Reprinted from the November 1998 issue of *Computer Graphics*

OpenGL: Agent of Change or Sign of the Times?

Rosalee Wolfe
DePaul University

Although curriculum is a recurring theme of conversation in the hallways at conferences involving graphics and education, *Curriculum 91*[1] marks the most recent formal discussion regarding the topics in an introductory computer graphics course, and it was published in February 1991, nearly nine years ago. Nine years represents a significant percentage of the discipline's life span. Further, Steve Cunningham, author of the computer graphics course that appears in *Curriculum 91*, points out that although the document was published in 1991, it reflects accepted practice from the late 1980s. So in fact it has been ten years or even longer since substantive discussions on this topic have taken place. Much has changed in that time.

Graphics in the late 80s

The computing environment of ten years ago presented several substantial technology challenges to a graphics instructor. In universities graphics hardware was still considered special purpose equipment and was not ubiquitous as it is today. *Curriculum 91* mentions "a high quality color display" as a special laboratory item. Intel-based PCs were expensive and many schools relied on graphics terminals connected to a mainframe via serial lines.

Another special need that *Curriculum 91* mentions is "a suite of graphics software tools." Graphics software was scarce, expensive and difficult to access. When trying to find appropriate software an instructor faced three difficult alternatives. The first was to find the considerable monetary resources necessary to purchase a graphics library from a vendor. At the time, these libraries ran into the tens of thousands of dollars, and many schools simply could not afford the purchase.

The second option was to use free software, typically developed at another university. This was at a time before Internet use was as widespread as it is today, and net searching was not possible. In many cases an instructor learned about such software by reading a published article or by word of mouth. Often, just to obtain the software would require sending a magnetic tape through the mail and waiting weeks for it to return. While the price was right, most freeware demanded large amounts of time to install and even then the instructor often had to spend additional time developing custom software in order to accommodate the peculiarities of the school's hardware.

As a result many instructors chose a third approach to software tools. On their local system, they wrote just enough software to supply their students with the bare necessities for graphics work. This included the functions GetPixel(), PutPixel() and maybe DrawLine() in addition to routines that began and ended a graphics session. This slim set of tools was all that students had for software development.

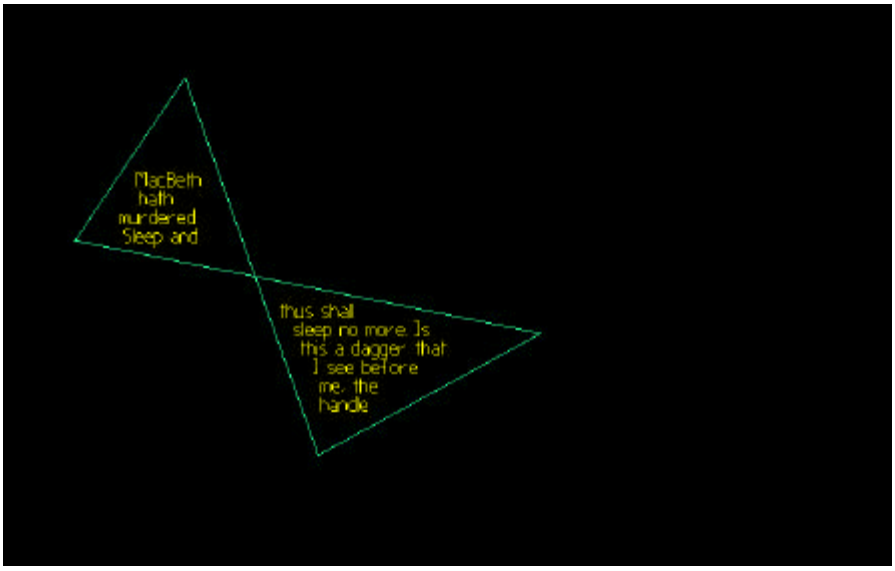
In this setting, students in an introductory graphics course would write an entire graphics package from the ground up. Class lectures would parallel the software development and cover such topics as those listed in Table 1.

- line and curve drawing
- window-to-viewport transformation
- clipping
- modeling transformations
- polygon rasterization
- viewing transformations
- surface algorithms
- simple illumination
- shading
- interactive techniques

Table 1: Graphics courses in the late 1980s typically devoted 50% of their time to two-dimensional topics.

The pace of coverage was somewhat dictated by the software implementation progress, and often the hardware added complexity to an already challenging software problem. Compensating for the low number of colors available on a graphics display required quantization and dithering. Merely viewing an image on a graphics terminal could be a time consuming process, as it took over 20 minutes to display an image with a resolution of 800x480.

All of these factors influenced course content. Typically fifty percent of a late 1980s introductory graphics course covered two-dimensional issues. Figures 1 and 2 are examples of student projects from that era and reflect the course organization. In both cases, students developed the projects from a very limited set of function calls. The first is an interactive text-formatting program that accepted a text file as input and allowed a user to specify a shape and font size via a mouse. The program used polygon rasterization to format the text to fit within the boundaries of the shape. Although it was the most engaging project from that quarter, it is two-dimensional. The second project, a Gouraud-shaded teapot display with dithering on a 16-color terminal, demonstrates student familiarity with 3D concepts.



Tom Stepnowski, 1989. DePaul University. From a DEC vt340 terminal

Figure 1

James Snow, 1990, DePaul University. Originally displayed on a DEC vt340

Figure 2

Graphics in the late 90s

In contrast, the computer hardware of the late 1990s has become so cheap that graphics has come into the mainstream of computing. Video cards with 24-bit per pixel capability are common on computers that retail for $1,000 or less. High school students with little or no programming background take pictures with digital cameras, download the images onto disk, use Photoshop for retouching, and post the results on the Web. Students arrive at introductory computer graphics courses with a far greater acquaintance with rudimentary basics than they did ten years ago.

Equally as important, software for three-dimensional graphics is cheap and easily available via the Internet. Today many 3D modeling and rendering packages are available for free or for a modest cost via the Internet [2] and require very little effort to install. Not only are packages widely available, but so are 3D libraries, the most notable of which is OpenGL.

OpenGL

The OpenGL[3] graphics library facilities the creation of interactive programs that can produce animations of three-dimensional objects. It has routines for line, curve, polygon and surface drawing, for viewing, lighting, shading, hidden line-surface removal, and for interaction. Using OpenGL frees a programmer from explicitly handling certain aspects such as clipping and polygon rasterization. When the author

converted a 7800-line rendering package to use OpenGL calls, the program collapsed to 2500 lines even though it contains additional features.

OpenGL is easy to obtain. Originally designed by Silicon Graphics, OpenGL comes bundled with Microsoft's Visual C++. If you are working on a UNIX platform that is not an SGI, you can use Mesa, a freeware library that uses the OpenGL API. Brian Paul developed it and it is available at http://www.ssec.wisc.edu/~brianp/Mesa.html .

Adding to its popularity is the fact that OpenGL is not only a graphics library, it is a software interface to graphics hardware. OpenGL cards, currently available for $250 or less, can execute OpenGL calls directly, which results in a dramatic boost in the graphics performance of a PC.

Possible effects on Curriculum

When comparing OpenGL's list of services to the topics covered in a 1980s-type graphics course, one can see that OpenGL provides routines to carry out many of the algorithms mentioned within the list of topics. This opens up the possibility of spending less time on the more elementary two-dimensional topics and concentrating on the three topics, which most students find more challenging. Spending less time of two-dimension issues also gives an instructor the option to cover more advanced topics such as texture mapping and animation. Student projects become more sophisticated. (See figure 3.)

David Perea, 1998, DePaul University

Figure 3

The acceptance of high-level APIs in the graphics industry raises even more possibilities and questions. Is it desirable to expose students to the tools that will be in prevalent use when they reach the workplace? What portion of employers will be willing to hire graduates to write code when there are $250 graphics cards that already provide equivalent functionality? Does this mean that we want to present some algorithms at a level that allows students to choose the appropriate tool instead of choosing a level that

allows students to implement them? Will this depend on the student's choice of employment or further schooling upon graduation? Are there topics that are technology independent and will be important regardless of the next wave of software and hardware to hit the market?

Syllabus Study Group

In an attempt to clarify some of these issues, Scott Grissom and I sent an email invitation to the Computer Graphics Educators listserve. We invited educators to meet at SIGGRAPH 98 and compare syllabi. (See http://www.education.siggraph.org/docs/cg_list.htm for details about signing up for this mailing list.) As a result, several people volunteered to gather information on current practice in teaching introductory computer graphics to undergraduate computer science majors. Each person solicited syllabi from computer graphics instructors at a variety of institutions across the country. We attempted to spread out our coverage on a regional basis. A list of those who generously contributed to the effort appears in Table 2. In addition to donating their syllabi, instructors answered questions regarding textbook choice, immediate prerequisite courses and the type of computing environment they used.

Results are still coming in while this article is being written, so an in-depth analysis is not possible for this issue, but more will be forthcoming in the February installment. A first look at the syllabi, however, did yield the following:

	low-level API	**high-level API (OpenGL)**
more than 20% on 2D topics	5	4
20% or less on 2D topics	2	12

Within this small sample, it is clear that courses incorporating OpenGL or another high-level graphics API tend to devote less time to two-dimensional topics. In fact, 80 percent of the high-level API courses spend 20 percent or less of their time on these topics, which is a marked change from the late 1980s.

What's next?

For one thing, we'd really like to hear from you – send us your syllabus, tales of your teaching experience, your thoughts about what belongs in the curriculum for this course. If you want to help out with gathering materials for this project, it would be great to hear from you, too. Please send feedback, URLs, and anything else relevant to wolfe@cs.depaul.edu

Thanks

Scott Grissom, Lew Hitchner, Bill Jones, and Susan Reiser have been heavily involved in the gathering and recording of syllabi information, and all of them deserve a warm thank you. Thanks also to Steve Cunningham for discussions regarding *Curriculum 91*.

David Anderson	Purdue University
Gary Bishop	University of North Carolina at Chapel Hill
John Canny	University of California, Berkeley
Dan Frost	University of California, Irvine
Scott Grissom	University of Illinois, Springfield
Donald Hearn	University of Illinois
Lewis Hitchner	California Polytechnic State University
Christoph Hoffmann	Purdue University
Philip Hubbard	Washington University, St. Louis
Maggie Johnson	Stanford University
William B. Jones	California State University, Dominguez Hills
Robert Kenyon	University of Illinois, Chicago
Cary Laxer	Rose-Hulman Institute of Technology
Suzanne Lea	University of North Carolina at Greensboro
Jiang Liu	Bradley University
David McAllister	North Carolina State University
Rich McMullen	Indiana University
Thomas Naps	Lawrence University
Alex Pang	University of California, Santa Cruz
David Salomon	California State University, Northridge
Carlo Sequin	University of California, Berkeley
Clarke Steinback	California State University, Chico
Michael Wainer	Southern Illinois University
Rosalee Wolfe	DePaul University

Table 2: Instructors who contributed syllabi

References

[1] ACM/IEEE-CS Joint Curriculum Task Force. *Computing Curricula 1991*. February, 1991.

[2] Wolfe, 3D Freebies: A Guide to High Quality 3D Software Available via the Internet. *Computer Graphics* Vol. 32 No. 2 (May 1998) 30-33.

[3] OpenGL Architecture Review Board. *OpenGL Programming Guide*. Addison-Wesley, 1993.