

# Powers of 10: The Case for Changing the First Course in Computer Graphics

Steve Cunningham\*  
Computer Science Department  
California State University Stanislaus  
Turlock, CA 95382  
rsc@eos.csustan.edu

## Abstract

The growing maturity of computer graphics technology now makes it possible to view the introductory graphics course in a general computer science curriculum in a new light. Instead of requiring highly specialized techniques and a great deal of mathematics before a student can produce significant work, the course can now be built around generally-accepted standard graphics standard APIs. This opens the door to making computer graphics available to a wider audience and moves the introductory computer graphics course in exciting new directions.

## 1 Introduction

During a panel on introductory computer graphics courses at the SIGGRAPH 99 conference, the author spoke about a high-level, API-based approach to the introductory course that is generally in line with the directions recommended at the 1999 Graphics and Visualization Education workshop in Coimbra, Portugal [7] [5] and with directions described elsewhere [6]. After his presentation, he was challenged by a long-time member of the computer graphics community as to whether he was really teaching computer graphics or was simply teaching computer graphics users. In this paper the author will describe the new approach and outline the case that this approach is the appropriate way to introduce computer graphics in the undergraduate computer science curriculum, and that the original course is now a more specialized course for students who want to focus in the subject and can be taught appropriately following the new first course.

---

\* For 1999-2000, the author's address is  
San Diego Supercomputer Center  
P.O. Box 85608  
San Diego, CA 92186  
rsc@sdsc.edu

The basis for the new approach to the subject of computer graphics is the current and evolving environment of graphics APIs. OpenGL and its successors have supplanted earlier standards such as GKS and PHIGS to allow students and professionals to do significant graphical work without the need to manage basic algorithms and techniques. In addition, a new generation of inexpensive, high-powered graphics boards with built-in support for OpenGL now form the hardware side of the environment. Together, these form the basis on which a new course can be built that will open the doors to many more people than the older course.

It is important to realize that this idea is not original with this author. As I tried to find ways to restructure my course, I had conversations with several people who use these ideas in one way or another. The ideas were also well understood by the participants in the Coimbra workshop mentioned above. But the small amount of literature on the curriculum in the first graphics course, together with an approach to the introductory course that focuses on science students and that articulates with a second course with more traditional goals, makes it important to put these ideas before the computer science education community.

## 2 Two Kinds of Courses

The new computer graphics course might be called "Computer Graphics Programming." It takes the viewpoint that the most critical part of computer graphics is thinking and problem-solving geometrically, and that students who have accomplished this should be able to learn to use generally-available tools to write interactive programs that create images and animations that are both visually interesting and have meaning for real problems. This can be accomplished by emphasizing geometric and spatial problem representation in class discussions, by using a graphics API such as OpenGL [8] as the basis for student projects and taking advantage of the capabilities such an API can provide, and by keeping the discussion of graphics technology at a level that can help students understand the concepts that underlie the images without focusing on the details of how these concepts function.

In contrast to this, the traditional computer graphics course might be called "Computer Graphics Algorithms and Techniques." It emphasizes the fundamental algorithms and techniques needed to manage the geometry and rendering directly. In this course, students focus on many specific techniques, including (to name only a few): how to

create transformation matrices and how to manage them to create hierarchical models, how to create scan conversions of graphical primitives to set individual pixels in a raster, and how to interpolate pixels across a polygon to handle shading or texturing. These are certainly worthy techniques, and only begin to describe the many specialized processes needed to create the wide range of special techniques that are possible in graphics.

There is also a hybrid approach, illustrated by the well-known Angel text [2]. In this approach, the main subject of the course is fundamental algorithms and techniques, but the course also includes OpenGL programming and the fundamental principles are expressed using the OpenGL API to carry out class projects. This makes the topic more approachable by the computer science student, but continues to focus on the underlying techniques instead of geometric problem-solving.

Comparing these course approaches, then, we see that the traditional approach emphasizes the details in image creation and challenges the student to understand and implement these details, while a graphics programming approach hides details in favor of teaching the student how to use computer graphics to take geometric or spatial approaches to problem solving and to create images that represent the problem and its solution. This contrasts a focus on computational techniques with a focus on the value of the images as communication and problem-solving tools.

The traditional approach formed the basis for the computer graphics recommendations in Computing Curriculum 91 [9], while a combination of the new first course and an advanced techniques course forms the basis for the author's recommendations for Computing Curriculum 2001 [4]: an introduction with the API-based geometric problem-solving approach, followed by a more detailed course based on the fundamental techniques approach. The content of the second course is not discussed here but is a straightforward implementation of the traditional course. It would present fundamental modeling and rendering techniques, using the fact that the students already have an understanding of modeling and rendering concepts and can move more quickly and fluently through the details of the many appropriate fundamental algorithms. During the discussion after the SIGGRAPH 99 presentation that was alluded to above, it was generally accepted that students who complete the new two-course sequence would have approximately the same depth of knowledge of the technical content as those who now complete the standard two-course sequence.

### 3 Powers of 10

There is only so much time in undergraduate studies to cover any material, and few of us have the luxury to offer specialized courses to limited audiences. Changing the content of the introductory computer graphics course will allow this course to serve a wider audience, as well as keeping the course consistent with standard practice in the field.

There are many persons who need some knowledge about computer graphics. These people form three basic groups: the group who create fundamental graphics systems, the group who use develop applications and tools using graphics systems, and the group who use graphics applications and tools at a sufficiently sophisticated level that they are well served if they have a basic working knowledge of how graphics works. This list omits people who simply use tools such as modelers or renderers without any real concern about how they function, because such persons are generally not interested in programming or other aspects of a computer science program.

We believe that the "power of 10" concept may roughly describe these three groups, whose sizes may be estimated by anecdotal and intuitive means. The number of people who create fundamental graphics systems is relatively small, consisting of persons doing graphics research and development in academic or industry settings. The number who develop applications and tools using graphics systems includes persons writing visualization, graphical application, and visual communication systems, and a quick "Delphi" estimate of the numbers of persons so engaged is at least an order of magnitude larger than the first group. Finally, the number of persons in the third group, those who use graphics tools and applications at a sufficiently deep level to benefit from knowing graphics programming, is again at least an order of magnitude larger than the second group. This is shown in Figure 1, with the system developer group at the bottom and the high-level graphics users at the top.

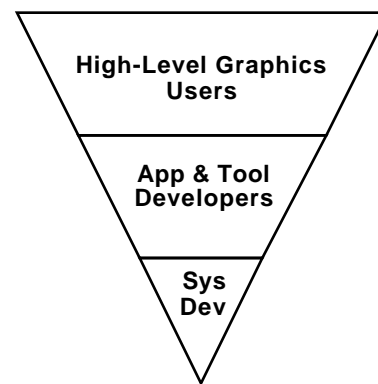


Figure 1: Relative sizes of possible audiences

So whom do we serve with the first computer graphics course in computer science? The traditional first course clearly served the system development group best, while providing some service to the application and tool development group, but left out the group of high-level graphics users entirely. The new approach to the first course described in this paper focuses on the application and tool developers group and has a great deal of value for high-level graphics users while not disenfranchising the system developers group; as noted above, with an appropriate second graphics course, students who go through the proposed first course can quickly learn the original fundamental concepts in the second. Thus the

1. Getting started: sample programs in the graphics API: reading, executing, and modifying simple programs that show examples of window management and very simple geometries
2. The graphics pipeline: the fundamental nature of graphics programming and how the components work together
3. Fundamentals of color and shading, including alpha blending
4. Event-driven programming: registering and using callbacks, use of keyboard as a simple motion controller
5. Simple modeling: built-in geometric primitives and how they can be used to create simple images
6. Scene modeling: adding a viewer, surface and material properties, hidden-surface elimination, and lights
7. Scene graphs done simply: organizing scenes so they can be processed readily
8. Simple animations by using time-varying modeling and eye motion
9. More on interaction : menus and virtual devices, adding simple controls with a light interface toolkit, evaluating interface alternatives
10. Additional rendering techniques: clipping planes, texture maps
11. Hierarchical modeling techniques
12. Evaluators and smooth surfaces

Figure 2: One Possible Outline of Introductory Course

approach described here serves a new audience and does not harm the original primary target audience for computer graphics in computer science.

#### 4 Details on the New Introductory Course

While the brief description of the API-based introductory course above gives an idea of the goals of the course, more details are needed to describe the full nature of this approach. An important point in the new approach is that when we do not require students to master the details of graphical algorithms and processes, we are able to lower the prerequisites to the course while still covering the key concepts of graphics through the API. Students no longer need to be skilled at the details of data structures and linear algebra; a solid CS2-level programming background and an instructor's informal and often intuitive descriptions of the graphics concepts and techniques should be quite enough. Thus the graphics course could move to the second year of a student's studies instead of its traditional third or fourth year timing, and could also be accessible to students with programming skills who are not specializing in computer science.

One of the most important points about a geometric problem-solving approach to computer graphics is that it allows the student and instructor to address directly the real reason graphics is valuable: it allows us to create visual communication between persons with the support of the data manipulation and communication capabilities of modern computers. Many of the course presentations start with questions about visual communications, such as "How would you communicate a smooth surface?" or "How would you communicate the shape of an object?" With this communication goal in mind, we can proceed to talk about how to represent things geometrically or visually, and then discuss the corresponding graphics processes: the modeling needed to represent that geometry or image for the graphics system, representing the viewing environment for the model, rendering that model to create an image, and interpreting that image in terms of the communication that was intended. This underlying concept can thread throughout the course and can motivate us to re-

visit earlier work when we have a more extensive understanding of how we can express ideas and create images. For example, in thinking about surfaces, we might start with a simple mesh-based surface and revisit the question when we have evaluators to support the creation of Bézier surfaces.

Adopting a useful API for the graphics course supports a broader concept of graphical communication than the traditional approach. Static images do not always communicate concepts or information fully, so we can use the interactive and animation capabilities provided by an API to create more interesting ways to communicate visually. The event management capabilities in APIs make it very easy for a student to introduce animation and interaction to a project. Thus an image might move on its own, based on dynamic issues in the problem being considered, or a student might create an image that could be moved around to see it from all sides. This not only improves visual communication but also gives the student an introduction to event-driven programming and callbacks that seems to be lacking in the usual computer science curriculum.

This idea readily goes further. A course based on a full-function API also includes opportunities for students to get user interface experience. Standard and straightforward mouse, keyboard, and menu callbacks let students compare these traditional modes of program interaction. Going one step further by using light user interface toolkits such as MUI for OpenGL also gives students a chance to develop experience with a broader kind of interface development, and possibly even with formal evaluations of alternate interfaces.

Instead of just talking about the course in general terms, let's look at one possible outline of the course. The outline is generally formed along the ideas of the graphics pipeline, but includes appropriate additional topics supported by the API used in the course, such as event-driven programming and the use of callbacks. This outline is given in Figure 2 above. Note that the outline introduces event-driven programming and simple interaction early so they can be included in later course projects. Of course, this outline is very sketchy, because the goal of this paper is

less to present the details of the course than to argue for the top-down high-level approach to the course. Details of the course will be presented after more development has been done.

While curriculum discussions seem to focus on lectures and course outlines, students experience a course very largely through the course projects. With the visual communication emphasis, we can shape projects to cover topics with real meaning for the students. For example, if we have science students in the computer graphics course, we can structure a project to focus on understanding and communicating in his or her science area. This starts by examining a problem in the science, getting geometry or spatial information from that problem, creating an image from that information, and asking the student to interpret the image back to the science. This is illustrated below in Figure 3, where the computer graphics project itself is the bottom row of the diagram.

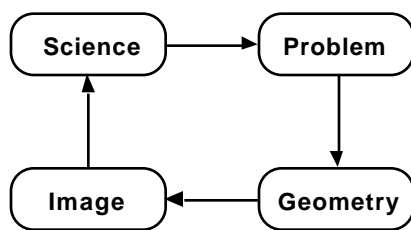


Figure 3: a course project's context

This orientation is the direction the author intends to take with his introductory graphics course, and developing that approach is underway in a project supported by the National Science Foundation. The new course concept will place an emphasis on science-focused projects and visual communication, which will make it a particularly valuable course for science students [3] and allow it to become a key course in computational science programs. Other kinds of projects could give the course other orientations that would be appropriate to the local conditions or philosophy of another institution. But the instructor must be on the lookout for unique ways to add visual interest to these problems. For example, a project might include creating and viewing stereo pairs instead of simply creating single images. Alternately a project could use Chromadepth™ color depth encoding to display spatial content. These depth techniques catch student interest and reinforces the true 3D nature of the graphics.

## 5 The Future

At this time, OpenGL is considered to be the assembly language of computer graphics [1] and is the foundation on which other graphics systems or APIs are being built. We really cannot consider an OpenGL approach to the introductory computer graphics course to be especially high-level; other higher-level APIs such as Java3D are available or are in development, and computer graphics instructors need to keep the course planning flexible to respond to continuing developments in the field.

An alternative to the OpenGL/Java3D kind of graphics programming is offered by some of the more powerful graphics systems, such as Open Inventor. The author is not yet comfortable with the idea of moving farther away from graphics programming based on familiar primitives, but others may find the power of such systems compelling. In any case, a focus on programming instead of more indirect approaches seems appropriate for a graphics course in the computer science curriculum.

## References

- [1] Akeley, C. Chief Technology Officer, SGI; personal communication
- [2] Angel, E. *Interactive Computer Graphics: a Top-Down Approach with OpenGL*, Addison-Wesley, 1997
- [3] Brown, J.R., Cunningham, S., and McGrath, M. "Visualization in Science and Engineering Education," in *IEEE Tutorial: Scientific Visualization*, Nielson, Gregory M. and Bruce Shriver, eds., IEEE Computer Society, 1990
- [4] Cunningham, S. "Outside the Box — The Changing Shape of the Computing World," invited editorial, *SIGCSE Bulletin* 30(4), December 1998, 4a-7a
- [5] Cunningham, S. "GVE '99: Report of the 1999 Eurographics/SIGGRAPH Workshop on Graphics and Visualization Education," to appear in *Computer Graphics and Computer Graphics Forum*
- [6] Cunningham, S. "Re-Inventing the Introductory Computer Graphics Course: Providing Tools for a Wider Audience," to appear in *Computers and Graphics* 24(2), April 2000
- [7] Graphics and Visualization Education '99; papers and reports on this workshop are online at [www.eg.org/WorkingGroups/GVE/GVE99](http://www.eg.org/WorkingGroups/GVE/GVE99) and [www.education.siggraph.org/conferences/GVE99](http://www.education.siggraph.org/conferences/GVE99)
- [8] Kempf, R. and Frazier, C., eds., *OpenGL Reference Manual*, second edition, Addison-Wesley, 1997
- [9] Tucker, A.B. and Barnes, B.H., eds., *Computing Curricula 1991: Report of the ACM/IEEE/CS Curriculum Task Force*, ACM Press/IEEE Computer Society Press, 1991

## Credits

This work is partially supported by National Science Foundation grant DUE-9950121. All opinions, findings, conclusions, and recommendations in this work are those of the author and do not necessarily reflect the views of the National Science Foundation. The author would also like to thank the San Diego Supercomputer Center (SDSC) and particularly Mike Bailey of SDSC for supporting his work.