TEACHING COMPUTER GRAPHICS IN A SMALL DEPARTMENT

Michael B. Gousie
Department of Mathematics & Computer Science
Wheaton College
Norton, MA 02766
(508) 286-3969
mgousie@wheatoncollege.edu

ABSTRACT

This paper describes a computer graphics course implemented in a small department where breadth of content is as important as depth. The course teaches traditional computer graphics while incorporating ideas from other computer science disciplines, such as Theory of Computation and Compiler Construction, into programming projects. This gives students a feel for some additional topics that may not be offered formally at a small school. The course content and the projects are described, and some improvements for the next iteration are presented.

1 INTRODUCTION

Computer graphics has been developing at an incredibly rapid pace. Whereas the subject was much constrained in the 80's because of poor software and expensive equipment, the advent of new technologies in the 90's has raised the level of courses at many schools. The new curriculum now in place at such institutions covers many more three-dimensional concepts instead of the more traditional two-dimensional background once the standard. These 3-D concepts can now be implemented more easily by students using an appropriate Application Programmer Interface (API), such as OpenGL [13] or the like. Indeed, the emphasis in computer graphics curriculums has changed so much that the ACM is currently working on a new standard.

Compared to a university or research institution with a large computer science department, smaller liberal arts schools with their attendant smaller computer science departments may have different goals or priorities for elective courses such as computer graphics. At Wheaton College, it is difficult for the department to service the basic computer science core courses, let alone offer many interesting electives. It is imperative, then, that the electives that are offered integrate as much useful, interdisciplinary material as possible. This approach offers students a chance to experience several aspects of computer science within one course. Computer graphics is one such course that allows much flexibility, especially in projects, that makes it very useful in a small department.

2 COMPUTER GRAPHICS CURRICULUMS

In the early days of computer graphics, only large universities had the funds for equipment and a large pool of motivated students necessary to tackle the difficult subject matter. Much of the course content was still in the realm of research at that time. Students were expected to learn the fundamentals of computer graphics, including graphics hardware, low-level algorithms for basic operations like line drawing and clipping, matrix transformations, and visible surface algorithms [9]. The students then had the unenviable task of trying to generate images using crude software and special-purpose hardware.

In the 80's, there was a surge of popularity in computer science. More students became interested in the subject as a major. Because of this and the perception that it was interesting to generate images, computer graphics courses came to be more in demand. Also at this time, new, lower cost computers began appearing in schools. The advent of these machines and the popularity of computer graphics had instructors searching for better software. Unfortunately, API's such as CORE, GKS, PHIGS, etc. were never widely accepted and were short lived. Thus arose API's with minimal functionality, but which ran on cheaper hardware, such as Apple's Turtlegraphics in the early 80's and Borland Turbo Pascal's graphics library somewhat later, among others. Another approach was to use software written by university staff or students; the latter was the case with a classmate of mine as I was attending graduate school in the late 80's. In response to the interest in computer graphics, the ACM specified a curriculum in 1991 which included topics such as: graphics hardware, two- and three-dimensional primitives, object representations, two- and three-dimensional transformations, interactive graphics, hidden lines and surfaces, and so forth [17]. Text books from this era included the classic Foley and van Dam [3], Hearn and Baker [6], and Hill [8]. Each of these texts concentrate on ideas presented in the ACM guidelines, with relatively minimal coverage of realistic 3-D modeling and rendering.

Technology continued to evolve at a rapid pace in the 90's. Processors became ever faster, and video cards became more sophisticated to the point that complex, 3-D images could be produced very quickly. Moreover, the availability of better hardware resulted in better 3-D algorithms and techniques, as well as API's, such as OpenGL, to take advantage of the new technology. These advances, plus the higher expectations of students raised on sophisticated computer games and other, graphics intensive software [9], has affected a change in computer graphics syllabi at many institutions [19]. Whereas in the late 80's over 50% of course time was spent on two-dimensional topics, now at many institutions (e.g. California State University Stanislaus [2], California Polytechnic State University [9], and Insituto Superior Técnico [12]), these topics are mentioned only in the first few weeks [19], and in some cases, such as at the University of Utah [16], even less. Such a shift has also prompted a change in text, with many now using the book by Angel [1]. It should be noted, however, that many schools still generally adhere to the ACM guidelines, some adding additional 3-D topics (e.g. Brown University [18] and Williams College [11]), while others advocate an approach more towards engineering (e.g., Rensselaer Polytechnic Institute [5] and Winthrop University [10]). Certainly, graphics is a rich subject that lends itself to many different viewpoints.

3 COMPUTER GRAPHICS AT WHEATON COLLEGE

There are only two full-time computer science professors in the Mathematics and Computer Science Department at Wheaton. The department offers a major in computer science, as well as a minor and an inter-disciplinary major in mathematics and computer science. To adequately service the majors, the two instructors must teach a broad variety of subjects. However, because most of the courses we teach are core courses, we can not offer many interesting electives. Indeed, the department did not offer courses such as Theory of Computation, Compiler Construction, and Software Engineering. These courses are often part of the core at larger institutions. Furthermore, C++ was the language of choice, with other languages used only in dedicated courses, such as LISP in Artificial Intelligence.

A new computer graphics course was instituted in the fall of 1998. Due to the lack of breadth available to computer science majors, the course was designed in such a way so as to fill some of the gaps in the curriculum at the time. Up until the computer graphics course, computer science students at Wheaton had used primarily Macintosh-based software, with some limited exposure to Windows/NT. The first decision for the new course was to choose Linux to expose students to a different platform. Students were forced to navigate UNIX-based commands for the first time, without having simple icons from which to choose. As part of the course, the students installed and used Mesa, a free graphics library similar to OpenGL for Linux [14]. The projects were written in C instead of C++ to give students some familiarity with a different, albeit similar, language to what they were used to.

The content of the course roughly followed the ACM guidelines, moving from hardware to 2-D issues to 3-D transformations, using the newer Hearn and Baker text [7] with additional material taken from the updated book by Foley et al [4]. Topics from formal languages, compilers, and software engineering were incorporated, so chosen because the department did not offer courses in those areas and because elements could be more easily integrated into projects. Handouts were used for these non-traditional computer graphics topics. The course included six programming projects; students were forced to implement mathematical ideas in some of these, which bolstered the mathematical skills of some of the weaker students. The goals of the projects were to:

- Enhance understanding of basic computer graphics principles;
- Force students to design and code large projects in a different programming language;
- Allow students to interact via group projects;
- Concretize how mathematical concepts are used in applications;
- Include topics outside the realm of traditional computer graphics to add more breadth to the curriculum;

- Have fun implementing interesting projects.

4 COURSE PROJECTS

The course required the completion of six projects. The first three were individual projects and dealt primarily with two-dimensional ideas. The last three projects were implemented by students working in pairs. These projects included three-dimensional concepts and additional topics not usually found in such a course.

The first project was a simple warm-up exercise. The students were to use as many 2-D OpenGL commands as possible to display a fun image. This allowed students to write their first C programs while familiarizing themselves with Linux and OpenGL. For many students, this was the first time using makefiles and linking libraries to their code. Because they also installed the software, they necessarily had to completely understand the path structure of the Linux machines.

The second project allowed students to code some simple mathematics and to examine, in depth, an existing algorithm: Bresenham's line algorithm. They were allowed to use only OpenGL's pixel drawing primitive. Students quickly learned that it is not always trivial to implement a known algorithm. It was also apparent how mathematics play a pivotal role in this algorithm, and by extension, many graphics topics.

The third project was meant to sharpen students' linear algebra skills by forcing them to transform 2-dimensional objects. They were not allowed to use OpenGL's transformation functions, which meant they had to write their own matrix multiplication functions, as well as keep track of the Current Transformation Matrix (CTM) in a stack. Although all of the students had done matrix multiplications in Linear Algebra, it was an eye-opening experience for some when faced with the task of actually implementing such code. The input to the project was a file containing various codes specifying window coordinates, displaying polygons, and defining transformations, among other items. To read the input, a form of parsing was required, something usually done in a compiler course.

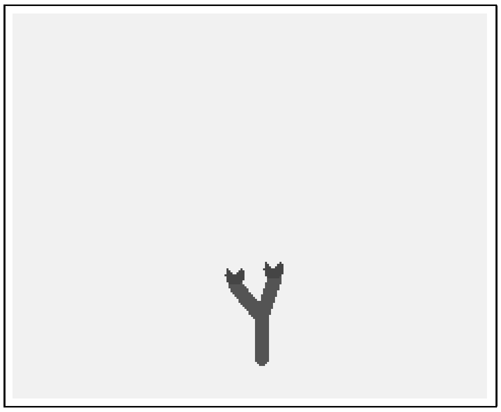

Figure 1: Initial tree using basic production.

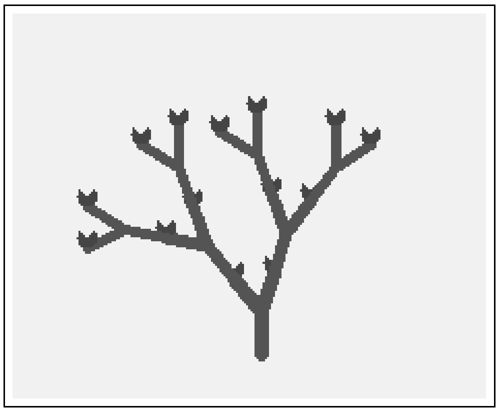

Figure 2: Tree after two additional iterations of string.

Projects four and five comprised the "Museum Project." The problem was to create an interactive program that allows museum visitors to create their own selfsimilar plants. Such plants can be defined by a Lindenmayer system (L-system), a notation for representing plants using strings with brackets [15]. The strings define plant portions, such as branches, leaves, and the like, while the brackets specify how the plant will "grow." For example, the production

```
S => B [ - - B F < S ] [ + B F < S ]
```

defines the basic tree template shown in Figure 1, where

S	=	start state
B	=	branch
F	=	flower/leaf
+,-	=	turn x degrees to left/right
<	=	growth point for subsequent iteration

The brackets show how the plant grows from the initial branch. Subsequent iterations copy this basic form at the indicated growth points, albeit in a somewhat smaller scale, resulting in the tree shown in Figure 2.

Some ideas from formal languages were covered in class to give students the necessary background in order to implement an L-system simulator, the heart of project four. The input to the project was a string defining a plant, the specifics of which were up to the students. The user could "grow" the plant to any desired size. Again, compiler parsing techniques were required in order to read in such strings. The project also required the file storage of the strings for use in the following assignment. A thorough "User's Guide" that explained the use of symbols and the formatting of input strings was a further requirement for the project, bringing in some documentation and specification ideas from software engineering. This project, then, was the back-end of a plant-creation program, letting users grow only plants already specified.

The next project was to put a Graphical User Interface (GUI) on top of the L-System simulator developed in the previous project so that casual museum goers could create their own plants, without any understanding of L-systems. The GUI had to allow the user to create a plant template, consisting of branches, leaves, and "growth points," all stored in a bracketed L-system string. The GUI portion proved to be more of a challenge than the students at first realized. A requirement of the project was for students to conduct an informal beta test, where users were to create their own plants. This survey showed the students how difficult it is to create a GUI that is truly easy and intuitive to use. Finally, the students had to submit a "User's Guide" for the project. In this, specifications had to be spelled out in detail. A sample of this project can be seen in Figures 3 and 4.

The final project was three-dimensional, incorporating fractals, transformations, triangularization, and color. The idea was to build a user-friendly package that allows a novice to create a three-dimensional fractal mountain, allowing her to change the size, orientation, and roughness (via the fractal dimension). By this time, all of the students were comfortable with OpenGL, and had no problems with writing a large amount of code dealing with 3-D concepts. A sample project can be seen in Figure 5.

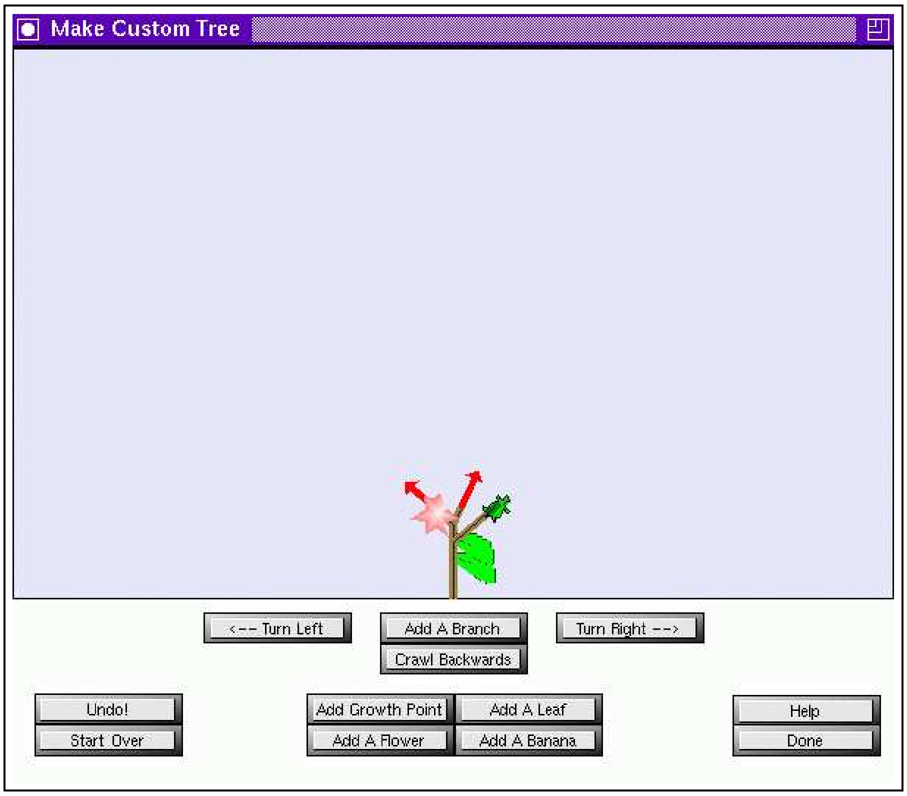

Figure 3: Output of Project 5 showing tree creation window. The arrows denote “growth points,” the turtle is the current position, and the other objects are primitives (flowers and leaves).

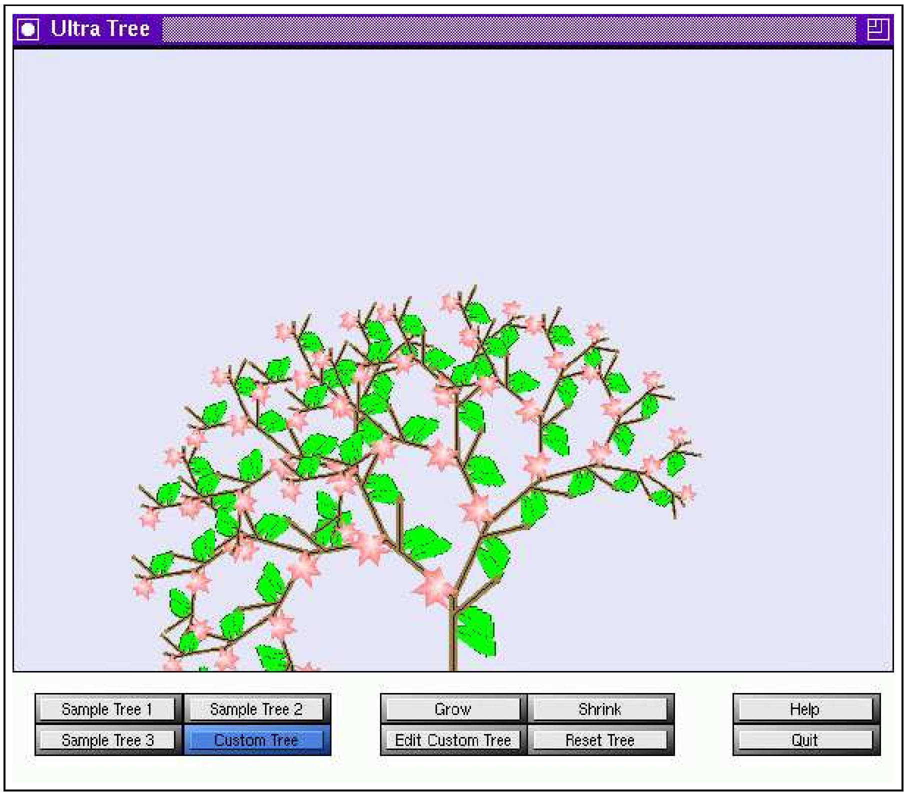

Figure 4: Output of Project 5 after several “grow” iterations.

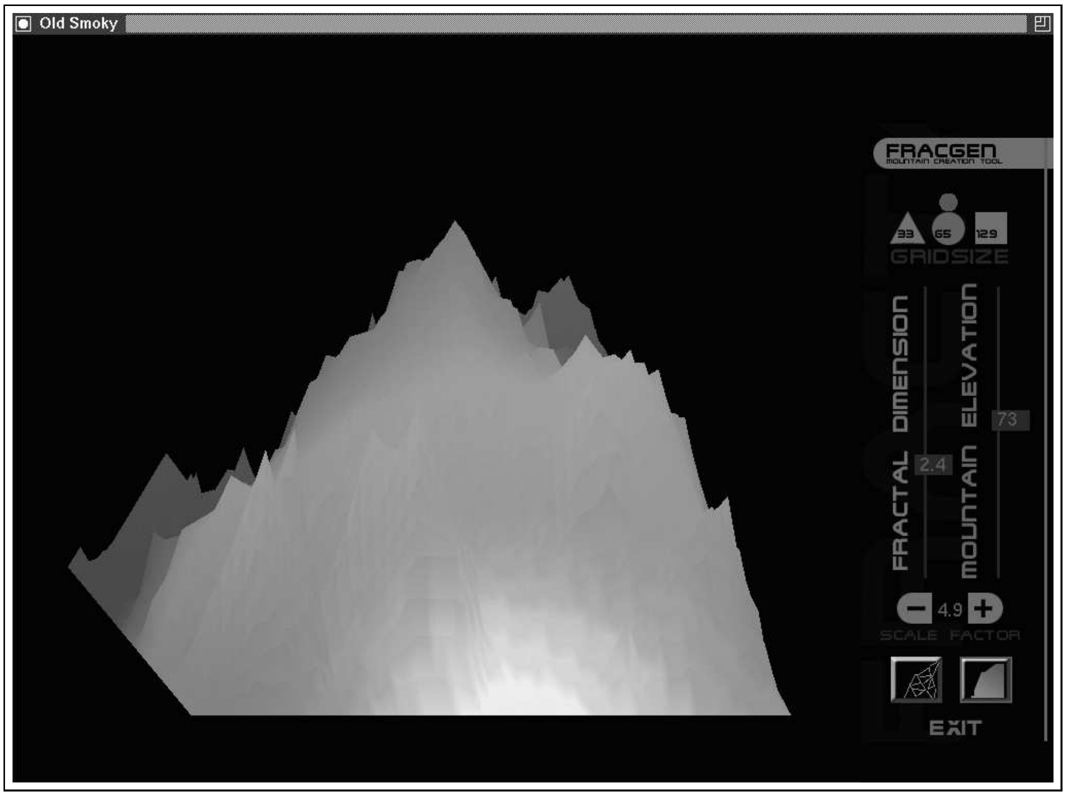

Figure 5: Output of Project 6.

5 CONCLUSION

The students seemed to enjoy the course, fulfilling one of the goals. They worked diligently on the projects, some of which were quite lengthy. For several students, it was the first time they had to design a large project on their own, which by itself is very worthwhile. Incorporating ideas from other areas of computer science worked quite well; students were especially excited about the museum project even though it required some understanding of formal languages. They also enjoyed the fractal mountain project, seeing three-dimensional concepts put to work. Class time spent on non-traditional computer graphics topics seemed very worthwhile. The extra topics only enhanced the projects without diminishing any computer graphics content, while giving students a broader appreciation of computer science and mathematics.

In designing a future computer graphics course, some changes are in order. For example, while the second project showed students how even a seemingly simple algorithm is not as easy to implement as one might think, the time taken on this project could have been better spent. Students also prefer working on projects that incorporate interactivity and three-dimensional objects. Following some of the larger institutions' lead, three-dimensional concepts can be brought in much earlier, while still allowing for the inclusion of non-traditional computer graphics topics. Putting aside some of the 2-D topics will allow more time for advanced 3-D topics. A switch to Angel's text [1] may promote a faster and easier transition to 3-D projects. Finally, one may envision that the approach taken with this computer graphics course can be applied to other upper-level courses, with equally interesting results.

References

[1] Angel, E. *Interactive Computer Graphics: A Top-down approach with OpenGL*. Addison Wesley, 1997.

[2] Cunningham, S. Re-inventing the introductory computer graphics course: Providing tools for a wider audience. In *Computer Graphics and Visualization Education 1999* (1999), J. C. Teixeira, W. Hansmann, and M. B. McGrath, Eds., EUROGRAPHICS and ACM SIGGRAPH, pp. 45–50.

[3] Foley, J. D., and van Dam, A. *Fundamentals of Interactive Computer Graphics*. Addison Wesley, 1982.

[4] Foley, J. D., van Dam, A., Feiner, S., and Hughes, J. *Computer Graphics: Principles and Practice*, 2 ed. Addison Wesley, 1990.

[5] Franklin, W. R. Computer graphics course calendar/syllabus. `www.rpi.edu/dept/ecse/graphics-s99/`, 1999.

[6] Hearn, D., and Baker, M. P. *Computer Graphics*. Prentice Hall, 1986.

[7] Hearn, D., and Baker, M. P. *Computer Graphics:C Version*, 2 ed. Prentice Hall, 1997.

[8] Hill Jr., F. S. *Computer Graphics*. Macmillan, 1990.

[9] Hitchner, L. E., and Sowizral, H. A. Adapting computer graphics curricula to changes in graphics. In *Computer Graphics and Visualization Education 1999* (1999), J. C. Teixeira, W. Hansmann, and M. B. McGrath, Eds., EUROGRAPHICS and ACM SIGGRAPH, pp. 30–36.

[10] Larrondo-Petrie, M. M., Bresenham, J., Laxer, C., Lansdown, J., and Owen, G. S. Approaches to teaching introductory computer graphics. In *Computer Graphics* (1994), pp. 479–480.

[11] Lenhart, B. Csci 371: Computer graphics. `www.cs.williams.edu/~lenhart/courses/csci371/assignments.html`, 1999.

[12] Lopes, J. B., Gomes, M. R., Bernardo, J., Jorge, J., and Pereira, J. M. Restructuring computer graphics and visualisation curricula at ist. In *Computer Graphics and Visualization Education 1999* (1999), J. C. Teixeira, W. Hansmann, and M. B. McGrath, Eds., EUROGRAPHICS and ACM SIGGRAPH, pp. 17–22.

[13] OpenGL Architecture Review Board, Woo, M., Neider, J., and Davis, T. *OpenGL Programming Guide*, 2 ed. Addison Wesley, 1997.

[14] Paul, B. The mesa 3d graphics library. `www.mesa3d.org`.

[15] Prusinkiewicz, P., and Hanan, J. *Lindenmayer Systems, Fractals, and Plants*. Lecture Notes in Biomathematics. Springer-Verlag, 1989.

[16] Shirley, P. Cs431: Introduction to computer graphics. `www.cs.utah.edu/~shirley/classes/cs431f96/`, 1996.

[17] Tucker, A. B., and Barnes, B. H., Eds. Computing curricula 1991: Report of the ACM/IEEE/CS curriculum task force. *Computing Curricula 1991* (1991).

[18] van Dam, A. Cs123 syllabus. `www.cs.brown.edu/courses/cs123/`, 1999.

[19] Wolfe, R. Bringing the introductory computer graphics course into the 21^{st} century. In *Computer Graphics and Visualization Education 1999* (1999), J. C. Teixeira, W. Hansmann, and M. B. McGrath, Eds., EUROGRAPHICS and ACM SIGGRAPH, pp. 3–8.