Challenges in the Development of Web-based Courseware using Virtual Experiments

F. HANISCH°, R. KLEIN*

° Graphisch-Interaktive Systeme
Universität Tübingen, Auf der Morgenstelle 10 C9, 72076 Tübingen, Germany
Tel.: +49 7071 29 75463. Fax: +49 7071 29 5466. E-Mail: Frank.Hanisch@WSI-GRIS.Uni-Tuebingen.DE
* Graphische Datenverarbeitung
TU Darmstadt, Rundeturmstraße 6, 64283 Darmstadt, Germany
Tel.: +49 6151 155 655. Fax: +49 6151 155 139. E-Mail: Reinhard.Klein@informatik.tu-darmstadt.de

Abstract

The development of modern Web-based courseware is an expensive and complex task. To fulfill the demand for better understanding of complex algorithms and complicated mathematical relationships traditional presentations of teaching contents like textbooks, hypertext, video or audio can be enhanced with virtual experiments.
Based on former experiences with computer graphics courseware we describe basic requirements on authoring and programming tools that allow for easy enhancement of existing courses as well as a rapid development of further courses in different areas. Our main concern in the development of virtual experiments is to include two important teaching methods: proper visualization and interaction.

INTRODUCTION

The development of modern Web-based courseware in the area of computer graphics requires the inclusion of its major topics: visualization and interaction. Presenting these issues using only traditional teaching methodologies and tools, such as textbook, hypertext, video or audio cannot provide real-life examples. Virtual experiments [18] fulfill the demand for better understanding of complex algorithms and complicated mathematical relationships by providing a proper visualization and allowing to interact with the contents. Working with such an experiment not only helps to illustrate problems but also increases the motivation of the students and enables him to repeat certain aspects and to build up his own settings.

The resulting courseware should be platform-independent and represent a unified framework [2, 16, 17, 19]. Therefore the programming tools should be based on one common graphics package. Many authors developed own packages based on standard packages like PHIGS or OpenGL or used modular visualization environments like IRIS Explorer, AVS or IBM Data Explorer [8, 13, 24]. But these approaches either cannot be satisfactory embedded into a Web-based environment or are linked with financial costs.

A solution matching our desires is the combination of Java and HTML [3, 4, 12]. The only requirements for the resulting courseware are a standard Web-browser and a Java development kit; both are available free of charge. In the following section we first give a short review of our first Computer Graphics Course as it was developed from 1995 to 1997. Afterwards we describe the basic requirements on authoring and programming tools that allow for an easy enhancement of our existing course but also for a rapid development of further courses in different areas. We present several case studies and close with a brief conclusion.

OUR FIRST GRAPHICS COURSE

1. Course Structure

Our first Web-based Computer Graphics course [12] provides a unified interface integrating course-related instruction manuals, the script, virtual experiments, programming exercises and links to external material.

The instruction manual hints for private studies and gives an overview of the course design, the programming architecture, the file structures and known bugs.

The hypertextual script provides the theoretical background and contains cross references to figures, tables, literature, exercises, footnotes, virtual experiments, videos and slides that were shown during the lectures.

Vice versa the virtual experiments are embedded in hypertexts and provide links into the script. They also supply an introduction, details on their features, a guided tour covering its essential topics and general information on its programming architecture with links into the source code. In such a way the user can switch between theory, example implementations of the presented algorithms and the programming details.

Lastly we include programming exercises together with small example programs and links to the necessary theoretical or technical background information - since the students have to get familiar with graphics programming it is important to work with the underlying Java source code.

2. Contents

The course covers all standard topics of the computer graphics curriculum: Computer graphics hardware, raster algorithms with aliasing and anti-aliasing, 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.

The contents is based on, or related to, the books [6, 7] and corresponding scripts from the Fernuniversität in Hagen (Correspondence University), Germany, and is tailored mainly to students of computer science.

3. Virtual Experiments

Regarding the variety of different topics a common, easy-to-use interface of all virtual experiments is essential. An overall design of colors, labels and control elements allows the viewer to implicate an elements' meaning from its outlook. The visible information is structured clearly as the student should recognize the topic, the key elements of the teaching content and the connectivity between the elements at one glance (Fig. 1).

The standard Java graphics engine was extended to our own 2D and 3D engines that supports all needed data structures and functionality. They do not require a browser plug-in and allow studying every single detail in the source code.

To enable us to rapidly deploy sophisticated virtual experiments by thinking about geometric objects and their composition rather than about complex rendering details we implemented a scene graph, a high-level programming paradigm of current graphics API [5, 20, 23] including Java3D [21]. A scene graph contains a complete description of the entire scene with its geometric data, attributes and viewing information. Individual graphics elements are constructed as separate objects and connected together in the graph. The basic part of our scene graph may be replaced by the corresponding components of Java3D, leaving most of our own parts intact. Similarly we implemented basic mathematical and geometric utilities like vectors, matrices, and triangles as own classes. With this, the developer can concentrate on the essential parts of the algorithms.

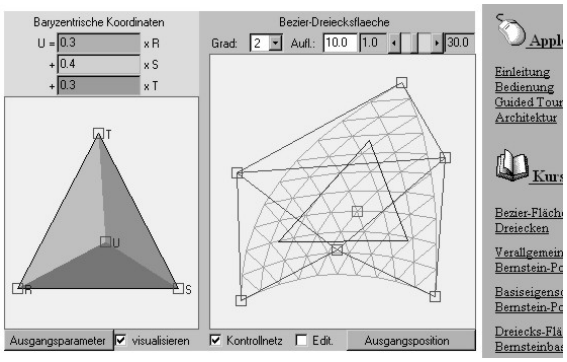

Fig. 1: This virtual experiment teaches the key elements of bézier triangle patches. The experiment is completely cross-linked with its documentation, source code and the script.

BASIC REQUIREMENTS

1. Authoring Tool

1.1 Separation of structure, content and design

Based on the experience with our courseware development we found that strictly separating between structure, content and design is a key towards easy modification and extension of an existing environment. It further allows to reuse course material and to manage multiple working groups. Separation of structure and content is well known in courseware generation [14]. Beyond this the design must also be described independently in order to quickly modify also the concrete appearance of structure and content, like colors, fonts, pictograms and layout information.

1.2 Organization of data using database concepts

Separation between structure, content and design is one of the key ideas in common database technology. Therefore, including well-known database concepts offers the opportunity for using advanced functionality for the organization of the core data. Furthermore, high-level interfaces like JDBC allow a flexible access to the core data within complex tasks like the organization of consistent links. As databases differ (for example in object-types and maximum allowed string-length) and data could be spread over many databases we developed an abstract database manager that interacts with the underlying databases.

To quickly create contents for multiple languages (internationalization) we suggest a layered database model [10]. Each supported language corresponds with a layer and all layers are stacked with the initial supported language as lowest layer. The abstract database manager tries to get the requested information for the current language and automatically moves down to the next layer if the data is not available on the corresponding layer (Fig. 2, left side).

This model facilitates the author's work in internationalization. He now may concentrate on a single language and stepwise include more translations. Language-independent data like structure, images or videos without text and audio are stored on the lowest layer only. Another problem solved by this model is to supply multiple levels of detail, used for instance in user knowledge adaption. Fig. 2 demonstrates how multiple layers for the same language realize different information depth for novices, average and expert users.

1.3 Architecture of an hypermedial authoring tool

The described requirements for an hypermedial authoring tool leads to the architecture outlined in Figure 2 (right side) that strictly separates structure, contents and design and organizes its data with an abstract database manager [10].

The authoring tool's interface integrates editors that allow to easy modify structure, contents and design as well as an extendable design template technology that describes the appearance of all kinds of elements, like document chapters, links, images, experiments or even didactical elements like multiple choice questions. To automatically generate the courseware out from its content, structure and design information a generator component is needed.

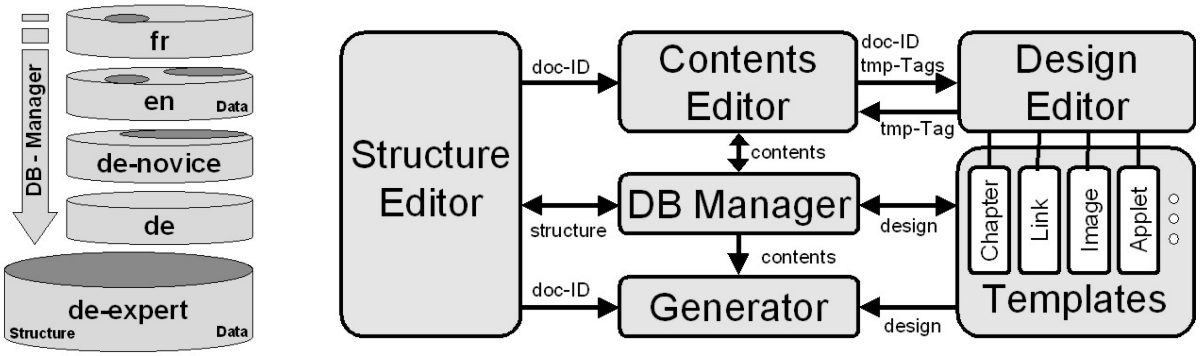

Fig 2: The requirements for data organization and the architecture of an hypermedial authoring tool lead to a layered database model (left side) and a strict separation of structure, contents and design (right side).

The main advantage of the suggested architecture is that all components, the diverse editors, the database, the generator and the templates are independent and exchangeable; for example the contents editor could be represented by an arbitrary text editor wrapped into a Java class or the generator could realize hypertext, slides, PostScript or PDF.

2. Programming Tools

2.1 Reusable software components

The manifold topics we covered with our virtual experiments required a strict object-oriented and component-based design (JavaBeans, [9]) of the underlying programming tools. Many classes can be extended, reused and visually programmed in a builder tool. Although the current specification of JavaBeans does not include a standard for semantic information and therefore complicates the inclusion of other peoples' components [15], visual programming greatly aids in developing and debugging code [22]. With components, students or teachers may concentrate on the structure and concepts of the algorithms even if they're not familiar with the programming interface. Nevertheless, completely new functionality still requires low-level programming.

2.2 Architecture of the virtual experiments

To implement the complex algorithms and complicated mathematical relationships of the virtual experiments we used a data-flow message concept. Components register as listeners to other components' property changes or events. Modifications may be vetoable, for instance if there is any constraint to fulfill like a point lying on a curve.

Often used visualizations are implemented as scene graph components with standard input and output properties or events. In such a way the left triangle in Fig. 1 allows the modification of its points and messages the resulting barycentric coordinates to the right triangle mesh.

Basic actions are defined in the base scene graph node. This allows for example picking and dragging of all objects. Furthermore, another component may listen to this action and react with another action, like e.g. a suitable translation, rotation or zoom.

2.3 Identification of basic components

As the goal of our virtual experiments is to visualize algorithms and interact with parameters of the algorithms, we identified basic components for data structures, 2D and 3D geometry, images and physical quantities (Fig. 3).

The virtual experiment in Fig. 4 serves as an illustration on how we visualized the idea of raycasting. Several scene graph components implement the camera, the screen (a 1D pixel array), the objects (two circles), a point light, grid and canvas. Using the standard dragging behavior all parameters can be visually modified. The screen component listens to camera modifications and automatically adjusts its parameters. Similarly, the circles listen to the camera ray property and send the intersection information to the light source that calculates the lighting model. Finally the screen component colors the pixel defined by the ray with the value calculated by the light source. All that is left to do for a proper visualization is to animate the camera ray property by looping through all values.

Basic Components	Examples
Data structures	List, heap, graph, tree, ...
2D and 3D geometry	Camera, screen, grid, ...
Images	Viewers, filters, ...
Physical quantities	Scalar, vector, field, light, ...

Fig. 3: Identification of basic components for the visualization of algorithms and for providing interaction with their parameters.

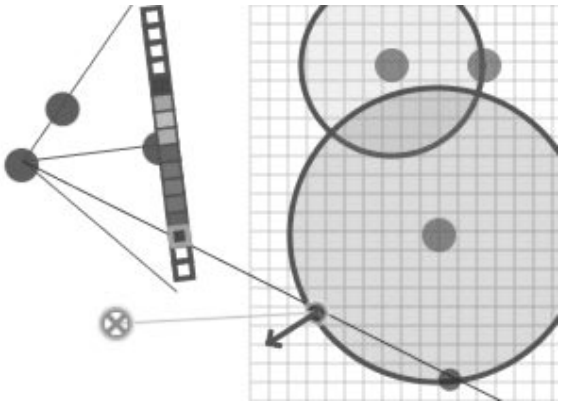

Fig. 4: This visualization of raycasting was created by connecting basic components. All visible parameters can be modified interactively.

CASE STUDIES

1. Electronic Webmaster

Using the described authoring tool the task to implement an Electronic Webmaster for the creation and maintenance of our institute's homepage was straightforward.

We designed several templates for the institute-specific outlook and for all its parts: staff, areas of research, projects, seminars, vacancies, gallery and bibliography. For each person a personal page is automatically generated including links to all other parts that are related to him, including his personal bibliography with accessible papers and last but not least a printable business card.

Most of the core data could be imported from other sources; only the program to import BibTeX entries into the bibliography made some effort.

To allow online modification of the data a dynamic server-side program was developed that implements an authentication mechanism. In case of a modification the Webmaster is notified by e-mail, and, if he accepts the changes, the generator updates the homepage automatically.

2. Interactive Image Processing Course

A small coursework for the lecture *Image Processing* held in 1999 at our institute demonstrates the strength of component-based programming [12].

During the implementation of the virtual experiments we frequently reused components of our first computer graphics course. The courseware integrates experiments for a standard course on image processing, that is: image histograms and their modifications, image operations and blending, discrete Fourier transformation, convolution kernels, image correction and reconstruction, decoding and encoding of image formats, edge detection and image warping.

The main goal was do teach image algorithms. As most of them are based on filter chains the student had to solve structural lessons by visually compose basic components to a filter chain. To illustrate low-level programming, some components were given as exercise, in which the student had to face with complicated details like serving different image formats or inventing tricky solutions of efficient image loops.

3. Visualization in Bioinformatics

Currently, we are working on a courseware for Scientific Visualization in Bioinformatics.

To improve self-learning aspects we integrated new templates for gap-filling or multiple-choice tests, automated indices of all keywords, illustrations, literature and a glossary.

Since most of the work was already done in other components, tasks like these only require some design instructions and, of course, the input of the core data. Also, many new virtual experiments have to be integrated. We will evaluate a first version of this courseware in the summer of 2000.

CONCLUSION

In this paper we presented some concepts for the design and implementation of modern Web-based courseware. We described a general architecture that strictly separates structure, contents and design. For the organization of the core data currently available databases in combination with a high-level programming interface are used. We presented a layered database model that simplifies internationalization and multiple levels of detail. The generation step is performed automatically by using design templates.

The programming of our interactive components is based on reusable software components that are connected with a message concept. Basic visualizations and actions are implemented as scene graph objects. We therefore identified several basic components that can be composed to virtual experiments.

Our main concern was to enhance traditional presentations of teaching content like textbook, hypertext, video, and audio with virtual experiments in a unified Web-based hypermedial environment. The virtual experiments we built suit the demand for better understanding of complex algorithms or complicated mathematical relationships using two important teaching methods: proper visualization and interaction. Like proper visualization tells us more than thousand words experience through interaction tells us more than thousand images.

References

[1] G. BELL, A. PARISI, 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, J. GALLOP, "Graphics and visualization – techniques and tools: A course for postgraduates of all discipline", Computer & Graphics 18(3): 263–268, Pergamon, 1994.

[3] COURSE CPSC 453, "Computer Graphics I", URL: http://www.cpsc.ucalgary.ca/local_interest/class_info/453/, University of Calgary, Computer Science Dept., 1994.

[4] COURSE COMPUTER SCIENCE 417, "Computer Graphics", URL: http://www.tc.cornell.edu/Visualisation/Education/cs417/, Cornell University, Theory Center, 1996.

[5] G. ECKEL, "Cosmo 3D Programmer's Guide", Silicon Graphics, Inc., 1997.

[6] J. ENCARNAÇÃO, W. STRASSER, R. KLEIN, "Graphische Datenverarbeitung I". Oldenbourg, 4th edition, 1995.

[7] J. ENCARNAÇÃO, W. STRASSER, R. KLEIN, "Graphische Datenverarbeitung II". Oldenbourg, 4th edition, 1997.

[8] D. FELLNER, "MRT: A Teaching and Research Platform for 3D Image Synthesis", Eurographics Workshop on GVE, September 1994.

[9] G. HAMILTON, "The JavaBeans API Specification", Sun Microsystems, July 1997.

[10] F. HANISCH, "Aufbau eines hypermedialen Autorenwerkzeugs für interaktive Web-basierte Lernumgebungen in der Computergraphik", Technical Report WSI-2000-5, University of Tübingen, Germany, February 2000

[11] F. HANISCH, R. KLEIN, W. STRASSER, "Ein Web-basierter Computergraphik-Kurs im Baukastensystem", Bildverarbeitung: http://www.gris.uni-tuebingen.de/study/bv/ss99, in: Martin Engelien and Jens Homann (eds), Virtuelle Organisation und Neue Medien - Workshop GeNeMe99, Eul-Verlag, Lohmar, 1999

[12] R. KLEIN, F. HANISCH, W. STRASSER, "Web based Teaching of Computer Graphics: Concepts and Realization of an Interactive Online Course", http://www.gris.uni-tuebingen.de/projects/grdev, in: M. Cohen, SIGGRAPH 98 Conference Proceedings, Addison Wesley, 1998

[13] B. R. LAND, "Teaching Computer Graphics and Scientific Visualization Using the Dataflow, Block Diagram Language Data Explorer", in: S. D. Franklin, A. R. Stubberud, L. P. Wiedeman (eds.), "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, Computer Science and Technology, pp. 33–36, July 1994

[14] M. MENGEL, "IDEALS – task-oriented authoring for learning and training used in SMEs", Abstract of Online Educa 1998, Berlin, 1998.

[15] K. MEISSNER, F. WEHNER, "Ein Dokumentenmodell für Kursdokumente in Webbasierten Virtuellen Lernumgebungen", in: Martin Engelien, Jens Homann (eds), Virtuelle Organisation und Neue Medien - Workshop GeNeMe99, Eul-Verlag, Lohmar, 1999

[16] A. C. NAIMAN, "Interactive teaching modules for computer graphics", j-COMP-GRAPHICS, 30(3) pp. 33–35, August 1996.

[17] G. S. OWEN, "HyperGraph – A Hypermedia System for Computer Graphics Education", in S. Cunningham and R. Hubbold (eds), Interactive Learning through Visualization, pages 65–77, Springer-Verlag, 1992.

[18] G. S. OWEN, "Teaching Computer Graphics as an Experimental Science", Eurographics Workshop on GVE, 1994.

[19] A. SHABO, M. GUZDIAL, J. STASKO. "Addressing student problems in learning computer graphics". j-COMP-GRAPHICS, 30(3), pp. 38–40, August 1996.

[20] P. S. STRAUSS, R. CAREY, "An object-oriented 3d graphic toolkit". Computer Graphics Siggraph 92, pp. 341–349, July 1992.

[21] SUN MICROSYSTEMS, "Java 3D API Specification, Version 1.1.2", June 1999.

[22] J. C. TEIXEIRA, "Environments for Teaching Computer Graphics: An Experience", in: Eurographics Workshop on GVE, September 1994.

[23] N. THOMPSON, "3D Graphics Programming for Windows 95", Microsoft Press, 1996.

[24] E. WERNERT, "A unified environment for presenting, developing and analyzing graphics algorithms", Computer Graphics, 31(3), pp. 26f, 1997.