Making of an Interactive Teaching Gem

Frank Hanisch, Wolfgang Straßer
WSI/GRIS University of Tübingen
{fhanisch, strasser}@gris.uni-tuebingen.de

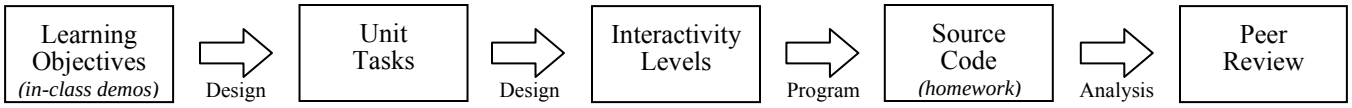

Figure 1: The full educational pipeline of an interactive teaching gem compared to classic in-class demos and homework.

Abstract

Current educational repositories lack in interactive material. We describe the full pipeline for creating an interactive teaching gem in computer graphics and related domains. Based on experiences from our own classroom use, we explain why and how we specify learning objectives, set up unit tasks, design interactivity levels, and provide source code for homework. The pipeline is illustrated with the making of a real interactive teaching gem. From this we derive implications for computer graphics educators and current community building efforts, and formulate two matching repository services.

Keywords: Community building, Educational Repositories, Interactivity.

1 Introduction

The Eurographics and ACM SIGGRAPH educational community created a peer-reviewed repository for teaching material, the international CGEMS – Computer Graphics Educational Source [Figueredo et al. 2004]. It is meant to offer authors reputation for a work acknowledged rarely, and to help educators in finding useful material. The CGEMS includes interactive teaching gems, software bits that highlight a particular problem interactively. Typically, educators in computer graphics (CG) and related fields use a diversity of them, different in concept, view, and technology, to make teaching more visual and learning more fun. They also impart practice: if students could work with the source code and analyze it with their peers, knowledge acquisition would not only be fostered, but transferred.

Now there's the rub. Current repositories lack in interactive material, and so do educators. While best-practices for developing non-interactive modules exist and lab notes with written problem sets are available, interactive learning material is hard to create, outdated soon with new technologies, and often ineffective [Hansen et al. 2002]. Design pattern and software componentry impart only a broad idea of reuse and adaptation [Spalter and van Dam 2003; Hanisch and Straßer 2003]. Most work stem from research or student projects and neglect didactics. We address this by a recipe for creating interactive teaching gems with real educational value, and true interactivity.

Starting with background on today's CG education (Section 2), we shape the use of interactive teaching gems for in-class demos and homework (Section 3). A matching creation pipeline (Sections 4.1 to 4.4) sequentially collects learning concepts from vocabulary to application to critique, designs interaction in terms of unit tasks and cognitive level, and renders source code useful for learning. The process is illustrated with an interactive teaching gem on signal processing, namely the convolution concept. We derive concrete steps to be taken for building a CG educational community with instrumental repository services and associated classrooms (Section 5).

2 Background on CG Education

Interactive teaching gems provide a visual prototype for a given abstract concept, and associate imparted interactions. It is commonly accepted that representing knowledge in these three modes deepens understanding. For a CG view on the psychology of mind see e.g. Alan Kay's [1990] thoughts on "doing with images makes symbols" or the Visual Learning program [Brown 2002]. In a cognitive sense, interactions can be classified by their learning quality [Schulmeister 2003] and structured into unit tasks, encapsulated activities with minimal interdependencies [Card et al. 1983]. Offering unit tasks is essential for letting learners explore a concept.

The Exploratory project [Spalter and Simpson 2000] studied approaches to integrate interactive material into established CG curricula, ranging from in-class demos to optional extra-curricular use to lab-sessions and homework. Due to logistical and pedagogical problems they found in-class demos and homework most rewarding; especially the latter activated autonomous student processes adding depth to the class presentation. They mention that advance planning was essential for learning success, e.g. lecture-lab timing, assistants' training, and preparation of new content for use in labs or homework.

Our own experiences back these results. For our CG courses we developed more than 100 interactive teaching gems, and employ them together with complementary material from Exploratory and similar sources. We assign student groups to program some of them in a fortnight. Notably, throughout the years, our students always browsed for other interactive material that provided yet another view of the topic, and integrated it into their work. Peer-reviewed repositories like the CGEMS [Figueredo et al. 2004] or MERLOT [Smith-Gratto et al. 2002] would provide a proper source for their studies.

Drawn on experiences in Fine arts, Whittington activated student learning in groups by introducing a peer reviewing process [Whittington 2004]. She describes how she set up CG student projects with explicit information on learning objectives, schedule, technical and design criteria, and critique presentation method. Students' work improved with each critique skill they learned, and with the conscious of being evaluated by peers. However, she reported that factual and positive constructive discussion had first to be learned.

Some of the critique took place online in a discussion board. Below, we turn this approach into a repository service, and replace Whittington's learning concepts, vocabulary and sample critique questions with an established taxonomy [Bloom 1956] from cognitive theories. Engineering practice for software peer reviewing is provided, e.g. by O'Neill [2002], who distinguishes formal software inspections and less rigorous software walkthroughs. While Whittington has chosen informal reviews, we let students critique formally. Review forms are available, e.g. from SIGGRAPH or Eurographics, or tailored for educational material from [Knox et al. 1999].

3 Teaching Gems in Practice

Let us consider the two proven applications of interactive teaching gems in CG education, in-class demos and homework. To stay concrete, we exemplify the current teaching of 'convolution', an integral concept of CG curricula to explain, e.g. aliasing, spline modeling, or image filtering.

An in-class demo is meant to illustrate a concept. Depending on the time it is shown, the demo either introduces and motivates the concept, or deepens it. The instructor might for example introduce convolution in the context of the B-spline basis, which is generated by convolutions of a box function. Assuming the instructor starts with formula, and then wants to associate a visual, interactive teaching gem – which material is appropriate?

One of our own applets presents box convolution clear and brief. Learners conceive a single action, convolving two boxes by direct manipulation (see Figure 2). The applet matches with the visuals printed in student reading, and time is kept to a minimum for associating doing, image, and symbols.

The Exploratories' convolution applet [Spalter and Simpson 2000] transfers a mechanical teaching material, layering and sliding of overheads. Users can draw two signals and convolve them physically by moving a slider. That way, instructors may reuse the material later for teaching other concepts, e.g. sampling with a Dirac series, interpolating with a triangle, or Gaussian smoothing. Functions are not pre-defined, which slows down work. Also, the convolved result cannot be copied to the input signal, so the instructor can only sketch spline basis generation.

The Joy of Convolution [Crutchfield and Rugh 1997] provides self-drawn signals and the convolution action, too, and includes pre-defined signals. Sadly, they target audio processing instead of CG applications, so the demonstration still requires live drawing. The instructor may choose between two applets for discrete and continuous signals according to his structuring. But the applet includes unnecessary obstacles that might not match the instructor's or students' view, e.g. fixed formula, parameter labels, and a time-based system.

These and other, subtle problems with the user interface and underlying model can be bypassed in an oral demonstration. Trouble starts when the material is used for homework and students have to explore facts on their own. The authors of Joy of Convolution for example show discrete-time applets in class, and leave the continuous-time applets entirely to self-studies. The transition itself, an intermediate learning task, is not explained. Our own approach proceeds reversely: we introduce the continuous case, explain the transition in class, and let students construct the discrete algorithm. Students are given a ready programming skeleton and a working example, e.g. the above material. Note that these miss source code, so the instructor still has to prepare a custom framework or let students start from scratch. Learning then occurs in small groups: for a few days, students discuss the convolution concept, share ideas, and finally program and debug the algorithm. Instruments like forum or wiki support knowledge exchange between groups.

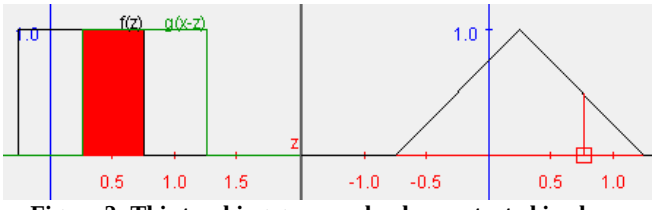

Figure 2: This teaching gem can be demonstrated in class within seconds. It associates the convolution concept with a physical action by letting users drag a box over a second one. It is not prepared for homework or self-studies.

They now face details not mentioned in class and student reading. Consider image processing that interprets one signal as kernel. Most kernels are symmetric about zero – students may start optimizing the algorithm and eventually end up with a limited understanding. Another complexity arises if learners can vary the kernel area in sketching, but expect it constant. The Exploratories includes normalization, the Joy of Convolution not; results differ and may confuse students. And what about drawn signals, are they cyclic? A sine should certainly be conserved, but for a sketching close to sign(x), -1<=x<=1, the answer is indeterminable. The working examples do not support periodicity, so homework should include clarifying instructions.

Note that a teaching gem recurs many times after the actual in-class demos or homework. Besides being referred in class at a later date, students might continue to rework it. Supposing we have managed the process above, the killer application is: students have explored/programmed convolution and now learn about reconstruction filtering; they want to explore Blinn's [1989] NICE filter, which is made by multiplying two truncated sinc signals – sin(x)/x –, one of them stretched 1/3 horizontally; in the Fourier domain their dual signals would be convolved. Should a teaching gem on convolution care about such ideas? We say: yes, but a proper planning of unit tasks and interactivity is required. Which brings us to the following section.

4 The Creation Pipeline

We have seen that, although the abstract convolution concept is simple, creating a matching teaching gem seems hard. The crux is that concepts like convolution are taught in context of other concepts, with different emphasis, data, and user control. Even implementations change. If the concept is presented fragmentary, student learning will consider teaching gems from different sources, mixing even more views, actions, and slightly different concepts. It takes planning to create a teaching gem, from learning objectives to objects and actions to the final source code.

4.1 Specifying Learning Objectives

Learning objectives state explicitly what the learner is expected to demonstrate in the final exam, or later. They should be clearly specified for a concept and given to the students. To understand how this works, let us classify learning objectives by their difficulty and apply them to the convolution concept. We adapt Bloom's [1956] cumulative levels for the CG domain; each level builds on and subsumes the ones below.

Knowledge: The novice CG student learns about a concept in class and from student reading. Afterwards he should, as a first goal, recognize the concept and use related vocabulary. Only for such memorizing individual learning performs best, any other level is reached more effectively by learning in groups.

Convolution vocabulary can be found, e.g. in Glassner [1995]. A novice should be able to write down the one-dimensional convolution $f = g \otimes h$ with signals g and h: $f(x)=\int g(a)h(x-a)da$. For self-studies he should know that authors may prefer other symbols and labels; x denotes a spatial system, whereas t would denote time. Knowledge includes commutative law and linearity, and

time-shifting the input causes the same time-shift in the output. 2D convolution works accordingly.

Comprehension: Learning vocabulary prepares the ground for understanding the concept and explaining it in own words. The student can now, if needed, reconstruct given examples or proofs. He has gained the ability to ask questions in class. Lacks in understanding are recognized and followed up in self-studies. Comprehension is inquired in oral exams.

Depending on the actual context, a CG student should understand convolution as follows with examples from lecture or reading. Signals in CG represent images and functions; they are convolved with a kernel. Most kernels are symmetric with limited support; they filter the original signal, e.g. smooth it. Convolving with a box increases the signals' continuity by one. Two boxes convolve to a triangle, repeated box convolution converges to a Gaussian; the respective support length can be explained. In Fourier theory, convolution is dual to multiplication: F=GH, large letters denote Fourier transformed signals. The learner should know that the proof centers on $e^{-a} = e^{-x}e^{-(x-a)}$. The Dirac impulse copies the signal. Having understood linearity, the learner can now imagine effects of mixing in a third signal, and of scaling inputs in time or intensity.

Application: If the learner has acquired a concept flexible enough, he can put it into practice, i.e. transfer it to a similar, but modified setup. The student may now generate custom examples. This learning task is covered by homework.

How is convolution applied? In modeling, the B-spline is generated by convolving the control polygon, over and over again, with a box. Convolving a signal with a Dirac series produces copies, and their possible overlapping explains aliasing. A sample & hold display corresponds to convolving d-distant points with a d-sized box. Using a double-sized triangle instead interpolates the points linearly. For reconstructing an image, it should be ideally convolved with a sinc; in practice, with a NICE filter (see above). And so forth.

Programming: Computer science students should also know how to implement the concept. Homework and student projects use a prepared framework, or start from scratch. Note that the student should be able to state the correctness of his approach.

In our case, the learner should be able to implement the infinite integral. According to the signals' internal representation and view resolution he must decide on boundaries, sampling, and normalization. The learner should choose an appropriate signal and kernel, e.g. (0 1 1 0 0 1 1 0) and (1 1), convolve them programmatically, and verify the result (0 1 2 1 0 1 2 1) manually. An exemplary output with our material is given in Figure 3.

Analysis/Critique: Besides knowledge exchange, students work in groups to learn arguing and represent ideas. On this cognitive level, the learner has gained the ability to critique views of the concept, owns and those of their peers. Approaches like Whittington's peer reviewing (see above) aim at including analysis into CG education.

Synthesis: If a concept is learned in depth, new material can be directly associated with it. A learner may, for instance, attend a lecture on scientific visualization and hear about line integral convolution. He might then understand the visualized flow directly as a filtering operation. Processes span years, with different instructors and material.

Evaluation: The expert level includes assessment skills. Learner can now evaluate work on the concept objectively. Evaluation skills are currently not addressed by CG curricula.

Collecting learning objectives for a given concept is straightforward and can be based on questionnaires and textbook analysis. For the primary ones, knowledge to programming, we interviewed our instructor and analyzed existing assignments and transcripts of the oral exam. Intermediate levels (analysis to evaluation) were extracted from questionnaires taken with the instructor and our students. Affective learning objectives like student awareness, motivation and participation, valuing, and knowledge organization were not considered.

4.2 Offering Unit Tasks

Now that we understand the concept and related learning objectives, how can we create a matching interactive teaching gem? Clearly, the material should support the cognitive tasks required for reaching our learning objectives. So let us structure cognitive tasks into unit tasks, operational building blocks with minimal interdependencies. Out of these, the instructor will then compile his in-class demo. Students will use them for exploration in homework and self-studies. Unit tasks are acquired and executed: users decide what to do, then locate objects and operate with them, and verify the result. If a material imparts all the tasks' objects and actions, it imparts all learning objectives.

Finding a concept's unit tasks is critical, but rewarding. We have experienced that a task-based teaching gem is easier to use, and even easier to develop. Basically, developers run through the learning objectives to work out a set of unit tasks. The process is repeated for different scenarios to refine the set. For the above learning objectives of convolution, we propose ten unit tasks:

1. Set parts of input k to zero
2. Draw parts of input k
3. Set input k to a box
4. Set input k to a sinc
5. Set input k to an impulse series
6. Set input k to the convolution g⊗h
7. Set input k to the multiplication g•h
8. Scale input k by α: k → αk
9. Shift input k by s: k(t) → k(t-s)
10. Stretch input k by α: k(t) → k(αt)

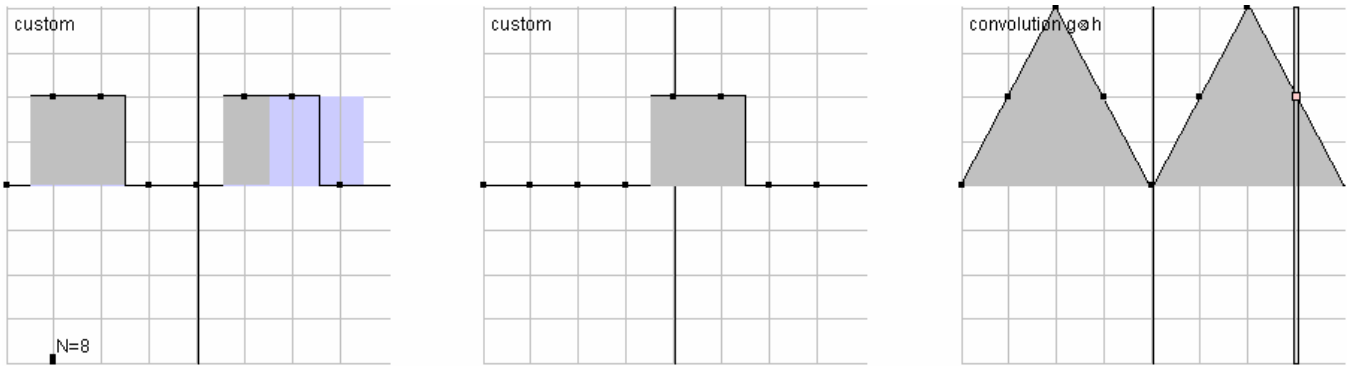

Figure 3: This material convolves two custom signals (left and middle). The samples let the user visually check the algorithm's correctness. By varying the sampling rate (here N=8), the transition between continuous and discrete view can be explored.

Users operate on the signal k∈ {g, h}. The first task allows for limiting the support. Tasks 2-5 include common signals in CG; tasks 6/7 repeat convolution and dual multiplication. The remaining tasks derive from the linearity and time-shifting properties. Note that impulse, triangle, Gaussian, and NICE can be constructed, the latter according to Blinn [1989] with the task series 4g-3h-10h-7g-4h-10h-7g-10g (letters denote objects): Multiply the ideal sinc filter with a truncation box, then multiply it with a stretched sinc (Lanczos window), and stretch it. Or, with the dual series 3g-4h-9h-6g-3h-10h in the Fourier domain: convolve the ideal box with a sinc stretching, then convolve (average) it with a stretched box and shrink the result.

Demonstrating aliasing, on the other hand, would, e.g. comprise the task series 2g-1g-5h-10h-10h: draw signal g, limit its support, set kernel h to a Dirac series, and vary its stretching.

The unit task principle clears the user interface and keeps teaching gems from getting too application-specific. For instance, adding stretched signal versions would introduce non-minimal dependencies; instead, stretching becomes an action that can be applied to any signal and that is useful for other explorations. Stretching a box towards the constant (DC) blurs the output towards the average, shrinking it again towards the impulse resharpens the output. Stretching might also help in discovering that support lengths sum up.

4.3 Designing Interactivity Levels

Having set up unit tasks, we now design the graphical user interface. The decisions we make here set the time and skills required to perform a task. Widgets and interaction style are straightforward: we choose, for each task, among direct manipulation, sliders, input fields, list selections, buttons, or others. Quantitative measures that count physical movements, e.g. Fitts' law, may help here. Accessibility for handicapped users should be assured.

Next, we consider the teaching gem's learning quality – the cognitive level the learner achieves while interacting with the material. We follow Schulmeister's [2003] cumulative levels, but leave out media-specific actions for navigation/animation and data selections, as they infer only little learning.

Varying concept views: The learner may vary the concept's cognitive level of detail, i.e. zoom in on specific aspects or select different points of views, in a spatial or cognitive sense.

A proper teaching gem on convolution visualizes the calculation of the integral at a given location x. The transition between continuous and discrete view of the concept can be explored, for instance by sampling both inputs g_s and h_s and visualizing the convolved samples $f_s = g_s \otimes h_s$, and then letting the user vary the sampling rate (see Figure 3). The correctness can be derived from convolution linearity.

Varying concept parameters: The learner manipulates the objects of a unit task.

In our case, this is accomplished by the unit tasks 1-10. Varying parameters takes place on a higher cognitive level than varying views because the learner must find reasonable inputs himself. This is another motive for excluding some of the signals (triangle, Gaussian, NICE) from selection and letting the user construct them.

Varying the concept: The learner questions and modifies the concept to understand what happens if it is defined differently. Today's teaching materials do not support this cognitive level. Instead, homework and self-studies let students work with the source code. They sharpen the concept's definition by trial and error, respectively debugging, which is acceptable only if students are used to programming or should learn it anyway.

For example, students should learn the effects of convolving with a kernel not normalized. What would happen if we choose other integral boundaries, and what if we don't mirror the kernel? Is it correct if we take the signals' intersection? – No, we have to multiply them. Et cetera.

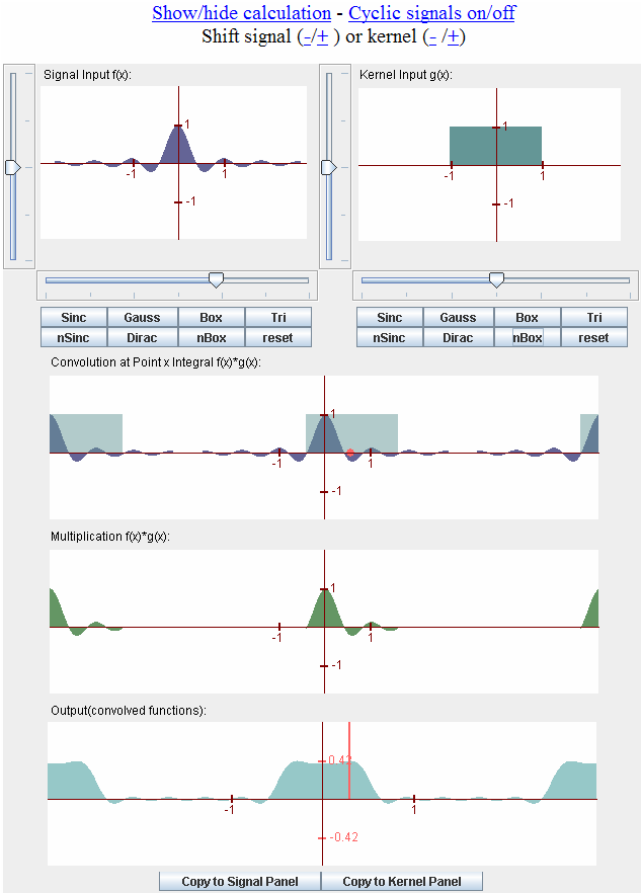

Figure 4: A student-made teaching gem from a lecture on interactive learning technologies. We specified learning objectives of convolution and asked students to set up unit tasks and interactivity levels. Through this we learned more about student's views and problems, and we could refine our own teaching gem.

Getting help: Help can be given globally for reaching cognitive tasks or while performing a single unit task. In the former case, an in-class demo made of unit tasks can be directly transformed into a guided tour. Local guidance should be offered if learners trouble with acquiring a task, with locating objects or operations, or with performing and verifying the operation.

Let us run through our convolution tasks again. Drawing a signal might be feasible, so the teaching gem should recognize common shapes, at least constant and linear input. Scaling and stretching can be supported by snapping to the values 0, 0.5, 1, and to the value for which the kernel becomes normalized.

Designing interactivity for a given concept takes time, so our approach was as follows. In 2005, we held a lecture on Interactive Learning Technologies in the CG domain. As part of homework, students had to design a teaching gem on convolution, i.e. we gave them a list of learning objectives and working examples (the ones above). They had to set up unit tasks, specify interaction styles, and cognitive levels. That way, we could not only check our own set, but learn more about the students' views, how they work with the concept, and where they need help. The same holds for the implementations they worked out subsequently. One of the outcomes is illustrated in Figure 4.

4.4 Preparing Source Code

Teaching gems that are repurposed from research or student projects typically neglect fundamental didactics. An advance planning of unit tasks and interactivity levels as we described above already generates acceptable results for in-class demos. The student work in Figure 4 for example visualizes convolution calculation and includes common signals. All unit tasks except multiplication are implemented. The students decided to toggle the visualized calculation and signal periodicity by scripting; shifting is also outsourced. Interaction and interface design, although moderate, can be handled by an intermediate user.

Yet, homework material must satisfy another need: it must bring with not only source code, but source code with educational value. This means that, at first, the learning time should be minimized. A heavy, object-oriented programming framework might be overwhelming if users want to solve a single learning task. Source code with many object dependencies cannot be simply exchanged among student groups. We have experienced this can effectively obstruct a peer reviewing approach.

In the student's applet, for example, source code is complicated by the stretching operator. Drawn signals that are shrinked and stretched back must be buffered in order to restore them correctly. Such unpleasant details should be encapsulated into a second object, so that the concept's object remains clear.

For our own teaching gems we use only few Java objects. The graphics engine, a base for various interactive gems, is made of a canvas with views and controllers, in which we render a list of graphics items. Math objects like a function are primitive floating point arrays, connected to a text parser for custom input. Several functions are pre-defined and connected to draggable handles. Controllers offer the model-to-view transformation and basic selection/dragging interactivity, the latter with drag-snapping.

Our final convolution applet (see Figure 5) then holds a list of signals as function objects, implements a custom view with a given sampling, and a custom controller for varying the sampling and for selecting signals. All unit tasks are supported and accessibility is granted; the controller further helps in drawing by recognizing partially linear shapes. Objects are available to the scripting interface, which allows for changing parameters without having to compile the source code. With that, users can customize the applet, e.g. input signals they miss like a high pass for edge detection or a ramp.

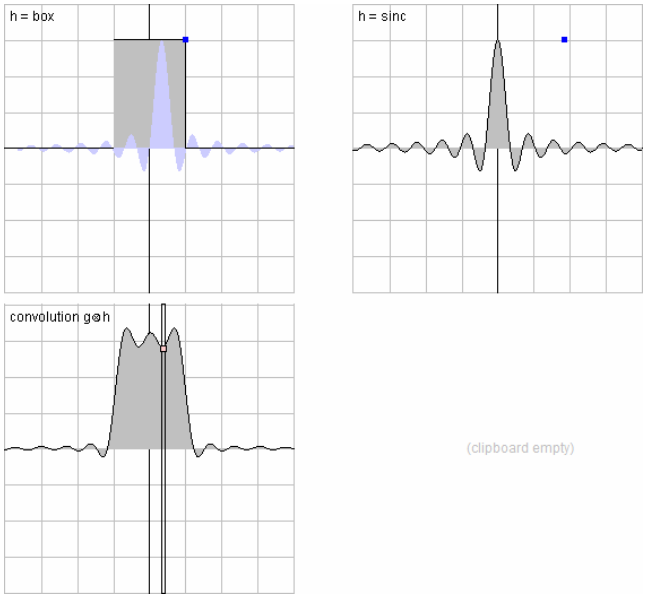

Figure 5: This teaching gem realizes learning objectives of convolution, e.g. Blinn's NICE filter design, by unit tasks with minimal interdependencies. Signals can be selected or drawn, and scaled/stretched with the rectangular handles. They can also be copied to the clipboard and reused as input. Text and appearance are adaptable by standard CSS.

Scripting becomes valuable if material is used in different curricula or by different instructors. In that case, a teaching gem must provide means for changing views, for instance in language, symbols, or item coloring. Such styling should not be made in source code, and neither by scripting. Rather, we suggest the use of cascading style sheets (CSS), a standard Web language for styling content by name, by class, and by a custom type. The latter enables developers to predefine styles for several types, and then change view details simply by switching object types.

Our Java graphics engine stores CSS information in a Java object, which is queried and cached by graphics items before rendering. We have defined a standard color scheme with primary and secondary foreground, background, filling, and emphasis. In the convolution applet, we have used that scheme to visualize the integral calculation at a given location x, and to hide the sampling. Instructors that want to draw attention to a different learning goal, for example aliasing, can change the CSS styling and highlight e.g. the copying effect of the Dirac series instead.

5 Implications on CG Community Building

5.1 Student Peer Critique

In specifying learning objectives we found that analysis skills and evaluation are not covered in current CG education. Whittington [2003] therefore introduced peer critiques to student groups. We propose to formalize this approach into a peer-reviewing process, which fosters student work by role-playing, peer critique, and structured group interaction. In winter 2005/2006, we evaluated in a field study how this process can be applied in the CG domain.

Perspective: Analysis skills can be deepened if students change roles, here from learners to reviewers and back, and compare their work with others.

Our field study assigned 60 CG students with programming and theory on aliasing. Half the students exchanged and critiqued their draft solutions; the other half worked as usual. Critique could be included in the final draft.

Peer Reviewing: Students can acquire assessment skills when they critique peer work. However, it is important to provide students with review criteria and guidelines for constructive critique.

Our students submitted their programming by e-mail. As part of the workflow, we had to anonymize the drafted source code and writings, and sent it, together with the SIGGRAPH review form and instructions, to another, randomly chosen group. Review criteria considered presentation, language/technology, correctness, and an overall rating. Comments could be made in the form and in the source code.

Organization: Group work is structured. The basic idea is that few learners know about effective group interaction, so they seldom come to appropriate results. The level of structuring is crucial: overstructuring reduces student motivation, too little structuring, on the other hand, does not improve performance.

Our assignment took a fortnight. Students had to draft half the source code within a week and review it the next day. We suggested 20 minutes for reviewing. The second week repeated the process for theory, again with reviewing at the end of the week. Control groups shouldn't get confused, so they also had to deliver drafts. All students demonstrated and explained their revised work in a final, oral exam.

Afterwards, we analyzed students' drafting and critique and interviewed them how they worked in practice and where they had problems. Data was collected anonymously, so students could stay plain. Students stated a significantly higher subjective learning success than control group members, which corresponds to the results of Whittington. They also claimed to have spent more time on drafting their solutions as they are reviewed by their peers. This did not cause a loss in motivation, neither subjectively or objectively. Students even seemed to be more intrinsic motivated than control groups. Objectively, we could measure a learning success only for peer critique; domain-specific knowledge did not differ significantly as both sides reached maximum grades in the oral exam. This should be evaluated again in a longer-term study.

By comparing individual drafts and reviews we found that results are best when authors and reviewers share the same high skills or at least one has higher skills. The better the draft the more learns the reviewer; the same is true for the review.

Introducing peer reviewing to current CG education is laborious due to the logistical problems. It would become manageable if the classroom is organized by an instrumental repository service. This would definitely increase the repository's educational impact. At first, the service would have to handle a workflow that resembles the one for technical papers. The real challenge, however, is how to exchange source code without technical hurdles: the repository should verify that material works before sending it, and store and present different material versions. A possible repository technology might be server-side compiling [Görke et al. 2005]. The approach lets users work with source code online and compile it server-side. Results are versioned and accessible to peers. However, for now only prototypes exists.

5.2 Scholarly Paper

We have created a teaching gem, so let's share it with the CG educational community. All that is left to do is to submit it to an educational repository, e.g. the CGEMS.

A valuable submission should contain a scholarly paper with information on learning objectives, unit tasks, interactivity, and source code. We can use the above classification, but compatibility to the repository's metadata must be assured. We suggest the XML schema for the IEEE LOM – Learning Object Metadata, standard no. 1484.12.3-2005, which covers the core of CGEMS, MERLOT, and other educational repositories. MERLOT's extensions for photo, source code availability, and subcategories are useful.

While learning objectives and interactivity levels are supported, we must extend the schema for information on how to use the material in class or for homework. First, any operation should be annotated with its physical interaction type and accessibility key. We propose to list the major cognitive tasks as a list (or hierarchy) of unit tasks, and let the educational community submit more ideas for demos and assignments. Similar to the MERLOT assignment, such comments describe applications not planned by the material's author; instead of free-form instructions, they provide concrete, task-based steps. Tasks not implemented by the material must be added as scripting comments.

Most repositories offer filling out the fields in a Web interface. Non-supported fields can then be moved to the comments or general description. Repositories then offer searching/browsing material by metadata, here by concept or interactivity level. But the XML representation of unit tasks and their structuring into cognitive tasks has more potential, e.g. guided tours could be generated system-side, or user operations could be tracked, which in turn would allow for adaptive interaction support.

6 Conclusion

In the hope that a recipe for interactive teaching gems can raise their quality and availability, we have run through the making of a single material. For each step in the pipeline, we have presented concrete actions to be taken. Planning includes what (learning objectives) and how (unit tasks) students learn from the material, and how they are presented to the user (physical and cognitive interaction). Our students managed this process in programming homework with success.

Based on a field study we propose to build a student peer reviewing service into a central, educational repository, which lets students exchange, analyze and evaluate such gems – including source code and theory. We finally describe how to add a matching scholarly paper and submit them to one of the available repositories. Possible evolutions of educational repositories are included.

References

BLINN, J. 1989. Return of the Jaggy. In *IEEE Computer Graphics and Applications* 9(2), 82-89.

BLOOM B. S. 1956. *Taxonomy of Educational Objectives, Handbook I: The Cognitive Domain.* David McKay, NY.

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

CARD, S. K., MORAN, T. P., AND NEWELL, A. 1983. The Psychology of Human-Computer Interaction. Hillsdale, Lawrence Erlbaum Associates, NJ.

CRUTCHFIELD, S. G. AND RUGH, W. J. 1998. Interactive Learning for Signals, Systems, and Control. In *IEEE Control Systems Magazine* 18(4), 88-91.

FIGUEIREDO, F. C., EBER, D. E., AND JORGE, J. A., 2004. Computer graphics educational materials source – policies and status report. In *SIGGRAPH'04 Conference Proceedings*, ACM.

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

GÖRKE, J., HANISCH F., AND STRASSER, W. 2005. Live graphics gems as a way to raise repositories for computer graphics education. In *SIGGRAPH'05 Conference Educators Program*, Los Angeles, ACM.

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

HANSEN, S. R., NARAYANAN N. H. AND HEGARTY, M. 2002. Designing educationally effective algorithm visualizations. In *Journal of Visual Languages and Computing 13(3)*, 291-317.

KAY, A. 1990. User Interface: A Personal View. In *The Art of Human-Computer Interface Design,* Laurel, B., Ed., Addison-Wesley, Reading, MA, 191-207.

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. In *ITiCSE-WGR'99: Working group reports from ITiCSE on Innovation and technology in computer science education*, 87–100.

O'NEILL, D. 2002. Peer Reviews. In *Encyclopedia of Software Engineering, volume 2*, John Wiley & Sons, NY, 929-945.

SCHULMEISTER R. 2003. Taxonomy of multimedia components: a contribution to the current metadata debate. In *Studies in Communication Sciences* 3(1), 61-80.

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

SPALTER, A. M., AND SIMPSON, R. M. 2000. Integrating Interactive Computer-Based Learning Experiences Into Established Curricula. In *Proc. of ACM ITICSE,* 5, 116-119.

SPALTER, A. M., AND VAN DAM, A. 2003. Problems with using components in educational software. In *Computers & Graphics 27(3)*, 329-337.

WHITTINGTON, J. 2004. The process of effective critiques. In *Computers & Graphics* 28, 401-407.