# Isis, Cabbage, and Viper:
## New tools and strategies
## for designing responsive media

by Stefan Panayiotis Agamanolis

B.A. Computer Science
Oberlin College, 1994

M.S. Media Arts and Sciences
Massachusetts Institute of Technology, 1996

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Media Arts and Sciences

at the Massachusetts Institute of Technology

June 2001

Author:
.............................................................................................................................................
Program in Media Arts and Sciences
May 4, 2001

Certified by:
.............................................................................................................................................
V. Michael Bove, Jr.
Principal Research Scientist
MIT Media Laboratory
Thesis Supervisor

Accepted by:
.............................................................................................................................................
Stephen A. Benton
Chair, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

# Isis, Cabbage, and Viper:
## New tools and strategies
## for designing responsive media

by Stefan Panayiotis Agamanolis

## ABSTRACT

*Responsive media* are media that can sense and react intelligently to factors like presentation conditions or equipment, audience identity or profile, direct interaction, history or anticipation of involvement, and so on. The emergence of modern computational devices and communication networks, with their power to manipulate media signals and objects, has spurred the development of new forms of responsive media. But a lack of a broad framework for understanding the characteristics of and motivations for these media has resulted in computer-based design tools that do not mirror the character of our projects and working practices and, in turn, compromise our ability to communicate effectively with these media.

This research strives to build such a framework and use it as a foundation for a suite of three new tools that better reflect the multi-layered, multi-sensory, and multi-person nature of responsive media. Aiming to rethink the principles underlying one of the most primary building blocks in the design process, the centerpiece of this suite, *Isis*, is a new programming language tailored to serve as a basis for responsive media. The second tool, *Cabbage*, is an experiment in employing a purist form of case-based reasoning in a system for creating responsive graphical layouts. Lastly, *Viper* is a new tool for making responsive video programs that can re-edit themselves to adapt to different viewing situations. These tools seek to enable the development of complex and meaningful input-output relationships through simple, elegant interfaces that emphasize visibility, accessibility, and extensibility.

# Doctoral dissertation committee

Thesis Advisor:
............................................................................................................................................
V. Michael Bove, Jr.
Principal Research Scientist
MIT Media Laboratory

Thesis Reader:
............................................................................................................................................
Glorianna Davenport
Principal Research Scientist
MIT Media Laboratory

Thesis Reader:
............................................................................................................................................
William J. Mitchell
Dean of the School of Architecture and Planning
Massachusetts Institute of Technology

# Acknowledgements

I want to thank a number of people who have assisted and supported me throughout my time at the Media Lab.....

First and foremost, Mike Bove, for giving me the chance to study at the Media Lab and for being the best advisor anyone could ever hope to have here.

Glorianna Davenport, who has in many ways been a second advisor and who has been just as influential in shaping my ideas and research directions.

Bill Mitchell, whom I have known for less time, but whose insightful comments from an outside point of view have been invaluable in honing my ideas.

Aisling Kelliher, Barbara Barry, Surj Patel, Bill Butera, Jacky Mallett, Alex Westner, Freedom Baird, Jon Dakss, Shawn Becker, John Watlington, and the many other Object-Based Media mavens and Interactive Cinema gurus of the past and present, for being great collaborators on so much of this work, and for being great friends.

My UROPs, Jeremy Lilley, Ann Bui, Christian Baekkelund, Matthew Palmer, and Eric Syu, for their hard work and long hours.

Pat Turner, Michail Bletsas, Jane Wojcik, Matt Tragert, and the many other staff people at the lab for keeping things running smoothly.

My officemates over the past 6 1/2 years, Chris Verplaetse, Steve Waldman, Dan Gruhl, Kathy Blocher, and especially Bill Butera, for keeping things real.

Richard Salter, Rhys Price Jones, Bob Geitz, Garold Garrett, and the many other figures from my educational past at Oberlin College and Copley High School for challenging me to be my best.

Annette, for her love and for always showing me the way to a good heart.

And finally, my parents, my sister, my grandparents, and the rest of my extended family, for their constant love and support throughout my life.

# Table of Contents

# Chapter 1
# Overview

## Introduction

*Responsive media* are media that have the ability to sense and react intelligently to factors like presentation equipment or conditions, audience identity or profile, direct interaction, history or anticipation of involvement, and so on. Unlike the classic media model, responsive media systems consist of a two-way channel between the audience or participants and the person or device generating or controlling the delivery of the media. This channel, whether physical or conceptual in nature, allows important data gathered at the delivery points to be returned and used to alter the presentation in a suitable way based on the goals of the experience.

Responsive media have existed since the beginning of civilization, in forms of oral storytelling that incorporate listener feedback and interaction for example. Thousands of years later, the emergence of modern computational devices and communication networks has created tremendous opportunities for developing new forms of responsive media. In the last quarter century alone we have seen the rise of things like video games, virtual reality, electronic information kiosks, interactive computer-based art, and the World Wide Web.

With their power to manipulate media signals and objects, computers make effective devices for mediating response behaviors, especially in cases where very fine grain control or instantaneous reactivity are necessities, or in situations where, for whatever reason, it is impossible or improper for a human to handle response activity from an audience. The programmability of such devices enables a creator to embed a sort of "philosophy of design" within a media presentation such that it has a sense of its delivery context and can modify itself appropriately to forge a richer connection with its audience. Advances in communication network technology have allowed the end points of a media experience to be great distances apart while still maintaining the capability for immediate feedback.

It is not surprising, then, that scores of computer-based tools for designing various kinds of responsive media have emerged in recent years. These tools vary in scope from general-purpose development environments, able to handle many forms of media, to tools tailored for very specific application domains, such as computer-based training systems, hyperlinked video, or adventure games. They also vary in user interface from completely textual programming languages to graphical flowchart and timeline-based environments.

## The problem

Over the years, with the introduction of modern operating systems and graphical user interfaces, the sophistication and accessibility of responsive media design

tools has improved somewhat, but the state of the art is still quite primitive, largely because we still lack a good understanding of the fundamental qualities of and opportunities presented by responsive media and how we, as designers, think about and work to create them. This lack has resulted in tools that exhibit a number of difficulties, because they fail to reflect the observations that responsive media consist of **multi-sensory interfaces and materials**, **multi-person collaborations and experiences**, and **multi-layered tasks and thinking processes**. These difficulties present obstacles in our ability to communicate effectively with responsive media and, in turn, stifle the enormous potential these media could have in our world and in our lives.

## Multi-sensory interfaces and materials

Responsive media are arguably at their best when they engage all of the senses through a variety of interfaces. Unfortunately, many tools, especially those on the commercial market, focus on mass consumption and concentrate support on only the most standard interface and delivery devices. Keyboards and mice are often the only options, which by their very form constrain many applications to a single small screen and a muted sense of physicality at best. Non-standard devices, if they are supported, are usually de-emphasized in different ways, especially in documentation. Additional constraints are imposed by the window managers and graphical widget offerings of the current popular platforms. Such tools foster an attitude that key presses and mouse movements are the only interesting or practical things an application might respond to, discouraging the use of sensing and display systems that may be more appropriate in situations where mass consumption may not be a concern at all.

Further, responsive media generally incorporate a rich array of media types, such as images, audio, video, and graphics, and applications frequently require complex analysis and recognition systems as well as a high degree of distributed processing, but tools are often not up to par in their ability to handle these demanding media types and operations, especially those that employ virtual machines or that require hefty overheads for data filtering and buffering, memory management, or multi-threaded processing.

## Multi-person collaborations and experiences

Responsive media projects often span multiple disciplines and involve many different kinds of people with many different kinds of expertise working in large collaborations to realize a common goal. Yet the vast majority of computer-based media tools are designed for a single user working at a single workstation and do not foster encounters where several minds might work together on a particular aspect simultaneously. This problem is compounded by the mere one-person physical design of modern computer workstations. These issues in turn discourage the creation of media experiences that might engage more than one person at a time, or that might utilize the resources of more than one processing node at a time.

## Multi-layered tasks and thinking processes

Making responsive media involves working incrementally and innovating at many different levels throughout the development process, from low-level matters like incorporating a new sensor or display technology to higher-level concerns like structuring media objects and their behaviors and adjusting content elements. Many tools offer more than one layer of abstraction, such as a graphi-

cal interface coupled with a textual scripting language, where each layer may present a different kind or amount of functionality or a different way of describing operations. This multilevel approach works well if a unified scheme exists in which the individual layers of a tool are consistent with and complement each other, and if it is easy for users to shift between and harness the power of several layers throughout the evolution of a project.

Unfortunately, this is often not the reality, especially with many of the most popular tools, which evolved, in some cases haphazardly, over several years in order to adapt to a new class of problems they were not originally designed to handle. Assumptions made and expectations created in one layer are often over-ruled and contradicted in the others, creating situations where designers must unlearn knowledge gained from using one layer when it becomes necessary to step into another in order to accomplish a task. These problems create additional difficulties in group endeavors by hampering the abilities of collaborators to understand and merge individual contributions based in different layers of a tool.

In addition, our ways of thinking about responsive media are widely varied, and whenever possible and appropriate, tools should provide interfaces that mirror these often visual thinking strategies and our inherent ability to learn and describe processes by example, as well as the increased emphasis of space and physicality in many responsive media domains. However, many tools instead require designers to describe response behaviors within very rigid systems of abstract rules and lists of instructions that usually don't reflect their internal understanding of the problems they are trying to solve. Many interfaces are difficult to extend and reconfigure, discouraging designers from breaking new ground by journeying beyond the bounds of what is offered by a tool. This often leaves designers with no choice but to base their efforts within the same general-purpose programming languages created decades ago for developing compilers and operating systems, before it was realized by most that the computer would become such a major player in responsive media.

# The journey

The research described in this dissertation uses these three basic observations about the multi-sensory, multi-person, and multi-layered nature of responsive media as the foundation for an approach for building a new genre of computer-based tools, and for evaluating those tools.

To begin the journey, Chapter 2 takes a step back and presents a brief history of responsive media forms and a discussion of some of the incentives for creating responsive media in the first place. Drawing on examples from prior work in diverse areas from interactive art to video games, this section also expounds on the three main observations described above, first explicating some basic properties of responsive media and some of the issues and techniques involved in creating them, and then presenting an evaluation of several current popular authoring tools and programming languages, leading toward a coherent strategy for constructing new tools.

The knowledge and insight gained from this exploration serves as the foundation for a suite of three new design tools. Chapter 3 describes the centerpiece of this suite, *Isis*, a new programming language, tailored to function as a basis for responsive media, that aims to rethink many of the principles underlying this perhaps most basic yet powerful implement at our disposal in creating innovative new media. Chapter 4 discusses *Cabbage*, an experimental visual tool, motivated in part by some of the shortcomings of Isis, that employs a purist form of case-

based reasoning in a system for creating responsive graphical layouts. Chapter 5 describes *Viper*, a tool for making video programs that can re-edit themselves during playback to adapt to different viewing situations and real-time audience activity, whose design is informed by the lessons learned from building the previous two tools.

These tools all strive to provide simple, elegant interfaces that enable a creator, at one level, to build complex and meaningful input-output relationships for controlling media behaviors, but at another, to programmatically express a "philosophy of design" that travels along with the media objects and dictates how they are presented in response to whatever factors are of interest. A host of prototype applications, including hyperlinked video dramas, telepresence environments, responsive television advertisements, ambient information displays, and interactive artworks serve to test these tools and to support the validity of the three basic observations about responsive media that form the basis of their design. Chapter 6 summarizes the contributions of this research and outlines several directions for further investigation.

# Chapter 2
# Understanding responsive media

## Terminology

In their seminal book on the mathematical theory of communication, Shannon and Weaver describe the symbolic components of a classic communication medium, in which messages are conveyed from a transmitter to a receiver over a communication *channel* [SW63]. Their definition implies that the channel is uni-directional—that messages may pass from the transmitter to the receiver, but not in the reverse direction.

A *responsive* communication medium, on the other hand, may be defined as a system consisting of a two-way channel, over which messages may pass from the transmitter to the receiver, but also from the receiver back to the transmitter. In such a medium, the technological distinction between the transmitter and receiver vanishes, and any remaining difference is one of subjective function. In many cases there is a clear distinction between the "originator" or "mediator" of an experience and the "audience" or "spectators," and in others the function of each party is more or less equal.

The term *responsive media* can be defined broadly as the collection of media forms in which there is any kind of response characteristic by which a presentation is altered based on one or more factors, such as information gathered about the audience, the type of presentation equipment in use, the physical or social context of the presentation, direct interaction from the audience, knowledge of what happened at previous points in time, or anticipation of what might happen in the future. At the very least, all responsive media forms create a two-way relationship between the parties in a media transaction that may involve physical sensing apparatus, access to memory storage devices, and systems for processing and analyzing that sensory and memory information.

The more familiar term *interactive media* has come to represent a large subset of responsive media that depends primarily on responses that are actively willed by participants. For example, arcade video games require the active and intentional button presses and joystick movements of players, whereas traditional theater productions often rely more on the passive and unwilled responses of audience members, such as laughter, facial expression, other attention indicators, or a lack thereof. *Personalized media* represents another subset that stresses situations where information about the audience, such as personality, preferences, interests, history, and so on, is utilized to alter a presentation accordingly.

# A brief history

Some of the earliest instances of responsive media are likely related to forms of oral storytelling that incorporate elements of listener involvement and feedback, and these forms are still very much alive today. By taking notice of how much an audience laughs, how attentive they seem, how disgusted they are, and so on, oral storytellers can alter the way present their material in real-time in order to tell the most engaging story possible. They can also respond to direct questions from the audience and steer the content of the presentation to better suit interests or preferences.

Addressing the reactions of an audience while simultaneously maintaining flow and delivering a compelling experience with a rich sense of reciprocity is by no means a trivial matter, as storyteller Rafe Martin stresses:

> "The teller must be able to look within to the unique images and understandings of the tale arising in his or her own mind, even as he looks outward to the needs, understandings, and rhythms of the audience. In the act of telling, the storyteller must be constantly gauging, constantly modifying and reshaping the story. In the act of telling, the teller is intuitively responding to such questions as: Are the listeners getting it? Is the heat making them restless? Is the lighting too dim? Is that noise too distracting? In the act of telling, the teller must work with such inner and outer contingencies (and many more!) to bring the story and its listeners onto the same wavelength and into a unified realm of imagined experience." [Mar96, p. 153]

Altering a media presentation while it is in progress is possible in oral storytelling and other "live" forms by virtue of the fact that the media are being experienced at the same time they are being generated, and the person generating them is intelligent enough to sense certain response behaviors and use them to achieve the goals of his media more effectively.

By contrast, in traditional theater and dance, performers serve more as "mediators" of content that is almost always generated in advance by someone else, but they are still endowed with the ability to modify their delivery of that material to a certain degree to engage the audience more effectively. Unless the piece involves improvisation, performers are often not at liberty to change the lines they deliver or create new movements in the middle of a performance, but they can raise their voices, adjust their timing, or increase their energy level if, for example, the audience seems bored or unresponsive.

The invention of written media, such as books, posters, letters, and newspapers, created a more pronounced barrier between author and audience, most notably because of the delay introduced in the processing of response activity. Marshall McLuhan alludes to this phenomenon in his discussion of the effect of the written word on society:

> "Oral cultures act and react at the same time. Phonetic culture endows men with the means of repressing their feelings and emotions when engaged in action. To act without reacting, without involvement, is the peculiar advantage of the Western literate man." [McL94, p. 86]

In written media, the physical book or newspaper itself acts a mediator of the content, and a fairly unintelligent one at that. A book has no means of, say, ad-

justing the words and the sentences on the page if the reader appears to be bored or confused. Of course, it is not totally devoid of response behavior—readers can turn the corners of pages, write notes in the margins, or spill extra coffee in certain chapters, altering the presentation of the media for future readings in a way that will reflect a history of its use and the personalities of its previous readers.

However, for more complex forms of responsiveness, the reader must actively invest time and energy to, say, write a letter to the editor or publisher. If responses do eventually reach the author of a book or newspaper article, it is only long after the process of creating the media has been completed—after the book has been published, after the newspaper has been printed. These delayed reactions may affect how the author writes in the future, but it is too late to alter the original piece to which the response was addressed.

The arrival of electronic communication media, such as telegraphy, telephony, radio, and television, provided the capability for physically distant parties to experience near instantaneous responsiveness. In the case of telephony, for example, two or more people separated by a great distance can speak and respond to each other through an audio channel almost as they would in person, without any noticeable time delay.

Although commercial television and radio broadcasting systems often incorporate very indirect forms of audience responsiveness (Nielsen ratings, complaints, and so on), a great deal of their programming is mostly one-way in nature, consisting of prerecorded shows or songs transmitted in sequence or at particular times of day. A television receiver, like a book, is a fairly unintelligent mediator of the content it carries. Viewers can make simple adjustments to sound volume, picture contrast, color balance, and so on, but the television itself does not have the ability to, say, increase the size of text captions, cut in extra close-ups, or show a scene from a different angle if it senses the viewer is having trouble seeing something or if the screen is very small in size. Combining television or radio with another electronic medium acting as a "back channel" can enable a richer form of responsiveness—for example, viewers can steer the content of a live talk show by telephoning the studio and asking questions to the show's guests.

The emergence of modern electronic computational devices enabled human media creators to embed knowledge about how to handle response behaviors in computer programs that control the delivery of media presentations. This new capability, coupled with advances in sensing and media display technology, has paved the road for the development of several new and exciting forms of responsive media in the past several years.

One of the most popular of these new forms is certainly the computer game. Interactive competitive play systems have existed since the early 1960s, most notably a game called *Spacewar* that was developed on a DEC PDP-1 mainframe at MIT. However, Atari's original arcade version of *Pong* is probably the most well-known early computer video game. Pong simulated an interactive tennis match in which two players paddle a tiny ball back and forth across a video screen. The game, with its unbelievably primitive graphics by today's standards, was an overnight sensation when it was first displayed at a local bar in Sunnyvale, California in 1972 and sparked the development of a huge industry for arcade and home computer gaming systems that even today shows no signs of waning. [Coh84]

Advances in video and audio storage and retrieval technology led to some early experiments in interactive and personalized movies, perhaps most notably the Aspen Movie Map project from the MIT Architecture Machine Group in the early

1980s [Moh82]. Users of the movie map could take a virtual "drive" around the city of Aspen, with interactive control over where turns are made, speed, and direction of view. Random-access videodisks held imagery of the entire city of Aspen, and the use of multiple disk players, video mixers, and computer animation facilitated smooth transitions between the individual recorded elements. Similar videodisk technology was later utilized to create the Movie Manual, a prototype of an "electronic book" that incorporates video, audio, and hyper-linking capability along with standard text [Gan83]. Two instructional manuals were created with this framework, one for repairing automobile transmissions and the other for servicing bicycles. Special attention was given to the ability of the viewer to customize his learning process by navigating and annotating the material in different ways.

The potential for computer-mediated responsiveness also spawned new developments in interactive expression. Frank Popper traces the roots of computer-based interactive art back through other "technological" forms like video, laser, holographic, and lumino-kinetic art, and further to the Futurism, Dadaism, and Constructivism movements of the early 20th century [Pop93]. Margot Lovejoy further traces the origins of interactive art to the new emphasis placed on the role of the spectator with the advent of photographic art:

> "Photographic reproduction and the cinema raised social questions about the artist's role, about the audience for art, about art as communication rather than art as object, and thus brought into focus the social function of art." [Lov97, p. 5]

In the past several years, many new computer-based responsive art forms have emerged, from multilinear hypertext novels to large interactive installation works. Artists working with these new forms have created many fascinating pieces dealing with a wide variety of themes. To give a few examples, Michael Naimark explores the recreation of "place" in his *Be Here Now*, a large installation in which the spectator stands on a rotating platform and experiences three-dimensional audio-visual recordings of several "endangered places" in the world at different times of day [Nai97]. In Toni Dove's *Artificial Changelings*, participants explore the thoughts, memories, and dreams of two characters, one a Victorian-era kleptomaniac and the other an encryption hacker from the future, in a multimedia installation that responds to physical body movements and hand gestures [Dov97]. In his *Video Place* experiments, Myron Krueger creates responsive environments in which the digitized silhouettes of spectators are transformed into objects in a shared video space that can be manipulated through body movement and gesture [Kru96].

The Media Lab has a history of pushing the boundaries of what Ricky Leacock called "opera in the largest sense of the word" with such installation works as *Antenna Theater*, *Wheel of Life*, *Brain Opera*, *KidsRoom*, and many others. In *Wheel of Life*, participants become either explorers or guides who are separated from each other and must collaborate through a variety of computational interfaces to successfully travel through three large-scale physical story environments, one each representing the traditional elements of earth, air, and water [DF95]. *Brain Opera* presents its audience with an opportunity to play several new computationally-mediated musical instruments and have their contributions incorporated into a multimedia performance inspired by the writings of Marvin Minsky [Mac96]. In the *KidsRoom*, animated characters and voices respond to the movements and gestures of children in a story environment fitted with computer vision technology [Bob99].

In many ways, these large-scale installation forms represent the pinnacle of computer-mediated responsive media because of the way they attempt to engage all the senses through a variety of interfaces to achieve the same rich feeling of reciprocity found in oral storytelling, but many other interesting forms have emerged over the years, too many to discuss in full detail in this brief document. Some other examples are virtual reality environments, telepresence systems, training simulators, and interactive television shows. Hypertext and other hyperlink-style forms have become especially popular for many different kinds of content, from large commercial Web sites and computer-based training applications to personal Internet "home pages."

# Motivations

Why would we want to create responsive media in the first place? And why would we be motivated to use computational devices to control the delivery of that media? Bill Harley addresses the first of these concerns in the context of storytelling, underscoring the importance in any media endeavor of building a two-way relationship between author and audience:

> "Storytelling's success depends upon the intimacy created between teller and listener. When the teller is powerful and the story well told, there is often the feeling in the audience that the teller is "speaking only to me"—a palpable vulnerability is present. When storytelling fails, the audience member feels she is closed off from the story—that the performance would take place in the same exact manner whether the audience were there or not. As a result, the listener doesn't invest in the story, and little is shared." [Har96, p. 130]

Shannon and Weaver approach this issue from an information theoretic standpoint in their explication of three different problem levels in communication [SW63, p. 4], as follows:

- *The technical problem:* How accurately can the symbols of communication be transmitted?

- *The semantic problem:* How precisely do the transmitted symbols convey the desired meaning?

- *The effectiveness problem:* How effectively does the received meaning affect conduct in the desired way?

The discussion and mathematical theory presented in their book applies almost entirely to only the technical problem of communication. It is in the last two of these areas, the semantic and effectiveness problems, that designing responsive behaviors into media can likely be of the most help.

Automating the control of responsive media with computers enables applications in which it is impossible or improper for a human to perform the response tasks manually—for example, if human bias in responses would be undesirable, or if the responses would be too complicated, too repetitive, or just too boringly simple for a human to generate effectively. It also enables applications that require complex and instantaneous processing of the media signals and sensing devices, as is needed in flight simulators or virtual reality systems for example.

## Education

In the realm of education, the development of a one-on-one relationship in which the teacher or professor understands the history, interests, and capabilities of each student and can provide individualized attention is of central importance in the learning process. Yet, much educational media is very unidirectional in nature, consisting of no mechanism to adjust presentation based on these kinds of factors. Furthermore, no teacher can have a specialized knowledge of every subject a student might wish to learn about, nor can teachers or other experts be physically present at every time a student is likely to ask questions or be inclined to learn.

A tremendous opportunity exists to develop responsive educational media that can gather or infer information about the student through a variety of means and use it to maximize their interest and learning potential, in a manner much richer than the branching teaching machines of the 1960s [Per64] and their more contemporary variants. In his discussion of the Movie Map and its aim to make the student the "driver, not the passenger" in the learning process, Robert Mohl points out such systems are "akin to a ping pong match."

> "It's either the student's turn or the computer's turn. One is always waiting for the other (and by implication not necessarily maintaining attention). True interactivity requires either agent to be able to intervene at any time deemed opportune." [Moh82, p. 197]

Utilizing new sensing technologies and models for understanding human behavior, students could delve into subjects that interest them in great detail outside of the shared classroom environment in systems that can sense and respond to subtle cues such as when a student is confused, bored, or overwhelmed, and can grow to understand over time how a student learns most effectively.

## Commerce

Responsive media also has enormous commercial potential, perhaps most notably in the domain of advertising, where the opportunity exists to create ads and propaganda that have a sense of their audience and the context of their presentation. A television receiver might be programmed to not display an ad for a local restaurant if the viewer has already eaten dinner, or more interestingly, to show an extended version of the ad if it senses the viewer is hungry, with imagery of food selections that might agree with the viewer's taste. Responsive media also has potential in other marketing applications, from responsive store windows and displays to hyperlinked video product placement and home shopping, some prototypes of which will be discussed later.

## Expression

Computationally-mediated responsive media opens vast possibilities for creating new forms of artistic expression. Advances in sensing and display technologies enable situations in which the presence, movements, gestures, images, and other aspects of spectators can be transformed and reflected by a work of art in a way that contributes to the artist's theme or message. Key elements of effective storytelling, such as mystery and identification, can be heightened and even directly driven by the audience of an experience, as shown in several experiments from the Media Lab's Interactive Cinema group [Dav00]. Most stories consist of an element of journey—a feeling of being on a road to discovery where there are

obstacles to overcome and where changes occur. Endowing audience members with the power to control various aspects of how a journey unfolds or to embark on their own journey through a complex story world can increase the audience's motivation to experience the story, thereby enhancing its emotional power and memorability.

## Communication and awareness

Building response behaviors into media can also change and improve the way we communicate and collaborate with each other. A great potential exists for developing new kinds of interpersonal communication environments that respond to the activities of participants in a manner that enhances the effectiveness of their communication even beyond what is possible in physical face-to-face meetings [HS92]. Responsive telepresence applications can foster a sense of community or connectedness among their users even though they may be physically separated, on opposite sides of a room or on opposite sides of the globe. Such systems might serve a larger social or political role by providing responsive forums for debate and consensus-building, markets for trade, and other means for heightening awareness and collaboration between governmental or political leaders and the common person. Other systems might aim to reflect aspects of our own activities in a way that increases awareness of the long term effects of our habits and practices on our lives and on the health of our communities. As Marshall McLuhan points out,

> "Electrical speed in bringing all social and political functions together in a sudden implosion has heightened human awareness of responsibility to an intense degree." [McL94, p. 5]

The motivation for utilizing computational power in creating new kinds of responsive media is strong in many other domains as well, too many to be discussed in detail here. Many other areas of interest, including several prototype applications, will be discussed in later chapters of this dissertation.

# The design process

In a very basic sense, designing responsive media involves creating relationships between "inputs" and "outputs"—between the factors and events that the media will respond to and the actual media that are presented. But more than that, it involves forming and embedding a "philosophy of design" in a presentation such that it can modify itself appropriately based on the creator's vision. Determining exactly what kinds of inputs and outputs the media should incorporate and what the relationship between them should be is not a trivial process and often involves an experimental trial-and-error strategy for finding configurations that produce the desired result.

## Understanding the goals

Whatever the exact approach, the initial and perhaps most important step in creating any kind of responsive media is to take a step back from the project and develop a careful understanding of the goals of the message or experience. For example, perhaps the intent is sell cars, or to increase awareness about how air pollution affects health, or to create an system in which children can collaborate on making a movie. In addition to any high-level objectives, there may be sev-

eral secondary or auxiliary aims as well. Below is a list of some important issues to consider at this stage:

- What information are you trying to convey?

- What behavior are you trying produce?

- What product are you trying to sell?

- What skills are you trying to develop?

- What feelings are you trying to induce?

- What thoughts are you trying to provoke?

- What memories are you trying to evoke?

Given the results of this analysis, the designer can begin to investigate how certain kinds of responsiveness might contribute to or detract from the fulfillment of the goals, the effectiveness of the message, or the enhancement of the theme.

## Choosing the variables

Conceivably, a presentation might be made to respond to almost anything that can be observed or recorded in some way, but grasping the wide range of possibilities and why they are meaningful can be a difficult and confusing matter. Any kind of diagram that tries to express the vastness of this input space almost always falls short, but below is one possible visualization of the many axes available to traverse:
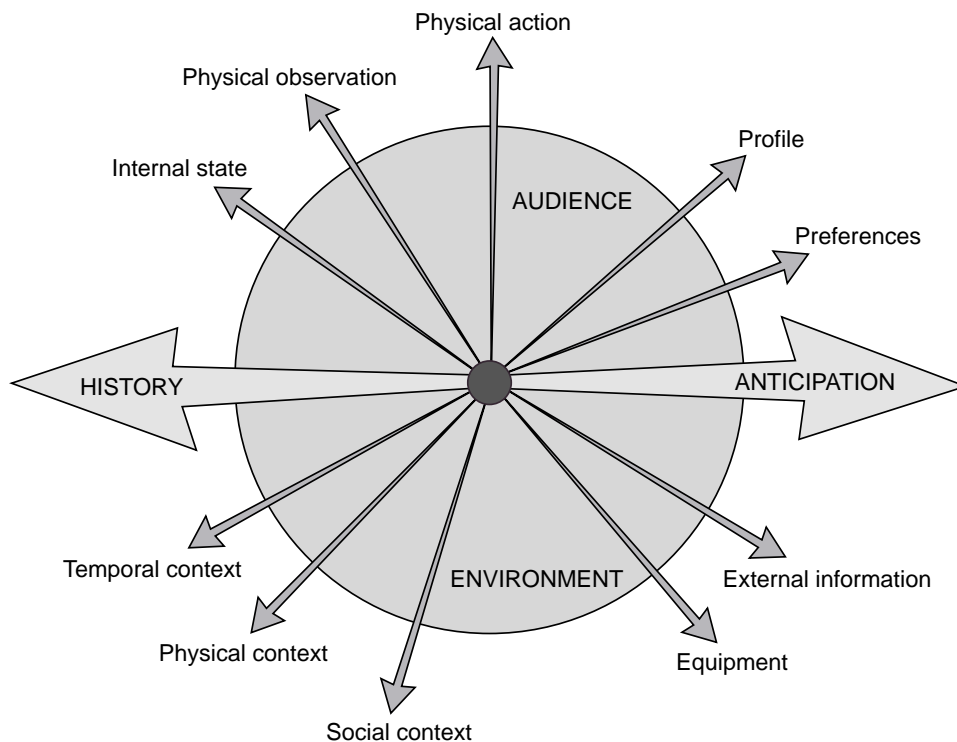
In this diagram, history and anticipation represent the "time" axis. The other axes are in no particular order, besides that the upper portion represents variables about the audience, and the bottom represents variables about the environment. Oral storytelling scores high on many of these scales, but that doesn't mean that all responsive media should aim to span all of these axes to their fullest. More important than knowing all of the possibilities available is understanding what each of these categories of responsiveness can offer to an experience, and thereby which are the most important to consider incorporating in a specific project.

- *Physical action:* key presses, mouse clicks, joystick maneuvers, hand gestures, body movements, other willed actions… Responsiveness of this straightforward nature is often the most useful where viewers or audience members are to be endowed with some direct control over how an experience unfolds—for example, to navigate among story elements or to directly act upon and change the state of a story world.

- *Physical observation:* facial expression, posture, laughter, crying, shirt color, number of audience members, spectator positions or motion… Taking notice and responding to passive or involuntary reactions of this sort can help establish a sense of reciprocity without the spectator or audience member having to actively initiate contact and thereby distract his mental immersion in an experience.

- *Internal state:* heart rate, breathing rate, body temperature, skin conductivity, stress level, sadness, confusion, fatigue, arousal, hunger, other emotions and mental states… Mammals are capable of reading each other's mental states through a variety of instinctual means, and we unconsciously use these cues to guide our interactions in a way that empathizes with the others involved [LAL00, p. 52]. Because empathy is an important step in establishing a feeling of trust with audience members, some forms of responsive media might benefit from the same close observation of bodily or mental state.

- *Profile:* name, age, height, weight, sex, race, home town, hair color, shoe size, nose size, hobbies, IQ, SAT scores, grade point average, stubbornness, sensitivity, restrictions, ailments, allergies, other physical or personal background information… Knowing your audience and tailoring the presentation accordingly shows you care about them, and they will in turn care more about what you have to say. Alternatively, adjusting material based on individual background can be used to create a greater sense of shared experience over an entire population.

- *Preferences:* favorite type of food, least-liked colors, favorite school subjects, favorite movie genre, dreaded clothing styles, best-liked traits in friends or love interests, other likes and dislikes… Taking into account preferences can help avoid situations that may put off or offend an audience member and can enhance the personal appeal of a message, making it more memorable.

- *Temporal context:* time of day, time of year, time zone, phase of the moon… People react differently at different times of the day, the week, and the year, and presentations can use this knowledge to match material to the prevailing sensibilities at different points in these temporal cycles.

- *Physical context:* location of presentation site, altitude, size and structure of site, physical arrangement of objects at the site, kind of furniture, lighting conditions, noise conditions, air currents… Oral storytellers and other per-

formers keep careful watch and adjust to lessen the impact of constraints and distractions in their physical presentation environments, and computational media can benefit by accounting for these factors as well.

- *Social context:* social function of the room or presentation site, type of activities in progress, dominant social norms or political attitudes, principal language spoken, local customs or restrictions… Taking into account these context factors and adjusting accordingly can help avoid uncomfortable or awkward elements that could drastically reduce the effectiveness of a message or, even worse, cause the audience to turn against the producer of the media.

- *Equipment configuration:* type of video projector, size of projection, number and type of audio channels available, processor clock speed, amount of memory, type of sensing devices in use, type of network connectivity, available bandwidth… Adjusting delivery based on these simple factors is an easy way to increase the effectiveness of a message—for example, reducing resolution to maintain the frame rate of a movie when less bandwidth or processing is available, or including more close-ups or tighter cropping when a video program is presented on a very small screen.

- *External information:* weather outside the presentation site, events in the news, air pollution index, unemployment rate, the current price of gasoline, the expiration date of the milk in your refrigerator, what tie David Letterman is wearing today… This category represents any other kind of external information, not fitting into one of the other categories, that a presentation may need to fulfill a particular goal.

- *History:* memory of physical actions and observations at previous points in time… Recording, learning from, and incorporating elements of a viewer's history of interaction with a system enables a richer sense of a sustained "conversation" and can increase a viewer's motivation to return to an experience at later points in time.

- *Anticipation:* predictions, through intelligent processing of previously gathered data, about what states and events may occur in the future… A sense of "lookahead" humanizes a presentation, making it surprising and memorable, and gives the impression that the system is listening to and learning from its audience and cares about improving their experience.

The designer must give some thought to how the mediator of the presentation will sense or otherwise obtain the selected response factors, considering tradeoffs between privacy, invasiveness, security, cost, accuracy, resolution, automation, and so on, with respect to the goals and constraints of the project.

## Building the mappings

Defining the method by which responses will map to alterations of the media is probably the most challenging aspect of designing responsive media. The general problem of creating automated processes for transforming "inputs" to "outputs" is a central focus in the realms of algorithm design and computer science, and much insight can be gained by examining some of the tools and techniques used in these domains for solving various kinds of problems.

In classic *rule-based systems*, knowledge about how to map input to output is embedded in a set of explicit rules or instructions. *State machines* combine sets of rules with the concepts of memory to form more complex mappings. *Case-based*

*reasoning* offers a more concrete methodology for representing knowledge and mappings that involves the notion of "reasoning from experience"—making decisions based not on abstract rules but rather on similarities to previously gathered *cases* [RS89]. This principle is a strong factor in systems for *programming by demonstration* in which the designer presents examples of input data and the corresponding proper output for those situations, and the computer generalizes from these examples to form a program that can handle other situations more intelligently [Cyp93][Lie01].

Neither case-based nor rule-based systems are always appropriate for all kinds of users and all kinds of projects, but these two complementary strategies serve as useful starting points for developing more domain-specific tools for controlling different kinds of response behaviors. Both styles will be explored in detail in later chapters of this document, in the context of the three tools developed in this research.

# Guiding observations

If responsiveness is to be automated or otherwise mediated by a computational device, the mapping must be described, through the use of one or more tools or languages, in a form the machine can interpret or reason from. To be maximally effective, the design of these tools must reflect the unique characteristics of responsive media and support the processes by which creators think about and work to build these media. Given an awareness of the wide variety of forms and the vast range of possibilities that exist, in addition to knowledge of how various kinds of responsive media have been designed in the past, many of these characteristics become more apparent and can lead to strategies for developing a new generation of tools as well as standards against which to evaluate those tools.

First of all, as discussed earlier, automating responsive media behaviors entails creating a programmatic relationship between inputs and outputs, a problem area that is well understood in the realm of algorithm and programming language design. Much thought has gone into understanding what constitutes a good programming environment. As Abelson and Sussman observe,

> "A powerful programming language is more than just a means for instructing a computer to perform tasks. The language also serves as a framework within which we organize our ideas about processes." [AS85, p. 4]

They continue by describing three mechanisms that powerful programming languages provide for describing and manipulating procedures and data:

- *primitive expressions*, which represent the simplest entities with which the language is concerned

- *means of combination*, by which compound expressions are built from simpler ones

- *means of abstraction*, by which compound expressions can be named and manipulated as units

These guidelines form a set of standards against which we can evaluate the basic expressive functionality and extensibility of programming tools and languages used for responsive media.

In addition, many responsive media projects have a distinctly multi-layered quality, requiring innovation at many different levels, from low-level concerns like driving a new sensor technology or configuring processors and communication channels in particular ways, to higher-level matters like describing the arrangement and behavior of visual objects in a video window or defining links in a hypertext document. Tools must support this multi-layered quality by providing interfaces and mechanisms that operate at different levels of abstraction yet complement and interoperate with each other. Designers must be able to shift between and harness the power of several layers simultaneously throughout the process of building an application.

For many, developing responsive media is a highly experimental process that involves incremental changes and improvements and a seemingly never-ending cycle of trial and error. Tools can foster this style of development and encourage experimentation in a number of ways—for example, by making the range of offered functionality more readily apparent and accessible, and by providing components that can be easily swapped for others without requiring extensive revisions. They should provide the ability to view results immediately as changes and additions are made incrementally, and they must be efficiently reconfigurable and extensible in order to support endeavors that push beyond the prepackaged limits of the tool.

Responsive media projects often span multiple disciplines and involve collaborations among many kinds of people, each of whom has their own strengths and weaknesses and who are all working together to realize a common goal. Design tools should reflect this reality by providing the means to elegantly and efficiently merge the contributions of several collaborators, each of whom may be working in a different layer of a tool. They should also supply a set of unchanging primitive constructs or other mechanisms that might better enable collaborators to comprehend and build upon each other's contributions. These observations also call for rethinking the mere physical design of modern computer workstations, which, considering the prevailing one-person form, can greatly hamper efforts for several people to work on a particular aspect simultaneously. Hiroshi Ishii's *Clearboard* [IKA94] is an excellent model of a telecollaboration system that allows more than one mind to participate in real-time in a particular activity, in this case drawing. This and other similar systems serve as good starting points for developing better tools for large collaborative ventures.

Finally, responsive media appeal to our senses, as do our ways of thinking about their design. In addition to handling a variety of interface protocols and real-time recognition and analysis engines, tools must be able to support the processing-intensive nature of images, video, audio, graphics, and the other complex data types typically found in responsive media. Where appropriate, authoring tools should emphasize the ability to display and manipulate media data in its native visual or aural form rather than through textual commands. The process of creating or demonstrating program elements and mappings should incorporate visual and gestural metaphors or other mechanisms that might better mirror the working processes of designers.

In his trilogy of books, Edward Tufte introduces techniques for presenting many different kinds of information, but the third of these books, *Visual Explanations*, is especially enlightening as a starting point for developing systems that represent, in his words, mechanism and motion, process and dynamics, causes and effects, explanation and narrative [Tuf97]. Nan Shu highlights several potential motivations for visual styles of programming in his book, where he asserts that pictures, when properly designed, can convey more meaning in a more concise unit of expression [Shu88, p. 8]. At the Media Lab, the Visible Language Workshop and more recently formed groups have focused on several aspects of visual informa-

tion design, including dynamic typography, methods for automating graphic design, and techniques for visually navigating large bodies of information. These and other projects, such as Marc Davis' *Media Streams* language for video annotation [DM95] and Henry Lieberman's programming-by-demonstration drawing editor *Mondrian* [Lie93], serve as important reference points for future endeavors.

Together, all of these observations and realizations about the characteristics of responsive media and the processes underlying their design can be summarized in three compact bullets—that responsive media consist of:

- Multi-sensory interfaces and materials

- Multi-person collaborations and experiences

- Multi-layered tasks and thinking processes

These three basic yet vital observations about responsive media form the basis of an approach for devising tools that meld more closely with the minds and bodies of their users and the character of their projects. This approach informs the design of the three tools developed in this research, described in Chapters 3 through 5, and these three bullet points will be revisited in those chapters to motivate the design of these tools and to evaluate the success of those designs.

# Survey of popular tools

There are several popular responsive media design tools worth examining more closely at this point, with respect to the three design guidelines stated above. Each of these tools offers a different range of functionality and a distinctive syntax for describing processes and behaviors. One of the most popular is Macromedia's *Director*, which provides a high-level visual interface coupled with a textual scripting language named Lingo. Its biggest weakness is that these two layers do not complement each other. Lingo overrides several assumptions made in the visual layer, forcing users to unlearn much of their expertise when it becomes necessary to use Lingo and causing the visual layer to become more of an encumbrance than an aid to the authoring process.

Another well-liked tool, especially in the music world, is Opcode's Systems *MAX*, which employs a flowchart-based graphical programming environment in which box-shaped functional elements are connected with "wires" to express various behaviors. However, it also suffers from many of the same problems faced by other flowchart environments in general. Typical projects require many more wires than higher-level conceptual process diagrams, and when building a very large program with complex object connections, the wires that go between objects can quickly overwhelm the screen and reduce readability, shifting the focus of activity away from programming and more toward graphic design.

IncWell's *Supercard*, Asymetrix's *Toolbook*, and Quark's *mTropolis* are similar to Director in that they feature both a visual interface and a scripting language. Pierian Spring's *Digital Chisel* and Intersystem Concepts' *Summit* authoring system offer changeable "user expertise" settings, from novice to advanced, that modify the authoring interface in order to ease the burden on beginners as they learn how to use the tool. Pictorius' *Prograph* is general-purpose visual programming language that resembles MAX with its flowchart-driven interface, except that connections in Prograph represent either data flow or synchronization rather than message-passing.

HTML, JavaScript, and Java form a multilevel suite of textual language tools for authoring hypertext content for delivery chiefly on the World Wide Web. The main issue working against this suite of languages is that there is no collective design strategy that underlies all of them. JavaScript looks like Java in terms of syntax, but completely different engines handle each language's interpretation, and manipulating Java objects from JavaScript, and vice versa, proves difficult. HTML is effectively isolated from both.

When other tools are insufficient for the demands of a particular application, writing a program in a general-purpose textual programming language like C is often the only option. C certainly offers unrivaled expressive power and fine control over all computational elements and I/O devices, but it also has tremendous weaknesses, not the least of which it was originally conceived 30 years ago not as a design environment for responsive media but rather as a platform for systems programming—developing operating systems, device drivers, and other heavy duty computing applications [Rit93]. Its syntax is complex and extremely fragile, and small errors can often result in entirely non-functioning code. Designers must concern themselves with things like type declarations, function prototypes, header files, pointers, and memory management in order to make successful use of the language, even though these details focus more on machine intricacies rather than the problems being solved.

Its object-oriented descendants (C++, Objective C, and so on) add an additional layer of complication onto a language that many dedicated computer scientists already find difficult to master. Java, which looks very much like C++ on the surface, also suffers from many of the same concerns, although the programmer is slightly more insulated from the internal complexities of the physical processing equipment. But Java's principle of platform independence can often be more of a obstacle than a blessing since many Java implementations and libraries vary slightly from machine to machine and version to version, and programmers end up having to tailor their code to work with multiple platforms or versions anyway, thus defeating the supposed purpose of using the language in the first place.

# Chapter 3
# Isis

## Why a new language?

As discussed in the previous chapter, because of the limitations of many of the most popular commercial new media tools, designers of experimental and cutting-edge responsive media are often left with no other option but to base their development efforts within general-purpose programming languages. This is really no surprise, since we have observed how, at its root, a central component of the design process involves building programmatic relationships between inputs and outputs, and the mechanisms provided by these languages afford the necessary expressive power and computational efficiency to handle the most complex kinds of mappings a media creator might imagine.

However, even as we cannot deny that the functional elements provided by languages like C and Java are extremely useful, we have also observed how these languages fail immeasurably as design environments for responsive media, due in large part to their complex and problematic syntax that traces its roots back to the days of early Unix systems programming, before it was even realized by most that the computer could be a valuable tool in the mediation and delivery of media content.

More recently conceived languages such as Lisp and Scheme possess many of the same semantic constructs as their C-like counterparts (functions, block structure, conditionals, loops), but a new ability to manipulate procedural expressions in the same way as other data made these the languages of choice in the artificial intelligence community, which found the concise and elegantly designed syntax more amenable to formulating intricate algorithms. On the other hand, these languages are substantially more divorced from the underlying hardware they operate on, and they have often been criticized as inefficient and impractical for serious applications programming. Strange names for common constructs and operators (`lambda`, `cons`, `car`, `cdr`...) can make them difficult to learn. And even though Scheme's facility for modifying the language's underlying syntax can be helpful in adapting it to specific preferences or problem domains, it can seriously compromise the ability of different programmers to understand and share each other's code.

## The Isis strategy

*Isis* is a new programming language that has been expressly crafted to serve as a basis for responsive media applications, and also as a foundation for more specialized media tools. It strives to support the unique characteristics of responsive media by reflecting the three main guiding principles discussed earlier in its inherent design.

## Multi-sensory interfaces and materials

Isis aims to support the performance requirements of the most complex real-time responsive media endeavors in a number of different ways. Isis is a complete language, in the same way that other general-purpose languages are complete, enabling an author to express arbitrarily complex relationships between response factors and presented media. As in Lisp and Scheme, procedures are first-class in Isis, meaning the programmer may create and manipulate them like any other data as a program executes, greatly extending the expressive capability of the language.

Conceptually situated much closer to the hardware it runs on, Isis is designed with a "lean and mean" attitude in mind, consisting of a small number of core constructs and primitive operations, internal structures based on native machine types and arrays rather than strings or linked lists, its own optimized memory manager, and no extra overhead for security and platform independence, resulting in a very small and fast interpreter compared to that of Scheme or Java. Isis provides a number of libraries that support a wide variety of input and output devices, as well as a way to efficiently interface to functionality written in other languages.

## Multi-person collaborations and experiences

Isis aims to be accessible to a much wider variety of people than typically use other programming languages. The minimalist syntax of Isis consists of a few basic types and constructs that cannot be changed or added to, thereby lessening the burden on those with less programming expertise and allowing collaborators to more readily understand at least the basic control flow of each other's contributions even if they are based in different layers of a software library. Yet while small and fixed, the syntax still provides all of the constructs seasoned hackers need to take full advantage of their skills. Isis is not object-oriented at its core, greatly reducing the language's complexity all around, although object-oriented styles of programming may be simulated easily and efficiently if desired.

## Multi-layered tasks and thinking processes

The software libraries developed for Isis strive to follow a multilevel design strategy, consisting of multiple interoperable layers, each of which offers a different level of abstraction of a particular kind of functionality and that together use the same language elements as a basis. In addition, Isis itself is an interpreted language, requiring no compilation phase, enabling the faster integration of incremental improvements and modifications throughout the design process.

The following sections delve into more detail on all of these points, beginning with a discussion of the syntax and internals of Isis, continuing with a description of the media functionality it offers along with several code examples and comparisons, and culminating with an overview of several projects built using Isis that demonstrate its viability and uniqueness as a platform to support responsive media.

# The syntax of Isis

The syntax of a programming language is not just a set of rules by which statements are formed. It is also a user interface that forms the basis for how authors

organize their thoughts about the problems they want to solve. The main goal of the syntax of Isis is to support a wider variety of users—to make the language accessible and easy to master by programming novices while at the same time supporting experienced hackers. It also aims to provide a concise and elegant means of formulating algorithms that minimizes extraneous details about machine intricacies.

Those familiar with the Scheme language will notice a few syntactical similarities in Isis, although the internal operation and interpretation of the two languages is quite different. The Scheme syntax was chosen as a model because the amount of text spent on handling incidental machine or object details in typical programs is qualitatively much lower compared to other languages like C and Java. However, the Isis syntax also differs greatly and addresses many of the difficulties of the Scheme syntax, as will become evident in the next section. Mainly, Isis is more of a minimalist language, consisting of a purposely small number of expressions, to reduce the number of things a novice has to learn to master the language, while still preserving the expressive power of a general-purpose language.

Isis is an *interpreted* language, which means it isn't necessary to "compile" programs prior to executing them. This feature helps to better support an incremental design process, since it is possible to add new instructions and change the behavior of a program while it is running.

An Isis programmer may either type expressions directly into the Isis interpreter on a command terminal, or he may write expressions in a file using a text editor and execute the interpreter on the contents of that file. Either way, the interpreter reads expressions one by one and "evaluates" them to produce a resulting "value," which is either printed on the screen for the programmer to see or suppressed if the interpreter is processing a file.

## Types

There are only seven kinds of data values in Isis, otherwise known as *types*. Below is a list of these types, along with some examples of *constant expressions* for each type, which is how a constant of each kind of value would be entered into or displayed by the interpreter.

- Integer          `42, -5, 0b0110`

- Real          `3.14, 6.02e-23, 3e8`

- Character          `'a', '\n'`

- Boolean          `True, False`

- Address          `0xF4E3D2`

- Procedure

- List

Integers are written as numbers without a decimal point, or as a binary numeral preceded by `0b`. Real number constants must be entered with a decimal point or using exponential notation. The range of possible values is dictated by the size of the native integer and floating point types of the computational hardware.

Characters are entered as a single letter between single quotes, although C-like control characters are also allowed. Boolean constants are expressed as either `True` or `False`, with a capital T and F. Addresses, expressed as a hexadecimal numeral preceded by `0x`, represent a location in the computer's memory, and they are almost never entered directly as constants but rather used most often internally to refer to raw data in memory.

Procedures represent an action that should be performed on a set of values that has yet to be determined (the arguments). They are created using the `proc` construct described later. Procedure values may refer to procedures written directly in Isis or external C functions, but there is no difference in the way they are manipulated or invoked.

The first six of these types are called *basic* types because they cannot be subdivided further. A *list* is the only *compound* type and is simply an ordered collection of values of any type, formed by enclosing a series of expressions in square brackets `[]`. Strings are represented as lists containing only characters, and they can be expressed by enclosing characters in double quotes as a short cut.

There is also a special value, entered as `Null`, that represents the absence or nonexistence of a value. `Null` is commonly returned from procedures when a return value doesn't make sense or if there is an error.

Expressions entered into Isis may span multiple lines—the interpreter evaluates the expression only when all opened brackets or parentheses have been closed. Any text entered following a # character on a single line is treated as a comment by the interpreter and not processed.

## Expressions

The rest of this section describes the core language expressions in Isis. In each definition, any *italicized* word may be replaced by any expression described here, including the constant expressions discussed above. Any elements that are optional are surrounded in curly brackets {}. Ellipses (…) indicate that an arbitrary number of elements following the established pattern is allowed. Any normal parentheses **()** or brackets **[]** and key words shown in **boldface** must be entered exactly as shown. Finally, `var` represents a variable name, which must be a sequence of characters that does not begin with a number, a period, or the characters + or −.

## Variables

**(set** var *exp***)**          variable binding

var                              variable reference

**(bound?** var**)**          variable query

Like other languages, Isis allows the programmer to store values in variables, except in Isis, to make things simpler, there is only one way to set a value into a variable, and only one way to reference the value of that variable. The variable binding expression sets the named variable to the result of evaluating *exp*. The same value becomes the result of the entire expression. Entering the name of the variable by itself returns the value referenced by it, or prints an error message and returns `Null` if none has been stored. The third expression allows a pro-

grammer to check if a variable has been bound to a value or not, returning either `True` or `False`.

## Lists

**[***exp  exp*  …**]**               list formation

**(***list  index***)**                 list reference

As discussed earlier, lists are created by enclosing several expressions in square brackets `[]`. The results of the expressions become the members of the list. The list reference expression retrieves a particular item in a list. *list* must evaluate to a list value and *index* must evaluate to an integer, with numbering starting at 0. `Null` is returned if the index is out of range. Because lists in Isis are internally stored in array-based data structures, a list reference is constant time O(1) operation, not requiring any traversal of a linked list to obtain the result.

## Native memory access

**(***addr  desc***)**                native memory retrieval

**(***addr  desc  val***)**           native memory storage

These expressions allow the programmer to access and manipulate data directly in the computer's memory, bypassing the Isis internal memory manager. They are most useful when dealing with large blocks of data, such as images or audio samples, or to format data for communication with a non-Isis endpoint. *addr* must evaluate to an address value, and *desc* must evaluate to a value that describes the native format of the data in memory. The result of the retrieval operator is a value representing the native data translated into the most appropriate Isis type. The result of the storage operator is simply the same value passed to it. The Isis web site documentation contains more details on these expressions.

## Conditionals

**(if** *exp  then*  {*else*}**)**                              2-way conditional

**(cond  (***exp  val***)  (***exp  val***)  …)**            N-way conditional

**(while** *exp  body***)**                                   conditional loop

**(switch** *exp*  **(***case  val***)  (***case  val***)**  …  {**(else** *val***)**}**)**

                                                            match conditional

These four expressions provide different kinds of conditional evaluation. The first is a standard `if` statement, common in many languages. If *exp* evaluates to a logically true value, the result of the *then* expression is returned, otherwise the result of the *else* expression is returned, or `Null` if it is not present. Isis considers a value to be logically true if it is a non-zero number, character, or address, or if the value is a procedure or a non-empty list, or if it is the boolean `True`. Any other value, including `Null`, is considered logically false.

The `cond` expression is simply an N-way version of an `if` statement in which each *exp* is evaluated in order until one results in a logically true value, and then

the result of its corresponding *val* expression is returned, or `Null` if no condition was true. The `while` expression repeatedly evaluates the *body* expression as long as *exp* evaluates to a logically true value, and returns the value of the body the last time it was evaluated, or `Null` if it was never evaluated. Finally, in a `switch` statement, *exp* is evaluated, and its value is compared in sequence to the results of the *case* expressions until a match is found, and the result of the corresponding *val* expression is returned. If no match is found, and a final default case isn't specified with the keyword `else`, `Null` is returned.

## Logical operators

**(and** *exp exp* …**)**                    logical and

**(or** *exp exp* …**)**                    logical or

These expressions are the logical `and` and `or` operators. They return either `True` or `False`, and only the necessary number of expressions are evaluated from left to right to determine the result with certainty.

## Sequential evaluation

**(begin** *exp exp* …**)**                    sequential evaluation

The `begin` statement allows several expressions to be evaluated in the place of one. The expressions are evaluated in order, and the value of the final expression becomes the result of the entire `begin` statement.

## Local environments

**(local (**var var …**)** *body***)**                    local environment

The `local` expression allows the programmer to specify a local environment, which is a collection of variables that will be accessible only from within the body of the local statement. Local environments are useful for specifying variables that are only needed temporarily, or for creating procedural "objects" that have their own private storage space. The interpreter gives an initial value of `Null` to each variable prior to processing the *body* expression, the value of which becomes the value of the entire `local` statement. Since the local variable names may duplicate those in the external environment, variable references in the body are resolved by first looking in the local environment, and if not found, the parent environment is consulted.

## Procedures

**(proc (**var var …**)** *body***)**                    procedure definition, fixed # args

**(proc** var *body***)**                    procedure definition, variable # args

(*proc exp exp* …)                    procedure application

The programmer may define procedures in one of two ways—by giving specific variable names for a fixed number of arguments, or by giving a single variable name which will hold a list of arguments passed to the procedure, which may vary in length. When a procedure is created, it remembers the environment

within which it was defined, and when it is invoked, the interpreter creates a new local environment on top of this environment in which the procedure's argument variable names are bound to the values passed to it. The *body* expression is evaluated in the context of this new environment, with variable references being resolved in a similar fashion to that of the `local` statement.

The result of a procedure definition is a procedure value. Procedures are *first-class* in Isis, which means they can be created within, passed to, and returned from other procedures as a program executes, just like the other basic data types. The user invokes a procedure using the procedure application construct, in which the *proc* expression must evaluate to a procedure value. The interpreter applies the procedure to the values of the remaining expressions, and the result is returned.

A few additional details about the operation of these constructs has been omitted from this document for the sake of brevity, but more information is available in the Isis web site documentation, which will be described later.

# Memory management

The main goal of the internal design of Isis is to provide the level of performance required to support the most demanding and complex real-time responsive media projects. One of the most unique features of Isis that differentiates it from other similar languages is how internal memory structures are handled by the interpreter in an effort to provide this support.

Relatively few kinds of data structures are needed during the execution of a program, although quite many of these structures may be required, and speed of execution depends to a large degree on how quickly these structures can be allocated and deallocated. Isis benefits greatly by employing its own memory manager that bypasses the standard C or C++ dynamic memory allocation mechanisms, which often become seriously inefficient in this kind of high turnover scenario. Additional speed gains are obtained by handling lists in array-based data constructs, and by storing values using native machine types (not as strings, as in languages like TCL and Perl), thus virtually eliminating any cost of data translation when invoking native routines from within Isis.

The Isis memory manager provides O(1) allocation and deallocation of all interpreter data structures and oversees a reference count system that tracks how many references there are to every structure instance dealt by it. When an instance's count drops to zero, it is deallocated and placed back in a pool of available instances. The manager is able to achieve O(1) efficiency by pre-allocating large blocks of memory to be used for particular kinds of structures, and by maintaining a database of pointers to available and outstanding instances such that obtaining a new instance or reclaiming an old one does not require traversing any lists. If space runs out in a pre-allocated block, another large block is added to the pool. This slight extra cost in memory usage results in a significant improvement in execution speed.

An important issue in language design is the system by which memory structures are reclaimed and reused when no longer needed or referenced. Languages like C provide no such system, requiring the programmer to manually keep track of outstanding memory and deallocate each block before the last reference to it is broken. As in Isis, languages like Java and Scheme employ a more sophisticated reference count mechanism that automatically deallocates structures when no longer needed. However, the underlying syntax of these other

languages is such that circularities will develop—where one or more structures refer to each other, keeping their reference counts above zero, but cannot actually be reached by any other structure in the active interpreter environment. If left unchecked, these circularities will accumulate and overwhelm the system's memory resources.

To avoid this situation, Java and Scheme have a "garbage collector" that periodically scans and marks every structure that can be reached from the top-level interpreter environment and deallocates any piece of memory that was not traversed. Such a system eliminates the programmer from having to do any memory management on his own, but it can also cause substantial interruptions and "hiccups" in program execution—a highly undesirable behavior in real-time media applications. Furthermore, because these languages allow circularities to arise without the knowledge or concern of the programmer, they can encourage styles of programming that result in extremely poor performance.

Isis avoids these problems because, unlike Java and Scheme, its core syntax does not consist of operators that modify internal value structures after they are dealt by the memory manager. Without these operators, known as mutators (such as `set!`, `set-car!` and `set-cdr!` in Scheme, and most assignment operators in Java), it is impossible to create a circularity among basic value structures, and therefore, a garbage collection phase is not needed. The language is no less powerful without mutators, which are merely programming conveniences that can become serious liabilities when overused. Thus, Isis is able achieve a level of performance suitable for demanding real-time media while remaining friendly to developers who do not want to be overly concerned about memory management.

There are other types of circularities that Isis permits to arise when procedures are defined within local environments, but these are of a much less serious nature and almost never result in memory overruns in typical programs. An automated collection scheme for these kinds of circularities could be developed, but they may be dealt with much more efficiently by adhering to a few simple programming style guidelines.

# Software libraries

The previous sections presented the basic syntax of Isis, which is quite general-purpose in nature. The full Isis development environment also consists of several "libraries" of software that provide many specialized kinds of media manipulation functionality as well as other basic utilities expected of good programming languages. Many of these libraries are designed with multiple layers of abstraction to support different levels of authoring complexity. The goal in each case is for these layers to interoperate with each other in the most seamless way possible, allowing the programmer to more easily shift between layers and use more than one at the same time to accomplish a task.

Many of these libraries have evolved to a point of stability, although each is still open to changes and improvement. For the sake of brevity, this section will only survey the various offerings, highlighting a few particular innovations.

## Basic utilities

Isis provides a full complement of mathematical and comparison operators, as well as utilities for manipulating lists and text in various ways. It also provides

several debugging facilities, including a system to track the values of specific variables and to trace the execution of particular procedures. There are also a number of utilities for accessing and working directly with system memory, by-passing the Isis memory manager, that are useful for handling raw image and audio data or for communicating with foreign protocols.

## Visual media

At the lowest level, Isis provides a library of optimized image processing primitives, such as composition, scaling, gain/bias, arithmetic, statistics, and so on. There are also some more complex operators available such as chromakey, noise removal, edge and feature detection, color balancing, and background segmentation. In addition, Isis has access to all of the routines in the OpenGL graphics rendering library.

At the mid-level, Isis provides an image buffer protocol so that all of the lower-level functionality described above may interface more cleanly and elegantly. This protocol handles the allocation and deallocation of memory space for images and simplifies isolating specific parts or channels of images for an operation. Within this protocol, all of the information about an image (size, format, memory addresses, and so on) is stored together in a self-contained list object.

At the highest level of image manipulation are Amazon and Macaroni, two different packages of scripted objects for handling complex compositions of images and video. Both these systems provide satisfactory support for experiments in the use of model-based representations as a basis for digital media [Bov95]. The two libraries provide similar functionality, the only difference being that Amazon employs in-house image processing routines whereas Macaroni uses OpenGL.

In both of these packages, the programmer creates a hierarchy of visual media "actors" placed in a three-dimensional space along with a "camera" that represents the viewer. These actors may be images, movies, video capture sources, text boxes, shapes, lines, and so on. Various transformations may be applied to a single actor or collection of actors to produce special effects. The view attained by the virtual camera is rendered on an output device. Each actor or transformation object contains parameters that the program can update between each rendering cycle to create animation. In the spirit of multi-layered functionality, these libraries aim to separate the programmer as much as possible from the internal details of how the media for each actor is obtained and rendered, but they also provide hooks to access actor "images" that can be processed with the lower-level routines.

## Aural media

Isis includes a similar multi-level set of operations for handling audio. There are several basic low-level audio processing routines available, as well as higher level "audio player" objects that insulate the programmer from the details of how that audio is handled on different platforms and with different audio interfaces. These objects enable the playback of several differently-filtered streams of audio simultaneously and the exact synchronization of that audio to other system events.

# Databases

The simplest way to create databases is with Isis lists, which internally are based on arrays rather than linked lists for speed in access. There are also a number of higher-level storage constructs available that use lists as their foundation—for example, the Isis *structure* construct (not to be confused with a C structure) is useful for storing arbitrary collections of key-value pairs.

Another useful construct, called the *timeline*, might be thought of as an array with real-number indices. Values of any type may be placed at real-numbered points along a one-dimensional "line." The programmer also specifies if and how the timeline should interpolate between each successive pair of points. Once created, a timeline may be queried at arbitrary real-numbered points, with a value being returned that depends on what type of interpolation was requested—none, linear, cubic, or a custom routine.

Timelines are valuable for multimedia purposes since they are helpful for coordinating time-dependent activities and expressing multi-dimensional spatial variations (or variations in any other analogous domain). Timelines can aid in creating complex animations by controlling the parameters of Amazon or Macaroni actors. Dynamic behaviors are possible since points on a timeline may be added, changed, or removed at will during execution.

# Networks and distributed processing

Many responsive media experiments require the power of several processes running on several machines simultaneously, all synchronized with each other and communicating in the most seamless manner possible. Isis has TCP/IP networking capabilities to support distributed processing. To ensure a high degree of interoperability, the same input and output primitives support reading and writing coded Isis values or raw data to disk files, network connections, serial ports, message pipes, and so on. Programmers may synchronize timers running in separate Isis processes or on multiple machines as well.

The low level I/O primitives, coupled with the memory manipulation operators, enable Isis to communicate with foreign network protocols with relative ease. High-level procedures that send email and retrieve documents and images directly from the Internet are among the most commonly used in many projects. Amazon and Macaroni both allow the programmer to reference images and movies by URL. Isis can also read and evaluate language expressions directly from any I/O port, allowing a single interpreter process to serve several concurrent command pipelines.

# Sensors and media I/O devices

Since the primary platform for Isis development thus far has been Unix-based workstations, Isis currently includes support for both the X window system and the Simple DirectMedia Layer (SDL) for video output, as well as certain raw video display cards. The goal has been to provide the simplest and most efficient environment for displaying images and video and receiving events, in order to enable typical kinds of window-based applications. Each of these output mechanisms has a separate interface that may be accessed either directly, for maximum efficiency, or through the use of a higher-level "video display" library that provides the same services in a platform non-specific form.

A "video capture" library is also available, which returns data in the image buffer protocol format described above. The programmer may alternatively choose to use lower-level direct interfaces to Video for Linux Two (V4L2) or the Compaq Tru64 MME video services. The situation is similar for audio, where Isis currently supports the Compaq Tru64 MME audio services, ALSA (Advanced Linux Sound Architecture), and the SGI Irix digital audio library.

Like with network protocols, a programmer can harness the Isis I/O primitives and memory manipulation operators to communicate with a variety of sensing devices, typically connected on a serial port of the computer. A number of useful libraries have been developed over the past several years for controlling many kinds of devices, such as MIDI synthesizers, lighting controllers, joysticks, video tape recorders, sonar and radar sensors, motion detectors, floor switches, electric field sensors, infrared transceivers, and eye tracking systems.

# A few examples

To illustrate the use of the language in a few different situations, this section presents a number of examples of Isis code, beginning with a simple one-line program that prints "Hello world!" on the terminal.

```
(print "Hello, world!" newline)
```

For comparison, below is the same "hello world" program written in C.

```
#include <stdio.h>

main()
{
  printf("Hello, world!\n");
}
```

The following is a recursive factorial function written in Isis.

```
(set fact
  (proc (x)
    (if (< x 2) 1 (* x (fact (- x 1))))))
```

The indentation of each line is not required and will not change the meaning of the program—it is merely a visual aid that helps a programmer keep track of the placement of nested expressions. Most text editors, such as Emacs, provide utilities that automatically indent each line of a program to the appropriate level based on the locations of unclosed parentheses and brackets. But the same program could be written on a single line, or on more than three lines with no change in meaning.

```
(set fact (proc (x) (if (< x 2) 1 (* x (fact (- x 1))))))

(set fact
  (proc (x)
    (if (< x 2)
      1
      (* x (fact (- x 1))))))
```

Here is a procedure that computes the distance between two points. Notice the use of a local environment to declare two temporary variables.

```
(set distance
  (proc (p1 p2)
    (local (dx dy)
      (begin
        (set dx (- (p1 0) (p2 0)))
        (set dy (- (p1 1) (p2 1)))
        (sqrt (+ (* dx dx) (* dy dy)))
        ))))
```

However, these temporary variables are not strictly needed, as the entire procedure could be collapsed into a single slightly longer expression that makes use of the map and apply primitives.

```
(set distance
  (proc (p1 p2)
    (sqrt (apply + (map (proc (x) (pow x 2))
                        (- p1 p2))))))
```

map is a useful operator that applies a procedure several times, once to each item in a list, and returns a list of the results. apply does the converse—it invokes a given procedure a single time on all of the items in a list together.

Here is a procedure that utilizes the TCP networking routines to communicate with an SMTP server and send a piece of email. It accepts the sender and receiver email addresses and the message text as arguments.

```
(set sendmail
  (proc (sender receiver message)
    (local (tcp)
      (if (!= Null (set tcp (tcp-open "outgoing.media.mit.edu" 25)))
          (begin
            (write-string tcp "mail from: " sender newline)
            (write-string tcp "rcpt to: " receiver newline)
            (write-string tcp "data" newline)
            (write-string tcp message newline "." newline)
            (write-string tcp "quit" newline)
            (close tcp))
          (print "Could not send that mail." newline)))))
```

The following uses high level library routines to retrieve and display an image in a window.

```
(set image (retrieve-url-image "http://www.bisquit.com/nana.jpg"))
(set win (easywin-create image "Nana"))
```

This program uses a few of the mid-level image processing routines to draw a colored cross shape on a red background. isolate-sub-image creates a separate reference to a rectangular portion of a larger image, while isolate-channels creates a reference to particular color channels of an image as specified. image-fill-constant fills an entire specified image with a specified pixel value.

```
(set image (new-standard-image 3 [400 400]))
(image-fill-constant 255 image)

(image-fill-constant 255 (isolate-channel 0 image))
(image-fill-constant 0 (isolate-channels 1 2 image))

(set vertpart (isolate-sub-image [150 50] [100 300] image))
(set horizpart (isolate-sub-image [50 150] [300 100] image))

(image-fill-constant 255 (isolate-channel 1 vertpart))
(image-fill-constant 255 (isolate-channel 2 horizpart))

(easywin-create image)

(read-string)
```

This example makes use of the video capture library to display live video in a window. The last line sets up an infinite loop to repeatedly capture frames and send them to the display.

```
(set framesize [320 240])
(set vc (video-capture-init [vc-framesize framesize]
                            [vc-streaming True]))
(set win (win-create [win-size framesize]))
(while True (win [win-put (vc [vc-read-frame])] [win-output]))
```

Below is a simple example that utilizes the Macaroni object-based media library to create a rotating graphic. Macaroni represents media actors and transforms as procedural objects with internal parameters that the programmer changes by invoking the procedure with arguments in a special message format.

```
(load "macaroni.isis")

(macaroni-initialize "Rotating image")
(set macwin (macaroni-create-window "Nana" [300 300] [100 100] True))

(set iobj (mac-new-image [mac-url "http://www.bisquit.com/nana.jpg"]))

(set itrans (mac-new-transform [mac-object iobj]
                               [mac-position [150.0 150.0]]
                               [mac-scale [0.5 0.5]]))

(set rot 0.0)

(macaroni-start
  (proc ()
    (begin
      (set rot (mod (+ rot 1.0) 360.0))
      (itrans [mac-rotation rot])
      (macaroni-update-window macwin itrans True))))
```
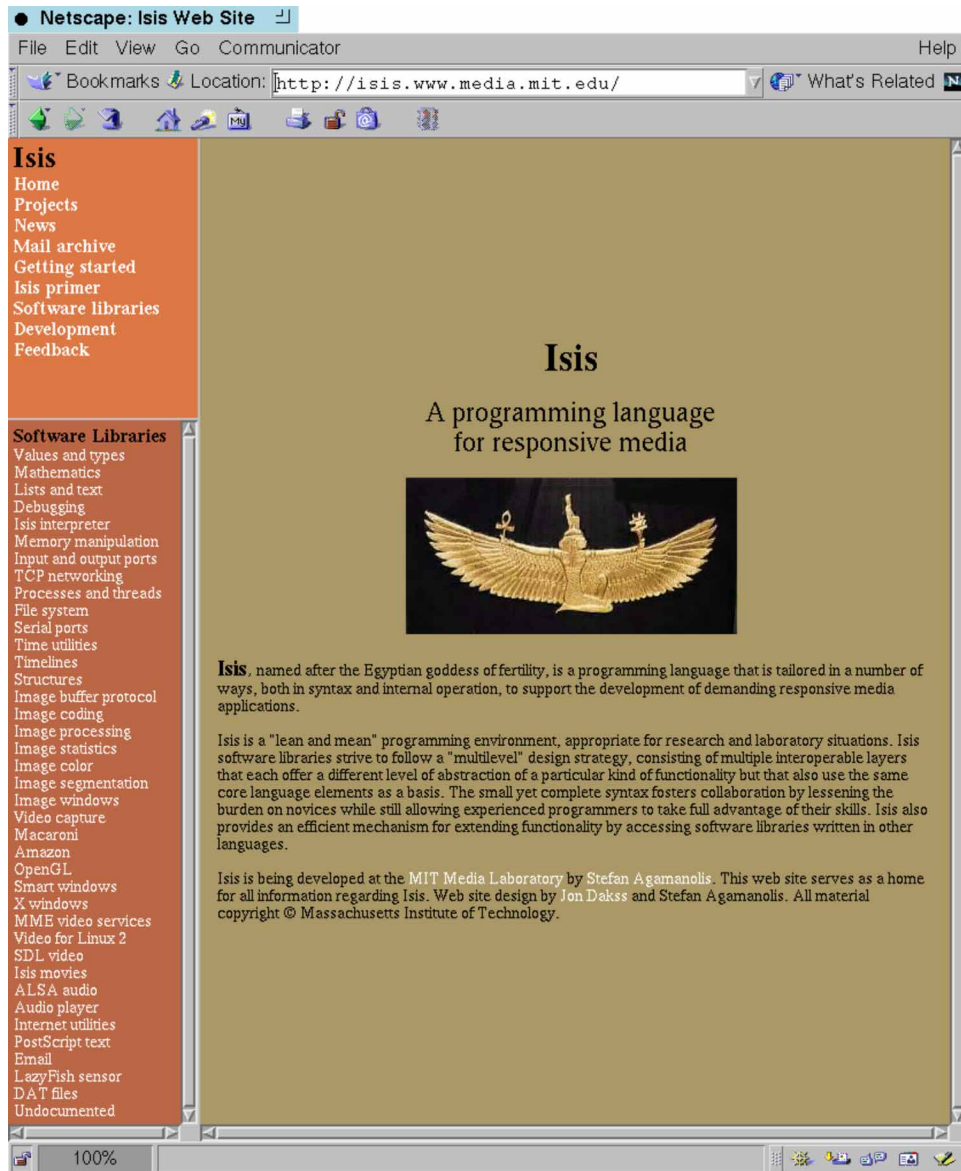
# User community

A language needs much more than a syntax and a collection of software libraries to be successful. The Isis web site (**http://www.media.mit.edu/isis**) has been a central component of the complete Isis development environment since its inception, and has been crafted over several years to provide concise yet accessible documentation of every aspect of the language. The web site includes a "primer" that helps new users learn Isis as quickly as possible, as well as program examples and information about all the media manipulation functionality.

But more than a simple storehouse of documentation, the Isis web site aims to provide a support structure for a growing community of Isis programmers. It includes links to projects built with Isis, a mailing list archive, and information on how individual users can add their own Isis functions and libraries to a central repository available to all others in the community. The web site seeks to foster a cooperative development attitude in which everyone using the language is encouraged to add new capabilities or contribute to the Isis environment in a way that sustains the "lean and mean" minimalist spirit with which it was created.

44

● Netscape: Isis Web Site

File  Edit  View  Go  Communicator                                             Help

Bookmarks  Location: http://isis.www.media.mit.edu/                    What's Related

**Isis**
Home
Projects
News
Mail archive
Getting started
Isis primer
Software libraries
Development
Feedback

**Software Libraries**
Values and types
Mathematics
Lists and text
Debugging
Isis interpreter
Memory manipulation
Input and output ports
TCP networking
Processes and threads
File system
Serial ports
Time utilities
Timelines
Structures
Image buffer protocol
Image coding
Image processing
Image statistics
Image color
Image segmentation
Image windows
Video capture
Macaroni
Amazon
OpenGL
Smart windows
X windows
MME video services
Video for Linux 2
SDL video
Isis movies
ALSA audio
Audio player
Internet utilities
PostScript text
Email
LazyFish sensor
DAT files
Undocumented

**Isis**

A programming language
for responsive media

**Isis**, named after the Egyptian goddess of fertility, is a programming language that is tailored in a number of ways, both in syntax and internal operation, to support the development of demanding responsive media applications.

Isis is a "lean and mean" programming environment, appropriate for research and laboratory situations. Isis software libraries strive to follow a "multilevel" design strategy, consisting of multiple interoperable layers that each offer a different level of abstraction of a particular kind of functionality but that also use the same core language elements as a basis. The small yet complete syntax fosters collaboration by lessening the burden on novices while still allowing experienced programmers to take full advantage of their skills. Isis also provides an efficient mechanism for extending functionality by accessing software libraries written in other languages.

Isis is being developed at the MIT Media Laboratory by Stefan Agamanolis. This web site serves as a home for all information regarding Isis. Web site design by Jon Dakss and Stefan Agamanolis. All material copyright © Massachusetts Institute of Technology.

100%

*The Isis web site*

# Evolution

Isis has been in development for more than five years, and over that time its face has changed greatly. In the beginning, it was meant to be replacement for an overly restrictive "scripting language" developed by Brett Granger and John Watlington for creating object-based media presentations on the Cheops image processing system [BW95]. In the course of its conception however, it became clear that any true basis for responsive media would have to include the capabilities of a full programming language, in order to support the widest possible range of mappings that a creator might imagine.

The earliest version of the Isis interpreter was implemented on Cheops, but the core system was soon ported to standard C, which enabled it to operate on a variety of standard Unix platforms. Isis also at first employed a much more struc-

tured type system, requiring programmers to explicitly declare the types of values and define new compound types prior to their use. But these and many other constructs and restrictions have been gradually dropped from the language, in the spirit of creating a minimalist language with the lowest possible interpretation overhead. Although it has reached a point of stability over the past year or two, further refinement is not out of the question.

# Experience

Over the years of its development, Isis has been used by many in the Media Lab community, including several sponsors, in the process of making new prototypes and demonstrations, and these experiences perhaps serve as the best indication of the viability of Isis, not just as a useful tool for creating responsive media, but as a platform for thinking differently about these kinds of media as well.

The knowledge and experience gained through creating these projects is in many ways the basis of the three guiding observations of this research pointed out earlier—that responsive media consist of **multi-sensory interfaces and materials**, **multi-person collaborations and experiences**, and **multi-layered tasks and thinking processes**. The following sections describe several of these projects in more detail, collected into three main categories: object-based media, interactive cinema, and telepresence.

# Object-Based Media

Isis has been used in several interactive and digital television experiments over the past 6 years. Many of these prototypes involved complex object-based representations of video and consequently put Isis to the test in its ability to handle this computationally-demanding type of media in real-time situations. Three such projects are described below.

## The Museum

David Tamés, David Kung, Stefan Agamanolis, Shawn Becker, Brett Granger,
V. Michael Bove, Jr., Glorianna Davenport

One of the earliest experiments produced in Isis, *The Museum* explored ways of better capturing the director's intent in video programs that will be viewed across a broad range of display sizes and aspect ratios. The goal here was to maintain recognizable actor expressions across a broad range of image sizes while also preserving a sense of the broader setting. Rather than simply scale or crop the image differently, the presentation incorporated an object-based representation of video that enabled changing the distance and focal length of a virtual camera as a function of the display size. The production ran on the Cheops computer and made extensive use of Isis Amazon objects and timelines to describe the variations in the camera and scene elements under the different viewing conditions.

Below are three images from *The Museum* that illustrate the difference in shot composition based on screen size at one moment during the movie:



## The Yellow Wallpaper

David Tamés, Stefan Agamanolis, Shawn Becker, Araz Inguilizian,
V. Michael Bove, Jr., Glorianna Davenport

*The Yellow Wallpaper* was a more ambitious interactive television production that enabled the viewer to explore a story from different camera angles as well as from different subjective points of view. The ultimate goal was not to take control away from the director, but instead to give the director more power over how a story is told in different circumstances and what freedom the viewer is afforded to explore that story in different ways.

Based on the short story by Charlotte Perkins Gilman, the program presents a "knob" that the viewer can turn to shift between the subjective points of view of the two characters, John and Kathy, even while it is in progress. The system renders changes in the point of view by altering aspects of the narration, such as editing, shot composition, background imagery, music, and sound effects, of an underlying story that remains constant. These images show a John-biased, neutral, and Kathy-biased composition at one point in the story:

The program also allows the viewer to take a virtual "walk" along a constrained path in the setting at the same time the action is proceeding. This is enabled by representing the scene as a three-dimensional computer model, and by having shot the actors separately from several camera angles on a blue-screen stage such that the most appropriate views can be layered into the model during playout. Here are four of the many spatial views possible at one moment in the playout:
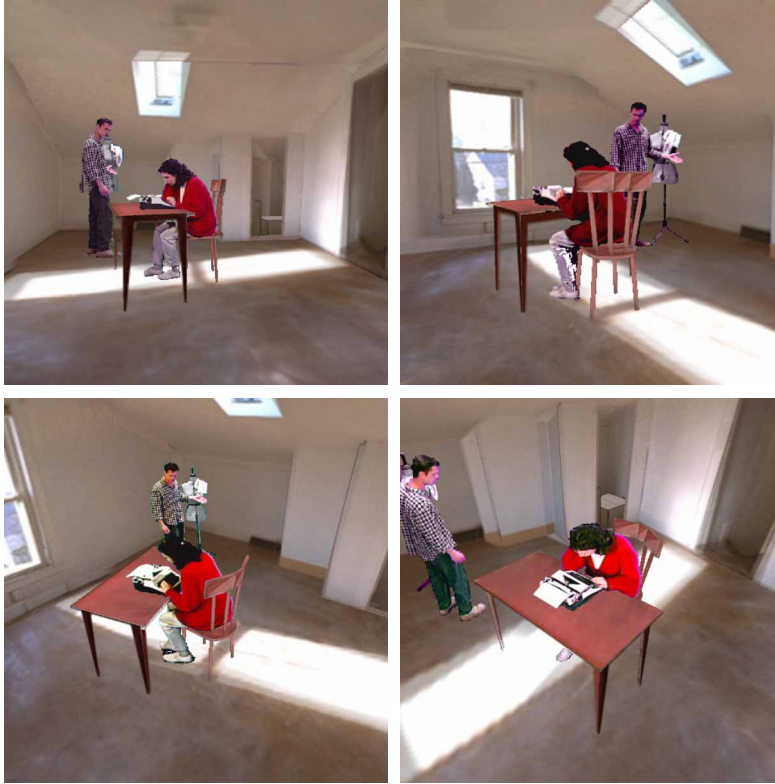


As in *The Museum*, Isis timelines and Amazon objects were invaluable in describing the scene elements and variations in the playout in a way that enabled a smooth interpolation between the two characters subjective viewpoints. Isis's multi-layered authoring characteristics were put to the test, as this was a collaboration between several researchers of different expertise, some working on low-level audio and image processing and others developing the scene structure and behaviors. Isis provided a common design environment for everyone involved and enabled the latest research from several groups to be combined into a single piece.


## HyperSoap

Jon Dakss, Stefan Agamanolis, Edmond Chalom, V. Michael Bove, Jr.

*HyperSoap* is a four-minute interactive television soap opera in which the viewer can click objects with a pointing device and find out how to buy them [Bov00]. Nearly everything that appears in the program is clickable, from the clothes and jewelry on the actors to the furniture and props in the background. In one mode, clicking an object displays a box with purchasing information while the program continues, and in another scenario, the program pauses at a natural point after an object is selected, allowing more detailed information to be presented, and then resumes in a fashion that maintains the flow of the story. The project, developed in collaboration with JCPenney, utilized a new system developed in the Object

Based Media group at the Media Lab for detecting and tracking objects in video using multiple feature attributes, such as color, texture, and motion [CB96].



*HyperSoap*

Isis was the development platform for not only the playback software but also the authoring tool that allows the video producer to digitize video clips, input training data needed by the object tracking algorithm, and correlate objects between different shots in the movie. The playback program uses the Isis Amazon library to combine the movie image with object highlights and information boxes. The language also supported drivers for a number of interaction devices, including a trackball, a touchscreen, and a system for detecting a laser pointer on a projection screen.

Another more recent production, *An Interactive Dinner at Julia's*, is a cooking show that allows a viewer to click on food and utensils and see video clips with more information about the selected objects. Created by Jon Dakss, the program features clips from old Julia Child cooking shows from the WGBH archive.



*An Interactive Dinner at Julia's*

# Interactive Cinema

Isis has also been used extensively in several story installation experiments, primarily developed in Glorianna Davenport's Interactive Cinema group at the Media Lab. An interesting feature of all of these projects is that they rely heavily on the ability to instantly and randomly access and edit together video clips and objects in a way that's driven by the audience or spectator in each case, in addition to some very complex sensing and gesture recognition systems. These projects also illustrate another key observation about responsive media—that it involves many different kinds of people, not just computer scientists, often working together in large collaborations. The five endeavors described below served as excellent experiments testing the accessibility of Isis in these respects.

## Sashay/Sleep Depraved

Freedom Baird, Glorianna Davenport

*Sashay/Sleep Depraved*, an installation conceived by Freedom Baird, lets a participant interact with a female video character, the Sleeper, through gesture [Bai97]. Each gesture causes animated sprites to appear on a wall-sized screen facing the viewer. As more gestures are performed, the animations accrue and play simultaneously. Depending on the number and type of gestures made, the multilayered animated "dream" constructed by the viewer affects the wakefulness and agitation of the Sleeper in different ways. The participant can build a peaceful animation that soothes the character to sleep, one that produces nightmares, or perhaps one that induces a calm insomnia. The system selects video and sound clips from a database and edits them together on the fly, in a manner that preserves continuity, to dynamically convey the Sleeper's current state.



*Sashay/Sleep Depraved*

*Sashay/Sleep Depraved* was collaboration among a small but diverse group of people. Baird created the design, content, and physical structure for the piece, including high-level animation behaviors. She also directed two assistants, one who worked on the low-level sensor interface, and the other who made the

Amazon scene structure and video editing system. Isis played an instrumental role in fostering the collaboration by successfully supporting all three of these efforts. The language was efficient enough to support a complex pattern-based gesture recognizer using in-house sensing technology, and it allowed Baird to build animations and integrate them into the piece with a minimum of effort and without undue worry about exactly how the rest of the system was implemented.

# Jayshree

Stefan Agamanolis, Glorianna Davenport

In early 1997, the Interactive Cinema group began working on a collaborative experiment in "very distributed storytelling" called the *Dream Machine*, which was meant to incorporate audience experiences on the World Wide Web, wireless devices, and in physical installations in casual architectural spaces [Dav97]. Several transcultural characters were envisioned as inhabiting a story environment spanning these venues. One such character was Jayshree, a skilled traditional Indian dancer, around whom a public video art piece was created.

The piece consists of a large video projection situated in a public passageway, in addition to sonar sensors capable of discerning when someone is standing near the screen. When nobody is nearby, the dancer peers out into the space, beckoning passers-by to come over and take a closer look:



If someone does approach, the dancer puts on her makeup and begins to perform a dance:

If, however, the spectator loses interest and begins to walk away, she will stop and glare angrily, turn her back, and retreat:



It is at this point that the character becomes a dynamic entity, one whose feelings the spectator has hurt. The effect is startling and invokes a sense of guilt in many who experience the piece. The story environment takes on a new reality as well, one in which the spectator's actions have consequences, possibly far beyond this particular venue.

## The Birds

Sammy Spitzer, Glorianna Davenport

In the spirit of the *Jayshree* piece, members of Glorianna Davenport's Fall 1997 Workshop in Elastic Movie Time course at the Media Lab used Isis to create several interactive art installations, each incorporating a large screen video projection situated in a hallway and sonar sensors to detect the activity of passers-by. Four groups of students each worked together to build their own projects, which were exhibited publicly at the conclusion of the course. Isis was an effective development platform because of the video and custom sensor functionality it offered, and because it presented a shallower learning curve to the time-pressed students, many of whom had little prior programming experience.

One student, Sammy Spitzer, created a simple piece called *The Birds* in which the video projection shows pigeons pecking at food on a brick sidewalk. When someone walks down the hallway past the screen, the pigeons are startled and fly away, just as they would in real life. Several seconds later they reappear and the cycle begins again. The effect is surprising and humorous, transforming the unsuspecting passer-by into an imposter capable of disrupting a computational story environment.
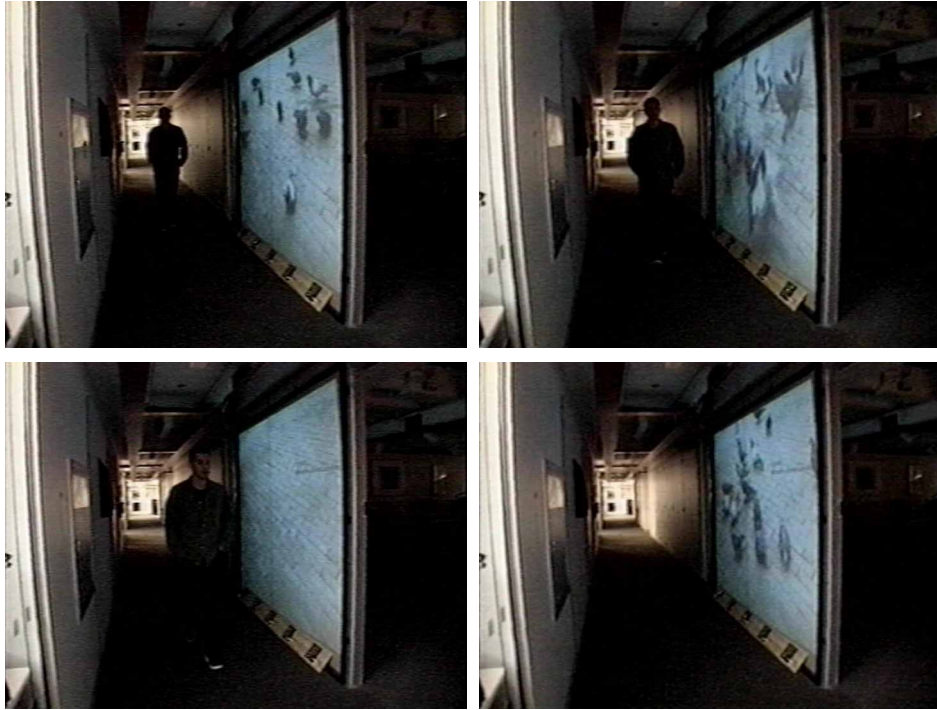
*The Birds*

# CINEMAT

Glorianna Davenport, Stefan Agamanolis, Barbara Barry, Brian Bradley

The *CINEMAT* is a collection of three interactive installation pieces, created as part of the *Dream Machine* project, each incorporating a large screen projection and a special piezoelectric sensing carpet capable of detecting footsteps [Dav00]. They were exhibited at both the 1998 Rotterdam International Film Festival and the Televisa-sponsored Espacio '98 convention in Mexico City. Isis was utilized at every step of the production process. The language supported a special tool for capturing and annotating video content for the project as well as an efficient real-time footstep tracking system.

In the first piece, spectators edit a movie by simply walking across or dancing on the carpet—each step or stomp causing a cut to a new video clip. Different kinds of video material appear depending on the location and weight of each step. The video clips, chosen by the system based on a complex set of rules, represent the dreams and memories of two characters, a girl and a boy, whose lives cross in a playful flirtation. Since it is highly uncommon to see the same series of clips twice, participants were encouraged to write about their individual experiences with the piece on a special web site in a process of coming to a collective under-standing of the story greater than any single encounter on the *CINEMAT* could convey alone.
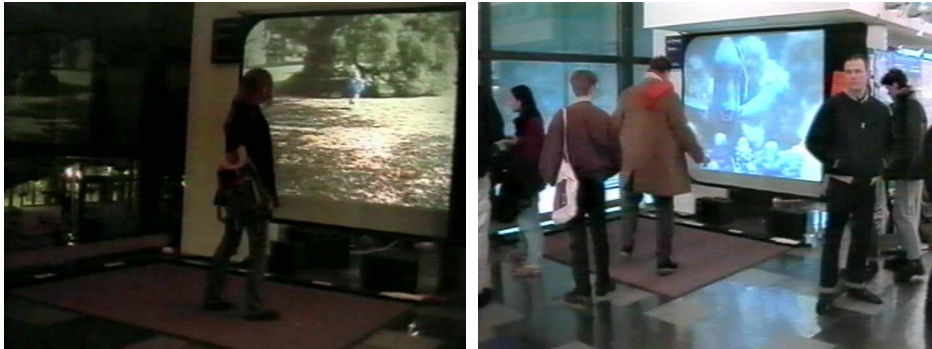


In the second piece, the participant becomes the victim of a sandwich thief and must chase a video character through the corridors and tunnels of MIT by run-ning on the carpet. Running faster or slower changes the speed of the chase.



And in the final piece, the participant must step carefully and quietly around the carpet to sneak to the top of a stairway and look through a keyhole to spy on the events behind a door. If the participant fails and is discovered, he is driven away by the inhabitants of the room without any reward.

# Magic Windows

Stefan Agamanolis, Barbara Barry

One more interactive installation to note is *Magic Windows*, which utilized a La-zyFish electric field sensor [Smi96] deployed in a unique fashion to detect a spectator's proximity to a small video frame suspended in mid-air. When no-body is nearby, the screen presents a still image from the beginning of one of several short video clips developed for the piece. When a spectator approaches, the clip plays forward in time to a point proportional to his proximity. If the spectator retreats, the clip plays backward in a similar fashion. The content of the clips is such that moving closer and farther increases and decreases the ten-sion conveyed in the image. For example, one clip shows a man slowly inflating a balloon such that when the spectator moves closer, the balloon becomes larger and larger, increasing the chances it will rupture. If the spectator tests the limits of the piece by approaching as close as possible to the video frame, a hand with a pin suddenly appears and pops the balloon, resolving the tension, and the rest of the clip plays and brings the story to a conclusion.
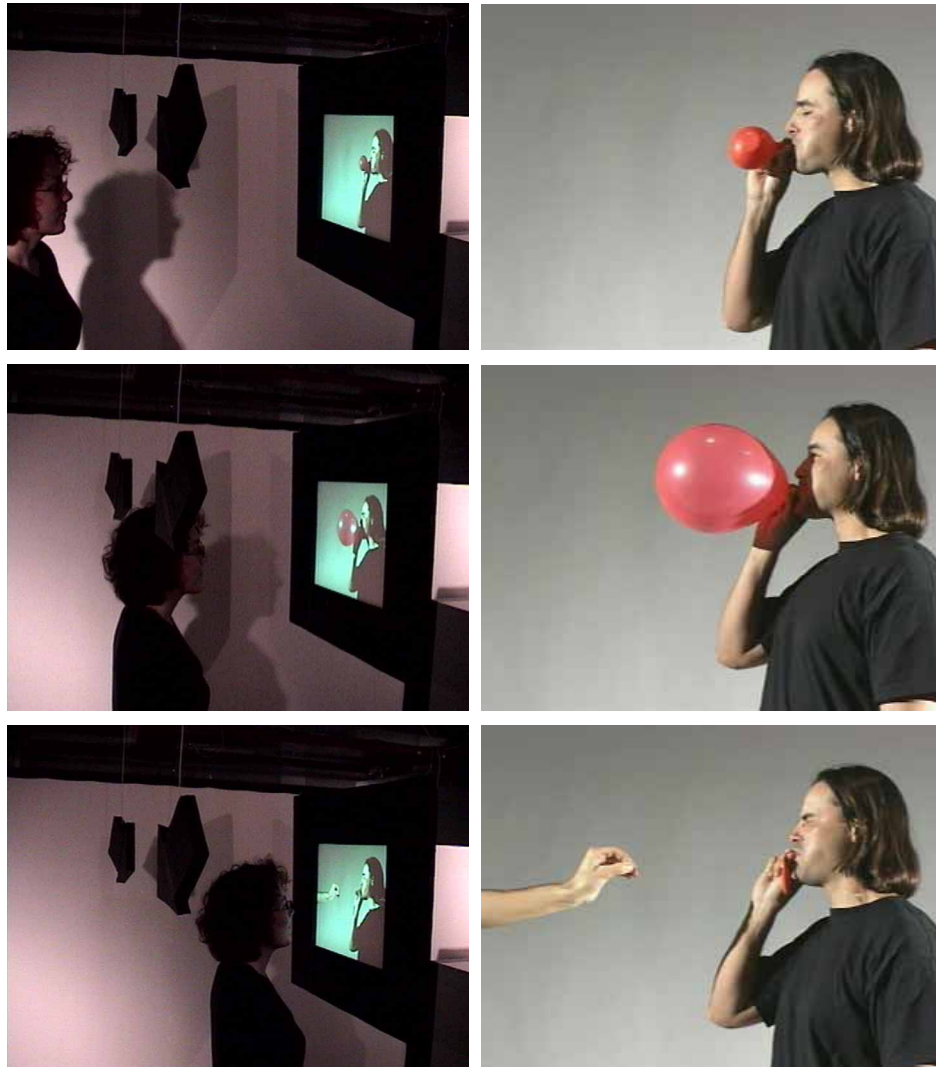


*Magic Windows*

# Telepresence

Isis has also served as the foundation for a number of extremely demanding prototypes relating to telepresence and video communities, three of which are described below. The most important aspect to note about these projects is that they all incorporate a very high degree of distributed processing, in some cases involving six or seven separate machines each running several Isis processes that all have to communicate with each other in the most seamless way possible.

## Reflection of Presence

Stefan Agamanolis, Alex Westner, V. Michael Bove, Jr.

*Reflection of Presence* is a teleconferencing prototype that creates a "magic mirror" in which one not only sees a reflection of oneself, but also the reflections of other participants from other locations, just as if everyone is standing in the same room looking at each other through a real mirror [AWB97]. The system detects speech and movement and uses these cues to dynamically vary the transparency, position and scale of each participant in a manner that reflects who the center of attention is at any moment. The participants use colored objects to interact with and customize the shared space in various ways, like selecting options from a menu, manipulating documents in the background, or controlling the playback of movies.

*Reflection of Presence* is one of the most computationally challenging projects ever written in Isis, incorporating over 10 processes running in synchrony on at least four separate workstations situated on a local area network. Client processes digitize camera images, extract participants from their backgrounds in real-time, and send the segmented images to a central server for layering, which in turn sends the composited image back to each of the clients to be displayed. Audio is handled by a separate set of processes that communicate directly between each other in a pseudo-multicast fashion rather than through the central server. Isis enabled rapid experimental changes to the behavior of the prototype as compared to a compiled language, without noticeably compromising efficiency. The system typically performed at full video rates on mid-range Unix workstations.

*Reflection of Presence*

# Vision Television

Stefan Agamanolis, Chris McEniry, Jacky Mallett, Bill Butera, V. Michael Bove, Jr.

The *Vision Television* employs a neural-network face detection algorithm [RBK98] to find and track the faces of its viewers. Once found, these faces are layered into the video image, together with faces of others viewing the same program from other locations, creating a visual community of television watchers. A two-way audio channel allows the viewers to communicate with each other. The robust face detection algorithm enables the system to operate in a much less controlled environment compared to that of *Reflection of Presence*, which requires a static background and controlled lighting to achieve good results. Additional analysis can be performed on the faces to recognize emotional responses or even the viewer's identity, and this information could be used to modify the video program in a useful way. Isis supported a distributed processing scheme that separated the video layering and playback part of the system from the face detection component, which ran at a slower rate.



*Vision Television*

# iCom

Stefan Agamanolis, V. Michael Bove, Jr.

The *iCom* is a multi-point awareness portal and video communication device that connects the MIT Media Lab with MediaLabEurope in Dublin. The most unique aspect of the system is its dual foreground/background design. It is meant to be turned on 24 hours a day to provide continuous ambient awareness between every station, but at any time, it can instantly morph into a dedicated foreground video conference system. The system synchronizes the screen projections at each station, ensuring that all participants see exactly the same thing. It also functions as a bulletin board for community messages and announcements, sent via email.

The *iCom* is probably the most complex distributed processing prototype created in Isis to date. The system must deal gracefully with the widely differing bandwidths available between the computing nodes at each station, from internal gigabit LANs to congested transoceanic links, while still providing low latency audio and video delivery and synchronized screen displays. The Isis Macaroni library serves as a basis for rendering these displays, which include images from every camera at every station. The system analyzes audio and video and adjusts transmission rates to conserve bandwidth when no activity is detected.





*iCom*

## Other projects

Isis has played a role in many other projects besides those discussed above. Christian Baekkelund used it to develop a touchscreen-based demonstration kiosk for The Garden at the Media Lab. Bill Butera used Isis to support an experimental model-based video codec that employs a collection of programmatic "experts" that make "bids" to encode regions of an image in as few bits as possible [BB00]. Surj Patel and Jacky Mallett utilize Isis to support interactive quiz shows and television programs that enhance their impact by controlling the behavior of objects in the viewing environment. Florian Mueller and Edison Thomaz harnessed the real-time image analysis components of Isis to create *Impact TV*, a unique television remote control in which viewers simply throw objects at the screen to change channels. The other two tools described in this dissertation, Cabbage and Viper, are written entirely in Isis as well.

# Evaluation

One of the problems in evaluating the success of a new programming language is the bias introduced by those with prior experience using and thinking in terms of other languages, and what they come to expect from new environments. Therefore, one of the best means to understand the viability of a language is to study how well novices and complete newcomers to programming can grasp the basic concepts of the language and use it to create something. In addition, we can study the extent to which a language changes working practices and supports a new and clearer way of thinking about problems within its intended domain, which in this case is responsive media.

In both of these respects, and with regard to the three basic design guidelines presented in earlier chapters, Isis has enjoyed a great amount of success, as illustrated by many of the project experiences described above. The technological design of the language and its interpreter has been able to support the demanding multi-sensory media materials and interfaces needed in these endeavors. But more importantly, the human interface design of the language—its syntax, libraries, and documentation thereof—has enabled its users to think about these complex and multi-layered design problems in a more organized, concise, and elegant fashion, and with better inherent support for collaboration. Anecdotal evidence suggests that it is much easier to master than its counterparts of comparable power. Many have successfully learned Isis on very short notice, days or even hours before an important demonstration or class project was due.

In its current form, Isis is not without a few weaknesses as well, and these lead to several initial ideas for further research. Although the usefulness of multi-threaded processing is often questionable, the memory management scheme employed by Isis would need significant changes to efficiently support a scenario where multiple Isis interpreter threads would execute within the memory bounds of a single machine process. Also, even though the Isis web site provides a good set of basic community support components, many improvements are needed, such as a search function and a more example-based approach to explaining the features of the language. In addition, Isis should have its own dedicated text program editor, one that could better incorporate the documentation, debugging, and community support tools directly into the coding process and include graphical organizational elements to assist in making programs easier to read and manipulate.

However, even if these weaknesses were dealt with, Isis would still be a primarily textual functional programming language, requiring its users to think about

and implement programs in terms of abstract rules and lists of instructions, which is not always the way many people think best about what they want to accomplish. To truly break new ground would require a reassessment of the usefulness of the *rule-based* design strategy inherent in virtually every widely used programming language ever invented. In this spirit, the next chapter describes an experiment that aims to push the boundaries of visual interface design and to explore the usefulness of employing a strictly *case-based* authoring approach in a tool for a specific subdomain of responsive media.

# Chapter 4
# Cabbage

## Realizations

The previous chapter describes a new programming language that, in its inherent design, aims to rethink many of the principles long taken for granted in traditional languages in order to provide a more appropriate platform for staging responsive media enterprises. However, its successes notwithstanding, Isis is still firmly rooted within the paradigm of *rule-based* thinking along with virtually every other programming language in existence.

Rule-based systems require a designer or programmer to describe the operation of an algorithm in abstract terms, in the form of instructions that describe manipulations on variables and data to produce a desired behavior. This strategy works well when the designer has a good understanding of the building blocks of particular kinds of algorithms and how they can be assembled to perform a task, or when the task itself is fairly simple and easy to grasp in abstract terms.

But there are problems with this approach too. For one, the task of exhaustively describing exactly what should happen in every combination of situations can be overly tedious, especially when the number of variables to which the system must respond is high. In many problem domains, such an exact description may be unnecessary, and the designer may be willing to accept some indeterminateness in return for a simpler expression of the solution.

Most of all, the rule-based approach does not match well with the way human beings are inherently skilled to describe and learn processes by example and deal with problems by adapting prior concrete experience. A rule-based approach somewhat assumes one "knows the rules" in advance, and many designers and artists often have hypotheses they want to test. But this approach generally conflicts with the process of organic discovery and growth in which many engage to experiment with the raw materials in search of interesting mappings and possibilities.

## Case-based reasoning

Case-based reasoning represents a different perspective on how one might program a computer to perform a task not by writing an exhaustive list of instructions but rather by showing the computer examples of the desired behavior for certain situations and allowing it to generalize from those cases to produce appropriate behaviors for other situations.

In many ways, this approach mirrors the way the human brain generalizes knowledge from past experiences in order to deal appropriately with new and novel situations. For example, in a dancing class, an instructor teaches different steps, showing variations of each routine depending on the rhythm of the music,

the amount of floor space available, the characteristics of one's dance partner, and so on. Outside of class and away from the instructor, when the student encounters a new environment, a new partner, and an unfamiliar piece of music, he will adapt the "examples" learned in the class as best he can to deal with the new situation. Rather than invoking a list of memorized rules, he recalls his old dancing experiences, recognizing any similarities between them and the new situation, and appropriately combines elements of those experiences to produce a new behavior.

Christopher Riesbeck and Roger Schank eloquently state the root strategy:

> "Case-based reasoning means reasoning from prior examples. A case-based reasoner has a case library. In a problem-solving system, each case would describe a problem and a solution to that problem. The reasoner solves new problems by adapting relevant cases from the library." [RS89, p. 25]

In their acclaimed book, *Inside Case-Based Reasoning*, they go on to describe different styles of case-based reasoning, examples of such systems, and comparisons to rule-based approaches. Agnar Aamodt and Enric Plaza restate the basic idea this way:

> "Instead of relying solely on general knowledge of a problem domain, or making associations along generalized relationships between problem descriptors and conclusions, case-based reasoning is able to utilize the *specific* knowledge of previously experienced, concrete problem situations (cases). A new problem is solved by finding a similar past case, and reusing it in the new problem situation." [AP94]

It is interesting, if not critical, to note that case-based reasoning is not completely devoid of rule-based principles. When the case-based reasoner must adapt previous cases to address a new situation, it must apply a specific rule-based method to first analyze the similarities between the old cases and the new one, and then combine elements of the old cases to generate a proper new behavior. These "adaptation rules" can be complex and are typically very domain and task specific, but their existence does not necessarily imply that case-based reasoning is just rule-based reasoning in disguise, as Riesbeck and Schank point out:

> "… the advantage of case-based reasoning is that, messy though they may still be, the adaptation rules can be much simpler than those required by a purely rule-based system, if the case library is reasonably filled out. In many real-world domains it is very difficult, if not impossible, to create a complete set of rules, and applying large numbers of rules is very inefficient. A case-based reasoner can get by with a much weaker set of rules, if the case library is broad enough." [RS89, p.42]

Although the details are too many to discuss in this brief document, the human brain handles this adaptation and generalization process in a particular way. The exact development and function of the brain structures involved varies from person to person based on a variety of hereditary and environmental factors, and so despite any similarity in background, different people have different ways of dealing with new situations. In our dancing class example, the students unconsciously adapt their dancing memories in slightly different ways, resulting in slightly different behaviors.

However, in order to create an artificial system capable of this kind of case adaptation, we need not model the complexities of the human brain in every respect. In certain restricted problem domains, it may be possible to distill a few concrete rules that embody the ways trained human beings handle case adaptation in that domain. For example, consider *CHEF*, a case-based reasoner that adapts old recipes to generate new ones that satisfy the needs of a new situation [Ham89]. In this system, the adaptation rules describe various well understood principles of modifying recipes—what kinds of substitutions are viable, how cooking times are dictated by the addition or removal of ingredients, what steps need to be added or eliminated, and so on. Although these rules represent a mere small subset of the knowledge and skills held by good human chefs, they are quite enough to handle many complex recipe modifications.

# Case based reasoning in design

The fundamental case-based reasoning strategy described above can be adapted to the domain of media design by thinking of media delivery contexts as the "problems" and the corresponding appropriate media to deliver in those environments as the "solutions." A delivery context may incorporate any combination of factors such as the type of presentation equipment in use, the preferences or actions of the audience, the conditions in the presentation area, and so on, as described in Chapter 2. The designer will provide several concrete examples of particular contexts and the corresponding media to present, and the system will adapt these examples to handle new and unexpected delivery contexts in an intelligent manner.

## Related work

Case-based thinking is certainly nothing new in the design world—every designer approaches a new problem largely by remembering previous experiences—what worked and what didn't—and adapting that knowledge in the creation and validation of a new piece. Maher and Garza present a broad survey of the methods many have applied to automate various aspects of this process in tools that support designers in the completion and evaluation of new works, or even systems that can generate new designs completely automatically [MG97]. They reference examples ranging from *Julia*, a system that automates meal planning by applying constraints learned from previous meals [Hin88], to *Kritik*, a tool that assists in repairing problems with mechanical devices using specific knowledge about the behavior of similar components in other devices [GBS97].

The Media Lab has been active in the domain of automated and assisted design for several years as well, with the work of Louis Weitzman figuring especially prominently:

> "Designers can no longer be present in the production cycle of every document. This suggests tools to *meta-design* descriptions of documents. This activity of meta-design creates descriptions that can subsequently be used to automatically design information under dynamic conditions." [Wei95, p. 12]

Weitzman utilized relational grammars to support a system that allowed its users to build layout rules through a graphical example-based interface. A designer could manipulate concrete graphical elements, such as titles, bylines, captions, and images, on a virtual drafting board, and the system would automati-

cally recognize relationships between these elements, such as aligned placement and similar sizing, and use them to generate a set of rules that could later control the automatic layout of particular sets of information. For example, a designer could create a layout "language" for a table of contents for one style of magazine, and other languages for other styles of magazines, and the system would apply these languages to automatically generate complete table of contents designs incorporating a desired set of contents information.

This approach is not strictly case-based in character, however, because although the designer can build several different visual "languages" for different layout designs, the system does not adapt and combine these languages in any way to create new ones appropriate for other situations. It is in this realm that the second tool of developed in this research, *Cabbage*, attempts to innovate.

# The Cabbage strategy

The second tool developed in this research, *Cabbage*, is a case-based reasoning system for developing responsive graphical layouts that can automatically redesign themselves appropriately as presentation conditions change.

## Multi-layered tasks and thinking processes

As discussed earlier, designers of responsive media often think visually about their problems and possess inherent skills for understanding and describing processes by example. With this basic realization in mind, Cabbage is chiefly an experiment, intended to push the boundaries of visual interface design and to investigate the usefulness of a purist case-based approach for creating responsive media in a complex problem domain. In this respect, Cabbage primarily addresses the third guiding observation presented at the beginning of this dissertation, that responsive media projects consist of **multi-layered tasks and thinking processes**, although the other two observations still inform many of the decisions made in Cabbage's design.

Cabbage aims to provide an intuitive, completely visual environment in which the designer can indicate the context factors to which a layout must respond and then demonstrate proper designs for particular contexts. In a purist case-based spirit, the goal is for the system to require as little rule-based human involvement as possible in order to infer the structure within and recognize common elements across the demonstrated designs. The designer will not be required to write any text program of any kind or encode any sort of abstract rule describing a response behavior. The user need only concern himself with creating concrete design examples appropriate for certain contexts. The system will adapt and combine aspects of these designs completely on its own to generate new designs for other contexts.

## Application domain

As an example of the task domain to which Cabbage is suited, suppose you are a graphic designer faced with the problem of creating an advertisement that will be displayed in hundreds or thousands of different ways—on a billboard, in a magazine, on a coupon, in a subway train, in an interactive web site, on the side of an old barn, and so on. Each situation consists of variations in the presentation context—canvas size, aspect ratio, audience type, viewing time, and so on—and your goal is to specially tailor the ad for each individual scenario.

Rather than make a thousand separate designs all by yourself, you might furnish examples of what you want the ad to look like in a few situations to the members of your staff and ask them to build the rest of the designs for you.

However, you may be in trouble if you don't have a staff, or if you are very short on time. Or perhaps you envision an advertisement that automatically reconfigures itself based on live sensor feedback, if such is available in certain presentation environments. And further suppose that you are not an experienced computer programmer and feel you are not likely to become one any time soon. A tool like Cabbage could be useful in such a situation. Here are some other application ideas:

- A responsive store window display that changes the way it presents its information based on the distance and activities of the people standing near it.

- A responsive airport display that redesigns itself to present information most effectively based on the walking speed and direction of passers-by.

- A responsive refrigerator door that displays information in an arrangement that is most appropriate for the time of day and the current activities in and occupants of the kitchen, and perhaps even the physical heights of these occupants which may range from children to adults.

# Visual interface

Cabbage aims to provide a visual interface that consists of features familiar to users of common computer-aided graphic design programs, but where the goal is not to create just one design but rather several designs of the same message. Each design represents an appropriate display of the message for a certain delivery context, and as such, they can be thought of as residing at points in a multi-dimensional "design space" whose axes represent the individual context factors that affect the look of the design (size, aspect ratio, viewing distance, viewing time, or whatever). The job of Cabbage is to, on request, appropriately adapt these examples to generate designs appropriate for other points in the design space.

The graphical user interface of Cabbage rethinks several aspects of interface design that have become commonplace in other tools. Written in Isis, utilizing the Macaroni object-based media library and OpenGL for graphics rendering, all of the interface "widgets" in Cabbage have been designed and implemented from scratch, giving it a unique look and feel.

Cabbage supports a single pointing device with a single button, and no more, which enables it to operate effectively in pen-based tablet interfaces. In the current version, the designer uses a separate editor to enter any text pieces needed for the design, prior to launching the main tool. To support experimentation, there are no commands or shortcuts invoked by keyboard, and there are also no pull-down menus and dialogue boxes, as these constructs often hide functionality or make it difficult to access or discover what operations a tool supports. Every action that a designer can take at any moment is displayed directly in the interface in a fashion that keeps the screen as uncluttered and organized as possible.

Below is a screen grab of the Cabbage interface:
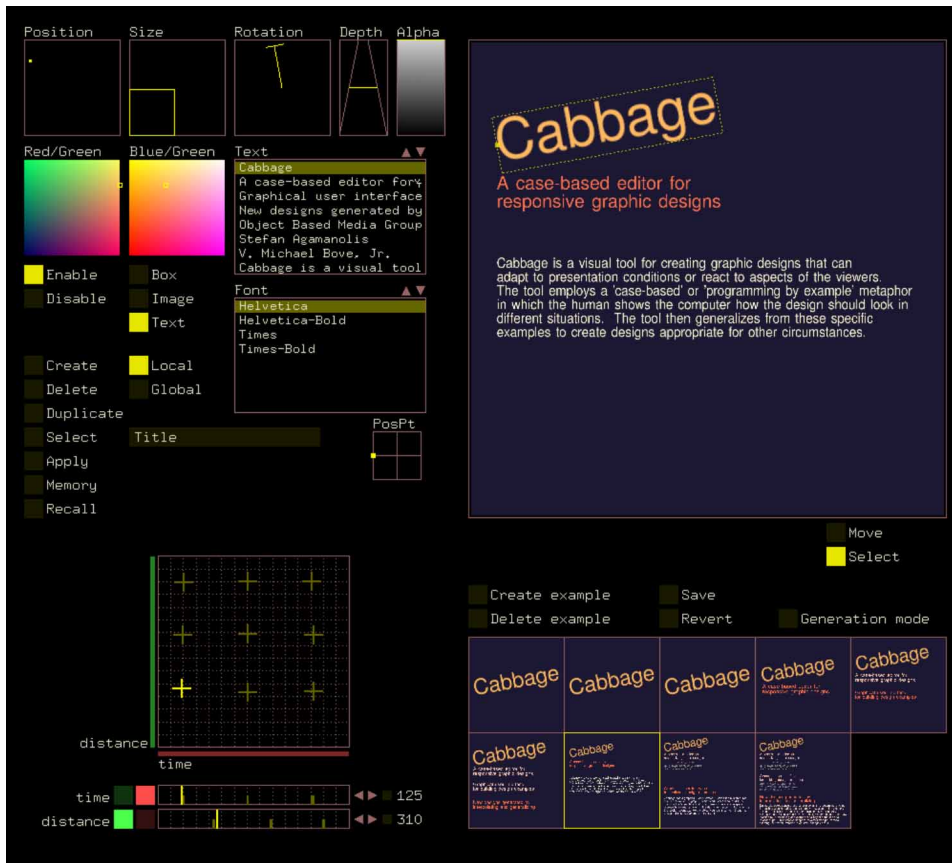


There are four main sections of the tool—the design space axes (lower left), the design case library (lower right), the object manipulation controls (upper left), and the canvas itself (upper right). Each of these is described below.

## The case library
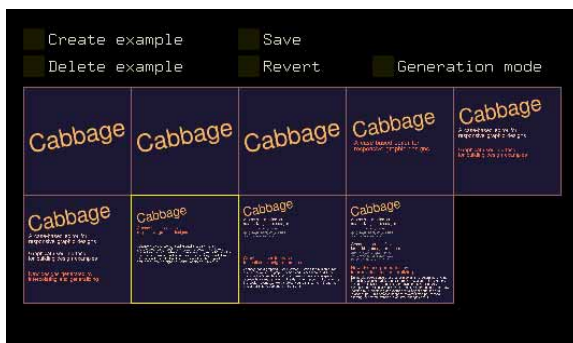


When the tool is first launched, the case library is completely empty, and the tool is ready for the designer to create the first design case. Clicking the `Create Example` button begins a new design case, which is initially completely empty. As designs are created, they are shown in miniature in the case library, to serve as a frame of reference for new designs. Clicking on a design selects it and

causes it too be displayed in the canvas. The user may duplicate or delete the selected case, as well as save and restore the entire library to and from a disk file.

## The canvas
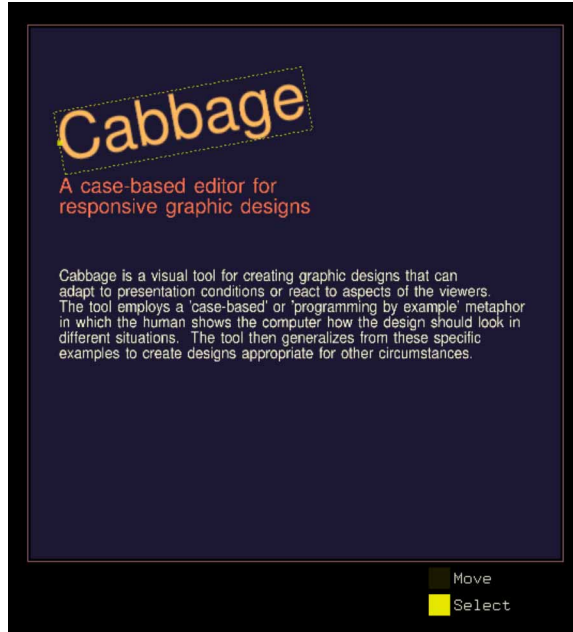


The canvas is the area of the screen that shows the actual design under construction. Only the currently selected design case is displayed here. An empty canvas simply signifies that there are no media objects in the design as of yet, or that all of the objects are invisible. The user may select and move objects by clicking and dragging. If several objects overlap, clicking on the same location several times will cycle through selecting the separate objects layered at that point in the canvas. The designer can modify various properties of each object in the object control section of the tool, described below.

# The object manipulation controls
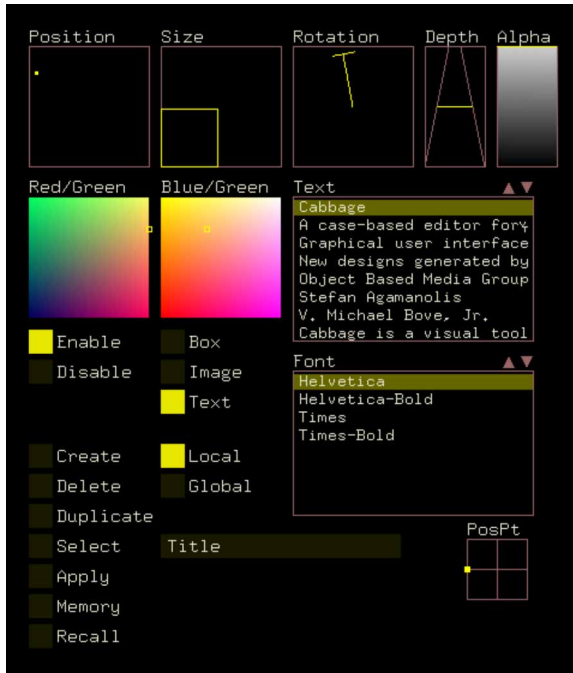


The first use of the object control area is to simply add new media objects to a design. Currently the tool supports three simple object types: box, image, and text. The `Delete` and `Duplicate` buttons perform the expected function on the object selected in the canvas. The `Select` button cycles through selecting all of the objects in the design, in case one cannot be easily reached on the canvas.

When the user creates a media object, it is added to every design case in the library, not just the currently selected one. The user may make an object invisible in certain design cases if it is not needed there, but as a matter of principle, every design case consists of an instance of every object, which greatly simplifies the problem of later determining object correspondences in the generalization component of the system. The user may also give a name to each object, which helps identify which one is selected if it is not easily visible in the current canvas.

When an object is selected, controls appear that represent all of the manipulations the user may perform on the object's properties. Each of these controls has a special design that visually illustrates the current value of each property as well as the range of possible states. There are controls for position, size, rotation, depth, color, and transparency. The user can also change the positioning point of the object.

Other controls appear only when the selected object is of a particular type. An image selection control lets the user select the source media of an image object, and a cropping control enables zooming in on a desired portion of an image. Font and text controls control the look of text objects, as well as the actual text that appears, which is entered in a separate editor prior to using Cabbage.

With the `Global` button, the user may specify whether he wishes to affect the properties of the selected object locally, in the currently selected design case showing in the canvas, or globally, in every design in the case library. Some properties, however, are always affected on a global level, such as the positioning point, image, text, and font settings.

70
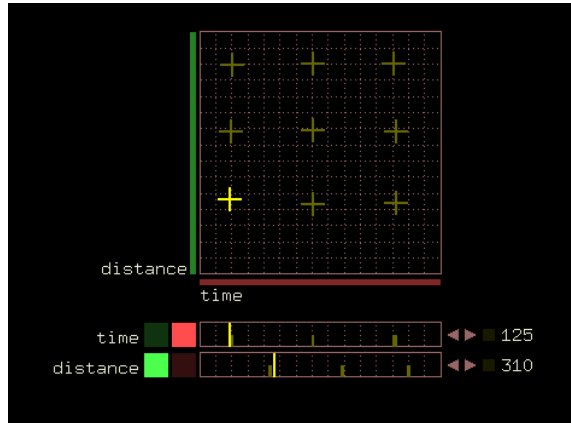
## The design space



Prior to launching the tool, the designer defines a name and a range of numerical values for each axis of the design space. Although only two are shown in the above image, there may be as many axes as needed to represent all of the context factors that will affect the design. As each case is completed, it must be placed at a particular point in this multidimensional space using the controls in this portion of the tool.

Each axis appears separately as a horizontal slider. The user can change the position of the currently selected case along each axis by simply clicking on the slider. The slider highlights the current value in yellow, but it also shows the values of the other unselected cases as dimmer tick marks. These marks can help the designer recognize what areas of the design space may be filled out too densely or sparsely.

This part of the tool also consists of a grid that enables viewing the relationship between two particular context factors. The user controls which two factors appear on the horizontal and vertical axes of the grid by clicking on the red and green axis selector buttons next to each slider. The grid works analogously to the one-dimensional sliders. The position of the current case with respect to the two selected factors can be changed by clicking directly on the grid. The positions of the other unselected cases are shown dimly. If one area is very sparsely populated, the designer may choose to add cases in that region to fill out the space more completely.

# Case adaptation algorithm

The most important component of Cabbage, its case adaptation algorithm, was also the most difficult to develop. Three main goals were kept in mind throughout. First, the algorithm should be able to generate new designs fairly instantaneously, at least several times per second. This requirement somewhat rules out extensive tree-search approaches for finding an optimal design in the face of ambiguity. Second, the algorithm should support extrapolation. That is, if a design is requested for a point outside of the main "cloud" of design cases provided by the user, the system should be able to analyze and extrapolate trends in the region of the cloud nearest the new point to determine a new design. Third, the algorithm should infer as much as possible on its own about the relationships between the media objects in each design and be able to instantiate similar relationships in an adapted design, or at least apply common rules of good graphic design to clean up any problem areas.

There were a few lesser goals as well, such as the algorithm should be able to deal with a very sparsely populated design space, as well as with areas of high case density, in an intelligent manner. Also, the algorithm should support continuity over an arbitrary path of solutions in the design space to whatever extent possible, so that any dynamic changes in the delivery context will result in smooth transitions in the look of the design.

Cabbage fulfills all of these goals to a certain extent, although these towering objectives are not to be taken lightly. Cabbage is an initial exploration into this realm, and there is much room for improvement and further research that will guarantee the problem of creating a truly general adaptation algorithm for this domain will remain an open question for a long time to come.

## Select nearest case

The current algorithm evolved incrementally over several months, beginning with the simplest possible approach, in which the system selects the design case nearest to the new point in the design space and uses that design without any changes. Nearness is measured by first normalizing the values of each context factor within its range and then computing a multidimensional distance in the usual way. This first-order algorithm is completely unintelligent, but may be appropriate for certain applications.

## Combine nearby cases

A second-order algorithm performs a linear combination of individual object parameters (position, size, color, and so on) from the library cases, weighted inversely by the distance of the old case from the new point. Then, a clean-up phase restores relationships between objects, aligning edges of objects that are already nearly aligned, and also moving and resizing objects slightly to ensure that any objects that did not overlap in the library cases do not overlap in the newly generated case.

Although this algorithm accomplishes the continuity goal, it is still quite unintelligent. It does not support extrapolation, nor does it properly heed any structure evident in the original library designs. But its biggest problem is in what happens in dense areas of the design space—if the new point has several library points in its immediate vicinity, there is no sense that certain cases are "behind" others, as viewed from the new point, and perhaps should have their contributions to the final design correspondingly reduced.

## Combine nearby unoccluded cases

The third algorithm incorporates the notion of points being "behind" others by performing a linear combination of cases, weighted inversely by distance and also by an occlusion metric. This metric is computed for a given library point P by considering all the other library points nearer to the new point N. For each nearer point Q, the system computes a dot product (dp) of the normalized vectors Q-N and P-Q. If the result is greater than zero, as would be the case in the following illustration, the point P is considered to be "behind" point Q to an extent proportional to (dp), and the contribution of point P to the final design is scaled by 1 – (dp).
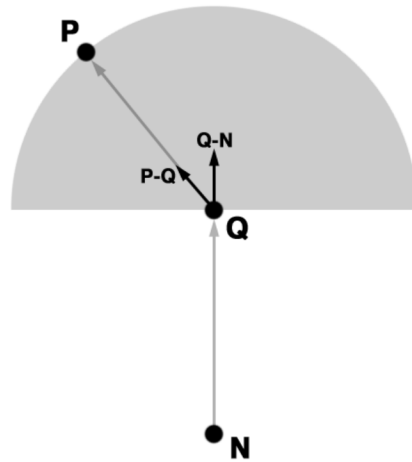


The same clean-up operation is performed on the resulting design. Although this algorithm still does not support any sort of extrapolation, it works surprisingly well, able to generate designs that intuitively make sense based on the contents of the case library.

## Combine nearby unoccluded extrapolations of cases

The fourth and best algorithm available in Cabbage abandons a simple linear combination approach to adaptation in favor of a limited trend analysis strategy that is capable of extrapolation. The system makes a list of lines passing through pairs of nearby library points. It then projects the new point onto each line and computes object parameters at the projection points by linearly interpolating or extrapolating the pair of library designs on each line. Each of these projection points is assigned a contribution factor equal to the product of the inverses of the distances of the two associated library points from the new point.

The algorithm continues by invoking the same occlusion metric as used in the previous scenario, but this time considering the projection points instead of the actual library points. The occlusion scale factor is applied to the contribution of each projection point, and the final output of the algorithm is a linear combination of the object parameters associated with each projection point, weighted by the final contribution factors. Qualitatively, this algorithm is more robust when the case library is very sparse, and the designs it generates make more intuitive sense than those created by the third algorithm, especially at points "outside" of any main cloud of design examples.

# Refining and presenting the result

The user can test the selected adaptation algorithm by clicking on the `Generate` button in the tool. The algorithm is invoked on the case library as it stands (minus the currently selected case) to build a new design for the design space point indicated on the design space axis controls. Each of the four algorithms is capable of generating designs several times per second, which enables the user to drag around in the sliders and grid to explore the design space with immediate feedback and smooth transitions. If the system generates an improper design, the user can make corrections and add it back into the library as a new case.

When the designer is satisfied with the system's performance, he can relinquish this manual control and use a playback program to connect the context factors for the design directly to real sensing mechanisms. This playback program presents the design full-screen and dynamically alters it with smooth transitions in response to sensor activity.

# Example project

In early 2000, Cabbage was used to create a responsive poster suitable for a Media Lab research open house or similar type of affair. The subject of the poster is Cabbage itself, and the design responds to two variables about the viewer:

- the distance of the viewer from the poster

- the length of time the viewer has been standing in front of the poster

The goal of the design is to grab attention and draw the viewer closer when he is far away, and then to try to retain the viewer for a longer period by presenting additional information over time. The two context factors are named "distance" and "time," and a total of nine design examples form the case library for the poster, as shown below. These designs are placed in a rough 3x3 grid arrangement in the design space, although this is not a necessity—any arrangement is allowed.
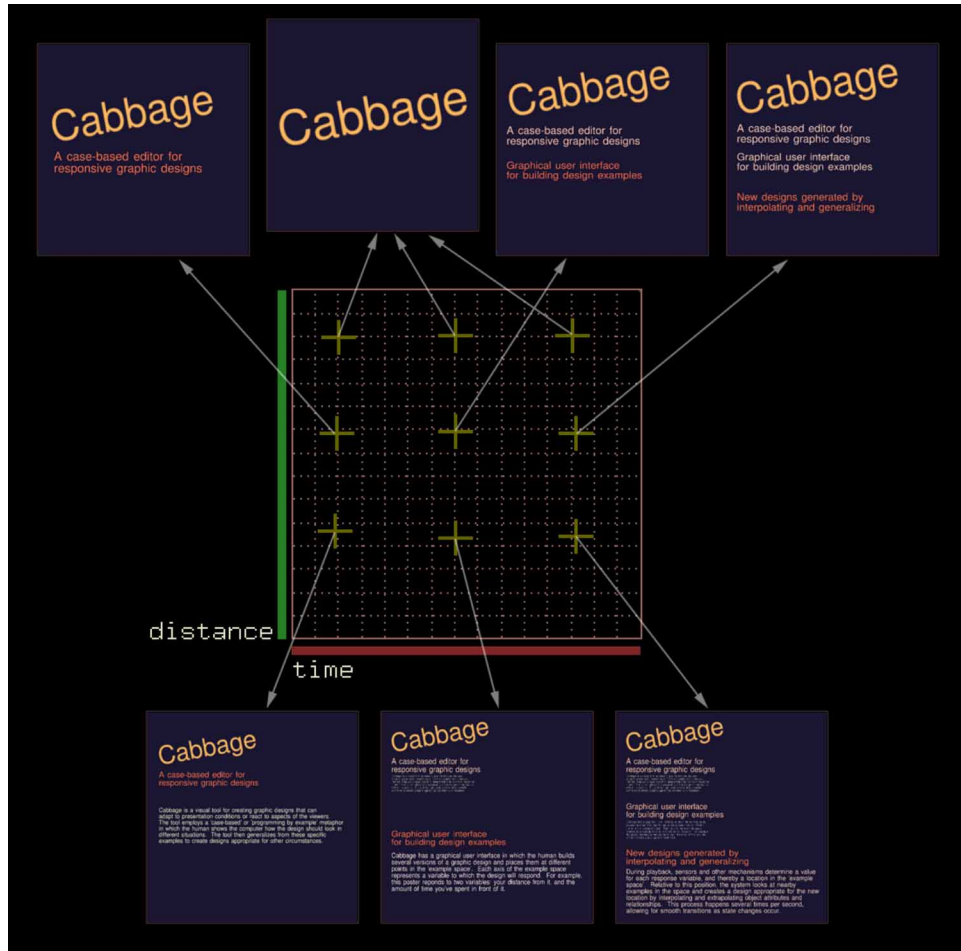


The intention was to place the design under the control of real distance sensors and timers. The system supported smooth transitions between intermediate designs. Below are some examples of how the design would respond in different situations.

When someone first walks in to the presentation space, the poster presents a single large title, to catch attention. If the viewer approaches, the main title will grow smaller and a subtitle will appear, further drawing the viewer in. When the viewer is close enough, the headings become smaller and a detailed description appears. If the viewer remains, a second and third page will appear. Below are a few stills from this kind of sequence:





1



2



3



4



5
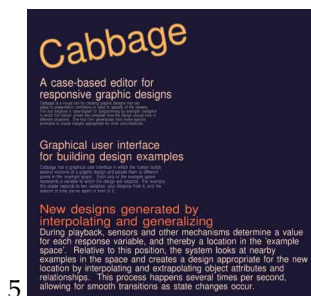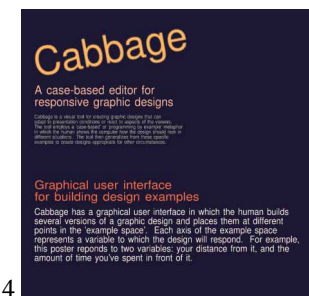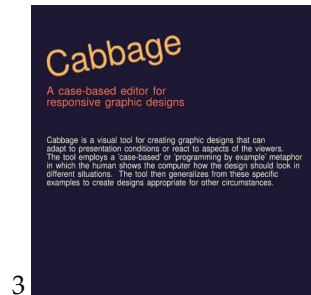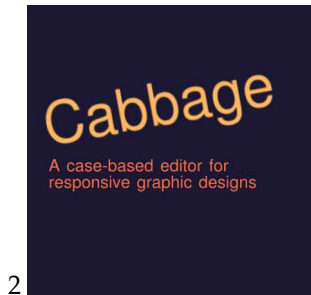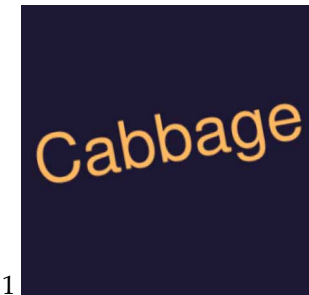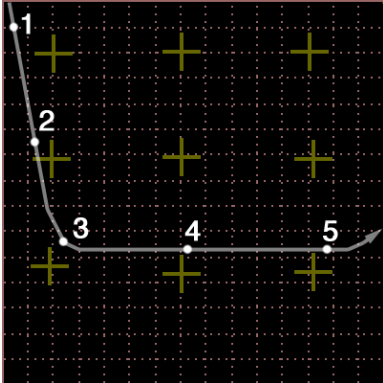
On the other hand, if the viewer only ever approaches halfway and then leaves early, the full descriptions will never appear:



1



2



3



4
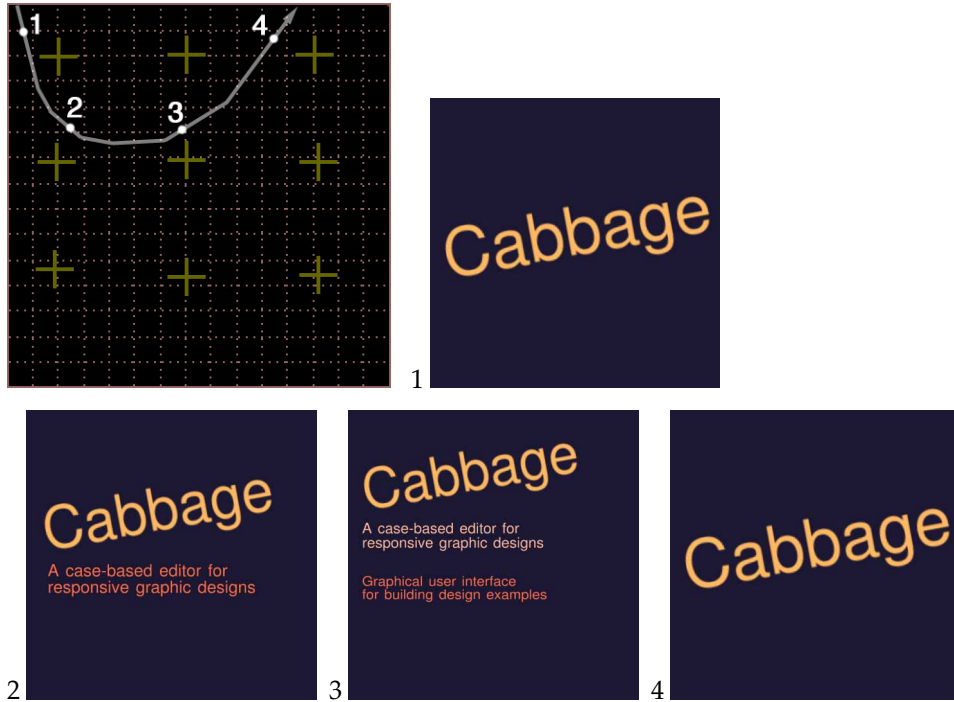
If the viewer approaches very close to the presentation screen, closer than was ever expected when the designs were first entered into the system, the adaptation rules support an extrapolation of the design where the headings and descriptions appear even smaller than they were originally demonstrated by the designer. This is an appropriate reaction as someone standing very close will find it difficult to read large text, although Cabbage accomplished this simply by extrapolating the trends in the nearby cases, not by applying any higher-level knowledge about the viewers of the poster.



1



2



3



4

# Evaluation

Perhaps the most successful aspect of Cabbage is its visual interface, which, as discussed, consists of widgets and controls implemented from scratch in Isis with OpenGL as the rendering engine. Showing each design in the case library in miniature makes it easy to compare designs and manipulate objects on a global level. Emphasizing visibility and accessibility to an extreme yields an interface that is more amenable to a multi-layered design process rooted in discovery and experimentation. Perhaps the best feature is that users of Cabbage deal only with concrete layout design examples and never have to write any code or describe behaviors with abstract rules. All of these results together suggest that Cabbage has the potential to support a much wider variety of users than typical responsive media design tools.

Despite their ability, at least in the simplified scenarios tested, to create results that intuitively make sense with a minimum of complex math, there are some important weaknesses with the adaptation rules employed in Cabbage. For one, none of them inherently take into account relationships that may exist *between* objects. For example, the system does not currently recognize if a text box is consistently placed below and left-aligned to an image object in every case and try to preserve that relationship in any newly generated design. Object parameters are

combined in a mutually exclusive fashion, and the "clean up" routine attempts to restore graphically appropriate relationships between objects afterwards.

However, it would not be difficult to implement a more structured approach. Weitzman demonstrated a method to deduce certain object relationships—if a user situates two objects near each other and nearly aligns them or makes them nearly the same size in a particular dimension, the system assumes that constraint was meant to hold, and if thereafter one object changes position or size, the other will follow correspondingly [Wei95]. However, his system only handles the detection and propagation of relationships, and not what should happen if, for example, the designer shows an object aligned with another in some cases but not in others. A system would need not only a method to deduce the structure hierarchy of a layout, but also to appropriately "interpolate" between and combine bits and pieces of several hierarchies to generate new ones. In addition, in situations of high ambiguity, it may be much more satisfying, for a designer who desires greater control, to explicitly indicate object relationships rather than have them deduced automatically.

However, even if these weaknesses were eliminated, there are still deeper issues to consider that threaten the viability of Cabbage and other case-based tools like it. Since the adaptation rules do not often handle sparsely populated design spaces very well, simply adding one additional context variable to a project can double or triple the number of designs the user must enter into the system in order to achieve good results. But more importantly, given the incomplete and amorphous role of the adaptation rules employed in case-based reasoners, a designer who uses Cabbage must be willing to accept a certain amount of indeterminateness in what the system will generate. This can be extremely frustrating, however, as many designers are attracted to the computer precisely because of its capability for exactness—because it *doesn't* reason like a human being.

The adaptation rules themselves may not match well with a particular designer's style, and understanding how these rules work to the point of modifying them could prove to be daunting. Because the algorithms in case-based reasoners like Cabbage cannot be accessed or changed through a friendly interface, if at all, designers may try to assert more control over the system by adding more and more examples to the database, but through this process, designers often learn enough about the input-output mappings they are trying to develop that a more rule-based approach begins to make more sense.

In the final analysis, as it stands now, Cabbage is not suitable as a serious production environment for making responsive layout designs. With a few changes, it could function perhaps a design exploration tool, allowing designers to experiment with mappings and combinations in a process of understanding their problems more fully. Since it is based in Isis, case libraries created in Cabbage can be stored in a format that can be manipulated from within an Isis program in a more rule-based fashion.

However, Cabbage *is* successful as an experiment to understand what kinds of things visual thinking and case-based reasoning can and cannot accomplish in a responsive media design tool, and this knowledge will ultimately prove very helpful in the design of the third new tool developed in this research, described in the next chapter, which utilizes many of the best visual interface concepts of Cabbage but returns to a rule-based approach for creating a different, very specific kind of responsive media.

# Chapter 5

# Viper

## Observations

A common thread that runs through many of the prototypes described in Chapter 3 is the idea of creating context-aware video programs that can re-edit themselves in response to a variety of factors—presentation equipment, viewing conditions, real-time audience activity, and so on. *The Birds*, *Jayshree*, *Magic Windows*, and the *CINEMAT* all consist of a dynamic editing component in which individual video clips are selected and assembled in real time in response to the movements of the spectator. The background of *Sashay/Sleep Depraved* is a movie edited on the fly to convey the mental state of the main character, altered through gestures of the spectator. *Reflection of Presence* and *Vision Television* incorporate activity, emotion, and identity detection elements that could drive intelligent alterations in the media being presented. Direct viewer control over the navigation of material is a central element in the *Aspen Movie Map* [Moh82], the *Movie Manual* project [Gan83], and many other earlier interactive video experiments.

Each of these prototypes is comprised of very similar components—a database of video clip objects, a set of rules governing how those clips should be selected and combined, and a playback program that actually renders the media. However, the lack of any specialized tool for making content of this type resulted in much extra time being spent tediously annotating video clips and coding database-lookup and editing routines from scratch in textual computer programs. Furthermore, because these programs were written without a common framework or set of primitives, the individual components that make them up cannot be shared or combined easily to make new systems. The designers of these projects will also find it more difficult to understand the inner workings of each other's systems.

The similarity in the types of tasks involved in making responsive video programs, together with the lack of a common framework for this kind of media and the non-optimality of using a general-purpose programming interface for many parts of the process, raises the idea of creating a new tool geared specifically for authoring this kind of content.

## Motivations

Consumers of Internet-mediated content have become accustomed to personalized and responsive media experiences that better match their needs and circumstances. The World Wide Web has presented new opportunities for storytelling as well, enabling authors to endow audience members with new freedoms to navigate and explore complex story worlds, whether fictional or non-fictional, in ways suited to each individual.

Producers of television and other video content have been left somewhat behind in this revolution, still often restricted to creating one-size-fits-all programs, or perhaps a very small set of alternative versions of a program for different audiences. But now that there is significant computational power and intelligence throughout the chain all the way from production to display, and now that there is often sufficient "extra" bandwidth available, it is possible to perform much of the final editing process on the receiver at the same time the program is viewed.

In a broadcast or multicast scenario, the source no longer transmits a linear video program, but rather a collection of media clips and objects along with procedural metadata, authored by the producer, describing how the receiver should assemble those objects, adjusting for whatever factors, to present a complete television program. Besides allowing for a fine-grained adaptation of content to each specific viewing situation, this approach doesn't require potentially sensitive information about the viewer to be transmitted outside of the viewing environment. Possible applications include advertisements that are more specifically targeted to particular people or situations, news programs that take into account a viewer's background, educational programs that adjust in real time to a student's attention level and other emotional responses, or environmentally responsive video installations in public spaces. The ultimate goal is not to take editing control away from the creator of a program and place it in the hands of the viewer, but rather to give video producers and directors *more* control over how their stories and messages are presented in different situations and over what freedoms, if any, are granted to the audience to pursue their own exploration of the content.

Providing at least a coarse degree of personalization, such as showing one of a small number of different versions of an advertisement transmitted in parallel bitstreams, has been relatively easy given currently deployed infrastructure [Sim00]. But achieving finer-grained personalization or real-time responsiveness has come at the cost of developing complex pieces of software using interfaces that do not match well with the thinking and working styles of most traditional video producers and editors, who are used to manipulating audio and video source material in a more direct and hands-on fashion.

# Automated and dynamic video editing

There is a moderate body of prior work that explores different ways of thinking about automating part or all of the editing process for video programs, in some cases where this editing is guided in part by the viewer or audience.

Frank Nack and Alan Parkes developed AUTEUR, a tool that employs a hierarchical system for describing shot content and scene structure that can generate a limited form of slapstick-style comical video sequences [NP97]. Their system incorporates a fixed knowledge base and editing model, gathered in part by observing real video editors at work, that dictates the rules by which clips are chosen and sequenced to accomplish a simple thematic goal, in this case humor. However, it is unclear how such a model would be extended to handle other themes or genres, or how it might successfully drive the editing of scenes more than a few seconds long without becoming inordinately complex and unwieldy. In addition there is no clear strategy for determining what type of video material needs to be captured and included in the system's database in order to fulfill the needs of AUTEUR's automated story planner.

Michael Mateas, Steffi Domike, and Paul Vanouse created a system called *Terminal Time* that assembles a montage of archival film and video footage to accompany a spoken narrative describing historical events [MDV99]. The focus and

bias of this constructed narrative is dictated in part by audience feedback in the form of answers to multiple choice questions at several points during the presentation. However, clips are selected from a database with a simple term indexing scheme to reflect the subject of the narrative as it progresses, without attention to whether the resulting juxtaposition of shots is cinematically appropriate or meaningful.

Rainer Lienhart has developed techniques for making "on the fly" annotations to home movies and for automatically summarizing those movies into short video abstracts [LR00]. Content is organized in a hierarchical fashion, and different kinds of transitions set off boundaries between different logical sections of the abstracts. However, the tool employs a highly problematic scheme for automatically extracting important parts of very long shots based solely on audio characteristics, and clips are chosen for inclusion in an abstract largely through random selection rather than with any higher level sense of cinematic meaning.

The Media Lab has been active in this domain for several years as well, especially within the Interactive Cinema group. In *Agent Stories*, a tool devised by Kevin Brooks, an author creates an annotated web of video story snippets from which programmatic story "agents" will construct and present complete stories [Bro99]. These agents, developed by Brooks, are each skilled at assembling a story a different way. *Contour* and *Dexter*, developed by Michael Murtaugh, together form a system that employs a spreading activation network to guide content selections in an interactive video program such as a documentary [Mur96]. The system selects segments from a large annotated database of video material in a fashion that's relevant to the interests of the viewer as reflected by his previous choices of content during the program. Gilberte Houbart's *Viewpoints on Demand* utilizes a semantic net representation of support relationships between opinions expressed in video clips and allows viewers to adjust "content knobs" to control the way an argument will presented about a particular topic [Hou94]. Lee Morgenroth's *Homer* allows an editor to build a story model from some simple building blocks and have the system select content from a logged database of video and generate a rough cut of a video story, acting much like an editor's assistant [Mor92].

Other systems are specifically geared for automatically generating multimedia presentations consisting of speech, written text, and graphics. Many such systems, such as that developed by Dalal et. al. [Dal96], use constraint satisfaction mechanisms to determine the sequence and duration of media elements, typically for presenting highly structured information, but it is unclear how these approaches would translate to more general forms of storytelling or entertainment.

Overall, there are three main observations to note about all of the works mentioned above:

- These systems are not capable of incorporating real-time audience activity or other feedback during a presentation and altering editing immediately. *Terminal Time* and *Contour/Dexter* come the closest, but both only process and incorporate viewer choices at the endpoints of long content segments.

- All of these systems assemble video programs in simple sequences of shots or segments separated largely by cut transitions. There is no sense of multiple tracks or layers on which more complex edits or transitions might be constructed, and it can be difficult to create a compelling narrative with high production values without these elements.

- There is no common framework that underlies all of the editing and annotation models and algorithms employed by these systems, and thereby no way to easily share or combine components of many to make new systems.

The third tool developed in this research, *Viper*, will address all three of these drawbacks.

# The Viper strategy

The third main contribution of this research, *Viper*, addresses this need for a common design framework for creating video programs that have the ability to react and re-edit themselves in response to audience activity, equipment configurations, preferences, or other factors. As with the other tools developed in this research, Viper aims to reflect in is design the three main guiding observations about responsive media highlighted at the beginning of this document.

## Multi-sensory interfaces and materials

In contrast to the approach of the some of the systems described above in which viewers make choices and provide feedback only at the endpoints of segments that form the branches of "multi-linear" video programs, Viper aims to support richer and more demanding forms of responsiveness that require complex sensing systems and the ability to alter editing in a seamless way at the same time a program is being viewed. Whereas the systems described above only support putting together clips end to end on a single logical track with simple cut or fade transitions, Viper enables higher production values by supporting multiple video and audio tracks and providing primitives for assembling complex edits and transitions, such as L-cuts, audio and video inserts, dissolves, graphic overlays, slow motion, and so on, and for finely adjusting these elements in response to real-time response activity.

## Multi-person collaborations and experiences

In an aim to support a wider variety of users, Viper offers a visual interface inspired by Cabbage for preparing video material and making custom annotations that is familiar and comfortable to users of traditional editing tools like Avid or iMovie. To support more effective collaborations, the individual tool components of Viper, described in later sections, are designed in a way that each can operate simultaneously with the others on different networked workstations. And as stated above, Viper aims to support richer forms of sensing in which several viewers or spectators might participate in a responsive video experience at the same time.

## Multi-layered tasks and thinking processes

Two major components of the design process for responsive video programs are the annotation of media clips and the development of rules to choose and assemble those clips into complete programs. Although these tasks are quite different in nature, the success of an application is highly dependent on how well the results of one component serve the needs of the other. Viper supports a direct hands-on approach in which designers can shift easily back and forth between these two task layers to achieve the best results.

Perhaps most importantly, Viper addresses the need for a common foundation for developing many different kinds of responsive video applications. In contrast to Cabbage and the other video tools described above, Viper does not consist of a predetermined knowledge base or domain-specific editing or adaptation model that controls how the system assembles its output. In a strategy similar to that of Isis, Viper aims to provide a common set of extensible primitives from which designers can build these models themselves to suit the needs of particular applications. Because these models are based within a common framework, components from one project can be shared and incorporated into other projects more easily, and the authors of different projects will be able to understand and learn from each other's systems more easily as well.

The following sections present more detail on the design of Viper, beginning with a description of the process of creating a production with Viper, followed by a discussion of an extended project example.

# The Viper production process

There are five main parts to the process of building a responsive video program with Viper: planning the production, capturing media, preparing and annotating clips, building editing guidelines, and presenting the result. Each of these task areas is described in detail below.

## Planning a production

The first step in creating a production with Viper is for the author to simply sit back and think about the production and draw up initial answers to these questions:

- What variables about the viewer or the environment will the program respond to?

- How will these response factors be sensed or gathered?

- How can these factors be represented numerically or textually in the simplest possible fashion?

- What is the general structure of the video program?

- What types of editing alterations will the program exhibit?

- What material needs to be shot or gathered to cover all of these possible alterations?

- How can this material be produced in the most efficient way?

## Capturing media

In a Viper production, not every piece of footage gathered will appear in every final version of the program, and so there is typically more shooting to be done to cover all of the possible editing variations the program will exhibit. This suggests a shooting strategy that de-emphasizes concentrating on the absolute perfection of individual shots and favors capturing the widest possible variety of imagery. At the same time, it is probably more important to have a clear plan of

what must be captured, to avoid spending extra time redoing setups and shooting additional scenes when something is forgotten.

In many situations, it may not take much additional time to produce any extra footage as it may involve fairly simple variations on a theme or shooting setup. For example, a responsive advertisement for a restaurant might feature imagery of specific dishes that are likely to appeal to the viewer. There might be 20 different plates that need to be highlighted in different situations, but all of them might be shot quickly with very similar camera and studio setups.

Once the video footage is produced, it must be digitized into the computer. The current version of Viper handles video and audio in the Isis movie format in which video frames are stored as JPEG images and audio is stored in raw PCM format. Viper includes a video digitizer that can capture media directly into this format, and also a program that will convert MJPEG AVI movies to the Isis movie format. All the video material for one production must be captured at the same resolution and with the same audio characteristics (sample rate, number of bits, and so on).

The organization of data in the Isis movie format supports extremely efficient random access to individual video frames and audio samples, which greatly increases the performance of the playback component described later. Future versions of Viper may support other movie formats that have similar features.

The user invokes a command to create a special disk folder to hold all of the data associated with a particular Viper project. The user edits certain text files within this folder to specify the locations of the digitized source material along with other information about the production needed by the tool. A newly created project folder contains documentation and templates that describe what information is needed and where it should be placed. A future version of Viper could include a friendlier graphical interface for these tasks.

## Forming a database of clips

The third part of the Viper production process involves splitting the captured video and audio source material into individual clip objects and annotating those clips with information about their content and function in the larger scheme of the production. The database of media objects created in this stage is accessed when later writing the editing guidelines for a production that express how clips will be chosen from this database and assembled into a complete program.

For this task area, Viper provides a graphical interface based on the same widgets that were successful in Cabbage. This interface aims to incorporate elements that are likely to be familiar to users of traditional computer-based editing systems.
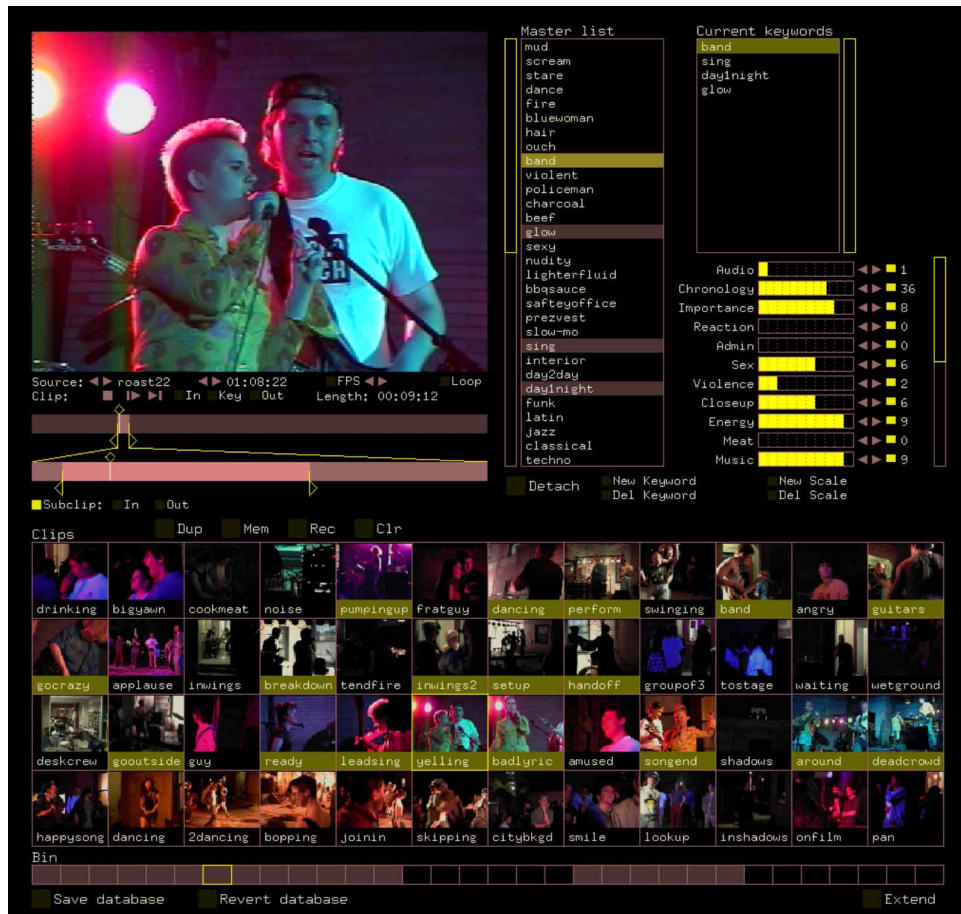
The bottom part of the tool is the clip database, which is initially empty. To begin making a new clip, the user can either click an empty box on the grid or copy the information of an existing clip with the `Mem` and `Rec` buttons or the `Dup` button. Additional grid pages may be added by clicking the `Extend` button. The `Save` button writes the database to disk, and `Revert` recalls the last stored version of the database.

The upper left portion of the tool provides an interface for selecting the source movie of each clip and for setting in and out points in a fairly traditional way. There are facilities for changing frame rate and for selecting a descriptive "key frame" of the clip, which is shown in miniature in its database grid box. There are two long rectangular bars, the upper of which represents the entire source movie, which may be several minutes in length. The lower bar represents the actual clip as a portion of this source movie between the currently selected in and out points.

The highlighted portion of the lower bar represents the "critical section" of the clip, which at first will include the entire clip. If the clip can stand being shortened from its full length and still preserve its meaning, the user can specify a smaller critical section to indicate a limit to how much the clip may be trimmed. When writing the editing guidelines later, the producer may select different lengths of the clip in order to fit a certain time slice, or perhaps to alter the pacing of a program. If a clip contains dialogue, typically the critical section will include all of the dialogue and helps to indicate how loosely or tightly the system may cut around that dialogue.

The user may browse through the video represented by these two bars by clicking and dragging within them. Clicking on the video frame itself starts playback from the currently selected point. If "loop" mode is engaged by clicking the `Loop` button, the clip is played repeatedly from beginning to end, allowing changes to in and out points to be tested more rapidly.

In a Viper production, it is important to realize that depending on how the editing model is written later, a single clip may be placed in sequence with many different clips in different versions of the final program, and it may be difficult to predict exactly what kinds of juxtapositions will be generated. This suggests a clip preparation strategy that is not overly focussed on perfecting the exact in and out points of each clip with respect to others it may appear next to, as is done in a traditional linear editing process. Rather, the author must emphasize building a database of flexible clips that may be used and sequenced in different ways, and tailor them only as much as necessary to support the types of editing alterations that will be devised later.

## Making annotations

In the process of building clip objects, the producer must add annotations that express something about the clip's content or what its intended use is for the program. Viper supports making two simple kinds of annotations, keywords and numerical ratings, in the upper right portion of the graphical interface. Other kinds of custom annotations, such as indications of relationships between clips, may be added in a text file and loaded into the system as well. New potential keywords and scale factors may be created or deleted as needed by clicking the appropriate buttons. Pressing the `Attach/Detach` button adds or removes the selected keyword from the list of annotations for the current clip. For scale factors, clicking directly on the slider adds the annotation to the current clip, and clicking on the box to the right of the slider will remove it.

In contrast to generic and exhaustive approaches to media annotation, such as that employed by Media Streams [DM95], Viper supports a very direct and manual system that allows the author to develop a concise set of potential annotations that are specific to the production at hand and most likely to be needed to support the editing model that will be constructed later. For example, if the plan is to create an educational program that will include more or less detail on a topic in different situations, the author may choose to rate clips on an "importance" level and later use this annotation to decide which clips to include in the final program. Or, in the process of annotating home movie material, the author of a responsive video family scrapbook may choose to add keywords that indicate which family members appear in each clip. The MPEG-7 standard outlines a format in which Viper annotations might be stored along with each clip [NL99], although the current version of the tool does not use the MPEG-7 format.

Although the most useful kinds of annotations are often too content-specific or subjective to be sensed by automatic means, a future version of Viper could support such systems. Techniques like those developed by Nuno Vasconcelos [Vas00] and others can identify textures and tones in video scenes, or even detect the presence of particular objects. The characteristics of actors, including their identities, could be determined automatically using several different systems, such as neural-network face detection [RBK98] or the Eigenfaces face recognition approach created at the Media Lab [MWP98]. Facial expressions or audio characteristics could be analyzed to detect the emotional qualities of clips [Pic00].

## Building editing guidelines

The next and most important step in making a Viper production is to build the editing model that expresses which clips will be chosen from the database and how they will be combined to create a complete video program in response to the chosen factors. For this task, Viper provides a set of primitives based in the Isis programming language that are designed to insulate the author as much as possible from the details of general-purpose programming while at the same time maintaining a high level of extensibility and flexibility in how mappings may be created. The author may incorporate standard Isis syntax in his guidelines if a more complex manipulation is needed. The goal of these primitives is to be able to support any of the editing models and algorithms employed by the other video tools described at the beginning of this chapter.

In contrast to Morgenroth's *Homer*, which was intended mainly as an editor's assistant in making a rough selection and ordering of content based on a story model, Viper integrates the process of building a story model with the intricate details of how the bits and pieces of that story will be rendered in editing with transitions and effects, extending the model's scope all the way to delivery. Viper also includes the ability to finely vary the characteristics of the components and operations expressed in the model based on possibly real-time variables about the viewer and other response factors. In addition, this component of Viper is designed to run in parallel with both the annotation tool and the playback system in order to better support an incremental design process in which clips or annotations may be changed or added and editing guidelines may be tested and refined as needed.

In this segment of the tool, the author must declare exactly what context factors the program will be responding to, including the name and type of each variable and its range of possible values. This information is used to generate a graphical user interface that provides a manual control over the values of each of the variables. This interface currently supports variable types of integer, string, and list of strings, although more may be added later. For example, a viewer's age might be specified with an integer variable in a range from 0 to 100, and his favorite genres of music might be given as a list of strings taken from a list of 15 possibilities.

Here is an example of a manual control panel displayed by Viper:



Once they are declared, the job is to use the values of these variables along with the Viper editing primitives to guide the selection of the appropriate clips from the database and their assembly into a complete program. This process happens in a text Isis program, upon entry to which all of the response variables (typically named with the prefix `rv-`) have been set to the values given in the manual controller, and the variable `clip-database` holds the entire database of media objects formed in the previous stage. By the end of the file, the user must set the variable `final-movie` to be the final edit decision list for the video program. The primitives provided for this task are described below and come in four flavors: database lookup, scoring and sorting, preparation, and editing. For the sake of brevity in the following sections, full documentation on each primitive is omitted.

## Database lookup primitives

These primitives allow the author to select a particular subset of clips from the database that satisfy certain properties or constraints. Essentially, this is how the author will choose the story elements that will appear in the final program. Any of the response variables defined in the production, as described above, may be used to guide the details of how these selections are made.

Below, `tag` represents the name of a keyword or numerical annotation, and `cd` represents the database (or list) of clips that will be narrowed by the primitive. Each primitive returns a new list of clips that satisfy the specified conditions, except for `select-head` which returns just a single clip. Primitives may be cas-

caded to express a more complex selection metric. The standard Isis list manipulation primitives may be used as well—such as `append` to concatenate two lists of clips together, or `reverse` to reverse the order of the clips in a list.

```
(select-tags tag tag … cd)
```
Select all clips that have at least one of the given annotations

```
(select-tags-all tag tag … cd)
```
Select all clips that have all of the given annotations

```
(select-tag tag cd)
```
Select all clips that have the given annotation

```
(select-val tag val cd)
```
Select all clips in which the given annotation has the given value

```
(select-ineq tag eqproc val cd)
```
Select all clips in which the specified inequality holds for the given annotation and value

```
(select-name name cd)
```
Select all clips that have the specified name

```
(select-first num cd)
```
Select the first N clips in the database or list

```
(select-first-tagsum tag initial target cd)
```
Select the first N clips such that the values of the given annotation sum to at least the given target value

```
(select-random num cd)
```
Select a certain number of clips at random

```
(select-head cd)
```
Select the first clip in the list

```
(select-proc sproc cd)
```
Select all clips for which the given procedure tests `True` when invoked on the clip

For example, the following chooses all the clips that have the keyword "jane" attached to them:

```
(set janeClips (select-tag "jane" clip-database))
```

This example selects out all the clips that have both the keyword "scenery" and the keyword stored in a response variable named `rv-city`, and also have an "importance" level greater than the value of a numerical response variable named `rv-threshold`:

```
(set cityClips (select-tags-all "scenery" rv-city clip-database)))
(set cityClips (select-ineq "importance" > rv-threshold cityClips))
```

## Scoring and sorting primitives

These primitives allow the author to control aspects of how the selected story elements will be ordered or arranged in other ways in the final program. Many times it will be necessary to sort clips based on the value of one or more annotations. For example, the author may need to sort a collection of clips in decreasing order of "importance," select a certain number of clips from the head of the resulting list to include in the final program, and then sort those clips in increasing order of a "chronology" annotation. The scoring primitives allow the author to add *new* annotations to each clip in a collection that express a certain metric, perhaps to sort by at a later time.

`(sort-tag-increasing tag cd)`

Sort clips in increasing order of the value of the given annotation

`(sort-tag-decreasing tag cd)`

Sort clips in decreasing order of the value of the given annotation

`(sort-tag-nearest tag val cd)`

Sort clips in order of the distance of the given annotation from the given value

`(sort-random tag cd)`

Sort clips in a random order

`(sort-proc sproc cd)`

Sort clips in a customized way based on the given procedure

`(score-tags newtag tags cd)`

Add a new numerical annotation that expresses how many of the given annotations are attached to each clip

`(score-length newtag trimval cd)`

Add a new numerical annotation that contains the duration of each clip, if trimmed the given amount (between 0.0 and 1.0)

`(score-proc newtag sproc cd)`

Add a new annotation to all clips based on the return value of the given procedure when invoked on each clip

`(clip-duration clip)`

Returns the duration of a clip in seconds

For example, the following sorts the selected Boston scenery clips from above in decreasing order of importance:

```
(set bosClips (sort-tag-decreasing "importance" bosClips))
```

The following selects the 4 Jane clips that are nearest to 5 seconds long:

```
(score-length "length" 0.0 janeClips)
(set janeClips (sort-tag-nearest "length" 5.0 janeClips))
(set janeClips (select-first 4 janeClips))
```

## Preparation primitives

Clips selected directly from the database are not "playable" until they have been converted into an edit decision list, or *movie* for short, using one of the following preparation primitives. Conceptually, these primitives represent the transformation of simple story elements into the details of how those elements will be rendered in the final program. It is at this point that the author can request that a clip should be shortened from its full size by a certain amount.

Each of these primitives returns a playable movie consisting of a single shot starting at time 0. The editing primitives described later allow these single-clip movies to be sequenced and layered on separate tracks to create more complex movies.

`(prepare-clip clip)`

> Make a movie from the specified clip

`(prepare-clip-time clip time)`

> Make a movie from the specified clip, trimmed to given duration, centered around the clip's critical section, as specified in the annotation tool

`(prepare-clip-trim clip trimval)`

> Make a movie from the specified clip, trimmed the given amount from 0.0 to 1.0, where 0.0 represents no trim, and 1.0 represents a trim that consists only of the clip's critical section

## Editing primitives

The single-clip movies created by the above primitives are fully playable by themselves. The author must use the following editing primitives to create movies that consist of more than one clip. These primitives provide more than just the ability to put clips together in a simple sequence. Viper supports multiple video and audio tracks, allowing the author to create complex edits, such as inserts and L-cuts. Track numbering begins at 1. Material on higher-numbered tracks overrules that on lower-numbered tracks. Fades provide the ability to control the visibility of each track. Transition and mixing primitives allow multiple video or audio elements to be mixed and dissolved together in differing amounts to create highly customized effects. The author may use any of the re-

sponse variables defined for the production to alter the parameters of any of these elements.

Each of the following primitives acts on *movies* and returns a *movie*. Again, operations may be cascaded to perform more intricate editing maneuvers. The author sets the variable `final-movie` to be the final version of the movie that he wishes to present.

`(video-only movie)`

    Extract only the video elements of a movie

`(audio-only movie)`

    Extract only the audio elements of a movie

`(shift-start time movie)`

    Shift the starting time of all elements in a movie by a given amount

`(change-start time movie)`

    Change the starting time of all elements in a movie

`(set-level level movie)`

    Set the video or audio level of all elements in the movie

`(ramp-in ramptime movie)`

    Place a ramp-in (dissolve) transition at the beginning of every element in a movie in which the level of each element will increase from 0.0 to 1.0 over the given duration

`(ramp-out ramptime movie)`

    Place a ramp-out transition at the end of every element in a movie

`(ramp-in-out ramptime movie)`

    Place both ramp-in and ramp-out transitions in every element

`(change-track tracknum movie)`

    Change the track number of all elements in a movie

`(isolate-track tracknum movie)`

    Extract only the elements on the given track in a movie

`(remove-track tracknum movie)`

    Remove all the elements on the given track in a movie

`(fade-in fadetime movie)`

    Place a fade-in transition at the beginning of a movie

`(fade-out fadetime movie)`

    Place a fade-out transition at the end of a movie

`(fade-only movie)`

    Extract only the fade elements in a movie

```
(sequence-movies spacing movies)
```

    Sequence the given movies such that each starts after the next in order, spaced by the given amount (which may be negative)

```
(combine-movies movie movie …)
```

    Combine the given movies into a single movie, without shifting the starting time of each movie

```
(movie-duration movie)
```

    Returns the duration of the given movie in seconds

There are additional primitives for creating graphics, such as title screens and credits, and for including them in a movie like any other audio or video element. For the purposes of brevity, details of these primitives are omitted here.

## Inspecting and refining

To assist in the task of building the editing guidelines for a program, Viper provides a way to view the edit decision lists generated by the system in a graphical form. These visual renderings resemble the program timelines found in traditional editing tools such as Final Cut Pro and Avid. Each audio, video, and graphic element appears as a colored rectangle—blue for audio, red for video, and orange for graphic. The position of the left edge of each rectangle indicates the starting time of the element, and its horizontal size indicates its duration. Transparency at the left or right edges of an element indicates a ramp or dissolve transition. Multiple tracks are separated by a thin horizontal line, with higher-numbered tracks appearing above lower-numbered tracks. A green bar running underneath the elements on a track indicates where the track is enabled and shows the position of fade-in and fade-out transitions. When a track is enabled, the video material on it overrules that on any track below it.
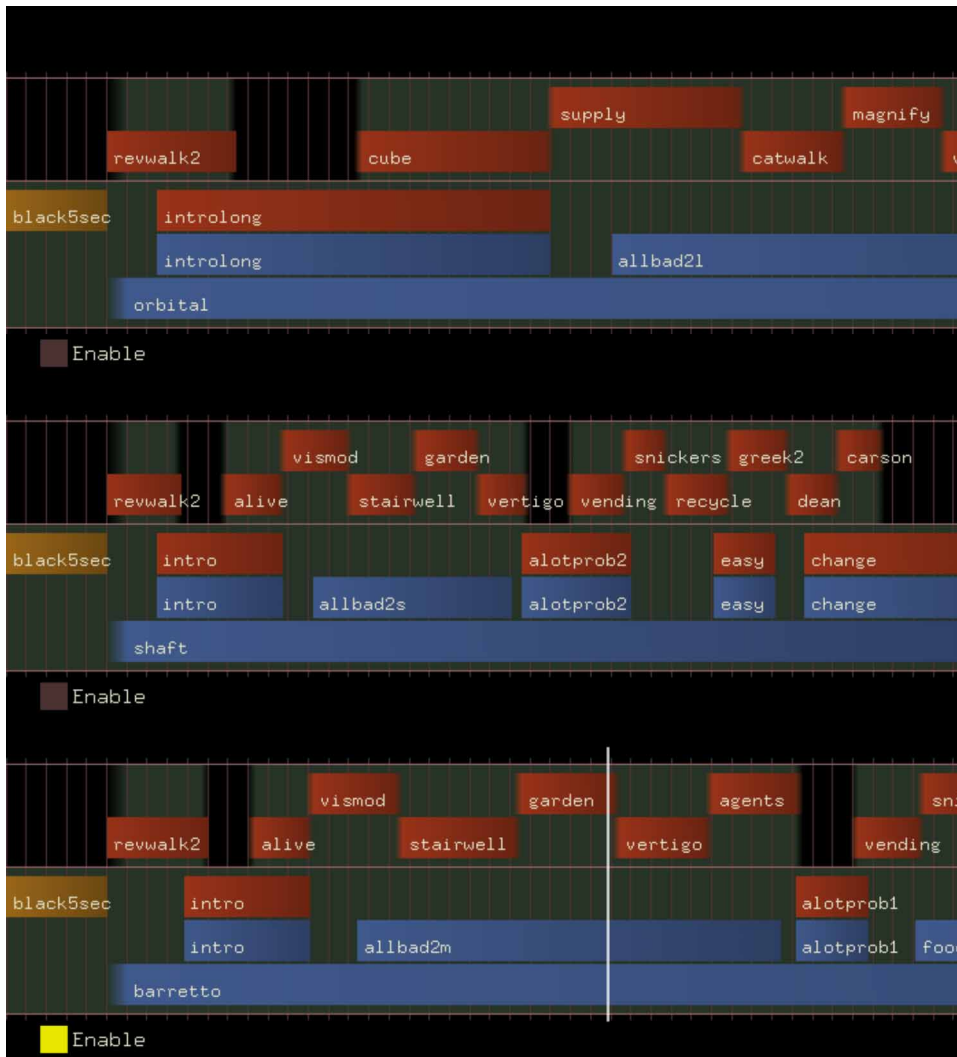
To facilitate comparisons between different generated versions of the program, this interface has three sections, each of which can display an entire edit decision log. To view an edit decision list (EDL), the author selects one of the three sections, enters values for the response variables, and finally clicks the `Edit` button to generate the program and display it in the viewer. The author may scroll horizontally and increase or decrease the time scale of the viewer in order to zoom in on particular segments or gain a more global view of the program.

The EDL viewer runs in cooperation with another important Viper component, the movie player. This movie player, the same as will be used later to deliver the final presentation, is capable of rendering the movie full screen at full frame rate on a moderate Unix workstation, assembling each audio and video frame from its constituent media clips and objects in real time. It also supports writing the movie to a disk file.

Clicking the `Play` button in the control panel will present the current version of the movie in the player. A white vertical bar travels slowly across the edit decision viewer indicating the current play point. The user can click on the EDL viewer to pause and jump to different point, and then click `Play` again to start playback from that point. Dragging on the viewer allows the user to "scrub" across the movie, to inspect a transition more closely for example.

96

Below are a few very simplified examples of the editing primitives in action, each accompanied by an example of what might be rendered by the edit decision list viewer in each case. A longer and more complex example will be presented later in this chapter, in the context of a full production created with Viper.

The following creates a movie that consists of the four selected Jane clips from above in simple sequence:

```
(set janeMovies (map video-only (map prepare-clip janeClips)))
(set final-movie (sequence-movies 0.0 janeMovies))
```
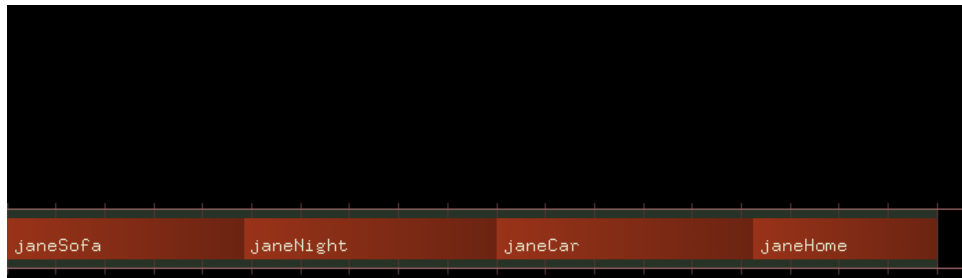


This example demonstrates how to make those clips dissolve into and out of each other instead of simply cut from one to the next. The length of these dissolves is controlled by a response variable named rv-abruptness that, for the purposes of this example, holds an integer from 0 to 10.

```
(set dislen (* 0.1 (- 10 rv-abruptness)))
(set janeMovies (map video-only (map prepare-clip janeClips)))
(set janeMovies (map (proc (movie) (ramp-in-out dislen movie)) janeMovies))
(set final-movie (sequence-movies (* -1.0 dislen) janeMovies))
```
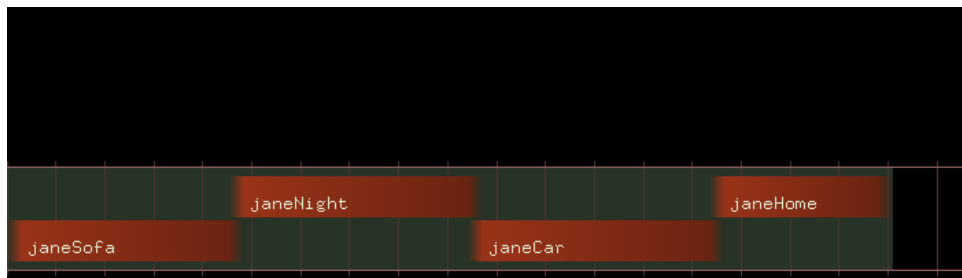
The following adds an audio track to the movie based on a response variable rv-
genre that perhaps indicates the viewer's favorite kind of music, assuming the
possible values of the variable will correspond to the keyword annotations made
in the "music" clips:

```
(set musicClips (select-tags-all "music" rv-genre clip-database))
(set musicClip (select-head musicClips))
(set runtime (movie-duration final-movie))
(set musicMovie (audio-only (prepare-clip-time musicClip runtime)))
(set musicMovie (ramp-in-out 0.3 musicMovie))
(set final-movie (combine-movies musicMovie final-movie))
```
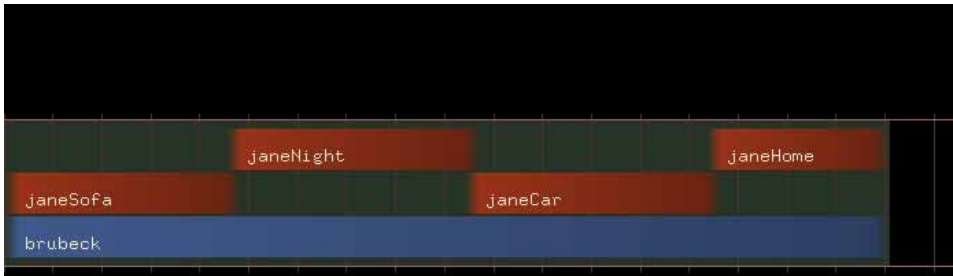


The following performs an insert edit in which an "interview" clip will fade in
and play starting at halfway through the second Jane clip:

```
(set insertClip (select-head (select-name "interview" clip-database)))
(set insertMovie (fade-out 0.5 (fade-in 0.5 (prepare-clip insertClip))))
(set startPoint (+ (clip-duration (janeClips 0))
                   (* 0.5 (clip-duration (janeClips 1)))))
(set insertMovie (change-track 2 (shift-start startPoint insertMovie)))
(set final-movie (combine-movies insertMovie final-movie))
```
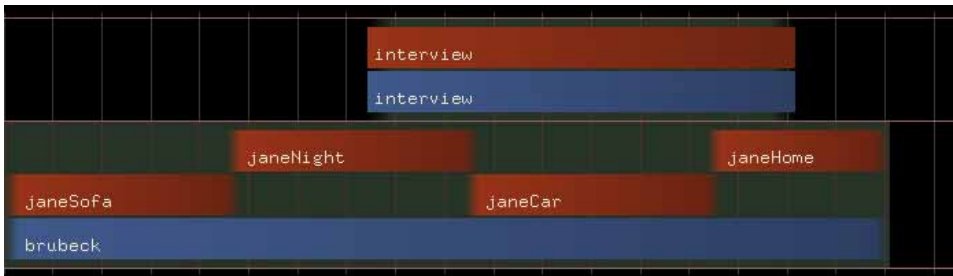


## Presenting the result

Once the author has completed building the editing guidelines and manually
tested the responsiveness of the program to his satisfaction, he is ready to con-
nect the system to the actual sensors or sources that will be used to gather the
response data. The author may choose to write a completely external computer
program that collects the desired response data and executes the Viper playback
program on that data. Alternatively, he may build the data collection component
with the utilities available in Isis and invoke the Viper program generator and
playback routines directly.

The author must also decide how real-time changes to the response variables will
be handled by the system, if such changes are possible. In every case, a change

to the state of the variables will cause the movie to be re-edited, but the author may choose one of several options for how the currently playing movie will transition into the new version, some of which involve named "checkpoints" that the author may add into the edit decision list to indicate appropriate transition points:

- Interrupt the current movie immediately

- Interrupt the current movie at a checkpoint

- Wait until the end of the current movie

- Enter the new movie at its beginning

- Enter the new movie at a checkpoint

- Enter the new movie at a particular time offset

Viper does not assume a particular kind of distribution channel. It targets scenarios in which the receiver is responsible for performing the final assembly of a program, but it will also support assembling personalized programs at a remote server and streaming them to the receiver in a traditional linear fashion.

# Experience

Perhaps the best way to understand and evaluate Viper is to see how it was used to create a real production. To date, two major productions have been completed, a third is nearing completion, and two more are in the planning stage. For the purposes of illustration, this section describes two of these projects in brief and then expounds upon a third in great detail.

## Steer Roast documentary







The first production, created with an early version of Viper, is a responsive documentary about the 2000 Steer Roast festival at Senior House at MIT, which features mud wrestling, live bands, and copious amounts of burning meat. The presentation device consists of "sex" and "violence" knobs that the viewer can turn to explore alternative ways of portraying and understanding the meaning of the event. There is also a "police activity" knob that allows the viewer to increase or decrease the perceived police presence at the event, and a "length" knob that controls duration of the program. The straightforward editing model developed for the production chooses clips for inclusion in the program based on sex, violence, and police presence ratings made on each in the annotation stage, as well as an importance rating that helps the system to omit passages that are less critical to show in shorter versions of the documentary.

## Personalized research open-house
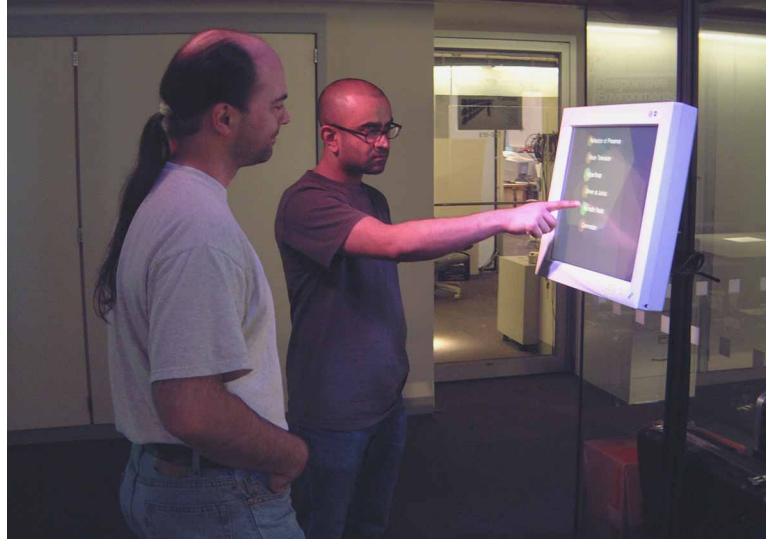


The most recent application created with Viper, still in development at the time of this writing, is an interactive kiosk that presents a personalized research "open house" for visitors to the Garden of the Media Lab. The viewer uses a touch-screen to select theme areas that he is interested in hearing about and an approximate duration for the program. The media database consists of imagery of various projects in the Garden over the past several years, as well as description segments with narration that are used partly as voiceovers. The editing model creates a survey of the research in the Garden that includes projects that are likely to appeal to the viewer based on his selections. More time is spent on projects that are a close match to the viewer's choices, and less time on projects that are only partially related. The editing guidelines emphasize the construction of an engaging program with a coherent thread that includes any introductory material about the Garden or specific groups that might be relevant to the projects that will be presented.

## Other projects

There are a few other projects envisioned for Viper at this point, including a cooking show that tailors its pace and level of detail in teaching the viewer how to make a homemade pizza, and a responsive video art installation in which real-time spectator movements affect the activities of a video character. In addition, one other major production has already been completed. This production, a responsive campaign advertisement that adapts to the preferences and concerns of the viewer to present the candidate in the most appealing manner, is discussed in detail below.

# A responsive campaign advertisement

Among the most prominent media phenomena of the year 2000 were the campaigns of Al Gore and George W. Bush for the U.S. presidency, and television advertising was a central method these and other candidates used to reach out and deliver their messages to voters. Every week would bring a new wave of advertisements, some critical of the opponent, some bragging about political

achievements, some introducing the family of the candidate, and so on. For the most part, these advertisements were hit-and-miss with the public, as each individual voter has different concerns about the country and about the two candidates, as well as different personal preferences and tastes that would render some of the messages more or less memorable and appealing. The one-size-fits-all advertisements simply could not cater to every expectation.

This raised the notion of creating a responsive political campaign advertisement—one that can adapt to the viewer's personal profile and concerns in order to "sell" the candidate in the most appealing and effective way for each individual. In the case of this production, the mock candidate is not running for the U.S. presidency, but rather for a position on the student committee at the Media Lab. The goal of the advertisement is to tailor the message of the candidate, played by Aisling Kelliher, based on the following response factors:

- The job position of the viewer:
  student, faculty, administration, sponsor

- The floor of the building the viewer normally inhabits:
  1st, 2nd, 3rd, 4th

- The general mindset of the viewer:
  negative, positive

- A list of concerns the viewer has about the Media Lab, one or more of:
  equipment, cleanliness, safety, space, food,
  facilities, classes, stress, distractions

- The genre of music the viewer finds most uplifting:
  jazz, classical, latin, funk, techno

- The viewer's attention span:
  short, medium, long

If the identity of the viewer is known, many of these variables might be gathered automatically, by consulting online databases for personal information and perhaps inferring his main concerns about the lab by checking if he is a member of certain mailing lists. In the current setup, the viewer simply fills out a brief checkbox form before watching the program.

The general structure of the ad that was envisioned and later modeled in Viper with respect to these response factors is as follows:

*Introduction*

Background music begins, the genre of which is controlled by the viewer's music preference. Fade in to an uplifting introductory image of the candidate, which is the same for any version of the program.

The candidate introduces herself, as if being interviewed. The length of her introduction is varied based on the viewer's attention span.

*Build relationship with viewer*

In a video insert, we see a series of clips of the candidate walking around in different parts of the lab, concentrating on areas that are on the same home floor as the viewer. Presented slightly in slow-motion, the pacing of these clips is increased or decreased based on the viewer's attention span, and only as many clips are included to cover the duration of the candidate's remarks.

During this insert, the candidate makes a statement about the general condition of the lab, tailored to appeal to the viewer's mindset.

*Address specific issues*

We see the candidate in the interview again, stating that there are a lot of problems at the lab that need to be addressed, and a second insert begins with a series of descriptive clips that illustrate different problems. The clips for this segment are chosen to appeal to the concerns the viewer has about the lab, and pacing is again varied to cater to the viewer's attention span.

If the viewer has a medium or long attention span, we hear a voiceover of the candidate making a longer statement about one of the items on the viewer's list of concerns.

*Build confidence in abilities*

The candidate, in the interview again, states why she thinks she is the best person for the job, length controlled by attention span again.

In a third video insert, we see a series of slightly slow-motion clips of the candidate talking to and shaking hands with people in the lab who are on the same floor and who have the same job position as the viewer. Pacing is again metered by attention span.

*Closing*

The candidate makes a brief final statement that caters to the viewer's mindset.

Fade slowly into a final uplifting image of the candidate emerging from the Media Lab looking confident and popular, the same image for every version of the advertisement. Fade to black.

Shooting for the advertisement was surprisingly straightforward. A single afternoon was needed to go from floor to floor and capture imagery of Aisling walking around and inspecting different parts of the building, shaking hands with as many people as possible in each area, and pointing out problem areas in a few entertaining vignettes. On a second day, a mock interview was set up in which Aisling made all of the statements that are used in different parts and in different versions of the program.

The annotations made on each clip make it easy to select out the most appropriate clips to include in each section of the advertisement. Each imagery clip is classified with keywords to identify the floor of the building it was shot on, the kind of people in the shot (students, professors, ...), the concern illustrated in the clip (equipment, food, ...), and the general function of the clip (a walking shot, a handshake shot, a lab concern shot, ...). Each clip includes a critical section that is

approximately half of its full length in order to enable the editing model to shorten it as necessary in adjusting the pacing of the program. The interview clips are annotated based on the part of the advertisement for which they are intended and on the mindset and attention span to which they cater. Various selections of background music are included in the database as well and classified by genre.

The resulting advertisements generated by Viper are surprisingly effective and entertaining. Below are a three storyboards that attempt to illustrate, in the absence of video, the progression of the program for three different viewer profiles:

## Viewer profile 1

- a student

- from the 1st floor

- with a negative mindset

- concerned about food and safety

- likes Latin music

- has a medium attention span

Fade in to opening shot of Aisling walking outside, looking confident and visionary. An energetic upbeat Latin tune plays softly in the background. Voiceover begins, followed by the visual of Aisling in her office: *"I'm Aisling Kelliher and I've just finished my first year here in the Media Lab in the Interactive Cinema group as a graduate student."*



As her introduction finishes, dissolve into shots of Aisling walking around different parts of the lab, all on the first floor (6 shots, 31 seconds total). Voiceover, catering to the viewer's negative mindset: *"Things at the lab couldn't be worse, to be honest, I mean, morale is low, faculty aren't talking to each other, students aren't talking to faculty or getting to know each other at all. I think the projects that are coming out of the lab reflect this, the fact that there's probably work that's been redone... only because students haven't heard about it or they aren't talking to each other."*
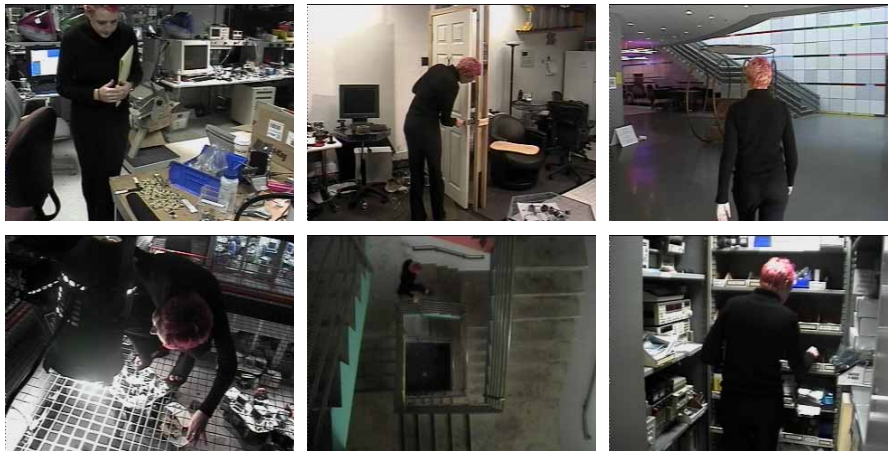
Return to office shot. *"There are currently alot of problems at the lab that really need attention."* Dissolve into a series of shots illustrating problems with food and stress in the lab (7 shots, 30 seconds total). Voiceover: *"I think the variety of food that students in the lab are subjected to is rather intense. I can't believe they expect us to survive on a diet of sea urchins and buffalo and venison—I mean, it's been so long since we've had good honest-to-god plain food in this lab. It's been a long time since I've had some nice lamb stew, and I do be fond of my bit of lamb stew."*
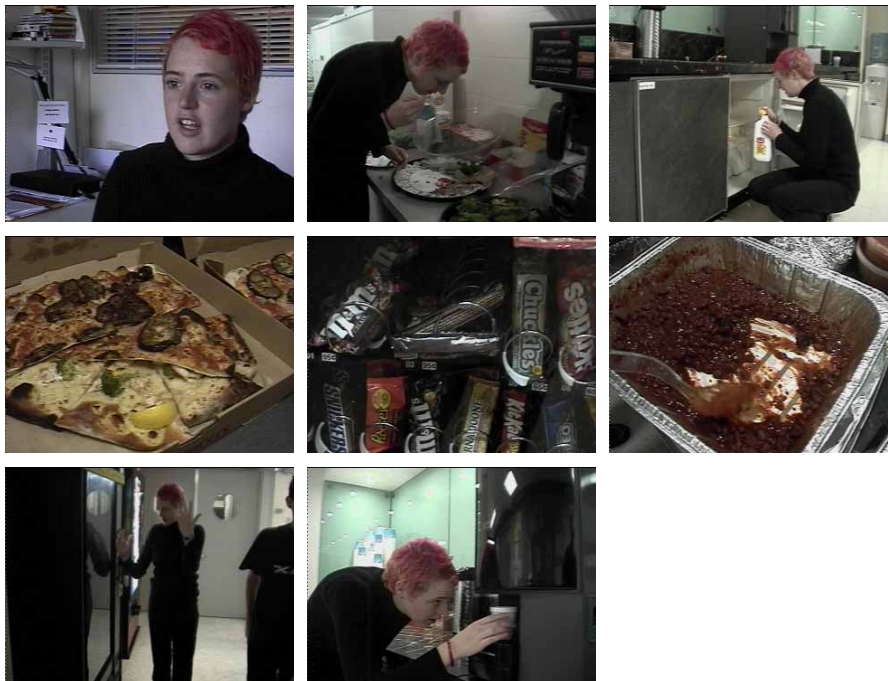


Office interview audio begins again, followed by visual. *"I feel very strongly about this issue and I think it's going to be a major part of me being on the student committee."* Dissolve into several shots of Aisling shaking hands with students from the first floor (4 shots, 16 seconds total). Voiceover: *"Primarily I think I'm quite an approachable person, so if somebody has a problem they shouldn't find it a big hassle to have to come to me and discuss it with me."*
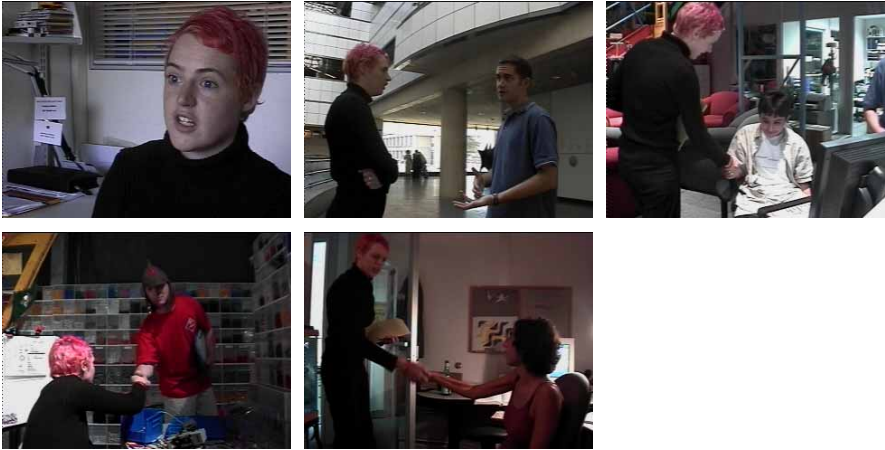
Interview AB roll again, in which Aisling makes a closing statement calling for change. Voiceover continues over final shot of Aisling emerging from the the lab, waiving and looking determined. *"My whole campaign is about change. I think things need to be shaken up, we need to see alot of changes, alot of differences coming out here in the lab and I'm really going to be the person that's going to help implement these and change things for the future of the lab."*



Dissolve to title. Fade to black. Total duration: 101 seconds

# Viewer profile 2

- a faculty member

- from the 3rd floor

- also with a negative mindset

- concerned about equipment and facilities

- likes classical music

- also has a medium attention span

The structure of the program in this case is generally the same as in the first example except that it includes imagery and voiceovers more appropriate for the new viewer.

Fade in to opening shot of Aisling walking outside, looking confident and visionary. An energetic yet soothing Bach piano concerto plays softly in the background. Voiceover begins, followed by the visual of Aisling in her office: *"I'm Aisling Kelliher and I've just finished my first year here in the Media Lab in the Interactive Cinema group as a graduate student."*

As the introduction finishes, dissolve into shots of Aisling walking around different parts of the lab, all on the third floor this time (7 shots, 31 seconds total). Voiceover, same as before: *"Things at the lab couldn't be worse, to be honest, I mean, morale is low, faculty aren't talking to each other, students aren't talking to faculty or getting to know each other at all. I think the projects that are coming out of the lab reflect this, the fact that there's probably work that's been redone... only because students haven't heard about it or they aren't talking to each other."*
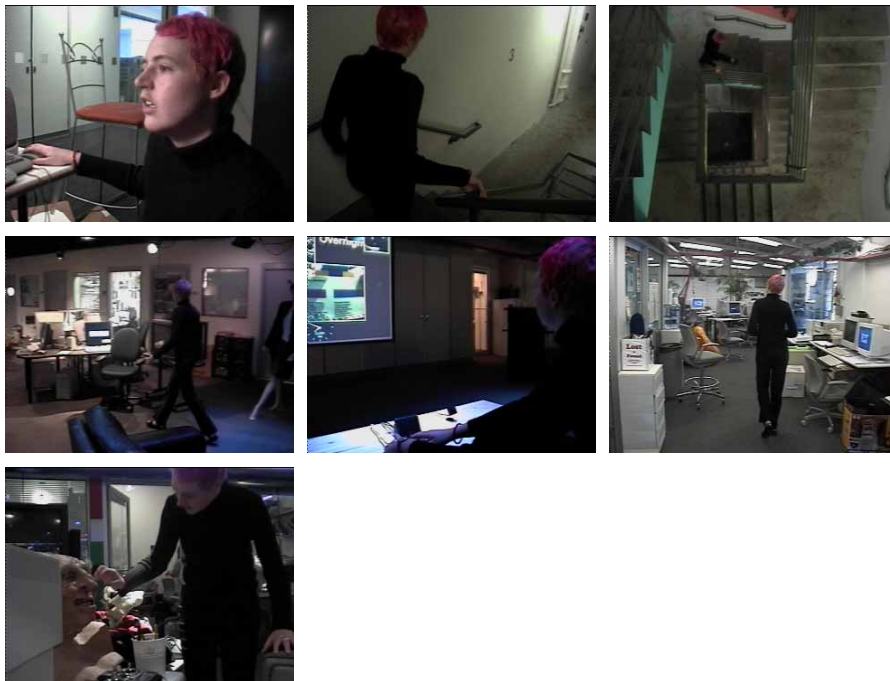


Return to office shot. *"There are currently alot of problems at the lab that really need attention."* Dissolve into a series of shots illustrating problems with equipment and facilities in the lab (7 shots, 32 seconds total). New voiceover: *"I think equipment is a problem in the lab, particularly with small pieces of equipment that need to be regularly reordered and kept in stock. You often get the case where you go to a bin and whatever it is you're looking for is not there. You might go to a LEGO bin, and the right arm of the little lady who sells the tickets or whatever piece ... is missing, and that can completely destroy the whole feeling and aesthetic of the project that you're working on."*
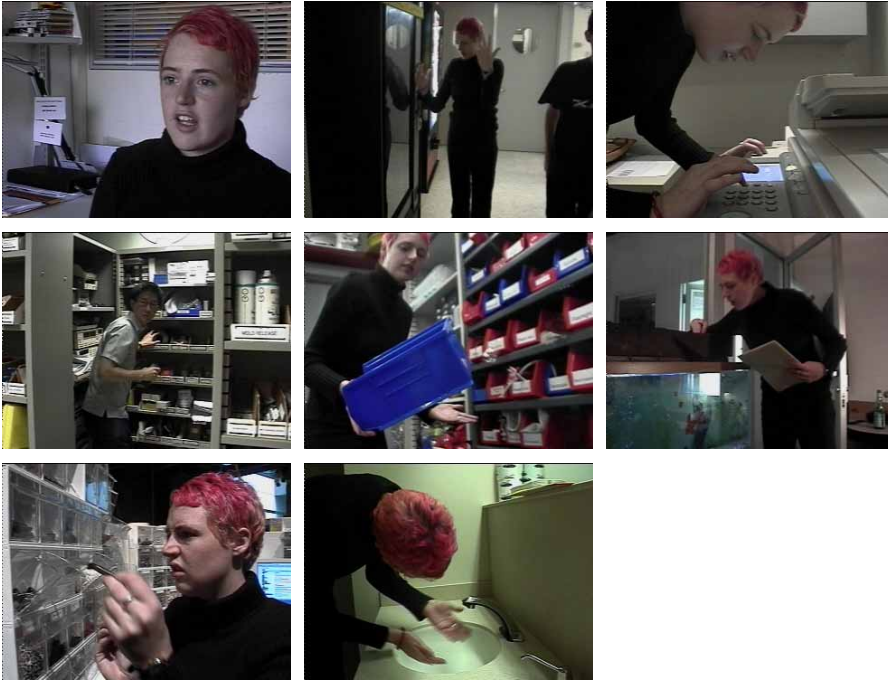
Office interview audio begins again, followed by visual. *"I feel very strongly about this issue and I think it's going to be a major part of me being on the student committee."* Dissolve into several shots of Aisling shaking hands with professors, as many from the 3rd floor as possible (4 shots, 15 seconds total). Same voiceover as before: *"Primarily I think I'm quite an approachable person so if somebody has a problem they shouldn't find it a big hassle to have to come to me and discuss it with me."*



Interview AB roll again, and Aisling makes the same closing statement calling for change. Voiceover continues over final shot of Aisling emerging from the lab waiving again. *"My whole campaign is about change. I think things need to be shaken up, we need to see alot of changes, alot of differences coming out here in the lab and I'm really going to be the person that's going to help implement these and change things for the future of the lab."*

Dissolve to title. Fade to black. Total duration: 104 seconds

## Viewer profile 3

- a faculty member

- from the 3rd floor

- with a negative mindset

- concerned about equipment and facilities

- likes classical music

- with a ** SHORT ** attention span

The following advertisement is for a professor with the same profile as in the second example, but who has a *short* attention span. The selection of imagery is approximately the same, but fewer shots will be shown and their pacing much more swift. Fades and dissolves are shorter, and voiceovers will be shorter as well. The middle section where Aisling speaks about the viewer's concerns in detail is omitted altogether.

Fade in to same opening shot, with same classical background music. Same introductory sequence: *"I'm Aisling Kelliher and I've just finished my first year here in the Media Lab in the Interactive Cinema group as a graduate student."*



As the introduction finishes, dissolve into a faster-paced montage of shots of Aisling walking around the third floor (5 shots, 15 seconds total). The voiceover is shorter: *"Things at the lab couldn't be worse, to be honest, I mean, morale is low, faculty aren't talking to each other, students aren't talking to faculty or getting to know each other at all. "*
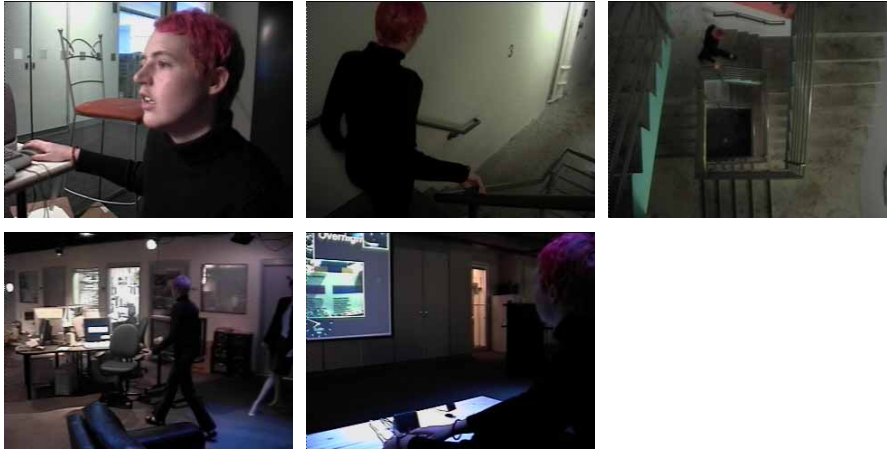
Return to office shot. Slightly longer statement about problems *"It's unfortunate that there are alot of things at the lab that need attention, and I really want to be able to fix these problems."* A moment after this statement begins, we see a quick three shots illustrating equipment problems at the lab (about 9 seconds). There is no detailed statement about these problems.
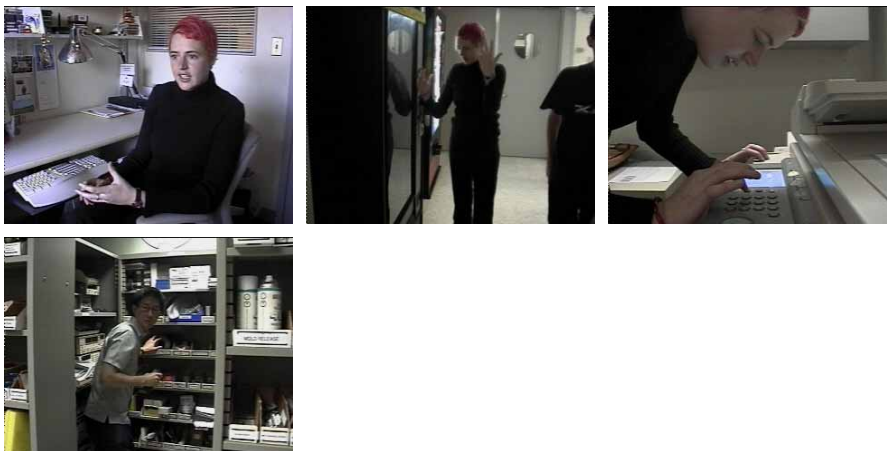


Dissolve directly into a series of shots of Aisling shaking hands with professors again, this time paced more quickly (3 shots, 8 seconds total). Shorter voiceover: *"I think people find it easy to talk to me, if there's a problem I'm quite approachable."*



Same final sequence where Aisling calls for change. Final shot of Aisling emerging from the lab is shorter. *"My whole campaign is about change. I think things need to be shaken up, we need to see alot of changes, alot of differences coming out here in the lab and I'm really going to be the person that's going to help implement these and change things for the future of the lab."*

Dissolve to title. Fade to black. Total duration: 50 seconds

A version of the advertisement for a viewer with a long attention span will have the same general structure as that for a medium attention span but with extended voiceovers and a more relaxed pacing throughout. The database includes enough material to adequately cover all of the other viewer profile settings as well. The system can also generate several slightly different versions of the advertisement for the *same* viewer profile, ensuring that a single viewer will rarely see exactly the same thing twice.

## Edit decision lists

For the purposes of comparison, the following diagram shows the three edit decision lists Viper generated for the three profiles described above. Following this image is a discussion of how the editing model is encoded for this production, and a discussion of the weakness of this production and how it could be improved with additional annotations and a more complex editing model.

**Profile 1:**

title · black5sec · bigdoor · jeana2 · cubekid2 · erik · atrium · change · change · vending · coffee · snickers · beans · milk · pizza · badfood · alotprob1 · alotprobfood1 · strongly qual · strongly qual · supply · catwalk · vertigo · context · comein · physics · intro · allbad2m · barretto · rewalk2 · black5sec · intro

**Profile 2:**

title · black5sec · bigdoor · kristin · mike · mitch · change · change · legos2 · playdoh · copier · fish · thief · vending · atrium · faucet · strongly qual · strongly qual · alotprob1 · alotprobequip1 · agents · garden · pond · icom1 · vertigo · stairwell · alive · intro · allbad2m · bach · rewalk2 · black5sec · intro

**Profile 3:**

title · black5sec · bigdoor · kristin · mike · copier · mitch · vending thief · change · change · easy · easy · alotprob2 · alotprob2 · garden · stairwe · icom1 · vertigo · alive · intro · allbad2s · bach · rewalk2 · black5sec · intro

# Editing guidelines

To illustrate the flexibility and extensibility of the Viper primitives, this section "lifts the hood" and presents the editing guidelines developed for the campaign advertisement production. These guidelines embody the editing model that maps the values of the 6 viewer profile variables to complete final edit decision lists like those shown graphically in the previous image.

It is important to realize that the complexity of a video program generated by Viper is only as rich as both the editing model and the system of annotations created for it by the producer. If one or both of these components is lacking in some respect, a program may not "hang together" as tightly as one would want. There are many such weaknesses in this particular example production that will be discussed later along with ways to alleviate some of the problems by adding additional material into the editing model.

This first section of the model creates some Isis variables that refer directly to the values of the 6 profile variables. The possible values of these variables were carefully chosen to correspond to many of the keyword annotations attached to clips in the media database. The `rv-length` variable represents the viewer's attention span, and consequently the desired length of the advertisement.

```
(set rv-position (current-state "position"))
(set rv-location (current-state "location"))
(set rv-mindset (current-state "mindset"))
(set rv-concerns (current-state "concerns"))
(set rv-music (current-state "music"))
(set rv-length (current-state "length"))
```

Two variables based on the viewer's attention span are created that will later help adjust the pacing of the program. One variable will control how closely clips are trimmed to their critical sections before they are assembled together, and the other will control the approximate speaking pause inserted between separate interview and voiceover segments. A third variable selects one item from the viewer's list of concerns to be the "main" concern that the middle part of the advertisement will emphasize.

```
(set trimval
  (switch rv-length
    ("short" 1.0)
    ("medium" 0.5)
    ("long" 0.0)))

(set pausetime
  (switch rv-length
    ("short" 1.5)
    ("medium" 2.333)
    ("long" 3.0)))

(set main-concern (rv-concerns 0))
```

The media database consists of a variety of interview clips, groups of which can serve as "introductions" or "wrap-ups" and so on, and each group consists of versions that are appropriate for different mindsets and attention spans. The following lines make the selection of the interview and voiceover clips that will be included in the final program.

```
(set interview-clips (select-tag "interview" clip-database))

(set intro-clip
  (select-head (select-tags-all "intro" rv-length interview-clips)))
(set mindset-clip
  (select-head (select-tags-all "howarethings" rv-mindset rv-length interview-clips)))
(set alotofprobs-clip
  (select-head (select-tags-all "alotofproblems" rv-length interview-clips)))
(set solveproblem-clip
  (select-head (select-tags-all "solveproblem" rv-length interview-clips)))
(set qualities-clip
  (select-head (select-tags-all "qualities" rv-length interview-clips)))
(set wrapup-clip
  (select-head (select-tags-all "wrapup" rv-mindset interview-clips)))

(set probvoiceover-clips (select-tag "probvoiceover" interview-clips))
(score-tags "probappeal" [ main-concern main-concern rv-position ] probvoiceover-clips)
(set probvoiceover-clips (sort-tag-decreasing "probappeal" probvoiceover-clips))
(set probvoiceover-clip (select-head probvoiceover-clips))
```

The following few lines select a few miscellaneous clips that will be included in all versions of the movie, such as the opening and closing shots. It also selects the background music to be included, based on the viewer's music taste.

```
(set music-clip (select-head (select-tags-all "music" rv-music clip-database)))
(set revwalk-clip (select-head (select-name "revwalk2" clip-database)))
(set atrium-clip (select-head (select-name "atrium" clip-database)))
(set elevlaugh-clip (select-head (select-name "elevlaugh" clip-database)))
(set bigdoor-clip (select-head (select-name "bigdoor" clip-database)))

(set music-level (/ (real (music-clip "audiolevel")) 10.0))
```

The following section transforms the selected interview and voiceover clips into self-contained movies with the proper transitions, if applicable. These movies will be combined together later.

```
(set intro-movie (prepare-clip intro-clip))
(set mindset-movie (audio-only (prepare-clip mindset-clip)))
(set alotofprobs-movie (prepare-clip alotofprobs-clip))
(set probvoiceover-movie (audio-only (prepare-clip probvoiceover-clip)))
(set solveproblem-movie (prepare-clip solveproblem-clip))
(set qualities-movie (prepare-clip qualities-clip))
(set wrapup-movie (prepare-clip wrapup-clip))

(set mindset-movie (set-level 1.2 mindset-movie))
(set probvoiceover-movie (set-level 1.2 probvoiceover-movie))

(set atrium-movie (prepare-clip-trim atrium-clip trimval))
(set elevlaugh-movie (prepare-clip-trim elevlaugh-clip trimval))

(set revwalk-movie (prepare-clip-trim revwalk-clip trimval))
(set revwalk-movie (fade-in 1.0 revwalk-movie))
(set revwalk-movie (fade-out 0.4 revwalk-movie))
(set revwalk-movie (change-track 2 revwalk-movie))

(set bigdoor-movie (prepare-clip-trim bigdoor-clip trimval))
(set bigdoor-movie (fade-in 0.4 bigdoor-movie))
(set bigdoor-movie (ramp-out pausetime bigdoor-movie))
(set bigdoor-movie (change-track 2 bigdoor-movie))

(set title-movie (prepare-graphic "title" "title" [0 0] (+ 0.5 (* 2.0 pausetime))))
(set title-movie (ramp-in pausetime title-movie))
(set title-movie (fade-out 1.0 title-movie))
(set title-movie (change-track 2 title-movie))

(set black5-movie (prepare-graphic "black5sec" "black" [0 0] 5.0))
```

114

This section calculates the desired duration of the three montage sequences appearing in the middle of the advertisement, based on the lengths of the interview and voiceover movies created above. Differing amounts of extra time, expressed by the `pausetime` variable created above, is added into these calculations to later adjust the pacing between segments.

```
(if (= rv-length "short")
    (begin
      (set walkabout-destime
        (+ (* 0.5 (movie-duration intro-movie)) pausetime
           (movie-duration mindset-movie) (* pausetime 1.333)))
      (set problem-destime
        (+ (* 0.6 (movie-duration alotofprobs-movie)) pausetime 2.0))
      (set handshake-destime
        (+ -2.0 (* 0.666 pausetime) (movie-duration qualities-movie) pausetime 4.0)))
    (begin
      (set walkabout-destime
        (+ (* 0.5 (movie-duration intro-movie)) pausetime
           (movie-duration mindset-movie) (* pausetime 1.333)))
      (set problem-destime
        (+ (* 0.25 (movie-duration alotofprobs-movie)) pausetime
           (movie-duration probvoiceover-movie) pausetime pausetime))
      (set handshake-destime
        (- (+ pausetime (movie-duration qualities-movie) pausetime 4.0)
           (movie-duration atrium-movie)
           (if (= rv-length "long") (movie-duration elevlaugh-movie) 0.0)))))
```

This next section selects the clips to be included in the three montage sequences: the series of clips where Aisling is walking around, the series of clips where she is pointing out problems, and the series where she shakes hands with various people. In each series, clips are favored that correspond with the viewer's profile—if the viewer is a professor, clips of handshakes with faculty members are favored, and if the viewer has chosen classes and distractions as his concerns, then clips illustrating those concerns are favored, with extra emphasis given to the "main concern" as determined above. Just enough clips are chosen to fill the desired durations calculated in the previous section.

```
(set walkabout-clips (sort-random (select-tags-all "walkabout" clip-database)))
(set problem-clips (sort-random (select-tags-all "problem" clip-database)))
(set handshake-clips (sort-random (select-tags-all "handshake" clip-database)))

(score-tags "walkappeal" [ rv-location ] walkabout-clips)
(score-tags "probappeal" (append [ main-concern ] rv-concerns) problem-clips)
(if (= rv-position "student")
    (score-tags "shakeappeal" [ rv-position rv-location rv-location ] handshake-clips)
    (score-tags "shakeappeal" [ rv-position rv-position rv-location ] handshake-clips))

(set walkabout-clips (sort-tag-decreasing "walkappeal" walkabout-clips))
(set problem-clips (sort-tag-decreasing "probappeal" problem-clips))
(set handshake-clips (sort-tag-decreasing "shakeappeal" handshake-clips))

(score-length "clipdur" trimval walkabout-clips)
(score-length "clipdur" trimval problem-clips)
(score-length "clipdur" trimval handshake-clips)

(set walkabout-clips
  (select-first-tagsum "clipdur" 0.0 walkabout-destime walkabout-clips))
(set problem-clips
  (select-first-tagsum "clipdur" 0.0 problem-destime problem-clips))
(set handshake-clips
  (select-first-tagsum "clipdur" 0.0 handshake-destime handshake-clips))
```

This section transforms the three groups of selected clips into self-contained movies. The clips are trimmed by the amount determined previously for the `trimval` variable to increase or decrease pacing based on the viewer's attention span. Fades are inserted, and these movies are placed on a second track so they act as "inserts" that will periodically overrule the interview video on the first track.

```
(set walkabout-movies (map (proc (clip) (prepare-clip-trim clip trimval))
                           walkabout-clips))
(set walkabout-movies (map (proc (movie) (ramp-in-out 0.4 movie)) walkabout-movies))
(set walkabout-movie (sequence-movies -0.4 walkabout-movies))
(set walkabout-duration (movie-duration walkabout-movie))
(set walkabout-movie (fade-in 0.4 walkabout-movie))
(set walkabout-movie (fade-out 0.4 walkabout-movie))
(set walkabout-movie (change-track 2 walkabout-movie))

(set problem-movies (map (proc (clip) (prepare-clip-trim clip trimval))
                         problem-clips))
(set problem-movies (map (proc (movie) (ramp-in-out 0.4 movie)) problem-movies))
(set problem-movie (sequence-movies -0.4 problem-movies))
(set problem-duration (movie-duration problem-movie))
(set problem-movie (fade-in 0.4 problem-movie))
(if (!= rv-length "short") (set problem-movie (fade-out 0.4 problem-movie)))
(set problem-movie (change-track 2 problem-movie))

(set handshake-movies
  (append (switch rv-length
              ("short" [])
              ("medium" [ atrium-movie ])
              ("long" [ atrium-movie elevlaugh-movie ]))
        (map (proc (clip) (prepare-clip-trim clip trimval)) handshake-clips)))
(set handshake-movies (map (proc (movie) (ramp-in-out 0.4 movie)) handshake-movies))
(set handshake-movie (sequence-movies -0.4 handshake-movies))
(set handshake-duration (movie-duration handshake-movie))
(if (!= rv-length "short") (set handshake-movie (fade-in 0.4 handshake-movie)))
(set handshake-movie (fade-out 0.4 handshake-movie))
(set handshake-movie (change-track 2 handshake-movie))
```

Below, the individual self-contained movies created in the previous sections are assembled piece by piece into a complete program. Each movie is shifted in time to a particular point relative to the movies that come before it in the sequence. The music is layered into the mix as well, at a lower volume. At the end, a single call to combine-movies assembles the final movie that is returned from the program. Five seconds of black is inserted at the beginning and end of the movie as a buffer.

The way the movie is assembled is slightly different for different attention spans. The following section shows the assembly for the short attention span, in which the long description of concerns in the lab is omitted and the problem description and handshake clips are combined into a single video insert.

```
(set intro-length (movie-duration intro-movie))
(set alotofprobs-length (movie-duration alotofprobs-movie))

(if (= rv-length "short")
    (begin
      (set intro-start (- (movie-duration revwalk-movie)
                          (* 0.2 (movie-duration intro-movie))))
      (set intro-movie (shift-start intro-start intro-movie))

      (set walkabout-start (+ intro-start (* 0.5 intro-length)))
      (set walkabout-movie (shift-start walkabout-start walkabout-movie))

      (set mindset-start (+ (movie-duration intro-movie) pausetime))
      (set mindset-movie (shift-start mindset-start mindset-movie))

      (set alotofprobs-start (- (movie-duration walkabout-movie) 0.4))
      (set alotofprobs-movie (shift-start alotofprobs-start alotofprobs-movie))

      (set problem-start (+ alotofprobs-start (* 0.4 alotofprobs-length)))
      (set problem-movie (shift-start problem-start problem-movie))

      (set handshake-start (- (movie-duration problem-movie) 0.4))
      (set handshake-movie (shift-start handshake-start handshake-movie))

      (set qualities-start (+ handshake-start pausetime -2.0))
      (set qualities-movie (shift-start qualities-start qualities-movie))

      (set wrapup-start (- (movie-duration handshake-movie) 4.0))
      (set wrapup-movie (shift-start wrapup-start wrapup-movie))

      (set bigdoor-start (- (movie-duration wrapup-movie) 3.5))
      (set bigdoor-movie (shift-start bigdoor-start bigdoor-movie))

      (set title-start (- (movie-duration bigdoor-movie) pausetime))
      (set title-movie (shift-start title-start title-movie))

      (set music-duration (movie-duration title-movie))
      (set music-movie (audio-only (prepare-clip-time music-clip music-duration)))
      (set music-movie (set-level (* 0.25 music-level) music-movie))
      (set music-movie (ramp-in 1.0 (ramp-out 2.0 music-movie)))

      (set final-movie
          (combine-movies revwalk-movie intro-movie walkabout-movie mindset-movie
                          alotofprobs-movie problem-movie
                          handshake-movie qualities-movie wrapup-movie bigdoor-movie
                          title-movie music-movie))

      (set final-movie (sequence-movies 0.0 [black5-movie final-movie black5-movie]))))
```

Below is the final assembly for the medium or long attention spans, which includes the long voiceover about concerns.

```
(begin  # medium or long attention span
  (set intro-start (- (movie-duration revwalk-movie)
                   (* 0.2 (movie-duration intro-movie))))
  (set intro-movie (shift-start intro-start intro-movie))

  (set walkabout-start (+ intro-start (* 0.5 intro-length)))
  (set walkabout-movie (shift-start walkabout-start walkabout-movie))

  (set mindset-start (+ (movie-duration intro-movie) pausetime))
  (set mindset-movie (shift-start mindset-start mindset-movie))

  (set alotofprobs-start (- (movie-duration walkabout-movie) 0.4))
  (set alotofprobs-movie (shift-start alotofprobs-start alotofprobs-movie))

  (set problem-start (+ alotofprobs-start (* 0.75 alotofprobs-length)))
  (set problem-movie (shift-start problem-start problem-movie))

  (set probvoiceover-start (+ (movie-duration alotofprobs-movie) pausetime))
  (set probvoiceover-movie (shift-start probvoiceover-start probvoiceover-movie))

  (set solveproblem-start (- (movie-duration problem-movie) pausetime))
  (set solveproblem-movie (shift-start solveproblem-start solveproblem-movie))

  (set handshake-start (- (movie-duration solveproblem-movie) 0.4))
  (set handshake-movie (shift-start handshake-start handshake-movie))

  (set qualities-start (+ handshake-start pausetime))
  (set qualities-movie (shift-start qualities-start qualities-movie))

  (set wrapup-start (- (movie-duration handshake-movie) 4.0))
  (set wrapup-movie (shift-start wrapup-start wrapup-movie))

  (set bigdoor-start (- (movie-duration wrapup-movie) 3.5))
  (set bigdoor-movie (shift-start bigdoor-start bigdoor-movie))

  (set title-start (- (movie-duration bigdoor-movie) pausetime))
  (set title-movie (shift-start title-start title-movie))

  (set music-duration (movie-duration title-movie))
  (set music-movie (audio-only (prepare-clip-time music-clip music-duration)))
  (set music-movie (set-level (* 0.25 music-level) music-movie))
  (set music-movie (ramp-in 1.0 (ramp-out 2.0 music-movie)))

  (set final-movie
      (combine-movies revwalk-movie intro-movie walkabout-movie mindset-movie
                      alotofprobs-movie problem-movie
                      probvoiceover-movie solveproblem-movie
                      handshake-movie qualities-movie wrapup-movie bigdoor-movie
                      title-movie music-movie))

  (set final-movie (sequence-movies 0.0 [black5-movie final-movie black5-movie])))
)
```

By the end of the script, the variable `final-movie` contains the complete edit decision list for the given set of profile values. This description is passed directly to the playback system which renders the audio and video for the viewer to watch.

## Possible improvements

As stated above, a program generated by Viper will only be as rich as the annotation system and editing model created by its producer. This responsive campaign advertisement production has many weaknesses that could have been avoided with some additional work on these components.

For example, the clips selected for each of the three montage segments in the middle of the movie are sequenced in random order, as this was seen as adequate in the initial version to get the message across to the viewer. But this approach results in some occasionally awkward juxtapositions—for example when there is camera motion in one direction in one clip and then in the other direction in the following clip, when two extreme close-ups of Aisling are cut together, or when, by chance, two otherwise unrelated shots that originated at the same location in the building are placed in sequence with one another.

All of these difficulties could be averted by adding a few additional annotations on the relevant clips and a few more lines in the editing model. For example, clips could be tagged with information about the direction of camera motion at the clip's entry and exit points, perhaps represented by a few keywords like "motion-left-entry," "motion-right-exit," or even a numerical rating expressing speed of motion various directions. The editing model would use these annotations and inspect the ordering of clips in these sequences to ensure that, for example, clips ending with motion to the right are not followed by clips with motion to the left. In addition, clips could be tagged based on framing—wide, medium, close, and so on—and the model adjusted to minimize placing clips with similar characteristics in series with each other. Similar strategies could be used to avoid other cinematic snafus like 180 degree cuts and other breaks in continuity. Some of these techniques are important components in *Sashay/Sleep Depraved* and the *CINEMAT*, as described in Chapter 3, and could have been modeled in Viper.

Although the graphical interface does not include this functionality yet, Viper also allows for annotations in a text file describing relationships *between* clips. For example, if two of the problem imagery clips for the campaign advertisement are related in some way—perhaps one must appear before the other—a relationship annotation could be entered and the editing model adjusted to ensure that if both clips are selected for inclusion that they appear in a particular order. If only one of the clips is selected, the editing model could force the other to be included as well, if desired.

Another complaint about the current advertisement as it stands is that in the middle segment, where Aisling addresses a particular concern about the lab, the sequence of imagery seen in the video insert does not really reflect the thread of her statements. A much better albeit more complicated approach is possible here as well. The clips illustrating problems in the lab could be annotated with more information about their specific content—for example the clip where Aisling finds an empty play-doh bin might include the keywords "playdoh" or "empty-bin." Then, if a transcript of Aisling's interviews with detailed timing information is entered into the editing model, the Viper primitives could support choosing and ordering clips to correspond to the moment-to-moment content of her statements.

Although the campaign advertisement did not require this technique, in assembling a sequence of shots that represent a continuous stretch of time, cutting from one shot to the next on a gesture or other gross event often produces the least awkward result. For example, if a character in a movie is about to open a door and walk into a room, an editor will typically cut from the exterior to the interior shot just as the doorknob is turned. This strategy of cutting on gestures could be automated in Viper. Currently, Viper's graphical interface does not have a friendly way of placing named gesture point annotations in the middle of a clip, but such annotations could be included as gesture/timestamp pairs in a text file. If the relevant gestures occur at the exact beginning or end of a clip then a keyword annotation may be adequate as well. The author could then use these annotations in the editing model to trim and sequence clips so that these gesture points coincide.

# Evaluation

Even though it has been in existence for a shorter period of time, Viper has proven successful in a number of important ways with respect to the three main design guidelines presented at the beginning of this document. Viper's main

technological strength is its inherent support for the kinds of multi-layered edits and effects that traditional video producers rely on in making effective programs with high production values. Because of its capability to assemble and render movies from their constituent media objects at full frame rate, Viper opens the door to the ability to instantaneously modify editing in the middle of playback in response to real-time audience activity and other factors, not just on a segment to segment basis like the other video tools described previously.

Viper's visual interfaces for preparing databases of clips and viewing edit decision lists puts a familiar face on a very different kind of production process and eases the burden on those who are skilled with traditional video editing tools in shifting to the new kinds of tasks associated with a Viper production. The individual components of these tools may operate synchronously on separate workstations connected by a network so that several collaborators can work on different aspects of a project at the same time.

Viper's main achievement is the formation of a common framework of primitives for developing the annotation schemes and editing models that govern how response variables map to complete video programs. These primitives are extensible and flexible enough to support any of the video systems described at the beginning of this chapter as well as the various storytelling experiments described in Chapter 3, yet specific enough to insulate the producer from unnecessary details of general-purpose programming. In an approach unlike Cabbage and other case-based reasoners in which the rules governing a system's operation are often hidden and unchangeable by the user of the tool, Viper has an open system in which a producer can share and build upon elements from prior projects and incrementally construct a customized model that satisfies the needs of a particular production.

Because it is still young, one weakness of Viper is that there is not yet a large body of prior work from which to draw higher-level components to use in the editing models for future productions. As more work is done using Viper, producers will be able to borrow more and more pieces from previous productions to reduce the amount of work that needs to be done for a new project. Over time, producers may build up large libraries of components that they commonly need in their productions, and such libraries might even be packaged and marketed to other users.

As discussed in previous sections, although keywords and numerical ratings can go a long way, the capability to log other types of annotations should be added to Viper's graphical interface. However, opinions differ on the usefulness of creating a more graphical interface for developing Viper editing models. Having these primitives based in textual Isis programs was helpful during the development of Viper because it enabled more rapid experimentation and testing of different kinds of operations. The availability of the general-purpose Isis syntax for use in making an editing model is also important in situations where highly customized calculations or mappings are needed, as was the case in the responsive campaign advertisement. However, a more graphical interface might better reflect the kinds of manipulations being performed with the media objects in a way that makes them easier to understand by a wider variety of users. These issues suggest many clear avenues for further investigation, some more of which will be highlighted in the next chapter.

# Chapter 6
# Conclusion

## Contributions

The first goal of this body of research has been to develop a broader under-standing of responsive media and the processes by which we conceive and build such media. As outlined previously, this understanding boils down to three basic observations—that responsive media consist of **multi-sensory interfaces and materials**, **multi-person collaborations and experiences**, and **multi-layered tasks and thinking processes**. These realizations arise from an awareness of the wide variety of forms and the vast range of possibilities that exist, in addition to the experience of developing different kinds of responsive media applications, a variety of which are described in detail in the latter part of Chapter 3.

These three guiding observations form an approach for constructing a new genre of computer-based design tools that allow their users to more efficiently create and more effectively communicate with responsive media. This approach stands as the foundation of the three main material contributions of this research:

- *Isis*: a new programming language, specially tailored to serve as a basis for responsive media

- *Cabbage*: an experiment in employing a purist form of case-based reasoning in a tool for creating responsive graphical layouts

- *Viper*: a tool for creating video programs that can re-edit themselves to adjust to presentation conditions, audience activity, and other factors

The contributions within each of these three tools are perhaps best collectively summarized within the context of the three design guidelines mentioned above.

### Multi-sensory interfaces and materials

Responsive media often involve a wide variety of sensing and display interfaces and a rich amalgamation of audio, video, graphics and other media types, possibly spread over several computing nodes.

The design of the Isis programming language aims to support these complex and demanding raw materials in a number of ways. Isis is a complete language, enabling programmers to implement arbitrarily complex input-output mappings, and provides a number of libraries for handling a variety of unusual input and output devices. The "lean and mean" internal design of Isis includes a small number of core constructs and primitive operations, internal structures based on native machine types and arrays rather than strings or linked lists, and no extra overhead for security layers or virtual machines. The syntax is designed to support the memory management system in a way that eliminates the need for garbage collection of basic values. Together these internal optimizations result in a

small and fast interpreter compared to other non-compiled languages, and one that doesn't exhibit "hiccups" during execution that can threaten the flow of real-time media experiences.

Building upon the foundation of interface functionality provided by Isis, Cabbage and Viper aim to support richer forms of spectator and audience sensing as well. Cabbage incorporates case adaptation rules that generate intuitive results yet involve a minimum of complex math, enabling smooth transitions in graphical layouts in response to real-time sensor feedback. Viper enables producers to make video programs that can instantly and seamlessly "re-edit" themselves at the same time they are being viewed, to adjust to real-time audience activity, environmental conditions, or other response factors. These programs can also take into account equipment configurations as well as viewer preferences or profile information. Extending upon the limited functionality of earlier automated editing tools, Viper's primitives support multiple video and audio layers and intricate types of edits and transitions, such as inserts, dissolves, graphics, and so on. Together these innovations enable producers to create video programs that exhibit rich forms of responsiveness while maintaining the high production values that are essential in making compelling experiences.

## Multi-person collaborations and experiences

Because responsive media projects often span multiple disciplines, they typically involve many different kinds of people with many different kinds of expertise, working collaboratively to realize a common goal.

Isis incorporates a fixed minimalist syntax, lessening the burden on programming novices wishing to master the language and providing a common framework for collaborators to more easily comprehend and build upon each other's contributions to a project. Yet while small and immutable, the syntax still provides all of the features expected of a good programming language that hackers need in order to take full advantage of their skills. Experience indicates that these design choices have resulted in a language that is indeed more accessible to a wider variety of users. The total Isis environment also includes a set of community support tools centered around a web site that allows users to publish and share customized packages of functionality while sustaining the "lean and mean" minimalist attitude of the core design of the language. The design of Viper goes one step further and incorporates interfaces that can operate synchronously on separate networked workstations, allowing several collaborators to work simultaneously on different aspects of a single project.

In addition, some of the best forms of responsive media are those that involve multiple audience members or participants. Reflecting this observation, all three of Isis, Cabbage, and Viper, as discussed above, strive for better support of non-standard and unusual interface devices, opening the door to new forms of computer-mediated responsive media that engage several viewers or spectators at the same time.

## Multi-layered tasks and thinking processes

Creators of responsive media have many different ways of thinking about and approaching the tasks that face them, and this design process typically involves the need to work incrementally and to innovate at many different levels.

As an interpreted language, Isis is more amenable to an incremental design process in which changes and improvements can be integrated into a prototype more

readily, even during execution. Wherever possible, the libraries of functionality offered by Isis follow a multi-level design strategy, consisting of multiple layers offering different levels of abstraction of a particular set of operations, designed to interoperate with each other in the most seamless way.

However, Isis still requires its users to think about input-output mappings in an abstract rule-based fashion which may not match well with the more concrete example-based thinking styles of many of its likely users. Further, its textual interface in many ways does not reflect the types of media being manipulated and the range of operations available. With this in mind, Cabbage is designed as an experiment to explore the opposite end of the spectrum—to investigate the viability of applying case-based reasoning in a complex responsive media problem domain. Cabbage aims to innovate in the user interface realm as well by incorporating tool components that emphasize visibility and accessibility to the highest degree possible. Cabbage allows designers to create responsive graphical layouts by demonstrating examples of what the proper layout should be at different points in an N-dimensional "design space" whose axes represent the individual response variables. The tool employs several new algorithms that "reason" from these specific examples and generate new designs appropriate for other situations.

Although successful in many ways, including that it does not require a designer to write a single line of code to create fairly elaborate dynamic behaviors, the results of the Cabbage experiment indicate that case-based approach alone may not be adequate to support a serious production tool for a non-trivial form of responsive media. The heuristic and incomplete nature of the adaptation rules in case-based reasoners does not appear to match well with designers who often approach computational devices in search of exactness. Further, the fact that these rules are usually not intended to be visible to or changed by the designer poses serious concerns about the extensibility of these systems beyond their specific problem realms.

Informed by these lessons, Viper presents an approach more like Isis in its programming strategy, but that borrows from Cabbage in the look and feel of its user interfaces, many of which are designed in a fashion to be familiar to users of traditional video editing tools and to support an incremental production process in which a producer can shift easily between the various tasks and view results immediately as changes are made.

Just as Isis aims to serve as a flexible basis for responsive media as a whole, Viper addresses the need for a common framework for creating responsive video applications. In contrast to the other tools and prototypes discussed in earlier chapters, the primitives offered by Viper enable producers to create a customized annotation system and editing model specific to the needs of a particular project. These primitives are carefully designed to shield the producer from excessive details of general-purpose programming while still providing a high degree of extensibility and flexibility in the range of applications supported by the tool. This common framework introduces new opportunities for producers to more readily share and build upon components from each other's projects.

## Other contributions

Along with these three main contributions, this research also presents several prototypes and applications created with these three tools that, on one level, demonstrate each tool's viability as a platform for thinking differently and more clearly about responsive media design projects. On another level, these prototypes, ranging from hyperlinked video soap operas to interactive art installations

to responsive political campaign advertisements, illustrate the enormous potential computer-mediated responsive media have in commerce, education, communication, expression, and other domains. This realization only intensifies the importance of understanding these media and building more appropriate design tools, to enable designers and producers to make the wisest and most effective use of these media for whatever purposes they are needed.

# The journey continues

As much as one would desire to bring a thread of research completely to a close, the first realization that arises from looking back at the body of work presented in this document is that it is not so much a "body," in the sense of being a closed entity, as it is only a small peninsula connected to a vast continent that has yet to be explored and is probably worth exploring. Chapters 3 through 5 present evaluations of the three tools created in this research and allude to some possible directions for further investigation, but a few more such "jumping off" points are described here.

The majority of programming languages in use today consist of simple textual user interfaces, in part because computers are very good at processing sequential lines of ASCII characters and because many of these languages trace their beginnings to a time when anything more than a text interface was the exception and not the rule. With advances in graphical interfaces, new languages, like MAX and Prograph, have tested more visual approaches utilizing flowcharts and other metaphors, which exhibit many problems of their own, as discussed in Chapter 2. In the end, it is not necessarily the abundant presence of text that is the crutch in programming as much as it is the way we create and organize that text, often in editors that have their roots in generic word processing and are not inherently tailored for the differently-structured tasks of designing algorithms and encoding their implementations. Parentheses, semicolons, ampersands, backslashes, and a variety of other symbols borrowed from human languages are unwieldy and unsightly as syntactical constructs and grouping mechanisms in computer programming languages. All of these issues call for a complete rethinking of the design of computer program editors to meld more closely with the characteristics of the languages and tasks at hand, perhaps by incorporating more than mere ASCII symbols, by including other graphical mechanisms to assist in the organization of ideas in a program, and by much more closely integrating the interpretation, debugging, and documentation systems of a language directly into the program creation process. A new system could also more closely incorporate many of the community support components provided by the Isis web site, making it easier for programmers to search, share, publish, and ultimately build upon each other's work.

First time users of Cabbage enjoy the simplicity of being able to create an intricate dynamic layout by example, without having to encode any rules in a programming language. But after this initial honeymoon period is over, the frustration level increases as a desire to increase the complexity of a design sets in. Through the process of working directly with the media objects and constructing scores of examples, users become more conscious of the behaviors they wish to encode in a design at a more abstract level and begin to rub up against the limitations of many case-based reasoning systems in handling subtle relationships, personal styles, and large numbers of variables. It is at this point in the design process that a more rule-based approach begins to make more sense. This observation suggests the formulation of a new tool that could enable a case-based process of discovery and exploration during the initial stages of a project's development, and that can then translate or transfer knowledge and relationships

gleaned from this exploration into a more rule-based interface for more precise refinement leading up to completion. In many ways this transition from the concrete to the abstract is what many designers experience in working on projects in many traditional domains, and this kind of tool would assist designers in forming that abstract understanding and representing it programmatically more efficiently. Such a tool could also morph into a form useful in education, perhaps in mathematics or computer science, where learning to think about problems in terms of abstract components and manipulations is a critical pedagogical goal.

Viper gives producers the ability to create a customized system of annotations, within which source video and audio material is organized in a database, as well as a customized model of editing to control how that material will be assembled in different situations. But some interesting opportunities result when one or more of these system components is generated by a different party or held fixed. Creating standardized systems of annotation, as is the broad aim of the MPEG-7 effort, would enable the development of very large repositories of specific kinds of video or audio clips, such as historical imagery or family tree material, from which different producers could draw to satisfy their own customized editing model. The converse is also interesting, in which a third party would provide initially fixed annotation and editing model templates. Producers could shoot their own content to fulfill a given basic model at first, and then make changes and additions to that model more quickly to tailor a production for specific needs. This idea would be particularly intriguing in an educational tool. Students could learn the basics of telling video stories by shooting their own material to fit a Viper template, and then later be able to rapidly experiment with different ways of telling those stories by making changes to the editing model. In another thread, Viper could also be extended to handle live video material. Assuming each live feed could be annotated in real-time with information about its content, Viper could control the selection of which stream to show at different points in time, opening the door to applications such as a live sporting event that allows its viewers to select preferences about how they want the event presented—team or player emphases, selection or non-selection of commentators, pacing of cuts, and so on.

These are just a few possible future directions related to the three tools created in this research. Other areas suggest themselves in pondering how the three guiding observations repeated throughout this document could be applied to rethink the designs of many of the currently popular tools and to develop new ones for more specific kinds of responsive media. Interestingly, one area that is still clearly lacking and ripe for innovation is physical design. All three of the tools described in this dissertation are bound to the same single-user single-workstation metaphor employed by virtually every other computer-based tool in existence. Although they support collaboration in other ways and consist of elements that could extend easily to multi-user scenarios, none of the interfaces present opportunities for several minds to work on the same aspect of a project or task simultaneously. As discussed earlier, the mere one-person physical design of modern computer workstations reduces the inspiration for and discourages the development of applications and experiences that involve more than a single viewer or spectator at a time, and this in turn harms our ability to communicate with each other and foster the kinds of relationships that are the basis of so many potential forms of responsive media. The technology exists to break away from these antiquated and ineffectual designs, and only when we have the strength to fully explore this territory will new ground truly have been broken.

# Bibliography

[AP94]
Agnar Aamodt and Enric Plaza
*Case-based reasoning: foundational issues, methodological variations, and system approaches*
**AI COMMUNICATIONS**
vol. 7, no. 1, March 1994, pp. 39-59

[AS85]
Harold Abelson and Gerald Jay Sussman
**STRUCTURE AND INTERPRETATION OF COMPUTER PROGRAMS**
MIT Press, Cambridge, Massachusetts, 1985

[AWB97]
Stefan Agamanolis, Alex Westner, and V. Michael Bove, Jr.
*Reflection of presence: toward more natural and responsive telecollaboration*
**PROC. SPIE MULTIMEDIA NETWORKS**
vol. 3228A, 1997

[Bai97]
Freedom Baird
*Tilting at a dreamer's windmills: gesture-based constructivist interaction with character*
**PROC. CAIIA CONSCIOUSNESS REFRAMED**
University of Wales College, Newport, Wales, 1997

[BB00]
William Butera and V. Michael Bove, Jr.
*The coding ecology: image coding via competition among experts*
**IEEE TRANS. CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY**
vol. 10, October 2000, pp. 1049-1058

[Bob99]
Aaron Bobick, et. al.
*The KidsRoom: a perceptually-based interactive immersive story environment*
**PRESENCE: TELEOPERATORS AND VIRTUAL ENVIRONMENTS**
vol. 8, no. 4, 1999, pp. 367-391

[Bov00]
V. Michael Bove, Jr., Jonathan Dakss, Edmond Chalom, and Stefan Agamanolis
*Hyperlinked video research at the MIT Media Laboratory*
**IBM SYSTEMS JOURNAL**
vol. 39, no. 3-4, 2000

[Bov95]
V. Michael Bove, Jr.
*Object-oriented television*
**SMPTE JOURNAL**
vol. 104, December 1995, pp. 803-807

[Bro99]
Kevin Brooks
**METALINEAR CINEMATIC NARRATIVE: THEORY, PROCESS, AND TOOL**
PhD thesis, MIT, 1999

[BW95]
V. Michael Bove, Jr. and John A. Watlington
*Cheops: a reconfigurable data flow system for video processing*
**IEEE TRANS. CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY**
vol. 5, April 1995, pp. 140-149

[CB96]
Edward Chalom and V. Michael Bove, Jr.
*Segmentation of an image sequence using multi-dimensional image attributes*
**PROC. IEEE INT. CONF. IMAGE PROCESSING**
1996, pp. 525-528

[Coh84]
Scott Cohen
**ZAP! THE RISE AND FALL OF ATARI**
McGraw Hill, New York, 1984

[Cyp93]
Allen Cypher, editor
**WATCH WHAT I DO: PROGRAMMING BY DEMONSTRATION**
MIT Press, Cambridge, Massachusetts, 1993

[Dal96]
Mukesh Dalal, et. al.
*Negotiation for automated generation of temporal multimedia presentations*
**PROC. ACM MULTIMEDIA**
1996, pp. 55-64

[Dav00]
Glorianna Davenport,  Stefan Agamanolis, Barbara Barry, Brian Bradley, and Kevin
    Brooks
*Synergistic storyscapes and constructionist cinematic sharing*
**IBM SYSTEMS JOURNAL**
vol. 39, no. 3-4, 2000

[Dav97]
Glorianna Davenport, Stefan Agamanolis, Brian Bradley, and Flavia Sparacino
*Encounters in dreamworld: a work in progress*
**PROC. CAIIA CONSCIOUSNESS REFRAMED**
University of Wales College, Newport, Wales, 1997

[DF95]
Glorianna Davenport and Larry Friedlander
*Interactive transformational environments: Wheel of Life*
**CONTEXTUAL MEDIA: MULTIMEDIA AND INTERPRETATION**
MIT Press, Cambridge, Massachusetts, 1995

[DM95]
Marc Davis
*Media streams: an iconic visual language for video representation*
**READINGS IN HUMAN-COMPUTER INTERACTION: TOWARD THE YEAR 2000**
Morgan Kaufmann, 1995, pp. 854-866

[Dov97]
Toni Dove
*Somatic ventriloquism: throwing the body, distributing the self*
**ABSTRACTS OF CAIIA CONSCIOUSNESS REFRAMED**
University of Wales College, Newport, Wales, 1997

[Gan83]
Steve Gano
**FORMS FOR ELECTRONIC BOOKS**
M.S.V.S. thesis, MIT, June 1983

[GBS97]
Ashok Goel, Sambasiva Bhatta, and Eleni Stroulia
*KRITIK: an early case-based design system*
**ISSUES AND APPLICATIONS OF CASE-BASED REASONING IN DESIGN**
Lawrence Erlbaum Associates, 1997, pp. 87-132

[Ham89]
Kristian J. Hammond
*Case-based planning: viewing planning as a memory task*
**PERSPECTIVES IN ARTIFICIAL INTELLIGENCE**
Academic Press, Boston, Massachusetts, 1989

[Har96]
Bill Harley
*Playing with the wall*
**WHO SAYS: ESSAYS ON PIVOTAL ISSUES IN CONTEMPORARY STORYTELLING**
August House, Little Rock, 1996, pp. 129-140

[Hin88]
Tom Hinrichs
*Towards an architecture for open world problem solving*
**PROC. CBR WORKSHOP**
Morgan Kaufmann, 1988, pp. 182-189

[Hou94]
Gilberte Houbart
**VIEWPOINTS ON DEMAND: TAILORING THE PRESENTATION OF OPINIONS IN VIDEO**
M.S. thesis, MIT, September 1994

[HS92]
Jim Hollan and Scott Stornetta
*Beyond being there*
**PROC. ACM CHI**
1992, pp. 119-125

[IKA94]
Hiroshi Ishii, Minoru Kobayashi, and Kazuho Arita
*Iterative design of seamless collaboration media*
**COMMUNICATIONS OF THE ACM**
vol. 37, August 1994, pp. 83-97

[Kru96]
Myron W. Krueger
*Responsive environments*
**THEORIES AND DOCUMENTS OF CONTEMPORARY ART: A SOURCEBOOK OF ARTISTS' WRITINGS**
Morgan Kaufmann, 2001

[LAL00]
Thomas Lewis, Fari Amini, and Richard Lannon
**A GENERAL THEORY OF LOVE**
Random House, New York, 2000

[LR00]
Rainer Lienhart
*Dynamic video summarization of home video*
**PROC. SPIE 3972: STORAGE AND RETRIEVAL FOR MEDIA DATABASES**
January 2000, pp. 378-389

[Lie93]
Henry Lieberman
*Mondrian: A teachable graphical editor*
**WATCH WHAT I DO: PROGRAMMING BY DEMONSTRATION**
MIT Press, Cambridge, Massachusetts, 1993, pp. 341-358

[Lie01]
Henry Lieberman, editor
**YOUR WISH IS MY COMMAND: PROGRAMMING BY EXAMPLE**
Morgan Kaufmann, 2001

[Lov97]
Margot Lovejoy
**POSTMODERN CURRENTS: ART AND ARTISTS IN THE AGE OF ELECTRONIC MEDIA**
Prentice Hall, second edition, 1997

[Mac96]
Tod Machover
*Brain Opera*
**MEMESIS, THE FUTURE OF EVOLUTION: ARS ELECTRONICA FESTIVAL 96**
Springer Verlag, New York, 1996

[Mar96]
Rafe Martin
*Between teller and listener: the reciprocity of storytelling*
**WHO SAYS: ESSAYS ON PIVOTAL ISSUES IN CONTEMPORARY STORYTELLING**
August House, Little Rock, 1996, pp. 141-154

[McL94]
Marshall McLuhan
**UNDERSTANDING MEDIA: THE EXTENSIONS OF MAN**
MIT Press, Cambridge, Massachusetts, 1994

[MDV99]
Michael Mateas, Steffi Domike, and Paul Vanouse
*Terminal time: an ideologically-biased history machine*
**PROC. AISB SYMPOSIUM ON ARTIFICIAL INTELLIGENCE AND CREATIVE LANGUAGE: STORIES AND HUMOUR**
1999, pp. 69-75

[MG97]
Mary Lou Maher and Andrés Gómez de Silva Garza
*Case-based reasoning in design*
**IEEE EXPERT**
vol. 12, March-April 1997, pp. 34-41

[Moh82]
Robert Mohl
**COGNITIVE SPACE IN THE INTERACTIVE MOVIE MAP: AN INVESTIGATION OF SPATIAL LEARNING IN VIRTUAL ENVIRONMENTS**
PhD thesis, MIT, February 1982

[Mor92]
Lee Hayes Morgenroth
**HOMER: A VIDEO STORY GENERATOR**
B.S. thesis, MIT, May 1992

[Mur96]
Michael Murtaugh
**THE AUTOMATIST STORYTELLING SYSTEM: PUTTING THE EDITOR'S KNOWLEDGE IN SOFTWARE**
M.S. thesis, MIT, 1996

[MWP98]
Baback Moghaddam, Wasiuddin Wahid, and Alex Pentland
*Beyond eigenfaces: probabilistic matching for face recognition*
**PROC. IEEE INT. CONF. AUTOMATIC FACE AND GESTURE RECOGNITION**
April 1998, pp. 30-35

[Nai97]
Michael Naimark
*What's wrong with this picture? Presence and abstraction in the age of cyberspace*
**PROC. CAIIA CONSCIOUSNESS REFRAMED**
University of Wales College, Newport, Wales, 1997

[NL99]
Frank Nack and Adam T. Lindsay
*Everything you wanted to know about MPEG-7: part 1*
**IEEE MULTIMEDIA**
vol. 6, no. 3, July-September 1999, pp. 65-77

[NP97]
Frank Nack and Alan Parkes
*The application of video semantics and theme representation in automated video editing*
**MULTIMEDIA TOOLS AND APPLICATIONS**
vol. 4, no. 1, January 1997, pp. 57-83

[Per64]
Donald G. Perrin
*A branching teaching machine using motion pictures*
**SMPTE JOURNAL**
vol. 73, September 1964, pp. 760-764

[Pic00]
Rosalind Picard
*Toward computers that recognize and respond to user emotion*
**IBM SYSTEMS JOURNAL**
vol. 39, no. 3-4, 2000

[Pop93]
Frank Popper
**ART OF THE ELECTRONIC AGE**
Thames and Hudson, London, 1993

[Rit93]
Dennis M. Ritchie
*The Development of the C Language*
**PROC. ACM HISTORY OF PROGRAMMING LANGUAGES II**
1993, pp. 201-208

[RBK98]
Henry A. Rowley, Shumeet Baluja, and Takeo Kanade
*Neural network-based face detection*
**IEEE TRANS. PATTERN ANALYSIS AND MACHINE INTELLIGENCE**
vol. 20, January 1998, pp. 23-38

[RS89]
Christopher K. Riesbeck and Roger C. Schank
**INSIDE CASE-BASED REASONING**
Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1989

[Shu88]
Nan C. Shu
**VISUAL PROGRAMMING**
Van Nostrand Reinhold, 1988

[Sim00]
David Simons
*Digital TV wins*
**FORBES**
vol. 28, June 2000

[Smi96]
Josh R. Smith
*Field mice: extracting hand geometry from electric field measurements*
**IBM SYSTEMS JOURNAL**
vol. 35, 1996, pp. 587-608

[SW63]
Claude E. Shannon and Warren Weaver
**THE MATHEMATICAL THEORY OF COMMUNICATION**
University of Illinois Press, Chicago, 1963

[Tuf97]
Edward R. Tufte
**VISUAL EXPLANATIONS: IMAGES AND QUANTITIES, EVIDENCE AND NARRATIVE**
Graphics Press, Cheshire, Connecticut, 1997

[Vas00]
Nuno Vasconcelos
*A probabilistic architecture for content-based image retrieval*
**PROC. IEEE COMPUTER VISION AND PATTERN RECOGNITION**
2000

[Wei95]
Louis Weitzman
**THE ARCHITECTURE OF INFORMATION: INTERPRETATION AND PRESENTATION OF INFORMATION
IN DYNAMIC ENVIRONMENTS**
PhD thesis, MIT, February 1995