A Media Computation Course for Non-Majors

Mark Guzdial
College of Computing
Georgia Institute of Technology
801 Atlantic Drive
Atlanta, GA 30332-0280
guzdial@cc.gatech.edu

Abstract

Computing may well become considered an essential part of a liberal education, but introductory programming courses will not look like the way that they do today. Current CS1 course are failing dramatically. We are developing a new course, to be taught starting in Spring 2003, which uses *computation for communication* as a guiding principle. Students learn to program by writing Python programs for manipulating sound, images, and movies. This paper describes the course development and the tools developed for the course. The talk will include the first round of assessment results.

1 Issues with Current Introductory CS Courses

Computer science departments are not currently successful at reaching a wide range of students who are taking introductory computer science. The evidence for this statement includes international studies of programming performance [10], declining retention rates [5], and failure rates sometimes as high as 30% [17]. In particular, participation of women in computer science is dropping. Studies suggest that computing courses are seen as overly-technical and avoiding relationships to real applications [9], and are frankly boring and lacking opportunity for creativity [1].

At Georgia Institute of Technology ("Georgia Tech"), all students are required to take an introductory course in computing, including programming skills. The current course[1] is undoubtedly one of the most unpopular courses on campus, especially among those *not* in explicitly computing-related fields. While this is certainly a problem for the College of Computing at Georgia Tech (where the course has its academic home), it points towards a larger problem for the field.

Alan Perlis in April 1961 made perhaps the first argument that programming should be part of a liberal education for *all* students. If Calculus is the study of *rates*, and that's important enough to be part of the liberal education, then so should computer science. Perlis argued that computer science is the study of *processes*, which is certainly relevant to even more fields than those concerned with rates. The argument has been echoed and strengthened over the intervening years—by Seymour Papert arguing for a programming as a way of learning about learning [13][14], to Andrea diSessa's arguments for "computational literacy" as a critical component of many fields [3]. As long as non-CS-majors have such a dislike for computing, the hope is diminished for computer science as an accepted part of a liberal education and for computing generally to meet its potential for intellectual impact across the range of disciplines, not just in computational science and engineering.

We are developing a new course *Introduction to Media Computation* around a theme of *computation for communications.* The premises and core concepts of the proposed course are:

- All media are being published today in a digital format.
- Digital formats are amenable to manipulation, creation, analysis, and transformation by computer. Text can be interpreted, numbers can be transformed into graphs, video images can be merged, and sounds can be created. We call these activities *media computation.*
- Software is the tool for manipulating digital media. Knowing how to program thus becomes a communications skill. If someone wants to say something that her tools do not support, knowing how to program affords the creation of the desired statement. If she

[1] A second course for Engineering students only is at a prototype stage.

understands what her tools are doing, she may become a more adept practitioner, and more capable of transferring knowledge between tools.

- Core computer science concepts can be introduced through media computation. For example, programs can get large and cumbersome. Abstraction is our tool for managing program complexity and allowing programs to become even larger yet more flexible. However, computing has limitations. There are some programs that cannot complete in our lifetime, and knowing that these limitations exist is important for technological professionals.

This paper describes the course, its approach, and the technological materials being developed for it. The presentation will also include results of the pilot course evaluations.

2 Current State in Development of "Introduction to Media Computation"

CS1315 Introduction to Media Computation will be offered for the first time in Spring 2003 to a pilot class of 100 students. We will iterate on the course during Summer 2003 and implement at a full-scale in Fall 2003, with two sections of 250-300 students. It will be offered at a similar scale in succeeding terms. Currently, we are planning the course and developing course materials

Our learning objectives are:

- Students will be able to read, understand, and make functional alterations to small programs (less than 50 lines) that achieve useful communications tasks. Note that this is a very different goal than being able to write 50 lines from scratch. We see these students as developing *tool building* skill, not *software development* skill.
- Students will appreciate what computer scientists do and the key concerns of that field that relate to students' professional lives.
  - Students will recognize that all digital data is an encoding or representation, and that the encoding is itself a choice.
  - Students will understand that all algorithms consist of manipulating data, iteration (looping), and making choices — at the lowest level, these are choices about numbers, but we can encode more meaningful data in terms of those numbers.
  - Students will recognize that some algorithms cannot complete in reasonable time or at all.
  - Students will appreciate some differences between imperative, functional, and object-oriented approaches to programming.
  - Students will appreciate the value of a programming vs. direct-manipulation interface approach to computer use and will be able to describe situations where the former is preferable to the latter.
- Students will be able to identify the key components of computer hardware and how that relates to software speed (e.g., interpretation vs. compilation)
- Students will develop a set of usable computing skills, including the ability to write small scripts, build graphs, and manipulate databases – not necessarily using the common tools, but in a manner that exposes concepts and enables future learning.

We have developed (and are continuing to develop) a set of course notes and lecture slides that support the course. Overall, the course is designed to meet the "Imperative First" CS1 general structure and requirements in the new ACM/IEEE Computing Curriculum 2001 [2]. The order of media covered in the course is arranged to correspond to an increasing level of complexity in data structures.

- A sound is an *array* of *samples*.
- A picture is a *matrix* (two-dimensional array) of *pixels*.
- A directory structure (of media files, to process many files with a single recipe) is a *tree* of files.
- A movie is an *array* of *matrices* (frames, as pictures).

The media thus serve as a way of visualizing and making concrete (and interesting, we believe) the programs that the students are writing. Once the students are writing programs of increasing complexity, we introduce the ideas of algorithm complexity, object-oriented programming, and recursion as techniques for managing that complexity. Java is introduced only briefly at the end as a means for accessing the lowest levels of these data representations. The plan is neither to teach the students Java nor to introduce all the features of Java. Rather, only those aspects of Java that directly correspond to language elements they're familiar with will be introduced. The points are to emphasize the existence of alternative notations and that what the students have learned is applicable in other contexts.

We have been developing the course in a collaborative process with a board of faculty advisors from across campus and a team of undergraduate and graduate students developing materials. We have been using both on-line and face-to-face forums to gather input on the course (`http://coweb.cc.gatech.edu/mediaComp-plan`). Both student and faculty feedback has been very positive.

3 Technological Support for the Course

The language for the course will be Python (`http://www.python.org`). Python is a popular programming language used today by companies including Google and Industrial Light & Magic. It's most often used for Web (e.g., CGI script) programming and for media manipulation. Python was specifically developed to be easy-to-use, especially for non-traditional programmers.

The specific version of Python that we'll be using is Jython (`http://www.jython.org`). Jython is an implementation of Python in the popular programming language Java. Anything that one can do in Java (e.g., servlets, database programming via JDBC, GUI programming via Swing) can be done in Jython. Jython *is* Python—learning one is the same as learning the other.

We chose Jython in order to enable cross-platform multimedia manipulation. We have written a set of Java classes that encapsulate the kind of multimedia functionality that our examples will require, as well as a set of Jython classes that provide a simple and useful API to those functionalities. The API was designed based on existing literature on challenges that students find in learning to program, e.g., we allow set-based manipulation of samples and pixels before more complex and general iteration structures are learned [11][12].

Our API allows for access to the samples that make up sounds and the pixels that make up pictures.

- Figure 1 is an example program using our API that converts a picture object to greyscale. It computes the intensity of a given pixel by averaging the red, green, and blue components, and then replaces the color of that pixel with a gray pixel (red, green, and blue components the same) with the same intensity. Notice that the loop in this example is phrased as a set operation—essentially, "for every pixel *p* in the pixels of the given *picture*, do...." Some research on novice programming suggests that this is a simpler concept to begin with, as a way of easing into iteration [11].

- Figure 2 is a program that normalizes sounds to a maximum volume, by searching for the largest sample, computing a multiplier so that that sample would reach the maximum amplitude, and then multiplies all samples in the sound to raise the amplitude of the overall sound. We continue to use the simpler form of iteration here, but using multiple loops—an increase in complexity.

- Figure 3 takes a filename, then returns the sound in that file in reverse. Here, we use a more conventional `for` loop, with explicit indices in order to copy the array elements correctly.

We have also created a set of tools to support the students' tasks in this course. Our first and immediate need was for some kind of development environment. Jython is a new language [15], so most developers simply use plain text editors, or make do with Python or Java development environments. We believe that non-CS major freshmen will require more support. A team of undergraduate senior design students created our tool for students, *JES (Jython Environment for Students)* as a simple editor and program execution IDE (Figure 4). JES runs identically on Windows, Macintosh OS X, and Linux systems. Further JES development is occurring with undergraduates (mostly programming) and graduate students (developing documentation).

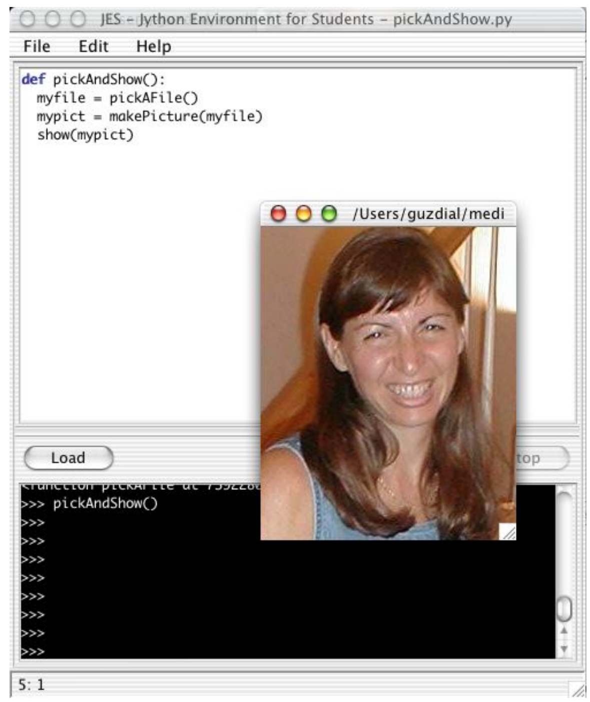

Figure 4: JES: Jython Environment for Students, with a graphics example running

We also realized that our students would have a need to visualize and explore media and to prepare media for use in their programs. For example, we would like to be able to look at sounds using a variety of visualizations, record their own sounds, investigate the RGB values in pictures of their choosing, and burst MPEG movies into folders of JPEG frames for ease in manipulation. By using their own media, we hope to make the student programming assignments into a creative activity, and thus, make it more attractive to women and others dissuaded by the stereotype of computer science as non-creative [9][16]. Another team of undergraduate students have modified the media tools in Squeak [4][6] to create cross-platform media exploration and manipulation tools (Figure 5).

Finally, we plan to use our CoWeb collaboration tool

```
def greyScale(picture):
  for p in getPixels(picture):
    intensity = (getRed(p)+getGreen(p)+getBlue(p))/3
    setColor(p,makeColor(intensity,intensity,intensity))
```

Figure 1: An example Jython program using our API to convert a picture to greyscale

```
def normalize(sound):
    largest = 0
    for s in getSamples(sound):
        largest = max(largest,getSample(s) )
    multiplier = 32767.0 / largest

    print "Largest sample value in original sound was",  largest
    print "Multiplier is", multiplier

    for s in getSamples(sound):
        louder =  multiplier * getSample(s)
        setSample(s,louder)
```

Figure 2: An example Jython program using our API to normalize sounds to a maximum volume

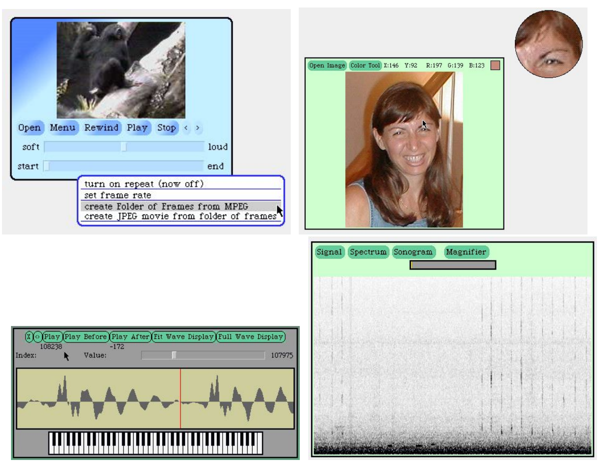

Figure 5: MediaTools: Movie tools (ul), image tools (ur), sound editing (ll), and sound views such as a sonogram (lr)

to support a collaborative experience for students in the Media Computation course. The CoWeb has been used successfully in a variety of classes, including computer science. By encouraging students to share their creative artifacts via the CoWeb, we further erode the perspective of computer science as a loner, non-creative activity. Further, we plan to use the CoWeb to support student, e.g. asking questions about the multimedia assignments. Such support may be critical to the success of the project. A factor analysis considering a range of variables influencing CS1 success, with completion as an outcome variable, suggests that comfort asking questions is the most critical factor for succeeding in CS1 [18]. Findings suggest that Web-based collaboration tools encourage much greater participation and comfort than in-class questions [7][8].

4 Summary and Future Directions

We believe that programming and computation will become part of a general, liberal education, but computing courses will have to change to make this happen. We are exploring a media computation approach that we feel will appeal to a liberal arts major, while still retaining a focus on programming. Our assessment will explore how well these students learn, but also how motivated they are. A key question for us is whether these students will have a continuing interest in learning about computation, a critical goal of any introductory course.

References

[1] AAUW. *Tech-Savvy: Educating Girls in the New Computer Age*. American Association of University Women Education Foundation, New York, 2000.

[2] ACM/IEEE. Computing curriculum 2001. *http://www.acm.org/sigcse/cc2001* (2001).

[3] diSessa, A. *Changing Minds*. MIT Press, Cambridge, MA, 2001.

[4] Guzdial, M. *Squeak: Object-oriented design with Multimedia Applications*. Prentice-Hall, Englewood, NJ, 2001.

```
def backwards(filename):
  source = makeSound(filename)
  target = makeSound(filename)

  sourceIndex = getLength(source)
  for targetIndex in range(1,getLength(target)+1):
    sourceValue = getSampleValueAt(source,sourceIndex)
    setSampleValueAt(target,targetIndex,sourceValue)
    sourceIndex = sourceIndex - 1

  return target
```

Figure 3: Return the sound in the file backwards

[5] Guzdial, M. Summary: Retention rates in cs vs. institution. Message posted on acm sigcse moderated members list, Georgia Tech, April 23 2002.

[6] Guzdial, M., and Rose, K., Eds. *Squeak, Open Personal Computing for Multimedia*. Prentice-Hall, Englewood, NJ, 2001.

[7] Guzdial, M., and Turns, J. Effective discussion through a computer-mediated anchored forum. *Journal of the Learning Sciences 9*, 4 (2000), 437–470.

[8] Hudson, J. M., and Bruckman, A. Irc francais: The creation of an internet-based sla community. *Computer Assisted Language Learning (CALL) 15*, 2 (2002), 109–134.

[9] Margolis, J., and Fisher, A. *Unlocking the Clubhouse: Women in Computing*. MIT Press, Cambridge, MA, 2002.

[10] McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., Laxer, C., Thomas, L., Utting, I., and Wilusz, T. A multinational, multi-institutional study of assessment of programming skills of first-year cs students. *ACM SIGCSE Bulletin 33*, 4 (2001), 125–140.

[11] Miller, L. A. Programming by non-programmers. *International Journal of Man-Machine Studies 6* (1974), 237–260. Participants strongly preferred to use set and subset expressions to specify the operations in aggregate.

[12] Miller, L. A. Natural language programming: Styles, strategies, and contrasts. *IBM Systems Journal 20*, 2 (1981), 184–215. Languages require iteration where aggregate operations are much easier for novices.

[13] Papert, S. Teaching children to be mathematicians versus teaching about mathematics. Ai memo no. 249 and logo memo no. 4, MIT, 1971.

[14] Papert, S. *Mindstorms: Children, computers, and powerful ideas*. Basic Books, New York, NY, 1980.

[15] Pedroni, S., and Rappin, N. *Jython Essentials*. O'Reilly and Associates, 2002.

[16] Pfleeger, S. L., Teller, P., Castaneda, S. E., Wilson, M., and Lindley, R. Increasing the enrollment of women in computer science. In *The Proceedings of the Thirty-second SIGCSE Technical Symposium on Computer Science Education*, R. McCauley and J. Gersting, Eds. ACM Press, New York, 2001, pp. 386–387.

[17] Roumani, H. Design guidelines for the lab component of objects-first cs1. In *The Proceedings of the Thirty-third SIGCSE Technical Symposium on Computer Science Education, 2002*, D. Knox, Ed. ACM, New York, 2002, pp. 222–226. WFD (Withdrawl-Failure-D) rates in CS1 in excess of 30

[18] Wilson, B. C., and Shrock, S. Contributing to success in an introductory computer science course: A study of twelve factors. In *The Proceedings of the Thirty-second SIGCSE Technical Symposium on Computer Science Education*, R. McCauley and J. Gersting, Eds. ACM, New York, 2001, pp. 184–188.