The Laptop Orchestra as Classroom

Ge Wang,[*] Dan Trueman,[†] Scott Smallwood,[†] and Perry R. Cook[*†]
[*]Department of Computer Science
Princeton University
Princeton, New Jersey 08544 USA
{gewang, prc}@cs.princeton.edu
[†]Princeton University
Department of Music
Woolworth Center
Princeton, New Jersey 08544 USA
{dan, skot}@music.princeton.edu

In its inaugural semester (Fall 2005), the Princeton Laptop Orchestra began as a seminar comprising 15 freshmen undergraduates (3 women, 12 men), 15 laptop and six-channel speaker-array stations, and equipment for networking and transportation (see Trueman 2006; Trueman et al. 2007; and Smallwood et al. in this issue of *Computer Music Journal* for details). The authors of this article served as the teaching corps but also participated in all other aspects of the ensemble. Software such as the ChucK programming language (Wang and Cook 2003), Max/MSP (Puckette 1991), the Audicle (Wang and Cook 2004b), miniAudicle (Salazar, Wang, and Cook 2006), sndpeek (Misra, Wang, and Cook 2005), and hardware input devices and sensors comprised our teaching tools and platform. A second PLOrk seminar and ensemble was taught the following semester (Spring 2006) as an upper-level undergraduate elective in both the Department of Music and Department of Computer Science. A third seminar took place in Fall 2006, focusing on composing and programming for laptop orchestra.

All three courses required students to submit a short application. The students were selected on the basis of enthusiasm, thoughtfulness, and balance to the class; no explicit technical or musical background was required. All 15 students in Fall 2005 entered the class with no prior programming experience but with great interest and varying backgrounds in music. The Spring-semester students included 25 undergraduate sophomores, juniors, and seniors, as well as graduate students with a wide range of technical experience and musical training. The Fall 2006 class was a graduate seminar, consisting of Music and Computer Science graduate students and three undergraduate PLOrk "alumni."

The PLOrk classroom takes place in two major formats and locations. There is the weekly class meeting at McAlpin Auditorium (see Figure 1), a rehearsal space shared with the traditional university orchestra, choir, and other Music Department ensembles. Each PLOrk class is scheduled to last 3.5 hours, at the beginning of which the members of the orchestra transport 15 sets of laptops, racks, hemispherical speakers, mats, pillows, and sensors to the classroom, where they are connected, powered, and booted. Additionally, a wireless (and sometimes wired) local area network (LAN) is established, and video projection and sound amplification are connected from the teaching machine. In this classroom mode, activities range from presentation of basic programming concepts, introduction and playing of compositions, individual and group student presentations of assignments, and rehearsal as an ensemble. In addition, students are taught to troubleshoot any hardware or software problems that they may encounter with their station, and they are expected to learn how to set up and "tear down" quickly and efficiently.

The other classroom venue is a smaller studio space (see Figure 2). It houses seven ready-to-use PLOrk stations and a studio machine with a projector for teaching and hosting a professional sound-editing environment. The PLOrk studio can hold an audience of up to 15 people. Weekly voluntary help sessions on topics such as programming in ChucK and Max/MSP take place here. In addition, students gain access to the space 24 hours a day to work on assignments, practice pieces, program, compose, "hang out," ask questions, and get help from instructors and peers. Whereas topics presented in the larger rehearsal space tend to be high-level, this

Computer Music Journal, 32:1, pp. 26–37, Spring 2008

Figure 1. PLOrk class in session. Here, the ensemble is learning about real-time sound visualization using sndpeek.

Figure 2. The PLOrk studio classroom houses multiple PLOrk stations with sensors and provides around-the-clock work, rehearsal, and PLOrk social space.

smaller studio classroom provides the specifics and hands-on examples needed to transfer concept into practice. If the rehearsal-hall classroom setting represents the training ground of PLOrk, the studio classroom serves as the "trenches."

The PLOrk Week at a Glance

A typical week in PLOrk, which begins on Thursdays, might follow the following pattern.

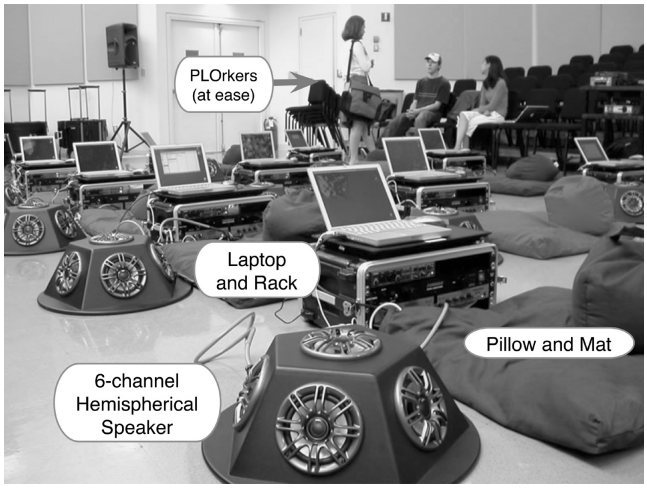
Thursday

In the morning, the instructors come into the studio and load data (programs, patches, sound files, scores) onto the main PLOrk "mothership" computer in preparation for class. All new data, including software updates, new pieces, ChucK and Max/MSP programs/patches, student homework submissions, and other shared files are synchronized across all PLOrk laptops via CVS (Concurrent Versions System; see www.nongnu.org/cvs/) over the PLOrk network. After this step, the laptops should then be identical in content, and ensemble members can log onto any PLOrk laptop (or *PLOrktop*) and expect the same data. Because the total size of new files can range into the gigabytes, this synchronization process can take significant time (sometimes 1–2 hours or more). Thus, it is important to complete the transfer beforehand to maximize class time for teaching and playing.

At 1:00 PM, students begin arriving to transport the stations from the studio space to the rehearsal hall. Setup begins in usual PLOrk formation (1–9, A–D, X–Y; see the companion article in this issue of *Computer Music Journal*). Ideally, the orchestra should be ready to rehearse by 1:30 PM. Students choose a PLOrk station and seat themselves on the mat and pillow (see Figure 3).

There is no formal class schedule. Typically, brief announcements are made, the day's goals stated, and the orchestra dives into its activities, which can take three forms. First, the ensemble may rehearse a new or existing piece of music while working with the composer. Second, technical material may be presented, often on programming with ChucK or Max/MSP, and also on such topics as sensor mapping, audio synthesis, sound design, computer music performance, physical modeling, and signal processing. Third, students may present solo or group works-in-progress for their assignments as well as independent work. This final component also serves as an opportunity for group critique and allows students to showcase their work. New ideas can also be tested here using the full PLOrk configuration, which is unavailable in the studio. Brief breaks and stretching sessions take place at transitions between activities. Homework is usually assigned, ranging from listen-and-response to

Figure 3. The ensemble is set up; students socialize before class.



weekly or bi-weekly projects in Max/MSP and/or ChucK. Around 4:15 PM, the orchestra packs up and transports gear back up to the studios.

Friday–Sunday

Unless there is an upcoming performance, there usually are no planned activities for these three days. Most of the students have heavy course loads in addition to PLOrk, and this is a good time to catch up on their work and sleep and to brainstorm on the assignment before the next week of rehearsals is underway. However, as due dates for PLOrk projects and performances draw near, the studio is often filled on the weekend by individuals and groups working on projects or rehearsing. A sign-up sheet coordinates studio access.

Monday and Tuesday

There is often a help session on one or both of these nights. Usually, it is a workshop session on ChucK or Max/MSP programming that addresses a current assignment. Programming examples are constructed and deconstructed line-by-line (or object-by-object), and participants are encouraged to follow along on their laptops and to ask questions freely. Although these sessions are voluntary, they are usually well attended. They offer hands-on expansions upon the concepts presented in class, transferring them into usable strategies for projects. At other hours, this studio is open for development and rehearsal.

Wednesday

The "official" deadline for PLOrk assignments is on the eve of the next class, or Wednesday night. Because the weekly file synchronization occurs Thursday morning, students are required to sign up for a Wednesday evening time slot to add and commit their work to the central PLOrk repository. One or more instructors are always on hand to assist. This evening is also sometimes used for sectionals to accommodate visiting composers arriving the night before their in-class rehearsals and resident composers working with groups within the ensemble.

This weekly routine provides stability to counter the "bombardment" of new information (which we typically race to prepare every week in response to the preceding week's experiences) and new assignments. Holding class on Thursday requires that assignments are due Wednesday night, which in turn means the students work on them mostly between Sunday night and Wednesday evening. Thursday's class provides punctuation and a sense of weekly closure: everyone showcases their work and plays music together, and they also prepare for upcoming tasks.

Approaches and Tools for Teaching

There have been many efforts at integrating laptops and digital music in general into teaching situations, including the Behaviour Ensemble (artswork.bathspa.ac.uk/labs/music/), the Interactive Electronica Ensemble (www.mnstate.edu/music/electronica/electronica.html), the Lucid Dream Ensemble (Moorefield and Weeter 2004), the Digital Music Ensemble at the University of Michigan (www.music.umich.edu/current_students/perf_opps/dme/), and the Mobile performance Group at Stetson University (www.mattroberts.info/mpg/), among others. In all of these ensembles, students create their own instruments and work together in largely improvised, collaborative contexts. The

design of PLOrk, with its individual, localized, hemispherical speakers, emphasizes the laptop as a musical instrument, and the sheer size of this collection of instruments forces a somewhat different approach. We have relied on individual composers to envisage and realize new pieces for and with the group. In this approach, the composers build the instruments (largely software-based), teach the students how to play them, and rehearse the group as a whole. In the process, the students learn deeply about the compositional, aesthetic, and technical approaches of many different composers. Many of the pieces are described in a companion to this article (Smallwood et al. 2008). As they develop skills, they are then also invited to develop their own pieces for the group or subsets of the group. (Composing for the entire group is often daunting, even for experienced composers.)

The PLOrk approach to teaching is experiential. This is essential, because our PLOrk classroom setting is a distinctly unique one, and so our approach has been to collaborate and engage with the students, providing experiences of learning through discovery. It is also worth noting that the focus of the three semesters greatly differed from each other. The first PLOrk seminar, a freshman "first-year-experience" seminar, embraced breadth that ranged from computer-music synthesis and history to live performance and music creation. It also was our pilot class, and so it represented many unknowns. We felt it was important to give the students the understanding that we were embarking on a unique project together, and that their participation would highly influence the direction of PLOrk's future. The Spring seminar course emphasized rehearsing and performing as an ensemble. This progression is perhaps not surprising, because we had only just begun to explore possibilities in composing for the laptop orchestra during the Fall. But by the middle of the Spring semester, we had over twenty pieces from a dozen composers and were scheduled to perform three concerts in April and May. In the Fall 2006 semester, now armed with an arsenal of completed pieces, we combined teaching compositional and technical issues for laptop orchestra with rehearsals and live performances. Despite these different approaches, the main themes described in the following paragraphs remained throughout the various seminars.

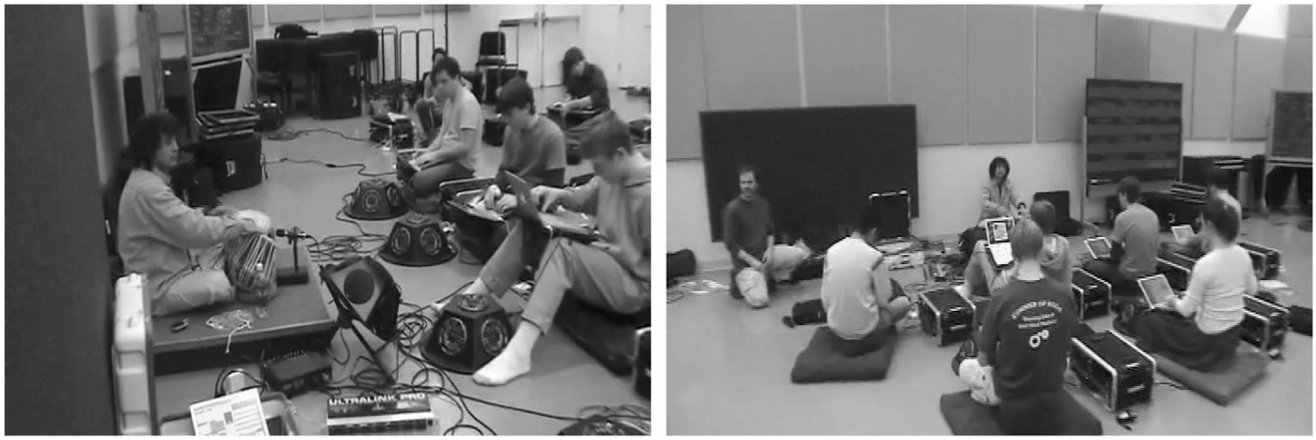
Learning and Playing Pieces

As a new piece is prototyped, the orchestra learns it and provides feedback for the composer (see Figure 4). This involves working with the composer, learning an interface, playing the piece as an ensemble, and providing suggestions for improvement. Almost all PLOrk pieces debut during an initial class session and then undergo modifications and bug fixes. As mentioned previously, the ensemble members are expected to have expert working knowledge of their instrument and be capable of debugging connections between laptop, rack, and speakers; they are also expected to understand the basic operation of ChucK and Max/MSP and to occasionally debug programs and patches. We liken this to the care and treatment of any musical instrument (making reeds, wrapping mallets, oiling valves, caring for bows, etc.). As the players learn their instrument, they become better at collaborating with the composers whose pieces they will ultimately perform by asking intelligent questions and making insightful suggestions.

Live Performance

PLOrk performances come in many different sizes and levels of interaction. Some require direct manipulation of hardware sensors or software interfaces to produce immediate sonic or musical gestures, whereas others call for higher-level influence over generative musical processes. The students learn to play a wide spectrum of interfaces in live performance contexts and encounter a challenging diversity of aesthetic approaches, ranging from the "Deep Listening" techniques of Pauline Oliveros (Oliveros 2005) to the intensely rhythmic works of Zakir Hussain. Furthermore, students are asked to create their own compositions in various configurations, including solo pieces, small group pieces, or works for the full ensemble. In these cases, students must not only create a piece of

Figure 4. PLOrk in-class rehearsal with Zakir Hussain in preparation for PLahara *by Dan Trueman, So Percussion, and Zakir Hussain.*



music but also negotiate issues of live performance and engagement with players and audience. Owing to the overwhelming possibilities offered by computers and by the scale of the PLOrk ensemble, the starting space for crafting such a performance can be somewhat nebulous. This space must be navigated, and decisions about modes and levels of interactivity between human and machine must be considered. In this sense, PLOrk provides a unique laboratory where students learn to craft both musical ideas and performances using computer-mediated technologies.

Programming

PLOrk does not assume any programming or technical background of its members. Some students enter the class with experience in C, Java, or another programming language, whereas others enroll with little or no prior programming experience. Part of the PLOrk approach has been to teach aspects of software development in order to (1) enable students to create sound and music (see Figure 5), (2) provide insight into how a PLOrk piece is assembled, and (3) teach the basic concepts of computer music. We cannot overemphasize the importance of the possibility of making music in motivating a student to learn how to program. The ultimate goal is to arm the student with practical know-how for realizing sonic and musical ideas using a computer, and crafting live performances on top of that.

Although we require that students learn enough programming to complete the assignments, we do not explicitly require that students achieve any particular level of proficiency in programming during the semester. We do, however, require that the end product of sound, music, and performance be interesting, thoughtful, and in accordance to specification. The programming language is to be used solely as a tool, a means to realize the tangible and rewarding goals of music creation, and not as an end in itself.

We teach (or at least expose) several concepts in programming, each in the context of making sound and music. Nearly everything is presented by example, and each topic on a need-to-know (or "fun-to-know") basis. Although it would be improbable (and counterproductive) to fully explore all of these topics in a single semester, we are able to present the fundamentals to the point where students can employ them in creating compositions and performances. These concepts include basic imperative programming (e.g., types, variables, values, operators, and control structures); the importance of time in audio programming, including (non-preemptive) concurrent programming; and functions and procedural abstraction (including event-driven programming, MIDI, Open Sound Control, networking, and object-oriented programming).

Nearly every new concept we introduced was eagerly received by the students, who immediately saw how they might use it to enhance their music programs or to program things they previously could not. As mentioned earlier, the primary lan-

Figure 5. Programming in Studio B. Here, Michael explains the inner workings of his software, which synchronizes multiple hosts for sound synthesis as part of a trio performance created with Brandon and Charlie. The software was implemented in ChucK.

guages we utilize are ChucK and Max/MSP. We have found these two environments to be good partners for teaching programming.

ChucK is a Princeton-developed programming language for real-time audio synthesis, composition, and performance. It is an ongoing, open-source research experiment in designing a computer-music language completely "from the ground-up." A main focus of the design is code readability and flexible control of time and parallelism; readability and clarity trumps performance and conciseness. Its main features include the ChucK operator (=>), the unification of audio-rate, control-rate, and musical timing into a single precise mechanism, a straightforward concurrent programming model based on time, event-driven processing, and provisions for code to be written "on-the-fly" (Wang and Cook 2004a). It is cross-platform and freely available, which allow students to work on their own machines if desired. Furthermore, the recently released Small Musically Expressive Laptop Toolkit (SMELT) provides additional resources in to quickly learn about and prototype new laptop-based physical instruments in ChucK (Fiebrink, Wang, and Cook 2007).

ChucK has proven to be a successful and interesting tool for teaching programming, audio/music synthesis, instrument building, and computer-music performance in PLOrk. Owing to its clear grammar and precise delineation of time and parallelism, the students are able to learn ChucK

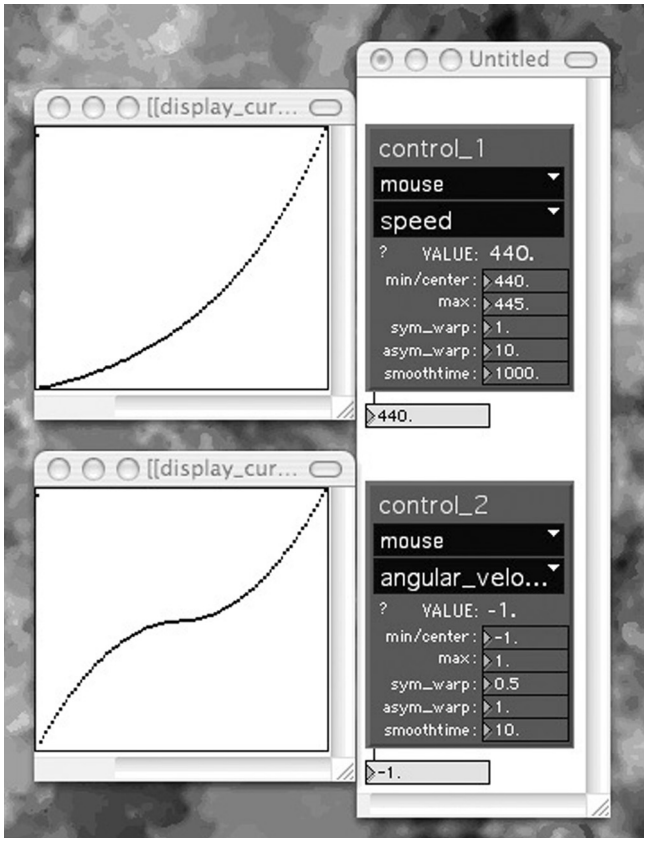


Figure 6. Examples of a PLOrk mapping abstraction with different settings. The user chooses the controller (the trackpad, in this case) and a feature of that instrument (which might be simple, like "x-position" or more abstracted, like angular velocity). "Warping" values of 1 leave the signal linear. Asymmetrical warping creates exponential-like curves, allowing the user to emphasize one extreme or another, while symmetrical warping allows the user to emphasize the middle or the edges. Both kinds of warping can be combined.

programming for sound synthesis and composition quickly, without prior programming experience.

Programming in Max/MSP can be powerfully rewarding but also deceptively difficult. At first, students might get the impression that it is relatively easy compared to text-based coding, but they usually encounter serious difficulties quite quickly. Part of this is due to the large vocabulary in Max. Another well-known challenge centers on order of operation (the "trigger" problem; see Puckette 1991). Finally, the control-rate/audio-rate distinction typically requires some time to understand. Because we do not have time in our curriculum to offer a full year of Max/MSP programming techniques, we rely heavily on prewritten abstractions so students can quickly learn how to create instruments by connecting control data to sound processes. They are introduced to Max/MSP programming, and some continue on to learn more advanced techniques. The abstractions prove to be important pedagogical tools for teaching about real-time controller mapping and PLOrk meta-instrument construction (see Figure 6). For instance, one suite of abstractions allows the user to easily grab a control signal (from any known PLOrk controller, including Teabox-based sensors), scale that input to an appropriate range, "non-linearize" it as needed using both symmetrical and asymmetrical filters, smooth it over time, and then connect it to other Max/MSP objects.

Other abstractions include similar filters for buttons, a wrapper for the Max/MSP `pattrstorage` that attempts to allow users to easily save and recall the state of their patches, and network abstractions.

One of the advantages of these abstractions is that they allow students with only a basic introduction to Max/MSP programming to quickly assemble and experiment with instrument building, focusing their attention on the "feel" of the mappings and features of the synthesis/signal-processing technique with which they are working. Those who are interested can delve further into the programming techniques behind these abstractions, which will reveal not only Max/MSP programming formalisms, but also Javascript and Java beneath the surface.

As an example, we present an early assignment in which we asked students to build a generative drum machine using ChucK, employing the timing and concurrency mechanisms in the language, and to learn to perform it on-the-fly. We were pleased to discover the students (most of whom had no prior programming experience at the time) delivered quality works that demonstrated both technical comprehension and creative zeal. Additional assignments included "PLOrk chamber compositions and performances" and "designing an ambient soundscape," each open to interpretation and yet designed to make use of newly introduced technical concepts.

PLOrk Assignment: Build a Drum Machine and Perform It

As a case study, we reproduce in Table 1 the specifications for the Drum Machine assignment and describe various duo/trio performances created by the students.

Table 1. PLork Drum Machine Assignment

Deliverables:

- create a drum machine using multiple ChucK sub-programs (or shreds)
- practice playing the drum machine using on-the-fly programming commands (+, -, =, —, ^, etc.); perform it in class
- write a short README text file that describes what you did and any interesting problems or challenges you encountered.

What to do:

- experiment with playing `otf_*.ck` and `e*.ck` examples using on-the-fly programming commands
- find or record drum samples (or other percussive sounds)
- each sound should contain only a single strike (and not a drum loop); you may wish to edit them slightly (using Audacity or another sound editor)
- this need not take a long time, but do pay attention to individual and collective feel of the sounds, as they will make a big difference in the final result
- put these files in a folder (your chuck files will refer to these files); if possible, credit the source of the sample in your README.
- make a ChucK sub-program (shred) to control each drum sound (feel free to base on examples shown in class or during help session)
- choose a base tempo that all shreds agree on (e.g., `.5::second => T;`)
- synchronize to this period at top of each sub-program (e.g., `T - (now % T) => now;`)
- in certain cases, it might make sense to control more than one drum sound in a shred (like if two sounds always go together). In general, however, you should split up the drum machine so that you can independently control each component.
- you may add additional parts (drone, melody, bass line, etc) if you like, but that is not required—focus on getting the percussive parts done first.
- practice playing the drum machine you created using on-the-fly programming commands.
- have fun!

Student Works

As students became increasingly proficient in programming, they were able to employ input devices/sensors, processing of live acoustic instruments (including voice), and networking in their compositions and performances. The trio of Ken, Jason, and Matt mapped WACOM graphics tablets, TriggerFinger MIDI drum pads, and pressure-sensing panels to sound-synthesis algorithms in ChucK and Max/MSP (see Figure 7). The trio of Anna, Zach, and Jason created the PLOrk-ified Cello (which processed Anna's cello in real-time using Max/MSP), the "Scrub Voice Synth," various "soundscapes," and an interactive drum machine. William and R.W.'s duo created a musical skit that involved playing an inflatable (or "air") guitar with a sensor-augmented glove, along with processed vocals. Janet, Brian, and Theo used light sensors to trigger sound samples with cups and hand movements. Janet also networked eleven PLOrk stations using ChucK and OSC (Wright, Freed, and Momeni 2003), and she controlled them independently from two MIDI keyboards located at the "command center" to create a distributed performance system. Michael, Brandon, and Charlie made use of networking and created a performance that involved on-the-fly ChucK programming and featured a tablet-based blues solo.

Listening

In addition to programming, weekly assignments sometimes include a listening component that asks students to experience one or more audio/video recordings and to write a brief response to each.

Results and Evaluation

One of our observations in teaching both Max/MSP and ChucK in the context of PLOrk was that, contrary to generally accepted wisdom, the text-based approach of ChucK was more easily accessible than the visual paradigm of Max/MSP; many of the students adopted ChucK as their primary language, and there was general resistance to "digging further" into Max/MSP. This was not universally true—some students did in the end prefer programming in Max/MSP—but the majority of students seemed to find the ChucK language easier to learn and use. We

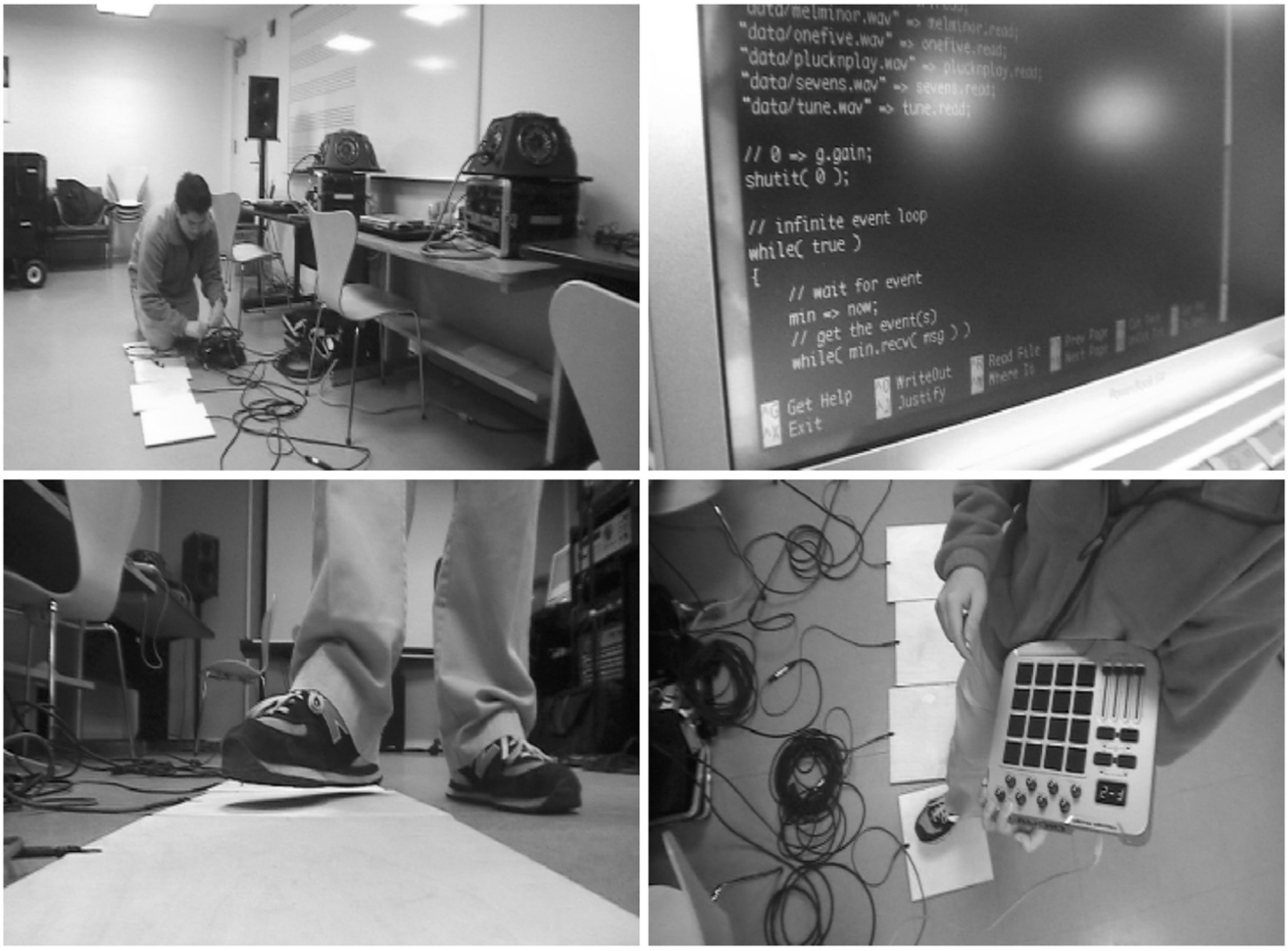


Figure 7. Pressure-sensing panels (bottom left) and MIDI drum pad (bottom right) are mapped using ChucK and Max/MSP as part of a trio performance created by students by Ken, Jason, and Matt in the freshman seminar.

should qualify these observations by noting that we may simply not yet have a compelling approach to teaching Max/MSP in the context of PLOrk, and the excitement around a new language like ChucK, especially one being developed here on campus and taught by its creators, undoubtedly had something to do with its appeal. Also, although the abstractions were definitely effective in allowing students to quickly get "up and running" while getting a taste of Max, they also tended to create a certain amount of confusion as to what exactly constitutes Max/MSP: are the abstractions part of the language, or were they built with Max/MSP? Perhaps what was best accomplished here was an exposure to the graphical programming paradigm and the acquisition of enough familiarity to "poke around" in a Max/MSP patch and make sense of it.

As mentioned earlier, nearly all students came into the Fall 2005 class without programming experience but had, by mid term, internalized ChucK to the point they could comfortably focus on creating compositions and performances. The language was straightforward to learn, and it proved effective in teaching programming concepts as well as sound synthesis. In fact, the two reinforced each other. Perhaps having immediately perceivable audio/musical feedback when programming allowed the student to more easily focus on the task at hand, instead of simply learning programming "for programming's sake."

We also noticed some specific characteristics of ChucK that lend themselves to teaching. For example, the ChucK operator (=>) removes confusion between assignment (e.g., =) and comparison (e.g., ==), which seems to often lead to unnecessary confusion in introductory programming courses that use C++ or Java. In fact, there is no = operator in ChucK, as all assignments are carried out by =>. Additionally, the left-to-right semantic of => promotes a strong sense of order of operations.

Traditionally, concurrent programming is rarely taught in introductory programming courses, often reserved for higher-level software design or operating systems classes, owing to the difficulties commonly associated with programming threads (Birrell 1989). However, we believe it can be beneficial to introduce certain types of concurrency early because it is highly expressive (especially for sound and music) and also reinforces the notion of interactive processes. ChucK's concurrent programming model is well-suited for this purpose because it is non-preemptive and based on time. This allows students to benefit from the expressiveness of concurrent programming (parallel program flow can be developed independently) without the unnecessary complexity associated with preemptive concurrency (e.g., indeterminism, race conditions, and dead-lock). Furthermore, simultaneities in music/sound can be directly modeled, providing the incentive to learn concurrent programming in the first place. Similarly, manipulation of time to generate sound is a useful teaching mechanism for representing the relationship between time and sound, and for clearly delineating synthesis algorithms.

We conclude this section with a snippet of student feedback, quoted from a README document submitted as part of the Drum Machine assignment:

> However, when everything worked the way it was supposed to, when my spontaneous arrangement of computer lingo transformed into a musical composition, it was a truly amazing experience. The ability to control duration and pitch with loops, integers, and frequency notation sent me on a serious power trip. . . .
>
> It was so exciting to figure out how to control the exact rhythm produced by the shred [ChucK process], and I started working out rhythmic patterns on scrap paper in the form of music notation and then transferring it mathematically to the shred composition itself.
>
> I really like the on-the-fly command system as well. It may have driven my roommate crazy, but I was definitely jamming the whole way through. The only real problem with this assignment was knowing when to stop and get on with the rest of my work. This is so much better than memorizing French verbs.
>
> —Anna, Fall 2005

Toward a Naturally Integrated Classroom

One of the most exciting promises being fulfilled by PLOrk as a teaching resource is the idea of moving away from strictly studio-oriented computer music courses, instead focusing on live performance. For example, traditionally in computer music courses we focus on techniques and software for constructing music in the studio and playing these back in concert. With the PLOrk classroom, we are able to truly integrate live performance into such courses, and we can emphasize crafting performances by providing meta-instrument platforms on which students can test out their ideas (see Figure 8). Each PLOrk meta-instrument, with its localized hemispherical speaker, has an instrument-like sonic presence that invites students to experiment with sound both individually and with others while retaining their own sense of identity (i.e., they are not being "swallowed" by a traditional sound reinforcement system).

Programming in the service of music-making is a powerful motivator for learning and experimentation. It emphasizes immediate sonic feedback, which can greatly encourage inquiry about how to make and change the sound. ChucK syntax and semantics enable students to learn quickly without sacrificing or compromising key concepts. Max/MSP and our abstractions provide useful tools for learning about and creating real-time controller mappings for new PLOrk meta-instruments. But PLOrk

Figure 8. Anna demonstrates two meta-instruments she has created. On the right is the PLOrk-ified cello (audio processed in Max/MSP; parameters controlled via MIDI foot pedals), and on the left is a "soundscape generator" (implemented in Max/MSP and ChucK, controlled via MIDI drum pad). This encapsulates a wide range of topics from musical performance, programming, controller mapping, and sound synthesis.

performers learn more than programming, interfacing, and studio techniques; they also learn how to be part of a group, and they have to practice, individually and in small groups. They learn to be better musicians.

Our hope for the PLOrk classroom and other similar learning environments is to create a truly integrated, interdisciplinary experience, where the mixing of ideas from different fields happens naturally in the service of a common goal, such as music-making in PLOrk. Our experiences these past two years have been encouraging, and we plan to further solidify PLOrk's place in our curriculum and to investigate new pedagogical practices and paradigms in the laptop-orchestra classroom.

References

Birrell, A. D. 1989. "An Introduction to Programming with Threads." *Technical Report SRC-035,* Digital Equipment Corporation.

Fiebrink, R., G. Wang, and P. Cook. 2007. "Don't Forget the Laptop: Using Native Input Capabilities for Expressive Musical Control." *Proceedings of the 2007 New Interfaces for Musical Expression Conference.* New York: Association for Computing Machinery, pp. 164–167.

Misra, A., G. Wang, and P. R. Cook. 2005. "SndTools: Real-time Audio DSP and 3D Visualization" *Proceedings of the 2005 International Computer Music Conference.* San Francisco, California: International Computer Music Association, pp. 278–281.

Moorefield, V., and J. Weeter. 2004. "The Lucid Dream Ensemble: A Laboratory of Discovery in the Age of Convergence." *Organised Sound* 9(3):271–281.
Oliveros, P. 2005. *Deep Listening: A Composer's Sound Practice.* New York: iUniverse Publications.
Puckette, M. 1991. "Combining Event and Signal Processing in the Max Graphical Programming Environment." *Computer Music Journal* 15(3):41–49.
Salazar, S., G. Wang, and P. R. Cook. 2006. "miniAudicle and ChucK Shell: New Interfaces for ChucK Development and Performance." *Proceedings of the 2006 International Computer Music Conference.* New Orleans, Louisiana: International Computer Music Association, pp. 63–66.
Smallwood, S., D. Trueman, P. Cook, and G. Wang. 2008. "Composing for Laptop Orchestra." *Computer Music Journal* 32(1):9–25.
Trueman, D. 2007. "Why a Laptop Orchestra?" *Organised Sound* 12(2):171–179.
Trueman, D., et al. 2006. "PLOrk: The Princeton Laptop Orchestra, Year 1." *Proceedings of the 2006 International Computer Music Conference.* New Orleans, Louisiana: International Computer Music Association, pp. 443–450.
Wang, G., and P. Cook. 2003. "ChucK: A Concurrent, On-the-Fly, Audio Programming Language." *Proceedings of the 2003 International Computer Music Conference.* San Francisco, California: International Computer Music Association, pp. 219–226.
Wang, G., and P. R. Cook. 2004a. "On-the-Fly Programming: Using Code as an Expressive Musical Instrument." *Proceedings of the 2004 International Conference on New Interfaces for Musical Expression.* New York: Association for Computing Machinery, pp. 153–160.
Wang, G., and P. Cook. 2004b. "The Audicle: A Context-Sensitive, On-the-Fly Audio Programming Environ/Mentality." *Proceedings of the 2004 International Computer Music Conference.* San Francisco, California: International Computer Music Association, pp. 256–263.
Wright, M., A. Freed, and A. Momeni. 2003. "OpenSound Control: State of the Art 2003." *Proceedings of the 2004 International Conference on New Interfaces for Musical Expression.* New York: Association for Computing Machinery, pp. 153–159.