Adapting Computer Graphics Curricula to Changes in Graphics

Lewis E. Hitchner
Dept. of Computer Science
California Polytechnic State Univ.
San Luis Obispo, CA U.S.A.
hitchner@csc.calpoly.edu

Henry A. Sowizral
Graphics Division
Sun Microsystems, Inc.
Menlo Park, CA U.S.A.
henry.sowizral@eng.sun.com

Abstract

Introductory computer graphics courses are changing their focus and learning environments. Improvements in hardware and software technology coupled with changes in preparation, interest, and abilities of incoming students are driving the need for curriculum change. Past courses focussed on low- and intermediate-level rendering principles, algorithms, and software development tools. Many of these algorithms have migrated into hardware. Though important knowledge for advanced graphics programmers, most graphics applications programmers have no need to study at this level, much as application programmers have no need to study hardware systems or assembly level programming. Courses need to focus on intermediate- and high-level principles, algorithms, and tools. A fundamental need in modern graphics curricula is integration of a 3D graphics API into the instruction. This paper presents experiences teaching this focus with both low and high level graphics programming API's. The experiences were gained in courses at an undergraduate university and in multi-day industrial courses for experienced professional programmers.
Keywords: *computer graphics, curriculum.*

1. The Changes in Directions of Introductory Computer Graphics Courses

The curriculum of courses for the study of computer graphics has changed and continues to change in response to technology and student needs. We first review graphics technology, students, and curricula, as related to computer graphics instruction during the three stages of acceptance of a new technology (ignoring the outlier stages): that of the "Early Adopters", that of "Early Majority", and that of "Late Majority"[1]. Next a set of principles and goals for a new curriculum for introductory computer courses is presented. We follow this with examples of practical experience in teaching such a course at a polytechnic state university and in multi-day short courses for practising professionals.

1.1. Graphics Technology, Students, and Course Curricula in the "Early Adopters" Era

From the 1970's through early 1980's computer graphics courses existed primarily at large universities. Computer Science education was still in its infancy and courses in computer graphics were rare. They were usually specialty courses at a few schools that could afford expensive graphics hardware, usually funded by government research grants. During that time the curriculum emphasis was on recent developments in research that had not yet entered the public domain or commercial software worlds. Research focussed on developing new techniques, more functionality, and algorithms that performed more efficiently.

Courses taught during this era assumed the following about students and graphics systems:

- students were highly motivated technologists with advanced science and mathematics preparation (computer science, mathematics, and engineering majors),
- most students had little or no experience with the new field of 2D and 3D graphics,
- students should learn all the fundamental principles, algorithms, and techniques that researchers had recently developed,
- graphics hardware devices were not widely available, and they depended upon special purpose software interfaces.

The curriculum of courses during the pioneer days reflected these assumptions. Typical topics included:

- an overview of graphics technology and applications,
- introduction to low level graphics hardware: CRT's, input devices, processors, peripheral devices,
- algorithms for performing basic rendering pipeline operations:
  - line and circle drawing (e.g., Bresenham's Algorithm),
  - line and polygon clipping,

- matrix transformations (primarily for 3D viewing, though also for modeling),
- lighting and shading algorithms, and
- image generation algorithms (polygon scan conversion, visible surface computation, Z-buffer)

- introduction to 3D modeling: parametric curve and surface generation, some procedural techniques,
- software technology for interfacing graphics processors and displays to mini-computers and mainframes using locally developed (one-of-a-kind), or proprietary commercial, software function libraries.

Most graphics textbooks used during this era reflected these assumptions. University courses followed these assumptions and covered topics with the same emphasis as stressed in popular textbooks. During this era there was also somewhat of an elitism. Only computer scientist-technologists with advanced university degrees were capable of developing (or even using) complex graphics display programs.

1.2. Graphics Technology, Students, and Course Curricula in the "Early Majority" Era

By the early 1980's, graphics technology became lower priced and more widely available. Graphics workstations, such as those made by Evans and Sutherland, SGI and other vendors, provided high-powered processors for those schools able to afford them. Graphics peripheral processors (e.g., the AED 512) and early versions of PC graphics systems also became available at somewhat more reasonable prices.

Early attempts at graphics software standardization occurred during this era. Some API's were short lived or never widely used (e.g., the "original" graphics standard, CORE), or were extended and revised as new needs and features evolved (e.g., GKS to GKS-3D, PHIGS to PHIGS+). Such moving targets were not very successful at becoming widely adopted. Proprietary graphics software packages dominated the industry.

Students during this era were somewhat similar to those of the earlier era. However, the popularity of computer science as a career attracted many more students from more educationally diverse backgrounds. Many graphics students during this era had weaker math and science knowledge than earlier students. Fundamental topic areas of graphics (geometry, physics of color and lighting, graphics hardware device operation) were unfamiliar to them, and had to be covered in course curricula. More students had some familiarity with using 2D (and some 3D) graphics in video games.

Although graphics software was more standardized, educators and textbook authors were wary of teaching students the "API du jour". They avoided teaching standards that were likely to be soon out of date. Instead, most courseware developers designed their own generic, "standard-like" software packages to use in accompaniment with a particular text or graphics hardware system. It was also common for a university or individual instructor to write his or her own graphics package (often publicly shared with other universities via anonymous ftp in the pre-WWW days). Unfortunately, these "teaching API's" were often minimal sets of functions. They were sufficient for learning the principals of graphics software development, but were not representative of real-world, industrial-strength packages. And, of course, knowledge of them was not portable to another school or to a job. However, since most student assignments were small, one-shot programs and never lived long enough to get to the maintenance phase of their life cycle, use of a "toy" API was acceptable. That met the goal of providing the necessary learning experience. Of course, after graduation, the employers of these students had to train them to use "real world" graphics software systems.

In response to the availability of and interest in graphics technology, many computer science departments adopted graphics as a part of their curriculum. The ACM Curriculum '91 [2] specified a graphics course as a technical elective. The use and development of graphics applications had now become accessible to any computer, math, engineering, or science student who enrolled in a university graphics course.

1.3. Graphics Technology, Students, and Course Curricula in the "Late Majority" Era

Current, late 1990's graphics hardware is orders of magnitude faster and cheaper, and it is much more robust and fully functioned, than earlier technology. The prevalence of personal computers with low cost, high-performance 3D graphics accelerators, is the dominant technology. However, extreme increases in performance to price ratios have occurred for products at all pricing levels.

Furthermore, graphics hardware technology has evolved to a state where rendering algorithms are

specialized for high performance. Certain aspects of geometry representation, lighting and shading, and interaction handling are primarily implemented within restricted preconditions rather than general purpose ones. For example, most modern graphics rendering hardware either requires all geometry to be represented using triangles only or else is tuned for highest performance using triangles. Also, some current 3D graphics hardware systems implement the entire OpenGL rendering pipeline in hardware.

Nearly as great an increase in performance and functionality has occurred for graphics software. The most significant aspect of software evolution has been the emergence and acceptance of standardized graphics software API's (Application Programmer Interfaces). Today there are several standard 3D graphics API's used in industry. The three leading, nearly universally used "low level" API's are OpenGL (evolved from SGI's proprietary GL and now available on nearly all platforms), Microsoft's Direct3D (Windows platforms), and Apple's QuickDraw 3D (Apple platforms). There are also many high level 3D API's, though currently there is not the dominance by one or two as there is at the low level.

In addition to graphics API's there are four other significant factors in graphics software development:

1. Powerful, low cost IDEs (Integrated Development Environments) for rapid and easy code development and debugging. For example: Microsoft and Imprise (formerly Borland) products.
2. Extensive libraries of general purpose support functions that provide higher level graphics capabilities and simplified GUI development, most of which are either free or very reasonably priced. For example: GLUT (GL Utility Library), MFC (Microsoft Foundation Classes).
3. A number of powerful software packages for high level, application independent development such as ray tracing and animation systems (many of which are public domain). For example: POV Ray.
4. The World Wide Web and its wealth of free and instantly accessible demos, software tools, data sets, and human resources (via email, newsgroups, and chat sessions).

Graphics students of today are quite different from those in earlier years. Some characteristics are:

1. Most technical university students (i.e., computer science, engineering, math, or science students) enter a first course in computer graphics with significant prior exposure to graphics concepts, such as lighting, perspective, and 3D viewing and navigation. Many students have had significant programming experience, and it is not uncommon for them to have already written 2D and 3D graphics programs.
2. Although most students come to school with greater computer skills and graphics experiences than their predecessors, on the other hand, many arrive with greater handicaps. Even among engineering and computer science students, their mathematical, problem solving, and logical reasoning skills appear weaker than in prior years.

In the opinion of these authors, contemporary curricula of most university level graphics courses and textbooks do not appropriately emphasize the most critical aspects of graphics hardware technology, nor have they appropriately adapted to the backgrounds of their students.

It is no longer necessary to introduce today's students to such topics as a pixel, a bit-mapped image, a color palette, or basic RGB color systems. For many, even more advanced topics such as 3D viewing, 3D navigation, and texture mapping are familiar. Avid, even casual, web-surfers or PC gamers of today are familiar with using these terms and features.

Many fundamental algorithms and procedures of graphics are no longer relevant, even though they are pedagogically valuable. For example, Bresenham's line and circle algorithm and the polygon scan conversion algorithm are not relevant. Such operations are done in very low level hardware (e.g., microcode or ASIC's) and even there, traditional algorithms are not always used. Clipping operations are also buried deep within the hardware. Cohen-Sutherland line clipping and re-entrant polygon clipping algorithms are good intellectual exercises for today's students, but have less relevance to helping them learn useful graphics software development techniques.

Now that small set of graphics API's are nearly universally used (and also becoming more similar), students should learn to apply the fundamental principles of graphics within these environments, not within make-believe, "toy" software environments. A few recent textbooks integrate modern API's, and one of them [3] has been adopted by a number of universities. However, to our knowledge no textbooks fully meet the criteria of presenting

graphics technology in the context of today's hardware and software systems.

Regardless of criticisms of graphics education weaknesses, the accessibility and usability of computer graphics technology today is truly phenomenal. No longer are those who study and use graphics members of an elite club. Nor are they restricted to university engineering and science students. We now have experienced the "democratization" of computer graphics.

2. New Models for Introductory Graphics Courses: SIGCSE and SIGGRAPH Panels

In response to the need for modernizing the curricula of graphics instruction, a panel at the recent ACM SIGCSE (Assoc. for Computing Machinery, SIG Computer Science Education) Technical Symposium [4] chaired by one of this paper's co-authors, discussed and presented a proposal for contemporary computer graphics curricula. In addition to the panel, a Birds-of-the-Feather session co-sponsored by the SIGGRAPH Education Comm. at the same conference held discussions following the panel. A similar panel with three of the four same panelists has been submitted and accepted for presentation at the ACM SIGGRAPH '99 conference (Los Angeles, August 1999).
This panel proposed the following philosophy of the first graphics course:

- Computer graphics is inherently 3D and courses should be also.
- The fundamental subject of a computer graphics course is geometry and how it is expressed in computational terms. Thus, geometry is a major part of the introductory course. Geometry is expressed in terms appropriate to the field, such as coordinate systems, transformations, and surface normals. The basic shape is the triangle.
- Computer graphics is intrinsically visual, and even the most technically oriented graphics practitioner must be aware of the visual effects of algorithms. Unlike other areas of computer science, algorithms must be considered not only for time and memory usage, but also for their visual effect.
- Besides geometry, computer graphics is about light and surfaces, and about developing algorithms to simulate their interplay. Courses need to include material about light and surface properties and about the distinction between the ways various algorithms present light and surfaces visually.
- Computer graphics has matured to a state in which there are a small number of high-level API's that support all the fundamental concepts needed for early work. Courses should be built upon this kind of high-level approach.
- Computer graphics should be interactive. Courses should include interactive projects and cover event-driven programming.

3. Experiences in Teaching Introductory Graphics Courses

One co-author has taught undergraduate university graphics courses from 1984-1988 at the Univ. of Calif., Santa Cruz, and from 1995 to present at California Polytechnic State University (Cal Poly). The syllabus and topics emphasis of courses has shifted significantly.

The original syllabi were similar to that of the "Early Majority" era model, based upon the same assumptions about available graphics technology and students as stated above. The topic focus was on fundamentals (hardware devices, geometry, viewing, lighting, and shading) and algorithms for implementing the rendering pipeline. Interaction methods were limited to fairly simple event handling, and they used console text i/o without a GUI. Some hierarchical modeling was taught using modeling transforms implemented by the programmer. Early versions of the course used a "home grown" API (implemented on top of a device dependent 2D graphics API). Later versions migrated to proprietary versions of GL (SGI and HP). Typical programming lab assignments included implementation of Bresenham's line drawing, Cohen-Sutherland line clipping, re-entrant polygon clipping, and simple hierarchical modeling applications.
The 1996-98 Cal Poly graphics courses use modern API's. The co-author teaches a version that uses OpenGL for the first half of the course, and uses Open Inventor for the second half [5]. In this course the topic focus is also on fundamentals. But, instruction in implementation techniques has shifted to higher level aspects of rendering, event-driven interaction handling with GUI's, complex hierarchical modeling including extensive study of scene graphs, and software design using the two API's. The first lab assignment draws complex 3D primitive shapes with interactive control of color, orientation, and line style. A second lab assignment draws chairs, a table, and a floor and requires interactive control of position and orientation for

each object and for the viewpoint. A third lab uses Open Inventor to model a 16-jointed robot with interactive control for joint manipulation. This lab also requires model development using a graphical scene graph editor without writing program code.

4. The Future: Java 3D as a Learning Environment for Introductory Graphics Courses

1.4. Overview of Java 3D Design Philosophy and Features

Java 3D is an API used for writing 3D graphics applications and applets. It is a library of basic and utility classes written in the Java language. Java 3D is platform independent and extends Java's "write once, run anywhere" benefits for application developers. It also integrates well with the Internet because applets written using Java 3D have access to the entire set of Java classes. A complete description of the Java 3D design philosophy and its features is available in published texts and online web documents [**6,7,8**].

The Java 3D programming paradigm includes these principles:

- Fully object-oriented implementation
- Classes are extensible and compatible with all other Java 2 Platform libraries
- Scene Graph based for both geometry and behaviors
- High performance a primary design and implementation philosophy:
  - Layered Implementation: Native code based, aiming at hardware acceleration,
  - Application programmer can specify what will change so that system can perform optimization,
  - Supports multiple rendering Modes: Immediate, Retained, Compiled-retained

1.5. Justification for Java 3D As A Learning Environment for Beginning Students

Although at first glance Java 3D may seem like an API most applicable for experienced professionals developing production quality software, several aspects make it desirable as an API for beginners.

- **Platform independence**
  Students usually prefer to work in labs at school, on their home computers, and, even at their place of work (for those employed full or part-time). Java and Java 3D's platform independence greatly simplifies portability, not only for 3D graphics but also for 2D GUI's.
- **Cost**
  The purchase price is zero, and there is no software maintenance cost. This is a significant factor for schools, as well as for students.
- **Programming Paradigm**
  The fully object-oriented environment is consistent with training students receive in their prerequisite courses. There is no need to kludge together OO with non-OO procedures.
- **High Performance**
  Students are accustomed to high performance -- or at least what they perceive to be high performance. The many PC games and Web applets that present apparent high performance rendering raise expectations. However, they can cause frustration for students if they are limited to programming lab examples that appear simplistic by comparison. A significant aspect of graphics education is the motivation provided by its appealing real-time, interactive, visual results. If that motivation is frustrated by low performance, the learning that occurs will be diminished.

5. Experiences in Adapting Course Curricula to Changes in Graphics Technology

We have put our recommendations into practice. In Spring 1999 one co-author taught a revised version of the Cal Poly undergraduate Introduction to Graphics course using OpenGL and Java 3D APIs. During Winter and Spring 1999 the other co-author taught three courses, varying in duration from one to three days, on Introduction to Java 3D for experienced professionals.

In the current university environment, students are entering graphics courses with less foundational knowledge than in the past. Although most students come to school with much greater exposure to computers and graphics, they have less exposure to mathematics, problem solving, and less of an idea where they want to focus their careers. Furthermore, the field of graphics has expanded greatly since its inception, and given the new APIs, computer graphics is no longer the exclusive province of experts. The challenge for educators is to provide enough information so students can learn graphics without requiring them to becomes masters of a particular sub-discipline. After all, twelve-year-olds are producing beautifully ray-traced images without having any concept of how ray tracing works.

A high level graphics API provides educators with the ability to introduce a broad range of fundamental concepts without detouring into the details important only to graphics professionals. Despite the higher level presentation of concepts, students can quickly learn how to write useful and compelling graphic applications. In the process, they are exposed to enough graphics concepts that they can decide whether to devote more of their student and professional career to graphics.

In the Cal Poly CSC 471 Introduction to Graphics Spring 1999 course [9] students used OpenGL and Java 3D APIs, a low level and high level API respectively. These APIs allow topics to be covered quickly in both breadth and depth. Programming assignments using a high level API (Java 3D) allow students to produce quite substantial results within a short period of time. In one programming assignment students wrote a Java 3D program to display a humanoid-like robot with 16 rotational joints and interactive behaviors that allow run-time modification of the joint angles. This project was completed within 3 weeks of their introduction to Java 3D (and, for some students, within 3 weeks of their first exposure to Java after having been trained in C++). At the same time students were learning the fundamental concepts of a scene graph and were solidifying their knowledge of complex transform operations, they also learned the details of Java 3D scene graph implementation.

Experienced computer professionals have significantly more exposure to mathematics, problem solving and logical thinking. However, when they decide to study graphics, they often revert to "student mode". Presentation order, thus, remains important. Fundamental material must be introduced in a constructive order rather than deconstructive order. Otherwise students become confused and frustrated. Although experienced professionals may be able to solve the most complex problems without hand holding -- outside the classroom -- when they become students they require the smallest details to be completely specified. This is not surprising, since much like university students they also cannot distinguish between fundamental concepts and unimportant details.

This becomes more of an issue given the breadth and depth of topics within the graphics field today and the short time frame of a typical professional short course. Despite this challenge, higher level APIs allow educators to introduce breadth as well as to delve into selected topics in depth within short time frame courses. By emphasizing fundamental ideas an educator can start such students in the necessary direction and leave them with sufficient landmarks to allow the students to explore the details further on their own.

6. Summary and Conclusions

Graphics technology has changed vastly in the nearly three decades since researcher-educators first started passing their knowledge on to students. Students and computing environments have also changed significantly. Course curricula and textbooks have been slow to adapt and have not kept up with these changes. Proposals for adapting curricula to match modern technological requirements have been presented here and elsewhere. Personal experience with such adaptation in university courses and in industrial short courses has demonstrated the effectiveness of using modern, high-level API's and emphasis upon student-centered learning.

References:

[1] Geoffrey A. Moore, *Crossing the Chasm*, HarperBusiness, 1991.

[2] Alan B. Tucker and Bruce H. Barnes, editors, *Computing Curricula 1991: Report of the ACM/IEEE/CS Curriculum Task Force*, ACM Press/IEEE Computer Society Press, 1991.

[3] Edward Angel, *Interactive Computer Graphics: A top-down approach with OpenGL*, Addison-Wesley Longman, 1997.

[4] Lewis Hitchner, Steve Cunningham, Scott Grissom, and Rosalee Wolfe, *Computer Graphics: The Introductory Course Grows Up,* Panel session, Proceedings of the 30th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '99), New Orleans, LA, USA, March 24-26, 1999.

[5] Lewis E. Hitchner, CSC 455 course web page, Spring 1998, http://www.csc.calpoly.edu/~hitchner/CSC455.S98

[6] Henry Sowizral, Kevin Rushforth, and Michael Deering, *The Java 3D API Specification*, Addison-Wesley Longman, 1995.

[7] Henry Sowizral, Kevin Rushforth, and Michael Deering, *The Java 3D API Specification* http://java.sun.com/products/java-media/3D/forDevelopers/j3dguide/j3dTOC.doc.html

[8] *Java 3D White Paper*, http://java.sun.com/marketing/collateral/3d_api.html

[9] Lewis E. Hitchner, CSC 471 course web page, Spring 1999, http://www.csc.calpoly.edu/~hitchner/CSC471