

Cambridge Companions Online

<http://universitypublishingonline.org/cambridge/companions/>



The Cambridge Companion to Electronic Music

Edited by Nick Collins, Julio d'Escriván

Book DOI: <http://dx.doi.org/10.1017/CCOL9780521868617>

Online ISBN: 9781139001762

Hardback ISBN: 9780521868617

Paperback ISBN: 9780521688659

Chapter

4 - A history of programming and music pp. 55-86

Chapter DOI: <http://dx.doi.org/10.1017/CCOL9780521868617.006>

Cambridge University Press

4 A history of programming and music

GE WANG

The computer has long been considered an extremely attractive tool for creating and manipulating sound. Its precision, possibilities for new timbres and potential for fantastical automation make it a compelling platform for experimenting with and making music – but only to the extent that we can actually tell a computer what to do, and how to do it.¹

A *program* is a sequence of instructions for a computer. A *programming language* is a collection of syntactic and semantic rules for specifying these instructions, and eventually for providing the translation from human-written programs to the corresponding instructions computers carry out. In the history of computing, many interfaces have been designed to instruct computers, but none has been as fundamental (or perhaps as enduring) as programming languages. Unlike most other classes of human–computer interfaces, programming languages don’t directly perform any specific task (such as word processing or video editing), but instead allow us to build software that might perform almost any custom function. The programming language acts as a mediator between human intention and the corresponding bits and instructions that make sense to a computer. It is the most general and yet the most intimate and precise tool for instructing computers.

Programs exist on many levels, ranging from *assembler* code (extremely low level) to high-level scripting languages that often embody more human-readable structures, such as those resembling spoken languages or graphical representation of familiar objects. *Domain-specific languages* retain general programmability while providing additional abstractions tailored to the domain (e.g. sound synthesis). This chapter provides a historical perspective on the evolution of programming and music. We examine early programming tools for sound, from the rise of domain-specific languages for computer music, such as Csound and Max/MSP, to more recent developments such as SuperCollider and ChuckK. We’ll discuss how these programming tools have influenced the way composers have worked with computers.

Through all this, one thing to keep in mind is that while computers are wonderful tools, they can also be wonderfully stupid. Computers are inflexible and demand precision and exactness from the programmer. They don’t know to inform us even of our most obvious mistakes (unless someone has told the computer precisely what to look for). Furthermore, since we must formulate our intentions in terms of the syntax of the underlying language,

musical ideas are not always straightforward to translate into code. On the bright side, programming allows us to explore sounds and musical processes otherwise unavailable or impossible to create. If you haven't programmed before, creating programs to make music can be one of the best ways to learn programming: the feedback can be immediate (at least on our modern systems) and programming is learned on the side – naturally (as a tool) rather than as an end in itself. For beginning or seasoned programmers alike, we highly recommend seeking out and playing with the various music languages and environments available. Have fun with it!

Early eras: before computers

The idea of programming computational automata to make music can be traced back to as early as 1843. Ada Lovelace, while working with Charles Babbage, wrote about the applications of the theoretical 'Analytical Engine', the successor to Babbage's famous 'Difference Engine'. The original Difference Engine was chiefly a 'calculating machine' whereas the Analytical Engine (which was never built) was to contain mechanisms for decision and looping, both fundamental to true programmability. Lady Lovelace rightly viewed the Analytical Engine as a general-purpose computer, suited for 'developping [*sic*] and tabulating any function whatever . . . the engine [*is*] the material expression of any indefinite function of any degree of generality and complexity.' Her musical foresight was discussed in chapter 1.

Lady Lovelace's predictions were made more than a hundred years before the first computer-generated sound. But semi-programmable music-making machines appeared in various forms before the realisation of a practical computer. For example, the *player-piano*, popularised in the early twentieth century, is an augmented piano that 'plays itself' according to rolls of paper (called *piano rolls*) with perforations representing the patterns to be played. These interchangeable piano rolls can be seen as simple programs that explicitly specify musical scores, and as Henry Cowell advised, and such composers as Percy Grainger and Conlon Nancarrow enacted, could be used by composers to take advantage of their special capabilities.

As electronic music evolved, analogue synthesisers gained popularity (commercially, around the 1960s). They supported interconnecting and interchangeable sound processing modules. There is a certain level of programmability involved, and this block-based paradigm influenced the later design of digital synthesis systems. Those readers new to this concept might simply imagine connecting equipment together, be it a guitar to an effects

pedal for processing, or a white noise generator to a filter for (subtractive) synthesis; there are sophisticated softwares that emulate this type of hardware modular system (also see chapter 11 for more on the sound synthesis aspects).

As we step into the digital age, we divide our discussion into three overlapping eras of programming and programming systems for music. They loosely follow a chronological order, but more importantly each age embodies common themes in how programmers and composers *interact* with the computer to make sound. Furthermore, we should keep a few overall trends in mind. One crucial trend in this context is that as computers increased in computational power and storage, programming languages tended to become increasingly high-level, abstracting more details of the underlying system. This, as we shall see, greatly impacted the evolution of how we program music.

The computer age (part I): early languages and the rise of MUSIC-N

Our first era of computer-based music programming systems paralleled the age of *mainframes* (the first generations of ‘modern’ computers in use from 1950 to the late 1970s) and the beginning of personal *workstations* (mid-1970s). The mainframes were gigantic, often taking up rooms or even entire floors. Early models had no monitors or screens, programs had to be submitted via punch cards, and the results delivered as print-outs. Computing resources were severely constrained. It was difficult even to gain access to a mainframe – they were not commodity items and were centralised and available mostly at academic and research institutions (in 1957 the hourly cost to access a mainframe was \$200!). Furthermore, the computational speed of these early computers were many orders of magnitude (factors of millions or more) slower than today’s machines and were greatly limited in memory (e.g. 192 kilobytes in 1957 compared to gigabytes today). However, the mainframes were the pioneering computers and the people who used them made the most of their comparatively meagre resources. Programs were carefully designed and tuned to yield the highest efficiency.

The early algorithmic composition experiments were conducted with mainframes, and only sought to produce a symbolic score (just as a list of characters rather than any neat notation), certainly without synthesised realisation. In one project, Martin Klein and Douglas Bolitho used the Datatron computer to create popular songs after formalising their own rules

from an analysis of recent chart hits – the song *Push Button Bertha* (1956) (with tacked on lyrics by Jack Owens) is jokingly attributed to the Datatron computer itself.² Sound generation on these machines became a practical reality with the advent of the first *digital-to-analogue converters* (or DACs), which converted digital audio samples (essentially sequences of numbers) that were generated via computation, to time-varying analogue voltages, which can be amplified to drive loudspeakers or be recorded to persistent media (e.g. magnetic tape).

MUSIC I (and II, III, . . .)

The earliest programming environment for sound synthesis, called *MUSIC*, appeared in 1957, developed by Max Mathews at AT&T Bell Laboratories. It was not quite a full programming language as we might think of one today. Not only were *MUSIC* (or *MUSIC I*, as it was later referred to) and its early descendants the first music programming languages widely adopted by researchers and composers, they also introduced several key concepts and ideas which still directly influence languages and systems today.

MUSIC I and its direct descendants (typically referred to as *MUSIC-N languages*), at their core, provided a model for specifying sound synthesis modules, their connections and time-varying control (Mathews 1969). This model eventually gave rise, in *MUSIC III*, to the concept of *unit generators*, or *UGens* for short. UGens are atomic, often predefined, building blocks for generating or processing audio signals. In addition to audio input and/or output, a UGen may support a number of *control inputs* that control parameters associated with the UGen.

An example of a UGen is an *oscillator*, which outputs a periodic waveform (e.g. a sinusoid) at a particular fundamental frequency. Such an oscillator might include control inputs that dictate the frequency and phase of the signal being generated. Other examples of UGens include *filters*, *panners*, and *envelope generators*. The latter, when triggered, produce amplitude contours over time. If we multiply the output of a *sine wave oscillator* with that of an envelope generator, we can produce a third audio signal: a sine wave with time-varying amplitude. In connecting these unit generators in an ordered manner, we create a so-called *instrument* or *patch* (the term comes from analogue synthesisers that may be configured by connecting components using patch cables), which determines the audible qualities (e.g. *timbre*) of a sound. In *MUSIC-N* parlance, a collection of *instruments* is an *orchestra*. In order to use the *orchestra* to create music, a programmer could craft a different type of input that contained time-stamped note sequences or control signal changes, called a *score*. The relationship: the *orchestra* determines *how sounds are generated*, whereas the *score* dictates (to the orchestra) what to play and when. These two ideas – the unit generator, and the notion of

an orchestra versus a score as programs – have been highly influential to the design of music programming systems and, in turn, to how computer music is programmed today.

In those early days, the programming languages themselves were implemented as low-level assembly instructions (essentially human-readable machine code), which effectively coupled a language to the particular hardware platform it was implemented on. As new generations of machines (invariably each with a different set of assembly instructions) were introduced, new languages or at least new implementations had to be created for each architecture. After creating MUSIC I, Max Mathews soon created *MUSIC II* (for the IBM 740), *MUSIC III* in 1959 (for the IBM 7094), and *MUSIC IV* (also for the 7094, but recoded in a new assembly language). Bell Labs shared its source code with computer music researchers at Princeton University – which at the time also housed a 7094 – and many of the additions to MUSIC IV were later released by Godfrey Winham and Hubert Howe as *MUSIC IV-B*.

Around the same time, John Chowning, then a graduate student at Stanford University, travelled to Bell Labs to meet Max Mathews, who gave Chowning a copy of MUSIC IV. *Copy* in this instance meant a box containing about three thousand punch cards, along with a note saying ‘Good luck!’. John Chowning and colleagues were able to get MUSIC IV running on a computer that shared the same storage with a second computer that performed the digital-to-analogue conversion. In doing so, they created one of the world’s earliest *integrated computer music systems*. Several years later, Chowning, Andy Moorer, and their colleagues completed a rewrite of MUSIC IV, called *MUSIC 10* (named after the PDP-10 computer on which it ran), as well as a program called *SCORE* (which generated note lists for MUSIC 10).

It is worthwhile to pause here and reflect how composers had to work with computers during this period. The composer/programmer would design their software (usually away from the computer), create punch cards specifying the instructions, and submit them as jobs during scheduled mainframe access time (also referred to as *batch-processing*), sometimes travelling a long distance to reach the computing facility. The process was extremely time-consuming. A minute of audio might take several hours or more to compute, and turn-around times of several weeks were not uncommon. Furthermore, there was no way to know ahead of time whether the result would sound anything like what was intended! After a job was complete, the generated audio would be stored on computer tape and then be digital-to-analogue converted, usually by another computer. Only then could the composer actually hear the result. It would typically take many such iterations to complete a piece of music.³

In 1968, *MUSIC V* broke the mould by being the first computer music programming system to be implemented in *FORTRAN*, a high-level general-purpose programming language (often considered the first). This meant *MUSIC V* could be *ported* to any computer system that ran *FORTRAN*, which greatly helped both its widespread use in the computer music community and its further development. While *MUSIC V* was the last and most mature of the Max Mathews/Bell Labs synthesis languages of the era, it endures as possibly the single most influential computer music language. Direct descendants include *MUSIC 360* (for the IBM 360) and *MUSIC 11* (for the PDP-11) by Barry Vercoe and colleagues at MIT, and later *cmusic* by F. Richard Moore. These and other systems added much syntactic and logical flexibility, but at heart remained true to the principles of *MUSIC-N* languages: connection of unit generators, and the separate treatment of sound synthesis (orchestra) and musical organisation (score). Less obviously, *MUSIC V* also provided the model for many later computer music programming languages and environments.

The CARL System (or ‘UNIX for music’)

The 1970s and 80s witnessed sweeping revolutions to the world of computing. The C programming language, one of the most popular in use, was developed in 1972. The 70s were also a decade of maturation for the modern *operating system*, which includes *time-sharing* of central resources (e.g. CPU time and memory) by multiple users, the factoring of runtime functionalities between a privileged *kernel mode* versus a more protected *user mode*, as well as clear process boundaries that protect applications from each other. From the ashes of the titanic *Multics* operating system project arose the simpler and more practical *UNIX*, with support for multi-tasking of programs, multi-user, inter-process communication, and a sizeable collection of small programs that can be invoked and interconnected from a command line prompt. Eventually implemented in the C language, *UNIX* can be ported with relative ease to any new hardware platform for which there is a C compiler.

Building on the ideas championed by *UNIX*, F. Richard Moore, Gareth Loy and others at the Computer Audio Research Laboratory (CARL) at University of California at San Diego developed and distributed an open-source, portable system for signal processing and music synthesis, called the *CARL System* (Loy 1989; 2002). Unlike previous computer music systems, *CARL* was not a single piece of software, but a collection of small, command line programs that could send data to each other. The ‘distributed’ approach was modelled after *UNIX* and its collection of interconnectible programs, primarily for text-processing. As in *UNIX*, a *CARL process* (a running instance of a program) can send its output to another process via the *pipe* (`|`), except

instead of text, CARL processes send and receive audio data (as sequences of floating point samples, called *floatsam*). For example, the command:

```
> wave -waveform sine -frequency 440Hz | spect
```

invokes the wave program, and generates a sine wave at 440Hz, which is then ‘piped’ (|) to the spect program, a spectrum analyser. In addition to audio data, CARL programs could send *side-channel* information, which allowed potentially global parameters (such as sample rate) to propagate through the system. Complex tasks could be scripted as sequences of commands. The CARL System was implemented in the C programming language, which ensured a large degree of portability between generations of hardware. Additionally, the CARL framework was straightforward to extend – one could implement a C program that adhered to the CARL *application programming interface* (or API) in terms of data input/output. The resulting program could then be added to the collection and be available for immediate use.

In a sense, CARL approached the idea of digital music synthesis from a *divide-and-conquer* perspective. Instead of a monolithic program, it provided a flat hierarchy of small software tools. The system attracted a wide range of composers and computer music researchers who used CARL to write music and contributed to its development. Gareth Loy implemented packages for FFT (Fast Fourier Transform) analysis, reverberation, spatialisation, and a music programming language named *Player*. Richard Moore contributed the *cmusic* programming language. Mark Dolson contributed programs for phase vocoding, pitch detection, sample-rate conversion, and more. Julius O. Smith developed a package for filter design and a general filter program. Over time, the CARL Software Distribution consisted of over one hundred programs. While the system was modular and flexible for many audio tasks, the architecture was not intended for realtime use. Perhaps mainly for this reason, the CARL System is no longer widely used in its entirety. However, thanks to the portability of C and to the fact CARL was open source, much of the implementation has made its way into countless other digital audio environments and classrooms.

Cmix, CLM and Csound

Around the same time, the popularity and portability of C gave rise to another unique programming system: Paul Lansky’s *Cmix* (Pope 1993; Lansky 2006). *Cmix* wasn’t directly descended from MUSIC-N languages; in fact it’s not a programming language, but a C *library* of useful signal processing and sound manipulation routines, unified by a well-defined API. Lansky authored the initial implementation in the mid-1980s to flexibly mix sound files (hence the name *Cmix*) at arbitrary points. It was partly intended to

alleviate the inflexibility and large turnaround time for synthesis via batch processing. Over time, many more signal processing directives and macros were added. With Cmix, programmers could incorporate sound processing functionalities into their own C programs for sound synthesis. Additionally, a score could be specified in the Cmix scoring language, called *MINC*.⁴ MINC's syntax resembled that of C and proved to be one of the most powerful scoring tools of the era, due to its support for control structures (such as loops). Cmix is still distributed and widely used today, primarily in the form of *RTCmix* (the RT stands for realtime), an extension developed by Brad Garton and David Topper.

Common Lisp Music (or *CLM*) is a sound synthesis language written by Bill Schottstaedt at Stanford University in the late 1980s. CLM descends from the MUSIC-N family and employs a LisP-based syntax for defining the instruments and score and provides a collection of functions that create and manipulate sound. Due to the naturally recursive nature of LisP (which stands for List Processing), many hierarchical musical structures turned out to be straightforward to represent using code. A more recent (and very powerful) LisP-based programming language is *Nyquist*, authored by Roger Dannenberg (Dannenberg 1997). Both CLM and Nyquist are freely available.

Today, the most widely used direct descendant of MUSIC-N is *Csound*, originally authored by Barry Vercoe and colleagues at MIT Media Labs in the late 1980s (Boulanger 2000). It supports unit generators as *opcodes*, objects that generate or process audio. It embraces the instrument versus score paradigm: the instruments are defined in orchestra (*.orc*) files, with the score in *.sco* files. Furthermore, Csound supports the notion of separate audio and control rates. The *audio rate* (synonymous with *sample rate*) refers to the rate at which audio samples are processed through the system. On the other hand, *control rate* dictates how frequently *control signals* are calculated and propagated through the system. In other words, audio rate (abbreviated as *ar* in Csound) is associated with *sound*, whereas control rate (abbreviated as *kr*) deals with *signals that control sound* (i.e. changing the centre frequency of a resonant filter or the frequency of an oscillator). The audio rate is typically higher (for instance 44100 Hz for CD-quality audio) than the control rate, which usually is adjusted to be lower by at least an order of magnitude. The chief reason for this separation is computational efficiency. Audio must be computed sample-for-sample at the desired sample rate. However, for many synthesis tasks, it makes no perceptual difference if control is asserted at a lower rate, say of the order of 2000Hz. This notion of audio rate versus control rate is widely adopted across nearly all synthesis systems.

This first era of computer music programming pioneered how composers could interact with the digital computer to specify and generate music. Its

mode of working was associated with the difficulties of early mainframes: offline programming, submitting batch jobs, waiting for audio to generate, and transferring to persistent media for playback or preservation. It paralleled developments in computers as well as general-purpose programming languages. We examined the earliest music languages in the MUSIC-N family as well as some direct descendants. It is worth noting that several of the languages discussed in this section have since been augmented with real-time capabilities. In addition to RTMix, Csound now also supports realtime audio.

The computer age (part II): realtime systems

This second era of computer programming for music partially overlaps with the first. The chief difference is that the mode of interaction moved from offline programming and batch processing to realtime sound synthesis systems, often controlled by external musical controllers. By the early 1980s, computers had become fast enough and small enough to allow workstation desktops to outperform the older, gargantuan mainframes. As personal computers began to proliferate, so did new programming tools and applications for music generation (Lyon 2002).

Graphical music programming: Max/MSP and Pure Data

We now arrive at one of the most popular computer music programming environments to this day: *Max* and later *Max/MSP* (Puckette 1991). Miller S. Puckette implemented the first version of Max (when it was called *Patcher*) at IRCAM in Paris in the mid-1980s as a programming environment for making interactive computer music. At this stage, the program did not generate or process audio samples; its primary purpose was to provide a graphical representation for routing and manipulating signals for controlling external sound synthesis workstations in realtime. Eventually, Max evolved at IRCAM to take advantage of DSP hardware on NeXT computers (as *Max/FTS*, FTS stands for ‘faster than sound’), and was later released in 1990 as a commercial product by Opcode Systems as *Max/Opcode*. In 1996, Puckette released a completely redesigned and open source environment called *Pure Data*, or *PD* for short (Puckette 1996). At the time, Pure Data processed audio data whereas Max was primarily designed for control (MIDI). PD’s audio signal processing capabilities then made their way into Max as a major add-on called *MSP* (MSP either stands for Max Signal Processing or for Miller S. Puckette), authored by Dave Zicarelli. *Cycling* ’74, Zicarelli’s company, distributes the current commercial version of *Max/MSP*.

The modern-day Max/MSP supports a graphical patching environment and a collection containing thousands of *objects*, ranging from signal generators, to filters, to operators, and user interface elements. Using the Max import API, third party developers can implement *external* objects as extensions to the environment. Despite its graphical approach, Max descends from MUSIC-V (in fact Max is named after the father of MUSIC-N, Max Mathews) and embodies a similarly modular approach to sound synthesis. Well-known uses of Max/MSP include Philippe Manoury's realisation of *Jupiter* (which in 1987, was among the first works to employ a score following algorithm to synchronise a human performer to live electronics, as well as being the test case for Puckette's work), Autechre's exploration of generative algorithms to create the recordings for their *Confield* (2001) album, and Radiohead's employment of Max/MSP in live performance. Cycling '74 themselves have a record company whose releases promote artists using the Max/MSP software. The Max community is extensive and the environment has been used in countless aspects of composition, performance and sound art.

Max offers two modes of operation. In *edit* mode, a user can create objects, represented by on-screen boxes containing the object type as well as any initial arguments. An important distinction is made between objects that generate or process audio and *control rate objects* (the presence of a '~' at the end of the object name implies audio rate). The user can then interconnect objects by creating connections from the *outlets* of certain objects to the *inlets* of others. Depending on its type, an object may support a number of inlets, each of which is well defined in its interpretation of the incoming signal. Max also provides dozens of additional widgets, including message boxes, sliders, graphs, knobs, buttons, sequencers and meters. Events can be manually generated by a *bang* widget. All of these widgets can be connected to and from other objects. When Max is in *run* mode, the patch topology is fixed and cannot be modified, but the various on-screen widgets can be manipulated interactively. This highlights a wonderful duality: a Max *patch* is at once a program and (potentially) a user interface.

Max/MSP has been an extremely popular programming environment for realtime synthesis, particularly for building interactive performance systems. Controllers – both commodity (MIDI devices) and custom – as well as sensors (such as motion tracking) can be mapped to sound synthesis parameters using Max/MSP. The visual aspect of the environment lends itself well to monitoring and fine-tuning patches. Max/MSP can be used to render sequences or scores, though due to the lack of detailed timing constructs (the graphical paradigm is better at representing *what* than *when*), this can be less straightforward. PD, maintained as an open source project, is itself still under development and freely available for all modern

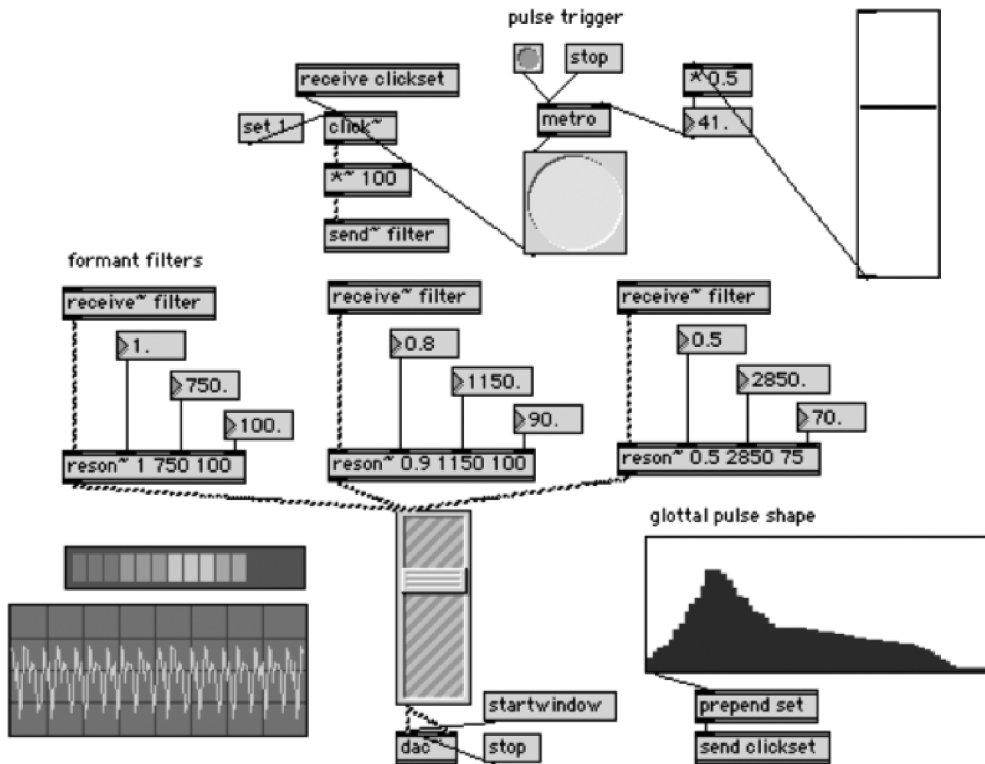


Figure 4.1 A simple Max/MSP patch which synthesises the vowel 'ahh'

operating systems. Additionally, the graphical patching paradigm has found its way into much modular synthesis software, including Native Instruments' *Reaktor*.

Programming libraries for sound synthesis

So far, we have discussed mostly stand-alone programming environments, each of which provides a specialised language syntax and semantics. In contrast to such languages or environments, a *library* provides a set of specialised functionalities for an existing, possibly more general-purpose language. For example, the *Synthesis Toolkit* (STK) is a collection of building blocks for realtime sound synthesis and physical modelling, for the C++ programming language (Cook and Scavone 1999). STK was authored by Perry Cook and Gary Scavone and released in the early 1990s. *JSyn*, released around the same time, is a collection of realtime sound synthesis objects for the *Java* programming language (Burk 1998). In each case, the library provides an API, with which a programmer can write synthesis programs in the host language (e.g. C++ and Java). For example, STK provides an object definition

called Mandolin, which is a physical model for a plucked string instrument. It defines the data types that internally comprise such an object, as well as publicly accessible functionalities that can be invoked to control the Mandolin's parameters in realtime (e.g. frequency, pluck position, instrument body size, etc.). Using this definition, the programmer can create instances of Mandolin, control their characteristics via code, and generate audio from the Mandolin instances in realtime. While the host languages are not specifically designed for sound, these libraries allow the programmer to take advantage of language features and existing libraries (of which there is a huge variety for C++ and Java). This also allows integration with C++ and Java applications that desire realtime sound synthesis.

SuperCollider: the synthesis language redefined

SuperCollider is a text-based audio synthesis language and environment first developed around the early 1990s (McCartney 2002). It is a powerful *interpreted* programming language, so that new code can be run immediately, even being written while existing processes occur, and the implementation of the synthesis engine is highly optimised. It combines many of the key ideas in computer music language design while making some fundamental changes and additions. SuperCollider, like languages before it, supports the notion of unit generators for signal processing (audio and control). However, there is no longer a distinction between the orchestra (sound synthesis) and the score (musical events): both can be implemented in the same framework. This tighter integration leads to more expressive code, and the ability to couple and experiment with synthesis and musical ideas together with faster turnaround time. Furthermore, the language, which in parts resembles the Smalltalk and C programming languages, is *object-oriented* and provides a wide array of expressive programming constructs for sound synthesis and user interface programming. This makes SuperCollider suitable not only for implementing synthesis programs, but also for building large interactive *systems* for sound synthesis, algorithmic composition, and for audio research. SuperCollider has been used in Jem Finer's millennial thousand-year installation *LongPlayer*, Chris Jeffs' *Cylob Music System*, Nick Collins' *Infinite Length Pieces*, electroacoustic works by Scott Wilson, Joshua Parmenter and Ron Kuivila, and many other compositions, artworks and systems.⁵

At the time of this writing, there have been three major version changes in SuperCollider. The third and latest (often abbreviated to SC3) makes an explicit distinction between the language (front-end) and synthesis engine (back-end). These loosely coupled components communicate via *OpenSoundControl* (OSC), a standard for sending control messages for sound over a network. One immediate impact of this new architecture is that

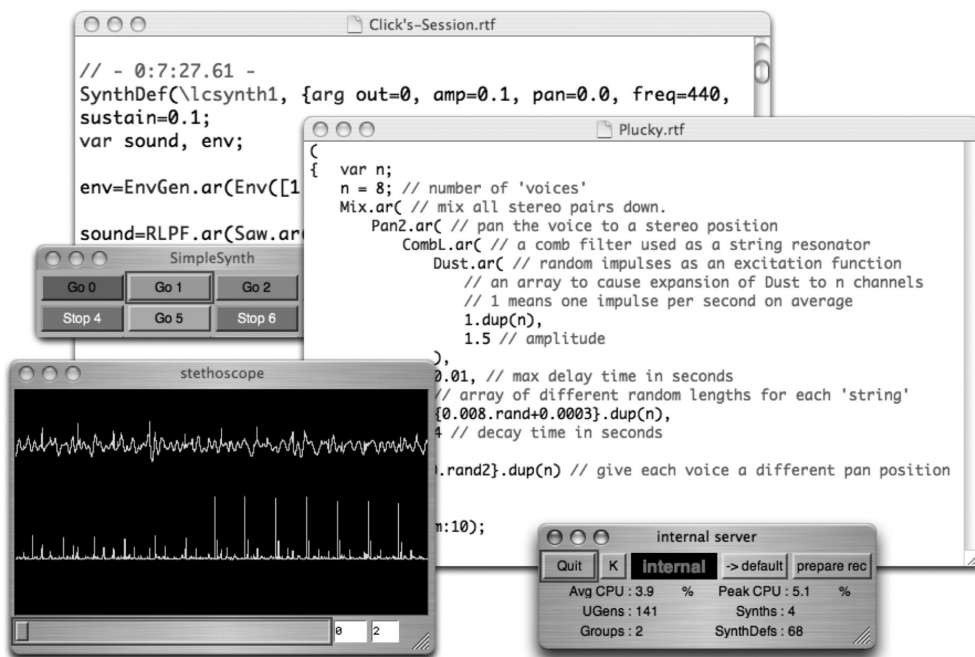


Figure 4.2 The SuperCollider programming environment in action

programmers can essentially use any front-end language, as long as it conforms to the protocol required by the synthesis server (called *scsynth* in SuperCollider). A second is that SuperCollider is inherently ready for network music.

Graphical versus text-based approaches

It is worthwhile to pause here and reflect on the differences between the graphical programming environments of Max/MSP and PD versus the text-based languages and libraries such as SuperCollider, Csound, and STK. The visual representation presents the dataflow directly, in a what-you-see-is-what-you-get sort of way. Text-based systems lack this representation and understanding of the syntax and semantics is required to make sense of the programs. However, many tasks, such as specifying complex logical behaviour, are more easily expressed in text-based code.

Ultimately it's important to keep in mind that most synthesis and musical tasks can be implemented in *any* of these languages. This is the idea of *universality*: two constructs (or languages) can be considered equivalent if we can emulate the behaviour of one using the other, and vice versa. However, certain types of tasks may be more easily specified in a particular language than in others. This brings us back to the idea of the programming

language as a tool. In general, a tool is useful if it does at least one thing better than any other tool (for example, a hammer or a screwdriver). Computer music programming languages are by necessity more general, but differing paradigms lend themselves to different tasks (and no single environment ‘does it best’ in *every* aspect: it’s important to choose the right tools for the tasks at hand). In the end, it’s also a matter of personal preference – some like the directness of graphical languages whereas others prefer the feel and expressiveness of text-based code. It’s often a combination of choosing the right tool for the task and finding what the programmer is comfortable working in.

The computer age (part III): new language explorations

With the growth of low-cost, high performance computers, the realtime and interactive music programming paradigms are more alive than ever and expanding with the continued invention and refinement of expressive interfaces. Alongside the continuing trend of explosive growth in computing power is the desire to find new ways to leverage programming for realtime interaction. If the second era of programming and music evolved from computers becoming commodities, then this third era is the result of programming itself becoming *pervasive*. With the ubiquity of hardware and the explosion of new high-level general-purpose programming tools (and people willing to use them), more composers and musicians are crafting not only software to create music, but also new software to *program* music.

As part of this new age of exploration, a recent movement has been taking shape. This is the rise of dynamic languages and consequently of using the act of programming itself as a musical instrument. This, in a way, can be seen as a subsidiary of realtime interaction, but with respect to programming music, this idea is fundamentally powerful. For the first time in history, we have commodity computing machines that can generate sound and music in realtime (and in abundance) from our program specifications. One of the areas investigated in our third age of programming and music is the possibility of changing the *program* itself in realtime – as it’s running. Given the infinite expressiveness of programming languages, might we not leverage code to create music *on-the-fly*?

The idea of run-time modification of programs to make music (interchangeably called *live coding*, *on-the-fly programming*, *interactive programming*) is not an entirely new one.⁶ As early as the beginning of the 1980s, researchers such as Ron Kuivila and groups like the Hub experimented with runtime modifiable music systems. The *Hierarchical Music Scoring Language* (HMSL) is a Forth-based language, authored by Larry Polansky, Phil

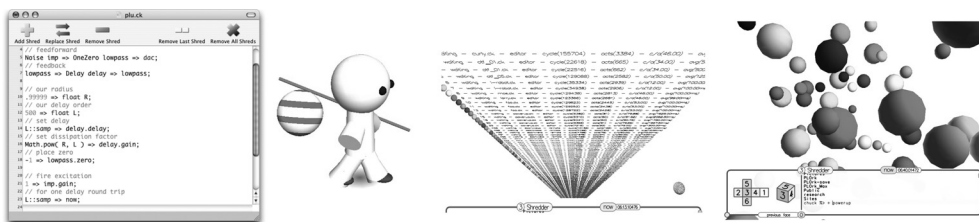


Figure 4.3 The Chuck programming language and environment

Burk, David Rosenboom and others in the 1980s, whose stack-based syntax encourages runtime programming. These are the forerunners of live coding (Collins *et al.* 2003). The fast computers of today enable an additional key component: realtime sound synthesis.

Chuck: a strongly timed and on-the-fly programming language

Chuck is one of the newest members to the audio synthesis programming language family, originated by the author of this chapter and Perry Cook (Wang and Cook 2003). *Chuck* is derived indirectly from the MUSIC-N paradigm with some key differences. First, the *chuck* operator ($\Rightarrow$) is used to perform actions in a left-to-right way, including UGen connection. The language also establishes a strong correspondence between *time* and *sound* in that the programmer controls temporal flow (via special language syntax) to allow sound to be generated (this is referred to as *strongly timed*). Additionally, different *processes* can synchronise to each other according to the same notion of time, or to data. There is no separation of orchestra versus score, and control rate is programmatically determined as a consequence of manipulating time. In this sense, *Chuck* is the first realtime synthesis language to move away from the classic notion of control rate. The programming model lends itself to representing low-level synthesis processes as well as high-level musical representations, and the language makes it straightforward for learning sound synthesis and for rapidly prototyping compositions. In addition to realising score-driven synthesis, *Chuck* is also used as a primary teaching and compositional tool in the *Princeton Laptop Orchestra* (PLOrk), serving as the workbench for creating live performances (for solos, duets, quartets, and for the full ensemble of fifteen humans, fifteen laptops, and ninety audio channels).

Custom music programming software

An incredibly vibrant and wonderful aspect of the era is the proliferation of custom, ‘home-brew’ sound programming software. The explosion of new high-level, general-purpose programming platforms has enabled and

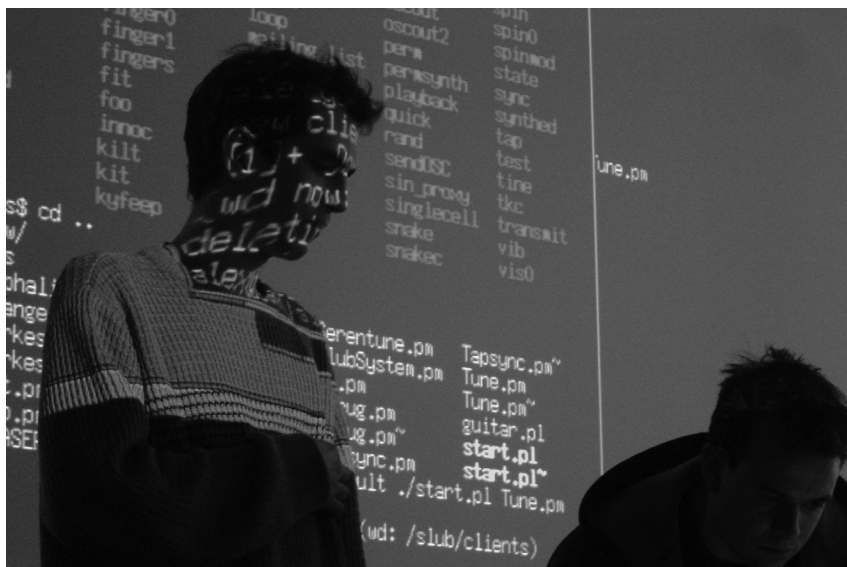


Figure 4.4 slub in action (photo by Renate Wieser)

encouraged programmers and composers to build systems very much tailored to their liking. Alex McLean describes live coding using the high-level scripting language *Perl* (McLean 2004), a technique he has utilised in club and gallery performances both solo and with the duo *slub*. Similar frameworks have been developed in *Python*, various dialects of *Lisp*, *Forth*, *Ruby*, and others. Some systems make sound while others visualise it. Many systems send network message (in *OpenSoundControl*) to synthesis engines such as *SuperCollider Server*, *PD*, *Max*, and *Chuck*. In this way, musicians and composers can leverage the expressiveness of the front-end language to make music while gaining the functionalities of synthesis languages. Many descriptions of systems and ideas can be found through *TOPLAP* (which usually stands for *Transnational Organisation for the Proliferation of Live Audio Programming*), a collective of programmers, composers, and performers exploring live programming to create music.

This third era is promising because it enables and encourages new compositional and performance possibilities not only to professional musicians, researchers and academics, but also to anyone willing to learn and explore programming and music. Also, the new dynamic environments for programming are changing how we approach more ‘traditional’ computer music composition – by providing more rapid experimentation and more immediate feedback. This era is young but growing rapidly and the possibilities are truly fascinating. Where will it take programming and music in the future?

Future directions

Lady Ada Lovelace foresaw the computing machine as a programming tool for creating precise, arbitrarily complex and ‘scientific’ music. What might we imagine about the ways music will be made decades and beyond from now?

Several themes and trends pervade the development of programming languages and systems for music. The movement towards increasingly more realtime, dynamic and networked programming of sound and music continues; it has been taking place in parallel with the proliferation and geometric growth of commodity computing resources until recent times. New trends are emerging. At the time of this writing in 2006, the focus in the design of commodity machines is shifting to distributed, multi-core processing units. We may soon have machines with hundreds (or many more) cores as part of a single computer. How might these potentially massive parallel architectures impact on the way we think about and program software (in everything from commercial data-processing to sound synthesis to musical performance)? What new programming paradigms will have to be invented to take advantage of these and other new computing technology such as quantum computers, and theoretical (for now) computing machines? Finally, an equally essential question: how can we better make use of the machines we have?

ARTISTS' STATEMENTS I

LAURIE SPIEGEL

From the earliest, I craved the arts, any of the arts, all of the arts. The feelings, thoughts and imaginings presented to me by other minds did not represent, reflect or resonate with my solitary subjective experience, nor did they provide the means I so urgently felt I needed of making life's moment-to-moment intensity more comfortable.

As a result I have always been involved in far too many things at once: writing, playing and composing music, making visual images and pursuing the externalisation of the evolving images and sounds that appear only on my imagination's retina, developing new tools for these tasks including sawing, soldering, coding for computers, and generally getting excited about ideas in many fields.

At every stage several threads intertwine, components not only of created work (sound, image and text), but of daily life (home, dogs, friends, beloved acoustic instruments, several sciences, electronic and mechanical tinkering . . .) and the pursuit of understanding. I have found myself almost always in overload, especially as a little goes a long way, any interesting idea tending to intersect with others to spin off into many more. As a teenager, a shy awkward 'girl nerd', I was seen playing guitar and banjo, taking woodworking shop, calligraphy and drafting classes, running little scientific experiments, drawing and sculpting, writing poems and fiction, doing science fair projects, inventing a phonetic alphabet, even winning a prize for advertising layout, and reading, reading, reading.

More recently, technology has furnished a means of interconnection for all the parts of this disparate array. Paradoxically, by specialising in music (always the least resistible of all my pursuits) I found that all the other domains that I thought I had traded off against it were drawn back in. Music does not exist in isolation any more than any individual, society or subject of study. Music touches upon everything else, from mathematics to philosophy to carpentry. Most important though, it touches our innermost selves.

I did not expect to become a composer. I just kept finding, when I went to my scores or record collection, that the music I was looking for and wanted to play was not there. So being a tinkerer prone to 'do it yourself', I would make myself some of whatever music I couldn't find. My computer music software, best known of which is my little program *Music Mouse*, is similar,

made for my own use, and only ex post facto discovered to be wanted by others. Though I have made music 'on demand' and to others' specifications, such as for dance, theatre or film, I am primarily inner-directed and all of what I consider my best work is always made for my own needs. This is another wonderful paradox that perhaps only the arts manifest well, that by ignoring others and pleasing the self we are more able to please others more effectively.

Because much of what I have felt and seen in my mind and imagination is difficult to mash into conventional media, I have spent astronomical amounts of time on the design and creation of tools, mostly electronic and computer-based, to create what previously could not be made or to do so by methods not tried before. I love this work almost as much as the music itself. Each tool (instrument, medium, technique) is like a language, able to express some things inexpressible by others of its kind, and yet full of commonality with them. Each may severely limit the nature of one's creative output but in the cause of revealing with a clearer focus a unique delimited aesthetic domain. In this way an instrument is like a person, and each individual artist has similar uniqueness and communality with others.

This is why I have also worked hard to make it easier for more people to be able to express themselves in music and art by use of new technology. There should never be a minority category of 'creative artist' from which most people are excluded. All who wish to speak any language – sound, sight, speech – should have the opportunity to do so. And I have long hoped that the immediacy of electronics and the logic of computers will make this possible for far more people than ever before. This is important because the benefit of creative self-expression falls mostly to the maker, leaving to any audience only a secondary level of involvement, a vicarious self-expression through the artefacts that the creative process leaves behind it.

These attitudes have of course reduced my apparent output not only due to the time taken from music for toolmaking but because I too rarely write down or record the music I come up with, though ironically it means the world to me for someone to listen and hear what I've made.

YASUNAO TONE

The origin of the 'Wounded Man'yo-shu' pieces

It was five years ago, for the opening of the Yokohama Triennial, that I created a sound installation from Walter Benjamin's text titled 'Parasite/Noise'. An interviewer from the *Bijutsu-Techō*, a leading Japanese art magazine, asked me why I liked to obsess with language and text. I answered that it is not just

language and text, but Chinese or Japanese text. Before the gramophone, as Kittler explained, there were only 'texts and scores, Europe had no other means for storing time'. Only the alphabet and its subsystem, staff notation, could preserve works, at least until mechanical reproductions were invented. In other words, Western music always coexisted with Western languages.

I have been working on creating music outside the European alphabetic/tonal system all along, since I started consciously using Chinese characters for a performance piece called *Voice and Phenomenon* in 1976. As you might suppose from the title, it was based on Derrida's texts. The piece resonates with Derrida's critique of alphabetical phoneticism as ethnocentrism. According to Derrida, the history of Western philosophy from Plato to Husserl is the history of complicity by logos and phonè based on the metaphysics of presence. That is why Western metaphysics has given a privilege to voice over writing (*écriture* in the Saussurian sense). *Voice and Phenomenon* was a seminal predecessor to other works such as *Molecular Music*, *Musica Iconologos* and *Musica Simulacra*. All these works are made of sound through many layers of conversion from Chinese characters. Other pieces are derivative of those pieces, by way of prepared CDR.

JOHN OSWALD

The term Electronic Music is used in an historical and mutually exclusive sense by both the electroacousmaticians, and the purveyors of the countless subgenres of Electronica (House, Techno, Jungle, etc.). However, electrification is now the dominant state in which virtually all forms of music exist. In the twentieth century electricity and music, gradually, through radio, recording, amplification, electrogeneration and processing, became an ubiquitous relationship to a degree which likely caused a decline in both the creation and perception of purely acoustic sounds intended as music, such as singing.

This attrition applies especially to music as a form of communication or social congress. Only the classical concert milieu remains acoustic, to a near militant degree, although the practice of classical music, in its dissemination, has been transformed by recording technology. When electricity arrives in a culture, musicians use it. Electricity has generally made music louder, more pervasive, and, globally, more homogenised. At the same time recordings have increased the available variety of music and personal playback devices have enabled private sound worlds devoid of immediate shared experience. Recording and broadcasting have spawned an increasingly homogeneous global music, even as the possibilities of electronic transformation of sounds have created an ever more multifaceted range of possibilities.

Perhaps as a reprieve to the long solitary hours I spend in the studio composing in an electronic environment of recordings, and re-recordings of recordings (of which a subset is plunderphonics), I have tended to eschew electricity when I play music as a sociable form. Like sex, music thrives on spontaneity, and there's nothing more sonically convivial than a group of musicians picking up things to pluck, blow into or bellow. I do look forward to someone showing up at one of these gatherings with a soniferous computelligent electrosonic device as sensitive, flexible, and quick-to-boot as these musicians who needn't plug in.

MATHIAS GMACHL (FARMER'S MANUAL)

<begin statement>

physical reality, as modelled by many temporal, local theories, appears to be fundamentally linked to a zoo of oscillation processes. our senses synthesise these cycles into experiences, be it sonic events, phenomena of light, awareness of space, wireless communication, plain matter and much more. as indicated by the order given, the sonic poses a very accessible and rewarding way to explore wave phenomena, but this is not where it ends.

engineering activities have uncovered the electromagnetic within certain limits, or to particular purposes, although to a great extent in disregard or disinterest of secondary effects on physiology and consciousness, in other words, the medium is not neutral and on its own commences sending messages of unintended content to unintended receivers.

indifferent music becomes sound as it identifies itself as operations on the organisation of weak processes, disregarding boundaries of occlusion. so that towards the end the authors become the audience and the audience stretches to the horizon, populated by machines infinitely rendering a stream of minute semantic lunacy.

what is left is the establishment of a musical use of a broader band of waves, both regarding frequency range and media.

a system regulating frequency, duration, event-clustering and arbitrarily more parameters is at least a musical system. in a stochastic system this means choice of probabilities, in a process-coupled system this means filling the system with life by choice of governing events.

i'm sorry i couldn't elaborate on a geometric perspective due to lack of time and misplacing the resonating compass.

<end statement>

ERDEM HELVACIOGLU

It has been nearly fifty years since we heard the first electronic music work by a Turkish composer. That was when Bulent Arel's piece *Music for String Quartet and Tape* was premiered. Looking back now, I am still amazed by the compositional virtuosity, talent and vision of the great pioneer, and the sheer beauty of this wonderful work. A monumental piece composed with very simple tools in an environment with political and economic problems, and where there was nearly no knowledge about international contemporary music. I wonder what Bulent Arel felt when the premiere took place: he must have been very proud!

It's now 2007, and today in Turkey, we have many opportunities, with the latest technology and well-equipped studios. We have a great number of talented people interested in music technology and young composers who want to create interesting electronic music, following the path of Arel.

I am one of those composers. My biggest aim is to contribute something meaningful and significant to this aural world. I hope my personal artistic statement stated here below, will be my guide in the forthcoming years: *'The crackle sound of a lo-fi record player and the pristine sound of a digital processor . . . the chaotic recording of a daily street life and the sterile studio sound . . . the sounds of traditional Turkish instruments and their contemporary versions . . . This musical and historical integration moulds the wide time span into one immediate moment, where the listener actually is: the present. And this is my music . . .'*

PAULINE OLIVEROS

As improvising agents, computers may push us or teach us about the mind and facilitate a quantum leap into unity of consciousness. Technology should provide tools for expanding the mind through deep listening. Music and especially improvised music is not a game of chess – improvisation, in particular free improvisation, could definitely represent another challenge to machine intelligence. It is not the silicon linearity of intensive calculation that makes improvisation wonderful. It is the non-linear carbon chaos, the unpredictable turns of chance permutation, the meatiness, the warmth, the simple, profound humanity of beings that brings presence and wonder to music.

I continue to work on my Expanded Instrument System (EIS) so that I may play with the developing intelligence of the machine. This enables me to explore improvisation beyond the boundaries of my own carbon-based physical system in collaboration with silicon-based systems. Note EIS takes

any acoustic or electronic input and processes it with delays and algorithms that I devised. I use this system most often with my accordion. The system began in 1965 with my early electronic music such as *Bye Bye Butterfly* and *I of IV*.

CHRIS JEFFS

Why am I making electronic music instead of making tracks some other way? The answers are boring and obvious: I can make many sounds at once, I don't have to explain my ideas to someone else in order to hear them, and most importantly, I can potentially make strange, new, previously unheard sounds.

Object-oriented programming has led me to view pitches, rhythms and timbres as nothing more than sets of data to be manipulated. In my uber-sequencer, the *Cylob Music System*, a sound can be made up of a sequence of other sounds, and a sequence can be played as if it is a sound in itself, so the dividing line becomes blurred, because it's all just data; any distinction of these types is conceptual, and impacts on the constraints of the user.

As regards the electronic music-making scene, I feel that as access to electronic music has spread, so any sense of ambition or notion of quality has decreased, and I wish everyone to rediscover the feeling of novelty that, for example, greeted the first 'sample' records (remember that barking dog one, or much better, early Art of Noise?), instead of treading the same paths all the time. This might happen because too many people use the same sorts of software. These programs have largely reached a consensus on how things are done. It's really useful and important to me to make my own software, then, because I can aim to sound as different as possible from those who remain constrained by the interfaces they simply adopt. It is very time and energy consuming, but that feeds into the ambition, and I hope, into the quality.

RODRIGO SIGAL

Music technology is a flexible and ever-changing tool with which one can solve problems and set personal challenges. The possibilities for macro and micro control of structure have generated a specific discourse which we as composers are just beginning to develop. However, it is evident that there is a temptation for the electronic music composer to take refuge in technology at the expense of musical concept. For this reason, it is important to hold a self-critical perspective so that musical ideas may be understood in regards

to their musical coherence as opposed to the complexities of the technology utilised to produce the sounds.

This perspective is especially relevant in Latin America, where the use of technology in concert music is a key factor in fundraising. This has led to short-term projects based mainly on the provision of technological infrastructure without much regard to solid artistic and educational concepts. I believe the future of electronic music in Mexico and other countries of the region resides in taking advantage of the democratisation of musical tools (as in the use of laptops and computer software, perhaps specifically open-source programs such as SuperCollider, PD, Chuck and CSound). It will make us independent of government grants and truer to our own creative directions. The studio, in electroacoustic composition, instead of being the main source of musical technology, will become a place for information, education and the sharing of ideas to better shape our artistic future.

MIRA CALIX

I was just a little miss when I first got a copy of OMD's *Dazzleships* on record. I used to listen to it in the dark and was completely transfixed by the sonar bleeps and the screwed up voice with numbers on *ABC*. I couldn't figure out how all these foreign sounds got onto the record – I had no idea but it completely absorbed and delighted me. I think I was hooked from that point onwards on all things a little odd and synthetic. I knew that what I was listening to weren't 'real' instruments, like I'd heard before, but I couldn't figure out what kind of devices were making these strange sounds. The first electronic instruments I really got my hands on were an MC202 and a TR606; I finally had a go at making my own strange sounds in the dark, and I haven't stopped.

As far as the future of electronics goes: I'm fairly easy to please, and as long as I can get my hands on a good string sound, muck up my own voice, and sample a couple of pebbles, I'll be happy (although a 120-piece orchestra and a Cray would be nice).

DENIS SMALLEY

The acousmatic

The whole point of acousmatic music, expressed in the meaning of the word 'acousmatic', is that there is nothing to watch, no observable activity to confirm how the sounds are made, and often no certainty about where the sounds originate. The implication is that we should perceive and respond

to the sounds – the music – through listening alone. Acousmatic music is by definition an invisible sonic art, which invests in the liberty of an open sound world and in the imagination of the interpreting listener.

Visible music, a music produced by singers and instrumentalists, automatically carries a sense – the sense of its making. It is 'understood' because the means of production are understood, and there is an intimate liaison between means and sounding result: sound is the direct utterance of, or an extension of, the body and bodily expression. This evident human foundation ensures a basic understanding regardless of our sympathies for the particular genre or style of music that emerges. But with acousmatic music we have to search for an understanding in the relations and implications discovered within the musical context as it unfolds. Sometimes we will recognise, through sound alone, the tell-tale signs of human presence, of sounding materials, of the sounding world. . . . or we think we can; but sometimes we cannot.

Acousmatic experience is part of culture. We often glean information about our surroundings from hearing alone, deducing potential consequences from what we interpret. (Anticipating what might happen based on what we have already heard is also central to music listening.) But in navigating our way in daily life we are not that aware of hearing as an independent sensory channel because it is so tied up in a convergence of the aural, visual and kinaesthetic senses, all of which govern habitation and physical experience. When one sense alone cannot fully inform us, we call on other senses to help sort out discrepancies, so that we can arrive at a better understanding about what is going on.

It is a mistake to think that because acousmatic music is perceived via a single sense, that the other senses lie dormant. In our imagination what we hear crosses over into what we know of the world via the collaboration among the sense modes, even if these modes are not directly activated in order only to listen. For example, texture we know to be physical (touch), and visual, not just (or even primarily) aural; when we mentally take in the motion of sounds, we refer to our kinaesthetic experience and observation of motion – our experience of space. This is *spectral space* – a uniquely sonic version of space and spaciousness produced by occupancy of, and motion within, the range of audible frequencies. Articulating and exploring spectral space is inherent to acousmatic music, and is often aligned with the perspectival aspects of space – the experiences of proximity, distancing, approaching, opening out, closing in, emerging, disappearing. There is an affinity with environmental experience.

From sound alone I may constitute, in my imagination, the configurations of a scene or narrative, or a spatial climate; the visual and physical appearances of objects and textures; the motion of entities and shapes in

spectral space. These may have a certain reality if I think I can identify possible sources or causes of sound, but they may equally be fantasy, a kind of 'vision' where the auditory sense calls on companion senses to participate in interpretation. This conjunction of imagined sensory phenomena may be captured in the idea of the 'acousmatic image'.

Acousmatic experience is more prevalent than many realise, and the development of technology has brought us new means of acousmatic experience. The telephone, in its masking of visual identity, concentrates our attention on the sound of the voice; it is remarkable what one can tell about the person on the other end of a phone line based on minimal phonemes of greeting: such timbral signatures can not only identify a particular person, but can also tell us about state of health or mind, via the manner of an utterance rather than what is actually uttered. This underlines our sensitivity to spectral detail and the subtleties of meaning carried by recognisable acousmatic sound-types.

Radio continues to be a robust acousmatic medium despite supposed competition from visual media. In our aural imagination we construct mental images of people, scenarios, and places, and programmes that explore mixes of sounds, music and voices might be regarded as a genre of acousmatic 'art'. With radio I am particularly interested in the spaces we associate with the sounds we hear; recorded sounds do not acoustically carry with them all their spatial attributes. Consider mono radio, which can technically carry depth perspective, but not panoramic breadth or spatial elevation. What we recognise are *source-bonded spaces* – the spatial zones and mental images produced by, or inferred from, a sounding source and its cause. Source-bonded spaces carry with them an image of the activity that produces them. In acousmatic music this is particularly important not only with sounds we recognise, but also with more ambiguous sounding contexts, which can take on attributes of known sonic behaviour.

Film absorbs acousmatic sound and music, and did so before electroacoustic music was developed. Acousmatic sound can be used to circumvent the physical frame of the screen, drawing attention to the fact that we rely on sound to gather spatial information. And acousmatic sound can also be used to create psychological ambience and states, as well as steer fluctuations in temporality. Acousmatic conventions have evolved in order to transcend the constraints of the silent visual image.

This brief sketch of acousmatic media (or in the case of film, a medium in which the acousmatic participates) serves to indicate that there are sets of expectations, or schemas, associated with acousmatic contexts. These are carried over into the comprehensive sound world of acousmatic music.

Many electroacoustic composers, and performers of electroacoustic music, are preoccupied by 'liveness', with rooting electroacoustic music in

the continuing tradition of instrumental or vocal gesture, of 'visible music'. Instrumental performance and acousmatic sound have tried to collaborate since acousmatic music began, but the relationship between the visible and the invisible has not always been comfortable: the acousmatic offers adventure while the instrument constrains the musical 'language'; the acousmatic taps into mental schemas of which instrumental gesture and human sound-making are but a part. Too often, the basic convention of relying on limited types of live processing of instrumental sounds to form a parallel fabric, and to create a semblance of interaction between the seen and the heard, results in a very predictable music and an interactive play appreciated more by composer and performer than by the listener.

With new performance interfaces (many of which have a fleeting life), and with laptop performances, what purports to be 'live' is often, from the listener's point of view, too acousmatic, in that the relationship between what is witnessed visually and physically, and the music emitted, can be so covert that the presence of 'performers' is mystifying. I christened this phenomenon *live acousmatic performance* as long ago as 1992. If only primitive vestiges of performance articulation remain, composer and performer need to realise that the covertness of their acts requires some elaboration in order to justify the public nature of the event, possibly by creating a richer audiovisual context. Furthermore, a greater awareness of acousmatic schemas and the meanings carried by sonic configurations, might aid the creation of a music better adapted to acousmatic reception conditions. If in seemingly live, public events, sound and music are to take on acousmatic guises, the one requirement is that they should richly do so.

Today, the prevalence of acousmatic sound and acousmatic music listening means, generally, that the need for a live listening experience is less of a concern for the listener, who now accesses music and sound via CD, SACD, iPods and computers, listening in domestic settings or on the move. In a way, the liveness of music lies in its nowness – its happening at the moment of listening. Whether the sound is physically created at that moment, or is capable of being generated or assembled in the instant of reception, is of secondary concern.

It is often said that acousmatic music could better communicate if aided by visual images. While I recognise that something unfamiliar may initially be disquieting, I do not favour a visual add-on strategy, particularly bearing in mind the transmodal perceptual nature imbedded in aural perception itself, and the vast expressive range of the purely sonic acousmatic 'image'. There is no need to use visual images to direct the listening imagination, no need to serve out semiotic supplements that restrict the 'meaning' of the music, no need to hold the listener's hand. Not that I am against audiovisual arts, given a properly considered trans-sensory conception. Listeners

need not be apprehensive about their personal aural imagination, or worry whether a particular interpretative response is expected.

The twenty-first century is an era where technologically mediated creativity offers an expanded awareness of the aural sense as a dominant, and highly valued channel for aesthetic communication. Acousmatic music, which embraces a mix of tendencies, offers us the richest realisation of this human need.

SEONG-AH SHIN

In 1990, when I was a freshman at the University of Korea, the department had a very small electronic music studio. It contained a NEC8081, an eight-bit computer that only ran Common Music, and a Roland 100M, an analogue synthesiser. Most students called the studio the 'ghost room' because of the strange sounds which emanated from that dark corner of the building. However, for me, it was the most interesting room in the department, with new sounds and fascinating equipment. It was not that easy to study electronic music; I still remember that reading the manuals, written in English, was like code-breaking.

I cannot even compare that studio with the quality of equipment and documentation we have now. I deeply appreciate the advances of technology in the recent years, but I also value the patience I learned through working with older equipment.

Now, I teach electronic music at my old school. No longer the ghost room, all the students are required to take the electronic music class and most of them enjoy it. I believe technology helps composers and students focus their musical imagination and express their emotions.

In my own work, I find and create sounds, allowing me the profound pleasure of living more than one life. I compose music in the present with sounds reaching into the past, guiding the future structure and substance of the sonic result. For me, this process of connection is composition, linking past, present and future – the inevitability of the musical experience transformed through technology and the paradoxical mirror of time.

CARSTEN NICOLAI A.K.A. NOTO/ALVA NOTO

I was mainly working as a visual artist when I made my first recordings in 1994–5; these originated from researching materials that would actually dematerialise. Without being aware of it at first, the sound experiments I did at that time were actually what I was looking for.

My main focus then was, and now still is primarily oriented towards the question of how time is perceived through sound. In the works under my pseudonym noto – that focused on the physicality of sound and its instantaneous effects – I did this very empirically by varying loop speeds and altering pitches to watch if and which emotional reactions I experienced. Also, I tried to find out more about other qualities of sound: the tracks were based on sine wave tones and avoided overtones, melody and musical orders, to rather establish structures that sprang from visual patterns or mathematical models. In experiments with ultra-high and ultra-low frequencies I pushed those explorations to physical and material limits.

But I was also interested in looking at the effects that occurred when visual structures are transformed into audible patterns and vice versa. In terms of synaesthetic perception I constructed some installations (*wellenwanne*, 2000/*telefunken*, 2000), where I documented sounds creating visible patterns. This was to demonstrate their hybrid value – to show and sensitise the viewer/listener for how those spheres intertwine. For my compositional work this has always been very important, as among other principles I used visual structures as a basis for tracks.

The notion to introduce all these experiments to a more musical approach came later when I did my first recordings under the name of alva noto. Here I tried to strip bare pop music standards to a minimum level of rhythmic structure and fathomed strategies to explore extreme poles in my collaborations with other artists: from micro-structure experiments to working with acoustic instruments.

In the context of the complete merging of our information system in the digital sphere – accompanied by the phenomena of acceleration, fragmentation and dispersion – I developed my own strategy to assess personal space. In my *transall series* I misread and transformed information, leading me to unusual results that challenge the rules of existing standards. You can listen to the rhythm of the code which by transformation reveals its sheer aesthetic.

My latest project *xerrox* engages with erroring and mutation. It observes the gradual transformations the sound undergoes by constant copying. The process of de- and recomposing of the musical source causes inherent errors and results in self-generated mutations that finally overcome the original and take over its essence. One of my key references for these experiments – and for many of my former works as well – was a text on artificial intelligence, Takashi Ikegami's article 'Active Mutations of Self-reproducing Networks, Machines and Tapes' (1996), which engages with the logic of self-regulation of cybernetic systems. He describes the emergence of new forms and patterns by mutation intrinsic to a system. Ever since, this has been very influential to me, and is as present in my early works based

on loops, white noise and mathematical principles as it is in my current projects.

Over the years I have had the wish to combine all of those ideas into an overall project simultaneously, a work, which would bring visual elements, sound and space together. An attempt to do so is the installation *syn chron* (2005), which fuses an interplay of laser light projection and sound with a crystalline shaped, architectural form. It enables the visitor to get an holistic experience from interacting elements of time and space. In this sense it comes close to an ideally autonomous Gesamtkunstwerk, which could prospectively, as a consequence, allow me to disappear.

WARREN BURT

Contradictory thoughts on electronic music

'Electronics has changed everything in music.'

'Electronics has changed very little in music.'

'We have extended our bodies and our consciousness in ways undreamed of.'

'We are still dancing, singing monkeys.'

(We were dancing, singing monkeys before we became jabbering apes.)

It's not a question of whether electronics. They're essential. But we shouldn't have an exaggerated idea of their importance.

I was studying Indian music. My teacher said, 'In Indian music, we say that you must sing the music before you can play it.' 'Like this?' I asked, doing my best vocal imitation of the first thirty seconds of Stockhausen's *Kontakte*. 'Precisely!' my delighted teacher said.

The biggest change electronics have made to music is sociological. It's certainly allowed the exploratory, inquiring side of classical music to survive the fossilisation of the orchestra . . . the interesting thinkers just went elsewhere . . . along with their machines.

I will allow that electronics has made some things faster and easier. For example, microtonality and works of extremely long duration. But people have been making microtonal music for centuries, and anyone who has ever heard a Hindu priest chanting the thousand names of the goddess knows that minimalism didn't begin in the 1960s.

'Electronics has removed the body from music.' Yet, the equipment is still operated by bodies, and bodies are listening to it, and even dancing to it. Is electronics any more of a conceptual extension of the body than, say, a clarinet?

Is there any music that's truly native to electronics? That couldn't exist without it? Maybe. Maybe a realtime algorithmic interactive piece with synthetic timbres is native to the medium. I don't think there's any way to do that without electronics. But I might be wrong.

MAX MATHEWS

The past and future of computer music

The year 2007 marks the fiftieth anniversary of the birth of music synthesis on a digital computer – an IBM 704 computer which filled a big Madison avenue showroom in New York City. At that time, I worked for the Bell Telephone Laboratories who rented time for me on the IBM for four 1957 dollars a minute. My job was studying new telephones with the computer. But, sound-is-sound, and if you can make speech on a computer you can also make music.

During this half-century the musical computer has changed from a huge expensive machine, affordable only by universities and research laboratories, to a laptop which almost anyone can afford. It took the IBM minutes to compute a second of music with limited timbres. Today's musical laptop is small enough to fit in an orchestra chair and can generate rich timbres in realtime, that is to say, it can be played live as any other traditional instrument.

A comparison of the IBM 704 to my current laptop is revealing:

ASPECT	IBM-704 (1957)	IBM-G41 (2005)	RATIO
CLOCK	1/10 MHZ	3 GHZ	30000
RAM	192K BYTES	1 G BYTES	5000
MEMORY	6 M BYTES (TAPE)	80 G BYTES (DISK)	10000
COST	\$200/HR (RENT)	\$2000 (BUY)	
SIZE	FULL ROOM	7 LB LAPTOP	

Computer technology no longer limits computer music. Now we can attack much more interesting problems than computer programs.

In the past I have often quoted the SAMPLING THEOREM which for music says: 'ANY SOUND the human ear can hear can be made from digital samples.' I still believe that beautiful as is a violin's tone, neither it nor any other musical instrument can make this claim. However, I have recently added a corollary to this theorem:

COROLLARY (MATHEWS-2006) For musical purposes, in the class ANY SOUND, almost all timbres are uninteresting, and many timbres are

feeble or ugly. Moreover, there is a great temptation to try to strengthen weak timbres by turning up volume controls thus creating dangerously loud sounds. It is VERY HARD to create new timbres we hear as interesting, powerful and beautiful.

Instead of technology, new music is now limited by the limits of our understanding of the perception of music by the human ear and brain. What is it in music that we find interesting and beautiful? What sounds do we find dull and unpleasant? What is learned from the music in our lives? What is inherent in our genes? These questions are much more challenging and interesting than writing new computer programs. Progress will require much more research in musical psychoacoustics. Computers have already supplied new tools to carry out this research. New brain-scanning equipment is already showing us what parts of the human brain are activated by music.

But research itself does not lead directly to new music: only musicians and composers create new music. We also must provide new education for these musicians. I believe new courses with names such as Orchestration for Electronic Music and Ear Training for Electronic Musicians must and will soon be added to the composer's training.

Computers in the last fifty years have changed our lives in many many ways. But I believe this is only the crack in the door and in the next fifty years we will see great progress in our understanding of our musical selves. Computers will help us in our quest by synthesising and analysing a lush jungle of new timbres which our ears and brains will hear and judge, will accept and reject, will love and hate, and which we will learn to understand and use in our music.