Re-Inventing the Introductory Computer Graphics Course: Providing Tools for a Wider Audience

Steve Cunningham
Computer Science
California State University Stanislaus
Turlock, CA 95382 USA
rsc@csustan.edu

Abstract

In ACM/IEEE Curriculum 91 [TUC 91] and in earlier curriculum statements on computer science, the computer graphics course in computer science focused on fundamental algorithms and techniques for creating images and animations. Forthcoming curriculum statements are going to find a need to expand the role of courses such as computer graphics in the undergraduate computer science curriculum [CUN 98]. In the last few years the approach to teaching this course has changed to take advantage of more powerful graphics tools. By using these tools, however, we are also able to make the graphics class accessible to students from other areas. This paper describes a set of presentations, examples, and projects that supports the introductory computer graphics course as a tool for the student in the sciences, mathematics, or engineering as well as for the computer science student.

1. Introduction

Computer graphics is an important tool for the working scientist, mathematician, and engineer, and has become a commodity item in the sciences just as it has in so many other areas of life. Its value is expanding as the scientific culture embraces scientific visualization and as powerful computers become more and more commonly available. However, the traditional computer graphics course focuses on fundamental graphical algorithms and processes, not high-level graphics programming. This is oriented to the computer science student who wants to pursue advanced work in the field, not the more general student in science or engineering.

This offers computer science an opportunity to serve this wider audience by creating an introductory computer graphics course that has dual goals. One goal is to introduce computer science students to basic geometric processes and to graphics programming, and the other is to give students in science, mathematics, and engineering a basic background in thinking and programming in 3D geometric terms.

2. The Course Philosophy and Outline

One fundamental goal of the revised introductory computer graphics course sets it apart from the traditional course. This goal is to create a sound course for science, mathematics, and engineering students (hereafter abbreviated as "science") that will give them the graphics and visualization tools that they need for their future [BRO 90]. In doing this, we must avoid creating a course that poses significant programming challenges for students, because many students from these fields will have less programming experience than a computer science student would. Thus the course must provide a sound graphics experience for a student who has had only a two-semester sequence in programming.

At the same time, another course goal is to provide computer science students with an introduction to the field of computer graphics. Thus the course must be useful to the general computer science student and provide him or her sound skills in graphics programming, because many applications will need to add graphical content in the future.

All the work in the course is done entirely in 3D, with no reference at all to 2D grapics except perhaps in a few specific examples. This supports both goals for the course and is consistent with current directions in API development and in graphics applications. The heterogeneous nature of the class does offer a challenge, but the expected benefits should outweigh this problem. Certainly computer science students should benefit from working with others outside our field, and science students should benefit from seeing a computer science approach that supports their fields.

The course outline is straightforward. It begins with an introduction to the fundamental notions of geometry as used by

OpenGL, and with simple examples of OpenGL programs to help students see how to build their first programming projects. It goes on to cover fundamentals of polygon-based modeling, including the use of instancing transformations; to discuss rendering including projections, viewing, lighting, and shading; and to work with event-handling to provide user controls and animation. It ends with some of the features that students rarely actually use in a traditional fundamentals course, such as alpha blending, texture maps, and fog.

This course is largely possible because of advances in low-cost and high-function graphics systems. Computers with these capabilities are now available for even budget-strapped universities. Graphics programming APIs such as OpenGL and Java3D are readily useable by students who have a programming background but are not computing specialists. Further tools such as the Fahrenheit API being developed by SGI and Microsoft [FAR 98] will also soon be widely available.

3. The Presentation Sequence and Examples

In order to introduce students to the subject, we begin with geometric and programming fundamentals and proceed to more sophisticated topics in modeling and rendering. The list below summarizes the contents, but it should not be taken as an indication that we proceed in a linear way through this material. Instead we do a little modeling, then a little rendering, then come back to more modeling, then more rendering, and so on until we have covered all this content in one semester.

In order to be valuable for the broad audience we noted above, the course material needs to be backed up with a collection of examples and projects that come from an equally broad set of areas. We need some examples and projects from mathematics, some from physics, some from chemistry, and some from various branches of engineering. Creating a set of materials with this broad coverage will take some time, and the author has begun to work on such a project. These materials will be developed and tested in collaboration with other faculty having many different kinds of experience and with students from other disciplines. The goal is to have a draft set of materials available by mid-2000 and a completed set a year from then. The materials will be distributed on the Web and will continue to evolve as our understanding of the needs of science and engineering students grows.

An outline of the overall set of topics covered is:

I. Introduction to geometry and to graphics programming
   a. Coordinate systems and geometry fundamentals in 3-space. Coordinates, normals, vectors, dot and cross products, polygons, etc.
   b. What is an OpenGL program? Example or two of some simple code in template programs to introduce the idea of using the OpenGL system.

II. Modeling
   a. Modeling primitives. Points, lines, triangles, quads, triangle strips, triangle fans, quad strips, etc. Using built-in GLUT primitives; modeling using them. Frames as transformed modeling coordinate systems.
   b. Modeling objects using transformations. Creating items for a scene by starting with primitive objects and using scaling, rotation, and translation to shape them and place them in a scene. Handling transformation stacks. How the stack of modeling transformations is used to handle several objects. LIFO behavior of modeling transformations and the distinction between function composition and stack behavior.
   c. Evaluators and Bézier splines. Defining control points; specifying evaluators; using evaluator to generate points in the line or surface.
   d. Scene graphs. Concept, examples, applications

III. Rendering
   a. Projections and viewing. Orthogonal and perspective projections, standard view volume. Setting the viewpoint; relation to both kinds of projections
   b. Hidden surfaces. Enabling and specifying; what effects do you get
   c. Lighting and materials. Lights in a scene; ambient, diffuse, and specular lighting. More than one light. Colored lights. RGB and other color models. How materials are specified; application to objects in a scene.

d. Shading. Specifying shading; use of normal vectors. The difference between flat and Gouraud shading.

IV. Interaction

a. Event and event handling concepts. Event queues, event types, concept of callbacks, registering callbacks. Keyboard, special keys, menu, mouse, idle

V. Additional features

a. Color blending (alpha channel). Specifying, effects of blends
b. Fog. Enabling; effect of fog color; effect of density; specifying front and back
c. Textures. Creating texture maps, specifying maps, 2D and 3D texture maps
d. Text as a graphic element. Labels, in-space menus, billboards.

As each part of this material is presented to the class, the presentation first focuses on what the feature means in creating or interacting with an image by presenting demonstrations of the feature's effect on an image and pointing out how the feature is invoked in code by showing code examples and experimenting with parameters. When the student has some idea of what the feature is, the presentation will go on to give some background on what the feature means in terms of the actual graphics system work, and on occasion to include a brief description of the algorithms involved. To put these in balance, if we consider the OpenGL Programmers Guide [WOO 99] on one hand and a standard textbook that uses OpenGL on the other [ANG 97], the presentation is in based 80% on the programmers guide and 20% on the textbook.

4. The Projects

One key to making this course effective for students in science, mathematics, and engineering is creating a good set of projects that are meaningful to the student. We cannot use a single set of projects or projects that ask a student to create good-looking but meaningless images. The author is working on a set of projects that will fit this orientation for the course, and hopes to have a set ready to distribute in mid-2000. Some initial projects that were used in the first instance of the course included:

a) 3D histograms that can be rotated around one axis; only ambient light
b) physical object with one light source, where both object and light source can be rotated
c) pseudo-fractal landscape with semi-transparent water and both snow and tree lines
d) N-body gravity problem with each body showing a short-term trace of its recent location, with time-step animation
e) a Bézier surface with selectable and moveable control points

All of these projects included user-controlled motions or some form of animation, and some examples of student work on these projects is shown below. Because we had not yet done any promotion of this course to students outside computer science, we did not have any student input into interesting project topics for non-computer science students. This will be one approach to developing such projects in future course offerings.

The language used for projects, either C or C++, is not an issue in the course or projects. It is possible to call any of the OpenGL functions through either language. Thus students can choose the language they are most familiar with and use whatever level of the language one can handle. In the future we expect to consider offering the course with Java3D or perhaps Fahrenheit, depending on the availability of the systems and the amount of programming background needed to use them. A relatively modest independence on the programming language is a very important part of making the course work for non-computer science students and cannot be overemphasized: students can create interesting and useful images without using sophisticated programming techniques.

5. The Sequel to This Course

At the end of the introductory computer graphics course, students have experience in thinking geometrically and in creating programs that express that geometric thinking and use some modest animation and interaction. They have seen a very modest description of how some graphics algorithms are used to create images, but have done no work directly with those algorithms. This gives my computer science students an excellent background for tackling a second graphics course containing the traditional fundamental algorithms content.

In the second course, then, we include familiar fundamental techniques. These include interpolation material such as scan-converting

polygons, shading models, Z-buffering, bump and environment mapping, as well as techniques such as antialiasing and ray tracing. We have found that straightforward modifications to the Bresenham algorithm [CUN 99] provide a general and straightforward approach to the whole range of interpolation algorithms on polygons, and between this approach and the experience students get from the first course, they are able to develop a good understanding of advanced graphics techniques.

6. Results so Far

The author has taught this course and its sequel only once, so we cannot give a good evaluation of this work. My computer science students are enthusiastic about this approach to the course and several took the fundamental graphics principles course that was taught as a second course in sequence. The results of the second course showed that students did learn many key concepts in the subject from the first course, and we were able to move quite quickly through material that had been fairly painful.

The value of the traditional computer graphics principles course does not seem to be reduced by beginning computer graphics studies with this new kind of course. Instead the new first course makes a very good introduction to the principles course by providing students with a background in realizing images from geometric concepts. Starting with the programming course allows a traditional principles course to move more quickly and to address more advanced concepts than a course that assumes no graphics background. The combination of a first course based on graphics API programming and a fundamental principles course seems to be an excellent way to introduce computer graphics content for computer science students. More advanced courses can be developed as needed when additional topics need to be covered.

In order to understand the level of this graphics programming approach, it is useful to see the kind of work done by students in the new first course. Below are some images from the last three of the course's five projects. Each project also involves animation and/or user interaction in addition to the image synthesis.

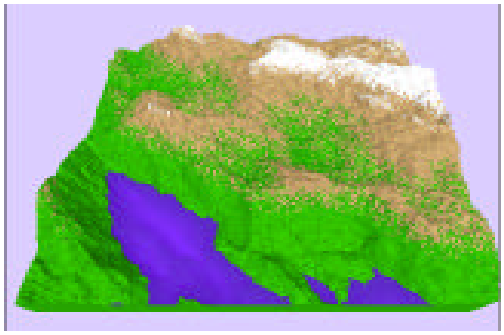

Figure 1. pseudo-fractal landscape with alpha blending in water

Figure 2. n-body problem with fade-out trails on bodies

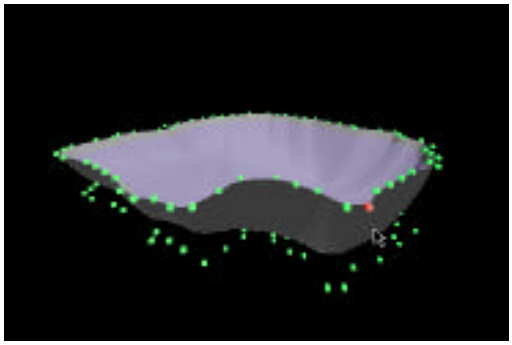

Figure 3. Bézier spline surface with control points shown; one is selected for movement

In the first example, students use keyboard callbacks to move around the landscape and see it from different viewpoints, and use mouse callbacks to work with a menu to change various features of the landscape. In the second example, the bodies move in real time driven by the idle callback, using simulated gravitational forces between the multiple bodies. In the third example, students move around the shape with keyboard control and can select and manipulate a

control point with the mouse, as shown; the shape of the surface changes as the point is moved with keyboard control.

7. Summary

One of the challenges of computer science is to find ways to serve our universities and still maintain the quality of our courses and programs. Because computer graphics is so useful to so many different fields, it is a natural course to take on a service role, but because the course has traditionally made significant mathematical and programming demands on its students, it has rarely been used in this way.

The approach outlined in this note offers us a way to provide a course that benefits both the computer science student and the student from mathematics, science, or engineering. It is accessible to faculty, is useful to computer science students, opens the door to studies of more advanced computer graphics topics, and is a valuable service to other students whose professional lives will require continual use of computer graphics.

And — let's say this only here at the end, and let's say it quietly, though it makes a real difference to the nature of the course — the course is a lot of fun to teach and students really enjoy the high-quality work they produce in the course.

References

[ANG 97] Angel, Ed, *Interactive Computer Graphics: A Top-Down Approach with OpenGL*, Addison-Wesley, 1997

[BRO 90] Brown, Judith R., Steve Cunningham, and Mike McGrath, "Visualization in Science and Engineering Education," in *IEEE Tutorial: Scientific Visualization*, Nielson, Gregory M. and Bruce Shriver, eds., IEEE Computer Society, 1990

[CUN 98] Cunningham, Steve, "Outside the Box — The Changing Shape of the Computing World," invited editorial, *SIGCSE Bulletin* 30(4), December 1998, 4a-7a

[CUN 99] Cunningham, Steve, "The Ubiquitous Bresenham Algorithm and Its Applications Across Many Graphics Processes," in preparation

[FAR 98] SGI Fahrenheit Web site, `http://www.sgi.com/fahrenheit/`

[NAD 98] Nadeau, David R. and Henry Sowritzal, "Introduction to Programming with Java3d," Eurographics 98 tutorial, also presented at SIGGRAPH 98

[TUC 91] Tucker, Allen B. and Bruce H. Barnes, eds., *Computing Curricula 1991: Report of the ACM/IEEE/CS Curriculum Task Force*, ACM Press/IEEE Computer Society Press, 1991

[WOO 98] Woo, Mason and Dave Schreiner, *A Visual Introduction to OpenGL Programming*, SIGGRAPH 98 course notes, 1998

[WOO 99] Woo, Neider, and Davis, *OpenGL Programming Guide*, third edition, Addison-Wesley, 1999