Return to Index

HACKING SOUND IN CONTEXT

Alex McLean
alex@state51.co.uk
state51, Rhoda Street, London, E2 7EF

Without software, a computer is like a stone. Consider a program that generates organised sound - what could it be but a music score, composed by humans? I suggest that ultimately, software can be nothing but human expression.

This paper is an exploration of the relationship between the programmer, the programming environment, and computer music applications.

Meaning of "Hack"

"Non-technical people seem to be largely unaware that the word "technician" is an insult. Like the term "lower class," it generally refers to someone else. What "technician" usually means is somebody whose work is less creative than your own."
Ted Nelson, 1974

The term "Hacker" is not so much misunderstood as over-appropriated. The first hackers were the members of a particular dorky culture of model train enthusiasts and phone phreakers at Massachusetts Institute of Technology. Later these young geeks found a computer, and made history by having a lot of fun with them, spending many hours at a time exploring computational possibilities. From then on, the label "hacker" stuck on those who revelled in the resultant new, wild realms of technological creativity.

Unfortunately journalists have generally ignored all the subtle nuances around the word, and instead use it to describe a computer criminal. But still, within many sections of geek culture, a "hacker" simply means a wildly talented programmer. That's the meaning I use, along with "hacking" to mean creative, individualistic programming, and "hack" to mean a creative (or weird) computer program.

Feel of Languages

"I would recommend that every student of music go dancing at least once a week. And dance. Please, really dance: three of four hours a week."
Karlheinz Stockhausen, 1989

In his book "The Aesthetics of Computing," David Gelernter enthuses at great length about "quicksort," a fast, elegant way of sorting data. He finishes his long and rather intense analysis with a jubilant, italicised line:

"Quicksort is powerful and simple: a beautiful algorithm."

I wonder how he would have felt if he had stumbled across the London "Perl Mongers"

pub meeting one dark London evening. The scene was set by drunken programmers fuelled with real ale, stumbling across the insane idea of acting out a human, sonified bubblesort. Bar staff could only stand and stare as grown men and women stood in an unruly semi-circle, singing instructions "Compare!", "Swap!" or "Stay!" at each other, choosing musical pitch reflecting their comparative heights.

Well, it was a strange sight, and a stranger sound. But perhaps it shouldn't be surprising to see programmers dancing about to algorithms. Consider this mailing list post, part of a discussion about the 'feel' of computer languages.

"I've always pictured programming as a dance of sorts, very slow and each gesture receives a great deal of attention, so that a limb or a step would not stray from a particular path. Loops look like pirouettes to me, I/O feels like delicate gestures with one's hands, no, just the fingertips drawing patterns in the air. Conditionals and cases are tumbles and jumps (into the air, that is). Arithmetic is kind of hazy, controlled slides with no traction (think 'slick moves'), things being pushed and brushed aside."

A program in execution is a program in motion. Control flows around the program, taking data with it. Data flows into the program, and is breathed out again. A hacker staring intently into her screen is probably turning somersaults in her mind.

Free Software

"Computers're bringing about a situation that's like the invention of harmony. Sub-routines are like chords. No one would think of keeping a chord to himself. You'd give it to anyone who wanted it. You'd welcome alterations of it. Sub-routines are altered by a single punch. We're getting music made by man himself: not just one man." - John John Cage, 1969

As a programmer who writes code to compose music, I was heartened to find this quote. Back then, Cage saw computer language as an expressive tool, and was excited about the possibilities that code could bring to music. But there is an implication that I'm not happy with; that to program is to lose a sense of individuality.

Every programmer I know has a very different programming style. This is especially true of very rich languages such as Perl, where often programmers will have such different approaches to the language that they will be unable to understand each other's code.

"Many computer scientists have fallen into the trap of trying to define languages like George Orwell's Newspeak, in which it is impossible to think bad thoughts. What they end up doing is killing the creativity of programming."
Larry Wall (the creator of the Perl programming language), 1997

Programming has changed since 1969. Thanks to the wider choice of languages and interfaces, programmers now make all the rules, making far more decisions about style and structure. Coding has become an individualistic past-time.

On a different tack, for the last couple of decades many people have kept sub-routines to themselves. Whether this is good or bad is subjective. The act of releasing source-code can be greatly rewarding in many ways, but if code is being written to make art, then we must surely respect the artist's decision to keep their methods secret. The jury is very much out on whether open source is good for commerce, but as far as the individual artist is concerned, the decision will always be a personal one.

Silent collaboration

Computer software operates on many levels. Many thousands of people have made building the lower levels their life's work. You may never see the source for the microcode, kernel, device drivers or libraries that work continually inside your computer, because you don't have to. But these systems and structures are there, and the industrious, synchronised tree of processes that silently exist before you are the result of many tens of thousands of man years.

A computer program is never created in isolation; the programmer is always indebted to silent and sometimes anonymous collaborators. The creators of the operating system and programming language provide the environment for the programmer's work to execute within. Using library code saves the programmer days, or perhaps years of work.

If you view a programmer as a creative individual rather than a technician, the computer ceases to be a grey and lifeless box, but an environment formed by human expression. You begin to see why programmers become so engrossed in this medium... In an environment formed by human expression, possibilities are seemingly endless.

The Birth of a Mailing List.

A mailing list is started to discuss the philosophy of programming. Like-minded people meet, make friends and share thoughts. The mailing list grows in size and intensity. For some reason, some enthusiastic members discuss how the mailing list archive could be turned into music. Sometime later, someone writes some code to do it.

Sounds start calmly and slowly, with occasional flurries of activity. Subject lines slowly scroll up the screen, with the authors flashing up below. As time goes on, the music increases in intensity, with more diverse sounds reflecting the greater number of threads running at the same time. The night-time pauses for sleep gradually become less pronounced as the subscribers cover wider parts of the globe. The music becomes manic as traffic increases, until becomes impossible to read the author's names as they flash by. And then, just as the relentless furiously is becoming really far too much, it all stops. There are no more posts yet - we have reached present day.

The music is performed at a party, and the effect on the members of the mailing list is extreme. This piece of code has brought a community close to their history. The hacker grins.

Audibility of Code

When we listen to electronic music we can sometimes identify the software that was used to create it - we 'hear' the original programmer's code. Perhaps this is evidence of the creative input a programmer has on every piece of music produced using her software?

This "audibility of code" is particularly evident in what is labelled generative software, such as koan pro. The process-based nature of the composition puts a great deal of control in the hands of the original coders. And so, using koan might be viewed as performing a duet with the creators of the software.

Flower

"When I started working with computers I had to learn to visualize what was going on in the machine. ... Curators and critics that look at net art have to go through this process as well, but they may not realize it."
Mark Napier, 2000

When I create a program it is in much the same way as I might create a piece of music in a sequencer. I have an abstract idea, concentrate on the most tangible part and place it in the computer as code. Then, I form more parts around it until a coherent whole appears. Some parts may get deleted, but have made their mark on the parts that formed around it.

After hearing "Speedy J" perform some gut-wrenching techno, I found myself thinking about movement in music. Thinking in the abstract, I imagined a network of loops, with things moving around making sounds. After a few hours I came up with what I now describe as insects moving around a flower, with several sound triggers on each petal. This was an arbitrary description as it had no visualisation or indeed any interface at all beyond three inputs - intensity, complexity and disorder.

It starts off with a circle. Increase the intensity and insects begin to travel around it, decrease and they're killed off one by one. Increase the complexity and petals get added to the flower in random positions, decrease to pluck them away. The insects may choose to travel around the petals too. Each petal has certain sounds along it which the insects play as they run past. If independence is on, the insects ignore each other, otherwise they interact. Increase disorder and they choose which petals to visit more randomly, decrease it and they fall into rhythm.

Then I start up another flower or two with different sample banks and they synchronise through a central server, making nice time-based counterpoints.

I had only a vague idea about what the program would sound like, but was stunned by the results -- the music is random at first. But with an intensity of 3 and complexity of about 9, the music gradually settles down to a beautiful polyrhythmic pattern. Increase the disorder a little and the beat would occasionally skip into randomness again before settling down to a different rhythm.

Rest

A well written computer program is the perfect resting place for its author. Programs originate in the human imagination, and so programmers know their creations extremely well. The first run of a program is therefore a precious time. The programmer's imagination is built into something intangible, but with presence gained from timelessness. Thoughts solidify into code, and become fluid once more in execution.

But a program is never complete, there are always more ideas to be eternalised, more algorithms to be perfected and more data to be accommodated.

Bibliography and recommended reading

1.Music and the Mind, Storr, Harper Collins, 1992

2.The Sound of Painting, Maur, 1999

3.John Cage, Writer - Selected Texts, Ed. Kostelanetz, MIT Press, 1993

4.The Computational Beauty of Nature, Flake, 1998
http://mitpress.mit.edu/books/FLAOH/cbnhtml/

5.Ocean of Sound, Toop, Serpents Tail, 1995

6.How I drew one of my pictures, Ward, 1999
http://www.generative.net/papers/autoshop/index.html

7.Computer Lib/Dream Machines, Ted Nelson, 1974
http://xanadu.com.au/ted/

8.The New Hacker's Dictionary - second edition, Ed. Raymond, MIT Press, 1994
http://tuxedo.org/jargon/

9.Talking Music: *Conversations with John Cage, Philip Glass, Laurie Anderson, and Five Generations of American Experimental Composers*, Duckworth, Schirmer Press, 1995

10.Aesthetics of Programming: Interview with Mark Napier, Brogger, "ON OFF", 2000
http://www.afsnitp.dk/onoff/Projects/napierinterview.html