

# Developing an Interactive Illustration: Using Java and the Web to Make It Worthwhile

Jeff Beall

Brown University site of the NSF Science and Technology Center  
for Computer Graphics and Scientific Visualization

Adam Doppelt

Brown University Computer Science Department

John F. Hughes

Brown University site of the NSF Science and Technology Center  
for Computer Graphics and Scientific Visualization

## 1.0 Abstract

The World Wide Web has been recognized as a powerful tool for communication. Java has enhanced this by providing an opportunity for rapid transmission of “condensed information” in the form of locally-executable programs. Our own research has been in interactive illustrations for electronic books, with a belief that active reader participation yields better knowledge transfer. The next natural step is to combine these two things into Web-based interactive illustrations. However, the expected payoff — an enlarged audience and the opportunity to communicate more meaningful content across the Web — must be balanced against the development costs for the authors.

We describe the development process for a set of interactive illustrations on elementary color perception, the costs of converting them to Java, and the costs and the potential benefits for future Web-based interactive illustration development.

## 2.0 Introduction

### 2.1 What is an Interactive Illustration?

An *illustration* in a text is something — a figure, a table, a photograph — that complements the text. The most important feature of illustrations is their reliance on a visual rather than textual metaphor as a means of conveying information. In the case of a table, the geometry is used solely as an organizational tool. In a figure, the geometry is more likely to be representative of some aspect of the idea being illustrated. Because of the power of communication through the human visual system, we see graphs in newspaper articles about finance, photographs in magazines, and numerous figures in almost every textbook. More modern “texts,” such as those found on CD-ROM, may go further to include short segments of recorded video as illustrations, just as the evening television news includes video as well as spoken text. Viewing such illustrations is fundamentally *passive* for the reader<sup>1</sup>: the material is presented, and the reader sees it.

By contrast, an *interactive illustration* allows and even requires reader-participation. We consider an illustration “interactive” if the reader, to obtain the content, must do something more than invoke the mechanisms that present the illustration. Thus a figure in a book is not “interactive” because the reader must only open the book to see it; similarly, a video clip is not interactive because the user must only push a button to watch it. A familiar example of an interactive illustration is the kit of balls and sticks that accompanies many organic chemistry texts. This is an extreme case in which the reader must actively create the illustration.

---

1. We use “reader” to describe the person who is looking at the illustration, even if it is not in a traditional book, and reserve “viewer” to describe programs that display the contents of electronic books.

Our belief is that an ideal illustration involves considerable interaction by the reader, but the interaction must be constrained. Constrained interaction favors a guided experience, but allows for the possibility of an unstructured experience. Thus, in the illustrations described later in this paper, the reader can edit certain graphs and see how other graphs update in correspondence with the changes [1]. Tools are provided to help the reader edit the graphs in ways that we, as the authors, believe will help him understand the concepts we wish to demonstrate. However, the reader can only edit certain graphs in the illustrations since we wish to communicate the significance of the relationships between input and output graphs.

## 2.2 The Motivation For Interactive Illustrations

As characterized above — allowing and requiring reader participation — interactive illustrations may appear to require more from the reader, and would therefore be off-putting. While it is true that they require the reader to be more active, we also believe that a *well-designed* interactive illustration more than repays the reader's efforts. There are multiple reasons for this. Interaction is engaging in and of itself. The feeling of control and the ability to answer “what if” questions make the viewing of an interactive illustration a richer experience. Furthermore, readers can play with an interactive illustration at their own pace: they can experiment until they understand something, or they can simply play around a little bit if they find the illustrated concept easy to grasp.

To further motivate the value of interactive illustrations, consider the power of the spreadsheet. It is often used as an interactive illustration to chart the performance of a company or product. The “reader” will vary some parameter, like an interest rate, and see how other things are influenced — how the cost of production changes, for example. Consider a specific example of this: you read an article about investment in the stock market, and there is an illustration showing how a \$10,000 investment in some company made in 1980 would have fared over the subsequent fifteen years. You might want to ask “how would it have looked if I had invested the \$10,000 in 1981 instead?” That information can be determined from the graph with a little calculation, but if the graph accompanying the article were actually a spreadsheet file, you could simply edit it and see the results.

## 2.3 The Real and Perceived Wonders of Java and the World Wide Web

The World Wide Web's growth has been astounding, and its power to reach a huge audience is well-understood. This wide distribution channel has provided opportunities in organization (many clubs use the Web as a communication medium), marketing, news releases, and numerous other applications. As originally developed, however, the Web only provided a means of sharing *documents*, albeit some quite rich documents. File transfers provided a means for sharing programs, but hardware and software compatibility issues lead to problems — someone can download a program, but it is not guaranteed to run on her machine. Furthermore, many documents contain lots of data and therefore take a long time to transfer across relatively slow networks. This led naturally to the idea of a machine-independent language and platform-independent “viewable” applications: when you access something from the Web, what you get is not a piece of text or an image, but a small program. It is downloaded to your machine, runs there, and produces the images its author wanted to show you, for example. If the program is small and the set of images is large, this is a clear improvement over transferring the images themselves. The Java language has begun to provide this functionality [2].

## 2.4 Combining Interactive Illustrations with Java and the Web

Because the Web is a document transfer system, and many documents have illustrations, and because image-based illustrations are slow, there seems to be a natural conjunction of Java and the Web for illustrations. Downloading small bits of program text that can generate an image may be much faster than downloading the image itself. Of course, many illustrations are not easy to generate programmatically, but for those that are, this can be an important advantage. For documents with *interactive* illustrations, the ability to download a program is essential: transfer rates for images are currently too slow to allow for a client-server model in which the client is an image-display program. The result is that the interactive illustration gets a wider audience, which helps to amortize the high cost of production (more on this below).

This seems ideal: the author gets wider readership, the costs of production are amortized (assuming, for the moment, that recognition is all the author wants by way of compensation), and the network and reader's hardware are both used efficiently. There are associated costs as well, and these are discussed in Section 5 below. The lessons we learned in developing our interactive illustrations for the Web were that developing a Java/Web illustration takes a lot of effort, it is easy to expend that effort inefficiently, and it is important to gear the content towards Web presentation from the start.

In the remainder of this paper, we'll describe the history of creating interactive illustrations at Brown University, our own experience in developing the color-perception illustrations described here, and then conclude with some observations about the illustration development process.

### 3.0 The History of Interactive Illustration Development at Brown University

The first large-scale effort at developing interactive illustrations at Brown came about in the 1980's, when Marc Brown developed *Brown ALgorithm Simulator and Animator* (BALSA) [3]. BALSA was used to help teach undergraduate courses on a network of Apollo workstations. Algorithms such as bubble sort, quicksort, extendible hashing, and breadth-first search were illustrated. In a typical illustration, the reader saw a program running and one or more visual representations of a data structure within it. The reader could step through the program and the visual representation of the data structure would change as steps were executed. Thus, in bubble sort, the two elements being compared would be highlighted, and then if they were to be swapped, they would blink. The basic idea was that certain lines of code were "annotated" with actions that occurred when the line of code was executed and highlighted in the "code view" window.

BALSA was a wonderful system for animating algorithms that operated on data structures, which was its intended use. It was used to teach the introductory data structures and algorithms course for many years. Each illustration in BALSA was therefore reused multiple times, and this compensated the considerable expense (in terms of programmer time) of annotating an algorithm. Based on BALSA, a later system, *TANGO*, was developed to include various animation features as intrinsic actions, so that it was easy to make smooth transitions (the two items being swapped in a sort could easily be made to smoothly change places) [4].

Shortly thereafter, Professor Thomas Banchoff of the Mathematics department began working with two undergraduates to develop *Fnord*, a mathematics visualization package. The idea was to be able to write mathematical expressions and see the results. There was no symbolic calculation such as that of Mathematica, but there were far richer facilities for graphical display. Courses in our Mathematics department continue to use *Fnord* to this day, and Professor Banchoff has written an electronic differential geometry text with *Fnord*-based illustrations. *Fnord* is an expressive language, but interaction with it is through a pre-defined metaphor: one of "adjustable constants." Certain entities within the language can be defined as the value of a function called *Slider*; so that one might write  $b = \text{Slider}(0, 100, 50, \text{"curvature"})$  to indicate that there should be a slider placed on the screen, labelled "curvature", with minimum and maximum values of 0 and 100 respectively and an initial value of 50. When the user adjusts this slider, the value of  $b$  is changed, and all things that depend on  $b$  are updated as well. This, together with a kind of virtual sphere interaction for changing viewpoints within 3D windows, constitutes the entire interaction model. Because of this, *Fnord* can only provide a limited set of possible interactive illustrations.

Having built previous animation and rendering systems, the Brown Computer Graphics Group undertook in the late 1980's to build a new system with the express purpose of aiding the creation of interactive illustrations [5]. The basic notion was that an object-oriented system would provide enormous leverage in interactive illustrations, because the entities in an illustration could become active: each would know something about how to behave in response to various stimuli. The object model we chose was not the standard class-instance model, but rather a delegation hierarchy in which objects were copied from other objects and then had certain methods overridden or new methods added. With a rich collection of basic operations defined on a standard object from which others were derived, we had a system that made it easy to produce a variety of interactive illustrations and

illustration-building tools. We developed several 3D widgets to help lift modeling operations from textual interfaces to “within the model’s world” interfaces, and tried in general to stress a direct-manipulation model whenever possible [6][7]. Because the system is based on a custom, interpreted language *Flesh* using the *Trim* interpreter, development of illustrations through rapid prototyping and frequent testing is very easy. The existence of *Flesh/Trim* and their power in helping with illustration development is the starting point for the story of the Java illustrations discussed in this paper.

## 4.0 The Story of the Color Perception Illustrations

Students in our introductory computer graphics course had a difficult time grasping the difference between “color” as a perceptual phenomenon, and “color” as a physical property of spectral light. Furthermore, the various models for representing color in computer graphics seemed baffling and the students had a hard time understanding why “just multiply the RGB values” did not always give the right answer to the question “what does the light reflected from this surface look like?” One of the authors, John Hughes, had some ideas for how to communicate these notions, but when he used them in the class — with hand-drawn figures — they were not entirely successful. He realized that his own internal sense of the things he was drawing was far richer than the drawing themselves and that the limitations of his drawing skills were a hindrance. Furthermore, when he drew a set of function graphs, he was internally seeing not only the particular graphs, but the underlying relationships between them — the sort of thing that would become obvious to a student who saw that whenever one graph changed, so did certain others. This motivated Professor Hughes to draw out a storyboard for a set of interactive illustrations about color and color perception.

At this point another of us, Jeff Beall, took these storyboards and used *Trim/Flesh* to make some rapid prototypes of the illustrations. Fortunately, he had considerable prior experience authoring *Flesh* applications and therefore was able to use the language effectively. Nonetheless, it took about two weeks of his time to create a “function graph” object that was user-editable and could be connected to other function graphs in particular ways, and to create a “bar graph” object that could take values from function-graph objects and display them. As he developed initial models of these

objects, the rapid prototyping was essential. For example, the “function graph” was really an array of thin rectangles side by side, and the number of such rectangles that could be displayed without unpleasant delays had to be determined (see figure 1). Furthermore, since *Trim* is really a 3D system, these rectangles were actually cuboids viewed from one side; the mouse-interaction with these cuboids had to be overridden so that the user could not accidentally rotate them. As we played with editing the graph objects on the screen, the need for certain graph-editing metaphors became evident (i.e. “make this a locally-constant section with this value”).

The illustrations were completed in time to be used in a special lab session for the introductory computer graphics course during the fall of 1994, but the results were not good. Simply put, the illustrations were too slow to be of practical use. This brought home to us the importance of *interaction*: in an interactive illustration, the interaction can be as important as the thing illustrated. If the feedback from the interaction is sluggish enough that readers’ actions and their results become

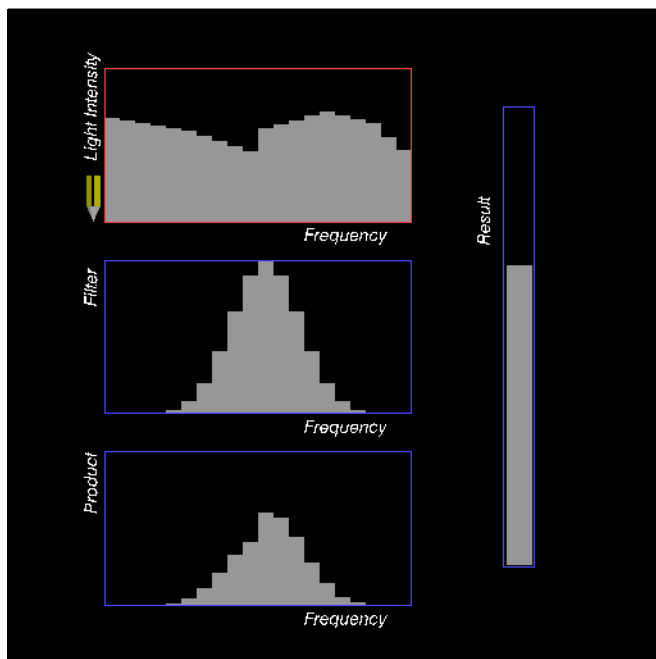


Figure 1: One of the color perception illustrations written in *Flesh*. The roughness of the graphs comes from using cuboids as elements, which was necessary given the constraints of the *Trim* system.

unrelated, then the illustration fails. It is easy to interact with an illustration in the “right” ways as an author, but the readers of that illustration will approach it from a different perspective. They regard it as a thing to explore and will often explore it in multiple ways before they see the author’s intended point. We realized that our system, carefully crafted to facilitate the *creation* of interactive illustrations, was not practically usable as a viewer of these illustrations, at least not when the illustrations were sufficiently complex.

#### 4.1 A Second Draft

We took the illustrations, whose conceptual flow and overall design we now believed in despite their failure in the lab, and felt that rewriting them in a compiled language would add the speed necessary to test our beliefs. Since we expected the subsequent alterations to be relatively minor, the increased turnaround time for revisions appeared to be a fair trade for the increased execution speed. The illustrations were rewritten in C++ during the summer of 1995 with a streamlined design and implementation. The reworked illustrations included cleaner inheritance relationships and better data structure that made the required numerical integrations particularly efficient. The result was that the graphs in the illustrations could have many more sample points and still achieve a level of interactivity we had never seen in *Flesh/Trim* (see Figure 2).

The new design used an object-oriented 2D graphics library, *GP*, that had been widely tested in our introductory programming course [8]. Because *GP* includes as primitive objects a great many interaction tools, the user-interface component of the conversion was relatively simple. The time to develop the revised version of the illustrations was roughly two weeks. It was this short primarily because of the rapid-prototyping that had been previously done in *Flesh*. Indeed, the main work involved was simply converting from *Flesh*’s delegation model to the class-instance model of C++ and optimizing the drawing routines for better interaction response.

The end result was the same set of illustrations, but with much better response to user input and with far more finely-sampled function graphs. Another one of the authors, Adam Doppelt, who was working with Java at the time and had seen the illustrations, thought it would be interesting to see them become more widely available. Recognizing that the first step to portability was making a Java version of *GP*, he converted the *GP* class library to Java, thus allowing the illustrations to have exactly the same user interface that they currently had. The remaining code was converted with an almost line-by-line conversion from C++ to Java, with only a few small adjustments required where an implementation had used a pointer to represent an association between objects. Once these pointers were removed, the conversion was complete.

When the undergraduate graphics course’s section on color theory occurred again during the fall of 1995, we decided to try the color lab using the updated illustrations. The lab in the second trial was a complete success. John Hughes gave brief lecture segments, and then the students played with the relevant illustrations for a while to experiment with the lecture concepts. There were no long “load up” times, so the illustrations and lecture ran comfortably together in this alternating format. The responsive and engaging visuals encouraged students to play, and even those students who did not completely understand the whole lecture got a feeling for the interaction between the spectrum of a light and its “red-green-blue” perceptual values (see Figure 3). The students remarked that they

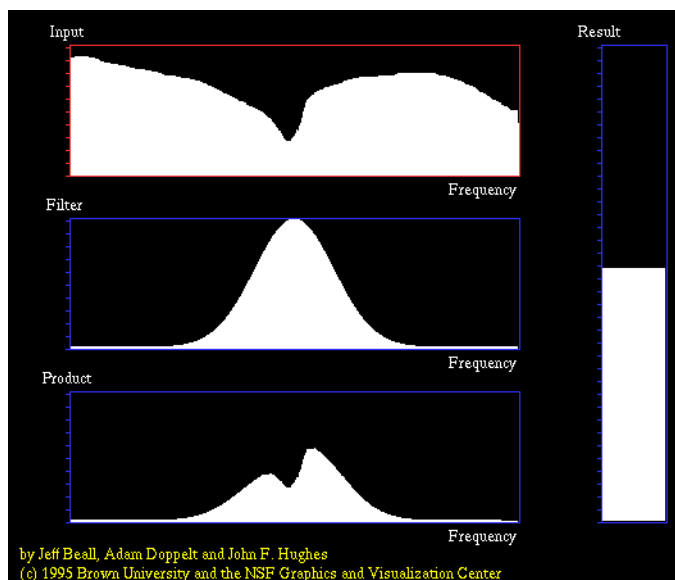


Figure 2: The Java version of the same illustration shown in Figure 1. The graphs are much smoother because the illustration was optimized for Java’s graphics support.

found the illustrations helpful, and several students went back to spend more time with them after the lab.

## 5.0 Developing Interactive Illustrations

The lessons provided by our experience are sobering. First of all, interactive illustration building is costly. To make a set of six illustrations, all of which are variants of one basic theme, took about five weeks of programming work. By contrast, producing the figures for a whole chapter of a computer graphics text took a Macintosh user about one day using a conventional drawing program [9]. Of course, none of those static illustrations conveys as much information as one of the interactive ones, but the disparity in cost is striking nonetheless. And the cost comes in two parts: the frequent redesign of the illustration itself — this is “author time” — and the subsequent reimplementation — “programmer time.”

Second, much of the time spent in developing the illustrations was spent in deciding how to present the content in the best way, albeit long after the basic format of the illustrations was determined. This is in sharp contrast to the time spent on presentation issues for static illustrations. Occasionally in creating the textbook figures mentioned above, it was necessary to reposition an element so that the labelling was clearer, but the pencil sketch of the figure carried essentially all the information about the final result. By contrast, for an interactive illustration there are issues of what sorts of interaction will be significant, what sort of speed is necessary (i.e., acceptable trade-offs between complexity and feedback), and what initial values for the editable components of the illustration are appropriate.

Third, good development environments come with a cost. Interpreted environments are flexible but slow. Interpreted code can easily execute so slowly that one cannot accurately judge the nature of the interaction in an illustration without substantial prior experience. The ability to change objects on the fly so that one can experiment with slight variants of a design may cost so much in terms of performance that the designs one is experimenting with are only represented approximately.

### 5.1 Using Java

Java, and the attendant possibility of distribution on the World Wide Web, presents interactive illustrations to a wider audience. The existence of cross-platform support for Java makes the author of an illustration less dependent on a particular computing environment, and the relatively small executables that result from compiled versions of interactive illustrations make for easy distribution.<sup>2</sup> Java’s built-in libraries for graphics support are particularly important in this regard. The small code-size of the executable, which is a big help in distribution, reflects the fact that the code is for a specific illustration rather than being for a whole illustration-development-and-viewing environment.

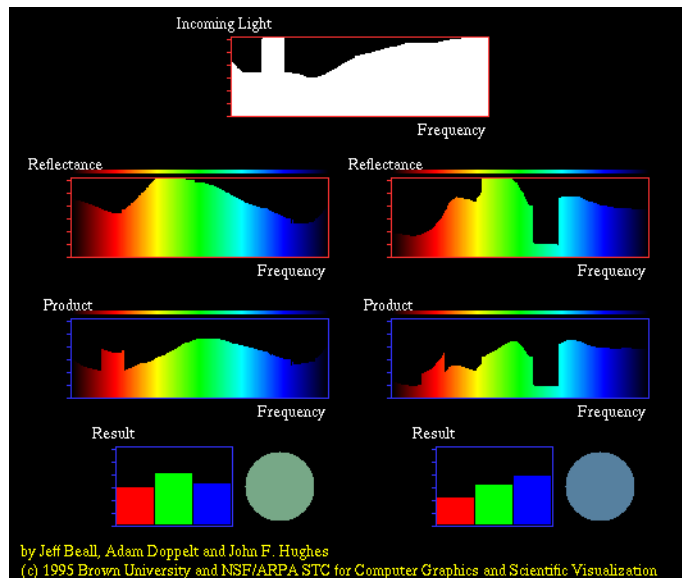


Figure 3: One of the color perception illustrations demonstrating the relationship between the spectrum of light, an object’s spectral reflectance, and its corresponding RGB color values.

2. By contrast, the Trim/Flesh system described above only runs on workstations such as the ones at Brown University, and the executable is about four megabytes. The system’s run-time requirements are even greedier!

However, Java is not a good environment for rapid prototyping of illustrations. Just as with C++, the nature of the language encourages a pre-planned design rather than one that evolves through prototyping. We believe that the success of our illustrations came precisely from the ability to let the design evolve as we developed them.

We thus believe that a good way to use Java in interactive illustration development is to start by developing in an interpreted-language environment, which although slow is very effective for realizing and developing concepts. Once a stable prototype is created, then the illustration can be reworked in Java. This is analogous to making a paper-and-pencil sketch for a figure first, and only then using a conventional drawing program to make a clean computer-based version. The “reworking” in Java should include consolidation of the design — a change of language is a good time to think through a design and discard unnecessary elements, combine redundant classes, etc. — and careful consideration of the speeds of the most critical components of the design, so that interactivity can be optimized.

Using Java helps amortize the cost of the author’s and programmer’s time because a wider audience can see the illustrations. (Of course, the amortization only happens if publicity is considered valuable — no direct payment for the content of the work accrues to the author or programmer.) Ironically, the Web which is so good for distributing the illustrations, turns out to be awkward to use when one wants to distribute the accompanying text which naturally contains some mathematics. The failure of HTML to include the full power of SGML means that any formatted equations must be sent as bitmaps rather than as marked-up text. An improvement in this situation may come in the form of embeddable SMGL viewers.

## 6.0 Future Work

We look forward to translating several other interactive illustrations to Java. Unfortunately, since these illustrations rely on the 3D aspects of our system, they cannot yet be converted. Work has been done at Brown to develop Java wrappers around Silicon Graphics’ Inventor library, which may provide the capabilities we need [10]. In addition, the recently announced VRML 2.0 specification integrates Java applets into the behavior of scene objects [11]. This feature and other new additions described in the specification show a great potential for authoring Web-based 3D interactive illustrations using VRML 2.0.

We also hope to develop larger collections of text and accompanying illustrations, forming true electronic books available for worldwide viewing. The potential for these illustrations is enormous, because so many concepts — especially those in which one tries to express the relationship among variable entities — lend themselves to interactive exposition.

## 7.0 Conclusions

Interactive illustrations have proven themselves to be effective learning tools because of their ability to present meaningful content within a guided interaction framework. As our experiences have demonstrated, illustration development in interpreted systems is effective but the results tend to be unusable. Our color perception illustrations were initially unsuccessful as teaching tools because of their inability to provide useful feedback to the reader. Once they were converted to C++ and later to Java, they gained the necessary speed increase to make the illustrations worthwhile. The Java versions have the added benefit of allowing us to share our work and experience as instructors with anyone in the world who has a Java-compatible Web browser. This is where the real potential of the Web begins to show itself. However, while using Java as a platform for interactive illustrations guarantees a large potential audience, Java itself is not likely to be the best platform for development. Java, like C++, has an overall structure that does not encourage the rapid prototyping that we feel is critical for successful illustration design. By first developing interactive illustrations in an interpreted system and later converting them to Java once they are “done,” the costs of development are balanced with their speed and accessibility.

## 8.0 Acknowledgments

This work was sponsored in part by the NSF Science and Technology Center for Graphics and Visualization. Additional support was provided by grants from Microsoft Corporation, Sun Microsystems, TACO, and NASA. We would also like to thank Hewlett Packard, Apple, and Silicon Graphics for donating equipment, and the members of the Brown Computer Graphics Group for their support.

## 9.0 References

- [1] Beall, J., A. Doppelt, and J. Hughes. "Interactive Illustrations of Color Perception," <http://www.cs.brown.edu/research/graphics/projects/igi/spectrum/>, Brown University, 1995.
- [2] Gosling, J., and H. McGilton. "The Java Language Environment: A White Paper," <http://java.sun.com/whitePaper/java-whitepaper-1.html>, Sun Microsystems Inc., 1995.
- [3] Brown, M. *Algorithm Animation*, Ph. D. Thesis, Brown University, Providence, RI, 1987.
- [4] Stasko, J. *TANGO: A Framework and System for Algorithm Animation*, Ph. D. Thesis, Brown University, Providence, RI, 1989.
- [5] Zeleznik, R., D. Conner, M. Wloka, D. Aliaga, N. Huang, P. Hubbard, B. Knep, H. Kaufman, J. Hughes, and A. van Dam. "An Object-Oriented Framework for the Integration of Interactive Animation Techniques," *SIGGRAPH 91*, 105-112.
- [6] Snibbe, S., K. Herndon, D. Robbins, D. Conner, and A. van Dam. "Using Deformations to Explore 3D Widget Design," *SIGGRAPH 92*, 351-353.
- [7] Zeleznik, R., K. Herndon, D. Robbins, N. Huang, T. Meyer, N. Parker, and J. Hughes. "A Toolkit for Interactive Construction of 3D Interfaces", *SIGGRAPH 93*, 81-84.
- [8] Conner, D., D. Niguidula, and A. van Dam. *Object Oriented Programming in Pascal: A Graphical Approach*, Addison-Wesley, Reading, MA, 1995.
- [9] Foley, J., A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics: Principles and Practice*, Addison-Wesley, Reading, MA, 1990.
- [10] White, J. "Kahlua," <http://www.cs.brown.edu/people/jsw/kahlua/>, Brown University, 1996.
- [11] "VRML 2.0 Specification," <http://cosmo.sgi.com/spec/>, Silicon Graphics Inc., 1996.

## 10.0 About the Authors

Jeff Beall ([jeb@cs.brown.edu](mailto:jeb@cs.brown.edu)) is an undergraduate at Brown University, whose research has largely been in authoring interactive illustrations. He will be beginning his Master's degree at Brown in the fall of 1996. Adam Doppelt ([amd@cs.brown.edu](mailto:amd@cs.brown.edu)) is also an undergraduate at Brown University and has released a number of popular Java applets to the public domain. John F. Hughes ([jfh@cs.brown.edu](mailto:jfh@cs.brown.edu)) is an Assistant Professor of Computer Science at Brown University. He co-directs the Brown Computer Graphics Group with Andries van Dam. The Java interactive illustrations described in this paper were awarded third place in Sun Microsystems' Java Applet Programming Contest in the fall of 1995. They can be found at <http://www.cs.brown.edu/research/graphics/projects/igi/spectrum/> and work with both the Alpha and Beta versions of the Java language.