Web-Based Teaching of Computer Graphics: Concepts and Realization of an Interactive Online Course

Reinhard Klein
Frank Hanisch
Wolfgang Straßer
Universität Tübingen

Abstract

Topics within computer graphics still cannot be adequately presented and explored with traditional teaching methodologies and tools. An integrative approach to combine lectures, examples, programming exercises, documentation, supported by a common sophisticated interface, is greatly needed. In this paper, we discuss and present the concept, realization, evaluation, and experiences of a computer graphics course that was developed at our lab to focus on this need. It is one of the first complete computer graphics course on the World Wide Web combining course text, programmed examples, and course exercises in a common hypertext framework.

Introduction

Visualization and interaction are the major topics of current computer graphics. Teaching these issues using only traditional teaching methodologies and tools, such as blackboards, slides, and even videos, cannot provide real-life examples. Only in real-life settings can the teaching react flexibly to the various questions that arise during discussions. As in physical research, problems and challenges in computer graphics can only be recognized through exploration in experimental situations. The experiments consist of programs.[19] Working with the programs not only helps to illustrate problems, but also increases student motivation. Furthermore, providing students with these programs enables them to repeat the experiments, build up their own settings, and deepen their understanding. This greatly improves learning.

In earlier times, such experiments were not included in traditional lessons. On the one hand, several different, expensive programs were necessary to cover the whole content of a computer graphics lecture, and, on the other hand, students didn't have appropriate hardware to run them at home. In addition, becoming familiar with all the different programs took too much time and could not be done in one or two semesters.

This also leads to serious problems with the practical exercises that accompany the lessons. A major goal of the exercises should be that students learn to implement the graphics algorithms discussed in the lecture. There were only two ways to achieve this goal: students built up their own graphics software systems from scratch or they used some of the existing program packages. Both were nearly impossible due to time constraints. The most common compromise was to present exercises that covered only a small amount of the teaching content.

A further major problem has been the number of different computing platforms and operating systems. It was nearly impossible to run the same program on all different platforms. Some of these ambiguities have been addressed by a number of colleagues working in the area of computer-supported education.[18, 2, 24, 12, 15, 17, 20] Many authors[23, 10] focus on the lack of a platform-independent graphics package that can be used as a basis for programming exercises. Fellner developed and used the object-oriented Minimal Rendering System (MRT) as a teaching and research platform for 3D image synthesis.[10] It is based on standard graphics packages like PHIGS, OpenGL, or XGL for graphics output.

Another approach is choosen by Wernert, [26 or 16] who describes a unified environment for presenting, developing, and analyzing graphics algorithms based upon IRIS Explorer, a modular visualization environment that provides a visual data flow language and allows users to link computational modules in order to create visualizations.

Lotufo and Jordan developed the well-known digital image processing system Khoros. A similar approach is described in.[16] This nice system applies high-level programming (network wiring) to more traditional coding of new modules. But there are also some drawbacks. First of all, IRIS Explorer is not platform-independent and is only available on high-end PCs and SGI workstations. Second, the application cannot be interlinked with a Web-based environment like our course. For students, even more interesting are the financial costs of a commercial system like IRIS Explorer, AVS or IBM Data Explorer.

In the meantime, the World Wide Web together with Java and Hypertext provide an appropriate framework to generate common interfaces for integration of all elements of interactive teaching courses, such as lectures, programs, exercises, and consolidating literature references. A certain number of computer graphics courses[4, 5, 14] already incorporate some of the ideas described above on the basis of HTML. The subsequent parts of this paper describe and evaluate the developed courses "Computer Graphik spielend lernen."[27] In the following section, we give a brief overview of the course, describe the Java-based programming environment and the compilation tools developed to integrate the different parts of the course, then consider how to generate reusable components using Java Beans. Finally, we conclude by reporting some experiences with the courses, describing our current work and giving a brief overview of our future activities.

The Course

Contents

The course contains the following topics: Computer graphics hardware, raster algorithms with aliasing and antialiasing, 3D transformations, visibility, color, local illumination schemes, modeling techniques, simple animation, texture mapping, global illumination techniques (ray-tracing, radiosity), basic image processing, and volume visualization. These topics are distributed through two courses. Both courses are based on, or related to, the books[8, 9] and corresponding scripts from the Fernuniversität (Correspondence University) in Hagen, Germany. The courses have been tailored to students of computer science and contain a variety of programming assignments.

In their current form, the courses are not well suited for engineering students: Such students prefer a course in which they can learn how to use commercially available packages instead of having to invest too much time in programming examples of basic computer graphics algorithms.

The programming exercises are done in groups of two or three students using the Java-based programming environment. The Java source code of the environment is available as a hyperlink and can be downloaded by the students. Comprehensive written instruction is available for each exercise in the form of HTML pages. In order to read it, students need to use a Web browser such as Netscape Navigator or Microsoft Internet Explorer. Since one of the aims of our course is to make the students familiar with graphics programming, it is important to provide them with sample programs to work with.

Structure

An HTML page provides a unified interface to our new Web-based computer graphics course. Starting at this page, one can follow links to HTML pages containing or referring to:

1. The instruction manual and editorial content of the course itself.
2. The script and list of available java-applets.
3. The programming exercises.
4. Links to external documentation and sources.

The instruction manual first gives a short introduction to the course and some hints for private studies, and provides a list of symbols used throughout the course. In addition, it gives a short overview of the design of the course, the programming architecture, and the file structures. Last but not least, it reports known bugs, such as the different behavior of certain applets on different browsers. This list of known bugs and comments can be extended by the user. External links provide tutorials for HTML and Java, public-domain software, other Web-based courses and further useful stuff.

The Script and the Java Applets

The entire course text is available in hypertext and presented in its own browser window. The contents and structure of the hypertext is the same as in the original course text. The hypertext contains not only cross references to figures, tables, literature, exercisesm and footnotes, but also links to corresponding Java applets and a number of videos and slides available via HTML that are shown during the lectures. If the user follows a link to an applet or a video, it is started in another browser window. The hypertext pages that contain the Java applets also provide links to the course text. In this way, the user working with an applet can immediately get the corresponding theoretical background. In addition, for each applet there is four-part documentation (cf. Fig. 1): An introduction, details on its features, a guided tour covering its essential topics, and general information on its programming architecture. Note that the course text, the applet, and its documentation are shown simultaneously in their own browser windows.

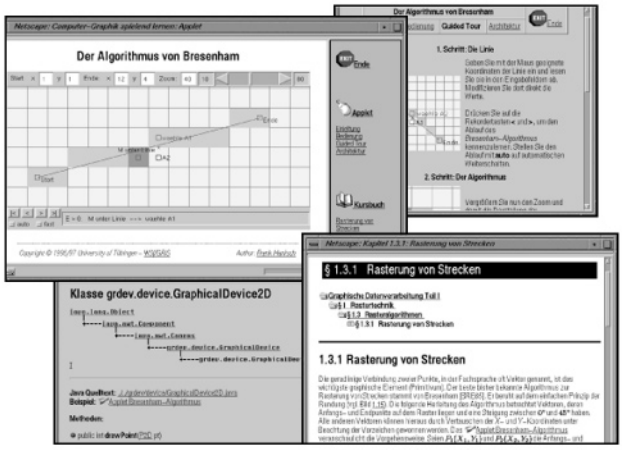

Figure 1 For each applet there is four-part documentation: An introduction, details on its features, a guided tour covering its essential topics, and general information on its programming architecture.

Both the course text and the hypertext pages containing the Java applets contain links to the Java API. In this way, students have direct access to example implementations of the presented algorithms. The classes of the Java API contain links to applets that demonstrate their usage in real examples. For certain classes, like Camera, there are also links to the corresponding course text.

The Programming Exercises

For each programming exercise, a small example of the source code can be downloaded, along with a short introduction to the topic. The source code of the exercises is also accessible through the Web, thus allowing for easy switching among the theoretical background information, the description of the exercises, and the source code of the latter. The aim is to support the students during completion of their programming tasks. Via hyperlinks to the course text in HTML format, including the Java applets, students can review the theoretical background for every single exercise.

Implementation

Generation of the Hypertext

The ingredients for the hypertext were the original course texts as LaTeX sources with images, the Java applets, and the exercises. In the first step, we manually generated an applet resource file containing all information necessary to automatically generate a hypertext environment for an applet. This includes the four-part documentation of the applet and keywords used to generate links into the course text. The actual pages containing HTML tags for the applet, including the header, the links to documentation, the applet, and the hyperlinks to the source text, are then automatically generated by a Perl script. This guarantees an easy-to-modify and unified outlook of all applets and saves an immense amount of time in page design.

After the individual applet pages are generated, the applet resource files are used by another Perl script to automatically generate a page that contains an index of all applets. In addition to links to the applets, this page contains the title, introduction, and a motivating image for each applet.

In an independent step, the original LaTeX source files were modified. First, links to the different applets were placed into the course text. Then anchor points named by the header itself were automatically inserted. These anchor points are used later as hyperlink anchor points and are referred by the java applets. The names of the anchor points are copied into the corresponding applet resource files. After these two steps, we used the program Latex2html[6] to convert the course text into HTML format, which still contains the unprocessed anchor tags. These tags are processed in a further step by an additional Perl script, which generates the special course text structure developed for our course. This structure allows for folding and unfolding the hierarchy of the chapters. Each text page on the lowest level of the hierarchy automatically links to the previous and next text page as well as to all predecessors in the hierarchy.

The Java Applets

We put special emphasis on a common, easy-to-use interface for the applets. This is especially important, as the whole course includes a variety of different topics and the content of the functionality of the applets differs greatly. Some of the techniques we use include: using the same colors for the same context, using the same labels and control elements with identical meanings, and the same mouse control. A clear structure for the visible information is very important for design of the applets for teaching purposes. At first glance, students must be able to recognize the topic of the applet and the key elements of the teaching content and the connections between them. Therefore, the visual part of the applet containing this information must attract their attention more than special control elements that steer certain parameters. Otherwise, the applet's forest of equivalent choices would overwhelm the students.

The Graphics Engine

The graphics engine plays the central role in implementation of all the applets. Previously, Java has provided the user only with a 2D graphics interface. For real 3D graphics programming, we implemented our own 3D graphics engine. It supports all 3D functionality used throughout the applets, such as cameras, shading techniques, 2D and 3D texture, and hidden-surface elimination. It is based on our own 2D engine, which extends the standard Java graphics engine. This was necessary to supply abstract data structures, like 2D grids, and abstract event handling, like special mouse interpretations, as well as totally new components such as highlight modes.

The Scene Graph

The scene graph was developed as a programming basis for two reasons. First. it should provide a rich set of features for creating interesting 3D worlds and provide a high-level, object-oriented, programming paradigm that enables us to rapidly deploy sophisticated applications and applets. Second, it should reflect the state of the art of current graphics APIs[21, 1, 25, 7] in order to make the students familiar with the newest concepts of graphics technology. The scene graph is an acyclic directed graph and contains a com-

plete description of the entire scene, or virtual universe. This includes the geometric data, the attribute information, and the viewing information required to render the scene from a particular point of view. The scene graph allows the programmer to think about geometric objects rather than about triangles, about the scene and its composition rather than about how to write the rendering code for efficiently displaying the scene. Applications construct individual graphic elements as separate objects and connect them together in the graph. If the Java3D[16] extension to Java is available, the basic part of our scene graph may be replaced by the corresponding components of Java3D, leaving most of our own parts intact.

Utilities

The basic mathematical and geometric material needed in most of the applets, like points, vectors, matrices, lines, circles, triangles, etc. is implemented as their own classes. This allows rapid implementation of even complicated mathematical and geometrical algorithms. Using these basic classes, the developer can concentrate on the essential parts of the algorithms rather than on basic mathematics. Furthermore, the resulting code for our applets becomes much easier to read and understand. This is especially necessary when motivating students to take a closer look onto the real implementation of an algorithm and try their own implementations.

The basic graphic components, like special interfaces for graphical in and output of the basic mathematical classes, the elements of scene graph, are realized in a similar way. A special component is a grouping panel to organize the contents of our course applets. Such components are used throughout the applets. Combining the standard Java graphics components with our own versions allows for rapid design of interfaces. Note that for all classes there exists hypertext-based documentation that can be invoked while browsing the source code and easily searched.

Reusable Components - Java Beans

Generating New Applets

The strict object-oriented design of our Java applets simplifies the programming of new applets and classes. Many of the existing classes can be reused and extended. Nevertheless, generation of new applets still requires low-level programming. Students or teachers not familiar with the existing classes have to wade through the jungle of hierarchical APIs instead of concentrating on the structure and concepts of the algorithms. As already shown by a number of authors, visual programming greatly aids in developing and debugging code.[26, 16]

The Programming Exercises

Since the low-level programming is too time-consuming without visual programming, our students only complete already existing applets. The drawback of this kind of programming exercises is that most of the students are not able to understand the overall structure of the program. Although they successfully implement the missing parts. there remains a bad feeling. Again, these problems can be solved using a visual application builder in combination with low-level programming and review of existing code.

The Beans

As stated in its specification,[11] a Java Bean is a reusable software component that can be manipulated visually in a builder tool. The components can be objects with or without graphical interfaces. In our course, typical beans are:

- Interface components that extend the standard Java AWT.
- Mathematical and geometrical utility components like vectors, matrices, triangles, spheres etc.
- 2D and 3D canvases that also extend the standard Java components.
- 2D and 3D scene graph for high-level graphics primitives and scene descriptions.
- Already-programmed “high-level” beans like image filters, function parsers, editable curves, etc.

Due to the strict object-oriented design of the Java classes used in the first version of the course, their conversion into Java beans was straightforward.

Examples

As in a visual data-flow language, beans can be visually composed into new customized applets. Figure 2 shows generation of a simple applet that demonstrates conversion of a gray-value image to a black-white image. The user can define the threshold value. To generate the applet, the image loader, two image beans for input and output image, the value panel, and the black-white filter are loaded into the application builder. In this example, the data flow is defined by so-called property-changed events. The property-changed event of the original picture (invoked by loading it) is propagated to the black-white filter that expects a reference to the input image and resolves a property-changed event with a reference to the output image. Last but not least, the property-changed event of the value panel changes propagates the threshold value to the black-white filter and invokes it. After the applet is designed in the application builder, it can be compiled into a new applet.

Figure 3 shows construction of a more complex applet demonstrating Bézier

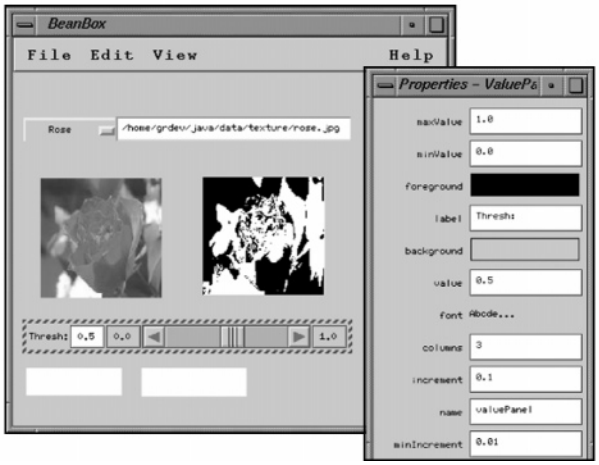

Figure 2 Building a simple applet demonstrating the black-white image filter in the Bean Box. Unfortunately, the current version of the Bean Box does not display the already-established links between the components.

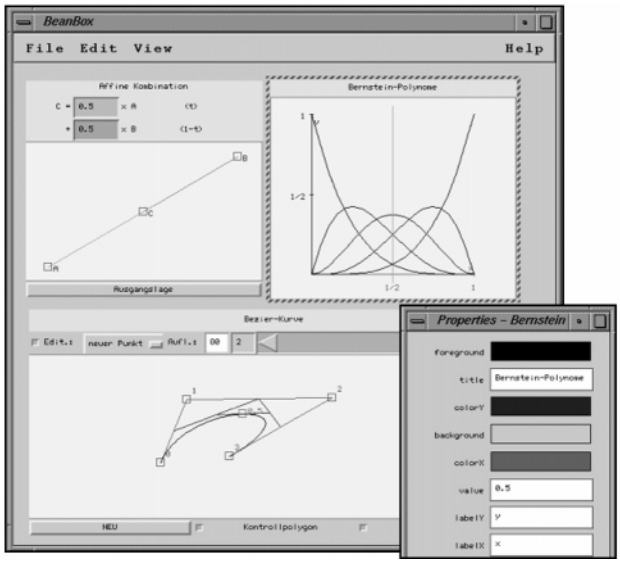

Figure 3 Composing three beans into a new applet demonstrating Bézier Curves together with affine combinations and Bernstein polynomials.

curves. Its main building blocks are three simpler beans, one realizing affine combinations between two points in 2D, one displaying the Bernstein polynomials, and one containing the Bézier curve itself. Each bean already implements all basic interactions to manipulate its contents. Several interface components link the beans.

In this way, the functionality of data-flow languages is available to build up new Java applications. Using this paradigm, teachers and students can easily develop their own new beans and integrate them into the course using existing components. Nevertheless, there are still cases, like the implementation of new filters, in which new beans must be implemented in the traditional way.

Discussion of Design Problems

One of the design problems of data-flow languages is how to manage the complexity of program networks. In our approach, this problem can (in most cases) be easily solved by grouping several beans into new ones and customizing their appearance within the builder tool. We made use of these techniques throughout the whole course. The problem that is inherent to most other systems based on the data-flow concept – propagating data through the network – does not occur, since different modules may share the same data in memory.

Experiences

Evaluation of the programming environment is based on two consecutive semesters of the computer graphics course offered in our department in 1995-96 and 1996-97. The complete first course consists of 28 hours of lectures and 26 hours of practical exercises; the second course consists of 24 hours of lectures and 20 hours of practical exercises. Forty-one students participated in the first course and 31 in the second. At the end of each course, the students were asked to fill out a questionnaire on their experiences with, and evaluation of, the complete course, the lectures, and the programming environment. The questions referred to issues like quality, effort with respect to results, learning success, and user content. The following trends could be traced:

- The overall content of the whole course was evaluated much higher than in the years before.

- The students themselves reported remarkable learning success.
- The complaints regarding the programming effort with respect to the outcome decreased.

We noticed that several students started to add non-requested features and functionalities to the system after they became familiar with it. These extensions referred to the user interface as well as the scene graph. Some students even implemented completely new applets in other topics in computer graphics that are not directly related to the central part of the lecture. This had never been the case before, and it strengthened our assumption that our concept would support and stimulate exploration and further development of the programming environment by the students.

Conclusions

Using our course, teachers and students can theoretically and practically prepare, catch up on, and deepen the course lectures. There is no need to provide for and install a variety of software packages to run the practical examples at home. Moreover, interested students are able to extend the examples. Due to the positive resonance in the course, we decided to refine and extend it to serve a larger community of people interested in computer graphics programming at universities, industry, and even high schools, and we are currently working on a translation of the whole course into English.

In addition to further development of our Java Beans toolkit, our future research in the area of graphics and visualization education has to consider appropriate user modeling, different levels of programming examples and even lessons, and, especially, different variants of course-related hypermedia user support.

Acknowledgments

We would like to thank A. Schilling for useful disscussions and G. Rössner and R. Schwering for their immense implementation efforts in realizing the system.

References

1 G. Bell, A. Parisi, and M. Pesce. The Virtual Reality Modeling Language: Version 1.0 Specification. URL: http://vrml.wired.com/vrml.tech/vrml10-3.html, 1995.

2 K. Brodlie, T. Hewitt, S. Larkin, P. Willis, and J. Gallop. Graphics and visualization - techniques and tools: A course for postgraduates of all disciplines. In 3., pages 263-268, 1994.

3 K. Brodlie and G. S. Owen, editors. Computer & Graphics, volume 18(3). Pergamon, May/June 1994. Special Issue on Computer Graphics in Education.

4 Course CPSC 453: Computer Graphics I.URL: http://www.cpsc.ucalgary.ca/localinterest/class info/453/, 1994. University of Calgary, Computer Scienc Dept.

5 Course Computer Science 417: Computer Graphics.URL: http://www.tc.cornell.edu/Visualization/Education/cs417/, 1996. Cornell University, Theory Center.

6 Nikos Drakos. The LaTeX to HTML translator. Internal report, Computer Based Learning Unit, University of Leeds, January 94.

7 George Eckel. Cosmo 3D Programmer's Guide. Silicon Graphics, Inc., 1997.

8 J. Encarnação, W. Straser, and R. Klein. Graphische Datenverarbeitung I. Oldenbourg, 4th edition, 1995.

9 J. Encarnação, W. Straser, and R. Klein. Graphische Datenverarbeitung II. Oldenbourg, 4th edition, 1997.

10 D. Fellner. MRT: A Teaching and Research Platform for 3D Image Synthesis. In 13., 1994.

11 Graham Hamilton. The javabeans api specification. Sun Microsystems, July 1997.

12 F. W. Jansen and P. R. van Nieuwenhuizen. Computer Graphics Education at Delft University of Technology. In 13., 1994.

13 L. Kjelldahl and J. C. Teixeira, editors. Eurographics Workshop on Graphics and Visualization Education (GVE), Oslo, Norway, 10-11 September. Eurographics, 1994.

14 Reinhard Klein and L. Miguel Encarnação. An interactive computer graphics theory and programming course for distance education on the Web. In 8th Int. PEG Conf:97, Sozopol, Bulgaria, May/June 1997.

15 B. R. Land. Teaching computer graphics and scientific visualization using the dataflow, block diagram language Data Explorer. In S. D. Franklin, A. R. Stubberud, and L. P. Wiedeman, editors, University education uses of visualization in scientific computing: proceedings of the IFIP WG 3.2 Working Conference on Visualization in Scientific Computing, Uses in University Education, Irvine, CA, USA, IFIP Transactions. A, Computer Science and Technology, pages 33-36, pub-NH:adr, July 1994. pub-NH.

16 Roberto Lotufo and Ramiro Jordan. Digital image processing with khoros 2.0. This is an interactive WWW course using the Khoros system, December 1994.

17 Avi C. Naiman. Interactive teaching modules for computer graphics. j-COMP-GRAPHICS, 30(3):33-35, August 1996.

18 G. S. Owen. HyperGraph - A Hypermedia System for Computer Graphics Education. In S. Cunningham and R. Hubbold, editors, Interactive Learning through Visualization, pages 65-77. Springer-Verlag, 1992.

19 G. S. Owen. Teaching Computer Graphics as an Experimental Science. In 13., 1994.

20 Amnon Shabo, Mark Guzdial, and John Stasko. Addressing student problems in learning computer graphics. j-COMP-GRAPHICS, 30(3):38-40, August 1996.

21 P.S. Strauss and R. Carey. An object-oriented 3d graphic toolkit. In Computer Graphics Siggraph 92, pages 341-349. ACM Siggraph, July 1992.

22 Sun. The JAVA 3D API. Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, CA 94303 USA., 1997.

23 J. C. Teixeira. Environments for Teaching Computer Graphics: An Experience. In 13., 1994.

24 J. C. Teixeira and J. S. Madeira. A Computer Graphics Curriculum at the University of Coimbra. In 3., pages 309-314, 1994.

25 N. Thompson. 3D Graphics Programming for Windows 95. Microsoft Press, 1996.

26 E. Wernert. A unified environment for presenting, developing and analyzing graphics algorithms. Computer Graphics, 31(3):26-28, August 1997.

27 Course Computer-Graphik spielend lernen. URL: http://www.gris.uniuebingen.de/gris/grdev/java/index.html, 1996/97. University of Tübingen, Interactive Graphics Systems Lab (WSI/GRIS).