# Rendering + Modeling + Animation + Postprocessing = Computer Graphics

**John L. Lowther and Ching-Kuang Shene**
**Department of Computer Science**
**Michigan Technological University**
**1400 Townsend Drive**
**Houghton, MI 49931**
**{john|shene}@mtu.edu**

## ABSTRACT

This paper discusses problems of current introductory computer graphics courses and the author's effort in overcoming them, which includes a new course proposal and its accompanying software tools. By tightly integrating software tools into course topics, we will be able to cover more modern topics (*e.g.*, radiosity, volume rendering, and animation) which are usually sketched or even skipped in the popular "programming" approach. Details of our software tools are also discussed.

## 1. INTRODUCTION

Nowadays, students coming into a computer graphics course have seen movies that have fantastic graphics effects (*e.g., Toy Story*, *A Bug's Life* and the *Star War* series). These students have also acquired a certain level of graphics knowledge by playing games and reading popular magazines. Their expectations are certainly high for their first graphics course. Moreover, many deep and powerful theories were developed during the past decade. Either because these topics are too new or because they are too difficult to teach, they are frequently only sketched or even skipped in favor of a "programming" approach. Hence, what a student learns in an introductory course might only be some programming skills using a graphics API, usually OpenGL, along with some fundamental computer graphics knowledge. Students are only exposed a little to modern developments which are frequently used by the graphics industry for creating fantastic special effects or realistic images.

At Michigan Technological University, we have three computer graphics related courses: an introduction to graphics course, a computing with geometry course for junior and senior students, and, an advanced graphics course for graduate students. We also have graduate courses on geometric modeling and visualization. The minimum prerequisite of the introductory graphics course is only a sophomore software development course which is usually taken after a data structure course. This introductory course covers essentially all traditional topics with the help of publicly available tools (*e.g.*, GLUT and GLUI) and other tools developed by the instructor of this course. We believe that our current approach is a successful combination of theory and the current popular programming approach using tools. However, we still feel that we need to do more to introduce students to modern theories and developments. While many educators have already

observed the need of a new graphics course and proposed some approaches,[1] our approach is more ambitious and non-traditional.

We believe that graphics consists of four major components: rendering, modeling, animation, and postprocessing. The rendering part has become the major topic in typical graphics textbooks and the main thrust of a "programming" approach. There is nothing wrong to mainly focus on rendering methods. The problem is that the "programming" approach cannot cover global illumination models (*e.g.*, ray tracing and radiosity) and volume rendering, because most popular API's only implement local illumination models. Every scene contains some objects and object building requires the knowledge of modeling. However, building a good and realistic model is not part of any API. Animation is the skill for creating animation sequences and simulating physical events. Finally, postprocessing makes created scenes satisfy additional requirements and requires a special set of 2D operations. Obvious topics should include, but not be limited to, filters, morphing, dithering, and other current techniques. Since the "programming" approach usually cannot go this far to cover all four components, additional course materials and working environments are required. This has become the main thrust of our project.

To address this need, our goal is to design a comprehensive introductory computer graphics course that covers all four major components to some depth with a breath-first and learning-by-doing approach. To support this goal, a pedagogical environment is required for students to experiment and visualize important concepts, and to perform a semester-long project of implementing various components. In the following, Section 2 discusses a number of problems that prompt the authors to initiate this project; Section 3 presents a description of our proposed course and its software tools; Section 4 addresses the software modules that are currently under development; and, finally, Section 5 has our conclusions.

## 2. CURRENT PROBLEMS

After teaching a computer graphics course for several years, we have found four major problems: **(1)** course materials are inadequate, **(2)** "programming" alone is not sufficient, **(3)** the "design" component is missing, and **(4)** graphics educational research is almost non-existent in spite of the recent and rapid advance of graphics technology. The following sections discuss each of these problems in some detail.

### 2.1 Course Materials Are Inadequate

Comparing the content of popular graphics textbooks[2] with features available in low-end graphics systems shows that some topics in our textbooks are a little out-of-date. A survey conducted by Wolfe[3] clearly shows that most syllabi did not cover all four important components of graphics. Topics frequently covered (*e.g.,* viewing transformations, lighting models, and 3D

---

[1] See, for example, Steve Cunningham, Powers of 10: The Case of Changing the First Course in Computer Graphics, *Thirty-first SIGCSE Technical Symposium on Computer Science Education*, March 8 to March 12, 2000, Austin, Texas, pp. 46-49, and Rosalee J. Wolfe, *3D Graphics: A Visual Approach*, Oxford University Press, 2000.

[2] Edward Angel, *Interactive Computer Graphics*, 2nd edition, Addison-Wesley, 1999; James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes and Richard L. Phillips, *Introduction to Computer Graphics*, Addison-Wesley, 1993; and Donald Hearn and M. Pauline Baker, *Computer Graphics*, 3rd edition, Prentice Hall, 2000.

[3] Rosalee Wolfe, A Syllabus Survey: Examing the State of Current Practice in Introductory Computer Science Courses, *Computer Graphics*, Vol. 33 (1999), No. 1 (February), pp. 32-33.

transformations) are now part of typical graphics programming API's or even have been incorporated into graphics hardware. Other important and frequently used features (*e.g.*, global illumination models - ray tracing and radiosity, animation, texture mapping, surface materials, design and modeling, and post-processing) are less frequently covered. These topics cannot be presented thoroughly without good pedagogical tools. A panel led by L. Hitchner[4] at the Thirtieth SIGCSE Conference also discussed this problem and made some suggestions. While the panelists correctly pointed out a right direction, their suggestions are still not enough to catch the trend. Recently, Cunningham proposed an API-based geometric problem-solving approach. His proposed outline of an introductory course[5] is basically the "programming" approach with some extensions. This approach has been used by the authors as well as many other graphics educators for years. Moreover, the meaning of "geometric problem-solving" is not clearly stated and the rendering techniques covered center around only local illumination models. Thus, ray tracing, radiosity, volume rendering, 2D image manipulation, and other modern developments are excluded. In contrast with Cunningham's point of view, our approach is to look into the future by incorporating modern as well as traditional topics into a comprehensive introductory course.

### 2.2 "Programming" Alone Is Not Sufficient

The current trend in teaching graphics courses is the "programming" approach typified by Angel.[6] There are three popular approaches: algorithmic, survey, and programming. The algorithmic approach teaches everything about what makes a graphics system works. As Lao-Tzu, a great Chinese philosopher (604-531 BC), observed,[7] students will hear about the theory and forget it quickly. The survey approach will certainly make students feeling good; but, they will remember many things without acquiring the skills for their career. The programming approach teaches students how to do things. This seems to fit the "I do and I understand" theory. Unfortunately, the coverage of the programming approach is quite limited compared with those theoretical and survey textbooks. Many topics (*e.g.*, shadows generation, radiosity, and curve/surface modeling) could be too difficult for students to implement from scratch. Moreover, without seeing the effect, it is even more difficult for them to program. Also, the programming approach often is constrained by the length of a course. There is not enough time to develop and implement every topic. As a result, pedagogical tools are required for students to practice before they implement.[8]

### 2.3 The "Design" Component May Be Missing

Another major problem is that the design and modeling component may be totally missing. We probably concentrate too much on rendering. While textbooks discuss curve and surface

---

[4] Lew Hitchner, Steve Cunningham, Scott Grissom and Rosalee Wolfe, Computer Graphics: The Introductory Grows Up, *Thirtieth SIGCSE Technical Symposium on Computer Science Education*, New Orleans, March 24-28, 1999, pp. 341-342.

[5] See footnote 1 above.

[6] Edward Angel, *Interactive Computer Graphics: A Top Down Approach with OpenGL*, second edition, Addison-Wesley, 2000, and Teaching a Three-Dimensional Computer Graphics Class Using OpenGL, *Computer Graphics*, Vol. 31 (1997), No. 3 (August), pp. 54-55.

[7] *I hear and I forget. I see and I remember. I do and I understand*. See Lao-Tzu, *Tao Te Ching*, translated by D. C. Lau, Knopf, 1994.

[8] C. A. R. Hoare said *You can't teach beginning programmers top-down design because they don't know which way is up*. If students do not know the effect of radiosity, they will not be able to recognize if their implementations are correct and of good quality. Thus, knowing the effects must be the first step.

representations (*e.g.*, Bézier, B-spline, and NURBS), students usually learn a set of cold formulas. They have little chance to learn the proper use and impact on design of these curves and surfaces. Design techniques are vital in graphics because they are required for putting objects into a scene. Many design tools are now available in most popular graphics systems (*e.g.*, trueSpace, LightWave 3D, and 3D Studio Max) and used in all typical graphics related productions. If we expect our students to take jobs in the graphics industry, these design and modeling related topics must be covered.

Incorporating design techniques into a graphics course is not a very difficult task, although striking a balance between design/modeling and programming is. In our Introduction to Computing with Geometry course, which does not focus on graphics but uses graphics as a vehicle for learning modeling and geometry related computation,[9] with a minimal introduction of design principles and modeling techniques, students were able to develop very creative and imaginative designs of good quality. What we educators need to do is provide students with an environment for them to develop their creativity and imagination. If only programming is taught, students' creativity and imagination will be severely limited by the difficulty of implementing some challenging features (*e.g.*, designing a scene which is rendered with radiosity is easier than implementing a good radiosity system).

### 2.4 Are We Ignoring Graphics Education Research?

The answer seems a "yes." Graphics technology has had an explosive development in recent years, programming API's have reduced to just a few standard ones, and good graphics systems for PC's cost as little as 200 dollars. But, compared with other topics, graphics education research seems missing in major education conferences and journals. There are three papers published in the 1995 SIGCSE Conference,[10] one in 1996,[11] none in 1997, one in 1998,[12] one in 1999,[13] and one in 2000.[14] SIGGRAPH's *Computer Graphics* also publish graphics education related articles. Professional journals such as *Computers & Graphics* also occasionally publish education research articles. This shows that the computer science education community has not yet done enough for computer graphics courses. While graphics may not be the main stream of computer science education research, we should not ignore the current trend and should provide our students with up-

---

[9] Note that the meaning of "geometry" here is very different from and has a much wider coverage than that of Cunningham's. We emphasize the development of understanding and skills of converting a geometric object into various representations, implementation of algorithms for each representation, and design activities. The interested readers may find the following URL helpful: `http://www.cs.mtu.edu/~shene/NSF-2/index.html`.

[10] Dino Schweitzer and Tom Appolloni, Integrating Introductory Courses in Computer Graphics and Animation, *Twenty-Sixth SIGCSE Technical Symposium on Computer Science Education*, Nashville, Tennessee, March 2-4, 1995, pp. 186-190; Andrew Sears and Rosalee Wolfe, Visual Analysis: Adding Breath to a Computer Graphics Course, pp. 195-198; and Lee H. Tichenor, Inexpensive Advanced Graphics Applications for the CS Majors Graphic Class, Integrating Introductory Courses in Computer Graphics and Animation, pp. 191-194.

[11] David Goldman, Richard E. Eckert, and Maxine S. Cohen, Three-Dimensional Computation Visualization for Computer Graphics Rendering Algorithm, *Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education*, Philadelphia, Pennsylvania, February 15-18, 1996, pp. 358-362.

[12] Yuan Zhao, John L. Lowther and Ching-Kuang Shene, A Tools for Teaching Curve Design, *Twenty-ninth SIGCSE Technical Symposium on Computer Science Education*, February 25-March 1, Atlanta, Georgia, 1998, pp. 97-101.

[13] Yan Zhou, Yuan Zhao, John L. Lowther and Ching-Kuang Shene, Teaching Surface Design Made Easy, *Thirtieth SIGCSE Technical Symposium on Computer Science Education*, March 24 - 28, New Orleans, 1999, pp. 222-226.

[14] Steve Cunningham, Powers of 10: The Case of Changing the First Course in Computer Graphics, *Thirty-first SIGCSE Technical Symposium on Computer Science Education*, March 8 to March 12, 2000, Austin, Texas, pp. 46-49.

to-date and comprehensive training to enhance their understanding and capability for their future careers.

## 3. PROPOSED COURSE AND TOOLS

To go one step beyond the API programming approach and to promote the "I hear, I see and I do" scheme, our proposed course is different from a traditional one. It is unique because all important components of graphics are covered in a coherent way with an environment for students to easily develop a comprehensive understanding, and to perform a semester-long programming project. The design of this course has the following purposes in mind:

- Free students from spending a tremendous amount of time in doing non-essential programming just to get a simple implementation working.
- Help students quickly appreciate each concept and algorithm. Some concepts and algorithms are easy to understand but difficult to implement (*e.g.*, ray tracing and radiosity).
- Provide an environment for students to practice basic design and modeling skills, and to visualize the inner working of various algorithms, special effects, and other concepts.
- Provide a system for students to perform semester-long programming projects. This approach has been very popular in other courses (*e.g.*, Nachos for operating systems). However, no similar system exists for graphics.

Our proposed course consists of five units: basic understanding, global illumination, object modeling, animation, and post-processing. The basic understanding unit has five sub-units: camera and viewing, textures, material and surface, lighting, and local illumination models. Color models and transformations are parts of the illumination models and camera and viewing units, and event-driven programming is an integral part of the programming projects using GLUT and GLUI. The following sections present some details of each unit.

### 3.1 The Camera and Viewing Unit

This unit covers camera, viewing, and geometric transformations. Software for this component serves as the foundation of the whole system. Topics covered in this unit are usually scattered throughout a traditional graphics course. In fact, these materials can be grouped together in a coherent way and covered quickly for students to understand the entire nature of a graphics system and to permit them produce good-looking images. Details can be discussed later after students have acquired sufficient experience. In this way, they will have enough incentive to learn some theory. The following discusses the software designed for the five sub-units.

**Camera**: This component utilizes four split windows for top/bottom, left/right, front/end, and camera views. The World View Window provides the position of the camera, view volume, front and back clipping plans, view plan, and up, right and viewing direction vectors. All viewing transformation matrices and their meaning are shown in separate windows, and are updated on-the-fly when the characteristics of the camera change. Hence, students can have a complete view and understanding of the inner working of viewing transformations. They can also control the focal length and depth-of-field of the camera.

**Lights:** A number of light sources are placed at default locations. The World View Window will show the location, direction, type (*e.g.*, spotlight), color, intensity, and other characteristics of each light source. Since all objects, including light sources, can be transformed, it is easy for students to see and learn the effect of all involved characteristics such positions of light sources, attenuation factors, specular highlights, textures and materials of objects, characteristics of the camera, and related topics.

**Object Primitives**: A number of primitives (*i.e.*, blocks, spheres, cylinders, cones, tori and various regular polyhedra) are available. Students can select and drop objects into a scene. Once an object is in the scene, it can be seen in all four split viewing windows and in the World View Window, and displayed in wireframe or rendered mode.

**Geometric Transformations**: Students can translate and rotate the camera, all light sources and all objects in the World View Window. They can also turn on/off and change the characteristics of any light source. Translations, rotations, scalings and shears can be applied to objects. In this way, students will be able to create a scene and setup the camera and light sources easily and quickly. The background theory will be discussed later in this course.

**Texture Mappings and Surface Materials**: Some predefined texture maps (*e.g.*, wood, marble, bump, and checker) and surface materials are available in the Texture Window and Material Window. Students can select color, texture, and material for any object. Fundamental concepts and programming skills of texture and material mapping will be discussed. The basics of texture mapping will also be covered.

**Local Illumination and Shading**: This part supports ambient and diffuse reflection, attenuation, and specular reflection. The Illumination Model Window can be used for modifying the parameters of the illumination equation. The Shading Window allows students to visualize the effect of these factors, and the rendering procedure of Gouraud shading and Phong shading.

### 3.2 The Global Illumination Unit

This unit includes ray tracing and radiosity. Students will learn reflection, transparency and refraction with recursive ray tracing. The radiosity component will implement some popular form factor computation methods, with and without progressive refinement. Software for this unit is not designed to compete with the existing ray tracing (*e.g.*, POV-Ray[15] and Radiance[16]) and radiosity systems (*e.g.*, Lightscape and Radiance). Instead, its purpose is to provide students with an easily accessible, unified and simple environment for them to see the internal working, be familiar with their effects, and have a chance to practice these features.

### 3.3 The Object Modeling Unit

---

[15] Chris Young and Drew Wells, *Ray Tracing Creations*, 2nd edition, Waite Group Press, 1994.

[16] Greg Ward Larson and Rob Shakespeare, *Rendering with Radiance: The Art and Science of Lighting Visualization*, Morgan Kaufmann, 1998.

This unit provides students with an environment to learn modeling and design techniques. In addition to the primitives described in the Camera and Viewing unit, this unit will initially support the following modeling activities: polyhedral objects, hierarchy of objects, extruding and sweeping, surfaces of revolution, cubic NURBS curves and bicubic NURBS patches, and constructive solid geometry. Polyhedra editing will also be supported to some extent so that students can practice a clay-modeling type shape editing operation. Constructive solid geometry objects can either be triangulated or exact (for ray tracing). All other objects, including surfaces of revolution and objects generated with extruding and sweeping, will be triangulated. This will simplify the implementation and achieve a faster speed but have the disadvantage that polyhedron models are not very accurate. This is what most inexpensive modeling systems do and should not affect our introductory graphics course.

### 3.4 The Animation Unit

This unit provides students with an environment to practice important animation skills. They can set up animation paths for camera, light sources, and objects using cubic B-spline curves. They can also specify key frames and perform forward and inverse kinematics, and add transformations to each component of an object for character animation. A bone-based animation subsystem is planned. Other interesting animation options (*e.g.*, particle systems, walking, and deformation of soft objects) will be added if time permits.

### 3.5 The Post-processing Unit

This module will provide students with some limited post-processing capabilities. Planned features include image enhancement, dithering, sharpening, blurring, warping, morphing, compositing and other frequently used filters and their implementations.

### 3.6 Programming Support

This system will be used not only in classroom as a tool for enhancing teaching, but also provide a platform for programming projects. In other courses, there are many tools for students to implement a significant portion of a working system. For example, Nachos and Minix have been very popular in operating systems courses, and Lex and Yacc in compiler design courses. Unfortunately, no similar system exists for graphics courses, although almost every instructor has tried his/her best to provide students with some modules for programming project use. Thus, a system that is similar to Nachos is urgently needed.

A library for creating buttons, slides, track-ball, and Gimbal lock has been developed and used in previous NSF projects. The GUI of this system will use this library with extensions. Once this GUI library becomes available, students can use it to create their own or improve the existing system. An instructor can replace one or more components with templates for students to develop their own versions, perhaps with extensions. For example, an instructor can replace the form factor computation module with a template and ask students to implement a modified or simpler algorithm. In this way, an instructor will be able to teach this course using this system for students to gain hands-on experience and deeper understanding, to learn the effect of each taught topic, and to do their programming projects.

## 4. PAST SUCCESS AND PROGRESS

The method used for developing this course has been very successful in a previous NSF supported course Introduction to Computing with Geometry. We have shown that, with properly designed tools, one can teach juniors about Bézier, B-spline and NURBS curve and surface design, and many important algorithms (*e.g.*, de Casteljau's, de Boor's, knot insertion, subdivision, and degree elevation) in an intuitive and non-mathematical way. These topics are frequently skipped in a graphics course due to their mathematical content and time constraints. The above mentioned course and DesignMentor, the accompanying software tool, have made all topics down-to-earth so that juniors can understand fundamentals, program algorithms, and carry out elementary design tasks. DesignMentor was announced in March 1999 and since then has had more than 900 downloads from all over the world. Approximately 32% are from CS and related departments. The course electronic book has 13 hits daily on average, most from off-campus. Hence, we believe that it is a success and anticipate that the same development process applied to this project will also be successful.

Currently, we are developing two of the most difficult and most needed modules. The first one is for radiosity. In this module, a user supplies a scene to the form factor computation subsystem, which, in turn, pipes the computed form factors to the radiosity rendering subsystem. This rendering system supports Gauss-Seidel (*e.g.*, gathering) iteration with and without subdivision, and progressive refinement (*e.g.*, shooting). A user can step through iterations to see an intermediate result, turn on the visualization component to see the computation process (*e.g.*, gathering and shooting radiosity, and subdivision), and perform a walkthrough. Each of the submodules is replaceable for students to implement their own version and/or improvement. The second module is about volume rendering. Volume rendering has been very popular in medical imaging and visualization, but does not catch much attention in the graphics education community. This module can read in a MRI scan, reconstruct the object, perform slicing operations, and help visualize the marching cube algorithm. We anticipate to complete these prototypes and the course units/modules mentioned in previous sections in about a year.

## 5. CONCLUSIONS

In previous sections, we have presented the problems leading to a new design of an introductory computer graphics course and its accompanying pedagogical software tools. Under NSF support, we have started the development of software tools. We believe our course will cover more modern topics without sacrificing the traditional and programming elements. The anticipated outcome of this project will include an electronic book, a set of programming notes, and a number of accompanying software tools and their user guides. All materials will be made available to educators on the Internet.

## ACKNOWLEDGMENTS

This work is partially supported by the National Science Foundation under grant DUE-9952621 and grant DUE-9696084.