Introducing "Introduction to Media Computation" A New Course For Computing Literacy General Education

Mark Guzdial
College of Computing
`guzdial@cc.gatech.edu`
404-894-5618

September 6, 2002

"Introduction to Media Computation" is a new three credit hour (three one hour lectures a week) course being developed within the College of Computing designed to meet the computer literacy general education requirements (formerly the "Core Area B" requirement) It has been approved to meet that requirement by the Institute Undergraduate Curriculum Committee, pending approval by the Academic Senate. The course is aimed to appeal to the needs and interests of non-Computer Science and non-Engineering majors[1]. It is *not* equivalent to CS1321 nor COE1361, nor does it currently[2] serve as a pre-requisite to any other course in the College of Computing or elsewhere.

The course will be offered as *CS 4803* (Special Topics) during the Spring 2003 Semester for a pilot offering of **100 seats**. Our plan is to propose the course as *CS1315* after the pilot and start offering it in Fall 2003 with two sections of 250-300 (as needed to meet demand).

This document introduces the course. We in the College of Computing encourage interested academic units to propose to the Institute Undergraduate Curriculum Committee that students in their programs be allowed to take this course to meet their Core Area B requirement. (See Joe Hughes Sept. 3, 2002, memo on changing undergraduate computing requirements.) If you have any questions, please contact Mark Guzdial.

It is important to realize that *Introduction to Media Computation* is **not** designed to be an *easier* course than the existing CS1321. It's designed to be a very *different* course—there will be less emphasis on abstraction and programming large systems, but there will be more emphasis on mathematics (just high school level, e.g., trigonometry for computing and adding sine waves in generating sounds) and physics (e.g., composing colors from red, green, and blue components). The course aims at developing professionals who (a) come to computing with data (media) that they want to process (e.g., a *"data-first"* approach) and (b) will never write programs larger than a couple hundred lines of code, and will probably only write small scripts of around a dozen lines.

[1]Certainly, Engineering students are welcome, and some Engineering programs may want to allow this course to their students for their computing gen ed requirement. The course will be lacking some important Engineering issues like Matlab and an emphasis on numeric methods

[2]A second course might be developed later if there is demand.

1 Registration Policy for Spring 2003

One of the common questions about the class has been "How are you going to handle the demand for this class?"

Here's the plan:

- During the Spring 2003 semester, we will offer **100** seats as part of a pilot, where we want the majority of these students to be first and second year students—that's who we want to be the common student in this class when it reaches steady-state, so we want to trial the course with that audience.
- During the Fall 2003 semester, we plan to offer two sections of 250-300.
- We plan to continue with two sections in the Spring 2004 semester, and then balance offerings of the course with load.

Here's how we're going to handle registration of the initial 100 seats. **Note that the class will be open *only* to those programs who permit their students to take the course for their computing literacy general education component.** While we'll welcome students from all programs (including those who may want the course as an elective) later, we want to focus on students who can use the course towards their requirements in the pilot. **Please do inform us if your program will be permitting students to register for the pilot.**

20 seats in the pilot will be open from the first day of registration. The other 80 seats will be opened on the first day that Freshmen can register, but the course will then be *closed* to all but first and second year students.

2 Topics of the Course

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.
- 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.

The course is being developed by Professor Mark Guzdial in a collaborative design process with students and faculty from outside of computer science. We have been using both on-line and face-to-face forums to gather input on the course—please see `http://coweb.cc.gatech.edu/mediaComp-plan`. Both student and faculty feedback has been very positive:

I'm very enthusiastic about your proposal, and know (others in my department) would be too. We very much need *this kind of class. I think the structure of the proposed class is really inspired and is exactly the right approach for our students.* **(LCC professor)**

*The proposed course is definitely a motivator, since the current requirement does little but get in the way of the courses for the major. Integrating the required computer education credit with an area that sufficiently yields to the material of the major/field would be enormously beneficial. I am pleased that GA Tech is responding to students concerns and allowing those of us not majoring in CS to take a more creative based course that will be beneficial instead of a chore.***(Architecture student)**

I think this is a good idea and I wished I'd had the opportunity to participate in this instead of (the current CS class). **(HST student)**.

The one problem that I am worried about if this class were to be added to the curriculum is the amount of spots open during registration. Everyone I know dreads taking the CS courses that are available now, and they'd jump at the chance to take this so it would fill up very quickly. **(Chemistry student)**

The faculty advisors who have been reviewing the course as it has been developed over the last few months are:

Athanassios Economou	Architecture
Jim McClellan	ECE
Donna C. Llewellyn	CETL
Patricia Kennington	Psychology
Mindy Millard-Stafford	Applied Phys.
Diane Gromala	LCC
Tom Morley	Math

The learning objectives for the course have been chosen to correspond closely to the General Education requirements for computing literacy.

- Students will be able to read, understand, and make functional alterations to small programs (less than 50 lines) that achieve useful communications tasks.
- 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, 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.

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. Java provides all the basic media manipulation functionality that we want in the class, but in a pretty complicated package. We have written code and tools so that students can do media manipulations (sound, images, movies) on Windows, Macintosh, and Linux systems identically with Jython.

2.1 A Relevant Jython Example for the Course

Here's a brief example of using Jython to give a flavor for what students might be doing in this class. The technique that weather forecasters use for displaying themselves in front of a weather map is called *chroma key*. It's not difficult to do, and it's a good example of a simple algorithm made concrete through the use of a visual representation.

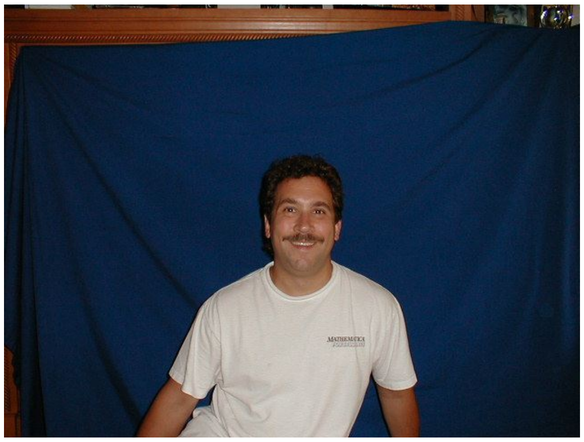

Figure 1: A subject in front of a blue background — our source

Pictures can be thought of as a two-dimensional matrix of *pixels*, each of which has a color that is a mixture of a red, green, and blue component. Imagine that you have a picture of a subject in front of a blue background (Figure 1)[3], and a new background that you'd like to have your subject in front of (Figure 2). The problem

[3]Which could just be your son's bedsheet attached to an entertainment center, as long as you promise to put it back after using it.

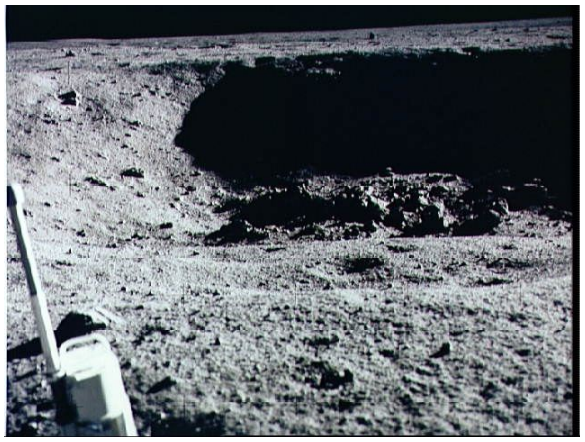

Figure 2: A new background for the subject — our background

is how to replace the blue (or mostly blue—there are clearly variations of blue in this picture) pixels with the color of the background.

The algorithm is very simple: Consider each pixel p in the source picture and whether it's mostly blue. How do we define "mostly blue"? Here's a simple definition: If there's more blue than the other components ($red + green \rightarrow yellow$), then we'll call it "mostly blue." We'll need the x, y position of the pixel p. We'll get the color of the pixel in the background at the same x, y, and set p's color to the color of that background pixel. In Jython using the media code we've developed, that algorithm looks like this:

```
def chromakey(source,bg):
  for p in getPixels(source):
    if (getRed(p)+getGreen(p) < getBlue(p)):
      setColor(p,getColor(getPixel(bg,getX(p),getY(p))))
  return source
```

The result looks like Figure 3. Could we do this in something like Adobe Photoshop? Of course we could. The advantage of having it encoded in a program (or *recipe* as a metaphor) is that we can now immediately replicate the process with a different source or a different background (Figure 4). Once we have the recipe, the computer will blindly follow the recipe wherever we ask it to. Knowing how to write such recipes provides one with enormous communications freedom (for websites, marketing, or any other place where one would want to create images).

3 Structure of the Course

The currently planned week-by-week outline of lectures and topics is described in Table 1. The order of media corresponds roughly 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*.

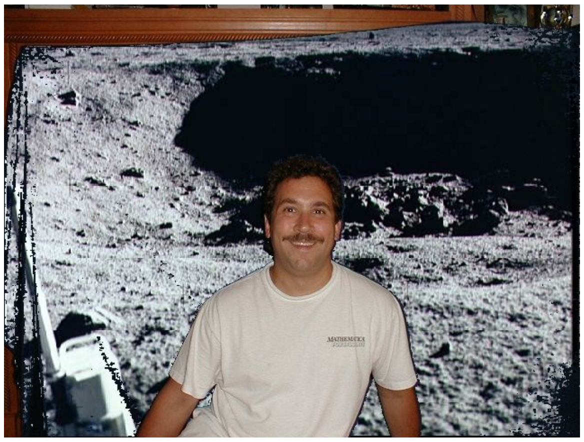

Figure 3: The subject on the moon

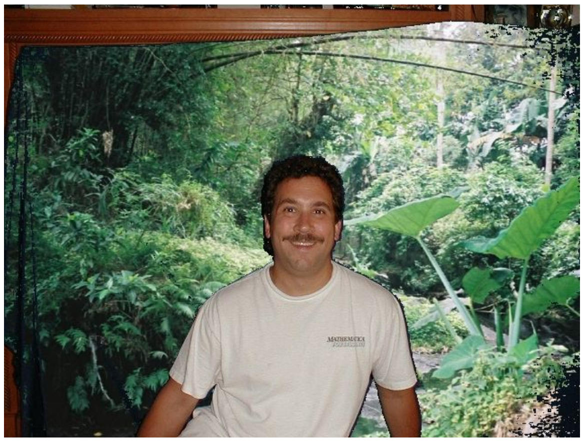

Figure 4: The subject in the jungle

- 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 hope) 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.

There will be a variety of different kinds of assignments in the course, designed to give the students ample opportunity to learn the material in a collaborative setting.

- Weekly *Lablets* (15% of grade) emphasize practice using concepts and learning basic computer skills (including Microsoft Office). Lablets are lab-like activities that students work on at home on their

Week 1 Introduction: What is Computer Science and Media Computation Variables and functions
Week 2 Sound as an array of samples Loops for manipulating samples
Week 3 How sound works and how it can be manipulated Increasing/decreasing volume, trimming sounds, creating reverb
Week 4 Developing a mental model of the program: Debugging Images as a two-dimensional array of pixels
Week 5 Manipulating images by changing RGB values Filtering images using conditionals (for thresholding functions)
Week 6 Manipulating a portion of an image: Masks and varying the loop endpoint Drawing on an image: Graphics on the image
Week 7 Developing a mental model of the program: Tracing conditionals and loops Manipulating the files that the media live in
Week 8 Writing scripts that move and process files Video: A series of images/frames in files
Week 9 Applying image techniques to video frames "Why is this taking so long?!?": An introduction to algorithm complexity
Week 10 Text as a media type: Manipulating text with programs Using databases to store media, text, and intermediate forms
Week 11 Graphing data: Media conversion from text to graphics Graphing data with an external program: Using Excel and preparing data for it
Week 12 "Can't we do this any easier?": Functional decomposition to reduce program complexity "Can't we do this any easier?": Recursion to traverse data
Week 13 "Can't we do this any easier?": Objects as a technique to manage complexity
Week 14 Applying these techniques to media manipulation Thinking about languages and representations for process: What computer scientists do
Week 15 Introduction to Java Java for Media Manipulation

Table 1: Outline of the proposed course

own computer. Lablets are collaborative: Students can work together, but each student turns in an assignment, identifying her collaborators. The goal is for these to take one to two hours to complete.

- There will be three to four *Quizzes* and *Pre-quizzes* (15%). A pre-quiz covers the same concepts as a quiz, and is collaborative. It's assigned on a Monday, due on Wednesday, with the matching (individual) quiz on the Wednesday. Quizzes will have a focus on *understanding* a given program, e.g., "At this point in the execution, what would the value of X be? What would be on the screen?"
- *Homework* (five or six a semester — 15% of grade) is assigned on Mondays and is due the following week. Homework can be collaborative. It will vary in complexity, but will typically take one to four hours. Much of the homework will be changing an *existing* program or combining existing programs rather than coding from scratch.
- *Projects* (two to three a semester—20% of grade) are similar to Homework, but are *not* collaborative. Projects will typically be slightly less complex than Homework, but not take less than one hour.
- There will be two midterm and one final *Exams* (35% of grade).