Rich Resources for Computer Graphics Educators

Contact
Dalton R. Hunkins
Computer Science Department
St. Bonaventure University
St. Bonaventure, New York
14778 USA
+1.716.375.2003
hunkins@cs.sbu.edu

David B. Levine
St. Bonaventure University
dlevine@cs.sbu.edu

Demonstration programs have a long and storied history in the field of computer science education. This is particularly true in the area of computer graphics where the displays are often worth more than the proverbial "thousand words." This paper presents a suite of such programs, discusses their pedagogical role within the field of computer graphics education, and places them in context with respect to other such suites currently available.

Introduction and Previous Work

Demonstration programs have probably been around as long as computer science has been taught. Examples can be found in essentially every programming language and for virtually every sub-field of computer science. It is now common for the source code to sample programs presented within textbooks to be packaged on CD-ROM or made available through the publisher's Web sites.

Historically, demonstration programs have included both text-based console applications and those using graphical interfaces. Not surprisingly, many of the best examples of the latter illustrate concepts within computer graphics. Whether the output is textual or graphical, a large number of demonstration programs are static. They illustrate their concepts through single runs and/or still images — possibly controlled through initial numerical input.

Within the field of algorithm design, it is not uncommon to see demonstration programs that featured static displays. On the other hand, systems such as BALSA[1] and GAIGS[3] offer students much more. Students are required to interact with the algorithms, and their interactions have noticeable, immediate effect, which, in turn, provide a greater benefit to the students. Studies have established that the increased interaction aids students in learning the concepts presented. Within the field of computer graphics, there are several suites of demonstration programs that are interactive. It would be impossible to list them all here, so only a few representative examples are discussed.

One of the best early examples of interactive programs within the field of computer graphics is Dino's Demos[8]. The demos were written by Dino Schweitzer and are a series of programs that primarily illustrate concepts related to two-dimensional graphics (for example, Bresenham's line drawing algorithm). Dino's programs permit the student to interact with the program, changing parameters dynamically and observing the effect. Variations of some of these demos have been rewritten in Java and are available[4].

Scott Owen's Computer Graphics and Visualization Courseware Repository Web page[6] has links to two suites of interactive demo programs, one of which is Avi Naiman's Interactive Teaching Modules for Computer Graphics[5]. While Naiman's tools also deal primarily with two-dimensional graphics, they are interesting because the package also includes a library of software routines that students (or professors) can use to build their own demonstration programs.

More recently, Rosalie Wolfe[9] has developed TERA (Tool for Exploring Rendering Algorithms). As its name implies, TERA concentrates on rendering algorithms and on how they make objects appear. "With TERA, students can change the appearance of objects in a scene by clicking on a rendering option."[9] One interesting feature of TERA is the opportunity for students to test their understanding of the concepts at hand through the use of built-in quizzes.

Additionally, Nate Robins has produced a suite of demonstration programs to aid in the understanding of OpenGL programming[7]. Of the three suites discussed, Robins' is the closest to the work presented here in that his tutorials cover similar topics and have a similar "look and feel."

Design Goals

The suite of programs presented here shares some design goals with each of the previously discussed systems. The primary goal, of course, is to create interactive programs that demonstrate concepts in the field of computer graphics. The other goals are outlined below:

- The programs must allow students to modify parameters of the simulation in a dynamic manner.
- The programs must update the display in real-time in response to student inputs.
- Each user interface must be self-evident.
- The user interfaces must be consistent across the suite.
- The programs should incorporate discussion of the processes being implemented, including the appropriate mathematics.
- The programs should provide an opportunity for informal student assessment.
- The programs should be implemented with reusability in mind so that students or instructors can easily modify and extend the suite to include other concepts.

All of the programs in the suite have similar user interfaces. Upon launching the program, the student is presented with a console window with any instructions specific to that tutorial. Upon dismissing the window, the student sees a split screen. The left-hand side of the screen is dedicated to windows for displaying the model and user controls while the right-hand side is used for discussion of the concept and informal student assessment. The discussion window displays a mix of static text and dynamically updated computations based on user input.

Diffuse Reflection

The Diffuse Reflection tutorial shows most of the features of the suite. Students interact with this program to see how the perceived color is determined. A simple Phong illumination model is used in this demonstration. A screen snapshot is shown in Figure 1.

The left-hand side of the screen consists of a main viewing window and two control windows. The main viewing window contains a model of two walls meeting at a corner. The student is looking at the corner from the outside. One control window allows the user to control the presence of two omni lights — one on each side of the model — as well as the presence of ambient light. This is accomplished through a radio button interface. The second control window presents a view from above the walls with vectors drawn to show both the vertex normals and the directions from the vertices to the lights. Since the angle between a vertex normal and the direction to a light is important to the diffused reflection computation, the student is allowed to adjust the vertex normals for either wall at their common edge. By selecting a vertex normal and rotating the normal by dragging the mouse, the student can see the effect on the computation and, in turn, the perceived color at the vertex. Each normal may be rotated individually or in concert with the other. The vectors start in their conventional orientation, but the student can experiment with them to learn about particular effects. For instance, when the normals are set for the adjacent edges to point in the same direction, the student will observe that the visible edge between the two surfaces vanishes. No matter what type of manipulation is performed (light changing or normal changing), the model view is updated in real time.

As with other demonstration programs in this suite, the right-hand sub-window contains text designed to support the teaching goals. Discussion of the Phong Illumination Model is given here. At the bottom of this window, the actual computations are shown for the color determination at one of the vertices with the student manipulated normal. The numbers used in these computations are updated in real time and are an accurate reflection of the student's current choice in the control windows.

As the student manipulates the controls (one of the normals, for example) both the input view, in this case the angle used for the illumination computation, and the model view with the resultant color are updated in real-time. Controlling the input helps the student understand the effects of the manipulation, while updating the result helps build connections between the mathematical representation and the visual one.

At any time, the student can ask for a quiz. The form of a quiz question is either multiple choice or true/false. In either case, the student selects an answer from the set of possibilities through mouse input. A hint is available after the student makes at least one attempt to answer the question. Since these programs are envisioned as teaching tools and since there is no control over the testing conditions in any event, no scoring of the quiz is done.

As with all programs within the suite, the quiz questions and the explanation of concepts are stored in separate files in "clear text" format. Thus, an instructor can customize the experience by providing instructions that match particular teaching goals by providing alternate explanations of the process (perhaps to emphasize other aspects of the subject matter), or by providing additional or alternate quiz questions. As a result, it is also a simple task to switch the delivery language for most of the application. One need only change the text files.

The Rest of the Suite

There are currently eight programs in the suite. The Diffuse Reflection program was described in the previous section. The other seven are:

Ambient Reflection

This tutorial focuses on the determination of color using the Phong Illumination Model. The student can control the color of an ambient light as it shines on four swatches of material (cyan, yellow, magenta, and white). The color computations for any one material (chosen by the user through a mouse selection) are displayed in the discussion window on the right while the result of the computation is shown in the model window on the left.

Gouraud Shading

This tutorial shows Gouraud Shading in action. The student is presented with a triangle and can select the colors at each of the vertices. The shading is updated in real time.

Gouraud Interpolation

This follow-on to the Gouraud Shading program focuses on the computations used to determine the color of "interior" points of a triangle using Gouraud shading. The student is presented with a triangle, whose vertices are red, green, and blue. The student selects a given point of interest within the triangle, and the interpolation computations are then displayed. The student may focus on either the final computation or upon one of the intermediary computations.

Spotlight
This tutorial demonstrates how changes in the different parameters associated with a spot light effect the image. A simple spotlight is directed at a mesh. The student can control the cutoff angle and the exponent of the light as well as the x-coordinate at which the light points.

RGB and HSV
These two tutorials are designed to help students develop intuition about the respective color models. The student is presented with slider values for the parameters of the model as well as a physical representation of the color space (cube or cone). The student can manipulate either the numeric values or the point of interest in the physical space, thereby developing the desired intuition.

Transparency (See Figure 2)
This program simply displays a cyan vase on a red and white checkerboard. The student can control the alpha value of the vase, sliding it from opaque to transparent. The blending computations for the selected alpha are displayed in the discussion window.

Classroom Use

Since the demonstration programs deal with graphics concepts and since the discussion can be easily changed with a switch in a text file, we have used these tutorials with two disparate audiences. At St. Bonaventure University, introduction to computer graphics is taught in two separate courses, Computer Graphics (CS 256) and The Science of Images (CS 126). Computer Graphics is the traditional computer graphics course taught to CS majors. In addition to general concepts in computer graphics, this course concentrates on understanding various algorithms and uses Ada and OpenGL as a teaching vehicle. On the other hand, The Science of Images has no prerequisites and is intended for a general audience. While some CS students do take this course as a university elective, it is primarily intended for students from other disciplines such as the visual arts. Many of the concepts taught in the CS majors' course are also included here. However, since algorithms and implementation are not as important to the non-majors, students in The Science of Images use 3D Studio Max[2] as their vehicle to model and render images.

In The Science of Images, the tutorials are used primarily to build intuition. A powerful tool such as 3D Studio Max may cause distraction when we want to focus on an underlying concept. In this case, the tutorials are presented during lecture to help separate abstract concepts from the manipulative overhead associated with complete image generation. Although there are no formal laboratories involving them, the tutorials are also available for students to use outside of class.

In Computer Graphics, the tutorials are used in three ways:

- The first way is similar to the use in The Science of Images class, but depending on the topic, more emphasis may be placed on the technical computations. (Note, again, that it is easy to craft different text files for display in the explanation window).

- The source-code listings for the tutorials are used as examples of programming with OpenGL. The listings are discussed in class, providing the students with examples of interactive programs and widgets built upon OpenGL and the glut utility library.

- The tutorials themselves are used as a springboard for student assignments. If it is true that we (as teachers) never really learn something until we teach it, then it should follow that the most effective way to teach our students would be to have them teach us. As the students have access to all of the source code and have studied it in class, it is a simple matter to have them construct new tutorials covering other topics. The common user interface and the extensive code reuse throughout the suite enables the student tutorial authors to concentrate on how they wish to present their subject matter, coding of the key concepts, and (gasp!) writing of the explanations and quiz questions. The source code may be given to students for tasks as mundane as changing menu items or as complex as the creation of an entirely new tutorial. In fact, some of the tutorials in this suite were originally conceived and partially developed by Matt Hartloff, one of our students.

Comparison With Other Works

The primary similarity between our work and that of Naiman, other than the subject matter, is that both systems provide libraries for students (or faculty members) to continue to develop more tutorials with the same look and feel. Naiman's suite of programs is a bit larger and more diverse, but as he notes, the tutorials were not written by native English speakers and are therefore a bit difficult for some students to use. Also, since it was developed by a larger group of individuals, there is a greater diversity of user interfaces among them.

The largest area of commonality between our suite and the TERA system is in the area of assessment. Both systems can be used either in an exploratory mode or in a mode involving a self-administered quiz. Both have simple user interfaces, but there is a significant difference between them. TERA appears to have all of its images pre-stored on its CD, and the user "controls" the display by making a selection among a small number of discreet display options. TERA's extensive library of images (over 500,000) guarantees that concepts are well covered, but there is a fundamental philosophical difference between the two approaches. TERA shows a number of much richer images, focusing the student's attention on the final image, and provides minimal user control, whereas our suite gives students more control and concentrates their attention on the process.

The suite most similar to ours is Nate Robins'. Like our tutorials, his have a model view and control windows. Whereas our programs focus on the concepts and algorithms used in computer graphics, Robins' programs demonstrate various OpenGL function calls and their effect. In turn, rather than providing user control through manipulation of the model as our suite, Robins provides user control through the varying of the parameters to OpenGL function calls with mouse click-and-drag operations. This is a very nice user interface that is consistent with the intent of being a suite of tutorials on OpenGL. In both suites, three-dimensional scenes are updated in real time, giving the illusion that an interpreter is running the code. There are cosmetic differences between the two suites, namely, explanatory text and quizzes within our suite.

There is no point in attempting to rate the four systems, as they have different intents and design goals. In fact, it is unnecessary to choose amongst them. While the user interfaces are different in each of the four environments, each is so simple to learn that a typical student can master the interface in less than five minutes. Further, it is reasonable for an instructor to use more than one of the suites within the same course, if the choice of topics makes an instructor so inclined. In fact, we have used Robins' tutorials along with ours in our Computer Graphics course for CS majors.

Further Development

Source code and executables (for Windows) for all of our tutorials can be found at: web.sbu.edu/cs/graphics/

Conclusions

We have developed and presented a suite of tutorials for use in teaching computer graphics. All of the tutorials permit the user to manipulate a model in a three-dimensional world and see the effects of those manipulations in real time. Each of the tutorials also features a combination of static explanatory text and dynamic displays of the computations being performed. The separation of the explanatory text from the application itself makes it simple to customize each tutorial for different languages or for different courses. The framework also supports the construction of new tutorials with minimal effort.

While other similar work has been presented in various forums, this suite is unique in both the particular concepts addressed and in its presentation of them. In many respects, it combines the best ideas of previous work within one framework. Rather than competing with the previous work, however, it can be integrated with other systems in a mix-and-match style to provide a teacher with the widest possible variety of tools.

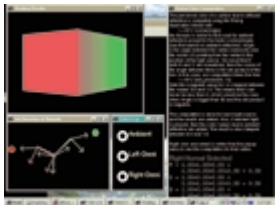

Figure 1. Diffuse Reflection

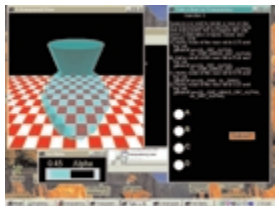

Figure 2. Transparency

References

1. Brown, M. (1987). *Algorithm animation*. MIT Press, Cambridge, MA.
2. discreet, 3d studio max (software package). (2000). URL: www2.discreet.com/products.
3. *GAIG User Manual, Ver. 3.0.* (1994). Lawrence Computing Center Publications.
4. Min, P. (1999). *Computer Graphics Applets*. Online, September 5, 2000, URL: www.cs.princeton.edu/~min/cs426/applets.html.
5. Naiman, A. (1996). Interactive teaching modules for computer graphics. *Computer Graphics* 30, 33-35.
6. Owen, S. (2000). Curriculum and instructional materials. Online. September 1, 2000, URL: asec.cs.gsu.edu/asecdl/materials/C_and_I.htm.
7. Robins, N. (2000). Nate Robins - Open GL - Tutors. Online, September 5, 2000, URL: www.xmission.com/~nate/tutors.html
8. Schweitzer, D. (1992). Designing interactive visualization tools for the Graphics Classroom. *SIGCSE Bulletin*, vol. 24, no. 1, 299-303.
9. Wolfe, R. J. (2000). *3D graphics: A visual approach.* Oxford University Press.