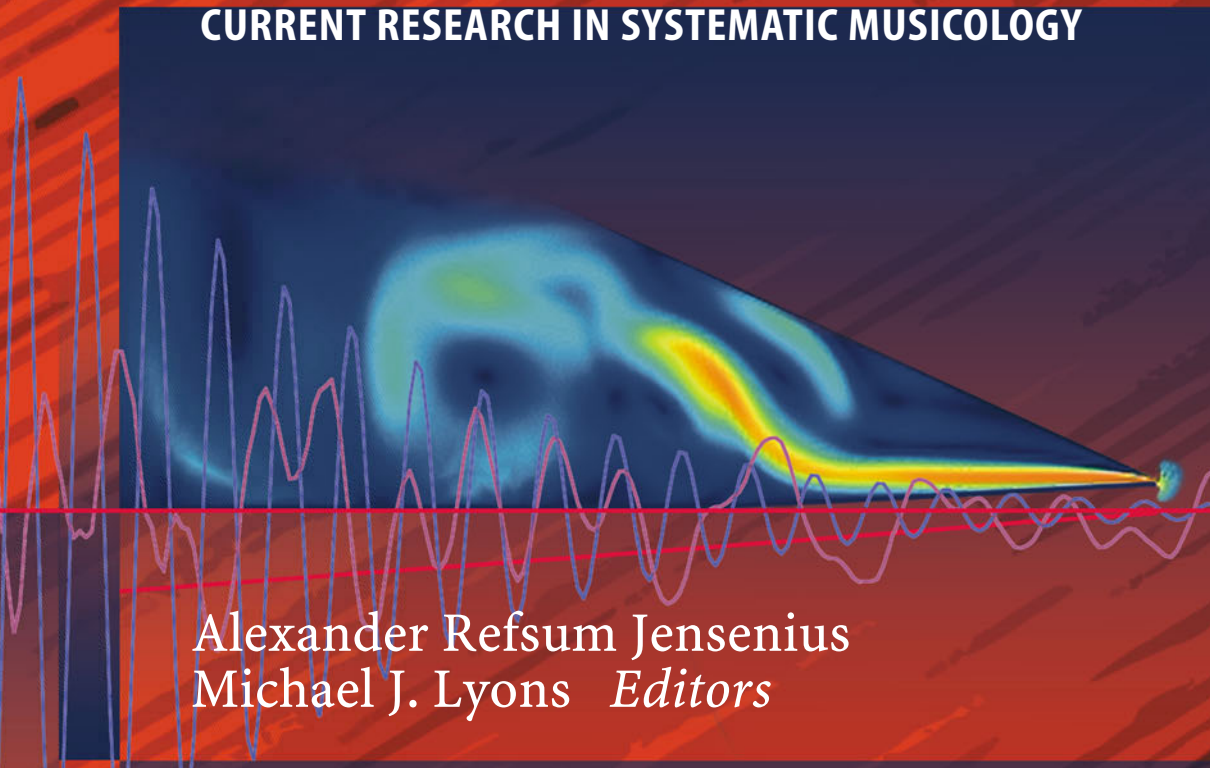
CURRENT RESEARCH IN SYSTEMATIC MUSICOLOGY



Alexander Refsum Jensenius
Michael J. Lyons *Editors*

A NIME Reader

Fifteen Years of New Interfaces for Musical Expression

Springer

Current Research in Systematic Musicology

Volume 3

More information about this series at http://www.springer.com/series/11684

Alexander Refsum Jensenius
Michael J. Lyons
Editors

A NIME Reader

Fifteen Years of New Interfaces for Musical Expression

Springer

Editors
Alexander Refsum Jensenius
Department of Musicology
University of Oslo
Oslo
Norway

Michael J. Lyons
Department of Image Arts and Sciences
Ritsumeikan University
Kyoto
Japan

ISSN 2196-6966 ISSN 2196-6974 (electronic)
Current Research in Systematic Musicology
ISBN 978-3-319-47213-3 ISBN 978-3-319-47214-0 (eBook)
DOI 10.1007/978-3-319-47214-0

Library of Congress Control Number: 2016953639

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

grrrwaaa@yorku.ca

Preface

NIME. A NIME. Two NIMEs. To NIME. Three NIMErs. Newcomers to the field often wonder about what the acronym NIME stands for. While the annual NIME conference is called *International Conference on New Interfaces for Musical Expression*, we like that the acronym may take on several different meanings, such as:

- N = New, Novel, …
- I = Interfaces, Instruments, …
- M = Musical, Multimedial, …
- E = Expression, Exploration, …

A little more than 15 years have passed since the small NIME workshop was held during the ACM Conference on Human Factors in Computing Systems (CHI) in 2001 (Poupyrev et al. 2001b). Already from 2002, NIME was a conference on its own, and today it is an important annual meeting point of researchers, developers, designers, and artists from all over the world. The participants have very different backgrounds, but they all share a mutual passion for groundbreaking music and technology.

More than 1200 papers have been published through the conference so far, and staying true to the open and inclusive atmosphere of the community, all of the papers are freely available online.[1] The archive is great if you know what to look for, but it has grown to a size that is difficult to handle for newcomers to the field. Even for long-timers and occasional visitors, it is difficult to get an overview of the history and development of the community.

At recent editions of the conference, we have seen a growing number of papers focusing on historical, theoretical, and reflective studies of the NIME community (or even communities) itself. As this level of meta-studies started to grow, we began to see the potential for a collection of articles that could broadly represent the conference series. This thought has now materialized in the anthology you are currently holding in your hand, or reading on a screen.

[1] http://www.nime.org/archive/.

Who Is the Book For?

The reader consists of 30 selected papers from the conference series, reflecting the depth and width of the publication archive. Each paper is followed by two new commentaries that add value to the original works while at the same time bring to life some important underlying discussions.

The anthology should be useful for several different groups of readers. Perhaps most importantly, we offer newcomers to the field an overview of some seminal works. They will find a broad range of topics and a large bibliography to continue their explorations outside the collection. We also hope that the book may be a useful reference point for researchers and artists who only "visit" the field and want a quick introduction and reference point to what is being done. Finally, we think that the book is relevant for long-time NIME participants. Such readers may be mostly interested in the following historical traces and emerging critical reflections.

The Selection Process

By nature, an anthology is a limited selection of chapters, and for this project we settled on a limit of 30 articles from the NIME archive. This number was arrived at by estimating the page count of the resulting volume, and it also fitted well with the idea of including approximately two published items for each year of the NIME conference up to and including the most recent edition in 2015.

As expected, selecting 30 papers from 1200 items was not an easy task. How could we possibly fairly represent the energetic and creative output in the field? From the outset, the NIME community has intentionally prioritized a diversity of research styles and approaches. The conference has also striven to offer an environment that can attract the participation not only of researchers working in an academic or institutional context, but also independent artists, researchers and inventors. Moreover, each of us has our own subjective interests and tastes as to what we consider significant prior work.

We started the selection process by creating an initial list of 50 or so articles perceived as "influential" by the community. This was accomplished by identifying the most-cited NIME papers using the Google Scholar index. Naturally, older papers tend to have more citations than new ones, so we also considered the number of citations-per-year. Both measures were used to create a "multi-subjective" initial list of works that have enjoyed impact.

From this first list of well-cited works, we each separately created our lists of papers to include. In many cases we agreed fully, but for others, discussion and sometimes multiple discussions were needed before we reached agreement. At the same time, we found that the initial lists had gaps in the coverage of some topics. For example, work that is primarily artistic in nature is usually not as highly cited as technological reports. This led each of us to propose some works that have not been

cited as highly as the others, but which we believe represent some of the diversity of the NIME corpus. Much discussion was also needed to reach agreement on which of these was to be included in the final list.

Needless to say, we have not been able to include all of the papers that we believe deserve a place in a NIME anthology. The final selection reflects a compromise to respect the limit of just 30 articles. This is also why it is consciously titled "*A* NIME Reader" and not "*The* NIME Reader." We hope that there will be other anthology projects drawing on the extensive NIME proceedings in the future.

Peer-Commentary Process

It was clear from the outset that further peer-review of the articles was not necessary, all of the articles had already undergone peer-review in order to be published at the NIME conference. However, an important feature of this Reader is the inclusion of commentary by the authors themselves as well as other knowledgeable participants in the NIME community (henceforth "experts" or "peers"). Commentators were selected from the group of active conference participants who were perceived to have a special interest in given articles. Authors and peers had a chance to exchange feedback on each other's commentaries. The feedback ranged from simple corrections to enthusiastic discussions of philosophical issues. The peer-commentary procedure, inspired by Stevan Harnad (Harnad 2000), really brought this project to life. The exchange between authors and experts illustrated how the articles continue to live, function, and contribute growth in the research community.

How Can NIME Continue to Be Relevant?

Our hope is that this anthology project will encourage reflection within the community and nurture ideas that have been presented at the conferences. Many NIME papers are fairly terse, with room to present only one, or a few, core ideas of a larger project. This book may provide a starting point for expanding, connecting, and developing new research directions. Originally, NIME was a spin-off from the CHI conference, and recently other events are spawning from NIME. This shows that the field is alive and blossoming, a healthy sign that it will continue to be relevant for another 15 years.

It is important to stress that this collection is not intended to be *the* ultimate NIME paper selection, or a definitive "canon" of relevant works. Going through the archives, we were astounded by the breadth, width, and quality of ideas presented, many of which deserve more attention. We would encourage others to conduct meta-studies, write review papers, and create further anthologies. The archive is large enough to consider more specialized anthologies focusing on,

for example, mobile or web technologies, machine learning, tactility, performance, and pedagogy, to name only a few.

In conclusion, we hope that this collection will be the first of many. After all, the NIME archive is a gold mine of good ideas, and a history of experience and knowledge gained through the dedicated work of many talented researchers and artists. These should not be forgotten but continue to be used in different ways. They may even spark entirely new ways of thinking about NIME itself.

Brisbane, Australia
July 2016

Alexander Refsum Jensenius
Michael J. Lyons

Acknowledgements

We would first and foremost thank all contributors to the NIME conferences throughout the years, and particularly all the chairs for their hard work in maintaining and keeping the community alive.

Contents

Editors and Contributors

About the Editors

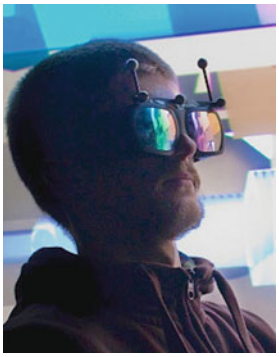
Alexander Refsum Jensenius is a music researcher and research musician working at the Department of Musicology, University of Oslo. He studied at the University of Oslo and Chalmers University of Technology and have been a visiting researcher at CNMAT, UC Berkeley and CIRMMT, McGill University. His research focuses on music-related body motion, from both analytical and creative perspectives. Alexander organized the NIME 2011 conference and is currently chair of the NIME steering committee.

Michael J. Lyons is Professor of Image Arts and Sciences at Ritsumeikan University. He studied physics at McGill University (B.Sc.) and the University of British Columbia (M.Sc., Ph.D.). He has worked in computational neuroscience, pattern recognition, human–computer interaction, and media arts, as a Research Fellow at the California Institute of Technology, Research Assistant Professor at the University of Southern California, and Senior Research Scientist at the ATR Research Labs. Michael proposed and co-organized the CHI 2001 workshop on New Interfaces for Musical Expression, and has served as a founding member of the NIME steering committee and advisory board.

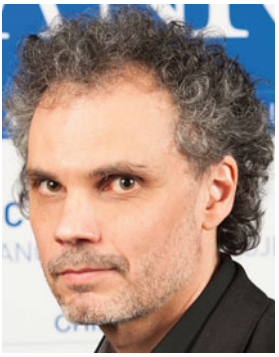
Contributors

Frauke Behrendt is Principal Lecturer in Media Studies at the University of Brighton. Her research interests, publications, and talks consider digital cultures, sound studies, mobility, interaction design, sustainable transport, and smart cities. Behrendt's funded research projects include "Smart e-bikes," "NetPark," and "Sonic Interaction Design." Previously, she worked at the University of Sussex, Rhode Island School of Design and Anglia Ruskin University.

Edgar Berdahl is Assistant Professor in Experimental Music and Digital Media at Louisiana State University. In collaboration with the Cultural Computing Group at the Center for Computation and Technology, he studies how new technology is influencing new music and vice versa. His research interests include open-source software and hardware, haptics, physical modeling sound synthesis, musical acoustics, algorithmic composition, digital signal processing, sound diffusion systems, embedded musical instruments, feedback control of acoustic musical instruments, and sensing.



Florent Berthaut is Assistant Professor at the University of Lille, France and Researcher in the MINT team of CRIStAL. He obtained his Ph.D. in Computer Science from the University of Bordeaux, in the SCRIME/LaBRI and the INRIA Potioc team. He then conducted a 2-year project at the University of Bristol with a Marie Curie fellowship. His research focuses on building connections between 3D user interfaces and new interfaces for musical expression. In particular, he has been exploring 3D interaction techniques adapted to musical interaction and mixed-reality display for augmenting digital instruments on stage.



Frédéric Bevilacqua is Head of the Sound Music Movement Interaction team at IRCAM in Paris. His research concerns the modeling of movement–sound interactions, and the design and development of gesture-based interactive systems. He holds master degree in physics and Ph.D. in Biomedical Optics from EPFL in Lausanne. From 1999 to 2003, he was researcher at the University of California, Irvine. In 2003 he joined IRCAM as a researcher on gesture analysis for music and performing arts.

David Birnbaum is Design Director at Immersion Corporation, and has been creating haptic experiences for over 10 years. He has worked in the fields of user experience, mobile communication, wearables, gaming, medical devices, and rich interactive media. A leading expert in tactile design, he is driven by a desire to tell stories with the sense of touch and to bring emotion and realism to digital experiences. David holds BS in Music Industry from USC and MA in Music Technology from McGill University. He is a named inventor on over 70 patents in the US, Korea, Japan and China.

Tina Blaine is visiting scholar at Carnegie Mellon University's Entertainment Technology Center where she taught for 5 years, developing collective experiences that integrate game design, sonic discovery, and interactive media. Before joining CMU, she worked at Interval Research as a musical interactivist, leading a development team in the creation of the Jam-O-Drum. Blaine is currently the executive director of Rhythmix Cultural Works, a nonprofit community theater/art space in the San Francisco Bay area where people come together to perform, inspire, teach, and interact. She is a cofounder of the NIME conference and served as co-artistic director for the Studio for Electro-Instrumental Music in Amsterdam in 2008.

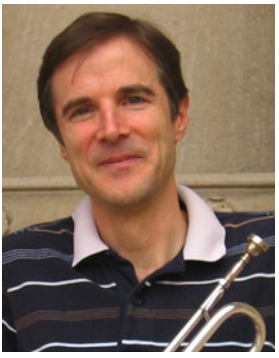
Andrew R. Brown is Professor of Digital Arts at Griffith University in Brisbane, Australia. He is an active computer musician and computational artist. His research interests include digital creativity, computational aesthetics, music education, and the philosophy of technology. He pursues a creative practice in computer-assisted music performance and audiovisual installations, with focus on generative processes and musical live coding. He is author of "Music Technology and Education: Amplifying Musicality," co-author of "Making Music with Computers: Creative Programming in Python," and editor of "Sound Musicianship: Understanding the Crafts of Music."

Nick Bryan-Kinns is Reader in Interaction Design and Deputy Dean at Queen Mary University of London, and Visiting Professor of Interaction Design at Hunan University, China. He leads Interactional Sound and Music at QMUL and has published award winning international journal papers from his multimillion pound funded research. He provided expert opinion for the NSF and European Commission, chaired ACM Creativity and Cognition conference 2009, and BCS-HCI 2006. He is BCS Fellow, recipient of ACM and BCS Recognition of Service Awards, and founding chair of ACM SIGCHI-CCaA Community. Bryan-Kinns is Chartered Engineer and received his Ph.D. in 1998.

Baptiste Caramiaux is a Marie Sklowodska Curie Research Fellow between McGill University and IRCAM. His current research focuses on the understanding of the cognitive processes of motor learning in musical performance and the computational modeling of these processes. Before, he worked on gesture expressivity and the design of musical interactive systems through machine learning. He conducted academic research at Goldsmiths University of London, and applied part of his academic research works on industrial products at Mogees Ltd. Baptiste holds Ph.D. in computer science from University Pierre et Marie Curie in Paris, and IRCAM Centre Pompidou.

Perry R. Cook is Emeritus Professor of Computer Science (also Music) at Princeton University, founding advisor to music app company Smule, and cofounder/Executive VP of Educational Technology Startup Kadenze, Inc. With Dan Trueman, he cofounded the Princeton Laptop Orchestra, which received a MacArthur Digital Learning Initiative Grant in 2005. With Ge Wang, Cook is co-author of the ChucK Programming Language. His newest book is "Programming for Digital Musicians and Artists: An Introduction to ChucK," with Ajay Kapur, Spencer Salazar, and Ge Wang. The recipient of a 2003 Guggenheim Fellowship, Perry lives in Southern Oregon where he researches, writes, composes, farms sunlight, and eats lots of locally grown food.

Palle Dahlstedt is a composer, improviser, pianist, and researcher. In his research on creative performance technologies, firmly rooted in his artistic practice, he has developed novel mapping algorithms for electronic improvisation, augmented hybrid keyboard instruments, and new kinds of computer-mediated interaction models for improvisers. Dahlstedt has performed solo and with the duo pantoMorf all over the world, and his compositions have been played on six continents. In 2001 he received the Gaudeamus Prize. Currently, Dahlstedt is Obel Professor of Art & Technology at Aalborg University, Denmark, and Associate Professor of Computer-Aided Creativity and lecturer in composition at University of Gothenburg. Sweden.



Roger B. Dannenberg is Professor of Computer Science in the Schools of Computer Science, Art, and Music at Carnegie Mellon University. Dannenberg has published pioneering work in the areas of interactive systems, computer accompaniment, computer music languages, and music education. He is the designer of Nyquist, a language for sound synthesis and music composition, cocreator of Audacity, the audio editor, and as Chief Science Officer to Music Prodigy, the developer of polyphonic pitch recognition software. Dannenberg is also a trumpet player and composer, writing and performing computer and experimental music.

Marco Donnarumma is a performance artist and scholar. In over a decade of practice, Donnarumma has developed deeply transdisciplinary research on body, sound, technology and performance, drawing equally from live art, hard science and cultural studies. He holds a Ph.D. in computing from Goldsmiths, University of London, and is currently a Research Fellow at the Berlin University of the Arts and Neurorobotics Research Lab Berlin. He invented the biophysical instrument XTH Sense, which was awarded the first prize in the Guthman New Musical Instrument Competition (2012). His works toured leading events in over 60 countries and his writings are published widely.

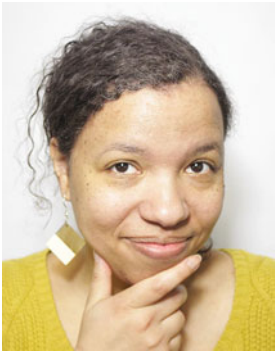
Georg Essl is Visiting Research Professor at the University of Wisconsin - Milwaukee. He received his Ph.D. from Princeton University in 2002 working with Perry Cook on real-time sound synthesis method for solid objects. After a year on the faculty of University of Florida, he worked at MIT Media Lab Europe with Sile O'Modhrain on tangible user interfaces. Then, at Deutsche Telekom Laboratories, he started to turn mobile phones into musical instruments. He then served on the faculty of the University of Michigan. He is founding director of a number of Mobile Phone Orchestras and Ensembles including Stanford, Berlin, and Michigan.

Sid Fels is a Full Professor in Electrical and Computer Engineering at the University of British Columbia (1998–). He was recognized as a Distinguished University Scholar in 2004. He was a visiting researcher at ATR Media Integration and Communications Research Laboratories in Kyoto, Japan (1996–1997). He is internationally known for his work in human–computer interaction, biomechanical modelling, new interfaces for musical expression and interactive arts with over 250 scholarly publications. Fels leads collaborative research projects on 3D biomechanical modelling of speech, mastication and swallowing. He is one of the cofounders of NIME.

Rebecca Fiebrink is Lecturer at Goldsmiths, University of London. Her research focuses on designing new ways for humans to interact with computers in creative practice, and she is the developer of the Wekinator software for interactive machine learning. She has worked with companies including Microsoft Research, Sun Microsystems Research Labs, Imagine Research, and Smule, where she helped to build the #1 iTunes app "I am T-Pain." She holds Ph.D. in Computer Science from Princeton University. Prior to moving to Goldsmiths, she was an assistant professor at Princeton University, where she codirected the Princeton Laptop Orchestra.

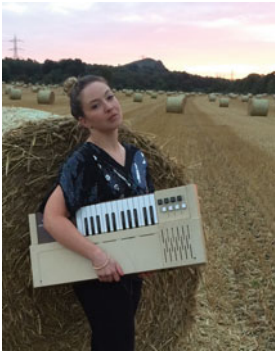
Emmanuel Fléty is an electronic engineer at IRCAM in Paris. He leads the hardware design for both research and art applications, focusing on wireless sensor technologies and gestural interaction. He holds a degree in electronic engineering from ENSEA and joined IRCAM in 1999 to develop stage interaction for performing arts.

Adrian Freed is Research Director of UC Berkeley's Center for New Music and Audio Technologies (CNMAT). He has pioneered many new applications of mathematics, electronics, and computer science to audio, music, and media production tools including the first graphical user interfaces for digital sound editing, mixing, and processing. He invented the audio plug-in, and coinvented Open Sound Control. (OSC). His current work involves movement and intermedia, gesture signal processing, and terra-scale integration of data from wearable and built environment sensor/actuator systems using printed electronics, electro-textiles, and other emerging materials.



Lalya Gaye is a digital media artist and interaction designer based in Newcastle upon Tyne, UK. She is the founder and director of digital media art practice Attaya Projects. Previously she worked as a researcher in human-computer interaction (in particular locative audio and mobile music) at Future Applications Lab, Viktoria Institute, in Gothenburg, Sweden, and at Culture Lab Newcastle, UK. She was also a visiting professor at RISD Digital+Media, USA. Her research and art practice focus on creative potentials at the interface between the physical and the digital in everyday public spaces. Her work has been featured internationally.

Michael Gurevich is Assistant Professor of Performing Arts Technology at the University of Michigan's School of Music, Theatre and Dance. His research explores new aesthetic and interactional possibilities that can emerge in performance with real-time computer systems. Other research areas include network-based music performance, computational acoustic modeling of bioacoustic systems, and historical aspects of computer music. His creative practice explores many of the same themes, through experimental compositions involving interactive media, sound installations, and the design of new musical interfaces. At Michigan, he directs an ensemble dedicated to the performance of electronic chamber music.



Lauren Sarah Hayes is a Scottish musician and sound artist who creates physical live electronic music. She is a regular improviser on both augmented piano and analogue/digital hybrid electronics. Her research examines the links between sound and touch and she has developed and performed with bespoke haptic and vibrotactile devices over a prolonged period of creative practice. Her work has been presented globally at festivals and within publications such as Organised Sound and Contemporary Music Review. She is currently Assistant Professor of Sound Studies at Arizona State University and an associate of the New Radiophonic Workshop.

Abram Hindle is assistant professor of Computing Science at the University of Alberta. His research focuses on problems relating to mining software repositories, improving software engineering-oriented information retrieval with contextual information, the impact of software maintenance on software energy consumption, and how software engineering informs computer music. Abram received Ph.D. in computer science from the University of Waterloo.

Lars Erik Holmquist is Professor of Innovation at Northumbria University. He has worked in ubiquitous computing and HCI for over 20 years, including as co-founder of The Mobile Life Centre in Sweden and Principal Scientist at Yahoo! Labs in Silicon Valley. He instigated the Mobile Music workshop series and organized a panel on Mobile Music at the ACM SIGGRAPH conference. He recently spent two years in Japan, where he was a Guest Researcher at the University of Tokyo, learned Japanese, wrote a novel about augmented reality and played in the garage punk band Fuzz Things.

Andy Hunt is Professor and Deputy Head (Teaching and Scholarship) in the Department of Electronics at the University of York UK. He worked on York's first Music Technology courses in 1988 and has developed the teaching there since that date. His special interests include the use of sound to portray data (sonification), and the human–computer interface for interactive performance instruments. Over the last decade, Andy has worked with Thomas Hermann and Roberto Bresin to push forward the field of interactive sonification, which is a blend of the two interests (HCI and auditory display). He is co-editor of "The Sonification Handbook."

Enrike Hurtado is a researcher at the University of Bilbao, and is currently exploring the digitalisation of the txalaparta, an ancient basque percussion tradition. Enrike's background is in fine arts and he has been strongly involved in experimental music working with improvisation, electronic music, and experimental rock. He is cofounder of ixi audio where he has developed experimental software and taught workshops in the creative use of programming. His interest ranges from process-based music to improvisational rule-based systems and live coding.

Andrew Johnston is a researcher, interaction/software designer, and musician based in Sydney, Australia. His work focuses on the design of systems that support experimental, exploratory approaches to live performance, and the experiences of the artists who use them. Andrew is co-director of the Creativity and Cognition Studios, an interdisciplinary research group working at the intersection of creativity and technology. He currently holds the position of Associate Professor in the School of Software, Faculty of Engineering and IT at the University of Technology Sydney.

Sergi Jordà holds BS in Fundamental Physics and Ph. D. in Computer Science and Digital Communication. He is researcher at the Music Technology Group, Universitat Pompeu Fabra, where he directs the Music and Multimodal Interaction Lab. In the early 1980s, after discovering programming, he devoted himself to computer music improvisation. Throughout the 1990s he developed interactive installations and multimedia performances, and re-entered academia by the end of the millennium. His main research interests are in the confluence of HCI, tangible, musical, and physiological interaction. He has received several international awards, including the Prix Ars Electronica Golden Nica (2008).

Wendy Ju is Executive Director for Interaction Design Research at Stanford's Center for Design Research, and an Associate Professor in the Graduate Design Program at the California College of the Arts in San Francisco. Her research is primarily focused on the design of interactive devices, particularly human–robot interaction and autonomous car interfaces. She received her Master's degree from the MIT Media Lab in 2001, and her doctorate from Stanford University in 2008. She is the author of the book "The Design of Implicit Interactions," which is available from Morgan and Claypool.

Ajay Kapur is Director of the Music Technology program (MTIID) at the California Institute of the Arts, as well as the Associate Dean for Research and Development in Digital Arts. He runs a Ph.D. Research Group in Wellington New Zealand called Sonic Engineering Lab for Creative Technology. He is also Founder and CEO of Kadenze, the Creative Arts MOOC. Kapur has published over 100 technical papers and presented lectures across the world on music technology, human–computer interface for artists, robotics for making sound, and modern digital orchestras.

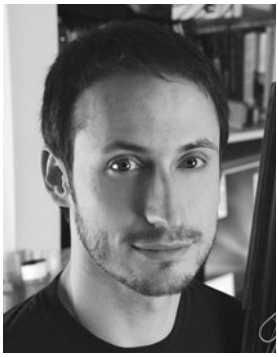
R. Benjamin Knapp is Director of the Institute for Creativity, Arts, and Technology (ICAT) and Professor of Computer Science at Virginia Tech. ICAT seeks to promote research and education at the boundaries between art, design, engineering, and science. For more than 20 years, Dr. Knapp has been working to create meaningful links between human–computer interaction, universal design, and various forms of creativity. His research on human–computer interaction has focused on the development and design of user interfaces and software that allow both composers and performers to augment the physical control of a musical instrument with direct sensory interaction.

Hans Leeuw is a trumpet player, composer, and band leader from the Netherlands. He led and got commissioned (by the Dutch art council) for the 15-piece big band Tetzepi between 1997 and 2014. This band was successful within the Dutch avant garde Jazz scene and toured through the Netherlands as well as Europe. In 2008, Hans started to develop the Electrumpet with which he won the international Guthman competition for new musical instruments in Atlanta in 2013. Hans is a senior Lecturer at the University of the Arts Utrecht since 1993 and started Ph.D. in Huddersfield with Professor Pierre Alexandre Tremblay in 2014.

Eric Lyon is a composer and computer music researcher who focuses on articulated noise, spatial orchestration, and computer chamber music. His software includes FFTease and LyonPotpourri. He is the author of "Designing Audio Objects for Max/MSP and Pd," which explicates the process of designing and implementing audio DSP externals. His music has been selected for the Giga-Hertz prize, MUSLAB, and the ISCM World Music Days. Lyon has taught computer music at Keio University, IAMAS, Dartmouth, Manchester University, and Queens University Belfast. Currently, he teaches at Virginia Tech, and is faculty fellow at the Institute for Creativity, Arts, and Technology.

Andrew McPherson is Reader (Associate Professor) in the Centre for Digital Music at Queen Mary University of London. Composer and electrical engineer by training, he studied at MIT (M.Eng. 2005) and the University of Pennsylvania (Ph.D. 2009) and spent a 2-year postdoctoral fellowship at Drexel University. His research focuses on augmented instruments, embedded hardware systems, the study of performer–instrument interaction, and user appropriation of technology. He is the creator of the magnetic resonator piano, an augmented acoustic piano which has been used by more than 20 composers worldwide, and the TouchKeys multi-touch keyboard which successfully launched on Kickstarter in 2013.

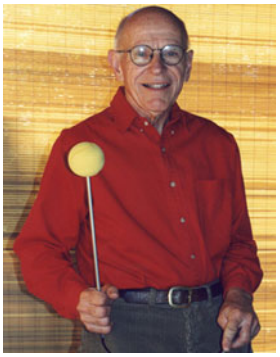
Thor Magnusson is a musician and software developer who lectures in music at the University of Sussex. His research focusses on the impact digital technologies have on musical creativity and practice, explored through software development, composition, and performance. He is cofounder of ixi audio, and has developed audio software, systems of generative music composition, written computer music tutorials, and created two musical live coding environments. As part of ixi, he has taught workshops in creative music coding and sound installations and given presentations, performances, and visiting lectures at diverse art institutions, conservatories, and universities internationally.



Joseph Malloch is a researcher and instrument designer based in Paris at Inria Saclay/Université Paris-Sud/CNRS. His research focuses on the design and use of new interfaces for live music performance, tools for supporting collaborative design of interactive systems, and methods for enriching human–computer interactions. He is also the designer/developer of several digital musical instruments, which have appeared in public performances around the world, including performances at new music festivals, international conferences, performances by dancers, and by blind performers in a larger ensemble.

Adnan Marquez-Borbon is a Mexican improviser, instrument designer, sound artist, and researcher. Currently involved in the interdisciplinary project "Into the Key of Law: Transposing Musical Improvisation. The Case of Child Protection in Northern Ireland." The music released under his name synthesizes electronic music, jazz, free improvisation, and ethnic music. His alter ego, duplex helix, produces beat-based music influenced by Hip Hop and various electronic music genres. He is cofounder of the Mexican improvisation collective Generación Espontánea and the multimedia collective NOR73 (now OpenL4B.Norte Hackerspace).

Charles Martin is a specialist in percussion, computer music, and human–computer interaction from Canberra, Australia. He links percussion with electroacoustic music and other media through new technologies. His works, described as "a thing of rare beauty" in The West Australian, have been performed throughout Australia, Europe, and the USA and presented at international conferences on computer music and percussion. Charles has released touchscreen apps for making music and is active in creating performances, workshops, and research that explore the musical potential of new computing devices.



Max Mathews (1926–2011) was a music technology pioneer and is often called the "father of computer music" after writing the first version of MUSIC in 1957. Several generations of MUSIC-inspired programs such as Csound, Cmix and Max, the latter named for Mathews in the 1980s. He earned his doctorate from MIT in 1954 and worked at Bell Labs until his retirement in 1987, when he joined Stanford's Center for Computer Research in Music and Acoustics (CCRMA). Mathews invented several electronic violins and the Radio Baton, a precursor to modern handheld controllers.

Ali Momeni is a builder, composer, and performer interested in using technology to bring people together. His work makes use of all manners of technology to explore the social lives of objects and their embedded performative qualities, and how computation can give us a more intimate understanding of the world around us. His creative output ranges from kinetic sculptures and sound installations, to urban interventions, music theater performance, and large-scale projection performances and writing. He especially likes working with plants and animals, and his favorite projects are collaborative, transnational, and transgenerational.

Kristian Nymoen is Associate Professor in music technology at the University of Oslo. His research interests covers a range of different areas including development of new musical instruments, motion capture technologies, analysis of music-related body motion, machine learning, and music cognition.

Sile O'Modhrain holds a joint appointment in the School of Information and in the School of Music, Theatre and Dance at the University of Michigan. She received her Ph.D. from Stanford's Center for Computer Research in Music and Acoustics (CCRMA). Her research focuses on human–computer interaction, especially interfaces incorporating haptic and auditory feedback.

Gascia Ouzounian is Associate Professor of Music at the University of Oxford. As musicologist and violinist, her work is situated in relation to experimental music traditions of the last century. Her research examines topics including the history of spatial audio, site-specific sound art, multichannel electroacoustic composition, and the intersection of experimental music and visual arts traditions. Ouzounian's articles appear in many of the leading journals of contemporary music, including Computer Music Journal, Contemporary Music Review, Organised Sound, and Leonardo Music Journal. With the architect Sarah Lappin, she co-directs the interdisciplinary research group Recomposing the City: Sound Art and Urban Architecture.

Dan Overholt is Associate Professor at Aalborg University Copenhagen, Department of Architecture, Design and Media Technology. His research and teaching are centered within the Medialogy, and Sound and Music Computing programs there. Prior, he was a postdoctoral fellow in the Media Arts and Technology (MAT) program and Lecturer / Researcher at the Center for Research in Electronic Art Technology (CREATE) at UC Santa Barbara. His dissertation, entitled "Musical Interface Technology: Multimodal Control of Multidimensional Parameter Spaces for Electroacoustic Music Performance," is the result of a long-term focus on the development of advanced technologies for interactive interfaces, and novel audio signal processing algorithms.

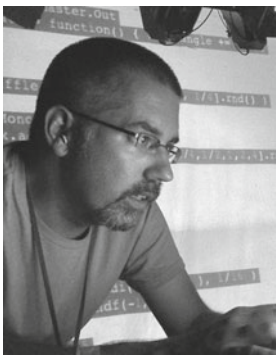
Garth Paine is fascinated with sound as an exhibitable object, enquiries into the materiality of sound and interaction inspired several interactive responsive environments in the 1990s where the inhabitant generated the sonic landscape through presence and behavior. More recent fixed media compositions and interactive music scores for dance, generated through video tracking and or biosensing of the dancers have been presented throughout Australia, Europe, Japan, USA, Asia, and Australia. His work on a NIME taxonomy was one of the initial efforts in the field.

Cléo Palacio-Quintin is a flutist-improviser-composer. Constantly seeking new means of expression and eager to create, she takes part in many premieres as well as improvisational multidisciplinary performances, and composes instrumental and electroacoustic music for various ensembles and media works. Since 1999, she has been developing the hyper-flutes. Interfaced to a computer by means of electronic sensors, these enhanced flutes enable her to compose novel interactive music and control live video processing. She is the first woman to earn Doctorate in Electroacoustic Composition from the Université de Montréal (2012) and collaborates with the Centre for Interdisciplinary Research in Music Media and Technology (CIRMMT).

Matthew Paradis is a senior engineer in the BBC Research & Development Audio Team. He leads an Interactive & Responsive Audio work-stream focusing on the development of new user experiences centered on the application of Object Based Audio. He received his PhD from the University of York studying the importance of parameter mapping for real time musical performance. He is an honorary Research Associate of the University of York, Department of Music and co-chairs the W3C Audio Working group delivering audio processing APIS to the web platform.

Joe Paradiso is Alexander W. Dreyfoos (1954) Professor in Media Arts and Sciences at the MIT Media Lab, where he directs the Responsive Environments group. He received his Ph.D. in Physics from MIT in 1981 and BSEE from Tufts University in 1977, and joined the Media Lab in 1994. He has been designing and building electronic music systems since 1974. His current research explores how sensor networks augment and mediate human experience, interaction, and perception—encompassing wireless sensing systems, wearable and body sensor networks, energy harvesting and power management for embedded sensors, ubiquitous and pervasive computing, human–computer interfaces, and interactive media.

Cornelius Poepel is a violist, audio designer, and sound artist. Since 2008, he is Professor of Audio Producing at Ansbach University of Applied Sciences. He received his Ph.D. from the University of Birmingham, UK working on audio signal-driven sound synthesis. After working at ZKM Karlsruhe, he joined the faculty of the Academy for Media Arts Cologne. He has performed worldwide as orchestraviolist and in experimental performances using the laptop, synthesizer, and acoustic instruments.



Charlie Roberts is Assistant Professor in the School of Interactive Games and Media at the Rochester Institute of Technology, where he researches human-centered computing in digital arts practice. He is the lead designer and developer of *Gibber*, a creative coding environment for the browser, and has performed with it throughout the US, the UK, and Asia in the experimental performance genre known as live coding.

Norbert Schnell is a researcher and developer focussing on real-time interactive digital audio processing and interaction design. Together with his colleagues of the Sound Music Movement Interaction team at IRCAM in Paris, he develops technologies and interaction scenarios on the frontiers between music listening and music performance. In 2006, he chaired the 6th International Conference on New Interfaces for Musical Expression (NIME). He currently coordinates the CoSiMa research project that investigates collaborative musical interaction scenarios using Web and mobile technologies.

Stefania Serafin is Professor with special responsibilities in sound for multimodal environments at Aalborg University Copenhagen. She received Ph.D. degree in computer-based music theory and acoustics from Stanford University, in 2004, and Master in Acoustics, computer science and signal processing applied to music from IRCAM, in 1997. She is president of the Sound and Music Computing Association, and Vice-President for the International Computer Music Association. She has extensively published in numerous international journals and conferences.

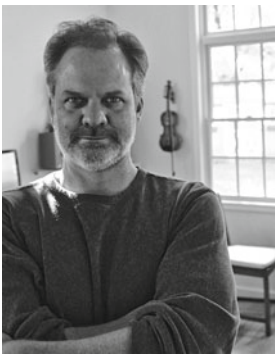
Scott Smallwood is a sound artist, composer, and musician who creates works inspired by discovered textures and forms, through a practice of listening, field recording, and sonic improvisation. He designs experimental electronic instruments and software, as well as sound installations and site-specific performance scenarios. He performs as one-half of the laptop/electronic duo Evidence (with Stephan Moore), and currently lives in Edmonton, Alberta, where he teaches composition, improvisation, and electroacoustic music at the University of Alberta.

Paul Stapleton is an improviser, sound artist, instrument designer, and critical theorist originally from Southern California. He is currently Senior Lecturer at the Sonic Arts Research Centre, Queen's University Belfast, where he teaches and supervises research in performance technologies, improvisation and site-specific sound art. His album FAUNA with saxophonist Simon Rose has received acclaim from critics such as Ken Waxman (Jazzword), Jean-Michel Van Schouwburg (Orynx), Mark Corroto (All About Jazz), and Marc Medwin (New York City Jazz Record). Paul also co-directs the Translating Improvisation research group with Sara Ramshaw and the QUBe music collective with Steve Davis.

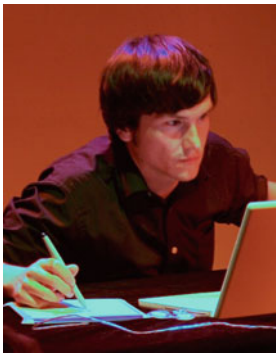
Koray Tahiroğlu is a research fellow, musician, and lecturer in the Department of Media at the Aalto University School of Arts, Design and Architecture, Finland. He is founder and head of the SOPI (Sound and Physical Interaction) research group. He practises art as a researcher focusing on embodied approaches to sound and music interaction, as well as a performer of live electronic music. Since 2004, he has also been teaching workshops and courses introducing artistic strategies and methodologies for creating interactive music. Tahiroğlu has performed experimental music in collaboration as well as in solo performances in Europe and North America.

Atau Tanaka is Professor of Media Computing at Goldsmiths. He has previously been Chair of Digital Media at Newcastle University (UK), and researcher at Sony Computer Science Laboratory (CSL) Paris. During his doctorate at Stanford University's CCRMA, he was awarded the Prix de Paris to conduct research at IRCAM. He has been artistic ambassador for Apple Computer, mentor at NESTA, and Artistic Co-Director of STEIM in Amsterdam. He has performed his work using physiological musical interfaces across Europe, in Japan, and the US. His research has been funded by the RCUK, ANR (France), and the European Research Council.

Jeffrey Treviño is Assistant Professor of Recording and Music Technology at California State University, Monterey Bay. His research and creative practice focus on feedback between symbolic representation, human–computer interaction, and sonic creativity, with an emphasis on the way novel technologies and representations impact established conventions of theory and practice. As a composer, his orchestral, chamber, and solo acoustic and electroacoustic works have been premiered internationally by acclaimed soloists and ensembles. As a music technology researcher and software developer, Treviño contributes code to the nCoda and Abjad Python music notation projects and is a member of the W3C Music Notation Community group.

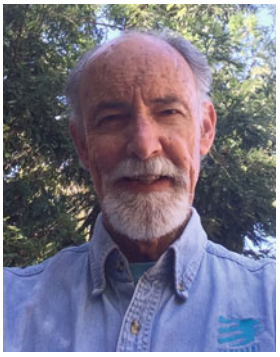


Dan Trueman is a composer, fiddler, and electronic musician. His work has been recognized by the Guggenheim and MacArthur Foundations, the Barlow Endowment, the Fulbright Commission, the American Composers Forum, the American Council of Learned Societies, Meet the Composer, among others. He is Professor of Music and Director of the Princeton Sound Kitchen at Princeton University, where he teaches counterpoint, electronic music, and composition.



Doug Van Nort is an artist, researcher, composer and performer. His work is concerned with questions surrounding distributed agency and sensorial immersion in technologically mediated performance. He creates works that integrate improvisation and collective performance with machine agents, interactive systems and experiences of telepresence. He often performs solo as well as with a wide array of artists across musical styles and artistic media. Van Nort is Canada Research Chair in Digital Performance in the School of Arts, Media, Performance and Design (AMPD) at York University, where he is founding director of the DisPerSion Lab.

Federico Visi is a researcher, composer, and performer. He conducted his doctoral research project on body movement in performance with traditional musical instruments at the Interdisciplinary Centre for Computer Music Research (ICCMR), Plymouth University, UK. He has composed music for films and installations, performed live in solo sets, with bands and in contemporary theater and dance performances and presented his research at several international conferences. He is working on collaborative, interdisciplinary projects with researchers in Europe (Ghent University, University of Bologna), North America (NYU, UCLA) and South America (Universidade Federal do Rio Grande do Sul).

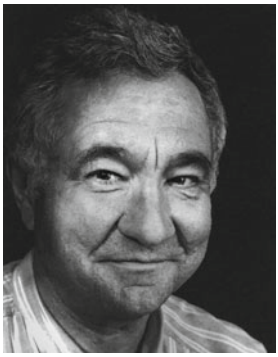


Bill Verplank is Lecturer at Stanford University (CCRMA, CS, ME) where he has taught and done research since retiring from Interval Research (1992–2000), IDTwo (1986–1992) and Xerox (1978–1986). He teaches one or two weeks a year at Copenhagen Institute of Interaction Design, and before that was on the Steering Committee of Interaction Design Institute Ivrea. His education is in man–machine systems (MIT Ph.D. 1971) and mechanical engineering (Stanford BS '65). With Sheridan he pioneered "man–machine systems," Xerox "user-interface design" with Bill Moggridge "interaction design," and with Max Mathews "new interfaces for musical expression."

Graham Wakefield his research-creation is founded upon a transdisciplinary academic training in interactive art, music, mathematics, and philosophy, partnered with an international exhibition record and extensive professional software engineering for creative coding in audiovisual, interactive and immersive virtual and augmented reality media. Currently Assistant Professor in Digital Media, Visual Art & Art History and Canada Research Chair in Interactive Information Visualization at York University, Canada, he previously held research appointments at KAIST, Korea, and in the AlloSphere Research Group at the University of California, Santa Barbara. He is also co-author of the widely used Gen extension for Max/MSP/Jitter.

Marcelo Mortensen Wanderley holds a Ph.D. degree from Université Pierre et Marie Curie (Paris VI), France, on acoustics, signal processing, and computer science applied to music. Prof. Wanderley chaired the 2003 International Conference on New Interfaces for Musical Expression, co-edited the electronic book "Trends in Gestural Control of Music," IRCAM, 2000, and co-authored the textbook, "New Digital Musical Instruments: Control and Interaction Beyond the Keyboard," A-R Editions, 2006. He is William Dawson Scholar and Professor of Music Technology at the Schulich School of Music, McGill University, Montreal, Quebec, Canada.

Ge Wang investigates artful design of interactive music software, programming languages, new computer-mediated instruments, expressive mobile and social music apps, as well as the evolution of laptop orchestras and education at the intersection of computer science and music. He is Assistant Professor at Stanford University's Center for Computer Research in Music and Acoustics (CCRMA), and the cofounder of Smule (reaching over 125 million users). He is the author of the ChucK audio programming language (with Perry Cook), the founding director of the Stanford Laptop Orchestra (SLOrk), and the designer of the iPhone's Ocarina and Magic Piano.



David Wessel (1942–2014) was the founding Director of UC Berkeley's Center for New Music and Audio Technologies (CNMAT) from its 1988 founding until his untimely passing. He performed as a professional jazz drummer in high school, earned BS in mathematical statistics from the University of Illinois in 1964 and Ph.D. in mathematical psychology at Stanford University in 1972, and conducted research and taught at SF State University and Michigan State University before working at IRCAM and founding CNMAT.

Matthew Wright is a computer music researcher, improvising composer/performer, musical ensemble leader, and media systems designer. He is currently Technical Director of Stanford's Center for Computer Research in Music and Acoustics (CCRMA), and was previously Research Director of UC Santa Barbara's Center for Research in Electronic Arts and Technology (CREATE) where his CREATE Ensemble provided a venue, users, and performance opportunities for early versions of Gibber. His research interests include interactive systems, musical rhythm, new interfaces for musical expression, sound synthesis, sonification and visualization, interactive audiovisual system design, musical networking, sound in space, and computational ethnomusicology.

Diana Young received her BS in Electrical Engineering and B.A. in Music from Johns Hopkins University, and her Performer's Certificate in Violin from the Peabody Conservatory of Music. She earned her M.S. and Ph.D. from the MIT Media Laboratory for her research on measurement of bowed string performance parameters. In her postdoctoral work at MIT she contributed to the development of measurement systems for active lower limb prostheses. Most recently, her interests in human motion and the development of HCI for skill acquisition lead her to the Wyss Institute at Harvard as a Technology Development Fellow.

2013: The Web Browser as Synthesizer and Interface

Charles Roberts, Graham Wakefield and Matthew Wright

1 Introduction

Web technologies provide an incredible opportunity to present new musical interfaces to new audiences. Applications written in JavaScript and designed to run in the browser offer remarkable performance, mobile/desktop portability, and longevity due to standardization. As the web browser has matured, the tools available within it to create dynamic musical content have also progressed, and in the last two years realtime and low-level audio programming in the browser has become a reality. Given the browser's ubiquity on both desktop and mobile devices it is arguably the most widely distributed run-time in history, and is rapidly becoming a write once, run anywhere solution for musical interfaces.

Additionally, web apps can now incorporate accelerometers, multitouch screens, gyroscopes, and fully integrated sound synthesis APIs, making web technologies suddenly very attractive for NIMEs. Our research stems from the desire to explore the affordances offered by NIMEs that are web apps and the belief that such instruments should be open and cross-platform. Ideally, a web-based musical interface should be able to run on desktop machines, laptops, and mobile devices regardless of the browser or operating system it is presented in. We believe the libraries described here are a significant step towards realizing this goal.

C. Roberts (✉)
School of Interactive Games & Media, Rochester Institute of Technology, Rochester, NY, USA
e-mail: charlie@charlie-roberts.com

G. Wakefield
School of Arts, Media, Performance & Design, York University, Toronto, Canada
e-mail: grrrwaaa@yorku.ca

M. Wright
Center for Computer Research in Music and Acoustics (CCRMA),
Stanford University, Stanford, CA, USA
e-mail: matt@ccrma.stanford.edu

A.R. Jensenius and M.J. Lyons (eds.), *A NIME Reader*, Current Research in Systematic Musicology 3, DOI 10.1007/978-3-319-47214-0_28

Our research is especially sympathetic to NIME developers wanting to take advantage of web technologies without having to learn the quirks and eccentricities of JavaScript, HTML and CSS simultaneously. We designed the syntax of Gibberish.js and Interface.js so that they can be utilized, both independently or in conjunction, almost entirely using JavaScript alone, requiring a bare minimum of HTML and absolutely no CSS (see Sect. 5). The library downloads include template projects that enable programmers to safely ignore the HTML components and begin coding interfaces and signal processing in JavaScript immediately.

2 Problems and Possibilities of the Web Audio API

In 2011 Google introduced the Web Audio API[1] and incorporated supporting libraries into its Chrome browser. The Web Audio API is an ambitious document; instead of merely defining a basic infrastructure for processing audio callbacks, the API defines how audio graphs should be created and lists a wide variety of unit generators ("ugens") that should be included in browsers as standard. These ugens are native pre-compiled objects that can be assembled into graphs and controlled by JavaScript. For better or for worse, the Mozilla Foundation had previously released a competing API ("Audio Data") (Humphrey et al. 2016) that they integrated into the Firefox browser. This API also provided a well-defined mechanism for writing audio callbacks in JavaScript, however it provided no specifications for lower level, native, pre-compiled ugens to be standardly included in browsers. Fortunately, the Mozilla Foundation recently declared their intention to adopt the Web Audio API, suggesting that it will become the single standard for browser-based audio synthesis. In October of 2012, Apple rolled out iOS 6.0 to their phone and tablet devices and integrated the Web Audio API into mobile Safari; it seems likely to be adopted in other mobile platforms in the future, with promise of longevity through standardization. Our research supports both these APIs and automatically selects the correct one to use.

Although the Web Audio API comes with highly efficient ugens for common audio tasks such as convolution and FFT analysis, there are tradeoffs that accompany its usage. Most relate to the architectural decision to process blocks of at least 256 samples (≈6 ms) instead of performing single-sample processing. This places two notable restrictions on audio programming performed using the Web Audio API: lack of sample-accurate timing and inability to create feedback networks with short delays, which are necessary to important signal-processing applications including filter design and physical modeling.

Fortunately, there is a workaround for these restrictions. The Web Audio API contains a specification for a JavaScript audio node that calculates output using JavaScript callbacks that can be defined at runtime. These guarantee sample-accurate timing and enable complex feedback networks, but at the expense of the efficiency that the native pre-compiled ugens provide. During our experiences developing

[1] https://wiki.mozilla.org/Web_Audio_API.

Gibber (Roberts and Kuchera-Morin 2011), a live coding environment for the browser, we found that existing audio libraries (described in Sect. 6) built for this runtime JavaScript node were not efficient enough to realize the complex synthesis graphs we envisioned and thus began work on our own optimized library, *Gibberish*.

3 Gibberish: An Optimized JavaScript Audio Library

The principal reason for JavaScript's excellent performance in the browser is the use of just-in-time (JIT) compilation: the virtual machine detects the most heavily used functions, path and type-specializations of the code as it runs, and replaces them with translations to native machine code. Although JIT compilation is making waves in the browser today, it can trace a long and diverse history to the earliest days of LISP (Aycock 2003). Now JIT compilers can approach and occasionally even beat the performance of statically compiled C code, though getting the best performance from a JIT compiler may call for very different coding habits than for a static compiler or interpreter.

We began our research by looking at the performance bottlenecks associated with the JavaScript runtime and analyzing what could be done to ameliorate them. The most significant cost we found while working at audio rate in a dynamic runtime is the overhead of object lookup. In complex synthesis graphs, the simple task of resolving object addresses thousands of times per second (and potentially hundreds of thousands of times) levies a substantial cost. Gibberish minimizes this cost by ensuring that all data and functions used within the audio callback are defined and bound within the outer local scope (as "upvalues"), avoiding the need for expensive indexing into externally referenced objects (Herczeg et al. 2009).

3.1 Code Generation

Unfortunately, ensuring the locality of data and procedures for performance is not compatible with the flexibility to dynamically change the ugen graph, since adding or removing ugens implies changing the set of data and procedures that should be considered local to the audio callback. Our solution to this problem, inspired by Wakefield (2012), utilizes run-time code generation.

To optimize the audio callback the Gibberish code generation ("codegen") engine translates the user-defined ugen graph created with object-oriented syntax into a single flat audio callback where all routing and modulation is resolved ahead of execution. This process is basically one of string manipulation. Each ugen is responsible for generating a fragment of code that invokes both its callback function and the callbacks of all ugens that feed into it. Since all inputs to ugens are resolved recursively, requesting the master output bus to perform code generation will effectively flatten the entire audio graph into a linear series of code fragments invoking ugen callbacks.

These code fragments are then concatenated to form a string representing the master audio callback which is then dynamically evaluated into a callable function using JavaScript's `eval()` function.

As a simple example of the codegen algorithm in action, consider the following high-level programming syntax to create a frequency-modulated sine wave fed into a reverb that in turn feeds the default output bus:

```
modulator = new Gibberish.Sine( 4, 50 ); // freq, amp
carrier = new Gibberish.Sine({ amp : .1 });
carrier.frequency = add( 440, modulator );
reverb = new Gibberish.Reverb({ input:carrier });
reverb.connect();
```

Traversing from the master output down through the graph, codegen will occur in the following order: `Bus > Reverb > Carrier > Add > Modulator`. The result is the following callback function, which will be called every sample:

```
function () {
  var v_16 = sine_5( 4, 50 );
  var v_15 = sine_8( 440 + v_16, 0.1 );
  var v_17 = reverb_11( v_15, 0.5, 0.55 );
  var v_4 = bus2_0( v_17, 1, 0 );
  return v_4;
}
```

The `sine_5`, `sine_8`, `reverb_11` and `bus2_0` variables are all upvalues defined immediately outside the scope of the master audio callback and thus resolved inexpensively. These variables refer not to the ugens but to their signal processing functions (aka audio callbacks). The simplistic style of the generated code generally leads to improved run-time performance. For example, passing values by simple function arguments rather than using higher-level object properties and instance variables is ten percent faster in Google Chrome 26.[2]

Note that parametric properties of the ugens are translated as numeric literals (constants) in the generated code. By storing properties as compile-time constants, rather than using dynamic state, expensive calls to continuously index them are entirely avoided. However the implication, which may appear unusual, is that code generation must be invoked whenever a ugen property changes, in addition to when ugens are added or removed. In the above example, if we changed the frequency of our modulator sine wave from four to three Hertz, code generation would be retriggered. To reduce the cost of frequent regeneration, Gibberish caches and re-uses code for all ugens that have not changed. Whenever a ugen property is changed, that ugen is placed inside of a "dirty" array. During codegen each ugen inside the dirty array regenerates the code fragment that invokes its callback function; the code fragments

[2] http://jsperf.com/parameter-lookup-vs-instance-variables.

Table 1 Gibberish ugens by type

Oscillators	Sine, triangle, square, saw, bandlimited saw, bandlimited PWM/square, noise
Effects	Waveshapers, delay, decimator, ring modulation, flanger, vibrato, chorus, reverb, granulator, buffer shuffler/stutterer
Filters	Biquad, state variable, 24 db ladder, one pole
Synths	Synth (oscillator + envelope), Synth2 (oscillator + filter + envelope), FM (2 operator), Monosynth (3 oscillator + envelope + filter)
Math	Add, subtract, multiply, divide, absolute value, square root, pow
Misc	Sampler (record and playback), envelope follower, single sample delay, attack/decay envelope, line/ramp envelope, ADSR envelope, karplus-strong, Bus

for the rest of the ugens remain unaltered. Even if the callback is regenerated dozens or hundreds of times a second it still winds up being fairly efficient as only ugens with property changes invoke their codegen method; all "clean" ugens simply provide the last fragment they generated to be concatenated into the master callback.

3.2 Gibberish Ugens

A variety of ugens come included with Gibberish, listed in Table 1. In addition to standard low-level oscillators such as sine waves and noise generators, Gibberish also provides a number of higher-level synthesis objects that combine oscillators with filters and envelopes. These high-level objects provide a quick way to make interesting sounds using Gibberish and their source code also serves as guidance for programmers interested in writing their own synth definitions.

It is fairly terse for users to add, redefine, or extend a Gibberish ugen at runtime. For example, here is the definition for the Noise ugen:

```
Gibberish.Noise = function() {
  this.name ='noise';
  this.properties = { amp:1 };
  this.callback = function( amp ) {
    return ( Math.random() * 2 - 1  ) * amp;
  };
  this.init();
}
Gibberish.Noise.prototype = Gibberish.Ugen;
```

The `init()` method is inherited from the Ugen prototype and sets up code generation routines for the ugen in addition to performing other initialization tasks. The *properties* dictionary in the ugen definition identifies properties that are used

in the ugen's callback method signature, and specifies their default values. When code generation occurs, the order that individual properties are enumerated in this dictionary defines their order in the ugen's callback method signature.

3.3 (Single Sample) Feedback

Allowing single sample feedback is a benefit of processing on a per-sample basis instead of processing buffers. It can be achieved in Gibberish using the Single Sample Delay (SSD) ugen. This ugen samples and stores the output of an input ugen at the end of each callback execution and then makes that sample available for the next execution. Below is a simple feedback example with a filter feeding into itself:

```
pwm = new Gibberish.PWM();
ssd = new Gibberish.SingleSampleDelay();

filter = new Gibberish.Filter24({
  input: add( pwm, ssd ),
}).connect();
ssd.input = filter;
```

In the generated callback below note that there are two functions for the SSD ugen. One records an input sample while the other outputs the previously recorded sample.

```
function () {
  var v_7 = single_sample_delay_3();
  var v_5 = pwm_2(440, 0.15, 0.5);
  var v_6 = filter24_1(v_5 + v_7, 0.1, 3, true);
  var v_4 = bus2_0(v_6, 1, 0);
  single_sample_delay_8(v_6,1);
  return v_4;
}
```

3.4 Scheduling and Sequencing

Although the JavaScript runtime includes its own methods for scheduling events, there is no guarantee about the accuracy of the time used by this scheduler. Accordingly, the only way to ensure accurate timing is to perform event scheduling from within the audio sample loop. This in turn means that any JavaScript DSP library that performs automatic graph management and callback creation (as Gibberish.js does) must also provide a mechanism for event timing that is synchronized to the audio sample loop. Gibberish.js contains a `Sequencer` object that can be used to

sequence changes to property values, calls to methods, or execution of named or anonymous functions at user-defined rates with sample-accurate timing.

The syntax for a sequencer uses a simple target/key mechanism. A sequencer's *target* specifies the ugen (or any other JavaScript object—scheduling is not limited to controlling ugens) that the sequencer will control. The *key* specifies the name of a property that should be set or a method that should be called by the sequencer. The *values* array defines the values that are passed to object methods or assigned to object properties, while the *durations* array controls how often the sequencer fires off events. By default time is specified in samples, but convenience methods are provided for converting milliseconds, seconds, and beats to samples. Thus, the following sequences a `FMSynth` looping through three pitches and alternating between half a second and a quarter second duration:

```
a = new Gibberish.FMSynth().connect();

b = new Gibberish.Sequencer({
  target: a, key:'note,'
  values: [ 440, 880, 660 ],
  durations: [ seconds(.5), seconds(.25) ],
}).start();
```

Musicians may also want to attach more than one property change or method call to each firing of a sequencer object, for example, changing an oscillator's amplitude whenever its frequency changes. To accommodate this the Sequencer object can accept a *keysAndValues* dictionary that can feature multiple key/value pairs to be sequenced.

```
a = new Gibberish.Sine().connect();

b = new Gibberish.Sequencer({
  target: a,
  durations: [ beats(2), beats(1), beats(1) ],
  keysAndValues:{
    frequency: [ 440, 880, 660 ],
    amp: [ .5, .25 ],
}).start();
```

This keysAndValues syntax has similarities to SuperCollider's *Pbind* object (Kuivila 2011), which also enables the sequencing of multiple parameters using the same timing.

When a values array contains a function, the Sequencer will simply invoke it (with no arguments) to determine the value it should use; this enables the sequencer to schedule arbitrary behavior such as randomly picking elements from an array. Functions within a sequencer's `values` array will still be called even if the Sequencer has no specified target, supporting sample-accurate sequencing of any arbitrary JavaScript function (for example, to synchronize state changes of multiple ugens).

4 Interface.js

Interface.js provides a solution for quickly creating GUIs using JavaScript that work equally well with both mouse and touch interaction paradigms. The output of these GUI widgets can control other JavaScript objects (such as Gibberish ugens) and/or be converted to MIDI or OSC (Wright 2005) messages via a small server program that is included in the Interface.js download. This means Interface.js can be used both to control programs running in the browser and also to control external programs such as digital audio workstations.

Interface.js includes a useful array of visual widgets for audiovisual control, including `Button`, `MultiButton`, `Slider`, `MultiSlider`, `RangeSlider`, `Knob`, `XY` (multitouch, with physics), `Crossfader`, `Menu`, `Label`, and `TextField`.

Interface.js also provides unified access to accelerometer readings and the orientation of the device. These motion-based sensors work on most mobile devices as well as the desktop version of Google Chrome.

4.1 Event Handling in Interface.js

Interface.js binds widgets to object functions and values using a target/key syntax similar to the Gibberish sequencer (see Sect. 3.4). A widget's *target* specifies an object and the *key* specifies a property or method to be manipulated. What happens when a widget value changes depends on what the key refers to: object properties are *set* to the new widget value, while object methods are *invoked* with the widget value as the first argument. Widgets providing multiple dimensions of control (e.g., XY, accelerometer...) are configured with arrays of targets and keys with one target and one key for each dimension of control.

Users can also register custom event handlers to create more complex interactions than direct mappings between widgets and property values. Although most browsers only deal with either touch or mouse events, Interface.js provides another type of event handler, *touchmouse events*, that respond to both forms of input. The specific events are:

- `ontouchmousedown`: when a mouse press begins or a finger first touches the screen
- `ontouchmousemove`: when a mouse or finger moves after a start event
- `ontouchmouseup`: when a mouse press ends or a finger is removed from the screen

There is also a fourth event handler, `onvaluechange`, that is called every time the value of a widget changes, regardless of whether the widget changes by touch, the mouse, or by motion. Procedurally changing a widget's value (perhaps due to the output of another widget) will also trigger this event. Although the touchmouse and

onvaluechange event handlers provide a unified interface for dealing with all events, users are free to register for dedicated mouse or touch events if they want to change how interactivity functions across different modalities.

4.2 Interface Layout and Appearance

The main container unit of Interface.js is the *Panel* widget. Panel wraps an instance of the HTML *canvas* element, a 2D drawing surface with hardware graphics acceleration support in all current desktop browsers and most current mobile browsers. Panel's constructor allows users to specify a HTML element for the canvas tag to be placed inside, allowing multiple panels to be placed at different points in a HTML page. If no container HTML element is provided, Interface.js will automatically create a canvas element that fills the entire window and attach it to the HTML page; this minimizes the HTML needed to create a GUI.

Widget sizes can be integer pixel values or float values giving the size relative to the widget's enclosing Panel. For example, consider placing a Slider inside a 500×500 pixel Panel. If the Slider's `x` value is 0.5, then its leftmost edge will be inset 250 pixels at the horizontal center of the panel. If the Slider's width is also 0.5, it will extend 250 pixels from the horizontal center to the rightmost edge of the Panel. Using relative layouts enables users to define interfaces without having to worry about varying window sizes or the screen sizes of different devices. An exception to this would be creating an interface for a large screen with a large number of widgets and then trying to view the same interface on a smartphone; in this case the widgets, although positioned and sized correctly, would be too small to use for accurate interaction.

In lieu of using CSS to specify colors, Interface.js instead allows them to be defined programmatically using JavaScript. By default, a widget uses the background, fill and stroke properties assigned its container Panel object. Thus, by default, all widgets in the same Panel use the same set of colors. Changing the color properties of a Panel object immediately changes the colors of its child widgets. If color properties for an individual widget are explicitly set their values will override the Panel's default values.

4.3 Preset Management

Interface.js allows programmers to easily store the values of widgets in an interface for instant recall. A call to the method `Interface.Preset.save('preset name')` will serialize the values of all widgets in the current interface and convert the resulting object to a string that can be stored using the HTML `localStorage` object.

`Interface.Preset.load('preset name')` loads a preset. When a preset is loaded every widget sends out its target/key message (if so defined) and executes its `onvaluechange` event handler if the new value differs from the widget's current value.

4.4 Networked Control

As an alternative to controlling Web Audio synthesis graphs or other JavaScript objects, Interface.js can also transmit output over a network via the WebSocket API in order to control remote applications. The WebSocket API sends messages over TCP but features a handshake mechanism relying on the HTTP protocol. Since most musical applications do not understand the WebSocket API, it becomes necessary to translate messages received from a WebSocket to a more appropriate musical messaging protocol, such as OSC or MIDI.

Although there have been previous efforts creating servers that translate WebSocket data into OSC or MIDI messages, we believe our solution is uniquely efficient. Instead of simply creating a server that waits for incoming WebSocket messages and then forwards them using OSC or MIDI, we have included a HTTP server that serves interface files to client devices. After launching our server application, any user directing a client browser to the server's IP address (on port 8080) will receive an HTML page listing available interfaces stored in the server's default storage directory. When a user selects an interface, it is downloaded to their browser and a WebSocket connection is automatically created linking their client device to the server; in other words, the computer that serves Interface.js pages is automatically also the recipient of the WebSocket messages generated by such interfaces. WebSocket messages sent from the interface are translated into OSC or MIDI messages based on the contents of individual interface files.

A distinct advantage of this system is that programmers do not need to think about establishing socket connections when developing their interfaces; there are no hardcoded IP addresses and no auto-discovery protocols needed. The IP address and port is entered once when first accessing the server; after that the location can be bookmarked for easy access. Individual interfaces can also be bookmarked; in this case the WebSocket connection is established as soon as the interface bookmark is selected. On mobile devices these bookmarks can be represented by icons on user's home screens; simply tapping such icons fetches the interface and immediately opens the appropriate connection.

In order to indicate that a widget's output should be sent over the network, a target of Interface.OSC or Interface.MIDI is assigned. For widgets that target Interface.OSC, the key property then represents the address the message will be sent to. For widgets targeting Interface.MIDI, the key is an array containing the message type, channel, and number (if it is a three-byte MIDI message) for the output MIDI message. Multiple devices (and thus users) can load interfaces from the server and send messages for translation and forwarding simultaneously.

5 Integrating Interface.js and Gibberish.js

There a few simple steps in order to use Interface.js and Gibberish.js together to make a musical interface. Starting with an HTML document containing the bare minimum necessary tags (`<html>, <head> & <body>`), add two `<script>` tags to import the Gibberish and Interface Javascript files. A third `<script>` tag inside the body element contains all user code to create the interface and/or audio graph. The following complete sample interface file creates a sine tone controlled by two sliders:

```
<html>
<head>
  <script src="interface.js"></script>
  <script src="gibberish_2.0.min.js"></script>
</head>

<body>
  <script>
  Gibberish.init();
  sine  = new Gibberish.Sine().connect();
  var panel = new Interface.Panel();

  sliderFrequency = new Interface.Slider({
    target:sine, key:'frequency,'
    min:150, max:1000,
    \label{Roberts::'freq,' bounds:[ 0,0,.3,1 ],
  });

  sliderAmp = new Interface.Slider({
    target:sine, key:'amp,'
    \label{Roberts::'amp,' bounds:[ .3,0,.3,1 ],
  });

  panel.add( sliderFrequency, sliderAmp );
  </script>
</body>
</html>
```

6 Related Work

There are a growing number of options for writing synthesis algorithms in JavaScript in the browser. Audiolib.js[3] was one of the first significant JavaScript audio libraries written and still provides an excellent array of synthesis options. It was our original choice for Gibber; we abandoned it only after discovering that code generation often leads to more efficient performance. Audiolib.js performs no graph management leaving programmers to implement their own audio graphs.

Other libraries take vastly different approaches in terms of the APIs they offer programmers. For example, Flocking[4] enables users to define new ugens declaratively using JavaScript Object Notation (JSON), while Timbre.js[5] is a very impressive library that enables users to compose ugens in a functional syntax inspired by jQuery, a JavaScript library for HTML manipulation. In addition to its extensive use of code generation techniques, Gibberish.js differentiates itself from these other libraries by containing several more complex pre-composed synthesis ugens. For example, Gibberish offers a polyphonic, enveloped two-op FM synthesis ugen; to our knowledge no other JavaScript libraries offer FM synthesis engines with easy control over carrier to modulation ratios and index properties. Another example of a complex pre-composed ugen is the Gibberish Monosynth, a three-oscillator bandlimited synthesizer with an envelope, 24db resonant filter, and independent tuning and waveshape controls for each oscillator. By including complex ugens we allow programmers to begin creating music with rich sound sources immediately.

Another solution for web based synthesis is JSyn (Burk 1998), a Java synthesis library originating over a decade ago. Unfortunately many browsers do not support Java by default, and many do not support it at all (including Safari on iOS).

Although there are many other HTML/JavaScript interface libraries, very few understand both touch and mouse modalities and almost none are catered towards the needs of musicians and live performers. A extension to jQuery named Kontrol[6] provides well-engineered virtual knobs and slider banks that respond to both touch and mouse events. However, there is no support for control areas tracking multiple touches across a single rectangle; in our opinion this is one of the most useful widgets for performance on touchscreen devices. The Interface.js XY widget tracks up to 11 touches simultaneously and also features a built-in physics engine, inspired by the JazzMutant Lemur.[7] In addition to providing only three widgets, the Kontrol library also requires knowledge of HTML and of the jQuery library. In general the library is targeted towards existing web developers while Interface.js attempts to shield programmers from HTML and CSS as much as possible.

[3] http://audiolibjs.org.

[4] https://github.com/colinbdclark/Flocking.

[5] https://mohayonao.github.io/timbre.js/.

[6] http://anthonyterrien.com/kontrol/.

[7] http://www.jazzmutant.com/lemur_overview.php.

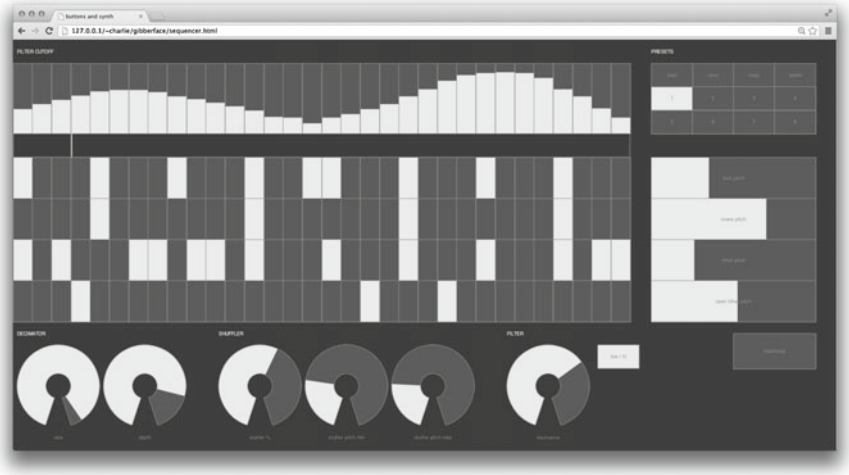


Fig. 1 A drum sequencer with four tracks of 32nd notes and controls for various effects

The massMobile project (Weitzner et al. 2012) allows audience participation in performances through web interfaces on mobile devices that output to a database that can be read by Max/MSP externals. It is interesting to note that this project started with dedicated client applications for iOS and Android before switching to use web technologies in order to become usable on a wider variety of devices. A notable precursor to the massMobile project was the composition Telemusic #1, by Young (2001). In this piece, users accessed a website with an embedded Flash animation that communicated with a Java applet running in the same page; this Java applet then forwarded information to a remote instance of Max/MSP.

7 Conclusions and Future Work

Both Gibberish.js and Interface.js are open-source and available for download on GitHub.[8] A separate code repository, *Gibberface*,[9] contains examples integrating both libraries, ranging from the simple (as in Sect. 5 above) to more complex interfaces and audio graphs such as the drum sequencer shown in Fig. 1.

Part of the inspiration for Interface.js came from our work on the mobile application *Control* (Roberts 2011). Control allows users to create interfaces for controlling remote applications with web technologies but also provides access to device features that are not exposed in browsers, such as the ability to send and receive OSC and MIDI. Advances in JavaScript APIs have exposed more and more functionality

[8] https://github.com/charlieroberts/Gibberish and https://github.com/charlieroberts/interface.js.
[9] https://github.com/charlieroberts/gibberface.

to mobile browsers' JavaScript runtime; gyro and compass sensors are two examples of sensors that were not accessible in the browser two years ago that now have established APIs. There are still a number of sensors commonly available on mobile devices that are inaccessible to JavaScript or impractical to process efficiently. As one example, detecting the surface diameter of individual touches can be used to create a crude yet surprisingly musical "aftertouch" signal that is not possible to read using browser APIs alone. As more APIs providing access to sensors become available in mobile browsers we will update Interface.js to take advantage of them.

In Gibberish there is a particular need for more analysis ugens beyond the simple envelope follower that currently exists. Gibberish has already been integrated into the live coding environment Gibber and numerous group and solo performances have been conducted using it as the synthesis engine. The first author has recently performed in concert using an instrument created with Gibberish.js and Interface.js on a tablet device; a modified version of this interface, which uses multitouch tracking to control bandlimited pulsewidth modulation oscillators, is included in the Gibberface repository.

Author Commentary: Reflections on Synthesis and Instrument Design for the Browser

Charlie Roberts, Graham Wakefield, and Matthew Wright

In 2013, fast JavaScript implementations were proliferating beyond desktop browsers to mobile devices, while new developments in browser technology were making plugin-free implementations of realtime audio synthesis feasible. Our article presented a portable, flexible, and optimized audio library, Gibberish.js, coupled with a library to bring together device capabilities and common user interface elements, Interface.js, to explore browser-based NIMEs within the capabilities of JavaScript running entirely inside a web browser.

Browser-based audio synthesis has since seen significant advances, including a standardized API (the WebAudio API, or WAAPI[10]) for creating audio graphs from a fixed library of precompiled unit generators that function across all modern web browsers. A number of excellent libraries have since been written using the nodes provided by the WAAPI (Choi and Berger 2013; Mann 2015). Although Google had begun work on this API when we began our research, it had yet to be adopted by other major browsers; using audio synthesis code written in JavaScript was the most portable solution at the time. Instead of processing unit generator nodes in blocks, Gibberish.js dynamically generates optimized JavaScript which processes the graph sample by sample, as the paper details. This enables important categories of synthesis and signal processing such as feedback FM, customized filter architectures, certain

[10] http://www.w3.org/TR/webaudio/.

physical models, and audio-rate modulation of scheduling. Precompiled unit generators can guarantee lower latencies and higher throughput, but only at the price of this flexibility. As of Fall 2015 still no other JavaScript audio synthesis libraries capitalize on per-sample processing techniques to the best of our knowledge. With upcoming updates to JavaScript engines and the WebAudio API the per-sample approach taken by Gibberish.js may become more appealing. Latencies will be reduced via WebWorkers[11] that perform audio signal processing written in JavaScript in a dedicated thread, free of pauses caused by UI interaction and networking. Efficiency may be increased by targeting code generation to lower levels via WebAssembly.[12] As options for running audio engines in the browser diversify, we predict the need for end-user languages that will use meta-programming to match user requirements to target language and engine capabilities, whether block-based WAAPI nodes, WebWorkers, WebAssembly, or otherwise. Researchers porting Csound to the browser have already made progress in this area, comparing implementations of Csound using WebAssembly to a PNaCl plugin in Google Chrome (Lazzarini et al. 2014), and we plan to continue exploring these ideas in Gibberish.js.

The current version of Gibberish.js has served as an important foundation for the creative coding environment Gibber, which has seen international use in a variety of performative and educational contexts. We are also happy to see increasing use of Gibber and Gibberish.js by other researchers. For example, Taylor et al. used Gibberish.js to create a live coding environment, *Braid*, for digital musical instrument design (Taylor and Allison 2015), while Lee et al. used it to explore the applications of recording and playing back text entry for live coding and live writing (Lee and Essl 2015). We look forward to continuing the improvement of both Gibber and Gibberish.js with help and critique from the NIME community.

The domain of plugin free, browser-based NIMEs is still nascent. Our contribution addressed a particular problem (crossbrowser, per-sample signal processing) during early days of architectural exploration; years later, critical components of a standardized audio engine for the browser remain in flux. We hope our research can help inform future synthesis libraries as the design, standardization and implementation of audio architectures in the browser continue to be refined.

Expert Commentary: The potential synthesizer in your pocket

Abram Hindle

Before the browsers clearly adopted HTML5, writing a coherent user interface usually required platform specific code. Writing an app for a mobile phone required

[11] http://webaudio.github.io/web-audio-api/.

[12] https://github.com/WebAssembly/design/blob/master/HighLevelGoals.md.

addressing each platform individually, and multimedia support was even more difficult. Yet a new breed of programmer was on the horizon, the client-side web programmer who were suddenly capable of writing multi-media rich portable graphical user interfaces (GUIs) that could run on desktops and mobile devices—but only if the technology could be adopted by enough end-users. 2012 was that time, with increased smartphone penetration combined with better mobile browser support 2012 ushered in web-GUIs that were mobile compatible.

Around the same time in 2012, as described by Roberts et al. (2013), the browser community finally started to come together around a coherent WebAudio specification,[13] a standard where a couple of extra lines Javascript could patch over any incompatibilities. In fact WebAudio was even working well within many smart-phone browsers, such as Mobile Safari and Firefox Mobile, by the start of 2013.[14] Thus the developments of 2012 enabled the mass adoption of HTML5 technologies by the majority of consumers.

The NIME community take-away is that HTML5 lets us deploy a synthesizer on every desk and in every smartphone pocket. Mass adoption combined with portability makes any HTML5-enabled device a potential synthesizer or NIME. This observation was not lost on the 2013 NIME community who in the same year published 4 or more web-GUI/WebAudio papers at the same conference. But there was 1 big glaring problem: your synthesizer is programmed in a flexible yet interpreted language, with a very limited audio API that makes lots of delays and filters quite difficult to implement efficiently or at all. Thus both developer productivity and run-time performance were at stake.

What Roberts et al. provide is a library of ugens, synths and temporal control structures leveraging the `ScriptProcessor node`, which affords the authoring of sample-level signal processing algorithms in JavaScript. This module lets one write raw samples to the soundcard's buffer, thus if you needed single sample accuracy you would have to do it here—which is exactly what they implemented. They did a lot of the heavy lifting of mixing and scheduling audio that addressed certain performance issues and certain missing `ugens` but it still didn't address the generally poor performance of Javascript and many WebAudio implementations.

Roberts et al. (2013) addressed some of Javascript's inefficiencies for signal processing by taking advantage of the language's flexibility. Modern JavaScript engines use just-in-time (JIT) compilation engines to compile well-written JavaScript into lower-level representations with improved efficiency. As JavaScript can also evaluate code contained within strings, Roberts et al. (2013) used their own code-generation engine to dynamically generate JIT-friendly Javascript from terse specifications.

In my opinion this work seemed immediately valuable to the NIME community for 3 reasons and is still relevant and valuable: (1) the software is still actively maintained,

[13] http://www.w3.org/2011/audio/.

[14] http://caniuse.com/#feat=audio-api.

it works, and you can deploy it on just about anything; (2) Roberts et al. (2013) have provided a clear framework and methodology for achieving similar successful results with WebAudio; (3) WebAudio is here to stay, Web GUIs are supplanting classic native GUIs as the functionality of HTML5 implementations improve.

The work has limitations that software tends to face, it was somewhat of a work in progress at the time, attempting to deal with the ever changing WebAudio specifications and implementations available at the time. In 2015 the WebAudio specification and capabilities continue to evolve, this violates Roberts et al.'s claim that WebAudio code is a "write once, run anywhere solution." For instance the WebAudio specification is deprecating the ScriptProcessorNode for the AudioWorkerNode. Yet the work is ever more prescient than ever as the adoption and support of HTML5 interfaces and HTML5 multimedia is more than ever before.

Gibber is still being updated to this very day and so is the WebAudio specification. I encourage anyone who takes part in low-level WebAudio development to take part in the W3C's web audio working group. This group is well meaning but they do not necessarily share the same problems or have the same needs and experiences as many of those in the NIME community; your feedback and experience is invaluable to a working group like this. Thus Gibber still fills many important niches that the W3C WebAudio specification has yet to fill.

References

Aycock, J. (2003). A brief history of just-in-time. *ACM Computing Surveys (CSUR)*, *35*(2), 97–113.

Burk, P. (1998). JSyn-a real-time synthesis API for Java. *Proceedings of the 1998 International Computer Music Conference* (pp. 252–255). CA: International Computer Music Association San Francisco.

Choi, H., & Berger, J. (2013). Waax: Web audio aplextension. In *Proceedings of the International Conference on New Interfaces for Musical Expression* (pp. 499–502). Korea: Daejeon.

Herczeg, Z., Lóki, G., Szirbucz, T., & Kiss, Á. (2009). Guidelines for JavaScript programs: Are they still necessary? In *Proceedings of the 11th Symposium on Programming Languages and Software Tools and 7th Nordic Workshop on Model Driven Software Engineering*. Tampere University of Technology.

Humphrey, D., Brook, C., MacDonald, A., Delendik, Y., Marxer, R., & Cliffe, C. (2016). Audio Data API—MozillaWiki.

Kuivila, R. (2011). Events and patterns. In *The SuperCollider book*, Chapter 6 (pp. 179–205). MIT Press.

Lazzarini, V., Costello, E., Yi, S., & Fitch, J. (2014). Csound on the web. In *Proceedings of the Linux Audio Conference*.

Lee, S. W. & Essl, G. (2015). Live writing: Asynchronous playback of live coding and writing. In *Proceedings of the International Conference on Live Coding*.

Mann, Y. (2015). Interactive music with tone.js. In *Proceedings of the 1st Annual Web Audio Conference*.

Roberts, C. (2011). Control: Software for End-User Interface Programming and Interactive Performance. *Proceedings of the International Computer Music Conference* (pp. 425–428). http://quod.lib.umich.edu/cgi/p/pod/dod-idx/control-software-for-end-user-interface-programming.pdf?c=icmc;idno=bbp2372.2011.086

Roberts, C., & Kuchera-Morin, J. (2011). Gibber: Live coding audio in the browser. In *Proceedings of the International Computer Music Conference*.

Roberts, C., Wakefield, G., & Wright, M. (2013). The web browser as synthesizer and interface. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, Daejeon, Korea.

Taylor, B. & Allison, J. (2015). Braid: A web instrument builder with embedded code blocks. In *Proceedings of the 1st Annual Web Audio Conference*.

Wakefield, G. (2012). *Real-time meta-programming for interactive computational arts*. Ph.D. thesis, University of California Santa Barbara.

Weitzner, N., Freeman, J., Garrett, S., & Chen, Y. (2012). massMobile—An audience participation framework. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, Ann Arbor, MI.

Wright, M. (2005). Open sound control: An enabling technology formusical networking. *Organised Sound*, *10*(3), 193–200.

Young, J. (2001). Using the web for live interactive music. In *Proceedings of the International Computer Music Conference* (pp. 302–305). Habana, Cuba.