Reprinted from the May 1999 issue of *Computer Graphics*.

New Possibilities in the Introductory Graphics Course for Computer Science Majors

Rosalee Wolfe, DePaul University
Steve Cunningham, California State University at Stanislaus
Scott Grissom, University of Illinois, Springfield
Lew Hitchner, Cal Poly State University

This column is the third in a series that re-examines the introductory computer graphics course for undergraduate computer science majors. In the ten years since the last format discussions on this subject, much has changed in the technology and practice of the discipline. To learn more about this change, several educators, including Scott Grissom, Lew Hitchner, Bill Jones, Susan Reiser and Rosalee Wolfe solicited syllabi form educators who teach this course, and the results were published in the February 1999 issue of *Computer Graphics*.

In this issue, four authors describe their introductory courses. Each of them teaches computer graphics at modest-sized campuses and the details of their courses vary rather widely. However, all of them attempt to address the substantial changes witnessed in the past ten years, and their course descriptions will illuminate how these changes are reflected in course design.

By distributing these course descriptions, the authors hope to open discussions that may lead to a philosophy of the first graphics course. We hope that this philosophy will evolve into a basis upon which people can develop courses that fit their local needs as well as reflecting the changing field. Fundamentally, we believe that the major points of this philosophy are:

- 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. The mathematics of curved surfaces is typically treated in a more advanced course.
- 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 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 APIs 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.

In the following paragraphs, the authors talk about the background and experiences that influenced the development of their courses.

Reprinted from the May 1999 issue of *Computer Graphics*.

Steve Cunningham

"In the late 1980s I had the opportunity to contribute to the development of Curriculum 91 through reviews of its recommendations in the areas of computer graphics and user interfaces. As part of that review, I wrote the course outline for computer graphics that is included in the curriculum.

The computer graphics field has changed dramatically since Curriculum 91 was written, and it is time to reflect on the changes and see how they affect the details of computer graphics courses. I have changed my introductory computer graphics course to be based on OpenGL and to become fundamentally a course in computer graphics programming with an emphasis on computer graphics applications. "

Steve's course description appears in Appendix 1.

Scott Grissom

"At University of Illinois-Springfield we only offer one undergraduate course in computer graphics. I therefore try to expose students to a wide range of computer graphics. I have been using C++ and OpenGL for three years. I want students to create interesting and motivating images as early as possible. Using a high-level API allows them to do that. Towards the end of the semester I briefly introduce concepts of ray tracing and have students use POV-Ray to render an image. POV-Ray requires an understanding about lighting models, view manipulation, and texture mapping and is available on all platforms. The final project involves an interactive application on the Internet using JavaScript, CGI, VRML or Java."

Scott's course description is in Appendix 2.

Lew Hitchner

"The Cal Poly introductory computer graphics course is a practice-oriented curriculum that combines fundamentals with intensive laboratory exercises and programming assignments. Students learn to apply fundamentals through programs that use two high level API's. Our one quarter course covers 3D geometry and transformation basics, event-driven interaction, hierarchical modeling, camera and lighting equations, and rendering techniques (visible surface, texture mapping, etc.). Students use a high level API in all assignments (Open GL and Open Inventor)."

Details of Lew's course appear in Appendix 3.

Rosalee Wolfe

"At DePaul, we have two introductory computer graphics courses. 'Computer Graphics Survey' covers the entire discipline and uses high-level packages (Rhino, POV-Ray) to teach topics from the areas listed in the philosophy and also covers animation. This course is often referred to as the seduction course because students taking this course often decide to embark on additional courses in graphics. Although both are entry-level courses, many find that the survey course helps them when they take 'Computer Graphics I'.

The second course, 'Computer Graphics I' uses Microsoft Visual C++ and OpenGL as a platform. Students are given a 'crippled' wireframe browser to which they add transformations, hidden surface removal, shading, texture mapping, and interactive elements. In addition to learning graphics principles, students learn how to read and modify a large corpus of code. A helpful aid for debugging shaders (faceted, Gouraud, texture mapping) is to inculcate an expectation of a shader's appearance. Computer science students can learn the visual behaviors of rendering algorithms in a small amount time when lectures include a well chosen set of examples."

The course description for 'Computer Graphics I' appears in Appendix 4.

Samples of Student Work

Several examples of student work from these courses are included in this issue of *Computer Graphics* in the "Student Gallery" section. [You should be able to find them in this same directory with a filename of may99gallery.pdf.] Many of these are screen dumps from interactive programs. Although only one of the four courses has a formal final project, all of them provide enough flexibility in the final assignment that students have the opportunity to pursue their own interests.

What do you think?

What do you think of these syllabi? Are there missing topics? We are in the process of gathering feedback on the general items mentioned at the beginning of these article and on the specifics of the syllabi in particular. We're interested in your opinions and experience. Send us email via wolfe@cs.depaul.edu. We look forward to hearing from you.

Reprinted from the May 1999 issue of *Computer Graphics*.

Appendix 1: CS 3600 Computer Graphics I

Dr. Steve Cunningham
Computer Science
California State University Stanislaus
rsc@castor.csustan.edu

Text: Mason Woo et al., The OpenGL Programmer's Guide, Addison-Wesley, 1992.

Course environment: OpenGL on any system that supports it; at this time we primarily use Macintosh G3 systems in our laboratory but students often use Windows systems with their own OpenGL-capable compilers.

Prerequisites: originally Data Structures (CS3) and linear algebra; moving towards CS1 and CS2 equivalents and the ability to represent and manipulate concepts geometrically.

Course philosophy: to develop skill in programming computer graphics as a means for students from many disciplines to communicate information in their fields. All work in the course is 3D from the beginning, and all incorporates animation and/or user interaction.

Topics:
1. Introduction to OpenGL concepts and sample programs.
2. Concepts from Euclidean geometry, with an emphasis on concepts as contrasted with computation. Points, vectors, line segments, planes, normals.
3. Modeling objects in 3-space with OpenGL tools: triangle strips, quad strips, etc. Getting normals for polygons. Using instancing transformations and keeping track of the frame for the object provided by the transformed coordinate system. The scene graph. Hierarchical objects.
4. Projections and viewing. Perspective and orthogonal projections, specifying a view, view volume.
5. Event concepts and the concept of callbacks. Registering callbacks for different kinds of events, including idle. PostRedisplay. Menus. The use of keyboard controls to manage high-DF problems without high-DF devices.
6. Creating a scene with hidden surfaces, Phong lighting, Gouraud shading, material specification, fog, clip planes, texture maps, color blending.
7. Controlling motion through various controls; managing transforms to allow world-centered or body-centered behaviors, moving an object or the camera through a scene.
8. Surfaces through mathematical modeling, pseudo-fractal techniques, and Bezier splines (evaluators)

Programming Assignments:

1. Show me the largest one! (2 weeks) One of the standard applications of computer graphics is creating charts from spreadsheets. In this very simplified version of that application, you are to create an NxM array of (positive) numbers and display a set of NxM 3D bars, each of which is in a position analogous to one of the numbers in the array and is a height that is proportional to the value in that number. You are also to place our example arrow (the "cocktail umbrella" or "paper cone") in the display, pointing to the top of the highest bar, with the axis of the arrow lined up parallel to the line x = y = z.

The display should be presented using perspective, with the view set up to display clearly both the bars and the arrow. It need not allow user-controlled or full-time rotation, but if you choose to use either of these, it will probably make the display clearer.

2. Surface Modeling of a Simple Surface (2 weeks) Create a surface model of a relatively simple object -- for example, a model of an airplane as a pair of tapered boxes for the body, and tapered wedges for the wings, ailerons, and tail. Display this model with one or more lights, and with each of the parts of the object colored uniquely so we can identify it. Use the keyboard to control the rotation of the object around each of the axes, and to control the rotation of the primary light source around each of the axes as well. (This will let you position both the object and the light source interactively wherever you wish, up to the limits of accuracy imposed by your angle granularity.)

3. Fractal Landscapes (2 weeks) Create a surface model of a fractal landscape, as described in section 8.8 of the text. Create four color schemes for the landscape, for winter, spring, summer, and fall, and include a menu that will allow you to choose which season it is -- and hence which landscape color scheme to use. Build your landscape from a triangular, not rectangular, mesh (the book doesn't have this right). Make the landscape interesting -- include such features as

(a) higher elevations of your landscape show rocks instead of trees; the highest elevations show snow. These are done with some randomness, however -- let the height be set as the actual height plus a random value that can be positive or negative, when you compute the color you set the piece.
(b) there is a sea level in your image, and anything lower than that sea level is displayed at sea level and in water color. This might mean that you need to check about splitting your pieces into parts above sea level and parts below sea level.

4. The N-body Problem (2 weeks) Imagine, if you will, a landscape of light and sound in which there are N bodies with masses m[0] ... m[N-1] (real numbers) and velocities v[0] ... v[N-1] (vectors of real numbers). Allow the force of inter-body gravitation to work on these bodies, and trace out the paths of the bodies in 3D space. Have this program animate the environment you produce, and allow the user to rotate the display to examine it from any point (world-centered rotation). The display you produce is to show, for each time step, the position of the objects and the recent path the objects have taken
in 3D space.

The path is probably best demonstrated by the use of some thin cylinders (the GLUT system includes a cylinder object) whose color is initially a fairly light gray (grey outside the United States) and which fades out over a modest number of steps. You can accomplish this in several ways, but the easiest might be to have your display function traverse a fixed-length list or array of cylinders (or cylinder parameters: endpoints, diameters, colors) and have the colors fade from the beginning of the list/array to the end.

5. Taking the Plunge into Surfaces (2 weeks) Design a swimming pool whose surface is created via Bézier splines. You may make this as simple or as fancy as you wish, but it should have at least a 16x16 set of control points. You need not have any sharp edges in your pool design. Design the pool with all the outside edges at "ground level" and see if you can include a pool border – nothing like a challenge for the last project! Use the usual rotation controls so the object can be viewed from all angles, and add a water level with a low alpha value so the pool looks realistic.

Reprinted from the May 1999 issue of *Computer Graphics*.

Appendix 2: CSC 481 Introduction to Computer Graphics

Scott Grissom
Computer Science Department
University of Illinois at Springfield
grissom@uis.edu

Text: Required: Ed Angel, Interactive Computer Graphics with OpenGL, Addison-Wesley, 1997.
Optional: J. Neider, T. Davis and M. Woo, OpenGL Programming Guide, Addison-Wesley, 1993.

Computing Environment: UNIX fileserver, Linux boxes with X servers, C++, OpenGL.

Prerequisites: Data Structures (CS7)

Course Description: Basic concepts, display hardware and techniques, raster graphics, 3D graphics, and processing of pictorial information.

Course Objectives: Understand the basics of computer graphic hardware and software; X windows, OpenGL, client-server model, and be able to write programs to display 2D and 3D data.

Topics:

Week		Reading (from Angel)
1	Introduction	Ch. 1; skip 1.4, 1.5
2	OpenGL at UIS	Ch. 2
3	Recursive Methods and Fractals	Ch. 8.7-8.8; 10.7
4	Input and Interaction	Ch. 3; skip 3.4
5	Geometric Objects	Ch. 4; skip 4.3, 4.8
6	3D Viewing	Ch. 5, 7.7
7	Shading	Ch. 6
8	Review, midterm exam	
9	Shading	Ch. 6
10	Image Analysis using TERA	Lecture notes
11	Ray Tracing with POV-Ray	Ch. 6.10; lecture notes
12	Discrete Techniques	Ch. 10, 7.11
13	Interactive Web applications	Ch. 10
14	Selected topics	Lecture notes
15	The Big Show (student presentations)	
16	Review; Final Exam	

Programming Assignments: Each assignment requires two to three weeks.
Assignment 1: 2D graphics including fractals, Koch Curves, Mandlebrot Sets and other recursively defined curves.

Assignment 2: Complete and extend an interactive paint program.

Assignment 3: 3D renderer that includes options for wireframe, hidden line, flat shading, smooth shading and Gouraud shading.

Assignment 4: Interactive Web application of student's choice. Platforms may include CGI scripts, HTML forms, Java applets, VRML environments.

Assignment 5: Use POV-Ray to implement the scenes created from Project 3.

Reprinted from the May 1999 issue of *Computer Graphics*.

Appendix 3: CSC 455 Introduction to Computer Graphics

Prof. Lew Hitchner
Computer Science Dept., Cal Poly State University
hitchner@falcon.csc.calpoly.edu

Texts: Interactive Computer Graphics, A Top-Down Approach with Open-GL, Edward Angel, Addison Wesley Longman, 1997 This course will cover all chapters except Chapter 9.

The Inventor Mentor, Josie Wernecke, Addison Wesley, 1994. Selected chapters will be covered. This book will serve both as a learning textbook and as a reference for one of your programming lab assignments.

Online Books:
The Inventor Mentor text is also available for reading online on the SGI workstations using SGI's Insight hypertext browsing software on the SGI workstations . Nearly all SGI manuals plus the OpenGL Programming Guide and OpenGL Reference Manual are also available as online books
readable with Insight (select "Help" then "Online Books" from main desktop menu).

Computing Environment: Most program assignment work must be done on the SGI workstations. Some work may be done remotely via network connection from another campus host (e.g., source code editing). Open-GL work may also be partially done on a Personal Computer, but final results will need to be demonstrated on the CSL SGI Indy workstations.
Prerequisites: CS2

Course Catalog Description: Graphics hardware and primitives. Modeling and rendering, geometric transforms, hidden-surface removal, the graphics pipeline, scan-conversion and graphics applications.

Topics: (Unless marked, the chapters are from the Angel book)

Week		Readings
1	Graphics Systems and Models	Ch. 1, 2
	Graphics Programming	
2	Graphics Programming cont'd	Ch. 3
	Input and Interaction	
3	Input and Interaction cont'd	Ch. 4
	Geometric Objects and Transformations	
4	Geometric Objects and Transf. cont'd	
5	Viewing	Ch. 5
6	Modeling	Ch. 8
	Open Inventor	Inventor Mentor
7	Modeling & Open Inventor cont'd	
8	Shading	Ch. 6
9	Implementation Pipeline	Ch. 7
10	Implementation Pipeline cont'd	Ch. 10
	Discrete Techniques	

Reprinted from the May 1999 issue of *Computer Graphics*.

Programming Assignments:

1 Drawing Simple Geometric Objects with OpenGL (2 weeks)
Learning Objectives:

a) 1.Learn the basics of writing a graphics program using the OpenGL (OGL) function library and the OGL Utility Toolkit function library (GLUT).
b) 2.Learn to draw simple primitives and geometric objects using OGL and GLUT functions.
c) 3.Learn to set drawing attributes using OGL functions.
d) 4.Learn to design and code an event-driven application program using GLUT functions.
e) 5.Learn to use the OGL GLUT library keyboard handler function for processing user keyboard interaction.

Design an interactive graphics display program that draws geometric objects. The program will draw one of 10 different shapes chosen by the user, and it will let the user choose either wireframe or solid mode, the color, and the linestyle of the object. Your program must handle user input from the keyboard, set various drawing modes as specified by the keys listed below, and display results determined by the currently selected drawing modes in a window on the workstation screen.

2. Using Geometric Transformations in OpenGL (2 weeks)
Learning Objectives:

a) Learn to construct (model) a 3D object from simple primitives and to compose the object to make more complex primitives using OpenGL functions
b) Learn to use OpenGL display lists for better drawing performance.
c) Learn to apply geometric transformations to 3D objects to compose them into a 3D scene using OpenGL transformation functions.
d) Learn to apply geometric transformations to the viewing projection to change the viewpoint using OpenGL functions.
e) Learn to use the GLUT (GL Utility Toolkit) library mouse, motion, and keyboard handler functions for processing user interaction and to use the GLUT menu functions.
f) Learn to implement drawing objects using object-oriented classes in conjunction with non-object-oriented OpenGL and GLUT functions.

This project will draw four different colored chairs and a table modeled from polygons and positioned on a wire frame floor grid. Using a pop-up menu the user can select a mode that translates or rotates each of the chairs and the table to change their position and orientation in the scene. In addition the view position and orientation can be changed by selecting a menu item. User interaction consists of first using the middle mouse button to select from the menu what will be transformed: a chair, the table, or the view. Next the user may select from the menu what type of transformation: translate or rotate. Then the user may select from the menu what axis or axes to apply the transformation: X, Y, Z, X and Y, X and Z, or Y and Z.
Finally mouse motion while the left button is pressed is used to select a screen position (mouse X coord.) or positions (mouse X and Y coords.) that supply the amount of translation or rotation about the selected axes for the selected object. A right mouse button press exits the program.

3. Drawing Hierarchical Models with Open Inventor (3 weeks)
Learning Objectives:

a) Learn to build and draw a complex hierarchical geometric model using a 3D graphical editor.

b) Learn the basics of writing a graphics program using the Open Inventor (OI) C++ class library.
c) Learn to use interactive control of OI nodes and their property values, and learn to use the OI Examiner Viewer class object.

Design an interactive graphics display program that draws a hierarchical model of a robot. The robot you model must be the 15-segment, 14-joint robot described in the handout given out in class titled "MODELING: Designing an Organizational Hierarchy". Your program must build the geometric model as described in that handout and draw the model on the SGI workstation. It must also allow the user to interactively specify a new "pose" for the robot and then redraw the robot on the screen. A "pose" consists of the set of rotation angle values for each of the robot's joints.

4. Student's Choice Project with OpenGL or Open Inventor (2 weeks)
Learning Objectives:

a) 1.Learn to write the specifications for a graphics design and development project.
b) 2.Learn to use multiple light types and light sources in OpenGL or Open Inventor.
c) 3.Learn to construct and use surface normals for a lighted, 3D object in OpenGL or Open Inventor.
d) 4.Learn to use object surface material types in OpenGL or Open Inventor.
e) 5.Learn to use a new feature in OpenGL or Open Inventor you have not used before.

You get to make up your own problem specification! Choose a graphics application or demonstration that you would like to do. Pick a problem to be solved that provides the following:

fun - you should do something that you enjoy working on and will feel proud of when you're finished (if you maintain a portfolio of your computer science course work, this project should be saved in your portfolio).
intellectual challenge - your problem and its solution must be non-trivial
educational value - you must learn something new

Reprinted from the May 1999 issue of *Computer Graphics*.

Appendix 4: CSC 329 Computer Graphics I

R. J. Wolfe
School of Computer Science, Telecommunications and Information Systems
DePaul University
wolfe@cs.depaul.edu

Texts: Required: Angel, Interactive Computer Graphics. 1997.
Optional: Fosner, OpenGL Programming for Windows 95 and Windows NT, 1997.
OpenGL ARB, OpenGL Reference Manual, 1992.

Programming Environment: Microsoft Visual C++ on Windows 95/NT

Prerequisites: CS2 and either of two quarters of calculus or linear algebra

Course Description: Basic graphics architecture. Coordinate systems. Three-dimensional representations and transformations. Simple visible-surface algorithms. Introduction to illumination. Gouraud and Phong shading. Antialiasing.

Topics:

Week		
1	Overview of pipeline	Notes 1
	Basic 3D viewing concepts	Angel: Ch. 1,2
2	Geometric objects; Transforms	Notes 2
	Data structures supporting model hierarchy	Angel: Ch 4
3	Z-buffer algorithm	Notes 3
	Phong illumination model	Angel: Ch 6.1-6.4, 6.7
4	Smooth shading	Angel: Ch 6.5, 6.9-6.10
5	Texture Mapping	Notes 5
		Angel: Ch 10.1-10.4
6	Midterm; More on Texture Mapping	
7	Graphics “Standards”, Interactive Techniques	Notes 7
		Angel Ch 3
8	Interactive Techniques ctd.; Working with Windows	Fosner Ch 2,
9	Modeling/Animation	Angel Ch 9
10	Windows NT/95 ; MFC	Fosner Ch 6
11	Final Presentations	

Programming Assignments:

1. Overview and Environment configuration. (1 week)

Goals:

a) Configure the computing environment for the quarter.
b) Learn how camera position, camera orientation and field of view affect the resulting image.

For the first goal, students receive source code for a general rendering package, but without most of the graphics functionality. They configure their computing environment to compile and link the modules of the package. They will use this code as a basis for completing subsequent assignments. For the second goal, students receive an executable of a fully functional version of the package which accepts a hierarchical 3D specification language and produces images. They use the working version of the package to manipulate the camera, view-up vector, point of interest and field of view as they learn to zoom, pan and fly through an image.

Reprinted from the May 1999 issue of *Computer Graphics*.

2. Transformations (1 week)
Students add code to correctly carry out transformations specified in the hierarchical 3D graphics language. This requires studying the data structures that support the hierarchy.

3. Faceted shading (1 week)
Students modify code to implement z-buffer shading, compute polygon normals and specify lights.

4. Gouraud shading (1 week)
Students implement smooth shading with highlights, which requires that they compute vertex normals.

5. Project proposal (1 week)
Students choose their final project. They can select one from a list of suggestions or describe one of their own choosing.

6. Texture mapping (2 weeks)
Students implement planar and spherical shading.

7. Interacting with MS Windows (1 week)
Students implement the “File Save” menu item, which saves a rendered image as a Targa file.