

# Granularity in the Design of Interactive Illustrations

Daniel L. Gould   Rosemary M. Simpson   Andries van Dam  
Brown University site of the NSF Science and Technology Center  
for Computer Graphics and Scientific Visualization  
Box 1910 Brown University; Providence, RI 02912  
{dlg,rms,avd@cs.brown.edu}

## Abstract

We describe some issues in designing and building educational Java applets for an introductory computer graphics course. The design problem involves balancing educational goals of building intuition about fundamental concepts in a domain against heterogeneity both in subject material and in student backgrounds. We present our design approach for resolving these forces — fine-grained units addressing small concepts — and discuss its effects on other areas including hypertext structure, interface design, and software engineering.

## Keywords

educational software, user interface, reuse, granularity, applets, hypertext, interactive illustrations

## 1 Introduction

All over the world, people are developing interactive educational applets. These applets are being used in classroom demonstrations, hands-on laboratories[2], self-directed work outside of class, and distance learning. Despite the fairly broad dissemination of these applets, however, little has been published about the tradeoffs in the design and implementation of this type of educational software.

While such issues as the user interface[15], navigation structures, and granularity are all important, we believe granularity has the most impact on the rest of the design process. By granularity we mean the conceptual scope covered; fine granularity usually leads to compact programs with relatively few features. Coarse-grained applets are similar to complete applications in that they attempt to demonstrate several related ideas. Fine-grained applets focus narrowly on an individual concept or a very small number of closely related concepts; several of these smaller programs are normally required to cover the material that a larger program addresses.

The purpose of this paper is to examine the benefits and costs of the fine-grained model. We have found that this model works well in several contexts, though we do not believe that it is necessarily the right solution in all situations.

Our current domain is Brown's introductory computer graphics course in which the teacher must deal with great diversity in both subject material and student backgrounds; computer graphics uses basic concepts from such disparate areas as linear algebra, geometry, physics, and human perception. This heterogeneity provides much of the motivation for the fine-grained approach. We hope in this paper to target not only the SIGCSE community, but also computer

graphics teachers — specifically those who develop educational applets. However, we believe that the principles of the fine-grained model discussed here will apply to other domains as well.

We first outline related work in educational technology, with particular reference to the thirty years of work at Brown University from which the current project evolved. Then we introduce the context that has provided both the resources and the forces we must balance to create effective teaching applets. We then discuss the fine-grained model and its impact on our development process. Finally, we describe our future plans.

## 2 Related Work

### 2.1 General

Educational software and distance learning have a long history, starting with Coursewriter, PLATO, and SCHOLAR in the 1960s, continuing with LOGO, ACT, WEST, SOPHIE, FRESS, and TICCIT in the '70s, BALSAs, Boxer, Intermedia, and Perseus in the '80s, then Port of Entry, Virtual Laboratory, the online computer graphics course at Universität Tübingen[13] and HyperGraph in the '90s. References for all these projects and many others may be found at [7].

Educational software in general suffers from problems with platform dependency, high development costs, and difficulties in reusing basic components. Java applets of the type we describe below present a partial solution to these problems.

### 2.2 Brown University

Brown's involvement with educational software and hypertext dates back thirty years to FRESS[6, 8] (File Retrieval and Editing System), a sophisticated multiuser hypertext system employing trails and user annotations, was used to teach poetry to undergraduates. The following decades saw the development of Intermedia[18], an object-oriented educational hypermedia system; a microprocessor simulator, MIDAS[11] and a 3-D synthetic camera simulator BUMPS[10], domain-specific applications; and the algorithm animation packages BALSAs[3], TANGO[16], more general frameworks for building interactive educational software. All were major monolithic projects involving months or years of work by advanced students. These systems taught us important lessons about the difficulties involved with the development and use of large educational software environments. In particular, we learned that the finer the grain of the module, the more likely it was to be useful in multiple contexts. We also learned the importance of reusability of components; here too, the finer the grain, the more likely the reuse.

## 3 Context

Our goal is to develop effective teaching applets for computer graphics that can be used for in-class demonstrations,

for self-study, and as assignment components. The context within which we work contains both significant resources and conflicting forces that have shaped our development approach and the resulting outcomes.

### 3.1 Resources

#### 3.1.1 Undergraduate Experience

Top-ranked computer science undergraduates at Brown may serve as TAs and as research assistants early on in their undergraduate careers. These experiences develop perspectives and skills normally unavailable on the undergraduate level.

In addition, CS15, the introductory programming course, has an unusual domain and scope: from the beginning class the programming model is completely object-oriented, the language is Java, the programs all use GUIs, and students are introduced to good software engineering principles, including simple design patterns[9]. By the end of their first semester, undergraduates have designed, programmed, documented, and demonstrated five increasingly difficult projects using Java, two-dimensional computer graphics, and design patterns. These students, one to two years later, are the developers of our graphics applets.

#### 3.1.2 Domain Expert Availability

While the actual software development on the applets has been done by undergraduates, domain experts serve a critical role in the development process. Typically members of the graphics research staff, they work closely with the undergraduate developers outlining sequences of applets and then consulting on the storyboarding, development, and documentation phases.

The domain experts benefit from working closely with novices because it is often hard for them to understand which parts of a concept or algorithm are difficult for a beginner to understand. They also must be able to explain their ideas clearly enough that the less experienced programmer can actually write the applet.

### 3.2 Forces

#### 3.2.1 Heterogeneity

As remarked above, the introductory computer graphics course must deal with great diversity in both subject material and student backgrounds; there is a particularly large variation in students' backgrounds in mathematics. This requires an approach that stresses flexibility in style, sequencing, and content.

#### 3.2.2 Multiple uses

As we said, we intend our applets to be used in many situations, including classroom demonstrations, hands-on laboratory use, self-directed work outside of class, and distance learning. Distance learning with the WWW adds a dimension, in that users may drop in to any applet without having seen related, perhaps prerequisite, material. We thus must design sites so that it is easy to discover and access related material from any individual applet.

#### 3.2.3 Undergraduate Developers

All software development for this project is done by undergraduates. This implies that there will be a rapid turnover in our development teams and that the time commitments of the applet developers will be highly unpredictable.

Among the adaptations we have found useful for this development model are internal web pages describing the previous work and resources available so that new students can quickly get up to speed on the project, and pattern-based templates for documentation that help students get

the lessons learned during development across to their successors.

### 3.2.4 Skill range

The production of good educational software requires many skills, including user interface design, software engineering, familiarity with pedagogical issues both in the domains addressed and with educational software in general, artistic abilities, information structure design, concept visualization and presentation, and knowledge of specific programming APIs.

No one student joining the development team will have all these skills. To leverage the skills of different students in the group, we encourage those with specialized skills in a particular area to assist others with things such as interface design. In addition to examining storyboards for programs and programs in development, students give short presentations describing important aspects of their projects at our weekly research group meetings. They also participate in peer storyboard and code reviews, which have quickly become an integral part of our development process.

## 4 Granularity

### 4.1 Coarse-grained applets

Our first Java applets were built as complete, relatively large stand-alone applications. Here, large applies to conceptual ground covered or features, not the actual size of the code-base — they were reasonably short. They illustrated many different sub-topics of the concept they were designed to teach, allowed the student to try many different combinations of parameters, and had features such as internal help systems.

These coarse-grained applets required significant development effort; only the most experienced developers could produce programs that were of a high enough quality to merit regular use in our introductory graphics class. They also took far longer to write than expected — approximately six months for new applets that weren't clones of programs from previous development efforts. We still are not sure why developing these applets, which were small compared to applications that took far less time to develop, was so difficult. And even though every effort was made to write reusable code, very little of the code was actually used in other programs. This is probably because the programs were too large for a new programmer to feel that they could read and understand the code, and because in any large system, the components begin to gain features (such as hooks to the help system) that make it hard for them to be used in other software.

The two useful applets we developed illustrate simple image scaling and lighting and shading. Both are similar to complete applications that use the algorithms they show. The lighting and shading applet lets the user vary the parameters of the lighting equations and see the results as simple objects are shaded. The difference between using the applet and a regular graphics package is that the applet shows the equations and vectors used in computing the lighting and it explains the coefficients. Likewise, the scaling applet lets users draw a filter that is then used to scale an image, as in Adobe Photoshop.

The main problem with coarse-grained applets is that they tend to show the algorithm in question along with only a few of the intermediate computations. For certain subjects — notably many of the topics encountered in an algorithms and data structures course — this is appropriate. For many other subjects, it is not. Many topics in graphics

arise from a sequence of small concepts that provide a theoretical framework for larger algorithms; the coarse-grained applets are useful in demonstrating the final algorithm, but are hard to use in teaching the individual ideas that build up to that algorithm.

## 4.2 Fine-grained applets

To address these issues, we tried a fine-grained approach, developing applets that deal with small, atomic concepts. The exact size is highly dependent on the domain being addressed and varies widely between domains. Our goal with the fine-grained model was to strive for the smallest possible scope for each applet while still covering a concept appropriately.

Several fine-grained applets related to signal and image processing build up to the previously mentioned scaling applet, providing more general coverage of the topic. Applets demonstrating the impulse response of a filter and properties of linear-shift-invariant systems lead to a discussion of one-dimensional continuous convolution, a central part of the theory behind image scaling.

The BUMPS[10] system, a single monolithic application for demonstrating a three-dimensional synthetic camera model, is being rewritten as a series of Java applets. The first will show how a parallel-projection camera is placed in the world and give a qualitative sense of what different camera parameters do; another applet will show the additional parameters available with a perspective camera. The next will illustrate the steps used to normalize a view volume; another applet will demonstrate the perspective transformation. The final applet shows how the canonical view volume is projected onto a 2D image.

Since the applets correspond closely with the course, they can be used by the instructor during class to explain the equations being derived, both where they would draw pictures to explain the concepts and where they allow the instructor to illustrate processes that could not be visualized in the past; this is in addition to being useful during lab sessions or for students exploring the material on their own. Most importantly, the applets can be set up quickly for new situations, so they are useful when the instructor gets unforeseen questions or takes a different track to clarify a misconception. We have built small applets covering certain topics that most of the students already understand — such as the dot product — in order to assist the few who do not, precisely because it is so easy to do so.

Once we shifted to a fine-grained model, our group's productivity increased. Each of the programs was far more manageable from a software engineering standpoint. On the other hand, more of the fine-grained applets are required to cover the same material as a single coarse-grained applet. Thus it required approximately the same amount of time for an individual developer to cover the same material as when we were writing coarse-grained applets. However, the advantage of the collection of fine-grained applets is that the same material is presented in a way that we feel is more pedagogically useful. Moreover, students new to the group could develop smaller applets to "get their feet wet" while gaining the skills to work on the more sophisticated ones; having so many more developers able to produce useful applets provided the largest productivity gains. Reuse also increased significantly, probably because each component had fewer dependencies and each of the programs was smaller and more readable: other developers could see how components were being used in one program, making it feasible to use them in another.

## 4.3 Variation in granularity

Not all of the applets for teaching a particular subject should be of the same size. For example, the first of a series of applets demonstrating scene-graph hierarchies allows the student only to change the values of certain nodes in the scene graph. Later applets start to add sections dealing with the traversal of scene-graphs and finally lets students to build their own scene-graphs from scratch. Sometimes this strategy calls for different applets and at other times it calls for versions of the same applet with certain constraints added.

Occasionally a much larger applet can be useful to culminate a series of small applets. For example, after a series of small applets dealing with the mathematics of transformations, our culminating applet, "The Transformation Game", asks students to use all the skills they've learned in the preceding set of applets to solve increasingly difficult puzzles. This applet is entirely different from all of the ones preceding it: the only commonalities are the concepts covered.

## 5 Consequences of fine-grained approach

The development of many applets using a fine-grained approach has both benefits and costs. The benefits include flexibility, reusability, and effective use of both undergraduate developers and domain experts. The costs include the need to provide supporting hypertext structure and descriptive materials, greater demands on user-interface design both in terms of consistency and transparency, and essential modifications to software engineering practices.

### 5.1 Benefits

#### 5.1.1 Flexibility

A primary advantage of the fine-grained approach is that, pedagogically, it allows the applets to be a much closer fit for the concepts that are being taught; each can have a specific idea that it is meant to get across or allow the student to explore.

Another advantage is that a collection of fine-grained applets provides great flexibility in the classroom and on the web. In the classroom, it is precisely when unplanned questions come up that tools that assist in demonstrating the concept can be of the most assistance. On the Web, visitors to a site will have vastly different backgrounds and different goals; this approach makes it easier for them to learn what they want to learn. Also, we found that we could make the fine-grained approach work well with our undergraduate development team.

#### 5.1.2 Reusability

The small size and emphasis on consistency across different applets facilitates the creation of components. We are using JavaBeans to create a publicly available component library that can be used by all team members as well as by others wishing to develop teaching applets.

Beyond single components, entire fine-grained applets are reusable by different courses. For example, we are starting a collaboration with Reinhard Klein[13] at Universität Tübingen in which we share applets, even though the courses have very different structures and goals.

#### 5.1.3 Enhanced Undergraduate Skills

Participation in the project has tangible benefits for the students involved. First, it gives them the opportunity to work closely with experts on their respective areas in examining different ways to develop the topic and its presentation. The students also learn a wide range of project development skills, including demonstrating their projects to a variety of

audiences (since there is a regular stream of visitors to the graphics laboratory). Finally, the project acts as a gateway for students to gain the necessary technical and group experience to move smoothly into other research projects in the lab.

## 5.2 Costs and Adaptations

The development of many applets using a fine-grained approach necessitates important changes to other parts of the software design process, the delivery context, and the modes of use for the software. The changes include developing suitable hypertext structure, creating appropriate user interfaces, and modifying software engineering practices.

### 5.2.1 Trails

In a large collection of applets intended for a wide range of uses and backgrounds, it is difficult to establish context, to ensure the acquisition of prerequisite knowledge, and to provide appropriate presentations of the material. In a single large application, the user's sequence of operations can usually be directed by the programmer. But with a large collection of individual applets, the number of possible ways someone could navigate among them is combinatorially huge.

While it is impractical to deal with all the different paths someone could take (especially since most do not make sense), we do want to present several different ways to traverse the site to take into account different backgrounds, different goals (some may only be concerned with getting enough information to implement something, or some subtopic), and different settings (demo, class assignment, self-study)

To balance these concerns, we present a few points of entry that lead to logical and coherent presentations of the material. We have found guided trails through the hypertext to be a key tool in accomplishing this goal. Trails in hypertext are not a new idea; there is a long history (dating back to Vannevar Bush's original proposal for the Memex<sup>[4]</sup>) of paths that guide the viewer along one particular development or thread.

The existence of trails does not mean that the user can't deviate from them — they are intended only to guide them from one topic to the next; at any point a user may choose from a variety of associated links. Once users have deviated from a trail, however, it should be easy for them to get back.

Trails can cover a variety of dimensions, levels, and approaches to a particular set of information, but there should, wherever possible, be a "main trail" which the author suggests the user follow. Having a main trail also provides focus for the author while doing the initial development of the site. In effect, the main trail functions as a sophisticated Table of Contents while the associated links serve as a distributed set of See Alsos.

### 5.2.2 Interface Issues

In moving to fine-grained applets, we have found several changes to the applets' interfaces to be useful. When moving between several different applets, each of which may only serve to illustrate a small point, students cannot be expected to learn to navigate a complicated interface with menus and icons. Not only do they take too long to learn, but they don't provide immediate feedback for actions and, they make the user feel more detached from what they are learning. This makes direct interaction instead of menu-driven interaction the primary interaction model for our applets. For example, instead of setting the values of a particular vector in a text field, the user can drag it around.

In successive applets, it is useful to develop a vocabulary of interaction tasks that the student can apply throughout

the sequence. In a series of signal processing applets, the student learns early on that dragging a slider below a graph widget (user interface component) shifts a filter function and evaluates the integral under the product of the signal that is being filtered and the shifted filter function. Since related applets may appear to have the same interface components but any particular applet may only allow specific components to be manipulated, it is particularly important to have strong visual cues, such as a color change on mouse-over, when building up an interaction vocabulary.

Constraints can help students quickly discover the concept that a particular applet addresses and help them see differences between other related applets. One applet demonstrating the dot product might tell the user to watch how the value changes proportionally to the length of a vector as its magnitude but not its direction is changed. Another has a direct manipulation handle that rotates both vectors at the same time to help them discover the dot product's rotational invariance. Coming up with good interaction models, especially constraints, typically requires the rapid prototyping of a number of styles of interaction, then comparing them side-by-side in an informal user study.

### 5.2.3 Changes To Software Engineering Practices

The fine-grained approach we have taken leads to several applets being developed to build up a single concept. Often the applets are very similar; in fact, to encourage ease of learning, consistency across applets written by the different developers is a major goal.

Though our undergraduate software engineering curriculum emphasizes reuse, we have found it difficult in practice to attain high reuse levels during development. Writing reusable code must be specifically encouraged as it is more difficult than merely writing code which works for the task at hand.

The first step is awareness; at our weekly meetings we demonstrate recently developed applets and make sure that everyone sees what components are used in the software. We are also writing a set of web pages to be a repository for reusable components. It contains short descriptions of the components, pictures of the components in action, and links to the code and documentation.

JavaBeans are becoming an important way of making sure that components are reusable and that it is easy to understand how to use them. They are, unfortunately, not a silver bullet: beans that use different sets of messages cannot interoperate without complicated adapters and some sections of code that merit reuse cannot easily be packaged as beans.

## 6 Availability

While the recently developed material is not yet publicly available, earlier work and project information is located at our web site: <http://www.cs.brown.edu/exploratory/>.

## 7 Future Work

### 7.1 Further Development

The first priority in continuing this work is the development of applets that cover a broader segment of the concepts in computer graphics. The new applets will cover a wider range of topics related to graphics, partially aided by new APIs including Java3D and Java Advanced Imaging. As we build up larger collections of applets for the various sub-fields of graphics, we will write more trails through the hypertext to cover different approaches to the material.

## 7.2 User Studies

We will begin user studies to compare both the effectiveness of using instructional applets in introductory graphics courses and to compare elements of the structure we have presented with other possible structures. We also plan on doing user studies of individual parts of our programs. However, we will not commence formal user studies until we have a far more extensive set of applets.

## 7.3 Handbook

Our eventual goal is to find more general principles for developing educational software of this sort. In order to encapsulate some of the lessons we have learned in our development process for internal and external use, promote consistency among applets, and quickly bring new students joining our development effort up to speed, we have started writing a handbook and style guide.

The handbook currently has four main sections: Planning, Design, Software Engineering, and Evaluation. The style guide deals not only with graphic design issues but also with interaction models and planning how applets and web pages communicate their messages. We are considering pattern languages[1] as a mechanism for encapsulating and expressing this information in certain areas.

To complement the handbook we are collecting materials for a resources database which contains information on conferences, institutions, people, projects, publications, software, and an educational technology time-line.

## 7.4 Navigation Tools

Large hypertexts are often confusing to navigate. Tools that assist the user in finding his or her way greatly improve ease of use. Such facilities have been available in hypertext systems since the 1960s but are lamentably lacking in HTML.

Such tools might be aware of where users currently are along a trail, where they could go next, what the prerequisite knowledge is for the current page and the next few pages along the trail, and use that information to select the correct elements to assemble web pages dynamically. HTML does not provide adequate support for such tools, but XML/XLink (aka XLL), which promises to supersede HTML, does. As a stopgap measure, we have started to use JavaScript to build navigation tools. These tools do not yet deal with trails, though rudimentary trail-related navigation tools are relatively straightforward to develop.

## 8 Acknowledgments

This work is supported in part by the NSF Graphics and Visualization Center, IBM, Sun Microsystems, and TACO. We would like to thank John Hughes for his extensive help with writing this paper and for serving as a domain expert *par excellence*. Anne Spalter, design guru and project director, and Matt Lerner have been essential to this effort and to the development of these ideas. We would also like to thank the current and former members of the illustrations team, especially Jeff Beall, Sascha Becker, Adam Doppelt, Marcin Romaszewicz, and Sam Trychin.

## References

- [1] Christopher Alexander. *A Pattern Language*. Oxford University Press, 1977.
- [2] Jeff Beall, Adam Doppelt, John Hughes. Developing an Interactive Illustration: Using Java and the Web to Make it Worthwhile. In *Proceedings of 3D and Multimedia on the Internet, WWW and Networks*. 16-18 April 1996.
- [3] Marc H. Brown and Robert Sedgewick. Techniques for Algorithm Animation. In *IEEE Software*, 2(1): 28-39. 1985.
- [4] Vannevar Bush. As We May Think. In *Atlantic Monthly*. July, 1945.
- [5] Andries van Dam. Exploratories: Computer Graphics in the Service of Education. *Talk Given at the Center for Innovation in Learning (CIL)*, Carnegie Mellon University (CMU). April 7, 1998. <http://www.cs.brown.edu/exploratory/resources/CMU-CIL/>
- [6] Steven deRose and Andries van Dam. The lost books of hypertext. To appear in *Markup Languages: Theory and Practice* 1.1 1998.
- [7] Educational Technology Timeline and Bibliography <http://www.cs.brown.edu/exploratory/timeline.html>
- [8] FRESS. [http://www.cs.brown.edu/memex/-HT\\_87\\_Keynote\\_Address.html#FRESS](http://www.cs.brown.edu/memex/-HT_87_Keynote_Address.html#FRESS)
- [9] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software* Addison-Wesley, 1995.
- [10] R. F. Gurwitz, R. W. Thorne, A. van Dam, and I. B. Carlbom. BUMPS: A program for animating projections. In *SIGGRAPH '80 Proceedings*, pp. 231-237. 1980.
- [11] R. Gurwitz, R. Fleming, and A. van Dam. MIDAS: A Microprocessor Instructional Display and Animation System. In *IEEE Transactions on Education*. February 1981.
- [12] G. Scott Owen. HyperGraph <http://www.education.-siggraph.org/materials/hygraph/hygraph.htm>
- [13] Reinhard Klein and L. Miguel Encarnação. A Web-based framework for the complete integration of teaching concepts and media in computer graphics education. In *Proc. ED-MEDIA '97*. Calgary, Canada. June 1997.
- [14] Port of Entry. <http://lcweb2.loc.gov/ammem/-ndlpedu/educator.html>
- [15] Ben Shneiderman. *Designing the User Interface: Strategies for Human-Computer Interaction*, Third Edition. Addison-Wesley Longman, Reading MA. 1988.
- [16] John T. Stasko. Tango: A Framework and System for Algorithm Animation. In *IEEE Computer*, 23(9): 27-39. 1990.
- [17] Virtual Laboratory <http://jersey.uoregon.edu/vlab/>
- [18] Nicole Yankelovich, Bernard J. Haan, Norman K. Meyerowitz, and Steven M. Drucker. Intermedia: The Concept and the Construction of a Seamless Information Environment. In *IEEE Computer*. January, 1988.