INTRODUCING COMPUTER MUSIC IN SECONDARY SCHOOL

Christopher Dobrian
University of California, Irvine
dobrian@uci.edu

RÉSUMÉ

Actuellement, l'éducation en l'informatique musicale a lieu presque exclusivement dans les universités, surtout au niveau du troisième cycle. Cet article propose que telle éducation puisse, et doive, s'introduire dans l'école secondaire. La formation en musique et en informatique peut être *combinée* de manière à ce que les deux sujets soient accessibles et engageants, même pour les élèves qui n'ont pas eu de formation traditionnelle en musique ou qui ne se croient pas doués en technologie. En rendant ces matières disponibles à un nouveau groupe d'élèves plus jeunes, on espère éviter que certains étudiants ne soient involontairement exclus de ces domaines en raison du manque d'accès à une formation précoce. Cela à son tour pourrait contribuer à accroître la diversité dans ces domaines au niveau postsecondaire. Pour que cette formation ait lieu, des ressources d'apprentissage de niveau secondaire qui combinent la musique et la programmation informatique doivent être développées, et des enseignants connaissant les deux sujets doivent être disponibles pour ces étudiants.

Computer music education currently occurs almost exclusively at the post-secondary level, mostly in graduate degree programs. This article proposes, however, that such education can and should begin in secondary school. Introductory training in music and computer programming can be *combined* in such a way as to make both subjects approachable and engaging, even for students who lack traditional music training or who might otherwise think of themselves as not technologically inclined. Making these topics accessible to a new constituency of students at a younger age may help to avoid students' being unintentionally excluded from participation in these fields due to lack of access to early training. That in turn may help to increase diversity in these fields at the post-secondary level. For such training to take place, secondary-level learning resources that combine music and computer programming must be developed, and teachers with knowledge of both topics must be available for those students.

1. INTRODUCTION

Computer music study is commonly initiated at the university level, usually in graduate school. However, it can and should be introduced at an earlier stage, in high school and even in middle school. I will explain the reasons why, and I will propose an approach that provides knowledge and experience in both music making and computer coding to students at a young age, preparing them for better footing in those topics when they pursue postsecondary studies.

Most secondary schools provide some opportunity for musical experience (orchestra, concert/marching band, jazz band, choir), and less frequently some sort of musical training (instrumental lessons, AP theory), but almost never any discussion of computer music. By the same token, many secondary schools provide some optional opportunities to learn computer programming, but rarely is such training explicitly linked with sound and music. At the university level, some undergraduate programs do exist that focus specifically on computer music, but because most music curricula are quite full already with classical music training—instrumental lessons, chamber ensembles, large ensembles, ear training, music theory, music history, analysis, literature surveys, and so on—most music students, even those majoring in composition, receive little or no computer music training as undergraduates. Thus, most computer music education occurs during one's graduate studies.

Because music and computer programming have historically been considered separate domains, they are still taught separately at nearly every level prior to graduate school. I will explain why combining those fields at an early stage of education may be preferable.

2. HISTORY

Forty years ago, "computer music" was still largely experimental, and was taught at only a very few universities, almost exclusively in graduate-level courses that included education in digital sampling theory, the mathematics of digital signal processing and synthesis, and computer programming of digital audio. Those courses included no training in music per se; basic knowledge of music theory, notation, and composition was assumed, because most such courses were hosted in the university's music department. The musicians most interested in computer music tended to be composers who had already had some experience with analog electronic music, or less frequently electrical engineers with an interest in digital audio. Thus, computer music and composition (two largely male-dominated areas) became linked, and computer music often was treated as a

sideline subtopic within the study of composition. Alternatively, music departments might hire a digital audio researcher as a somewhat oddball lone member to represent this burgeoning field. MIDI and digital synthesizers were very new, as were personal computers, and were not particularly prevalent, so the hardware for such courses was usually a timeshared mainframe computer, and the computation was mostly non-realtime. In short, computer music equipment and knowledge was accessible only to advanced students at a research university, often in a research lab that was tangential to any academic department. Thus, the content of computer music education, and the locus of computer music activity, was established by the inherently elite nature of the field.

The definition of computer music changed somewhat in the 1990s with the greater prevalence of personal computers and speedier processors. Computers are now the basis of all music recording and production, so the term computer music includes the use of DAW software, software synthesizers, and user-friendly applications running on reasonably affordable laptop computers supplemented with affordable input control interfaces (e.g. MIDI controllers) and audio output interfaces. Thus, computer music activity has been to some extent "democratized", and extended beyond academia to popular musicians and hobbyists. The term computer music now means different things to different people, and can have different foci in different institutions and environments. Likewise, whereas the genre of academic computer music began, and remains, closely tied to mid-twentieth century modernism, outside of academia computer music includes minimalist looping, ambient music, hip-hop, electronic dance music, and other popular styles. I personally welcome this artistic and stylistic diversity, and I hope to see that diversity increase within academia.

3. DIVERSITY

I have written previously[5] about the value of diversity in institutions, especially in academia, in the interest of promoting intellectual "hybrid vigor". The expression of diverse viewpoints leads to healthy debate of new ideas, whereas having all like-minded people together in a room can lead to ossification of a single view, stunting growth. If only for that reason alone, I propose that diversity in computer music is inherently productive and necessary. For me, an equally strong reason for promoting diversity is the pursuit of social justice. In the United States, in many fields, implicit biases and systemic prejudices historically have privileged some people, and have tended to discourage or exclude others, due to their race and/or gender.[1] Rectifying that wrong is of equal importance to the value gained by having diverse scholarship and aesthetics in the field. For both of those reasons—intellectual hybrid vigor and social justice—it is the responsibility of the current practitioners in the field to consciously promote greater diversity in its membership. One way of doing that is to encourage participation by as many different sorts of people as possible, namely by providing opportunity and education to all at an early age.

3.1. The pipeline problem

Youths who belong to a lower economic stratum often do not have the same access to musical instruments, music lessons, a personal computer, and after-school educational activities as their more affluent counterparts enjoy. Also, from an early age, girls—and often Black and Latino boys—are subtly and not-so-subtly steered away from mathematical, scientific, and technical topics, and also away from building and engineering (tinkering) pursuits. This early bias, especially in conjunction with lack of access to learning resources, leads to what has been termed the "pipeline problem":[2] the pipeline through which potential women and minority applicants may get to graduate programs is constricted at many stages along the way, allowing only a trickle to reach the destination. This constriction of the pathway to computer music, the filtering out of certain types of people from the field, actually starts almost from birth, continues at all levels of education, and culminates in the job market. The pipeline problem is only one of the many factors that can contribute to lack of diversity in a graduate program, but it is a notable one, and is one that can potentially be addressed by providing students better access to computer music education at an earlier age.

But what can a university professor do to address this problem? University faculty are invariably extremely busy with the demands of their job: composing and/or conducting research, running computer music facilities, mentoring student creative and research projects, staying abreast of new developments in a rapidly changing field, and teaching. Even if one has the will to try to ameliorate the pipeline problem, the prospect of having to take on new work is discouraging. Reaching younger students requires extra outreach effort. My own solution has been to frame educational outreach and development as a specific research focus.

4. COMMUNITY-BASED RESEARCH

Professors in a research university are often accused of operating in an ivory tower, undertaking research and creative activity that appears to have little effect upon, or relevance to, the surrounding community. In the

[1] I recognize that some people deny that this has been the case, but they are demonstrably wrong. I will not expend the time and space here to demonstrate that.

[2] This reference to the so-called pipeline problem is not meant to ignore the equally serious "trapdoor problem" whereby systemic racism and sexism in this field causes many people to drop out or be excluded from (or avoid) the field altogether. Both problems exist and must be addressed.

University of California system, faculty research that explicitly addresses community concerns is increasingly validated and encouraged.

4.1. Outreach

One way that professors can address the pipeline problem is by leveraging the infrastructure and staff support of their own institution to offer educational outreach programs. In my school, with the help of a staff member from the Outreach office, we have instituted two-week summer academies in Digital Audio Production and in Computer Music Programming. In an effort to promote inclusivity, we offer need-based scholarships—effectively simply reducing or waiving tuition for students with demonstrated financial hardship—so that students from underprivileged areas are encouraged to participate. In addition to teaching those summer courses, I have offered weekend workshops for young student groups (from thoughtfully selected neighborhoods) so that they may be introduced to the ways in which computing and music making can be combined. Thus, the existing facilities and equipment of the university are used during times when they're otherwise unused, and the organization is facilitated by outreach staff who are happy to assist with new activities that fit perfectly within their job description and add to the university's community interaction. Once the summer and weekend courses were well established, I was able to delegate the teaching of those courses to capable graduate students, providing them with valuable teaching experience and much-needed summer employment.

4.2. Education research

Short-term outreach programs such as those are helpful for providing young students with exposure to computer music and the educational culture of the university. Their actual educational value is limited, though, because of their brevity. In order for more thorough education and deeper involvement to occur, the teaching really needs to be integrated into the students' activities on a regular and ongoing basis. This can be in the form of an after-school class (or club)—hosted either at the university or at their own school—or in some cases can even be integrated into their school's existing curriculum in math, computing, and music. Establishing such activities requires coordination with high school faculty and administrators, a task which again can be much facilitated by the university's outreach staff. In my own department at UC Irvine (a Hispanic-serving institution), a small team of three faculty members collaborate with teachers at a local high school in a low-income primarily Hispanic neighborhood, to organize additional activities in orchestral music, jazz, and music technology. In that instance, the work is *pro bono*, but again, once established, it can often be delegated to university students, who enjoy and benefit from having the teaching experience.

In most university computer music programs, pedagogy is not a primary research focus, but there's no reason it cannot be. I recently have chosen to steer my own research toward secondary-level pedagogy, with an aim toward developing and instituting a curriculum of study that will provide combined training in both music and computer programming.

As noted above, these two topics are traditionally treated completely separately until graduate school. In that way, most musicians come to programming rather late, and are often intimidated by the learning curve in a brand-new field of such complexity. Similarly, computer science and electrical engineering students casually go through their entire university curriculum of programming and/or digital signal processing without ever thinking of the potential musical applications of what they've learned. If students were to gain familiarity with both fields—music and programming—at an earlier age, combined in ways that intentionally show the potential interrelationship, those students will have a better chance of reaching a higher level of sophistication eventually.

What should be taught in a curriculum that strives to provide the basics of both music and programming, and what tools and resources should be employed? Because the field of computer music is so broad, and the hardware and software tools are so varied, and musical stylistic possibilities are so numerous, there may be many answers to that question. Even though most students today have access to computers, they are likely using different platforms—macOS, Windows, Linux, Chrome—and they may have limited access to musical instruments, keyboards, other equipment. Access is often limited by financial constraints. Thus, in my own pedagogy I am focusing on a few computer music software tools that are free, cross-platform, and easy to acquire.

5. AVAILABLE FREE TOOLS

5.1. Programming languages

• Pd (PureData)[9] is a free object-based music-oriented graphical programming language developed by Miller Puckette, creator of the ubiquitous (but not free) Max programming language. It is often cited as a free Max equivalent. It is not as fully capable as Max, but it is powerful and has a significant user base and support community.

• SuperCollider[8] is a free text-based music-oriented programming language developed by James McCartney with a significant user base and support community. As a text-based object-oriented coding language, it may be useful in teaching concepts of object-oriented programming, as is JavaScript.

• Sonic Pi[2] is a free, cross-platform high-level text programming interface designed for realtime live coding of music. It was designed mindful of a simple learning curve, for use in education.[1]

• JavaScript[6] is the predominant programming language for Web development, is free, can be composed in any text-editor, and can run in any browser. It provides access to the Web Audio API for handling a variety of audio operations and the Web MIDI API for handling MIDI data; these libraries—along with the graphical capabilities of HTML 5 (and freely available interface-building libraries such as React)—provide sufficient tools for making a browser-based music app.

5.2. DAWs

• Audacity[7] is a highly capable cross-platform sound editor, but lacks many music-specific capabilities commonly associated with DAWs, and lacks full VST support. It serves well for viewing sound characteristics in both the time domain and the frequency domain, and for teaching audio editing, mixing, and the basics of audio effects processing.

• Waveform[10] is a free cross-platform DAW with VST support and a pretty full feature set.

• Reaper[3] is essentially as powerful as a professional commercial DAW. Although not technically free, it is low-cost shareware, and the demo version continues to run with full capability (with only a brief imposed pause when loading) even after the trial period has expired.[3]

5.3. Synthesis

• VCV Rack[11] provides a well-conceived digital emulation of a modular analog synthesizer, retaining traditional standards such as 1V per octave for pitch, while also providing MIDI control and other benefits of digital emulation. Synthesis concepts that may be difficult to grasp in the abstract, such as low-frequency modulation, can be introduced both sonically and visually in VCV Rack before teaching them in a less visual context like Pd or a text-based context like SuperCollider or JavaScript.

6. CURRICULUM

The design of a curriculum, and a set of useful tutorials, that coherently teach both music and computer programming at an introductory level, is the subject of my new and ongoing research. There are many possible approaches to this challenge of teaching the fundamentals of music and programming simultaneously. What works best will depend on the orientation and preferences of the teacher(s), and even more importantly on the students and the context in which the teaching will occur.

In collaboration with a few of my graduate students, whose employment was funded by a grant from the University of California, I have published an online textbook of sorts, a set of lessons titled simply *Computer Music Programming*.[4] It was created for a specific course that I teach, titled Computer Audio and Music Programming. That course is not actually intended to teach programming. It is intended for people who already know how to program in some language, and its main focus is to teach the principles of music, audio, synthesis, and processing that a programmer needs to know in order to program audio and music effectively.

The website demonstrates a modular organizational scheme consisting of many free-standing lessons, organized by topic. The approach in designing that site was that every topic has multiple lessons, and that each lesson is thought of as a "bite"—a small, concise chunk of information. Each bite should be manageably small, and if a bite has other requisite knowledge beyond its own scope, that knowledge should become its own bite, a separate lesson. The goal was to arrive at a collection of lessons—standalone-but-sometimes-interdependent educational chunks—any one of which could be useful on its own. A teacher or autodidact can then assemble those chunks in whatever order they deem appropriate for their purpose. This idea of treating information in standalone bites intended for assembly in various configurations is useful model for conceptualizing a curriculum.

Fortunately, there also already exist a great many teaching materials and videos freely available. One can take heart in the fact that much work has already been done, in the form of online textbooks, tutorials, courses, YouTube videos, etc. So, one doesn't need to reinvent the wheel when it comes to explaining coding concepts and music concepts to beginners. There are many resources for the fundamentals of both music and coding. Where new ideas need to be developed is primarily in the meeting and merging of the two. The teacher's task becomes one of compiling and curating existing materials, and then devising clever ways to link music theory and practice to math and coding. Here are just a few examples of "music meets coding".

6.1. Music meets coding

• Counting is a crucial part of music composition and performance. Music is full of instructions like "play this four times, then play that two times," or "play this chord for four measures, then play that chord for two measures," or "play seven notes in C major as fast as you can," or "the third time this happens, proceed to the next section." Counting is such a well-learned procedure, used dozens of times every day in all sorts of situations, that we generally don't think about how we do it. The process of describing exactly how we count, with the intention of teaching it to a computer, gives the student experience in formalizing a task such that a computer can carry it out. Writing a program that counts requires that the student learn about storing a number in a variable, incrementing that number, repeated processes (loops), relational operators (tests), and conditional statements.

[3] I advocate that everyone pay for the shareware they use. The generous policy of Cockos to leave the Reaper software fully enabled even after the trial period expires does give users the ability to pay at whatever time they're able.

• In most musical situation, timing is crucial. Whereas our counting program would normally run as fast as possible in a few nanoseconds, for it to be musically useful we need to introduce the idea of timed counting, which involves calculating time intervals and scheduling events with whatever scheduling mechanisms exist in the programming environment we're using. That might be **metro**, **delay**, **pipe**, **line**, etc. in Max or Pd, or it might be `setTimeout()`, `setInterval()`, `setValueAtTime()`, `linearRampToValueAtTime()`, etc. in JavaScript. All of this can be made more fun and engaging if the results are expressed sonically as well as numerically, which is now possible because we have introduced the capability for timing and rhythm.

• For music theorists and composers, particularly when discussing post-tonal music, it's common to refer to pitch classes as integers, with C=0, C#(Db)=1, D=2, etc. And for computer musicians, it's common to refer to pitches of the 12-tone equal-tempered scale with their MIDI equivalent, with middle C=60 and each semitone ±1 from that. But why not introduce the pitch-number equivalencies immediately when discussing the fundamentals of music? That makes the pitch relationships more comprehensible to musicians and novices alike, and provides the most obvious way to manage pitches and pitch classes in a computer program. Combined with our timed counting program, we can now figure out how to play an ascending and/or descending chromatic scale, then how to alter that program to perform a whole tone scale (increment by twos) or arpeggiate a diminished seventh chord (increment by threes). Diatonic scales have an irregular pattern of intervals, usually best implemented by looking up pitches or pitch classes in a lookup table, thus introducing the concept of an array, indexing, and modulo arithmetic.

• We all have the apparently innate ability to measure the amount of time between two events, and then project that same distance mentally into the future to predict the next beat, and we can even divide that period of time into rational fractional parts with remarkable accuracy. Even though we perceive rhythm and tempo sonically, viscerally, and intuitively, conceptually rhythm is inherently linked to math, because it is conceptualized as divisions of time (as well as groupings/multiplications). In American English, the names of rhythmic values are all expressly based upon divisions of a whole unit: whole note, half note, quarter note, eighth note, etc. Rhythm and rhythmic notation in Western music have traditionally been taught through a combination of imitation and memorization of rhythms and their notated symbolic representations; however, conceptualizing rhythm as timepoints on a temporal grid, as in a drum machine or a DAW, allows one to treat rhythm mathematically and thus be better able to program rhythms in a computer.

The development of lessons that consciously integrate both music and coding simultaneously is usually best done by thinking of a very small musical concept one wants to convey, and then thinking how you would teach that to a computer step by step. From that program you will see the sort of computer programming concepts and structures that will be necessary, and you begin to develop a set of programming techniques and their musical relatives. Conversely, one can also start with a programming concept and think of a musical use for it. For example, to demonstrate and employ the terminology of object orientation, such as *class*, *instantiation*, *properties*, and *methods*, one might choose to describe a musical note as a sound *object*. Because it is an object (in the programming sense of the word), it will have properties, such as pitch, duration, starting time, instrument, etc., and it may have methods such as play(), stop(), transpose(), and so on. Multiple instantiations of the note object class can be used to comprise a musical melody or phrase (or, for that matter, an entire score), which may in turn be stored as an array of note objects.

7. CMERG

To focus on this project of developing and implementing a course of study that integrates music and programming in new and effective ways, I have established at my own university a Computer Music Education Research Group (CMERG). My intention is to leverage the resources of my own institution—web hosting, physical facilities, equipment, staff, graduate students, and colleagues—and to collaborate with local secondary school educators and their high school students, trying out new ideas for education.

The goal of the group is to provide new educational opportunities, especially to students who might otherwise not receive exposure to one or both of these topics in their high school. In addition to conveying information, our goal will be to make these topics fun, engaging, and welcoming, so that all students will feel encouraged to pursue whatever interests them. Some students may develop a particular interest in programming because they are shown fun ways to use it to make music; others may develop an interest in music because they've been shown a new way to think about it, relevant to their existing interest in coding.

Realizing this goal requires compiling and curating existing educational resources, developing new lessons that integrate music and coding, collaborating with high school educators to organize classes or clubs, and then actually delivering the education. By using free, cross-platform tools and very inexpensive equipment, we can make these activities available even to students of limited financial means.

Because the effort is inherently collaborative, our first step is organizing the interested parties—university faculty and students, outreach staff, high school administrators and teachers, and eventually high school students—and setting up meetings and shared online spaces for brainstorming and planning. We welcome as many group participants as are interested in this topic, wherever they may be. Because of the ease with which it's currently possible to meet virtually and share information online, a research group like this can easily

have global scope. Thus, we welcome collaborators from other U.S. universities, and we invite our international colleagues, too. You are welcome to contact me at the email address listed in the heading of this article.

8. REFERENCES

1. Aaron, S. « Live Coding Education », *The MagPi*, Educator's Edition, v. 1, Raspberry Pi Press, Cambridge, 2016. < https://sonic-pi.net/files/articles/Live-Coding-Education.pdf >
2. Aaron, S. *Sonic Pi* < https://sonic-pi.net/ >
3. Cockos. *Reaper*. < https://www.reaper.fm/ >
4. Dobrian, C. et al. *Computer Music Programming*, online textbook developed at and hosted by the University of California, Irvine, 2019. < https://dobrian.github.io/cmp/ >
5. Dobrian, C. and Jones, M. « Intentional Inclusion: Promoting Diversity in Graduate Study of Music Technology », *Journal SEAMUS*, 2016 v. 1 no. 1-2, Society for Electro-Acoustic Music in the United States, Beverly Hills, 2016. < https://music.arts.uci.edu/dobrian/IntentionalInclusion.pdf >
6. ECMA International. *JavaScript*. < https://developer.mozilla.org/en-US/docs/Web/JavaScript >
7. Mazzoni, D. and Dannenberg, R. *Audacity*. < https://www.audacityteam.org/ >
8. McCartney, J. *SuperCollider*. < https://supercollider.github.io/ >
9. Puckette, M. *Pure Data*. < https://puredata.info/ >
10. Tracktion. *Waveform*. < https://www.tracktion.com/products/waveform-free >
11. VCV. *VCV Rack*. < https://vcvrack.com/Rack >