Interactive Teaching Modules for Computer Graphics

Avi C. Naiman

Department of Computer Science
Hong Kong University of Science & Technology

Abstract

We are developing a suite of interactive modules for teaching Introductory Computer Graphics, using a library we have built to support such efforts. The modules provide explanations of fundamental concepts, tutorial demonstrations of basic techniques, and interactive control of the parameters used in and execution of high-level representations of elementary algorithms. The modules are used in laboratory sessions to supplement formal lectures. Here, we present our philosophy in building the library and modules, discuss the lessons we have learned in our initial implementation, and outline our future plans.

Introduction

Computer Graphics is, by its very nature, highly visual. To supplement traditional teaching aids such as whiteboards and overhead projectors, 35 mm slides and videotape segments can be used to illustrate relevant concepts. However, for students to appreciate fully both the algorithmic and visual aspects of the fundamental topics presented in an Introductory Computer Graphics course, it is usually necessary for them to implement the various algorithms introduced.

We are creating a suite of interactive modules that both present the fundamental computer graphics concepts and algorithms and allow students to manipulate the principal variables involved. This approach not only obviates the need for students to 'reinvent the wheel' with a lot of tedious programming, but also allows them to explore many more of the basic concepts than is possible to cover in great detail in class presentations or programming assignments.

Previous work has focused on using computer graphics to visualize algorithm execution [1, 5], assist the instructor in presenting in-class demonstrations [4], and provide hypermedia access to tutorials and demos [2, 3]. Our goal is to provide a self-contained, self-paced learning environment for students of computer graphics. Although our current use of the modules is supplementary to the material presented in formal lectures, we have built the modules with no assumption that the students have had any prior exposure to the concepts. In this way, a module is usable by those who have either missed or not understood the companion lecture, or are not even enrolled in the course. Around-the-clock access in our laboratories allows the students to repeat the module until they understand the material and provides a useful source for review (e.g., before an exam).

Because a module assumes no prior exposure to the topic presented, and because students may use the modules outside of scheduled lab times, progress through the module is deliberately sequential, ensuring that the concepts are presented in a logical order. A user can simply press the Cont(inue) button to step through the explanation and demonstration sections, until such time as interaction is allowed or required.

Modules

We are implementing the modules using the C programming language and the Silicon Graphics GL library. The target machines for our use are SGI Indigo XS24 Workstations. To provide for ease of porting to other hardware and software platforms, we have isolated all of the GL code within our library routines. The following modules have been implemented and are currently being used in our courses:

- Windows & Viewports (Fig. 1);
- 2D Transformations (Figs. 2-5);
- Bresenham Line Drawing (Fig. 6); and
- Raster Filling.

We are currently designing and implementing modules on Line & Polygon Clipping, Polygon Scan Conversion, Color Manipulation, and 3D Transformations.

Our modules contain the following components:

- descriptions of the basic concepts: e.g., transformation matrices, boundary- vs. interior-filling;
- demonstrations of the basic concepts: e.g., animation of elementary transformations tied to the matrix representations, the effect of resolution on line quality;
- exercises: e.g., after introducing rotation, the 2D Transformation module presents the current time and requires the user to apply the appropriate rotations to the hour-, minute- and second-hands (Fig. 4). Students must print out and submit the completed exercises;
- algorithm execution: e.g., the Bresenham Line Drawing module allows simultaneous execution of two Line Drawing algorithms, demonstrating the equivalence of the results but the difference in execution speed (Fig. 6); and
- interaction: e.g., after the basic concepts and exercises have been covered, the 2D Transformation module allows the user to create figures composed of simple geometric objects, applying up to 3 elementary transformations to each object (Fig. 5).

Although the user controls the progress from step to step through a module, the pacing and location of concept presentation and demonstration within a step is predetermined by the module designer. To focus the user's attention appropriately, a hand with an extended finger is used to guide viewing during animations (Fig. 1) and algorithm tracing (Fig. 6).

Use of animation and interaction extends the modules beyond being simply an electronic textbook. In particular, we want the students to be able to manipulate the variables in a demonstration, so that they can pose their own 'what if' questions and view the results. We have also incorporated features that are difficult or impossible to achieve in lectures or reading material. For example:

- Instead of having to choose between row- and column-vector representations of a point, the module on 2D Transformations supports both representations throughout, allowing the user to switch between them with a simple button press (Figs. 2 & 3).
- To demonstrate the difference between pre- and post-concatenation of transformation matrices, both World- and Relative-Coordinate System transformations are supported.
- To allow users to read the generic code for an algorithm as well as see the status of its variables, the Bresenham Line Drawing module provides a button for 'Variable / Value Mode' (Fig. 6).
- The text used in the modules is stored in external files, embedded with a simple formatting and color-specification language. This allows for easy updating of descriptions without the need to recompile a module, as well as the possibility of supporting multiple languages.

Library Routines

To support the building of interactive teaching modules, we have implemented over 100 support routines, in the following categories:

- window handling;
- manipulation of primitives (pixels, lines, grids, rectangles, circles, etc.);
- buttons (with callback functions);
- text display;
- event handling;
- pseudocode tracing; and
- application control.

Although some of the library routines have been designed specifically to make development of computer graphics modules easy, the intention is to support a wide variety of disciplines. A series of interactive modules previously developed for demonstrating aspects of the human visual system helped us keep a generic perspective in the library design.

To orient the novice module developer to the library's routines, a series of some 25 demos have been created, incrementally introducing the features of the library. Documentation outlining the concepts in each demo, suggested exercises for adapting the demos, and on-line manual pages for the library routines are also available.

Fourteen teams (totaling 52 students) in the current Introductory Computer Graphics course are using these demos and the accompanying documentation to learn how to use the library. For their course project, they must design and (partially) implement any of the course topics for which we have not already provided a module. This exercise will provide valuable feedback for improvements to the library routines and will generate useful ideas for future modules. We will report on the success of this experiment at a later date.

Lessons Learned

Feedback from the students using the modules already built and trying to build their own modules using our library has proved invaluable. Some of the lessons we have learned from our initial implementation include:

- Even with the pointing-hand to focus users' attention, it is inevitable that some information will be missed. It is essential to be able to repeat the most-recent step in a demo, as often as the user finds necessary to understand the concept presented. A more general facility for backing up through the demo would also be beneficial, but would be very costly to implement.

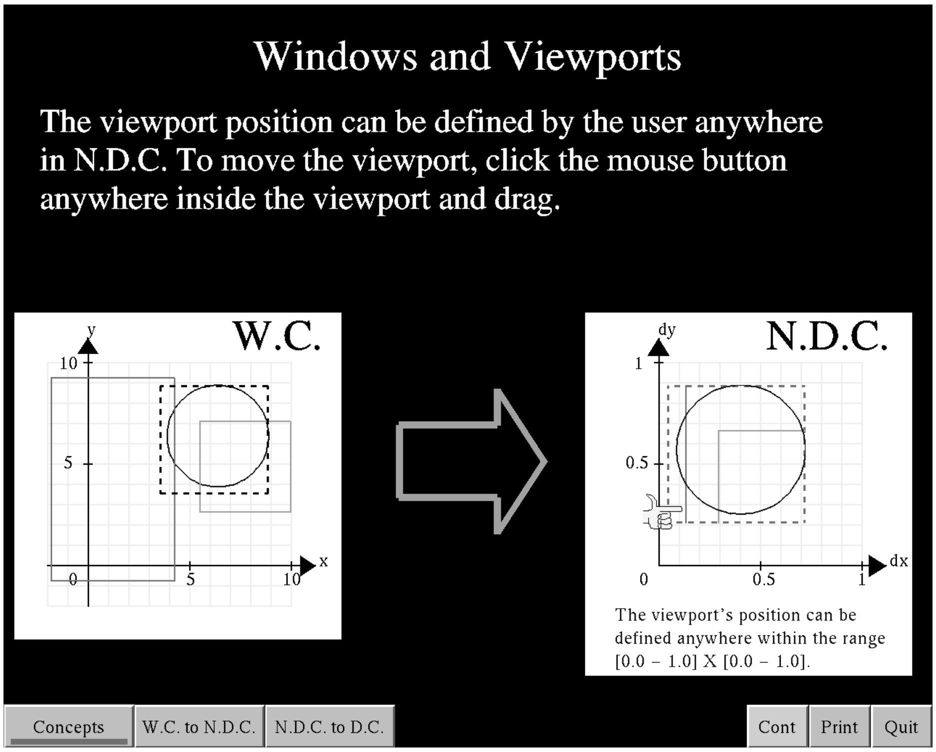

Figure 1.

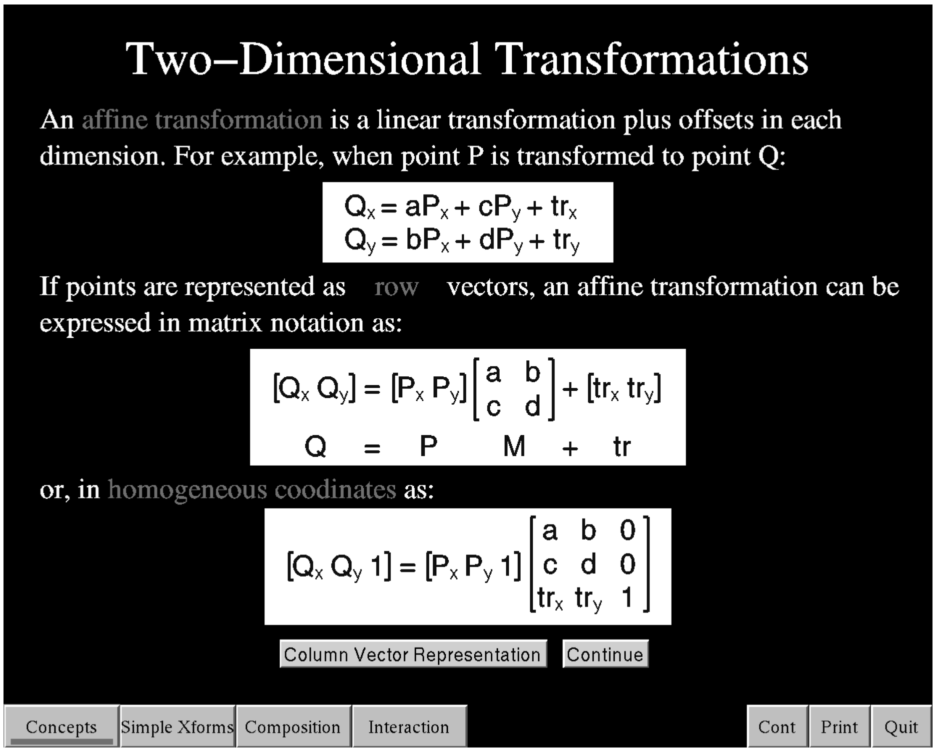

Figure 2.

- For communication in large groups, a mechanism is needed to uniquely identify individual frames or states of a module, analogous to page numbers in a textbook. A further feature would allow dynamic access through the module, thereby making it easy for students to quickly turn to the frame of interest. Currently, users can, at any time, skip to the beginning of a module's major sections by selecting from the buttons at the lower-left of the screen.
- Our original implementation of pseudocode tracing used highlighting of a line of code to indicate the next line to be executed. However, as there is some ambiguity as to whether the highlighted line has *already been* executed or is *about to be* executed, we instead place a pointing hand in between two lines of code, indicating where the simulation has stopped.
- When pseudocode tracing is in auto-play mode, progress through the algorithm is often too quick. Yet single stepping through the code is too tedious. A speed control mechanism during auto-play would be useful.
- To save file storage, network bandwidth and paper, an automated mechanism is needed for submitting the students' solutions to exercises directly to the instructor. Ideally, only the relevant state of the system would be submitted, and the system would allow the instructor to initialize the exercise portion of the module with the submitted state.

Future Plans

This first year of building the support library, implementing a few modules and having students use the library to implement their own modules has provided many insights. Our future plans include:

- To evaluate the efficacy of the modules, we will carry out user testing on the modules already built, administering a questionnaire and interviewing students about their experiences using the modules and soliciting suggestions for improvements to the support library. We will also videotape novices using the modules to determine problems with the system that students did not report.
- We plan to have a graphic designer review our implementation, making suggestions to help achieve better text clarity, color combinations, and control flow.
- We will implement the recommendations described above, such as 'repeat last step', numeric identification of frames, and electronic delivery of exercise results.

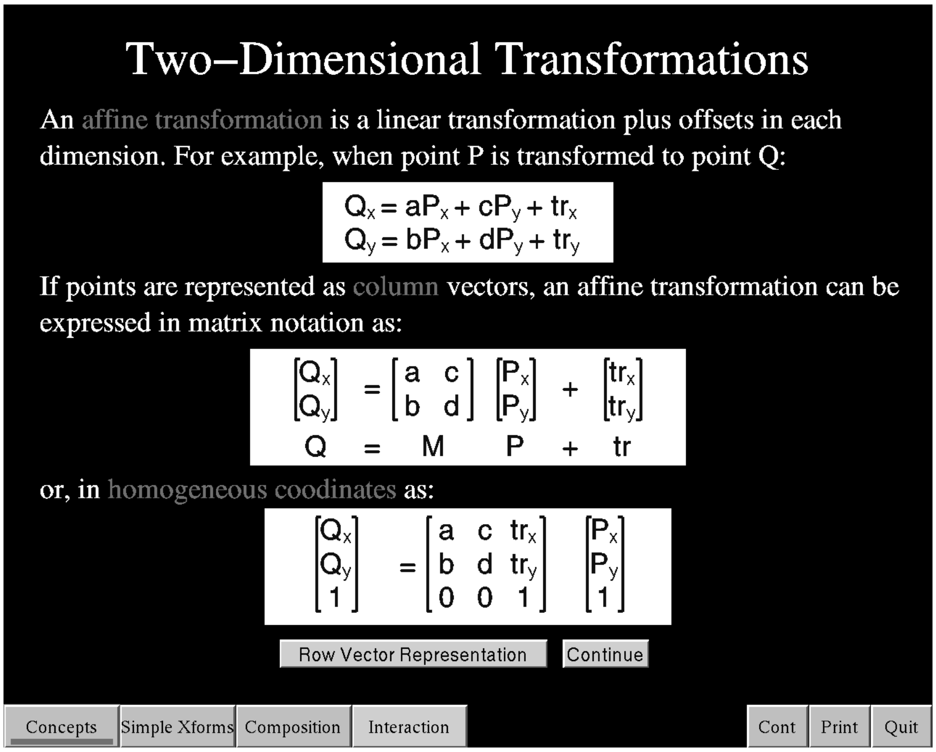

Figure 3.

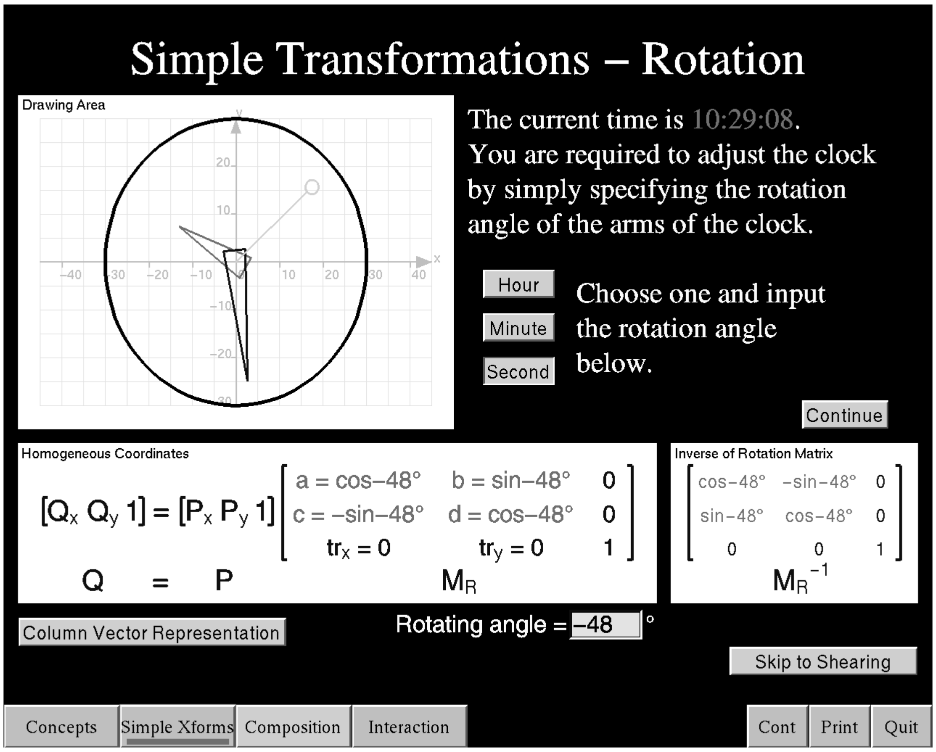

Figure 4.

Conclusions

We have implemented a set of library routines to support development of Interactive Teaching Modules as well as a few modules for Introductory Computer Graphics. The primary benefit to students is that, rather than learning from strictly textual material (perhaps supplemented with static imagery), they are exposed to dynamic tutorials of the concepts, which is essential for the comprehension of many visual phenomena. The dynamic nature of the modules increases student interest, attention and comprehension. The interactive nature of the modules allows students to explore concepts at their own pace and focus on areas of particular interest.

Acknowledgements

The work reported here was supported by a HKUST School of Engineering Multimedia Teaching Initiative Grant. I thank Nelson Mak, Gibson Lam, Kwok Ki Wan and Matthew Wong for their programming efforts on this project.

References

1. Marc H. Brown and Marc A. Najork, ‘‘Algorithm Animation Using 3D Interactive Graphics,’’ in *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology (UIST93)*, ACM, New York, November 1993, pp. 93-100.
2. G. Scott Owen, ‘‘HyperGraph: A Hypermedia System for Teaching Computer Graphics,’’ in *Interactive Learning Through Visualization: The Impact of Computer Graphics in Education*, Edited by Steve Cunningham and R. J. Hubbold, Springer-Verlag, Berlin, 1992, pp. 65-77.
3. G. Scott Owen, ‘‘Integrating World Wide Web Technology into Courses in Computer Graphics and Scientific Visualization,’’ *Computer Graphics*, Volume 29, Number 3, August 1995, pp. 12-14.
4. Dino Schweitzer and Bill Young, ‘‘Design and Integration of Visualization Tools for the Graphics Classroom,’’ *Computer Graphics*, Volume 29, Number 3, August 1995, pp. 9-11.
5. John T. Stasko, ‘‘Tango: A Framework and System for Algorithm Animation,’’ *IEEE Computer*, Volume 23, Number 9, September 1990, pp. 27-39.

Avi C. Naiman
Department of Computer Science
Hong Kong Univ. of Science & Technology
Clear Water Bay, Kowloon, Hong Kong
Tel: +852 2358-6997
Fax: +852 2358-1477
Email: avi@cs.ust.hk

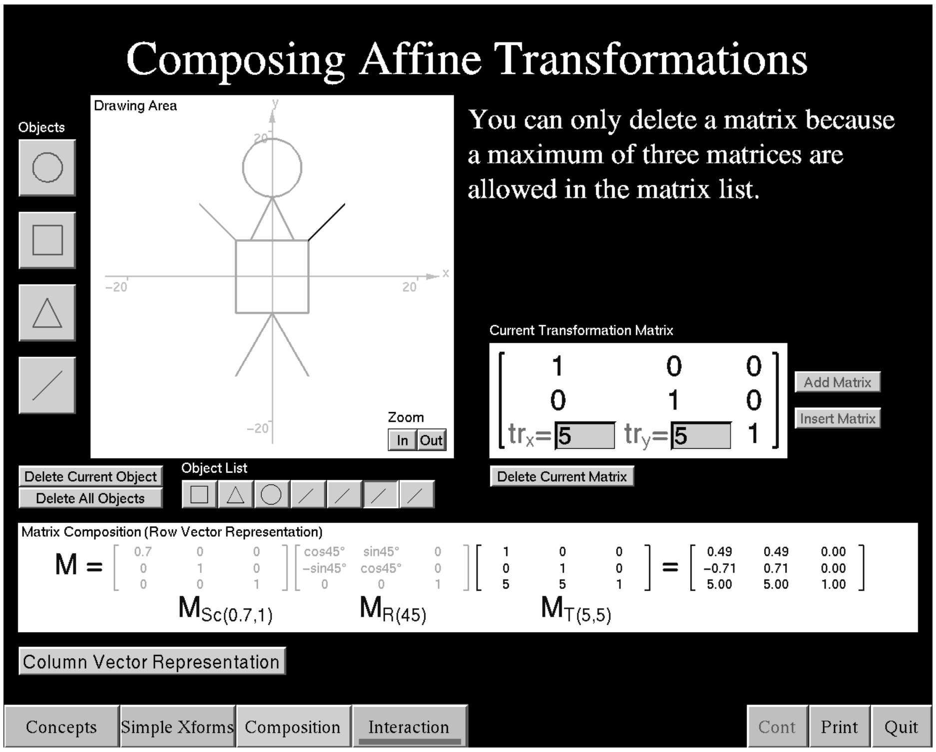

Figure 5.

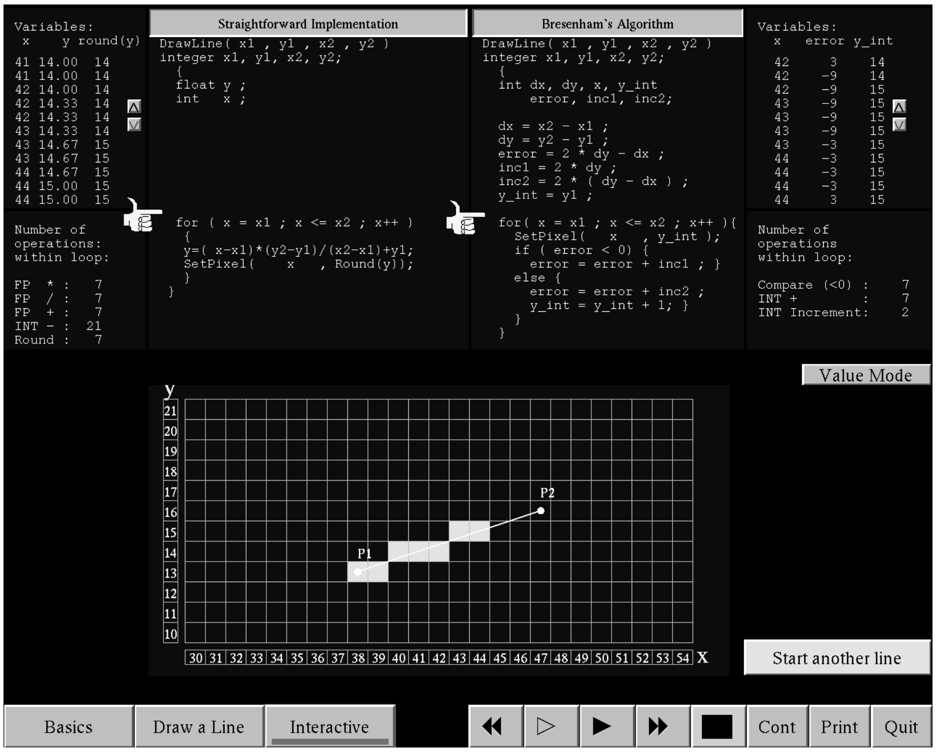

Figure 6.