TEACHING COMPUTER SCIENCE TO DIGITAL ARTISTS THROUGH MUSIC AND SOUND

Ajay Kapur
California Institute of the Arts
Music Technology

Perry Cook
Princeton University
Computer Science and Music

Michael Bryant
California Institute of the Arts
Critical Studies

ABSTRACT

This paper describes the development of an introductory curriculum in computer science modeled on a traditional Applied Introduction to Programming and Algorithms course sequence, but designed specifically for artists as a means of furthering their creative work. Computer science theory is presented in lecture/demos, with weekly assignments that consist of making 30-second compositions incorporating the skills gathered from class. With this project, our goal is to improve the quality, breadth, and effectiveness of technology and computer learning for an entire undergraduate and graduate art school student body. A broader objective of the project is to develop an experimental, trans-disciplinary model for teaching computer science curriculum that can be replicated at other arts institutes, and extended to students in similar non-traditional computer science contexts.

1. INTRODUCTION

As artists of all disciplines increasingly use technology in their creative practice, it is essential that arts institutions provide foundational courses in STEM (Science Technology Engineering and Mathematics) disciplines so that students may conceive of and have the ability to generate new ideas, new artistic approaches, and new technologies. For contemporary artists, adequate knowledge of technological trends and hands-on experience with technology can be crucial for career success. This paper describes a curriculum that addresses this need and offers students more than basic computing literacy—that is, they learn algorithmic techniques, programming, and problem-solving in a student-friendly manner and within a context that inspires engagement, interactivity, and creativity.

The intellectual merit and broader impact of the project lies in the innovative approach to introducing students with little or no computing background to programming and computational thinking. While there has been significant work in developing CS curriculum for non-majors or novices, there have been fewer courses in the area of sound and music. Additionally, the proposed course features ChucK [1] as a primary teaching tool. Chuck is an open source programming language for real-time audio synthesis, composition and performance developed at Princeton by Ge Wang and Perry Cook. This project represents the first formal ChucK-based curriculum developed for undergraduate art students, and has potential to be applicable to and replicable within an array of contexts for teaching introductory computer science to undergraduates and even graduate students. Additionally, it could be appropriate for high-school students in certain contexts.

In creating this curriculum, our team seeks both to enrich the ability of our students to create technology-driven art and to develop new and engaging instructional approaches to the incorporation of STEM learning into arts education.

Heavily inspired by the curriculum designs of PLorK[2] and SlorK[3] the goal of this project is to bring ChucK beyond the laptop orchestra and into a classroom for all digital artists who can use the strengths of the language to make art, and learn key computer science concepts. In this paper, Section 2 discusses related work on computer science education through the arts. Section 3 discusses syllabus and learning outcomes for our new course. Section 4 presents evaluation of the first implementation of our course in Fall 2012 at California Institute of the Arts. Section 5 presents discussions and future work.

2. RELATED WORK

There have been several successful curriculum designs at the intersection of computing and the arts for undergraduate students. The University of Massachusetts' *CPATH CP Performamatics* is an interdisciplinary project that developed a series of CS courses. bringing together faculty and students from computer science, arts, and humanities departments to build connections and community between computing and the arts at the school.

Bryn Mawr College, in partnership with Southern Methodist University, developed a new visual Portfolio based CS1 course based on the programming environment Processing, with the goal of creating an inspiring and engaging CS course for novices and non-traditional programmers such as artists.

Both of these projects are related to and serve as inspiration for our project, with a primary distinction: the aforementioned projects use the Processing language to teach visual graphics integrated with computer science education, while our project is primarily based in audio and music. We endeavor to focus on sound and music teaching ChucK; student learning and assignments are sonically oriented.

There have been many projects and offerings at almost all universities possessing sizable CS/EE departments, where the faculty have worked to create courses that endeavor to teach computer science concepts to non-majors. These often use Java [4], so called "toy" languages [5], or specialized languages such as logo [6], RAPTOR [7], and others. Some have even proposed teaching a scripting language such as PYTHON as the first computing language [8].

By far the most systematic efforts to teach CS to non-majors using media as a focus are the Contextualized Computing Education projects and studies started at Georgia Tech, and since transferred to many other institutions [9]. The curriculum is based on using media to acquire and retain the interest of students in CS courses (Media-centered CS curriculum). These studies showed positive results, attracting students to CS in a large technical university, and then it was shown how that curriculum ports to a small 2-year college.

Other studies by Dorn and Guzdial [10], also looked at specific populations, such as graphics designers who program. Conducting a series of interviews and assessment activities, the researchers found that these subjects want more computer science, but don't find courses (and most other resources) adequate. It was also shown that the designers used cases (case library with code, concepts, AND context) more than a simple library/repository of available code, and that the cases actually "colored" the way the graphics designers wrote their own programs.

The studies of Dorn and Guzdial are the closest to our proposed curriculum, in that they take a specific dedicated population and expose them to concepts, algorithms, and programming via their disciplines and practices of interest. Our curriculum aims exactly at that, teaching DSP and computer science in the strong context of arts projects, via best practices code examples that actually do highly useful things, AND in real time as the students and instructors code (thanks to ChucK).

Our target student audience, digital artists, need the tools and technology to do their projects, in their work lives, and in further education if they pursue it. They want to be able to make better art. We feel they also need knowledge of the underlying science, algorithms, and techniques. The course we designed also gives them this via tools they can afford (open source), modify (open source), and actually use (designed for real-time digital art).

Other courses that involve media programming are taught at various music and digital arts programs and schools. Languages used here range from Max/MSP (audio/DSP) and Jitter (graphics) by Cycling 74[1], to standard graphics software packages by Adobe[2] and AutoCad[3], and animation/modeling software such as MAYA[4]. These are generally geared toward teaching artists and about the production tools they might use in their future professions, but there is little notion of teaching any real foundations of computer science, math, physics (acoustics), and/or engineering.

There are also other projects that combine music and programming, in order to create breadth in music students by teaching them some engineering, and breadth in engineers by teaching them some about music. For example, the "Music, Signals, and Systems" project [11] at Rowan University is a good example of a general education course with no pre-requisites that combines music/ DSP and programming in order to get and keep the interest of students, while teaching them something about engineering and music. The emphasis of that course is on making a laptop orchestra from non-majors. This course is quite similar to Princeton's cross-listed CS/Music course created by Ken Steiglitz and Paul Lansky [12] in the early 1990s. The theme there was to teach music to engineers while teaching engineering to musicians. Another related recent project is that of [13] (U. Mass. Lowell), which proposes a course that uses programming assignments that are music-related; composition, web pages with music, etc. to attract and retain students.

3. LEARNING OUTCOMES AND SYLLABUS

Our courses covers material typical to an introductory computer science course, but our additional goal is to teach computational thinking and programming skills to students new to programming and to extend these skills to all interested undergraduates. Thus, an important component of our project has been to develop new ways to meet this objective, enabling undergraduate art students to engage with the course material and to successfully apply their learning to their creative work.

We developed a sequential course within the context of multimedia arts and with a focus on audio applications, covering the basics of programming. In the first semester we introduce students to ChucK, allowing them to master programming basics in a manner relevant to their artistic fields. Assignments revolve around 30-second compositions written in ChucK where each student demonstrates their understanding of the programming skills they are learning (e.g. loops, functions, arrays, classes, multithreading). This leads to an end of semester concert where students present 2-minute compositions, projecting their computer code in the concert hall. **Figure 1** describes the syllabus of the class in detail.

[1] http://cycling74.com/ (Available Feb 2013)
[2] http://www.adobe.com/ (Available Feb 2013)
[3] http://usa.autodesk.com/autocad/ (Available Feb 2013)
[4] http://usa.autodesk.com/maya/ (Available Feb 2013)

Week	Syllabus
1	Basics: Sound, Waves, and ChucK Programming (if/else, while, for)
2	Libraries and Arrays
3	WaveTable Synthesis & Sound File Manipulation
4	Functions
5	Unit Generators
6	Mid Term Test
7	Multi-Threading and Concurrency
8	Objects and Classes
9	Polling vs. Events
10	Final Project Development
11	Final Project Development
12	Final Concert
13	Final Test

Figure 1 – Intro. to Programming for Digital Artists, Syllabus

4. EVALUATION

This section describes evaluation of our course using data captured from student work and responses from Fall 2012 at California Institute of the Arts. There were 26 students who took the class. There were 4 main ways in which we evaluated the course: (1) Student surveys (voluntary and anonymous) at the beginning and end of the course, (2) breaking down each weekly assignment into grades for each learning goal, (3) a Midterm and Final Exam, (4) a final project combining all learning goals from the course.

Figure 2 shows data from the surveys taken at the beginning and end of the course. The pre-course survey revealed that most students had never taken a formal computer science course, although some had studied programming on their own. **Figure 2** reveals that on average students had a causal familiarity with the concepts, mostly on the level of vocabulary. Few had ever tried to implement the concepts into an actual program. The students who had programmed before, claimed to know topics such as variables, if/else, loops, arrays, functions, random numbers, objects and classes. However, they did not seem to know the advanced topics like recursion, overloading, and multi-threading. At the end of the course, concepts that students reported having only a limited familiarity with were now seen as practically mastered. Similarly, for the more advanced concepts, students reported a positive shift from largely unknown to a level beyond simply basic understanding.

Though the course is offered at the undergraduate level, we did allow graduate students who were interested in the subject matter to take the class. In **Figure 3** we see the total scores for all the assignments, midterm, final exam and final projects. We can see that the 85% of the time, the graduate students performed stronger than the undergraduate students. This is to be expected and has the potential positive effect of the undergraduate students getting influenced by work of their older peers.

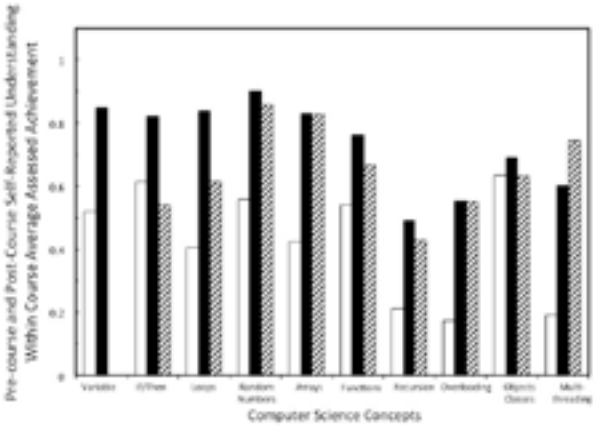

Figure 2 - Results of the pre-course (N = 26) survey compared to post-course (N = 22) surveys. Solid bars show pre-course (white) and post-course (black) self-reported understanding of various computer science concepts. To make the pre-course survey comparable to the post-course survey (the average of a numerical scale from 0 = no understanding at all, 0.5 = a basic understanding, to 1 = I have mastered the concept), the pre-course responses "No familiarity at all", and "I have heard of it, but to not know what it is" were pooled and valued at 0, "I know the concept, but have never used it in a program" was coded as 0.5 and "I have implemented the concept in a program" was coded as 1. The stripped bars show the average achievement as assessed by projects and exams. Note: assessment of "Variable" was embedded throughout the course but never formally assessed in an isolated manner that would provide an average score to plot.

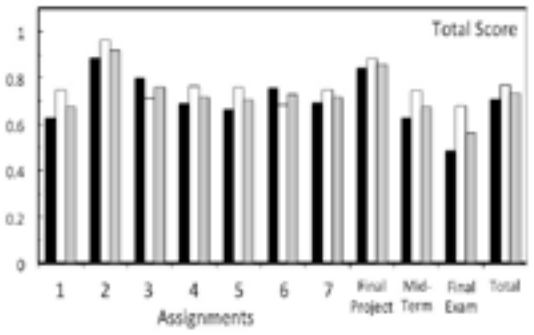

Figure 3 - Graph showing total scores on all assignments, midterm, final exam and final project. Solid Black is Undergraduate, White is Graduate and Grey is everyone combined.

Each week, the assignment was to create a 30-second composition using the key computer science skills learned from the lecture. The aesthetics of each assignment were assessed by outside music TA's who gave numbers from 0-3 (3 being most innovative and musically explorative). There is a general trend that if the students were having trouble with the computer science topics, the music also suffered.

Another observation from the data and graphs is that test scores show lower achievement than projects in general. This is somewhat obvious because when a student writes a program, the system tells them pretty quickly when it is wrong. Consider Loops: this was assessed early and was generally low but then again on the low on the final exam. Obviously students used loops throughout the semester but in a programming context students may have shifted being able to do it right the first time or spending a little more time debugging. On the exam where they could not let the lack of a functioning program tell them they made a mistake they may have been at a disadvantage from an objective evaluation standpoint. But, students who really "got it" did well on the tests as well.

5. DISCUSSION

The unique strength of our course is that we teach and provide digital arts students with tools they can use for serious projects and future work, but we also provide them knowledge and example code that demonstrates particular computer science and DSP concepts. The example code is parsimonious (due to the nature of ChucK), and is heavily commented so the students feel free to re-use and modify it. Each exercise and example the students learn, program, and modify demonstrates an important concept, algorithm, mechanism, etc.

Finally, teaching our courses in ChucK, which is free and open-source, gives art students the promise that they can use these in the future without prohibitive personal cost. Unlike engineering students, art students cannot be assured they will have employers that can afford expensive professional versions of software such as MATLAB, Max/MSP/Jitter, MAYA, etc. ChucK has a growing base of users, academic and also in production coding (a number of popular iPhone/iPad Apps are written at least in part in ChucK).

We have learned much, and plan to modify and assess our curriculum as we continue to offer the course(s) in the future. The course will be offered in the future, and to a wider population within the art school. The goal is to have all art majors take this course sequence, to offer the course to other institutions for adoption and/or modification, and to potentially offer a version of it online. All course materials and other supplemental materials are available at http://www.chucku.org.

6. ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1140336. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. The authors also wish to thank Ge Wang and Spencer Salazar for their help and advise, and continuing to hack ChucK to make it more useful for all. We also thank Dan Trueman, Ge Wang, Scott Smallwood, and Rebecca Fiebrink for their PLOrk/SLOrk curriculum materials. We would also like to thanks Sarah Nelson for her administrative support. We would also like to thank the Teaching Assistants Colin Honigman, Raphael Arar, Jon He, & Kameron Christopher

7. REFERENCES

[1] G. Wang and P. R. Cook, "ChucK: A concurrent, on-the-fly audio programming language," in *Proceedings of the International Computer Music Conference*, 2003, pp. 219–226.

[2] D. Trueman, P. Cook, S. Smallwood, and G. Wang, "Plork: The princeton laptop orchestra, year 1," in *Proceedings of the international computer music conference*, 2006, pp. 443–450.

[3] G. Wang, D. Trueman, S. Smallwood, and P. R. Cook, "The laptop orchestra as classroom," *Computer Music Journal*, vol. 32, no. 1, pp. 26–37, 2008.

[4] S. Dexter, "Teaching applet programming to non-majors-virtually," in *Frontiers in Education Conference, 2000. FIE 2000. 30th Annual*, 2000, vol. 2, p. S2D–6.

[5] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, "The scratch programming language and environment," *ACM Transactions on Computing Education (TOCE)*, vol. 10, no. 4, p. 16, 2010.

[6] W. Feurzeig, "An Introductory LOGO Teaching Sequence: LOGO Teaching Sequence on Logic - LOGO Reference Manual." ERIC Clearinghouse, 1970.

[7] S. Hambrusch, C. Hoffmann, J. T. Korb, M. Haugan, and A. L. Hosking, "A multidisciplinary approach towards computational thinking for science majors," in *ACM SIGCSE Bulletin*, 2009, vol. 41, pp. 183–187.

[8] J. M. Zelle, "Python as a First Language," presented at the Proceedings of the 13th Annual Midwest Computer Conference, 1999.

[9] A. Forte and Guzdial, M., ", A. M. . Motivation and non-majors in computer science: Indentifying discrete audiences for introductory courses," *IEEE Transactions on Education*, vol. 48, no. 2, pp. 248–253, 2005.

[10] B. Dorn, A. E. Tew, and M. Guzdial, "Introductory computing construct use in an end-user programming community," in *Visual Languages and Human-Centric Computing, 2007. VL/HCC 2007. IEEE Symposium on*, 2007, pp. 27–32.

[11] Head, L., "NSF 1044734, Music, Signals & Systems: Non-disciplined Education in a Multi-Campus Setting." National Science Foundation, Oct-2011.

[12] Steiglitz, K. and Lansky, P., "EIN: A Signal Processing Scratchpad," *Computer Music Journal*, vol. 19, no. 3, pp. 18–25, 1995.

[13] Heines, J., "NSF 1118435, Computational Thinking through Computing and Music." National Science Foundation, Aug-2011.