# Live graphics gems as a way to raise repositories for computer graphics education

Johannes Görke*
WSI/GRIS University of Tübingen

Frank Hanisch†
WSI/GRIS University of Tübingen

Wolfgang Straßer‡
WSI/GRIS University of Tübingen

## Abstract

This contribution presents two technologies, code-based interactivity and server-side compiling, that add value to an educational repository and address its difficulties in achieving a critical mass of submissions. It is part of current efforts in the computer graphics community to collect, preserve, and share educational material. We propose a novel repository service introducing "live" graphics gems to the material, i.e. modifiable program listings with corresponding interactivities that are compiled and versioned server-side. That way, repository material evolves on the fly. Authors not willing to submit their material as open source are given a granular source control. We describe implications, e.g. a common, repository-driven content markup, didactics of code-based interactivity, and crucial technical services and tools. Prototype implementations and showcases are included.

**Keywords:** educational repository, communities, interactivity

## 1 Introduction

The computer graphics (CG) educational community has originated the Computer Graphics Educational Material Source (CGEMS), a worldwide refereed repository for sharing educational contents in CG-related curricula. It was presented to a major public in 2004 [Figueiredo et al. 2004] and now calls for submissions in order to establish the repository's user community. In this contribution, we develop a pair of technologies to attract authors, code-based interactivity and server-side compiling, and present prototype implementations.

The difficulty in raising a repository is the critical mass of submissions it has to achieve in order to become valuable. Currently, the only reward for CGEMS authors to carry out the painstaking process of modularizing educational material [Spalter and van Dam 2003] is peer recognition. It would be more sound for authors to contribute to the repository if they would be given an added value besides credit, for example a significant speedup in authoring and facilities to convert their work to a variety of forms. We summarize the current status of CGEMS and investigate other repositories' concepts in Section 2. In particular, we describe how simple XML-based markup and transformations would form a proper base for authoring, interlinking, and publishing educational material with mathematical and graphical-interactive content. As a result, the repository would supply authors with valid templates that accelerate development and production of courses, modules, and teaching gems.

Many valuable materials exist and are ready-to-use. In that case, authors will accept an additional markup only if the repository returns considerable, didactic improvements. A possible starting point is the fact that in CG-related domains programming effectively deepens the understanding of theory, but is always carried out separately. Therefore, we have developed a new didactical element, code-based interactivity, which introduces "live" Graphics Gems to the material. Our approach essentially adopts the concept of combining the advantages of textbooks, technical journals, source code implementation, and discussion boards (as argumented e.g. in Heckbert's foreword to Graphics Gems IV [Heckbert 1994]); however, for the first time source code is integrated directly into the material, and reloaded dynamically into the corresponding visual or interactivity. We motivate the didactics of code-based interactivity in Section 3.

Technically, the repository provides an additional server-side compiling service. We discuss architectural requirements for repositories with such a service in Section 4, and present a prototype implementation. In that context, we broaden the denotation of "source" to media other than software, e.g. text sources, XML-based graphics, and Flash objects. Our approach allows for a new "partly open source" licensing model, which can be applied for authors that would otherwise not submit their work due to reasons of copyright or privacy. The model presents only the source for a desired learning objective, and blends out any other material's source. It supports privacy at a maximum granularity, e.g. contributing no source at all except for a single algorithm.

We illustrate our ideas in Section 5 with material teaching signal processing. Our prototype repository is available online at http://www.gris.uni-tuebingen.de/gris/ilo and offers a guest account for experiencing code-based interactivity embedded into theory together with server-side compiling.

## 2 Related Work

The CGEMS project [Assunção et al. 2002] arose from fruitful discussions around, during, and after the Eurographics/SIGGRAPH workshops on Computer Graphics Education in Bristol (CGE'02) and Hangzhou (CGE'04), from an idea that originated in Graphics and Visualization Education Workshop [Cunningham 2000] in Coimbra (GVE'99). The current server version supports online management of reviewing and publishing workflows [Figueiredo et al. 2004]. While the journal-like publication model aims at leveraging content quality and peer recognition, the planned peer collaboration functionality will support user feedback and versions.

The publishing and reviewing processes of educational material differ from the ones of research papers. The issue is known and is handled by different review forms and after-publishing processes [Knox et al. 1999]; in short, the after-life of published educational material in classroom and other scenarios implicates that the repository must not freeze submissions into final versions, but support its community in evolving the material. Educators usually adapt the material's design, language, and terminology to their specific needs, and modify parts of the content, e.g. by replacing some slides, examples, or exercises. Even more important with respect to the anticipated use by educators, technical updates are required as software environments change constantly. At CGE'04, we therefore discussed the CGEMS peer-reviewing process for material versions.

*e-mail: goerke@gris.uni-tuebingen.de
†e-mail: hanisch@gris.uni-tuebingen.de
‡e-mail: strasser@gris.uni-tuebingen.de

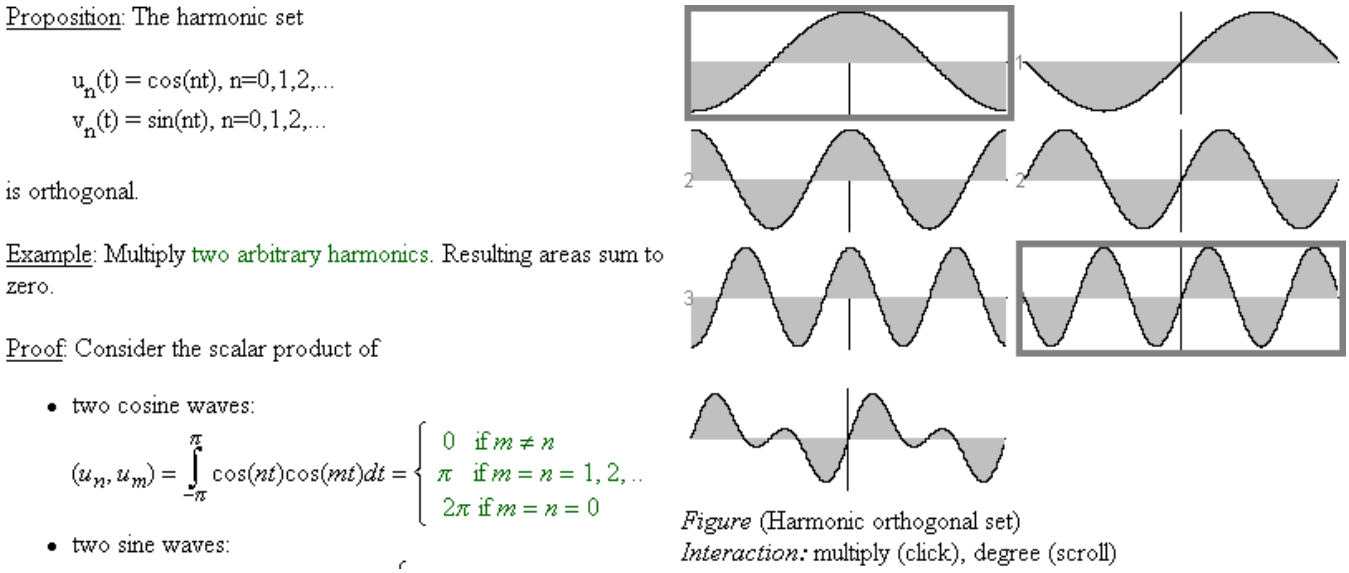


Figure 1: A visual proof for the orthogonality of harmonic sinusoids. Users may multiply specific waves by scripting the interactivity from theory. Here, the highlighted sentence 'two arbitrary harmonics' and the three rows in the formula resolve corresponding scripts. Theory, including mathematics, therefore provides a common content markup (XML/MathML).

The use of a coherent XML schema for content markup is exemplified, e.g. by Connexions (http://cnx.rice.edu, [Baraniuk et al. 2004]), a collaborative module repository launched in 1999 at Rice University. Here, we see some state-of-the-art concepts realized that let the repository grow to about 2200 freely available, educational modules in diversified domains. The Connexions markup language (CNML) defines a common ground for authoring, instead of letting each author choose the markup language. Currently, no standard markup of educational material exists, and considering the 400 DocBook [Walsh and Muellner 1999] elements, many authors will choose to define their own markup scheme (XSD) for organizing content (abstract, sections, theorem, proof, example, equations, programs, etc.), knowledge (concept, skill, etc.), and didactics (exercise, homework, test, etc.). Without a shared markup language (or at set of sanctioned languages with matching transformations) the repository would have no means for processing its material. Different modules could hardly be interlinked and sequenced. Connexions in turn may switch material presentation or publishing format simply by switching appropriate styles (XSL) and transformations (XSLT). Note that DocBook compatibility should be considered, as this automatically makes a large set of transformations to Web pages, PDF, slides and RTF (Word) available. Besides providing authors a rapid development process and a variety of publishing formats, the Connexions repository offers services like a course composer, annotation and printing facilities, and a browser plug-in superseding hyperlink navigation by user-defined sequencing and navigation, which enables users to define and follow paths through the material, i.e. courses and curricula.

A second line of tactic in getting authors to contribute to the repository becomes evident when we take into account the characteristics of educational material specific in the CG domain. State-of-the-art markup covers, at best, accurate rendering of formula (MathML), graphics (SVG, X3D), and parameter interactions that trigger computations (MeML [Wang et al. 2003]). Direct manipulation of graphical content, either view or model, cannot be described. Instead, proprietary scripting instructions are embedded into XML documents to connect text and formula with visuals and interactivities. Scripting is mostly used for illustration of specific parameter setups, but hardly ever rearranges inner parts of educational material or transfers them to other material. A repository might express such cross-material actions by visual scripting [Hanisch and Straßer 2003a]. Deeper modifications are performed in the material's source, e.g. in Java or C++ source files and corresponding makefiles and compilers, but also in the case of text, graphics, and slides (e.g. production formats LaTeX, Word, MathPage, Photoshop, PowerPoint, and Flash FLA versus publishing formats PDF, DHTML, and Flash SWF). Repositories like CGEMS or Merlot [Smith-Gratto et al. 2002] consequently include the material's source - as an option, as some authors might not be appealed with or simply are not be allowed to distribute the material's source. However, opposing users with source code and a makefile introduces additional technical hurdles and, in many times, requires users to install additional software packages. The proposed server-side compiling directly integrates into other repository services [Owen et al. 2000] like versioning, annotations, and reviewing, and can be implemented, e.g. as CGI, servlet, or Web service.

Our proposal transforms the material's source into a didactic element. In the case of software material, we result in code-based interactivity, which can directly be integrated into theory and discussion. In that way, it extends Glassner's idea of Graphics Gems (http://www.graphicsgems.org, [Glassner 1990]), which was founded 1990 and later continued as the ACM Journal of graphics tools. While each graphics gem represents a separate package with sources, makefile, and installation help, we merged both educational and code repositories, which enables the embedding of modifiable source code into modules, compiled server-side and requiring only browser functionality.

Our show case was inspired by Brown's Exploratories [Laleuf and Spalter 2001] and Tübingen's applet-based CG course [Klein et al. 1998]. We have developed interactive materials that recreate the look of a classic textbook, but make the best possible use of interactivity. Figures are interactive illustrations [Beall and Hughes 1996], i.e. software elements without buttons, input fields, or other widgets. Parameters are directly manipulative and further functionality can be triggered from theory or programmed at place. Didactics support visual cognition and offer visual proofs [Brown 2002].

## 3 Code-based interactivity

Code-based interactivity features three educational concepts, a (1) visual or interactivity illustrating (2) theory with (3) real source code. It turns the ”textbook with program listings” approach to dynamic Web pages representing program listings as an interactive editor, and reacts to user input by compiling the visual or interactivity server-side. In the following, we describe didactics of code-based interactivity; the technical framework is given in Section 5.

The idea behind a textbook with program listings is to enrich the high-quality explanation of an algorithm with an implementation [Heckbert 1994]. This is true especially for CG-related domains, where many users tackle with getting pseudo-code description actually to work. Research papers focus theoretic aspects and leave out ”easy” programming code, i.e. practical details a user must face. Textbooks and online journals therefore provide an additional Web page with discussion boards and user annotations, and there are many other, unofficial boards for source code discussions. Yet, they remain unrefereed, and are not interlinked with theory.

At first, embedding code-based interactivity into theory introduces an interactive element. Users experience the actual implementation of an algorithm, and may manipulate essential parameters. Sophisticated interactivities (microworlds [Papert 1980]) further allow for modifying the underlying model, that is, internal parts and structure. The interlinking of interactivities and theory can be performed by scripting. Consider, for instance, the interactivity in Figure 1, which serves as visual proof for some mathematical theorem. We have modified the traditional didactics, a theorem-proof-example sequence, by replacing the example with an interactive illustration. Further, we have provided proper XML-based markup for the theory, including the mathematical equations (MathML). Users may now select the theorem and specific parts of the equations to rearrange the interactivity and illustrate the described concepts visually. Educators may use such a module to build a visual cognition of the learning objective first, and then provide the correct proof.

Secondly, material now includes real source code. As we will demonstrate in our showcases, material can be presented as standard textbook with program listings. The only difference is that users can manipulate the program listing, and so the interactivity, at place. Authors therefore markup source code, which is similar to other content markup. Normally, only parts of the source will be included into a teaching module; parts that are useful for some cognitive or technical reason. Further code can, e.g. be provided on demand. Authors may declare their interactivity as open source, or restrict access for parts with non-didactical or contra-productive value. Learners, on the other hand, may start their programming with already working source code, without having to interrupt their learning process.

Code-based interactivity brings with new didactics. Educators may e.g. create multiple choice tests with programs. Besides selecting the correct algorithm, the learning objective could include code debugging and development. Algorithms can intentially be presented false, incomplete, or not optimized. As the source is compiled server-side, user input can be tracked and analyzed. Consider further user collaborations: the authoring workflow stores and versions each source code modification in the repository (see Figure 2). This potentially lets users share their work with peers. For example, an educator executing a course could ask students to check out a specific version, work it over, and check in their own proposal. Group members could create and discuss intermediate versions, again stored server-side. The technical framework therefore has to provide user management, rights and roles, and differentiated versioning. Note that repository material evolves on the fly.

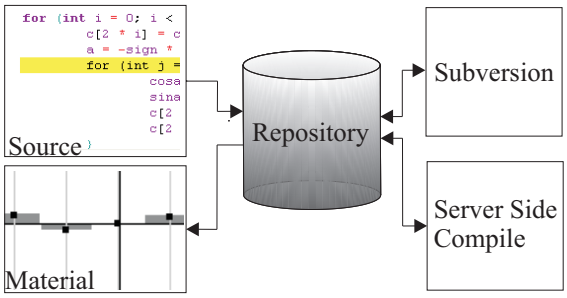


Figure 2: Code-based interactivity includes real source code. User modifications are compiled server-side and versioned, which allows users for discussing custom developments, and sharing evolved material by means of the repository.

Code-based interactivity applies to pure visuals, too. As above, the user works with the visual’s source, performs some cognitive actions, and recreates the visual. It is not required to present the visual as software – an alternative approach would compile the target server-side, convert it to the desired publishing format, and update the module. An example: considering SVG-based graphics, we could allow the user for modifying the object of interest in the XML source, compile the full source server-side, and then feed it back to the browser’s SVG plug-in. Alternatively, we could convert the compilation server-side to an image representation, and update the material accordingly. The latter approach has advantages in the case of exercises and tests, as it becomes possible to hide the material’s source.

## 4 Repository support for code-based interactivity

Repositories are collections of shared data, stored either in databases or file systems. Educational repositories in turn provide material usable in classroom or for self-studies, supporting a distributed, networked user community to work with, share, and evolve it. Repositories are well established as a communication and information exchange tool in the scientific community. They represent digital libraries offering publishing, quality assessment, distribution and licensing of modules and provide uniform resource names (URN) for referencing. State-of-the-art functionality [Owen et al. 2000; Figueiredo et al. 2004] includes organization, categorization, and search and browse facilities, together with management of users, rights, and roles. Evolving content is further supported by version control services, community ratings and annotations. Some repositories offer material templates, style sheets, and transformations in order to enlighten authoring and publishing processes, and to enforce or recommend a common content markup.

URNs guarantee valid referencing of repository material in textbooks and scientific papers over years. In the fast-changing Web, it is advisable for an educational repository to provide not only an URN, but store the referenced material itself. Following typical publishing mechanism, authors categorize their submissions themselves. The repository deploys this information for indexing and makes it available for browsing and advanced searching.

User management with rights and roles grants different levels of access to the material. Educators for example should be able to access material in publishing formats, whereas authors or licensed developers might be granted to access the material’s source. On

the other hand, suggested solutions should generally be locked for learners.

Modern repositories work like content management systems. The authoring process is facilitated by requesting user input step-by-step: starting with module creation, markup and upload to the repository, users may evolve material and submit new versions. Improvements may include errata, translations, extensions, and the inclusion of new media, e.g. visuals or interactivities. The repository therefore reverts to version control mechanisms or a database with versioning facilities. Versioning even supports rapid content creation as it allows authors for uploading materials incrementally. Offering peer reviewing for versions provides instant feedback.

We may apply this repository functionality also to source code, which can be processed like other material, i.e. submitted, reviewed, published, evolved, and versioned. In addition, we propose two repository extensions, an online source editor tool and a server-side compiling service. Server-side compiling processes and compiles the modified source to a publishing format, eventually including required binary libraries, and updates the material accordingly. The source editor replaces the static program listing by an interactive input field with, e.g. convenient syntax highlighting and indenting. We will present technical details in the following section. In essence, our approach introduces no extra hurdles, i.e. software installation or additional workflows, for end users.

Server-side compiling permits for a ”partly open source” licensing model handling privacy and copyright issues. The model presents only the source for a desired learning objective, and blends out any other material's source.

# 5 Technical Framework

On the technical side, we had to decide a) which programming language we want to support, b) which software we needed on the server as well as on client-side, and c) how we organize the client-server system. We had to specify which markup to use in our meta-language and how the sources of code-based interactivities had to be organized. We provide a prototype repository that accomplishes our requirements: it includes a Web-based editor, server-side compiling and the integration of a version control system.

## 5.1 Content markup

A markup language categorizes and structures the content of materials in the repository. By suggesting a defined DTD or XML schema, our repository may generate a presentation suitable for the different forms of publishing format: it may generate HTML, PDF, SVG, or other formats when given an adopted transformation style sheet. We have chosen an approach similar to MeML [Wang et al. 2003], a markup language that was designed to support mathematics (formulas) and calculation on the web. A brief example shows how a Java applet is included in the authoring context:

```
<interaction target="applet" id="dft1">
  <name>dft</name>
  <code>applets.dft.MainClass</code>
  <codebase>classes/</codebase>
  <width>850</width>
  <height>250</height>
</interaction>
```

This marks up an applet as an interactive object. The interaction block is stored in the repository with a unique identifier (”dft1”). The name and the starting main class define the applet's name and where the startup method is found. The location of necessary classes and libraries is defined by the ”codebase” tag. Further tags may concern the layout, e.g. set the applet's appearance.

A second example shows how source code manipulation of interactivities may be included:

```
<code type="compile" target="dft1" id="dft2">
  <main>MainClass</main>
  <package>applets.dft</package>
  <classes>DFT,BarsGraphics</classes>
  <version>1.0</version>
  <source class="DFT">
    <methodname>dft</methodname>
    <signature>
      <parameter number="0">int</parameter>
      <parameter number="1">double[]</parameter>
      <parameter number="2">double[]</parameter>
    </signature>
  </source>
</code>
```

During the stylesheet translation, the ”code” tag, which targets the previous applet, triggers the generation of an HTML form. The ”main” tag and the ”package” tag define the main class respectively the Java package name reflecting the directory structure of the applet sources. The ”source” tag defines the class file (here: ”DFT”) as attribute, and its sub-nodes declare the exact method name and its signature (here: ”dft(int,double[],double[])”). The class file including the source code should appear in the class file list of the ”classes” tag. The ”version” tag finally tells the repository which revision to checkout from the version control system.

## 5.2 Source code

Although the approach covers also media from text, slides, and graphics, our discussion will target mainly graphical-interactive material available as software programs. Here, publishing formats are represented by class libraries, for example directories with binary files, Java archives (JAR), dynamic link libraries (DLL), or dynamic shared objects (DSO). We prefer dynamic loading as this potentially lets users recompile and plug-in modified objects at runtime.

Only little effort is required from authors to turn their educational material into a code-based interactivity. In particular, the material source must be separated into two parts; a source package available for server-side compiling and the remainder that is already compiled to a publishing format. Parts of the source package that will represent program listings in the material must then be marked up. For our Java-based show case, we simply have used JavaDoc-style markup tags commenting a class or method.

A second prerequisite originates from our goal to replicate the classic textbook look and provide visuals and interactive illustrations without extra UI. Functionality going beyond direct parameter manipulation must therefore be scripted from other material parts. Scripting instructions are embedded as event handlers into text, graphics, and formula. Usually, the scripting target provides public parameters and methods that may be used in scripting. A fallback of this approach is that scripted material tends to become obscured if modified by different community members. An approach that encapsulates scripting instructions into components and organizes them by the repository [Hanisch and Straßer 2003b] might have advantages with respect to the embedding, interlinking, and update of scripted material.

### 5.3 Source editor

We required a technical solution for assisting users in the manipulation of program listings embedded into Web pages, even if they only have a browser with JavaScript at their disposal. The Muze project (http://www.muze.nl) has developed a free, Web-based editor that integrates JavaScript programs into the Web page to perform syntax highlighting and indenting for standard text fields in HTML forms. This a big step in terms of convenience compared to the plain text forms provided by the (X)HTML standards of the W3C.

Syntax highlighting and indenting already offers the most needed functionality of integrated development environments (IDE). Whereas IDEs additionally feature code completion, integrated documentation and debugging, the installation of an IDE is not advisable due to different reasons: IDEs are complex software products and require users to download a lot of data – but not every user has a high-speed access to the internet. The installation of an IDE is often complicated and assumes administration rights – which we cannot rely on. Third, an IDE is in contrast to a Web browser not inherently prepared to connect smoothly into our repository, whereas our proposal integrates both the authoring process and source code manipulation into a single Web page. At last, the sources of interest are not complete software packages but often very short, mathematically based algorithms with a length of about 10–20 lines. So we talk about algorithms that are well manipulative in a syntax highlighted text form.

### 5.4 Repository

The first prototype of our repository is based on Apache Group's Jakarta Tomcat project (http://jakarta.apache.org). Tomcat is used as the official reference implementation for the Java Servlet and JavaServer Pages (JSP) technologies. As we have prototyped our repository in JSP, the deployment of Tomcat is obvious. Tomcat allows for integrating additional server software like Cocoon, another widely used project of the Apache Group (http://cocoon.apache.org). Cocoon is a Web development framework based upon two major concepts: separation of content, layout, and presentation by the use of XML in conjunction with XSLT, and as secondary concept the notion of a sitemap as central control element that steers pipelines, which in turn compose requested pages of the Web application.

In fact, our repository has to cover two different requirements: on one side, we want to manage users, rights, and roles, steer our version control system efficiently, and compile source code on the fly; on the other side, we handle XML based documents that need to be transformed into user readable formats like HTML or PDF. We covered the first requirement straight-forward by implementing JSPs and Java servlets. The translation of our XML source pages by XSLT stylesheets necessitates Cocoon if implemented on the server. Here, we prefer an offline approach by deploying an efficient stylesheet processor like Xalan (http://xml.apache.org/#xalan) – the same processor that is also integrated into Cocoon for server-side stylesheet processing. This assures for coherent preparation of all needed XSLT stylesheets that will be deployed by Cocoon in the near future.

### 5.5 Server-side compiling

The compilation process is implemented completely server-side. The user adopts the source code accordingly, and then uploads his modification. On the server, a new version of the source file with regard to the user's identity and the version is generated, replacing simultaneously the original source with the user's proposal. Different versions of user runtime environments are supported by setting flags for the compilation process in order to generate compatible class files. The compilation is invoked on the fly, when the repository is receiving new input. Compiled code is stored on the server in binary files. The user is given a new Web page, where he may take a look at the new version of the interactivity. If the compiling process fails due to programming faults, resulting error messages are displayed on the Web page, too.

For the compilation process, each user has to log-in to the repository. This is not only for rights management but also for identification purposes: for each user the latest version of the source code has to be activated and preloaded. The compilation itself is realized by invoking a Java compiler from within the JSPs on the dynamically generated source class.

### 5.6 Source code versioning

Managing lots of code from different users with the need to restore old versions implicates the use of a version control system.

Commercial and free version systems are available. We have chosen Subversion [Collins-Sussman et al. 2004] as version control system for the following reasons. Subversion is freely available on different operating systems. It is a successor of CVS, the world's most used versioning system. In contrast to CVS, Subversion offers directory versioning, true version history, atomic commits, versioned metadata, different network layers, consistent data handling and efficient branching and tagging. For our purposes some features are of special interest: branching allows for handling different development branches with same files efficiently. Connectivity to the Subversion server from within of our repository is achieved by a freely available Java API (http://www.tmatesoft.com). The Java API offers the same functionality as the Subversion command-line client.

Our prototype solution commits each source code manipulation into Subversion. When manipulating source code for the first time, the server copies the original class files, including the modified source passage, into a newly created directory named the user's login name. All Java class files now have to be adapted to this new path, as the package name in Java classes reflects the directory structure where the class file is stored. After this renaming procedure and the replacement of the original code by the user's version, the class files are compiled. For checking a version back into Subversion, the files have again to be moved, as we want to store only one copy of class files per user. We plan further to make use of Subversion's ability to hold branches of files. This will allow Subversion for efficient handling of files and we will need fewer file handling operations on our application server.

By establishing a nomenclature how versions are named, we not only store persistently all source code versions per user, but we also inhibit caching conflicts that may occur from browser caching mechanisms, from caching mechanisms of the installed JRE plug-in or from caching implemented on proxies in the network connection: when new versions of a method in a Java class file are programmed, the possible caching mechanisms would not reload the newly compiled version, as they may not handle different versions of class files correctly in case they have still the same names. By giving each version new class names, we enforce the browser to load the desired version. For an efficient naming system, we have chosen a nomenclature in the form "user name":"version number":"original class name". This guarantees that new class files always have new

names, that all class files may easily be recognized, and that users, versions, and class files may correctly be associated.

As version control systems do not work with binary files in a satisfying way, software in the form of binary files like libraries or other resources have to be stored in a file system. They have to be accessible for the linking process and remain downloadable to the client's browser.

## 6 Showcase

In order to exemplify code-based interactivity and server-side compiling, we have chosen to prepare modules teaching sampling theory. It is a fundamental module in many SMET domains and presented in depth, e.g. in Glassner's textbook [Glassner 1995]. In the classic CG curriculum [Foley et al. 1990] the module explains aliasing effects and formulates the theoretical base for image filtering. Excellent educational online materials exist, e.g. MathWorld (http://www.mathworld.com) with comprehensive illustrations, Bourke's web site (http://astronomy.swin.edu.au/∼pbourke) providing source code extracts, or the corresponding Connexions course. Together with interactivities of Exploratories [Beall and Hughes 1996] and our own courses [Klein et al. 1998] they are instrumental in teaching and self-studies.

Yet, still the computer revolution has not happened. Theory is separated from the interactivities, and both of them are separated from their implementation. Content is not compatible, as they use different nomenclature, different approaches, and views. For example, teaching and learning is complicated by the use of different normalizations of the Fourier transformation, different orderings of the Fourier coefficients, and different visualizations of the Fourier spectrum. Many programs for sampling, filtering, reconstruction, and interactive presentation of signals exist, but all work different, and they cannot be combined and used, e.g. in classroom or in a Web browser.

On the contrary, our prepared material integrates theory, interactive illustrations, and real source code into single Web pages. Let us consider one of these modules, a Web page teaching the discrete Fourier transformation (DFT, see Figure 3). At a first glance, the material looks like a textbook with program listings, with one exception: an execute button. In fact, the material is dynamic. Users may interact with the illustration, i.e. they may directly manipulate the samples and the sampling rate in spatial and frequency domain. During their interactions, we approximate the signal in both domains. The program listing, here the core DFT algorithm, can be modified. Pressing the execute button sends the user input to the repository, which in turn sticks the sources together, checks in a new version, compiles the interactivity and updates the Web page. Compiler messages are printed below the program listing.

Educators can use such a module as base for exercises. For example, they might create a buggy version, or remove some code fragments. Learners could further be prompted to replace the DFT with an optimized fast Fourier transformation (FFT). As each version is tracked, educators can get an insight in lacking understanding, and react properly.

As the software material is written in Java, the author has marked up the DFT algorithm by a JavaDoc comment. Similar markup describes the input and output parameters of the algorithm, and related source code. We have included the latter directly below the program listing. Users may open the given methods and get either documentation or another source code editor.

The material can be experienced online at http://www.gris.uni-tuebingen.de/gris/ilo. We have created a guest account with proper rights to modify the source code. The only software requirement is a modern browser and a Java virtual machine or plug-in.

## 7 Conclusion and future work

Our approach has integrated for the first time educational material and real source code, respectively merged the ideas of educational repositories and code repositories. We have turned the textbook with program listings into dynamic material. Theory gets illustrated by visuals or interactivities and their implementation, which users may modify online. As the source is organized, compiled, and versioned server-side, repository material is evolved on the fly. The growing set of alternative implementations could form a proper base for adaptive software components, i.e. adaptive educational systems that alter source code.

We proposed server-side compiling to address difficulties in achieving a critical mass of submissions, as many authors would only accept a partly open source licence. The resulting code-based interactivity, however, potentially creates new didactics like code-based multiple choice, code debugging, and code development, which should be exploited in future work.

To include code-based interactivity into their material authors have to markup both their material and source code. While the latter only involves some commenting, content markup in general implicates the use of a standard XML scheme. We suggested letting the repository enforce a specific set of supported markup languages, e.g. a subset of DocBook.

We have presented crucial technical services and tools, and provided prototype implementations. Extending markup and styles for code-based interactivity is straightforward, and appropriate online source code editors exist. The repository services for code compiling and versioning required more attention in order to synchronize workflow and updates, circumvent browser and plug-in caching, and develop a matching user management, roles, and rights.

## References


ASSUNÇÃO, S. A., FIGUEIREDO, F. C., AND JORGE, J. A. 2002. Proposal for a CG education content online submission and reviewing system. In *Eurographics/SIGGRAPH Workshop on Computer Graphics Education (CGE02)*.

BARANIUK, R. G., BURRUS, C. S., JOHNSON, D. H., AND JONES, D. L. 2004. Connexions – sharing knowledge and building communities in signal processing. *IEEE Signal Processing Magazine* (May).

BEALL, JEFF E., D. A. M., AND HUGHES, J. F. 1996. Developing an interactive illustration: Using java and the web to make it worthwhile. *Proceedings of 3D and Multimedia on the Internet, WWW and Networks*.

BROWN, J., Ed. 2002. Visual Learning for Science and Engineering. Report of Eurographics/SIGGRAPH Visual Learning Campfire.

COLLINS-SUSSMAN, B., FITZPATRICK, B. W., AND PILATO, C. M. 2004. *Version Control with Subversion for Subversion 1.1*. O'Reilly & Associates, Inc.

CUNNINGHAM, S. 2000. GVE '99: Report of the 1999 eurographics/siggraph workshop on graphics and visualization education. *ACM SIGGRAPH Computer Graphics 33*, 4, 96–102.

FIGUEIREDO, F. C., EBER, D. E., AND JORGE, J. A., 2004. Computer graphics educational materials source - policies and status report. ACM SIGGRAPH 2004, Aug.

FOLEY, J. D., VAN DAM, A., FEINER, F., AND HUGHES, J. F. 1990. *Computer Graphics, Principles and Practice, Second Edition*. Addison-Wesley, Reading, Massachusetts.

GLASSNER, A. S., Ed. 1990. *Graphics Gems I*. Academic Press.

GLASSNER, A. S. 1995. *Principles of Digital Image Synthesis*. Morgan Kaufmann, San Francisco, CA.

HANISCH, F., AND STRASSER, W. 2003. Drag & drop scripting: How to do hypermedia right. *Eurographics 2003 Education Presentations (EG'03)*.

HANISCH, F., AND STRASSER, W. 2003. Adaptability and interoperability in the field of highly interactive web-based courseware. *Computer & Graphics 27*, 4, 647–655.

HECKBERT, P. S., Ed. 1994. *Graphics gems IV*, vol. 4 of *Graphics Gems*. AP Professional, Boston, MA, USA.

KLEIN, R., HANISCH, F., AND STRASSER, W. 1998. Web based Teaching of Computer Graphics: Concepts and Realization of an Interactive Online Course. In *SIGGRAPH 98 Conference Proceedings*, Addison Wesley, M. Cohen, Ed., Annual Conference Series, ACM SIGGRAPH.

KNOX, D., GOELMAN, D., FINCHER, S., HIGHTOWER, J., DALE, N., LOOSE, K., ADAMS, E., AND SPRINGSTEEL, F. 1999. The peer review process of teaching materials: Report of the ITiCSE'99 working group on validation of the quality of teaching materials. In *ITiCSE-WGR '99: Working group reports from ITiCSE on Innovation and technology in computer science education*, ACM Press, 87–100.

LALEUF, J. R., AND SPALTER, A. M. 2001. A component repository for learning objects: a progress report. In *JCDL '01: Proceedings of the 1st ACM/IEEE-CS joint conference on Digital libraries*, ACM Press, 33–40.

OWEN, G. S., SUNDERRAMAN, R., AND ZHANG, Y. 2000. The development of a digital library to support the teaching of computer graphics and visualization. *Computers and Graphics 24*, 4 (Aug.), 623–627.

PAPERT, S. 1980. *Mindstorms: Children, computers, and powerful ideas*. Basic Books, New York.

SHNEIDERMAN, B. 1997. Direct manipulation for comprehensible, predictable and controllable user interfaces. In *IUI '97: Proceedings of the 2nd international conference on Intelligent user interfaces*, ACM Press, 33–39.

SMITH-GRATTO, K., WICKS, D., AND BERGER, C. 2002. Merlot: Reaping the on-line vineyard. In *Proceedings of ED-MEDIA 2002*.

SPALTER, A. M., AND VAN DAM, A. 2003. Problems with using components in educational software. *Computers & Graphics 27*, 329.

WALSH, N., AND MUELLNER, L. 1999. *DocBook: The Definitive Guide*. O'Reilly & Associates, Inc.

WANG, P. S., KAJLER, N., ZHOU, Y., AND ZOU, X. 2003. WME: towards a web for mathematics education. In *ISSAC '03: Proceedings of the 2003 international symposium on Symbolic and algebraic computation*, ACM Press, 258–265.

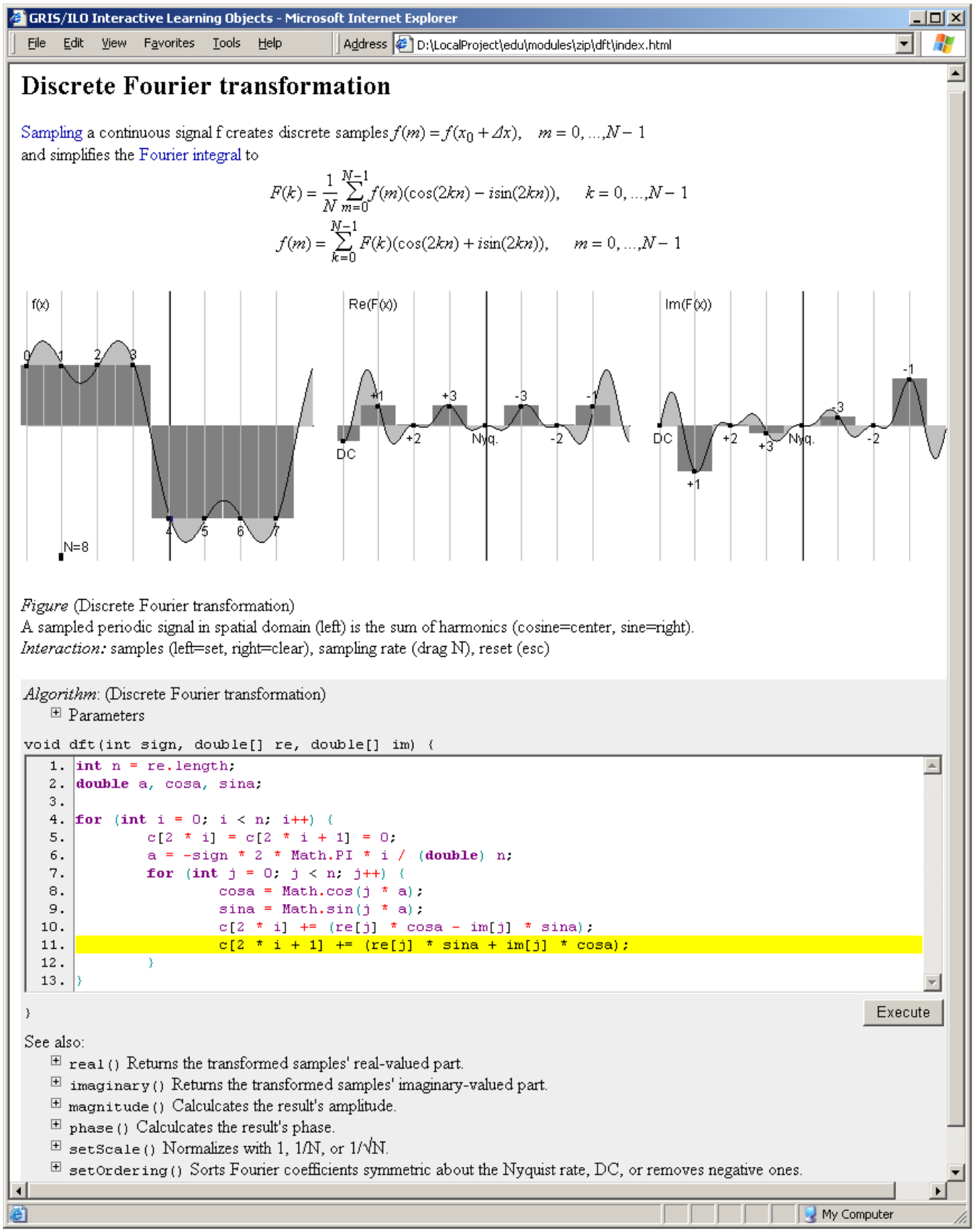


Figure 3: Server-side compiling turns the textbook with program listings into dynamic material. This material teaches signal processing. Samples and sampling rate are directly manipulative in spatial and frequency domain. Users may modify the implementation of the discrete Fourier transformation at place. As the source is organized, compiled, and versioned server-side, material evolves on the fly.