ICMC 2003 Keynote Address

Computer Music as a Research Community

Roger B. Dannenberg

Introduction

I am honored to be invited to give this address. I have been involved in computer music for over 20 years now, and although I have had many excursions into other fields, I feel most at home working on computer music technology and creating music with computers. The International Computer Music Conferences (ICMC) and the International Computer Music Association (ICMA) have therefore been very important to my career and my sense of direction. The most important part of these organizations is the people involved, and I would like to talk about these people, about us, as a community. This document is not a literal transcript of my keynote address, but rather an approximate recreation. I welcome any comments you may have.

I really want to address two topics. The first is about the nature of our community, how it functions, and what it might be doing differently. My thesis is that the Internet can make a big difference in the way a small global community operates, so we should think about the implications of new technology, not just for computer music but for the computer music community. If the first topic is about how we might do things, the second topic is what we might be doing. I will describe a personal selection of favorite research challenges that I hope to be working on for the next decade, and perhaps I can interest some others in joining me.

Before jumping into these topics, I thought it might be appropriate to talk a bit about "the good old days" of computer music, how I got started, and how things have changed in the last couple of decades.

Computer Music Then and Now

I started studying computer music in the late seventies, mainly by reading everything I could get my hands on. The first ICMC I attended was in 1983, twenty years ago, at the Eastman School of Music. One of the talks was about ultrasonic sensors for conducting, and my immediate impression was that it would make more sense to follow human musicians than conductors. (At least in my experience playing in orchestras, that's what I learned to do!) So I set about creating what I called "computer accompaniment." To test and demonstrate my ideas, I designed and built a small computer that included hardware support for pitch tracking and sound synthesis (see Figure 1). This work was presented at the 1984 ICMC in Paris. In the figure, you can see the luggage handle I attached to the box to make it easier to carry, and the whole system was designed to fit under an airplane seat to avoid damage.

Figure 1. A small computer built by the author to develop his first computer accompaniment system.

This illustrates what research was like then. There was a lot of focus on hardware because computers were just too slow to do many of the things we wanted to do. By 1980, there were a number of synthesis languages running on mainframe and minicomputers, but microprocessors were the new thing, and the area of real-time synthesis and control was full of possibilities. Computers and software were relatively simple, making it possible to build or modify systems without a large investment of time and money. Hardware promised to make things fast enough for real-time synthesis, so a lot of effort went into creating systems and not so much work went into exploring what these systems could actually do.

Hardware versus Software

Most hardware design efforts by researchers were not great successes. Figure 2 illustrates why. Suppose you started building hardware that you expected to be 10 times faster than a software approach. (Theoretically, hardware could give much better than a factor of ten speedup, but in reality, we are matching state-of-the-art processors against home-brew hardware.) Whatever the speedup of the hardware, notice that the software approach is going to get exponentially faster over time due to Moore's law: processors are going to double in power every 18 months. So by the time the hardware is designed, built, debugged, and supported by a suite of software, it is lucky to have a useful life of a year or two before it gets overtaken by software running on the latest, greatest microprocessor.

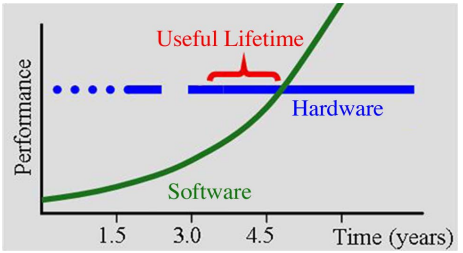

Figure 2. Software always overtakes hardware at an exponential rate.

Tools in the Early Days

Unlike today, when there are so many computer music products on the market, experimenters in the early 80's had to either build their own systems or adopt one of the few systems that worked. As a result, there was a lot of do-it-yourself work, including drivers, interfaces, and synthesizers. People tended to gather around and support systems that worked, because there were relatively few choices. Also, there was a small separation between tool builders and tool users. If you wanted the tools, you pretty much had to be proficient in building or maintaining them. On the other hand, systems were much simpler than typical computer systems today. One could build hardware at the gate level, interface it to a computer, and write instructions that would directly read or write to the hardware. Today, there are many layers of abstraction in both hardware and software, so building or customizing systems is much more difficult.

Tools Today

Of course, a lot of the work we did twenty years ago is no longer necessary. Every personal computer has audio I/O built-in, complete with software. The speed at which computers are progressing has led to rapid software evolution, and there are many different interfaces supporting sound on computers. For example, WinMM, DirectX (versions 1 through 9), and ASIO all provide software support for sound I/O under Windows. Instead of too few things that work, we are cursed with too many choices. Another change is that today's computers are generally less suitable for real-time control. Even though they are much faster, the layers of software and hardware abstraction create a lot of overhead. The 4 MHz microprocessor in Figure 1 handled interrupts faster than modern programs running at clock rates a thousand times faster. Of course, the modern program is protected and scheduled by an operating system, so it may take a thousand times more work to transfer input data to the program, but that's exactly my point. Earlier computers were simpler in many ways.

Commercial vs. Research Systems

One thing that has remained true is that computer systems become obsolete quickly as newer, faster systems replace them. However, since today's systems are so much more complicated, it is especially hard to keep up-to-date. This is not such a problem for commercial systems, where upgrade costs are amortized over many customers, but for researchers and special-purpose systems, system complexity often kills off new developments. For that reason, we are more dependent on commercial systems such as audio interfaces, synthesizers, and even the software that drives them. This puts more separation between the tool builders and the tool users than in the past.

Commercial systems are a mixed blessing. We benefit from commodity products like fast processor chips, CD recorders, digital audio interfaces, and laptop computers. The music industry also builds synthesizers, controllers, and software that are invaluable for research. On the other hand, we suffer from poor design and rapid turnover. Commercial systems make many assumptions that are simply wrong for our admittedly small community of researchers. (Note that by "researchers" I include scientists, engineers, and composers.) Due to these assumptions,

- most off-the-shelf systems do not support interaction and live performance,
- we are often limited by conventional media formats, e.g. DAT recorders and CD's are limited to two channels,
- systems change rapidly on the assumptions that users have not built their own extensions and modifications, so replacement is simple,
- making software obsolete is good for the industry--it reduces maintenance costs and increases sales.

The Computer Music Community

In my opinion, the community of computer music researchers suffers more than necessary. We tend to build tools for personal use or for a very narrow distribution when we have to, and otherwise rely on the mass market to generate products that we can use. Between these two extremes of personal and mass market development, I believe we should spend more effort at the level of the community, pooling our limited resources to everyone's benefit.

One would expect that in an active community like ours, there would be some well-developed resources, including:

- standard portable libraries for audio I/O, MIDI I/O, sound file I/O, unit generators and common DSP functions, and digital audio compression,
- editors for audio, events, and music notation, allowing annotation, display, visualization, and composition,
- collaboration on network-based music performance, including theory, practice, tools, servers, and codecs,
- benchmarks and datasets for analysis/synthesis, DSP performance, pitch estimation, music transcription, and other tasks,
- curriculum design: what are the core concepts to understand, and what

are the great works everyone should hear?

What can we do to achieve these and other goals? I believe that people generally make rational decisions, so it must be that there is simply not enough reward to justify community-oriented effort. However, we need to realize that this is *our* community. Collectively, we establish the norms and in many ways the reward system. We teach those that follow us, we review proposals and papers, and publish a large fraction of our own work through the ICMC proceedings. Our members are on editorial boards of most of the publishers that are important to us. I think we can change things if we want to, and this might be a very rational decision.

The Internet

How can we change things? One strategy that seems obvious is to leverage the Internet as a repository of the shared community. Email and web sites offer 24-by-7 access to information across the globe. Open source software is a good model of community cooperation. Facilities like SourceForge and CVS support global software development teams quite effectively, enabling cooperation that never would have worked in the past.

The Internet has already defined the way many of us conduct our research. I have trouble getting students to visit the library, which means anything not on the Internet is effectively lost to collective thought. Virtual documents are defining our collective knowledge more than physical books and journals. All researchers will change the way they work to take advantage of communication and information available electronically. I would like to see a community like ours set an example for other fields.

Organization

Simply working with the Internet will not guarantee results. I believe some organization is required; good work does not simply appear without high-level planning and design. One could argue that the free software movement is a counterexample, where there is no top-level organization and good systems evolve in a bottom-up process. The problem I see is that, often, free software is actually not well designed and lacks input from the experts who might improve matters. Free software often works best when recreating the functionality of an existing software product, eliminating some of the need for top-down design, but reducing the degree of innovation.

Initiative

To overcome the limitations of free software development paradigms, contributions from experts are essential. Academics have a tendency to write about the flaws in the current practice and to propose improvements. This is different from actually making the improvements and changing the current practice, as this would be "non academic" work. I think when we talk about building a community and supporting the community practice with tools and resources, it is necessary for the experts to be involved, to take the initiative to make things better.

Examples

All this is easy to write about, but much harder to turn into practice. Perhaps these ideas are naive, and certainly changing a culture is not easy. At least it is a useful exercise to create a vision of how things might be. Creating and sharing a vision seems to be necessary for change, if not sufficient. On the positive side, I think many of us already share this vision and many have made progress. There are many good examples of software created with the needs of our community in mind, and I will mention some of them here:

- PortMusic: PortAudio, and PortMidi are APIs implementing cross-platform access to audio and MIDI I/O. Think of these as a "stdio" library for music. (For non-C-programmers, "stdio" allows C programs to read and write files; where would C be without it?)
- Audacity is a cross-platform, free software audio editor that is especially good at handling large files.
- Synthesis systems, including SuperCollider, csound, Nyquist, jMax, JSyn, and STK are perfect examples of well-designed and supported systems created especially for our community.
- Open Sound Control (OSC) offers real-time, cross-platform, network-based communication especially for music applications.
- PlanetCCRMA, while not really a software project, organizes knowledge and software distributions to help the computer music community use a Linux optimized for music.

I am sure this list could be longer, but these examples are sufficient to illustrate my point, that this community is capable of working together to improve our tools and resources. While all of these projects offer usable software now, most of them could use more help to make them complete and reliable.

As a community, one of the toughest problems is to collectively identify the small number of projects we can support with a critical mass of developers and users. Again, the tendency for academics and hackers alike is to make small improvements or to focus on one narrow aspect of a problem, then distribute a half-baked

"research system" with many rough edges. As a result, we often find many programs available, none of which actually work well enough to be worth using. We need to find a balance between innovation and standardization.

Research Challenges

Building tools is great fun and a worthy occupation, but it is only part of the picture. Most researchers are excited by the really big questions that stimulate our curiosity and imagination. I wish I could formulate grand challenges for computer music the way David Hilbert did for mathematics a century ago, but frankly, it would be foolish for most of us to invite any comparison with Hilbert. Instead, I will offer a more personal view. These are my challenges, and I hope you will find them interesting. You may even want to tackle them yourself, and I would welcome anyone to do so, independently or in collaboration.

Machine Identification of Musical Structure

When we listen to most music, we hear relationships and structure. For example, we may recognize that a melodic phrase is repeated. We can think of the two phrases as related by a time difference. A transposition occurs when there is a time difference and a pitch difference. There are many possible relationships within a piece of music. Some are important and intentional while others are random and accidental. Recently, I have been working on getting computers to find structure in music, looking mainly for repetition, and then building simple descriptions of the implied structure.

Figure 3 shows some input and output of this program. The audio is from the John Coltrane Quartet playing Coltrane's composition "Naima" and was taken directly from an audio CD. Below the audio you can see a transcription of the saxophone solo represented in piano roll notation. (The middle part is a piano solo, and the transcription did not recover much.) The transcription is far from perfect, but not bad considering that the source is polyphonic audio. The structural analysis program looks for similarities within the transcription. Just below the transcription, there are colored bars representing the final output. Bars with similar colors represent similar phrases. You can see that the opening phrase is repeated immediately (the first two red bars), and then there is a shorter repeated phrase (the green bars), and so on. "Naima" is a ballad in AABA form. The "A" parts are the red bars, but in this analysis the "B" part is subdivided into three parts (green, green, magenta). You can see from this analysis that Coltrane opens by playing the AABA form and closes with just BA. You can also see a repeated 2-measure phrase at the end.

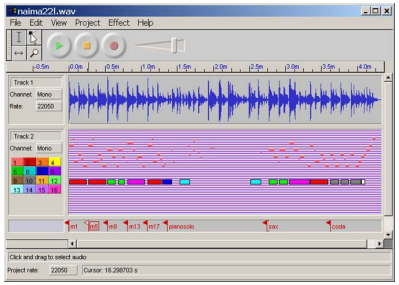

Figure 3. Structural analysis of John Coltrane's "Naima."

(As an aside, Figure 3 is a good example of the kind of research-oriented software tool I advocated in the previous section. In addition to displays of audio, MIDI, and label data, this editor can display time-aligned graphics. I used this feature to create the colored bars appearing below the MIDI data. This facilitates music data visualization and interaction that could not be accomplished with any commercial software).

My advisee, Ning Hu, and I have also looked at spectral features as a way of detecting music similarity, and in general, this works better for polyphonic music. There is much work to be done in this area. How can we detect transposition and other structural relationships? What kinds of musical structure do people actually hear? How do we decide which relationships are significant and which are random? Can we combine information at the symbolic level with features from acoustic representations?

Phrase-Based Synthesis

The second problem I would like to discuss is music synthesis, a topic that has been central to this field from the beginning. It is standard practice in science and engineering to subdivide big problems into smaller problems that are easier to solve. In the synthesis area, this led to a standard model in which music is divided into notes which are synthesized independently and then combined to complete the synthesis process. This divide-and-conquer approach works well in science because most things are independent enough to make progress even when the assumptions are not entirely true. If I drop a ball, the acceleration is about 1G, and even though it depends on such things as my blood pressure, whether I'm wearing gloves, and the air temperature, I do not even know how one could measure all these microscopic effects.

Music is different. The sound of one note depends on the next, so we cannot simply create notes in isolation and expect to combine them to create music. Of course, this independence assumption is fundamental in Music *N* languages and MIDI, and the assumption is truer of some instruments (the piano) than others (the violin). Figure 4 illustrates the difference between traditional note-by-note synthesis and the concept of phrase-based synthesis.

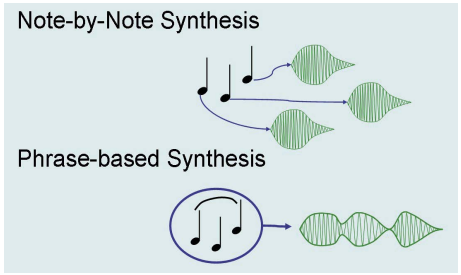

Figure 4. Synthesizing a note-at-a-time fails to capture important contextual information.

To illustrate the effect of phrase and context, Figure 5 shows two amplitude envelopes extracted from a trumpet performance. Both notes have about the same duration, dynamic level, and pitch, but the left one is articulated using the tongue and the right is slurred. The two shapes are nothing alike. You might also notice the quick release at the end of the tongued note. This is characteristic, but only appears when the note is followed by another tongued note. We have always known that the independence assumption is wrong, but we have not made much progress getting rid of it.

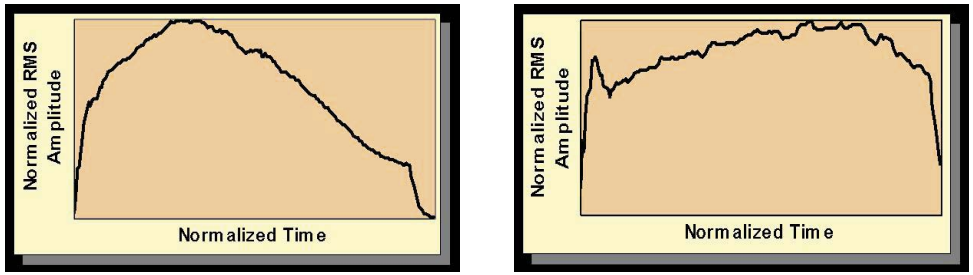

Figure 5. Amplitude envelopes of tongued and slurred

In my work on Combined Spectral Interpolation Synthesis with Istvan Derenyi, we treat trumpet synthesis as a two-step process. First, a score containing phrasing information and other annotations is translated to continuous amplitude and frequency control functions. There are no separate notes in this intermediate representation, just continuous control functions. In the second step, these control functions drive a synthesis process that creates continuously evolving spectra that are appropriate for the given amplitude and frequency controls. It is interesting that we start with distinct notes, move to an intermediate representation where notes do not exist, generate audio, and end up with the *perception* of a sequence of notes.

Our work has only just begun, but I believe at least the concept that notes are not separable can be applied to many synthesis techniques, from spectral models to physical models, and even to samplers and MIDI. In the case of trumpet synthesis, I believe our sound examples are quite convincing. By using the same synthesis algorithms with note-by-note synthesis and with phrase-based synthesis, we can hear a dramatic difference. I hope to extend our work to handle a greater range of articulation, to extend the work to other instruments, and to automate the construction of instrument models using machine learning techniques.

Combining Light and Sound

Musicians have been interested in the combination of light and sound from ancient times. Recently, however, computers have made it possible to synthesize images at video rates using inexpensive, portable equipment. The use of video projections in concerts is becoming almost commonplace (see Figure 6), but there is still much to be learned. I believe there is room for exploration at many levels. At the systems level, how do we organize software and hardware to facilitate the coordination of images and sound? At the music theoretical level, how do we analyze music that includes images, video, and/or animation? How should composers think about images and sound? From the level of psychology, how do images affect our perception of sound (and vice versa)? Of course, composers are not the only ones thinking along these lines, so there is also a need for cross-disciplinary exchange of ideas.

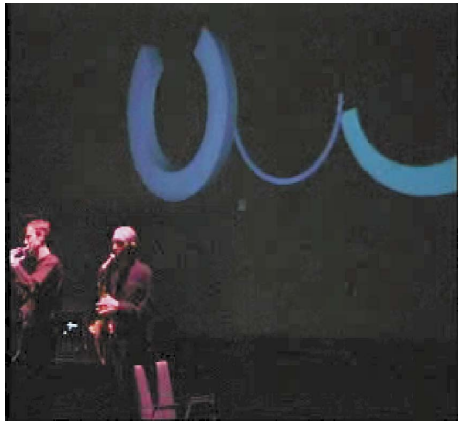

Figure 6. A performance of "Uncertainty Principle" with the author and Eric Kloss, soloists. The image in the background is a projected interactive computer animation that reacts to the soloists in real time.

These questions have no simple answers, and this is a perfect example of why I think of composing as research. Like scientific research, we must begin by studying isolated instances (i.e. composing pieces). As we become familiar with more examples, we develop taxonomies, identify concepts, and form hypotheses. Eventually, our experience is organized into theories such as harmonic theory (in music) or signal processing (in engineering). Some might say that we already have enough music problems to solve, but I think the potential to link images to sounds in live performance is especially important for computer music and is therefore something particularly interesting for our community to explore.

Languages and Systems

To facilitate research in all these areas, we need good tools and good ways to express ourselves. Our community has developed many interesting language concepts (see Figure 7). Music *N* offers some unconventional semantics that are both effective for music and lacking in more conventional programming languages. MAX-like languages have proven to be very effective for visual programming of interactive systems. Still, a number of problems remain in music programming systems.

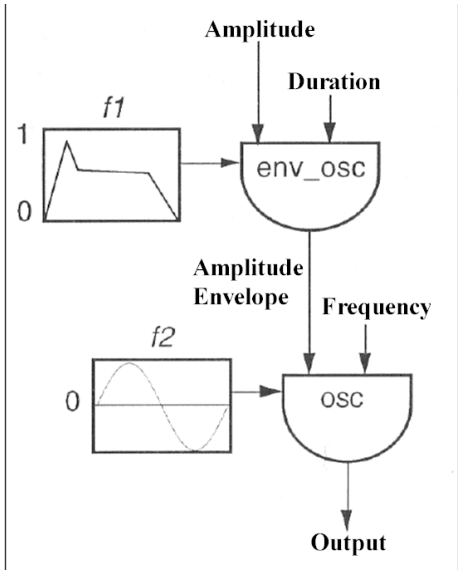

Figure 7. Concepts such as unit generators, patches, instantiation, and scores have led to interesting specialized languages for computer music.

One observation is that all computer music systems seem to have a "sound" of their own. Every system makes certain things easier than others, and this inevitably leads composers, for better or worse, down certain paths and away from others. Most feel that computer music systems should not interfere with the goals of their users, and so language designers must always strike a balance between power and generality. Power comes when you can produce a lot of output or complexity with a minimum of effort. Generality comes when you can achieve exactly the output you want. In contrast, many commercial programs take the opposite approach. They provide strong support for certain styles or techniques of music generation, thereby offering creative opportunities to less-technically inclined musicians.

I believe we are just beginning to explore interactive music systems, especially those that deal with a mixture of audio, control, and sensors. For example, composers are now thinking about signal processing not as an effect but as an integral aspect of composition. While the focus used to be on synthesis, now we see signal analysis as an important component of interactive systems. Music theory, music structures, signal processing, and sound synthesis are beginning to merge in interesting ways. Progress may depend upon systems that simplify the interactive manipulation of signals. Languages and systems can also support new directions, including music performances over networks and the incorporation of images and animation into music. There are many challenges, and we need to experiment with many new approaches to better understand language and system design for music.

Conclusions

Just as the explosion of computer technology is surely a milestone in human civilization, I believe computer music represents a significant turning point in music history. Instruments augmented the human voice with new sounds and techniques, music notation augmented human memory, and now electronics and computers bring us new ways to store, generate, and process music. What could be more interesting than to be in the middle of an artistic and cultural revolution!

I hope I have motivated you to at least think about ways we can work together to make our work more productive and more rewarding. I have also suggested some research topics that I find most interesting, and perhaps some will join me in their exploration. Regardless of what the future brings, these are interesting times, and we should all be thankful that we can play a small part in their unfolding.

References

Note: A paper like this touches on the work of hundreds of authors and papers. I have elected to cite only the specific work mentioned here, and I will ask the reader to consult the references in these papers and web sites for a broader coverage of these topics.

My work on computer accompaniment was first presented at the 1984 ICMC, although the publication date for those proceedings is 1985: Dannenberg, "An On-Line Algorithm for Real-Time Accompaniment," in *Proceedings of the 1984 ICMC*, Computer Music Association, (June 1985), pp. 193-198. Some relatively early discussion of software synthesis appeared in Dannenberg and Mercer, "Real-Time Software Synthesis on Superscalar Architectures," in *Proceedings of the 1992 ICMC*, ICMA, (October 1992), pp. 174-177. Work on music structure appeared in Dannenberg, "Listening to 'Naima': An Automated Structural Analysis of Music from Recorded Audio," in *Proceedings of the 2002 ICMC*, ICMA, (2002), pp. 28-34, and Dannenberg and Hu, "Pattern Discovery Techniques for Music Audio," Journal of New Music Research, 32(2), (June 2003), pp. 153-164. Phrase-based synthesis, which in our papers is called "Combined Spectral Interpolation Synthesis," is described in Derenyi and Dannenberg, "Synthesizing Trumpet Performances," in *Proceedings of the ICMC*, ICMA (1998), pp. 490-496, and Dannenberg and Derenyi, "Combining Instrument and Performance Models for

High-Quality Music Synthesis," Journal of New Music Research, 27(3), (September 1998), pp. 211-238.

I have written about systems aspects of working with animation and music in Dannenberg, "Real Time Control For Interactive Computer Music and Animation," in *Proceedings of The Arts and Technology II: A Symposium*, Connecticut College, (February 1989), pp. 85-94, and Dannenberg and Rubine, "Toward Modular, Portable, Real-Time Software," in *Proceedings of the 1995 ICMC*, ICMA, (September 1995), pp. 65-72, and Fred Collopy has an interesting web site devoted to this topic. In the computer music languages and systems area, see Dannenberg, "The Canon Score Language," *Computer Music Journal*, 13(1), (Spring 1989), pp. 47-56 for an introduction to the idea of scores as programs and how temporal semantics can be added to a programming language to create a music language. This work was extended in Nyquist as described in Dannenberg, "Machine Tongues XIX: Nyquist, a Language for Composition and Sound Synthesis," Computer Music Journal, 21(3), (Fall 1997), pp. 50-60. More recently, I have turned to language support for real-time systems, and some of the most recent ideas are in Dannenberg, "A Language for Interactive Audio Applications," in *Proceedings of the 2002 ICMC,* ICMA, (2002), pp. 509-515.

I gave some examples of what I consider to be good examples of research-oriented software tools. More information can be found in Bencina and Burk, "PortAudio -- an Open Source Cross Platform Audio API," in *Proceedings of the 2001 ICMC*, ICMA, (2001), the PortMusic web site, Mazzoni and Dannenberg, "A Fast Data Structure for Disk-Based Audio Editing," in *Proceedings of the 2001 ICMC*, ICMA, (2001), pp. 107-110, James McCartney, "Rethinking the Computer Music Language: SuperCollider," *Computer Music Journal*, 26(4), (Winter 2002), pp. 61-68, the csounds.com web site, the Nyquist web site, Déchelle, Borghesi, De Cecco, Maggi, Rovan, and Schnell, "jMax: An Environment for Real Time Musical Applications," *Computer Music Journal*, 23(3), (Fall 1999), pp. 50-58, Burk, "JSyn -- A Real-time Synthesis API for JAVA," in *Proceedings of the ICMC,* ICMA (1998), Cook and Scavone, "The Synthesis ToolKit (STK)," in *Proceedings of the 1999 ICMC,* ICMA, (1999) and the STK website, Wright and Freed (1997), "Open Sound Control: A New Protocol for Communicating with Sound Synthesizers," in *Proceedings of the 1997 ICMC,* ICMA, pp. 101-104 and the OSC web site, Lopez-Lezcano, "The Planet CCRMA Software Collection," in *Proceedings of the 2002 ICMC,* ICMA, pp. 138-141 and the PlanetCCRMA website.

Impressions of ICMC 2003

Yuan Peiying

As the final notes of Steve Everett's *Gamelan Asmaranda* lingered in the air, signaling the end of the International Music Conference 2003, I was overwhelmed with a cascade of thoughts. What came to mind immediately was most definitely regret and remorse that there would possibly never be another chance to participate in such a conference in Singapore ever again. Thankfully, however, I had managed to learn many things from the conference and I am glad that I have been given this invaluable learning opportunity.

There were several memorable events that had taken place throughout the five eventful days of the conference. When attending concerts, what amazed me most was the realization that computer music is actually a very 'real' form of music. The sounds, although digitally enhanced and altered, are most realistic and are a reflection of the everyday sounds we come across. There is no shortage of compositions by various composers to prove this point. Rikhardur H. Fridriksson's Lidan II, showcased during the last day's evening concert, is one example. The sounds used had real-life origins, which were the coughs and gasps produced by the human vocal cords. Moreover, the piece was inspired from and a direct consequence of a period of bad health and respiratory disorder of the composer himself, thus highlighting the reality attached to the sounds of computer music.

I sat on the bus the other day and was perturbed by the screeching noises of the brakes. I walked past a construction site this afternoon and for the first time, I wasn't irritated by the blast of sounds. Instead, my mind was imagining how I would be able to use these sounds in my compositions. These are but two examples of how my participation in the conference has broadened my perception of music and triggered off much creativity and imagination.

The range of compositions presented was wide and a definite eye-opener. Besides more 'traditionally' computer music sounding works, like Apostolos Loufopoulos' *Night Pulses*, whose night sounds were quite distinctly simulated through computer mediums, there were slightly more avant-garde sounding works, as well as works that challenged the norm and brought in other influences. Works like Naotoshi Osaka's *Mirrors* for hichiriki (a traditional Japanese wind instrument),