Linking Computer Science, Art, and Practice Through Digital Sound

Jennifer Burg
Wake Forest University
Winston-Salem, NC 27106
burg@wfu.edu

Jason Romney
University of North Carolina School of the Arts
Winston-Salem, NC
romneyj@uncsa.edu

ABSTRACT
This paper reports on an NSF-grant supported summer workshop that brought music and computer science students together for eight weeks to explore creative projects in digital sound production. The dynamics of the students' collaborations were observed as they crafted experimental projects weaving together music, theatre production, sampled digital audio, and MIDI. Moving among various levels of abstraction, the students found practical and artistic motivations to learn the science of digital sound. The projects they produced suggest ways to revitalize computer science courses by linking science, art, and practice through digital sound, a subject naturally interesting to students.

Categories and Subject Descriptors
J.5. [**Computer Applications**]: Arts and Humanities

General Terms
Experimentation

Keywords
digital audio, MIDI, curriculum development, interdisciplinarity, collaboration

1. INTRODUCTION
As we computer scientists look for ways to draw students back into our programs, we sometimes miss the most obvious hooks. How many students do we see walking across campus with buds sprouting from their ears and music players on their belts? Have we forgotten the language that spoke to us most stirringly when we were 18 and 19 years old, the language of music? Sound captures the attention of young students. Digital sound and music are all around us, part of our everyday lives. Digital sound is founded on concepts central to computer science. So why is it that nowhere within the computer science curriculum can we find a place for digital sound? No ACM category used in the classification of SIGCSE papers relates to the study of digital sound. Courses in digital audio are rare in the computer science curriculum. Yet there is a great deal of computer science that can be learned by means of sound digitization and processing.

SIGCSE'09, March 3–7, 2009, Chattanooga, Tennessee, USA.

Interdisciplinarity has come into favor in academia. The current trend is to break the barriers between departments and to explore conceptual relationships and synergies between disciplines. Incorporating digital sound into the computer science curriculum fits within this collaborative spirit. Digital sound production is part of music, theatre, television, radio, and video production. It has its technical and mathematical roots in digital signal processing, traditionally within the realm of engineering. Thus it brings together science, art, and practice – a combination that satisfies the students' need to relate their learning to something "real" that matters to them.

Our work focuses on finding ways to make science, art, and practice meet in the computer science curriculum via topics in digital sound. One of us is a computer science professor, the other a digital sound designer who teaches at a performing arts conservatory. Our NSF grant involves mixing music/theatre students with computer science students to see how their interactions naturally evolve. How does the perspective of one type of student affect the work of the other? Where are the synergies? What types of projects emerge, and how can the concepts learned in these projects be mapped to topics in traditional computer science teaching? How much of the science of digital sound must students know in order to improve their art and practice? What elements of the science leap out to students and demand to be understood as they try to create the work that inspires them? These were the questions we posed at the outset of an eight-week digital sound production workshop held at Wake Forest University in the summer of 2008.

For eight weeks – five days a week, eight hours a day – three computer science and three music students worked together in our digital sound lab. Some worked on Windows computers; others worked on Macs. Sampled digital audio and MIDI were combined using software like Audition, Audacity, Reason, Cakewalk Music Creator, Logic Pro, Pro Tools, and Max/MSP. The students were charged with finding collaborative partners and doing experimental projects that combined music or theatre production with digital audio seen from a computer science perspective. They also were permitted to work on personal projects of their own interest. The students' music, writeups, and videos of their talks at a workshop can be accessed at http:/www.cs.wfu.edu/~burg/CCLI/DigitalSoundProductionWorkshopSummer08/DigitalSoundWorkshopProceedings.html.

In this paper, we describe the students' outcomes from the workshop, our observations of the their work, and the conclusions we draw from these. Because the workshop was set up quite differently from a standard course – in that the students had freedom to choose their own projects and partners and had a great deal of time to focus on these projects – the setup of the workshop

is not directly transferrable to the computer science classroom. However, the types of projects completed in the workshop suggest assignments and topics related to digital sound that could be woven into existing computer science courses. Thus, this paper may be of interest to anyone looking for ways to invigorate their courses or introduce new courses that attract students while retaining the rigor and relevance of computer science.

Very little background literature exists on this subject, for the very reason that digital sound is not often taught as computer science. We are aware of another recent project that combine the science and music of sound has come out of Duke University, a collaboration of their Engineering Visualization Technology Group and Music Department. Their results are viewable online. However, resources on pedagogy in this area are scanty.

2. INTERDISCIPLINARY COLLABORATIVE PROJECTS

2.1 The Assignment

We wanted the students to do more than record and edit songs in the sound production lab. We urged them to try to be experimental and strike out into areas that would require that they learn something new about music or computer science. At least one project had to be collaborative with a person from the opposite discipline. Within the first week the students had chosen projects and partners. They divided into two collaborative groups – one with two computer science students and one music student, and the other with the reverse combination.

2.2 The Music of Nature

The first collaborative project to emerge was "Nature Fusion." One of the music students was interested in recording and cataloguing sounds of nature ranging from wind and birds to human coughing and breathing. She began recording, downloading, and cataloguing these sounds. As the ideas became more focused, the music student began to work with a CD of bird calls. She listened to each one, found its pitch zone, and mapped it to a zone in a MIDI sample bank. The sample bank could then be used as an instrument for an original composition created by the other music student in the group.

In the meantime, the computer science student in the collaboration set about creating an experimental environment in Max/MSP where note sequences and chords within any given key could be randomly generated. The program could use whatever sample banks were created by the music student. The idea was to generate novel chords and sequences as a way to inspire elements in the music composition that might not immediately come to mind through the composer's usual creative process.

When the sample bank had been completed by the first music student, it was tested on a jazz song. A MIDI version of this song was found online, imported into Cakewalk Music Creator rewired to Reason, and played using the bird call sample bank – an amusing and not unpleasing rendition.

The sample bank was finally used in an original composition by the second music student, a techno-rock piece the students called "Jungle Banga," which we all considered an excellent composition that captured the spirit of the project.

Figure 1. Turning Bird Sounds into Patches in Music Creator

2.3 Screamin' Demon Music Creator

The second collaborative project to emerge was the Screamin' Demon Music Creator. In this interactive computer environment envisioned by the students, players could stand in front of a camera, view themselves on the computer screen, and point to icons of instruments that were arrayed around their heads (shown on the computer screen) to select what they wanted to play. When their gestures caused their videoed hands to move across an instrument icon on the screen, loops of that instrument would start to play.

To implement this music playground, the students chose to use Max/MSP and Jitter, a visual programming environment for MIDI, digital audio, and digital video. The two computer science students did the programming while the music student produced music loops for various instruments. Through Jitter, successive video frames could be captured and checked for activity in the vicinity of the icons on the screen. Max allowed the music loops to be played continuously, with the volumes of only the chosen instruments set above 0. Since loops are automatically synchronized, the instruments were synchronized as they were turned on and off. After they had the basic program running, the students added more features. They rewired Max to Reason, a MIDI sampler and synthesizer package that provides a wide range of both realistic instruments and creatively manipulable sounds. They also added buttons for changing tempo or key.

The students began this project on the first week with great enthusiasm and, impressively, had the foundation of it implemented by the end of the first week. The system could

respond to gestures and start loops running. By the end of the second week, the program had grown to be rather unmanageable, its patched-together objects filling up a full 20 inch Mac monitor. (Patching objects together is how you program in Max/MSP.) By the end of the third week, with Max rewired to Reason and the extra features added in, the program began to bog down and not react fast or predictably enough to be fun to play with. The students believed they could optimize and refine the program if they spent more time, but they were really ready to go on to the next thing for the second half of the workshop.

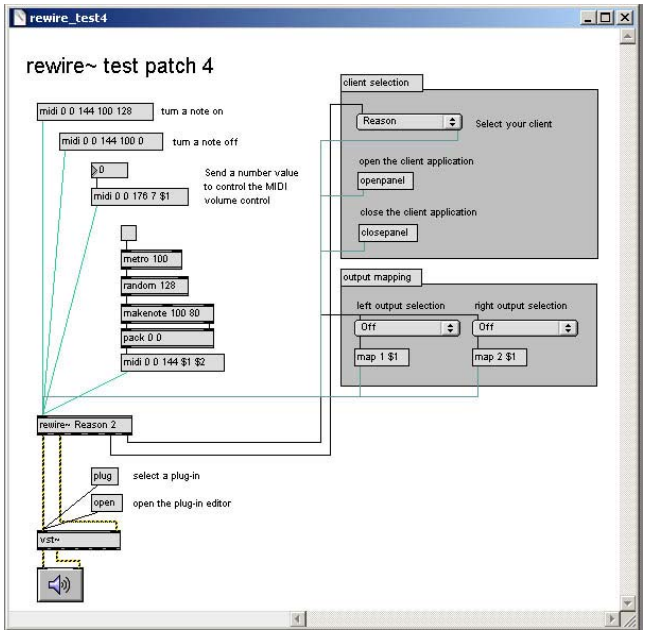

Figure 2. Rewiring to Reason

We realized in retrospect that a weakness in the project was a lack of integration of the music and computer science elements. The two elements didn't seem to feed each other. The music student worked separately creating music loops, and then handed these over to the computer science students. The computer science students paid little attention to the musicality of their production, and in the end their "game" lacked that attraction for the players. There was nothing particularly engaging about the music that was produced. It was simply turned on and off. This is not to say that the project was a failure. The students were inspired to learned a great deal about Max/MSP, Jitter, digital audio, MIDI, and loops in a very short period of time. However, it seemed that the work reached a point of diminishing returns, and we agreed that the students should strike out in new avenues that interested them.

2.4 Creating Echoes for *Floyd Collins*

A third collaborative project was completed with the help of a computer science student who was not officially part of the sound workshop, but who worked part-time along with this group (supported under a separate part of the NSF grant). While the other students had considerable creative freedom to choose their projects, this student was given a problem to solve. This was a theater sound design problem from the musical *Floyd Collins*. In one scene of this musical, the main character is in a cave singing along with his own echoes, each echo being delayed by one musical measure. For theatrical authenticity, it's desirable that the echoes to be created live, reflecting the singer's voice exactly as he is singing for that performance. This requires audio delays, which can be accomplished either by hardware or software. The student in the workshop was challenged to try a software solution using Max/MSP, exploring the extent to which the tempo of the music could be set by the conductor and altered by small amounts, as is natural in real-time human-set tempos.

It was interesting to note that this project – though not chosen by the student and not ostensibly a collaborative project – turned out to be quite successful in interactions and outcome. We decided to try the student's Max/MSP implementation in the theater, with one student singing the echo song, another playing piano accompaniment (a music student), and a third working as assistant sound engineer (a computer science student). (The student who sang was a computer science student who happens to have a double major in theater). The Max/MSP implementation worked even in its first rough cut, and from the experiment in the theater, we identified additional issues to address. In particular, the students were able to observe the significance of latency and delay caused by processing, which is important in synchronizing the voices of the singer and the echoes.

2.5 Observations on the Interdisciplinary Synergies

It was clear in the workshop that the presence of students from one discipline catalyzed the work of students of the other. Although we sometimes wished for better integration of the disciplines within the collaborations, we came to recognize that collaboration takes different forms. Just having the students in the same room was of value, as some of the collaboration involved their simply helping each other when they got stuck, overhearing things that the others were doing that gave them ideas, learning something serendipitously that fed their work, and getting inspired by each other's enthusiasm. The fact that the learning had meaning and results was the main thing.

We also observed that different types of projects have different benefits. On the one hand, allowing students to choose their own projects ensured that they were interested in what they were doing. The students were willing to spend hours figuring out how to do something that mattered to their creations. (This was particularly true of the personal projects described below.) This approach, however, can also let students fall into the trap of aiming for more than can realistically be accomplished in the amount of time they have. Surprisingly, "assigned" projects worked out quite well. (We had another successful one not described in this paper.) Assigned projects resemble real-world ones in that the expectations and time limits are set by the "employer." We found that the students rose to the challenge, particularly because the projects we assigned had clear applications in the theater production.

A final observation is that the dichotomy between "music" and "computer science" students is largely a false one. It turned out the one computer science student had a double major in theater and could sing and play the keyboard. A second one could play the guitar and sing and had clear musical sensitivity. (The third computer science student had no music background and was delighted to play the role of stereotypical computer geek.) Two of the three music students were technically adept with their computers, and the third found his stride easily by the end of the

second week. The main difference in the group was the computer scientists' ability to program and understand the mathematics in more depth, and the music students' knowledge of music theory and experience in performance.

3. INDIVIDUAL PROJECTS

3.1 Covers and Original Music Compositions

Two students created "covers" of songs composed by other musicians. This gave them an opportunity to learn multitrack recording and editing using digital audio and MIDI. They recorded their own voices, added MIDI instruments, and edited and mastered the final songs.

Two of the students – one music student and one computer science student – did arrangements of songs they composed themselves. These were done in multitrack editing software using both digital audio and MIDI tracks. Their songs can be heard at our website.

Five of the six group members also participated in a live recording session where a song written by one of the computer science students was recorded. The composer played keyboard and sang, a music student played guitar, two students sang backup, and a computer science student assisted as sound engineer.

3.2 Sound at a Low Level of Abstraction

One of the students – a computer science major – found himself gravitating to a lower level of abstraction before too long in the workshop. He was interested in doing things himself, that is, by means of his own programs – reading a playing *.wav* files, capturing MIDI messages and converting them to digital audio, and creating his own vocoder and autotuner. This student was a rising sophomore with only one year of programming experience and no music background. The programs he produced proved to be among the most useful outcomes of the workshop.

3.3 Observations on Individual Projects

We've found that there are a lot of students interested in learning how to produce music. For such students, doing "that first song" seems to be something that have to get out of their system pretty early on, and it's an excellent place for them to start since it motivates them to learn the fundamentals. To pull the computer science out of digital sound production, however, students need to go beyond (or perhaps beneath) this first experience to more experimental work or work at a lower level of abstraction. For this reason, we were happy to have the more traditional computer science student in the mix. As he unselfconsciously chattered to his vocoder (as he was programming) and talked it over with us while he was debugging it, the music students learned a little of what was going on underneath the software and could envision ways of making the software work more to their own specifications. In the end, even the individual projects became collaborative in some respects and the students sought help and inspiration from each other.

4. THE COMPUTER SCIENCE IN SOUND

As a foundation for working with digital sound, students need to understand the digitization process – sampling and quantization, along with the implications of sampling rate and bit depth. Sounds modeled as sine waves and the relationships between frequency and pitch and between amplitude and volume are also basic, as are frequency components and dynamic range. For a complete foundation in digital sound, students need to understand the difference between sampled digital audio and MIDI, and the difference between MIDI samplers and synthesizers.

We taught short classes to the students on the topics above at the outset of the workshop. We also set them up with Cakewalk Music Creator on the PCs and Logic Pro on the Mac, and showed them how to rewire their MIDI sequencers to Reason on their respective platforms. After that, we threw them into the water and told them to learn how to swim while we sat back and watched.

We were especially interested in finding the places where the students wanted and needed to know the science to accomplish their goals. We can cite a number of interesting examples.

More than one student was baffled at first about the concept of rewiring a MIDI sequencer (e.g., Logic) to a MIDI sampler/synthesizer (e.g., Reason). The idea is that the sequencer sends the MIDI messages to the sampler, the sampler interprets the messages and turns them into digital audio sounding like some chosen instrument, and the sound is routed back to the sequencer possibly for further processing before it goes on to the sound card. To demystify what at first seemed like an exercise in clicking here and choosing a menu selection there, the students had to conceptualize the signal flow, understanding the changing nature of the data (from MIDI to digital audio) as it moved from one place to another.

Early in summer, we became aware limitations of some of audio equipment. In particular, students experienced the frustrations of playing their MIDI keyboards and hearing the sound some milliseconds later. To fix this problem, the students had to learn something about MIDI lag, the implications of buffer sizes, and the comparative characteristics of sound drivers like ASIO vs. MME vs. CoreAudio.

Quite a few activities required that the students understand frequency components: microphone receptivity, speaker characteristics, equalization of digital audio, musical harmonics, timbre of instruments, and modulation of the human voices. Dynamics were equally important, allowing for an adjustment of the difference between the loudest and softest parts of a song. The students wanted their personal projects – music productions – to sound *really* good, and this wasn't possible without well-informed editing. In the end, they were especially enamored of tools that allowed them to master their music by dividing it into frequency bands and applying dynamics processing band-by-band – a process that combined two basic concepts they had learned.

The students became interested in quantization in two contexts – first in choosing the bit depth for a recording, and then in choosing the resolution of MIDI notes. They were able to see the effect of the first type of quantization on the dynamic range of sampled digital audio. The second type of quantization is the process of "snapping" notes to certain time units in MIDI, resulting in a more precise tempo. Applications such as this clarify a central computer science concept, where data types have only so much potential precision dependent upon how many different discrete values are possible within a range. The resulting rounding error manifests itself in a variety of contexts.

Students came to a more mathematical understanding of the word "modulation" as they investigated the possibilities of vocoders and autotuners. It was one of the music students who brought these devices to the attention of the others, since he had always wanted to use them. We lectured to the group on the basic design of a vocoder and encouraged them to take it from there. A music student investigated the vocoder in Logic Pro while a computer science student tried to implement his own vocoder and autotuner. In the process, the computer science student came to understand harmonic frequencies in the human voice. The lights really came on when he asked his music partner to sing a middle C into the microphone. The computer scientist recorded the note, did a Fourier transform on the sound file, and viewed the frequency components. Spikes showed up at integer multiples of the fundamental frequency, but the student was surprised to find that the fundamental frequency wasn't the biggest spike. He had another "ah-ha!" moment when he tried to snap the fundamental frequency component in his partner's voice to precisely 262 Hz, the frequency of middle C, moving all the other frequency components by the same amount (implementing, he thought, the activity of an autotuner). The resulting tone was dissonant, the so-called harmonics no longer in harmony with the fundamental. This experiment showed, in action, the non-linear nature of human hearing. (Notes separated by an octave sounds essentially like the same note to the human ear, but you actually double the frequency each time you move up an octave – a non-linear progression.)

5. INTEGRATING DIGITAL SOUND CONCEPTS INTO THE COMPUTER SCIENCE CURRICULUM

The most direct mapping of exercises to existing computer science courses grew out of the projects of the computer science student who chose to work at a lower level of abstraction. As he learned about the difference between digital audio and MIDI, the various formats of audio files, and the behavior of vocoders and autotuners, he wanted to create these things himself. In doing so, he had to teach himself quite a few other elements of computer programming: (1) File i/o, including random access files (2) Discovering the existence of and linking to specialized libraries (e.g., for sound) (3) Headers on certain file types (e.g., *.wav*), reading, writing, or parsing them (4) Uses of hexadecimal in programs (5) Directly addresses certain devices in a program, like /dev/midi or /dev/dsp in order to read or write MIDI and audio data (6) Complex numbers and the implementation of the Fourier transform (7) Fairly complex uses of arrays (also used in the Screamin' Demon program) (8) Dynamic memory allocation of arrays (9) Other miscellany like command-line arguments, storage sizes for primitive data types, etc. The programming exercises created by these students can be made available to the reader. These exercises can be fined-turned so that they in the range of first and second year programming and would be appropriate in CS1 and CS2 courses.

6. CONCLUSIONS FROM THE SOAP BOX

These examples show how students can be motivated to learn mathematical and scientific concepts when they work with digital audio. The reader might argue that these are not necessarily computer science concepts and they don't really belong in the computer science curriculum. But why not? We all know at this point that we need to rethink how and what we've been teaching, as the world has marched merrily on into the digital age and pretty much left computer science education behind. Students want to know how computers and things-digital relate to them and the world they live in. Anything that is digital and processed by a computer is fair game for computer science. This is not to say that we abandon the mathematical and scientific rigor of our discipline, but instead that we find a core for the discipline that goes beyond computer programming. The digitization process – sampling and quantization and all the implications thereof – is a concept just as fundamental to computer science as loops and variables. Essential mathematics, algorithms, and technology can be taught by means of applications that give life and meaning to concepts. We're missing good opportunities to interest students in ways that combine relevance with rigor.

7. FUTURE WORK

In our future work, we will continue to sort out the concepts and assignments that we've extracted from this experience. The programming assignments discussed above can be mapped to topics as they are ordered in existing CS1 and CS2 courses. We will to make these assignments publicly available, indicating on them what programming, mathematical, and digital sound concepts they cover so that others can make use of them. Additionally, we add to a repertoire of MATLAB exercises that we have already developed for digital sound (which can be made available to the reader.) We also plan to offer a redesign of computer science courses, from the ground up, incorporating concepts of digital media throughout (digital audio, video, and multimedia programming), in an effort to help revitalize the curriculum. This is part of two NSF grant projects that include multiple faculty workshops each year.

8. ACKNOWLEDGMENTS

We would like to acknowledge the work of our students: Dan Applegate, Tyson Badders, Joshua Bennett, John Brock, Daniel Habib, Shanee Karriem, and Nate Vogt. This work was supported by a National Science Foundation CCLI grant DUE-0717743.

9. REFERENCES

Below are some references that we have found helpful in the development of our own courses:

[1] Burg, Jennifer. The Science of Digital Media. Prentice-Hall, 2008.

[2] Ifeachor, Emmanuel C., and Barrie W. Jervis. Digital Signal Processing: A Practical Approach. Addison-Wesley, 1993.

[3] Loy, Gareth. Musimathics: The Mathematical Foundations of Music. Vols. I and II. Cambridge, MA: The MIT Press, 2006.

[4] Smith, Julius O., III. Mathematics of the Discrete Fourier Transform (DFT) with Audio Applications. 2nd ed. Seattle: Book Surge Publishing, 2007.

[5] Tranter, Jeff. Linux Multimedia Guide. Cambridge, MA: O'Reilly, 1996.

[6] Winkler, Todd. Composing Interactive Music: Techniques and Ideas Using MAX. Cambridge, MA: The MIT Press, 1998.