most important domain-specific audio programming languages, with potential applications that include real-time interaction, installations, electroacoustic pieces, generative music, and audiovisuals. The SuperCollider Book is the essential reference to this powerful and flexible language, offering students and professionals a collection of tutorials, essays, and projects. With contributions from top academics, artists, and technologists that cover topics at levels from the introductory to the specialized, it will be a valuable sourcebook both for beginners and for advanced users. SuperCollider, first developed by James McCartney, is an accessible blend of Smalltalk, C, and further ideas from a number of programming languages. Free, open-source, cross-platform, and with a diverse and supportive developer community, it is often the first programming language sound artists and computer musicians learn. The SuperCollider Book is the long-awaited guide to the design, syntax, and use of the SuperCollider language. The first chapters offer an introduction to the basics, including a friendly tutorial for absolute beginners, providing the reader with skills that can serve as a foundation for further learning. Later chapters cover more advanced topics and particular topics in computer music, including programming, sonification, spatialization, microsound, GUIs, machine listening, alternative tunings, and non-real-time synthesis; practical applications and philosophical insights from the composer's and artist's perspectives; and "under the hood," developer's-eye views of SuperCollider's inner workings. A Web site accompanying the book offers code, links to the application itself and its source code, and a variety of third-party extras, extensions, libraries, and examples.

This book is a standard tutorial targeted at game developers which aims to help them incorporate audio programming techniques to enhance their gameplay experience.This book is perfect for C++ game developers who have no experience with audio programming and who would like a quick introduction to the most important topics required to integrate audio into a game.

A comprehensive text and reference that covers all aspects of computer music, including digital audio, synthesis techniques, signal processing, musical input devices, performance software, editing systems, algorithmic composition, MIDI, synthesizer architecture, system interconnection, and psychoacoustics. The Computer Music Tutorial is a comprehensive text and reference that covers all aspects of computer music, including digital audio, synthesis techniques, signal processing, musical input devices, performance software, editing systems, algorithmic composition, MIDI, synthesizer architecture, system interconnection, and psychoacoustics. A special effort has been made to impart an appreciation for the rich history behind current activities in the field. Profusely illustrated and exhaustively referenced and cross-referenced, The Computer Music Tutorial provides a step-by-step introduction to the entire field of computer music techniques. Written for nontechnical as well as technical readers, it uses hundreds of charts, diagrams, screen images, and photographs as well as clear explanations to present basic concepts and terms. Mathematical notation and program code examples are used only when absolutely necessary. Explanations are not tied to any specific software or hardware. The material in this book was compiled and refined over a period of several years of

teaching in classes at Harvard University, Oberlin Conservatory, the University of Naples, IRCAM, Les Ateliers UPIC, and in seminars and workshops in North America, Europe, and Asia.

An encyclopedic handbook on audio programming for students and professionals, with many cross-platform open source examples and a DVD covering advanced topics. This comprehensive handbook of mathematical and programming techniques for audio signal processing will be an essential reference for all computer musicians, computer scientists, engineers, and anyone interested in audio. Designed to be used by readers with varying levels of programming expertise, it not only provides the foundations for music and audio development but also tackles issues that sometimes remain mysterious even to experienced software designers. Exercises and copious examples (all cross-platform and based on free or open source software) make the book ideal for classroom use. Fifteen chapters and eight appendixes cover such topics as programming basics for C and C++ (with music-oriented examples), audio programming basics and more advanced topics, spectral audio programming; programming Csound opcodes, and algorithmic synthesis and music programming. Appendixes cover topics in compiling, audio and MIDI, computing, and math. An accompanying DVD provides an additional 40 chapters, covering musical and audio programs with micro-controllers, alternate MIDI controllers, video controllers, developing Apple Audio Unit plug-ins from Csound opcodes, and audio programming for the iPhone. The sections and chapters of the book are arranged progressively and topics can be followed from chapter to chapter and from section to section. At the same time, each section can stand alone as a self-contained unit. Readers will find The Audio Programming Book a trustworthy companion on their journey through making music and programming audio on modern computers.

Sound Synthesis and Sampling' provides a comprehensive introduction to the underlying principles and practical techniques applied to both commercial and research sound synthesizers. This new edition has been updated throughout to reflect current needs and practices- revised and placed in a modern context, providing a guide to the theory of sound and sampling in the context of software and hardware that enables sound making. For the revised edition emphasis is on expanding explanations of software and computers, new sections include techniques for making sound physically, sections within analog and digital electronics. Martin Russ is well known and the book praised for its highly readable and non-mathematical approach making the subject accessible to readers starting out on computer music courses or those working in a studio.

A book for anyone who wants to learn programming to explore and create, with exercises and projects to help the reader learn by doing. This book introduces programming to readers with a background in the arts and humanities; there are no prerequisites, and no knowledge of computation is assumed. In it, Nick Montfort reveals programming to be not merely a technical exercise within given constraints but a tool for sketching, brainstorming, and inquiring about important topics. He emphasizes programming's exploratory potential—its facility to create new kinds of artworks and to probe data for new ideas. The book is designed to be read alongside the computer, allowing readers to program while making their way through the chapters. It offers practical exercises in writing and modifying code, beginning on a small scale and increasing in substance. In some cases, a specification is given for a program, but the core activities are a series of "free projects," intentionally underspecified exercises that leave room for readers to determine their own direction and write different sorts of programs. Throughout the book, Montfort also considers how computation and programming are culturally situated—how programming relates to the methods and questions of the arts and humanities. The book uses Python and Processing, both of which are free software, as the primary programming languages.

This book provides an introduction to Bluetooth programming, with a specific focus on developing real code. The authors discuss the major concepts and techniques involved in Bluetooth programming, with special emphasis on how they relate to other networking technologies. They provide specific descriptions and examples for creating applications in a number of programming languages and environments including Python, C, Java, GNU/Linux, Windows XP, Symbian Series 60, and Mac OS X. No previous experience with Bluetooth is assumed, and the material is suitable for anyone with some programming background. The authors place special emphasis on the essential concepts and techniques of Bluetooth programming, starting simply and allowing the reader to quickly master the basic concepts before addressing advanced features.

Originally developed by James McCartney in 1996 and now an open source project, SuperCollider is a software package for the synthesis and control of audio in real time. Currently, it represents the state of the art in the field of audio programming: there is no other software available that is equally powerful, efficient or flexible. Yet, SuperCollider is often approached with suspicion or awe by novices, but why? One of the main reasons is the use of a textual user interface. Furthermore, like most software packages that deal with audio, SuperCollider prerequisites a series of skills, ranging from expertise in analog/digital signal processing, to musical composition, to computer science. However, as the beginner overcomes these initial obstacles and understands the powerful flexibility of SuperCollider, what once were seen as weaknesses become its strengths. SuperCollider's features also mean versatility in advanced software applications, generality in terms of computer modelling, and expressivity in terms of symbolic representations. This book aims at providing a brief overview of, and an introduction to, the SuperCollider programming environment. It also intends to informally present, by employing SuperCollider, a series of key notions relevant to what is broadly referred to as computer music. Andrea Valle is a researcher/aggregate professor in film, photography and television at the University of Turin-DAMS, and is active as a musician and composer. He has been a SuperCollider user since 2005.

The new edition of an introductory text that teaches students the art of computational problem solving, covering topics ranging from simple algorithms to information visualization. This book introduces students with little or no prior programming experience to the art of computational problem solving using Python and various Python libraries, including PyLab. It provides students with skills that will enable them to make productive use of computational techniques, including some of the tools and techniques of data science for using computation to model and interpret data. The book is based on an MIT course (which became the most popular course offered through MIT's OpenCourseWare) and was developed for use not only in a conventional classroom

but in in a massive open online course (MOOC). This new edition has been updated for Python 3, reorganized to make it easier to use for courses that cover only a subset of the material, and offers additional material including five new chapters. Students are introduced to Python and the basics of programming in the context of such computational concepts and techniques as exhaustive enumeration, bisection search, and efficient approximation algorithms. Although it covers such traditional topics as computational complexity and simple algorithms, the book focuses on a wide range of topics not found in most introductory texts, including information visualization, simulations to model randomness, computational techniques to understand data, and statistical techniques that inform (and misinform) as well as two related but relatively advanced topics: optimization problems and dynamic programming. This edition offers expanded material on statistics and machine learning and new chapters on Frequentist and Bayesian statistics.

Structure and Interpretation of Computer Programs has had a dramatic impact on computer science curricula over the past decade. This long-awaited revision contains changes throughout the text. There are new implementations of most of the major programming systems in the book, including the interpreters and compilers, and the authors have incorporated many small changes that reflect their experience teaching the course at MIT since the first edition was published. A new theme has been introduced that emphasizes the central role played by different approaches to dealing with time in computational models: objects with state, concurrent programming, functional programming and lazy evaluation, and nondeterministic programming. There are new example sections on higher-order procedures in graphics and on applications of stream processing in numerical programming, and many new exercises. In addition, all the programs have been reworked to run in any Scheme implementation that adheres to the IEEE standard.

Created in 1985 by Barry Vercoe, Csound is one of the most widely used software sound synthesis systems. Because it is so powerful, mastering Csound can take a good deal of time and effort. But this long-awaited guide will dramatically straighten the learning curve and enable musicians to take advantage of this rich computer technology available for creating music. Written by the world's leading educators, programmers, sound designers, and composers, this comprehensive guide covers both the basics of Csound and the theoretical and musical concepts necessary to use the program effectively. The thirty-two tutorial chapters cover: additive, subtractive, FM, AM, FOF, granular, wavetable, waveguide, vector, LA, and other hybrid methods; analysis and resynthesis using ADSYN, LP, and the Phase Vocoder; sample processing; mathematical and physical modeling; and digital signal processing, including room simulation and 3D modeling. CDs for this book are no longer produced. To request files, please email digitalproducts-cs@mit.edu.

An introduction to the engineering principles of embedded systems, with a focus on modeling, design, and analysis of cyber-physical systems. The most visible use of computers and software is processing information for human consumption. The vast majority of computers in use, however, are much less visible. They run the engine, brakes, seatbelts, airbag, and audio system in your car. They digitally encode your voice and construct a radio signal to send it from your cell phone to a base station. They command robots on a factory floor, power generation in a power plant, processes in a chemical plant, and traffic lights in a city. These less visible computers are called embedded systems, and the software they run is called embedded software. The principal challenges in designing and analyzing embedded systems stem from their interaction with physical processes. This book takes a cyber-physical approach to embedded systems, introducing the engineering concepts underlying embedded systems as a technology and as a subject of study. The focus is on modeling, design, and analysis of cyber-physical systems, which integrate computation, networking, and physical processes. The second edition offers two new chapters, several new exercises, and other improvements. The book can be used as a textbook at the advanced undergraduate or introductory graduate level and as a professional reference for practicing engineers and computer scientists. Readers should have some familiarity with machine structures, computer programming, basic discrete mathematics and algorithms, and signals and systems.

A distinguishing feature of video games is their interactivity, and sound plays an important role in this: a player's actions can trigger dialogue, sound effects, ambient sound, and music. This book introduces readers to the various aspects of game audio, from its development in early games to theoretical discussions of immersion and realism.

Numerical Algorithms: Methods for Computer Vision, Machine Learning, and Graphics presents a new approach to numerical analysis for modern computer scientists. Using examples from a broad base of computational tasks, including data processing, computational photography, and animation, the textbook introduces numerical modeling and algorithmic desig

This title gives students an integrated and rigorous picture of applied computer science, as it comes to play in the construction of a simple yet powerful computer system.

A single line of code offers a way to understand the cultural context of computing. This book takes a single line of code—the extremely concise BASIC program for the Commodore 64 inscribed in the title—and uses it as a lens through which to consider the phenomenon of creative computing and the way computer programs exist in culture. The authors of this collaboratively written book treat code not as merely functional but as a text—in the case of 10 PRINT, a text that appeared in many different printed sources—that yields a story about its making, its purpose, its assumptions, and more. They consider randomness and regularity in computing and art, the maze in culture, the popular BASIC programming language, and the highly influential Commodore 64 computer.

This textbook provides both profound technological knowledge and a comprehensive treatment of essential topics in music processing and music information retrieval. Including numerous examples, figures, and exercises, this book is suited for students, lecturers, and researchers working in audio engineering, computer science, multimedia, and musicology. The book consists of eight chapters. The first two cover foundations of music representations and the Fourier transform—concepts that are then used throughout the book. In the subsequent chapters, concrete music processing tasks serve as a starting point. Each of these chapters is organized in a similar fashion and starts with a general description of the music processing scenario at hand before integrating it into a wider context. It then discusses—in a mathematically rigorous way—important techniques and algorithms that are generally applicable to a wide range of analysis, classification, and retrieval problems. At the same time, the techniques are directly applied to a specific music processing task. By mixing theory and practice, the book's goal is to offer detailed technological insights as well as a deep understanding of music processing applications. Each chapter ends with a section that includes links to the research literature, suggestions for further reading, a list of references, and exercises. The chapters are organized in a

modular fashion, thus offering lecturers and readers many ways to choose, rearrange or supplement the material. Accordingly, selected chapters or individual sections can easily be integrated into courses on general multimedia, information science, signal processing, music informatics, or the digital humanities.

How Britain lost its early dominance in computing by systematically discriminating against its most qualified workers: women. In 1944, Britain led the world in electronic computing. By 1974, the British computer industry was all but extinct. What happened in the intervening thirty years holds lessons for all postindustrial superpowers. As Britain struggled to use technology to retain its global power, the nation's inability to manage its technical labor force hobbled its transition into the information age. In Programmed Inequality, Mar Hicks explores the story of labor feminization and gendered technocracy that undercut British efforts to computerize. That failure sprang from the government's systematic neglect of its largest trained technical workforce simply because they were women. Women were a hidden engine of growth in high technology from World War II to the 1960s. As computing experienced a gender flip, becoming male-identified in the 1960s and 1970s, labor problems grew into structural ones and gender discrimination caused the nation's largest computer user—the civil service and sprawling public sector—to make decisions that were disastrous for the British computer industry and the nation as a whole. Drawing on recently opened government files, personal interviews, and the archives of major British computer companies, Programmed Inequality takes aim at the fiction of technological meritocracy. Hicks explains why, even today, possessing technical skill is not enough to ensure that women will rise to the top in science and technology fields. Programmed Inequality shows how the disappearance of women from the field had grave macroeconomic consequences for Britain, and why the United States risks repeating those errors in the twenty-first century.

A commonsense, self-contained introduction to the mathematics and physics of music; essential reading for musicians, music engineers, and anyone interested in the intersection of art and science. "Mathematics can be as effortless as humming a tune, if you know the tune," writes Gareth Loy. In Musimathics, Loy teaches us the tune, providing a friendly and spirited tour of the mathematics of music—a commonsense, self-contained introduction for the nonspecialist reader. It is designed for musicians who find their art increasingly mediated by technology, and for anyone who is interested in the intersection of art and science. In Volume 1, Loy presents the materials of music (notes, intervals, and scales); the physical properties of music (frequency, amplitude, duration, and timbre); the perception of music and sound (how we hear); and music composition. Calling himself "a composer seduced into mathematics," Loy provides answers to foundational questions about the mathematics of music accessibly yet rigorously. The examples given are all practical problems in music and audio. Additional material can be found at http://www.musimathics.com.

Summary Programming for Musicians and Digital Artists: Creating Music with ChucK offers a complete introduction to programming in the open source music language ChucK. In it, you'll learn the basics of digital sound creation and manipulation while you discover the ChucK language. As you move example-by-example through this easy-to-follow book, you'll create meaningful and rewarding digital compositions and "instruments" that make sound and music in direct response to program logic, scores, gestures, and other systems connected via MIDI or the network. Purchase of the print book includes a free eBook in PDF, Kindle, and ePub formats from Manning Publications. About this Book A digital musician must manipulate sound precisely. ChucK is an audio-centric programming language that provides precise control over time, audio computation, and user interface elements like track pads and joysticks. Because it uses the vocabulary of sound, ChucK is easy to learn even for artists with little or no exposure to computer programming. Programming for Musicians and Digital Artists offers a complete introduction to music programming. In it, you'll learn the basics of digital sound manipulation while you learn to program using ChucK. Example-by-example, you'll create meaningful digital compositions and "instruments" that respond to program logic, scores, gestures, and other systems connected via MIDI or the network. You'll also experience how ChucK enables the on-the-fly musical improvisation practiced by communities of "live music coders" around the world. Written for readers familiar with the vocabulary of sound and music. No experience with computer programming is required. What's Inside Learn ChucK and digital music creation side-by-side Invent new sounds, instruments, and modes of performance Written by the creators of the ChucK language About the Authors Perry Cook, Ajay Kapur, Spencer Salazar, and Ge Wang are pioneers in the area of teaching and programming digital music. Ge is the creator and chief architect of the ChucK language. Table of Contents Introduction: ChucK programming for artistsPART 1 INTRODUCTION TO PROGRAMMING IN CHUCK Basics: sound, waves, and ChucK programming Libraries: ChucK's built-in tools Arrays: arranging and accessing your compositional data Sound files and sound manipulation Functions: making your own tools PART 2 NOW IT GETS REALLY INTERESTING! Unit generators: ChucK objects for sound synthesis and processing Synthesis ToolKit instruments Multithreading and concurrency: running many programs at once Objects and classes: making your own ChucK power tools Events: signaling between shreds and syncing to the outside world Integrating with other systems via MIDI, OSC, serial, and more

"This is Volume 3 in a sequential series of bi-annual volumes, with each volume comprised of 20-25 chapters written by game audio programmers and sound designers. Basic to advanced knowledge of programming and audio integration techniques is presented. One of the goals of this book is to raise the general level of game audio programming expertise, so it is written in a manner that is accessible to beginners, while still providing valuable content for more advanced game audio programmers. The authors of the chapters will have used all of the techniques in shipping games, so readers will learn about techniques that are actually practical, with plenty of code examples and diagrams"--Computers are at the center of almost everything related to audio. Whether for synthesis in music production, recording in the studio, or mixing in live sound, the computer plays an essential part. Audio effects plug-ins and virtual instruments are implemented as software computer code. Music apps are computer programs run on a mobile device. All

these tools are created by programming a computer. Hack Audio: An Introduction to Computer Programming and Digital Signal Processing in MATLAB provides an introduction for musicians and audio engineers interested in computer programming. It is intended for a range of readers including those with years of programming experience and those ready to write their first line of code. In the book, computer programming is used to create audio effects using digital signal processing. By the end of the book, readers implement the following effects: signal gain change, digital summing, tremolo, auto-pan, mid/side processing, stereo widening, distortion, echo, filtering, equalization, multi-band processing, vibrato, chorus, flanger, phaser, pitch shifter, auto-wah, convolution and algorithmic reverb, vocoder, transient designer, compressor, expander, and de-esser. Throughout the book, several types of test signals are synthesized, including: sine wave, square wave, sawtooth wave, triangle wave, impulse train, white noise, and pink noise. Common visualizations for signals and audio effects are created including: waveform, characteristic curve, goniometer, impulse response, step response, frequency spectrum, and spectrogram. In total, over 200 examples are provided with completed code demonstrations.

Accompanying CD-ROM contains complete code for all projects presented in the book. The Max/MSP externals are designed for use with Max 5.

The untold history of women and computing: how pioneering women succeeded in a field shaped by gender biases. Today, women earn a relatively low percentage of computer science degrees and hold proportionately few technical computing jobs. Meanwhile, the stereotype of the male "computer geek" seems to be everywhere in popular culture. Few people know that women were a significant presence in the early decades of computing in both the United States and Britain. Indeed, programming in postwar years was considered woman's work (perhaps in contrast to the more manly task of building the computers themselves). In Recoding Gender, Janet Abbate explores the untold history of women in computer science and programming from the Second World War to the late twentieth century. Demonstrating how gender has shaped the culture of computing, she offers a valuable historical perspective on today's concerns over women's underrepresentation in the field. Abbate describes the experiences of women who worked with the earliest electronic digital computers: Colossus, the wartime codebreaking computer at Bletchley Park outside London, and the American ENIAC, developed to calculate ballistics. She examines postwar methods for recruiting programmers, and the 1960s redefinition of programming as the more masculine "software engineering." She describes the social and business innovations of two early software entrepreneurs, Elsie Shutt and Stephanie Shirley; and she examines the career paths of women in academic computer science. Abbate's account of the bold and creative strategies of women who loved computing work, excelled at it, and forged successful careers will provide inspiration for those working to change gendered computing culture.

"This book introduces you to R, RStudio, and the tidyverse, a collection of R packages designed to work together to make data science fast, fluent, and fun. Suitable for readers with no previous programming experience"--

The complex material histories of the Nintendo Entertainment System platform, from code to silicon, focusing on its technical constraints and its expressive affordances. In the 1987 Nintendo Entertainment System videogame Zelda II: The Adventure of Link, a character famously declared: I AM ERROR. Puzzled players assumed that this cryptic mesage was a programming flaw, but it was actually a clumsy Japanese-English translation of "My Name is Error," a benign programmer's joke. In I AM ERROR Nathan Altice explores the complex material histories of the Nintendo Entertainment System (and its Japanese predecessor, the Family Computer), offering a detailed analysis of its programming and engineering, its expressive affordances, and its cultural significance. Nintendo games were rife with mistranslated texts, but, as Altice explains, Nintendo's translation challenges were not just linguistic but also material, with consequences beyond simple misinterpretation. Emphasizing the technical and material evolution of Nintendo's first cartridge-based platform, Altice describes the development of the Family Computer (or Famicom) and its computational architecture; the "translation" problems faced while adapting the Famicom for the U.S. videogame market as the redesigned Entertainment System; Nintendo's breakthrough console title Super Mario Bros. and its remarkable software innovations; the introduction of Nintendo's short-lived proprietary disk format and the design repercussions on The Legend of Zelda; Nintendo's efforts to extend their console's lifespan through cartridge augmentations; the Famicom's Audio Processing Unit (APU) and its importance for the chiptunes genre; and the emergence of software emulators and the new kinds of play they enabled.

This book is a fast-paced, practical guide full of step-by-step examples which are easy to follow and implement.This book is for programmers with a basic grasp of C++. The examples start at a basic level, making few assumptions beyond fundamental C++ concepts. Those without any experience with C++ should be able to follow and construct the examples, although you may need further support to understand the fundamental concepts.

The professional recording industry is rapidly moving from a hardware paradigm (big studios with expensive gear) to a software paradigm, in which lots of expensive hardware is replaced with a single computer loaded with software plug-ins. Complete albums are now being recorded and engineered "inside the box"-all within a computer without hardware processing or mixing gear. Audio effect plug-ins, which are small software modules that work within audio host applications, like Avid Pro Tools, Apple Logic, Ableton Live, and Steinberg Cubase, are big business. Designing Audio Effect Plug-Ins in C++ gives readers everything they need to know to create real-world, working plug-ins in the widely used C++ programming language. Beginning with the necessary theory behind audio signal processing, author Will Pirkle quickly gets into the heart of this implementation guide, with clearly-presented, previously unpublished algorithms, tons of example code, and practical advice. From the companion website, readers can download free software for the rapid development of the algorithms, many of which have never been revealed to the general public. The resulting plug-ins can be compiled to snap in to any of the above host applications. Readers will come away with the knowledge and tools to design and implement their own audio signal processing designs. Learn to build audio effect plug-ins in a widely used, implementable programming language-C++ Design plug-ins for a variety of platforms (Windows and Mac) and popular audio applications Companion site gives you fully worked-out code for all the examples used, free development software for download, video tutorials for the software, and examples of student plug-ins complete with theory and code

A comprehensive introduction to type systems and programming languages. A type system is a syntactic method for automatically checking the absence of certain erroneous behaviors by

classifying program phrases according to the kinds of values they compute. The study of type systems—and of programming languages from a type-theoretic perspective—has important applications in software engineering, language design, high-performance compilers, and security. This text provides a comprehensive introduction both to type systems in computer science and to the basic theory of programming languages. The approach is pragmatic and operational; each new concept is motivated by programming examples and the more theoretical sections are driven by the needs of implementations. Each chapter is accompanied by numerous exercises and solutions, as well as a running implementation, available via the Web. Dependencies between chapters are explicitly identified, allowing readers to choose a variety of paths through the material. The core topics include the untyped lambda-calculus, simple type systems, type reconstruction, universal and existential polymorphism, subtyping, bounded quantification, recursive types, kinds, and type operators. Extended case studies develop a variety of approaches to modeling the features of object-oriented languages.

Bridging the gap from theory to programming, Designing Software Synthesizer Plug-Ins in C++ For RackAFX, VST3 and Audio Units contains complete code for designing and implementing software synthesizers for both Windows and Mac platforms. You will learn synthesizer operation, starting with the underlying theory of each synthesizer component, and moving on to the theory of how these components combine to form fully working musical instruments that function on a variety of target digital audio workstations (DAWs). Containing some of the latest advances in theory and algorithm development, this book contains information that has never been published in textbook form, including several unique algorithms of the author's own design. The book is broken into three parts: plug-in programming, theory and design of the central synthesizer components of oscillators, envelope generators, and filters, and the design and implementation of six complete polyphonic software synthesizer musical instruments, which can be played in real time. The instruments implement advanced concepts including a user-programmable modulation matrix. The final chapter shows you the theory and code for a suite of delay effects to augment your synthesizers, introducing you to audio effect processing. The companion website, www.focalpress.com/cw/pirkle, gives you access to free software to guide you through the application of concepts discussed in the book, and code for both Windows and Mac platforms. In addition to the software, it features bonus projects, application notes, and video tutorials. A reader forum, monitored by the author, gives you the opportunity for questions and information exchange.

A practitioner's guide to the basic principles of creating sound effects using easily accessed free software. Designing Sound teaches students and professional sound designers to understand and create sound effects starting from nothing. Its thesis is that any sound can be generated from first principles, guided by analysis and synthesis. The text takes a practitioner's perspective, exploring the basic principles of making ordinary, everyday sounds using an easily accessed free software. Readers use the Pure Data (Pd) language to construct sound objects, which are more flexible and useful than recordings. Sound is considered as a process, rather than as data—an approach sometimes known as "procedural audio." Procedural sound is a living sound effect that can run as computer code and be changed in real time according to unpredictable events. Applications include video games, film, animation, and media in which sound is part of an interactive process. The book takes a practical, systematic approach to the subject, teaching by example and providing background information that offers a firm theoretical context for its pragmatic stance. [Many of the examples follow a pattern, beginning with a discussion of the nature and physics of a sound, proceeding through the development of models and the implementation of examples, to the final step of producing a Pure Data program for the desired sound. Different synthesis methods are discussed, analyzed, and refined throughout.] After mastering the techniques presented in Designing Sound, students will be able to build their own sound objects for use in interactive applications and other projects

Describes the LISP programming language, and covers basic procedures, data, and modularity.

Audio can affect the human brain in the most powerful and profound ways. Using Apple's Core Audio, you can leverage all that power in your own Mac and iOS software, implementing features ranging from audio capture to real-time effects, MP3 playback to virtual instruments, web radio to VoIP support. The most sophisticated audio programming system ever created, Core Audio is not simple. In Learning Core Audio , top Mac programming author Chris Adamson and legendary Core Audio expert Kevin Avila fully explain this challenging framework, enabling experienced Mac or iOS programmers to make the most of it. In plain language, Adamson and Avila explain what Core Audio can do, how it works, and how it builds on the natural phenomena of sound and the human language of audio. Next, using crystal-clear code examples, they guide you through recording, playback, format conversion, Audio Units, 3D audio MIDI connectivity, and overcoming unique challenges of Core Audio programming for iOS. Coverage includes: mastering Core Audio's surprising style and conventions; recording and playback with Audio Queue; synthesizing audio; perform effects on audio streams; capturing from the mic; mixing multiple streams; managing file streams; converting formats; creating 3D positional audio; using Core MIDI on the Mac; leveraging your Cocoa and Objective-C expertise in Core Audio's C-based environment, and much more. When you've mastered the "black arts" of Core Audio, you can do some serious magic. This book will transform you from an acolyte into a true Core Audio wizard.

Designing Audio Effect Plugins in C++ presents everything you need to know about digital signal processing in an accessible way. Not just another theory-heavy digital signal processing book, nor another dull build-a-generic-database programming book, this book includes fully worked, downloadable code for dozens of professional audio effect plugins and practically presented algorithms. Sections include the basics of audio signal processing, the anatomy of a plugin, AAX, AU and VST3 programming guides; implementation details; and actual projects and code. More than 50 fully coded C++ audio signal-processing objects are included. Start with an intuitive and practical introduction to the digital signal processing (DSP) theory behind audio plug-ins, and quickly move on to plugin implementation, gain knowledge of algorithms on classical, virtual analog, and wave digital filters, delay, reverb, modulated effects, dynamics processing, pitch shifting, nonlinear processing, sample rate conversion and more. You will then be ready to

design and implement your own unique plugins on any platform and within almost any host program. This new edition is fully updated and improved and presents a plugin core that allows readers to move freely between application programming interfaces and platforms. Readers are expected to have some knowledge of C++ and high school math. Books on music synthesizers explain the theory of music synthesis, or show you how to use an existing synthesizer, but don't cover the practical details of constructing a custom software synthesizer. Likewise, books on digital signal processing describe sound generation in terms of complex equations and leave it up to the reader to solve the practical problems of programming the equations. BasicSynth takes you beyond the theory and shows you how to create a custom synthesizer in software using the C++ programming language. The first part of the book explains the basic computer algorithms used to generate and process sound. Subsequent chapters explain instrument design using actual synthesis instruments. The example instruments are then combined with a text-based scoring system and sequencer to produce a complete working synthesizer. Complete source code to the C++ classes and example programs is available for download from the Internet.

Learning programming with one of "the coolest applications around": algorithmic puzzles ranging from scheduling selfie time to verifying the six degrees of separation hypothesis. This book builds a bridge between the recreational world of algorithmic puzzles (puzzles that can be solved by algorithms) and the pragmatic world of computer programming, teaching readers to program while solving puzzles. Few introductory students want to program for programming's sake. Puzzles are real-world applications that are attention grabbing, intriguing, and easy to describe. Each lesson starts with the description of a puzzle. After a failed attempt or two at solving the puzzle, the reader arrives at an Aha! moment—a search strategy, data structure, or mathematical fact—and the solution presents itself. The solution to the puzzle becomes the specification of the code to be written. Readers will thus know what the code is supposed to do before seeing the code itself. This represents a pedagogical philosophy that decouples understanding the functionality of the code from understanding programming language syntax and semantics. Python syntax and semantics required to understand the code are explained as needed for each puzzle. Readers need only the rudimentary grasp of programming concepts that can be obtained from introductory or AP computer science classes in high school. The book includes more than twenty puzzles and more than seventy programming exercises that vary in difficulty. Many of the puzzles are well known and have appeared in publications and on websites in many variations. They range from scheduling selfie time with celebrities to solving Sudoku problems in seconds to verifying the six degrees of separation hypothesis. The code for selected puzzle solutions is downloadable from the book's website; the code for all puzzle solutions is available to instructors.

Key ideas in programming language design and implementation explained using a simple and concise framework; a comprehensive introduction suitable for use as a textbook or a reference for researchers. Hundreds of programming languages are in use today—scripting languages for Internet commerce, user interface programming tools, spreadsheet macros, page format specification languages, and many others. Designing a programming language is a metaprogramming activity that bears certain similarities to programming in a regular language, with clarity and simplicity even more important than in ordinary programming. This comprehensive text uses a simple and concise framework to teach key ideas in programming language design and implementation. The book's unique approach is based on a family of syntactically simple pedagogical languages that allow students to explore programming language concepts systematically. It takes as premise and starting point the idea that when language behaviors become incredibly complex, the description of the behaviors must be incredibly simple. The book presents a set of tools (a mathematical metalanguage, abstract syntax, operational and denotational semantics) and uses it to explore a comprehensive set of programming language design dimensions, including dynamic semantics (naming, state, control, data), static semantics (types, type reconstruction, polymporphism, effects), and pragmatics (compilation, garbage collection). The many examples and exercises offer students opportunities to apply the foundational ideas explained in the text. Specialized topics and code that implements many of the algorithms and compilation methods in the book can be found on the book's Web site, along with such additional material as a section on concurrency and proofs of the theorems in the text. The book is suitable as a text for an introductory graduate or advanced undergraduate programming languages course; it can also serve as a reference for researchers and practitioners.

Copyright: b1409115257d75bb2c02e96a8eb756f4