Preface

The field of computer music can be thought of as having two fundamental branches, one concerned with the manipulation of musical sounds, and the other concerned with symbolic representations of music. The two are iconized by Max Mathews's MUSIC program and Lejaren Hiller's *Illiac Suite*, both of 1957, although both have important antecedents. The two branches might provisionally be given the names "Computer Generated Music" (Denis Baggi's term for it) and "Computer Aided Composition"—or CGM and CAC for short. (In France the latter is called "Composition Assistée par Ordinateur". The corresponding English acronym, "CAC", is less than mellifluous and someday we should settle on a better one.)

As a field, CAC has flown a very different trajectory from CGM. While in the United States the great strides between 1957 and 1980 were on the CGM side, in Europe during the same period we saw work by Xenakis (starting as early as 1962), Koenig (*Project 1*, 1964), and many others that, taken together, can be seen as the first proof that CAC could be widely useful in creating new music. Meanwhile in the United States, many people, myself included, thought of CAC as a topic of computer science research, not likely ever to give rise to tools useful to musicians. Interest in CAC has grown in the USA in the intervening years, but excepting the brilliant example of David Cope, work in the USA on this topic has lagged behind that in Europe.

Today CGM is ubiquitous and CAC appears futuristic. An entire generation of composers and other musicians has learned to use the computer to synthesize and process musical sound. It can be argued that the computer has been the one addition to the classical orchestra since the advent of percussion early in the twentieth century. This is a great achievement. CGM is now generally accepted, and the status of a musician in the Mathews tradition essentially depends on how good his or her output sounds, in the same way as that of an orchestral string or wind player. CGM has become a normal, respectable, middle-class occupation.

The development of CAC, on the other hand, has seen a deepening realization that the problems of the field are much more difficult than they may have appeared at first. In hindsight, this should have been obvious to everyone all along: CGM is in effect building instruments (which were previously made of wood and the like), but CAC is in effect making the computer carry out thought processes previously carried out in human brains. Clearly, a piece of wood is easier to understand than even a small portion of a human brain. Ultimately, CAC researchers will have to settle for much less than a full understanding of even a single musical phenomenon. The best that can be hoped for is partial solutions to oversimplified versions of the real problems.

Computational Complexity

From my point of view, having come to computer music in the year 1979, the comparative situations of the two branches in the year 2006 come as a surprise. To the user of a 16-bit

DEC PDP-11 computer, the manipulation of musical symbols looked trivial compared to the hard practical problems posed by the sheer size of the problem of sound synthesis. A five-second, one-channel soundfile was a large object in 1979, and the realization of a piece of computer music lasting ten minutes could easily require a solid week of computer time. Back then I was exceedingly lucky even to have that possibility. Now, in 2006, three IRCAM pieces from the 1980s can be run simultaneously in real time on a computer small and light enough that it could easily be thrown several rows into the audience.

The symbol manipulators, on the other hand, whose programs once only took a few thousand arithmetic operations and a one- or two-inch stack of punched cards to run, now take their place as the heaviest computer users in all of music. The growth in complexity of CAC algorithms appears to have outrun the ability of computers to run them. The Markov chains of the early days (a few hundred arithmetic operations per note, say) have given way to combinatorial search and optimization problems requiring as many trillions of calculations as the user can afford to wait for. The imagination of composers and researchers in CAC far outstrips the supply of available computation power.

In general, the cost of CGM per audio sample has not remained constant, but has not grown quickly. The best CGM of the seventies, thirty years ago say, probably cost less than ten thousand arithmetic operations per sample of output. The speedup in computing in the intervening years has allowed us the luxury of running the classic CGM algorithms in real time, thus changing the nature of the pursuit fundamentally. The algorithms themselves have grown somewhat more complex as well, but the universal preference for real-time synthesis and processing has put a lid on that growth.

CAC has not become real-time at all. Since it appeals to the composer in us (whereas CGM appeals to the performer in us), it seems reasonable to expect that CAC software will continue to absorb all the computing resources that can possibly be brought to bear on it. Anything allowed to grow will naturally do so.

Programmers and users

About 1970, when I started using the computer—there was only one in my town—there was no distinction between computer users and computer programmers. It only occasionally happened that someone used a program that someone else had written. For the most part, each program was written for its own particular, special purpose. Over the following decade, however, this began to change for two reasons. First, people learned to write large, flexible programs (troff being my favorite example) powerful enough that different users could turn a given program to very different purposes.

Second, and more subtly, the possibilities for combining different bits of software together began to multiply. An important step was the invention of the Unix operating system, which unified all I/O through one very simple and clean abstraction. This permitted users to direct one program's output to another program's input, frequently without the need even to choose a name for the data passing between the programs. This made it possible for a relatively simple program such as `tr` to be put to all sorts of different uses. Try it on a headerless soundfile, for instance, to put a classical music lover's teeth on edge, in a way I doubt the program's original designer had imagined.

A parallel development was underway in the computer music community. Here the roots of the idea reach all the way back to about 1958 when Max Mathews's MUSIC N programs began to offer reconfigurable *unit generators*, which the user configured in a

network to generate musical sounds, predating and anticipating the modular synthesizers built by Moog and others in the 1960s. By the mid 1980s many researchers were thinking about trying to turn this notion of reconfigurability to use in passing data of other formats than audio signals.

Programs themselves (such as MUSIC, Max/MSP, or OpenMusic) have become complicated and difficult to develop, but once the paradigm for making interconnections has been worked out, they are comparatively easy to extend. Users can contribute extensions and benefit from each other's work, without having to worry about the tricky stuff such as GUIs or file formats.

In another sense, however, a patch is itself a program, and the job of connecting simple functions together to make larger ones can be thought of as programming. In this sense, the trend toward patch-based software can be seen as shifting the level on which the user programs the computer away from the C or Lisp code itself and into the "language" of patches. It may be that this is fundamentally a better level at which to operate a computer, than either that of code or that of the user of a large, monolithic program such as a database application.

Art music and the computer

In Europe and its former colonies such as the U.S., composers, since early in the twentieth century, have paid much attention to problems of symbol manipulation and combinatorics. This idea found an early expression in Schoenberg's 12-tone harmony, continued through the serialism typified by Webern and later Boulez, and may have culminated in the various mathematics-inspired approaches of Xenakis. The affinity of composers such as Xenakis and Koenig for computers seems to grow naturally from their symbol-based and/or quantitative approaches to musical composition.

It is no accident that computers were used in experimental classical composition, whereas more tradition-bound musics such as Jazz stayed far away from the computer room. And researchers in CAC repaid the compliment by paying close attention to sets and permutations, and less so to melodic contour, for example. To this day, the field of CAC looks primarily to the classical 'art music' tradition as a source of working assumptions and problems.

Since computers are well adapted to symbolic and quantitative manipulation, it is not surprising that 'art' composers have often turned to the computer, sometimes merely for assistance, and sometimes for inspiration. The strange field called "computer science" (which has little to do with writing or using computer programs) is often invoked as well. Certain metaphors from computer science, such as hierarchies and networks, machine learning, and database operations, often can be explicitly mapped to musical processes, and this is useful to some of the more formally procedural composers.

I think these formalistic tendencies are now giving way to a more intuitive approach among 'art' composers. Whether or not that is true, there is certainly more crosstalk today between 'art' composers and musicians of other idioms. This is reflected in a general movement in CAC away from formal and mathematical tools, in favor of more intimate modes of interaction with the computer, even up to direct manipulation of data structures by composers. So for instance when early CAC researchers wrote computer programs whose output might be an entire piece of music, today we see developments such as OpenMusic which, in their graphical orientation and patching metaphor, encourage

the composer to proceed by experimentation and intuition instead of by formal planning and specification.

The field of CAC in general is moving away from mathematical and computer science constructs, and toward a more useful and powerful working relationship with the rest of the composition process. A greater fluidity of interchange between the problem-solving or material-generating functionality of a program such as OM, and the higher-level, partly intuitive thought processes that must reside in the human brain makes the entire field of CAC more accessible and more widely useful than ever before.

Software and Computer Aided Composition

In CAC, the variety of approaches and the flexibility of applications have grown as time has passed. Back when computer programs used stacks of cards as input and output, it was natural to think of "composition" as an atomic computer job: in go the program and some parameters, and out comes music. As computing became interactive, a much more powerful mode of working emerged, in which the computer might be called on hundreds or thousands of times to solve specific problems, such as voicing a chord or quantizing a rhythm.

Lisp, which is widely used in AI circles, was an early favorite among CAC researchers. In return, the AI community, especially around MIT and Stanford, has long taken a strong interest in music. The history of CAC software is dominated by large Lisp-based systems. Perhaps the most important advance in the field of CAC was Patchwork by Mikael Laurson, a direct ancestor of OM. Patchwork (which Laurson still develops) presents the user with a patching GUI, in which the semantic is that each object, to produce its output, asks the objects upstream of it to compute their outputs, recursively. This demand-driven dataflow model is also used in OM, although the nature of the function calls has been greatly generalized compared to those of Patchwork.

Central to the success of both Patchwork and OM is the presence in each of tightly integrated music notation display packages. A transparent connection is maintained between the displayed score and the (user-accessible) data underneath, allowing for easy transitions between the two media. This greatly enhances the ability of the user to tightly integrate the algorithmic part of the work (on the data structures) with the intuitive aspect (in the composer's mind, transmitted via the notation). Initially, the notation GUI functions as an invitation to composers to try the software. After the composer is attracted, the notation package serves as his or her personal interpreter to the language of Lisp.

That the developers of Patchwork and OM have actively sought to involve composers in the earliest stages of the design of the software is itself another decisive reason for their success. Few centers, anywhere in the world, have ever managed to match IRCAM's simultaneous ability to attract world-class music production projects and to support researchers in the field of computer music. Even though state support for IRCAM has eroded since the golden age of the 4X and the ISPW, the creators of OM maintain this spirit, of which the present book is an important manifestation.

OM is almost certainly now the world's dominant platform for doing CAC research and practice, despite the presence of several other approaches (including one, by Karl-heinz Essl, that runs within Max). Is this because OM's design is the best, or is it that OM has benefitted from the presence at IRCAM of so many willing composers, such as

the ones represented in this book? The two rival explanations are impossible to extricate from one another.

This doesn't imply that all interesting research in CAC is being done in OM. David Cope's work seems to me the most interesting CAC research from a theoretical standpoint. His particular software solutions belong to the class of "automatic composition" programs and are hence less adaptable to the needs of the main body of composers today than systems such as OM. I hope someday to see his ideas brought out in more modular form.

Promising areas of current and future research

An excellent trend is underway, and has been for at least several years, in that composers of computer music today no longer immerse themselves in one primary software package to realize works. The possibility of passing between one world and another (such as Max and OM, for example) would not have occurred to many researchers or composers in the days when mastery of any one idiom could take years of study and work. But as computer music software in general has become more open and more modular, the opportunities for interchange of data have increased, and at the same time the initial cost (primarily in time) of using a new software package has gone down. Many new and interesting sparks should fly from the collisions between the vastly different software packages that now can be brought together.

Related to this, perhaps even a case of the trend toward interoperation, is the growing involvement of CAC in manipulating sounds directly (not through the mediation of a score). Such work lies simultaneously within CGM and CAC, and in one possible future the distinction between the two will simply disappear. This is in high contrast to the early days of CAC in which the output was a stack of punched cards, or even to the situation only ten years ago, in which Patchwork users needed MIDI hardware to hear their musical ideas. (CGM people like me scoff at the practice of using MIDI to synthesize music.) The world of sounds is much richer than the world of musical note heads and stems. The latter will always be a useful organizing and mnemonic device, but the former is what actual music is made of.

In more general terms, I look forward to an increased concern in CAC about continuously variable quantities, such as parameters of analysis or specification of sound (as well as the function of time that represents a recorded sound itself). In the future, computer music in general will have to deal with high-dimensional objects such as would specify a timbre. The dimensionality of a set such as the "set of all possible sounds" is probably not even well defined. New techniques will be needed to describe and manipulate such quantities.

I'm also very excited about the recent work in OM on generalizing constraint problems to optimization problems. The cool thing about treating compositional problems as optimizations instead of constraints is that you can juggle the weights of the terms of the function and watch, incrementally, as the machine tries to optimize the ever-changing function. This can be used to find solutions to standard constraint problems, by adding and dropping component constraints and watching how the solution changes. It's almost never interesting to see ALL the solutions of a constraint problem anyway; most of the time there's either no solution or else there are lots that you'd be equally happy with.

Next, I would like to see some of the techniques now only available in OM become usable some day within real-time environments. This is clearly a huge undertaking, since the style of programming currently used in real-time applications is so different from that in OM. But there would be much gained if this became possible. In the meantime it's possible to send messages back and forth between OM and some lower-latency process that takes care of real-time performance. But in the ideal, the connection between the real-time and the compositional environments would be much more based on sharing data and functions, rather than just communication protocols.

Another word for real-time composition is "improvisation", and in this view George Lewis was doing CAC research when he developed Voyager in the 1980s, and Salvatore Martirano was apparently also doing CAC with the SalMar in the 1970s. Improvisation is not only important in its own right, but also as the primary means by which composers and performers have extended instrumental language, probably long before music was ever written down. Improvising with computers will lead us to a greater mastery of the computer as a musical instrument in composed settings as well as improvised ones, and I think CAC will play an important role in this development.

One more area. I've mentioned David Cope's automatic composition project, and although I can't say I understand his methods very well, it's clear that his use of natural language recognition tools (in particular, "augmented networks") has somehow captured something essential in the way classical musical forms work. No other research I'm aware of has ever gained any real purchase on the questions of how musical motion, tension and resolution, and musical themes work; and no other software algorithms I have seen can make music that develops (in the classical sense) over the time scale of an entire piece in the way Cope's can. There is clearly something vitally important in this work, and other researchers (myself not excepted) should be making a greater effort to learn its lessons.

Read this book

Perhaps I have named the frontiers well and perhaps not (only time will tell), but at least as far as the present is concerned, this volume contains the best summary of current work in the field that I know of. That each chapter of the book concerns the composition of a real piece of music reassures us that the methods described here can indeed be brought to musical fruition. And while, by the design of the book, composers using software other than OM aren't represented here, perhaps two thirds or three quarters of the entire field creeps in at one spot or another. OM has established itself as the most important locus of convergence of researchers and composers working on, or in, CAC; and, well, here they are.

This book should prove useful not only to those wishing to learn how to use OM in its present state (at least at the moment before OM develops further or is replaced by something else) but also in the longer term as a repository of ideas, many of them having roots in the past, even in non-computer-music, and many of which will reappear in different musical and software contexts in the future. This is how music works, after all—a musical idea is important in the way it speaks to the rest of the world of musical ideas. It's not the individual notes that count: it's their interrelationships.

Miller Puckette
February 20, 2006