A Digital-Video Storyboarding and Sequence-Visualization Computer Program

by

Eugene J. Rhough

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science in Electrical Science and Engineering

at the Massachusetts Institute of Technology

May 1993

Author ______________________________

Department of Electrical Engineering and Computer Science
May 17, 1993

Certified by ______________________________

Associate Professor Glorianna Davenport
Thesis Supervisor

Accepted by ______________________________

Leonard A. Gould
Chairman, Department Committee on Undergraduate Theses

A Digital-Video Storyboarding and Sequence-Visualization Computer Program

by

Eugene J. Rhough

Submitted to the

Department of Electrical Engineering and Computer Science

May 17, 1993

In Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science in Electrical Science and Engineering

ABSTRACT

A computer program has been implemented that allows a user to prototype and experiment with different digital-video movie sequences. The computer program, written on an Apple Macintosh computer, explores new interface mechanics that better exploit the non-linear, low-latency characteristics of digital movie files. The program allows a user to organize, annotate, and sequence movie clips. Final sequences can be saved as pointer-based files, allowing further fine-detail editing with other programs.

Thesis Supervisor: Glorianna Davenport
Title: Associate Professor of Media Arts and Technology, MIT Media Laboratory

Acknowledgements

I would like to thank Professor Glorianna Davenport for her tremendous help, support, and encouragement these past four years. This thesis could not have been completed without her insight and guidance. I would also like to thank Hans Peter Brøndmo for encouraging me to think critically of digital video editing.

Table of Contents

Overview

This research thesis concerns the new challenges and possibilities that digital video sequencing provides. It consists of eight sections: "Background," in which we place recent advances in digital video technology in context by briefly reviewing the history of film and video; "Introduction," which outlines some general thoughts on digital video and discusses the motivations behind this thesis project; "Scenarios," which fleshes out some of the motivations of the previous section by describing detailed, specific situations in which new ideas, like those of this research thesis, may be helpful; "Applications Survey," which looks at current, commercial responses to digital video; "Functional Specification," which discusses the functional, mechanical, and interface features of a computer program that has been implemented to specifically address some of the concerns raised in "Scenarios;" "Technical Specification," which describes the interesting engineering and implementation issues encountered while coding the computer program; "Further Ideas," which explores possible responses to some of our motivating problems that were not implemented in our computer program; and "Conclusions," which reviews the lessons learnt from this thesis and attempts to generalize ideas that may be of use in future work.

Background

This section provides a very brief history of film and video, emphasizing the close relationship between the technology and expression of film and video. We will focus specifically on the claim that the expressive capabilities, creative environment, and aesthetics of the moving-image art have been, and continue to be, indivisibly intertwined with the functional mechanics and technical requirements of moving-image capture and reproduction. This section provides a context with which to evaluate the proposed motivations for this thesis, which we discuss in the next section. We divide the relevant history of film and video into three chronological categories: film, analog video, and digital video.

Film

The story of moving-image capture and reproduction begins with film, an invention of the mid-nineteenth century. Static film cameras used chemical technology to capture and store the light and features of an image, imprinting high-resolution, sharply-defined images onto special film stock. A number of inventors, perhaps spurred on by the suggestive experiments of Edward Muybridge, soon began toying with the notion of "animated" film, or moving-pictures. Moving-pictures, or "movies," exploited the limited nature of human vision: a succession of static images, if presented rapidly enough, could present the illusion of smooth and natural animation. By the turn of the century, motion-picture film cameras, devices which conceptually remain essentially unchanged, were being used to modest success, and the foundation for the film art was firmly in place.

Basic Technical Features of Film

The basic mechanics of motion-picture film capture and reproduction are simple. A motorized camera, fed a long strip of film, is used to capture a long succession of images in time. The earliest motion-picture film cameras could capture several minutes of animated footage. Cameras typically capture 24 frames-per-second, an animation rate that can comfortably trick the human eye into perceiving real motion. Sound cameras also record sound information, which is usually placed next to or on top of the film image on the film strip. Editing footage requires physically cutting and re-ordering segments of film in a film strip. Film footage is displayed through a projector, which runs through a film strip while throwing light through each frame, usually maintaining a constant time rate that matches the same time rate used to capture with.

The Relationship between Technology and Movies

Over the years, some of the most distinctive, salient characteristics of film pieces have remained closely tied to the technology of film. Physical constraints always limit what images a movie-creator may capture, and the functional mechanics of editing similarly constrain how a movie-creator may present a piece: so while a person is free to mentally visualize anything his or her imagination can create, a person is constrained to filming and presenting only what is technically possible. We might say that these technical constraints partly dictate what images are captured and how stories are told, and that they help direct the maturation of film style and aesthetics. Technology plays an important role in how movies are conceived and realized.

We can illustrate these thoughts by reviewing some examples of how changes in film technology directly, and sometimes dramatically, altered the film experience:

- Early motion-picture pieces by the Lumière brothers were short tableaus, quick glimpses of life and people. Limited in length by the size of the film reels, and lacking sound, these shorts resembled postcards or paintings. Their stories were events or scenes: women leaving a factory at the end of the day; a wall falling; children swimming; people play cards. By 1920, films had become considerably more expressive: they used multiple reels to provide more playing time, offering more complex narratives and sophisticated stories. Films became a vehicle for mass story-telling. Films tended to be shot in the open-air, a requirement of insensitive film stock. Lacking sound, films also developed a visual and communicative vocabulary uniquely suited to the silent film, inspiring dramatic visual styles like Expressionism, Dadaism, and Surrealism.
- The introduction of sound a decade later would significantly change the nature of movies. Sound would change established film dramatics, forcing a new look at critical issues like storytelling, continuity, and aesthetics. The sensitivity and

bulkiness of sound equipment would initially destroy the emerging fluidity and freedom of camera placement and motion.

- Improvements in on-location synchronized sound and film recording would allow film pieces to move on from canned, controlled shoots to more unpredictable, observational or news-reporting shoots.

There are also some technological constraints and influences on the general *environment* of film-making whose impact cannot be directly measured. Consider the "no-feedback" nature of film shooting and the high costs of film editing. Cameramen cannot immediately see what they are filming, since there is no direct feedback, and footage must be sent for processing before it may be reviewed. Cameramen can only attempt to predict how lighting, film exposure, and other factors will affect the final image. Film editors are "penalized" in time and effort for attempting to experiment with multiple cuts of a sequence, since source footage is physically cut up when sequenced, and each cut requires new source footage and physical construction work. Editing requires manual handling, which results in footage wear-and-tear, and it may be difficult to "back up" or undo bad decisions.

Analog Video

If we accept that technology has played a critical role in the evolution of movies, then we raise the interesting possibility of influencing movies by changing the technology. The introduction of analog video, an electronics-based system of movie capture, transmission, and display, did just that: analog video changed the nature of the movies by offering a new, unique set of capabilities and limitations.

Analog video differs from film in a number of ways, and has advantages that allow it to operate as the low-cost workhorse of movie production. In many ways, it offers a more flexible, and less expensive, creative environment:

- First of all, analog video permits the live capture, broadcast, and display of synchronized audio/visual images: we can now instantaneously see images, as they occur, from thousands of miles away.
- Analog video is relatively cheap, and has "democratized" movies. For little money, amateurs and enthusiasts can use VCR's to copy and play their own movies from their own library, as well as use low-cost camcorders to record their own stories and family activities. Analog video has allowed non-professionals to express themselves with an art form that had been previously inaccessible.
- Analog video provides direct-feedback for camera operators.
- Analog video has simplified the postproduction process, while opening up new, low-cost possibilities like easily-accessible special effects.

Basic Technical Features of Analog Video

Analog video is superficially similar to film, except that it involves electronics. A video camera converts an image into a series of electrical signals. If the camera is tied directly to a television, the television will "read" the signals to drive an electron gun, which will produce an image on a monitor screen. If a video camera is recording to video tape, the camera translates the electrical signals to magnetic information on the video tape. A video cassette player would reverse the process, converting the magnetic information on a tape into an electrical signal, which is then sent to a television or monitor for display as an image.

A simple editing situation would involve three video cassette recorders (VCR's). We would have two decks with source footage, "Deck A" and "Deck B," and we would have a record deck. An edited sequence would be produced by copying footage from the source decks A and B to the record deck. There is no physical, manual cutting-and-taping, as there is with film editing. We can now generate multiple sequence cuts while retaining our original source footage, we can pre-specify and automate the assembly of sequences, and we can re-use our video stock. We also have an easy way of introducing effects like wipes and fades, since special-effects generators are easy to add within the editing process.

Analog video has some disadvantages. It has a lower resolution than film, resulting in images that are less sharp. It has a smaller information bandwidth, which means that video has a more difficult time dealing with extreme light gradients or ranges. New analog video technologies like High Definition video have begun to address these deficiences.

Summary

Analog video's main contributions have been: live broadcast, a sharp decrease in operating costs relative to film, a simplification of editing and production, and the introduction of sophisticated, low-cost special effects. It has increased participation in motion-picture production, and because of its low expense, has allowed a greater flexibility for experimentation and variety.

Digital Video

Digital video represents a quantum leap over film and analog video technologies, and holds the potential to significantly change how we perceive and work with motion-pictures. We'll first take a look at what digital video is, then examine what possibilities digital video offers.

Basic Technical Features of Digital Video

Digital video, like analog video, is an electronics-based technology, and represents images as electrical signals. There is a basic difference, though: analog video represents images as continuous voltage curves, while digital video translates an image into a series of zeros and ones. This discrete characterization of image information allows us to use computers to manipulate and display digital video. The use of computers, as we will later see, offers powerful possibilities.

Digital video movies can be acquired by using regular analog video cameras. An analog video camera would translate a movie into a series of continuous voltage signals. These signals would be sent to an Analog-to-Digital Converter (ADC), a microchip which translates this information into a series of one's and zero's. These one's and zero's would be sent to a computer, which could then store the information on magnetic media (disk or tape drives) or in electronic memory (computer RAM). To display a movie, a user would run a computer program which would be able to read the digital video information and translate it into a movie on the computer screen.

Promises of Digital Video

The integral use of computers in digital video reflects an important change in the technology of moving-image capture and reproduction: the replacement of a number of specialized, single-purpose film/video editing/display devices with a single powerful, flexible, and programmable machine — a computer. This introduction of flexible computational power offers an opportunity to radically change the capabilities of movies, and allows us to critically re-examine the experience of movie-watching and the mechanics of movie-creation.

Changing the Movie Experience

With analog video, as we noted, we deal with a number of specialized, "dumb" machines, and the machines we deal with depends on who we are. A movie-creator has a video camera, editing decks, editing computers, special effects generators, an editing controller, expensive monitors, and high-end video-cassettes. A movie viewer might just have a black-and-white TV. With digital video, there is no such distinction: both a movie-creator and a movie-viewer just need a single, general-purpose machine — the computer. The consequence is that both movie authors and viewers have access to the same editing and movie-creation capability. For now, at least, the same digital-video tools a creator uses to create a movie are also available to a viewer — which allows us to begin to blur the line between movie creation and consumption. A viewer has the potential power to alter movies, to participate in the creative process.

Computers also offer a way of imbedding "intelligence" into movies. Movies could be given information that allow them to respond to viewer cues: a viewer could thus interact with a movie, influencing and codirecting his or her movie experience. If movie-creators create rich, multi-threaded stories specifically for digital video, we might envision digital stories where the viewer actively cooperates to create a unique movie experience.

Changing the Mechanics of Movie-Creation

Digital video and computers also offer the opportunity to rethink how we *create* traditional, linear movies. The introduction of analog video necessitated a different editing paradigm to better exploit and utilize the new technology: it seems reasonable to begin exploring how we might better exploit *digital* video to enhance the creative process. Digital video offers numerous powerful capabilities, including: instantaneous access to any location in any clip; video duplication without any generational quality loss; discrete-time signal processing for filters; the spatial arrangement of video streams; and relational indexing for hypermedia cross-referencing. These new capabilities suggest a number of interesting questions. How can we bring these new features to bear? Does the traditional analog editing process adequately present or help exploit these capabilities? How can these features contribute to a new editing philosophy and process that better encourages creativity and productivity?

Thesis Introduction

This thesis explores some of the interface and procedural issues digital video introduces. As we noted in the previous section, digital video motivates some interesting and significant questions on the nature of our traditional editing processes; is there a way that better exploits the power of digital video, a process that better encourages creativity and productivity?

This thesis seeks to begin to answer this question. We note that while the "movie-creation process" includes numerous stages — pre-production (planning), production (shooting), and post-production (editing) — we will focus specifically on digital video in the post-production process.

A couple of software companies — Adobe and DiVA — have already begun to seriously explore the role of digital video in post-production. Hans Peter Brøndmo, the Director of Engineering at DiVA, divides the digital-video postproduction process into four stages: the acquisition of video, the management of video, the editing of video, and the presentation of video. The ideal movie-creation process would provide a powerful, suggestive, elegant, and integrated solution for all four stages.

There is another important stage, however, that has received little to no attention: the "storyboarding," or story-visualization, stage. The four-stage editing model clearly addresses the *mechanics* of movie creation: one first digitizes a pool of video, arranges them for reference, edits them into a sequence, and then shows the movie. The movie-creator though is constantly grappling with a number of creative issues as well: the movie-creator must make hard decisions on what to show, when to show it, and for how long — and he or she doesn't necessarily know exactly what best "works." The storyboarding stage, stage "two-and-a-half," is an opportunity for the movie-creator to experiment with the gross ordering of a sequence; the movie-creator can juxtapose different shot selections, different sequence orderings, and different soundtracks. The storyboarding stage encourages creative experimentation and visualization by allowing a movie-creator to make non-binding decisions that illustrate different sequence possibilities.

We use the term "storyboarding" in a very particular way. Storyboarding traditionally refers to an initial, pre-production process of story visualization. This thesis refers to "storyboarding" as a post-production process: really a sort of story "*re*-visualization" process. After footage has been shot, collected, and digitized, and before the editor begins to put together the shots into sequence constructions, we define an intermediate period in which the editor attempts to determine how to best use his or her shot pool to reconstruct (or elaborate) earlier visions or ideas. This new storyboarding stage allows an editor to visualize sequences with actual footage.

This thesis attempts to explore how one might implement a storyboarding stage: what functionality to offer, what interface to use, and how one might integrate the stage into the larger editing process.

Scenarios

This section looks closely at two "typical" editing scenarios — two situations that illustrate some of the functional demands a movie-creation process would need to address. We examine the construction of a montage-style commercial advertisement and the planning of a simple narrative: these two scenarios demonstrate particularly well why and how one might need a storyboarding capability.

Montage-style advertisements are an interesting "genre" of commercials. They stimulate "good feelings" about a product by associating the product with pleasurable or glamorous images. Montage commercial designers isolate a theme, usually something youthful, dynamic, sexy, and glamorous, and attempt to visually communicate that theme with short clips. They assemble a 30-second spot featuring these themed clips interspersed with shots of their product. Beer companies, for example, often construct commercials that feature their beer among partying, fun-loving, happy, and sexy young people. The intention is to closely associate a product with a lifestyle or emotions that people desire.

The challenge is to pick the shots that best evoke a certain lifestyle or theme. An advertising-commercial creator has three primary sources to look through: stock footage, company archival footage, and new, original footage. During the planning of a commercial, the creator must constantly attempt to recall and compare shots in order to visualize the best possible sequence. The creator must decide how to order and select shots from a pool of hundreds or thousands of shots. An ideal storyboard would help her track the best candidate shots, allowing her to easily compare different versions of the same sequence. An ideal storyboard would allow that creator to retain access to the different sequences she has considered, allowing her to instantly compare or recall different possibilities. An ideal storyboard might also allow her to track illustrative footage she has "stolen," and therefore must replace, or footage that must be licensed. She needs a tool that allows her to easily track and compare her different ideas, and perhaps present those ideas to a client. She needs a tool that offers her "what-if?" capabilities, to allow her to quickly and easily see how shot changes affect the tempo and feel of a sequence.

A creator planning a traditional narrative sequence might also find a storyboarding capability useful. A dramatic scene may be captured and visually communicated in a number of different ways: with different camera placements, different camera angles, different lighting, etc. It might also be useful to be able to quickly compare different realization possibilities, and to experiment with "what-if" variations in shot selections and orderings. The same narrative scene edited and presented in different ways can suggest different feelings and interpretations. An example might be having both "R" and "PG" versions of a particular scene available, either one of which may be appropriate depending on later decisions.

Applications Survey

With the introduction of Apple Computer's QuickTime™ digital video enablement standard, the technology necessary to implement new ideas for a digital video editing process is now easily available. Two companies, Adobe Systems Inc. and DiVA Corporation, currently offer full-process digital video editing applications. This section will briefly examine each application, commenting on their approach to the editing process and evaluating their storyboarding abilities.

Adobe Premiere

Premiere, attempting to leverage off of the familiar mechanics of analog video editing, merely transplants the analog video editing process to the computer. So, after recording digital video footage on her computer, a video editor would order footage on her "A-deck" and "B-deck" to produce a final movie on her "record deck." This approach has an obvious advantage: people familiar with the analog video editing process immediately feel comfortable working with digital video. Digital video becomes another form of analog video: a little faster to edit with, perhaps, and stored on computer disks instead of conventional video tapes, but used in exactly the same way.

This approach however also has numerous disadvantages, all of which arise from the fact that we are treating digital video like analog video. The analog video metaphor used in *Premiere* ties us to the peculiar limitations and constraints of analog video editing. In analog editing, for example, we have two source decks to allow us to insert special effects like transition graphics or dissolves between two shots — two source decks are physically necessary to produce these effects. In digital video, however, we can merely specify the ordering of two shots and have the computer add effects at any time — we have different real-time performance or hardware requirements. What does it mean in *Premiere* when we have an "A-deck," "B-deck," and "record deck?" These elements are interface "sugar" that obscure the real dynamics of what we are doing. We hide the capabilities of digital video, resulting in a process that mirrors the analog video process, and results in movies which are no different than analog video movies. *Premiere*'s weaknesses therefore aren't visible until we consider DiVA's *Videoshop*, which breaks new ground in the possibilities of digital video.

DiVA Videoshop

Videoshop practises a radically different philosophy than *Premiere*, preferring to re-evaluate the entire editing process. *Videoshop*, unsurprisingly, breaks down the editing process into the four stages mentioned earlier, and attempts novel, powerful approaches to each.

Videoshop addresses the first stage, video acquisition, by providing a straightforward video-capture capability. Its approach to the second stage, video management, is significantly more interesting. *Videoshop* creates a video-management environment it calls the "visual desktop," an environment that is actually a special copy of the "Macintosh Finder," the standard file-management system. The visual desktop allows users to arrange their video in clips, annotate them with key words and general information, and use powerful string and word-association searches to track down interesting clips. Users can "browse" through the desktop by looking at "micons," or animated previews, of all the available clips. The new organizational metaphor emphasizes and exploits a significant new characteristic of digital video: digital video is just another computer data type, like text or pictures. All of the powerful computer-based techniques for information management and retrieval can now also be extended to digital video clips.

Videoshop also treats video clips as objects. Rather than thinking of putting video clips "into" some video "play deck" and recording on a "record deck," *Videoshop* encourages thinking of video as objects that can be manipulated and arranged. A video editor that wishes to create a sequence would "grab" and "drag" a video clip into a "sequencer," a computer window that allows visual reordering of video clips. On a gross sequencing level, an editor visually reorders clips on a timeline in front of her — a process that is much closer to what is really occurring on a conceptual level than "A-decks" and "record decks."

Videoshop addresses fine editing in a novel fashion, too. Users treat video much like they treat text in a word-processor: video can be "cut, copied, and pasted" by direct selection. A user can add special effects like filters or transition effects by directly selecting the video to alter and ordering the computer to apply an effect. In *Premiere,* to contrast, a user would place an effect between the "A-deck" and "B-deck" to simulate an analog "video effects generator" — interface baggage that obscures the direct-manipulation qualities of *Videoshop*'s approach.

Movie delivery has also been rethought. Rather than limiting movie delivery to the single-screen format of analog video, *Videoshop* allows movie creators to think in term of movie "tracks." A movie can have multiple motion-picture streams displaying at different sizes and at different locations on a screen: we can now exploit the ability of computers to scale and move motion-pictures in real-time. Traditional analog video only offers single-track, single-size movies, a limitation that *Premiere* also retains because of its total reliance on the analog video metaphor. Working in *Videoshop* highlights the impressive capabilities of digital video: we experience an editing process that is dramatically different and potentially more powerful than the old analog video process. We begin to see the possibilities that digital video offers.

Storyboarding

Neither application addresses storyboarding very satisfactorily. *Premiere* avoids the capability entirely, instead concentrating solely on fine, detailed editing work. *Videoshop* offers a nominal shot-ordering capability, but also concentrates more on other issues, like fine editing, clip management, and clip presentation.

Functional Specification

The experimental application written for this thesis attempts to address some of the storyboarding deficiencies of *Premiere* and *Videoshop*. See Figure 1 for a picture of the application. This section examines the features of the application and discusses their importance as storyboarding tools. The functionality of the application falls under two sections: the visual bins and the storyboard sequencer.

The Visual Bins

The "visual bins" address "stage 2" of Brøndmo's four-stage editing model: the bins allow a sequence creator to organize and annotate the shot pool that he or she will work with. See Figure 2 for a picture of the visual bins.

The visual bins provide the functionality that allows a creator to maintain control over the shot pool: they allow a creator to quickly identify interesting shots. They are the tool that allow a creator to work efficiently with a shot pool that is probably too large to memorize or be closely familiar with. The ability to organize the shot pool is an important feature of the application: it allows creators to maintain a sequencing environment that maximizes creative efficiency.

The bins offer a succinct and elegant way of getting a lot of information about a clip,

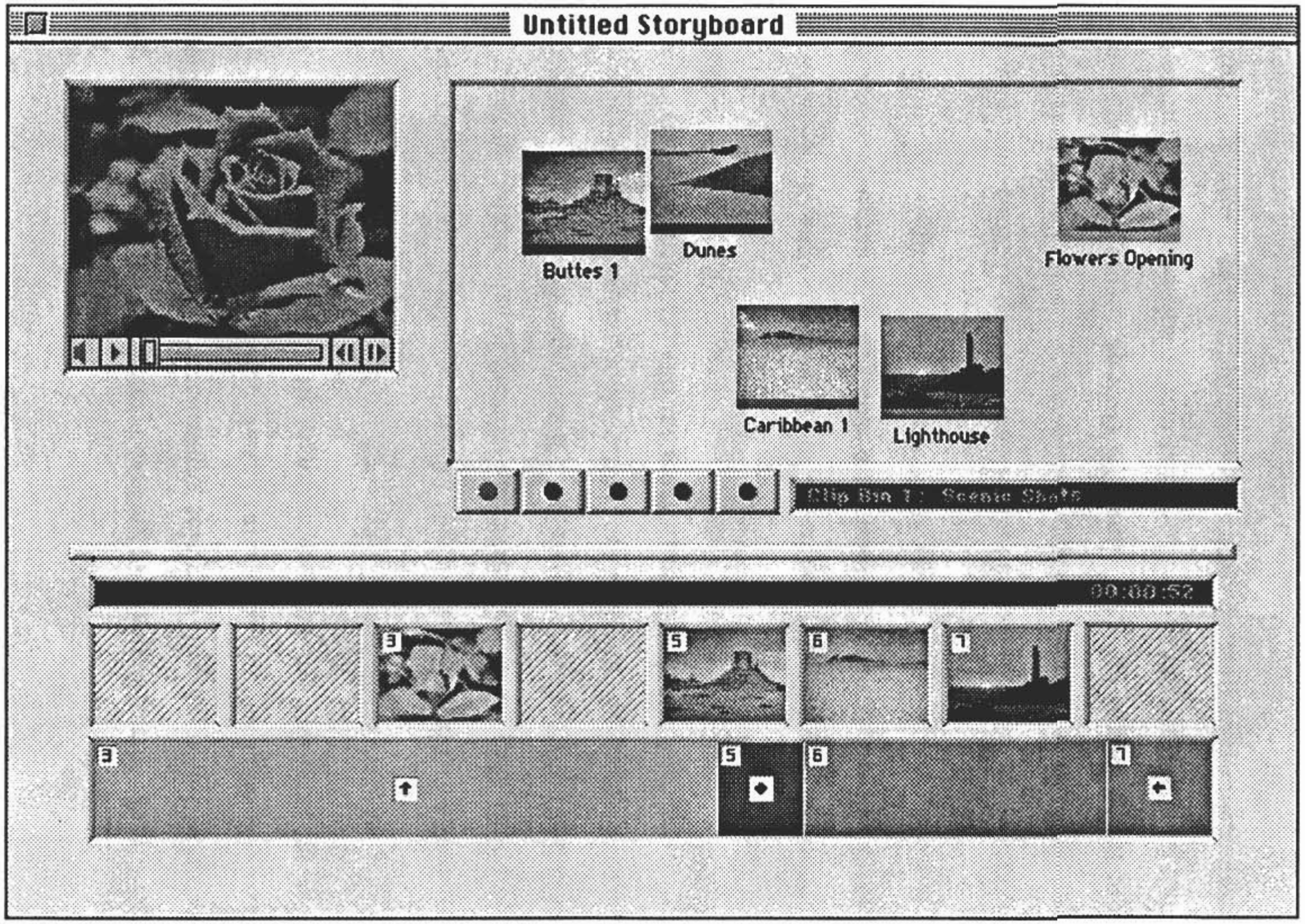
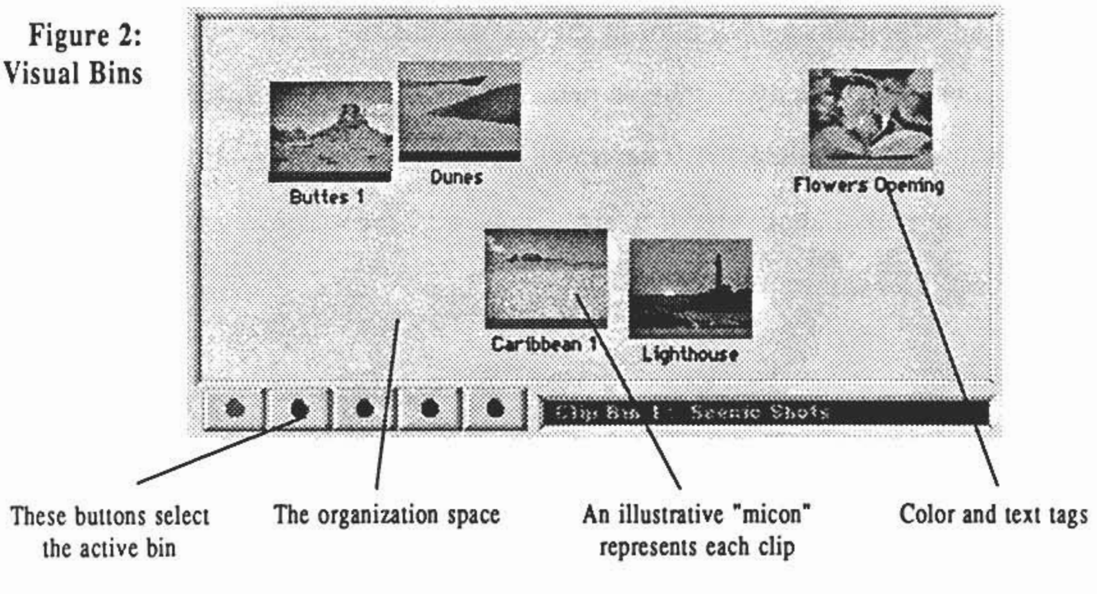


Figure 1: The Storyboarder Window

without actually seeing the clip. They provide immediate information about a clip in five ways: bin number, micon, spatial location, text tag, and color tag. The application offers an arbitrary number of "bins," or top-level repositories, for movie clips. A user can provide contextual information about a clip by placing it in a particular bin: for example, a movie clip found in the "Winter Sports" bin could be assumed to be about "Winter Sports." Movie clips themselves are represented by "micons," or short, animated, and thumbnail-sized previews of the full movie clip. A user can therefore see a quick preview of the actual clip, giving them an immensely illustrative glimpse at the content of the whole clip. Micons can be arbitrarily arranged on the screen, within their bin, allowing users to associate more closely-related clips by placing them together in groups on the screen. Users can reposition a micon by simply "dragging" the micon to its new location. Creators can select to have text tags display under micons, choosing from name, duration, or copyright tags. Creators can also have color-coded duration tags displayed under micons: the application draws a blue-colored bar under the micons to represent different duration lengths — the longer the clip, the lighter blue used.

Figure 2:
The Visual Bins



A user can now easily identify shot candidates. For example, consider a person wishing to find a good clip showing a person skiing. We might have a bin called "winter sports," in which different winter sport clips are spatially organized by sport type. Assuming a mechanism to locate interesting bins, a person can — by simply looking at a bin — visually identify micons whose clips show potential. The person, by looking at the micon tags, might also be able to visually confirm the copyrights and relative lengths of the clips. In *Premiere,* to contrast, a person would just have a single long, unorganized list of shots, with each shot entry showing a picture of the shot's first frame and a list of text fields. A *Premiere* user would have to scroll through a long list of shots and read lines of information — information that is more succinctly communicated visually in the test application.

The bins also allow a user to annotate a clip with more detailed information. Each shot can have profile information stored with it, such as camera movements and general information. The information is stored as part of the clip itself, and so is available by other programs that follow the proper QuickTime™ interface.

The Storyboard Sequencer

The storyboard sequencer is where the actual sequence layout is done. After a user collects and organizes a shot pool in the visual bins, he or she can "drag" shots into the sequencer working area. The sequencer allows a user to specify the organization and content of a sequence. This section discusses the major features of the sequencer: organization, active shot selection, storylines, and the summary window. See Figure 3 for a picture of the sequencer.

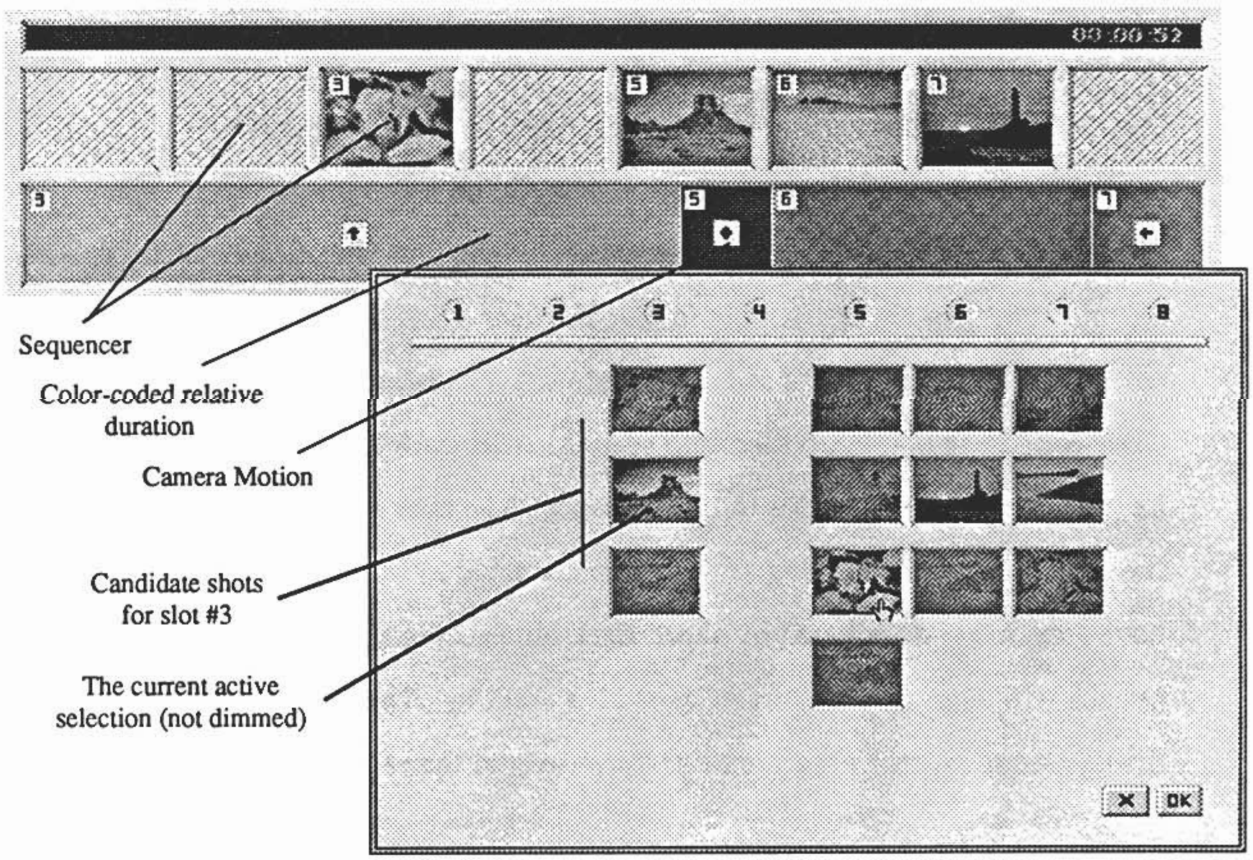
Organization

The sequencer is organized by "shot-slots." A sequence is broken down into a series of "slots," each of which represents one shot in the sequence. A 6-shot montage sequence then would have six "slots" that one could "insert" shots into. The slots can be reordered by dragging a slot to a new location: so slot six could be moved to the slot one location by merely dragging the graphic representation of slot six to slot one.

Active Shot Selection

A user may drag an arbitrary number of shots from the visual bins to a slot. For example, a user may have a number of candidate shots that he or she is considering opening a sequence with; the user therefore drags all of them to the slot one position. The final sequence can only have one of those shots actually shown as the first shot, so the user must select which of the candidate shots is currently "active" and part of the sequence. The active shot is selected from a special dialogue window that displays all of the candidate shots for all of the slots of the sequence. The user can

Figure 3: The Storyboard Sequencer and Active Shot Selection Dialogue


00:00:52
Sequencer
Color-coded relative duration
Camera Motion
Candidate shots for slot #3
The current active selection (not dimmed)
1
2
3
4
5
6
7
8
OK

visually compare and choose which shot she wishes currently active. This active shot system allows a user to maintain and organize all of her candidate shots, and allows her to delay the final decision of which shot to finally use. It allows her to quickly change and compare how a sequence looks with different candidate shots.

Storylines

The application also provides the ability to "freeze" and track different sequence possibilities. A user, while experimenting with different active shots, may find a number of sequence constructions that show a high amount of potential. The user can mark each candidate sequence as a different "storyline:" storylines provide the ability to quickly jump back and forth between different sequence layouts. Storylines complement active shots: together, they allow a user to experiment and track an arbitrary number of shot candidates and sequence candidates, and they allow a user to delay final, binding creative decisions until all of the sequence possibilities have been explored.

Summary Window

One of the interface challenges of digital video, as we saw with the visual bins, is to communicate the time-axis information of a clip on a static computer display. The bins attempted to provide time information by showing duration color tags and using micons. The sequencer attempts to give the user a feel for the temporal qualities of a sequence by providing a "summary window." The summary window displays a color-coded bar that is divided into sections, with each section showing the relative lengths of the different clips in the sequence. The user can get a feel for the tempo of the sequence by just looking at the summary window: a quick, staccato sequence will have a color bar dominated by short, dark-blue sections, while a slow, languid sequence will have a lot of longer, light-blue sections. The summary window will also display icons representing the camera motions of the sequence. The user can visually confirm the tempo and dynamics of the sequence with a simple glance.

Integrating the Storyboarding Stage

Final layouts of sequences can be saved as "referenced" movies, or movies that contain pointers to other files that contain the actual movie image information. Users can then edit these referenced movies in fine-editing programs like *Premiere*. The test application can be smoothly integrated among a suite of movie-creation tools, allowing a user to mix-and-match among the programs for different tasks.

Summary

The application program provides a creative platform for storyboarding development by emphasizing two ideas: the elegant communication of shot/sequence information and the delay of final, binding decisions. Together, they allow a user to quickly identify shots for a sequence and to easily experiment and compare different layouts of sequences.

Technical Specification

This section outlines the source code organization of the application program and discusses some of the engineering design decisions made in implementing the program.

The Application Shell

The core of any Macintosh program is a generic application shell, a base-level program that sets up the application and monitors for user- or computer-generated events. The application shell used by this thesis was first used in MoviePads, a movie clip organization tool similar to the visual bins of this thesis's program. MoviePads was written during the summer of 1992 as an Interactive Cinema Group UROP project. The application shell initializes various Macintosh toolbox managers, polls the Event Manager for events, and responds to events by calling the appropriate handlers.

<u>The Menus Library</u>

Another important component of the program was the "menus library," a set of routines that handled all of the GUI requirements for menus. The menus library, together with the application shell, form a generic program from which to expand and spin off any specialized program.

<u>The Storyboarder Library</u>

All of the functionality of the storyboarding application is stored as a set of routines in a separate library, so the rich function set of the storyboarder is completely separate from the generic program code described earlier. This provides the important ability to easily recycle minor or major portions of the storyboarder in other programs.

Movie Data Structures

A major component of the storyboarder library is the movie clip management system. The library header file specifies specialized element structures and operator macros that simplify the tracking of important movie clip information, such as: file reference information, movie-box location, controller information, etc. All movie references are made indirectly through these data elements.

The Visual Bins

The visual bins sub-library is responsible for maintaining and supporting the functionality of the visual bins. The code is set up to support an arbitrary number of visual bins, with bins of arbitrary sizes and locations. The code could be re-used to support any clip organization-space with the same functionality requirements and interface mechanics of the visual bins used by the storyboarder.

The Sequencer

The sequencer sub-library is responsible for maintaining and supporting the functionality of the storyboard sequencer. The sequencer code, like the visual bins code, has been designed to support a maximum amount of flexibility. The code can support an arbitrarily-large sequence of clips, of arbitrary size and arrangement, and is completely autonomous — allowing other programs that wish to have a sequencing ability to easily "borrow" and use this code.

The Movie-Playout library

The final major component of the storyboarder library is the movie-playout library. This library handles playing out a separate movie window when a user specifies "play movie" or double-clicks on a visual bins micon. Any application wishing to generate QuickTime™ movies with separate windows and controllers can simply call appropriate movie-playout routines.

Further Ideas

This section explores some further storyboarding ideas or issues that were not implemented in the test application:

- Numerous people complained about the two-step active-shot-selection process. In the test application, a user must select the active shot of a slot by choosing from a separate dialogue window. It might be useful to experiment with a different interface that allowed that selection directly on the sequencer itself.
- The current application has a summary window that responds to user selections in the sequencer. The ability to *reverse* that relationship might provide a powerful "computer advice" tool. For example, after dragging in a set of candidate shots to the sequencer, a user might find it difficult to find a good set of active shots. To save time, the user might wish to specify the tempo and camera motions of a desirable sequence directly in the summary window itself, with the computer then

selecting the "closest-match" shots in the sequence. The computer, instead of describing a sequence a user created, would create a sequence the user described.

- Further elaboration on the summary window might also be interesting. Could the computer provide useful critiques of a sequence by analyzing its tempo and dynamics characteristics? It might be interesting if a user could specify certain "model sequences," other sequences that are similar to the one the user intends to create. The computer could critique the user's work-in-progress by comparing it to the model sequences: it might recommend shorter cuts, different camera motions, different lighting, different colors, etc.

Conclusion

Digital video, as we've seen, brings a new opportunity: an opportunity to rethink the movie creation process. Its wide flexibility and unique features suggest that movie-creators have a new power to explore and focus their creative impulses, rather than worry about the constraints and technical features of the physical editing process. Commercial products like *Videoshop* show that movie-creation can be done in freshly different ways, and with great success. The experimental application of this thesis proves that we can start addressing the movie creation process on a *creative,* as well as *technical,* basis. The movie creation process can now include electronic tools that not only make detailed editing easier, but also enhance the creative process by assisting movie-creators as they attempt to visualize their stories.

Further Reading

- Brøndmo, Hans Peter, and Davenport, Glorianna, "Creating and Viewing the Elastic Charles — A Hypermedia Journal," in McAlesse, Ray, and Green, Catherine, Hypertext: State of the ART, Intellect, 1990.
- Harber, Jonathan, and Mayo, Daniel, DiVA Videoshop Reference Manual, DiVA Corporation, 1991.
- Mark, Dave, and Reed, Cartwright, Macintosh C Programming Primer, Addison-Wesley Publishing Co., 1992.

Appendix A: Source Code Samples

```
/***********************************************************************************

    This file contains representative coding excerpts from the thesis
    test application.

 **********************************************************************************/

#include "quicksilver.h"
#include "error.h"

/* general globals */
WindowPtr       gSequencerWin=NIL_POINTER;  /* window ID of the sequencer window */

/* clip bin globals */
BinHandle       gBins[NUMBER_BINS];      /* array of visual bin info */
short           gCurrentBin,             /* bin currently being displayed */
                gTextTag,gColorTag;      /* text/color tag codes (for micon display) */
Movie           gActiveMicon,            /* the currently active micon */
                gSoundClip;              /* SOUND HACK */

/* sequencer globals */
ShotHandle      gShots[NUMBER_SHOTS];    /* array of sequence shot info */
short           gShotRankToIndex[NUMBER_SHOTS], /* specifies the sequencer order of
                                                   gShots (which goes 1st, 2nd, etc.)
                gStoryScrollOffset;      /* index of the 1st shot displayed in the seq
short           gActiveStoryline;
Str255          gStorylineNames[NUM_STORYLINES];
Boolean         gShowShotNumbers,gShowShotCam;

/* movie player globals */
Movie           gStoryMovie;
MovieController gStoryController;


/* handle events ****************************************************************/

Boolean HandleSequencerEvent(EventRecord theEvent, Boolean *alterMenus,Rect dragRect)
{
    WindowPtr       frontWindow;
    Boolean         eventHandled;

    frontWindow = FrontWindow();
    eventHandled = TRUE;
    *alterMenus = FALSE;

    /* check if one of the associated movie controller windows is the guy; if
       so, pass the event to it and let it take care of it */
    if (HandleMovieWinEvent(theEvent,dragRect))
        return(eventHandled);

    /* check if the story movie controller is being used */
    if (gStoryController)
        if (MCIsPlayerEvent(gStoryController,&theEvent))
```

```
            return(eventHandled);

    switch (theEvent.what)
    {
        case nullEvent:
            if (frontWindow==gSequencerWin)
                HandleSequencerNullEvent(theEvent);
            eventHandled = FALSE;
            break;
        case mouseDown:
            eventHandled = HandleSequencerMouseDown(theEvent,dragRect);
            *alterMenus = TRUE;
            break;
        case updateEvt:
            if (((WindowPtr) theEvent.message)==gSequencerWin)
                UpdateSequencer();
            else
                eventHandled = FALSE;
            break;
        case activateEvt:
            if ((theEvent.modifiers & activeFlag) == ACTIVATING)    {
                if (frontWindow==gSequencerWin)
                    ActivateSequencer();
                else
                    eventHandled = FALSE;
            }
            else
                if (frontWindow==gSequencerWin)
                    DeactivateSequencer();
                else
                    eventHandled = FALSE;
            break;
        default:
            eventHandled = FALSE;
            break;
    }

    return(eventHandled);
}

void HandleSequencerNullEvent(EventRecord theEvent)
{
    Point           mouseLoc;
    MoovHandle      moovNode;
    short           shotRank;
    ControlHandle   dummyControl;

    /* service micon */
    if (gActiveMicon!=NULL)
        if (IsMovieDone(gActiveMicon))
            GoToBeginningOfMovie(gActiveMicon);

    MoviesTask(NIL,NIL);
    /* update cursor to grabber if necessary */
    if (gSequencerWin)   {
        /* first check if cursor is above a clip bin micon */
        mouseLoc = theEvent.where;
        GlobalToLocal(&mouseLoc);
        moovNode = LocateMoovNode(mouseLoc);
        if (moovNode)
            HandCursorOpen();
        else    {
            /*  now check if in a storyboard frame */
            if ((shotRank = FindShotRank(mouseLoc))!=NO_SHOT)    {
                if (ShotRankNumberClips(shotRank)) HandCursorOpen();
            }
            else    {
                /* check if over a control button */
                FindControl(mouseLoc,gSequencerWin,&dummyControl);
                if (dummyControl)
                    HandCursorPoint();
                else
                    /* isn't in anywhere!! */
                    PointerCursor();
            }
        }
    }
}

Boolean HandleSequencerMouseDown(EventRecord theEvent, Rect dragRect)
{
    WindowPtr        theWindow;
    short            thePart;
    Rect             dummyRect;          /* inDrag/inGrow */
    ControlHandle    theControl;         /* inContent: controlID of clicked control */
    short            newHeight,          /* inGrow */
                     newWidth;
    long int         windSize;
    Point            mouseLoc;           /* a bunch: local coordinate mouse click */
    Boolean          eventHandled;

    /* verify that pad window was clicked */
    thePart = FindWindow (theEvent.where, &theWindow);

    /* make sure we have a sequencer */
    if (theWindow!=gSequencerWin) return (FALSE);

    SetPort(theWindow);

    eventHandled = FALSE;
    mouseLoc = theEvent.where;
    GlobalToLocal(&mouseLoc);                      /* mouse loc in local coords */

    switch (thePart)
    {
        case inDrag:
            DragWindow(theWindow,theEvent.where,&dragRect);
            eventHandled = TRUE;
            break;
        case inContent:
            eventHandled = TRUE;
            /* click on inactive pad? */
            if (FrontWindow()!=theWindow)    {
                SelectWindow(theWindow);
                break;
            }
            /* click in control? */
            FindControl(mouseLoc,theWindow,&theControl);
            if (theControl!=NIL)    {
                HandleSequencerControl(theControl,theEvent.where);
                break;
            }
            /* other */
            HandleMouseDownEvent(mouseLoc);
            break;
        case inGrow:
            break;
        case inGoAway:
            /* return false and let the main shell.c handle the close-window */
            break;
        case inZoomIn:
        case inZoomOut:
            break;
    }
    return(eventHandled);
}

void HandleSequencerControl(ControlHandle theControl,Point mouseLoc)
{
    Str255      buttonName;
    long        buttonID;
    Rect        dummyRect;

    GetCTitle(theControl,buttonName);
    StringToNum(buttonName,&buttonID);

    switch ((int) buttonID)
    {
        case BIN_BUTTON_REF:
        case BIN_BUTTON_REF+1:
        case BIN_BUTTON_REF+2:
        case BIN_BUTTON_REF+3:
        case BIN_BUTTON_REF+4:
            if (TrackControl(theControl,mouseLoc,NIL)!=NIL) {
                /* draw the hilited button */
                HiliteBinButton(buttonID);
                /*  unhilite the old hilited button */
                if (gCurrentBin!=buttonID) {
                    /* erase the old hilited button */
                    theControl = (*gBins[gCurrentBin])->button;
                    Draw1Control(theControl);
                    /* update button marker */
                    gCurrentBin = buttonID;
                    /* disable current micon */
                    SetActiveMicon(CLEAR_MICON);
                    /* update bin and led graphics */
                    SetRect(&dummyRect,LED_LEFT,LED_TOP,LED_RIGHT,LED_BOTTOM);
                    InvalRect(&dummyRect);
                    SetRect(&dummyRect,BIN_LEFT,BIN_TOP,BIN_RIGHT,BIN_BOTTOM);
                    EraseRect(&dummyRect);
                    InvalRect(&dummyRect);
                }
            }
            else {
                HiliteBinButton(gCurrentBin);
            }
            break;
        default:
            break;
    }
}

/* note mouseLoc is in local coords */
void HandleMouseDownEvent(Point mouseLoc)
{
    MoovHandle  moovNode;
    Movie       movieID;
    short       userAction,shotRank,newshotRank;
    Rect        sourceRect,newRect,tempRect;

    moovNode = LocateMoovNode(mouseLoc);
    movieID = MoovNodeMovieID(moovNode);

    /* check if clicked on a bin micon */
    if (moovNode)   {
        /* else branch on action */
        userAction = UserMouseAction();
        switch (userAction)
        {
            case DRAG_ACTION:
                sourceRect = MoovNodeMiconFrame(moovNode);
                /* set the current micon to the one now being dragged */
                SetActiveMicon(moovNode);
                /* drag it */
                newRect = DragObject(gActiveMicon,NIL);
                /* ensure that still in bin rect */
                SetRect(&tempRect,BIN_LEFT,BIN_TOP,BIN_RIGHT,BIN_BOTTOM);
                SectRect(&tempRect,&newRect,&tempRect);
                if (!EqualRect(&tempRect,&newRect)) {
                    /*** attempted to drag the movie outside the bin */
                    /* check if dragged to storyboard */
                    RectToCenterPoint(newRect,&mouseLoc);
                    if ((shotRank = FindShotRank(mouseLoc))!=NO_SHOT)    {
                        AddClipToShot(shotRank,moovNode);
                        InvalRect(&newRect);
                        EraseRect(&newRect);
                        SetRect(&newRect,
                            FRAME_LEFT+(FRAME_WIDTH+10)*(shotRank-gStoryScrollOffset),
                            FRAME_TOP,
                            FRAME_LEFT+(FRAME_WIDTH+10)*(shotRank-gStoryScrollOffset)+
                            FRAME_BOTTOM);
                        ComposeStoryboardMovie();
                        SetRect(&tempRect,STORY_LED_RIGHT - 60,STORY_LED_TOP,STORY_LED
                        InvalRect(&tempRect);
                    }
                    /* clean up the dragged new rect */
                    InvalRect(&newRect);
                    EraseRect(&newRect);
                    InvalStoryboardAnalysis();
                    SetMovieBox(gActiveMicon,&sourceRect);
                }
                else    {
                    /* moved movie within the bin, update old location */
                    InvalRect(&sourceRect); /* inval & erase old loc */
                    EraseRect(&sourceRect);
                    DisplayMoovNodeTextLabel(moovNode,TRUE);
                    DisplayMoovNodeColorLabel(moovNode,TRUE);
                    MoovNodeMiconFrame(moovNode) = newRect;
                    DisplayMoovNodeColorLabel(moovNode,NIL);
                    DisplayMoovNodeTextLabel(moovNode,NIL);
                }
                break;
            case SINGLE_CLICK:
                BusyCursor();
                SetActiveMicon(moovNode);
                PointerCursor();
                break;
            case DOUBLE_CLICK:
                BusyCursor();
                SetActiveMicon(moovNode);
                AddPlayoutWindow();
                PointerCursor();
                break;
        }
    }
    /* check if clicked in a sequence slot */
    else if ((shotRank = FindShotRank(mouseLoc))!=NO_SHOT)   {
        if (ShotRankNumberClips(shotRank)==0) return;
        /* else branch on action */
        userAction = UserMouseAction();
        if (userAction==DRAG_ACTION)    {
            /* drag picon */
            newRect = DragObject(NIL,ShotRankPicon(shotRank));
            /* see to which shot frame was dragged to */
            RectToCenterPoint(newRect,&mouseLoc);
            if ((newshotRank = FindShotRank(mouseLoc))!=NO_SHOT)    {
                MoveShot(shotRank,newshotRank);
                if (shotRank>newshotRank)   {
                    InvalStoryboardDisplayFrames(newshotRank,shotRank);
                    InvalStoryboardAnalysis();
                }
                else    {
                    InvalStoryboardDisplayFrames(shotRank,newshotRank);
                    InvalStoryboardAnalysis();
                }
                ComposeStoryboardMovie();
            }
            /* clean up the dragged new rect */
            InvalRect(&newRect);
            EraseRect(&newRect);
        }
    }
    /* didn't click in a sequence slot OR on a bin micon */
    else    {
        if (gActiveMicon)   {
            movieID = gActiveMicon;
            SetActiveMicon(CLEAR_MICON);
            DisplayMoovNodePicon(FindMoovNode(movieID));
        }
    }

}



/* pad init/allocation routines *****************************************************/

void NewSequencer(void)
{
    register short  x;

    /* allocate new sequencer window */
    gSequencerWin = GetNewCWindow(MAIN_WINDOW_ID, NIL_POINTER, MOVE_TO_FRONT);
    if (!gSequencerWin) {
        DisposeHandle(gSequencerWin);
        ErrorDisplay(OOM_SEQ,NIL);
        return;
    }

    /* set as active grafport, show, and select window */
    SetPort(gSequencerWin);
    ShowWindow(gSequencerWin);
    SelectWindow(gSequencerWin);

    /* setup the bins */
    for (x=0; x<NUMBER_BINS; x++)   {
        /* allocate bin */
        gBins[x] = (BinHandle) NewHandle(sizeof(struct Bin));
        if (!gBins[x])  {
            CloseWindow(gSequencerWin);
            ErrorDisplay(OOM_SEQ,NIL);
            return;
        }
        /* allocate matching control */
        BinButtonHandle(x) = GetNewControl(BIN_BUTTON_ID+x,gSequencerWin);
        if (!BinButtonHandle(x))    {
            CloseWindow(gSequencerWin);
            ErrorDisplay(OOM_SEQ,NIL);
            return;

        }
        /* set bin name */
        BinNameHandle(x) = NewString("\pdummy string");
        SetHandleSize(BinNameHandle(x),256);
        SetString(BinNameHandle(x),"\pUntitled");
        /* set bin moov list */
        BinMoovNodeList(x) = NIL_POINTER;
    }

    /* setup the storyboard */
    InitStoryboard();

    /* setup storylines */
    gActiveStoryline = 0; /*
    gStorylineNames[0] = STORYLINE_0;
    gStorylineNames[1] = STORYLINE_1;
    gStorylineNames[2] = STORYLINE_2;
    gStorylineNames[3] = STORYLINE_3;
    gStorylineNames[4] = STORYLINE_4; */

    /* set quickdraw params */
    TextSize(9);

    /* init other vars */
    gCurrentBin = 0;
    gActiveMicon = NIL;
    gTextTag = NO_TAG;
    gColorTag = NO_TAG;
    gShowShotNumbers = FALSE;
    gShowShotCam = FALSE;
    gSoundClip = NIL_POINTER;   /* sound hack */

    InitMovieWinList();
    InitCursorLibrary();
}

void DisposeSequencer (void)
{
    register short x;
    /* deallocate sequencer */
    if (gSequencerWin) {
        /* deallocate controls */
        KillControls(gSequencerWin);

        /* deallocate bin structures */
        for (x=0; x<NUMBER_BINS; x++)   {
            RemoveAllBinMoovNodes(x);
            DisposHandle(BinNameHandle(x));
            DisposHandle(gBins[x]);
        }

        /* deallocate storyboard stuff */
        DisposeStoryboard();

        CloseWindow(gSequencerWin);
    }

    gSequencerWin = NIL_POINTER;
}

void DisposeAll (void)
{
    DisposeSequencer();
}

/* sequencer window routines ******************************************************/

void UpdateSequencer(void)
{
    Rect        boundsRect;
    GrafPtr     oldPort;

    GetPort(&oldPort);
    SetPort(gSequencerWin);

    BeginUpdate(gSequencerWin);

        /* update graphics */
        PaintSequencer();
        ShowMoovNodeMicons(gCurrentBin);

        /* show controls */
        UpdtControl(gSequencerWin,gSequencerWin->visRgn);
        HiliteBinButton(gCurrentBin);

        /* update sequencer (done last cuz so slow) */
        DisplayStoryboardPicons();
        DisplayStoryboardAnalysis();

    EndUpdate(gSequencerWin);

    SetPort(oldPort);
```

```
}

void PaintSequencer(void)
{
    Rect        dRect;
    CIconHandle dCIcon;
    short       x;
    int         frameLeft;

    RGBColor    dColor;
    char        ledDisplay[1],seqMovieDuration[9];
    Str255      storylineName;

    /*** draw bin area */
    DrawBeveledFrame(BIN_TOP,BIN_LEFT,BIN_BOTTOM,BIN_RIGHT);

    /*** draw clip bin LED area and update display*/
    DrawLED(LED_TOP,LED_LEFT,LED_BOTTOM,LED_RIGHT);
    /* update LED */
    dColor.red = 10800;
    dColor.green = 35500;
    dColor.blue = 20900;
    RGBForeColor(&dColor);
    TextFace (bold);
    MoveTo(LED_LEFT+6,LED_TOP+9);
    DrawString("\pClip Bin ");
    NumToString(gCurrentBin+1,ledDisplay);
    DrawString(ledDisplay);
    DrawString("\p:  ");
    DrawString((char *) *BinNameHandle(gCurrentBin));

    /*** draw movie area */
    DrawBeveledFrame(MOVIE_TOP,MOVIE_LEFT,MOVIE_BOTTOM+MC_HEIGHT,MOVIE_RIGHT);

    /*** set off sequencing area */
    DrawBeveledBar(BAR_LEFT,BAR_TOP,BAR_RIGHT,BAR_BOTTOM);

    /* draw sequencer LED */
    DrawLED(STORY_LED_TOP,STORY_LED_LEFT,STORY_LED_BOTTOM,STORY_LED_RIGHT);
    GetMovieDurationString(gStoryMovie,seqMovieDuration);
    TextFace (bold);
    MoveTo(STORY_LED_RIGHT - 60,STORY_LED_TOP + 9);
    DrawString(seqMovieDuration);
    MoveTo(STORY_LED_LEFT+6,STORY_LED_TOP+9);
    TextFace (0);

    /* draw frames */
    for (x=0,frameLeft=FRAME_LEFT; x<NUMBER_PHYSICAL_FRAMES; x++,frameLeft+=FRAME_WIDT
        DrawBeveledFrame(FRAME_TOP,frameLeft,FRAME_BOTTOM,frameLeft+FRAME_WIDTH);

    /* draw analysis frame */
    DrawBeveledFrame(ANALYSIS_TOP,ANALYSIS_LEFT,ANALYSIS_BOTTOM,ANALYSIS_RIGHT);
    /* restore fore/back colors to ensure proper drawing */
    RGBForeColor(black);
}

void ShowMoovNodeMicons(short binNum)
{
    MoovHandle  moovList;

    moovList = BinMoovNodeList(binNum);
    while (moovList)    {
        DisplayMoovNodePicon(moovList);
        DisplayMoovNodeTextLabel(moovList,NIL);
        DisplayMoovNodeColorLabel(moovList,NIL);
        moovList = MoovNodeNext(moovList);
    }
}

void DrawLED(int top,int left,int bottom,int right)
{
    RGBColor    dColor,curFore;
    Rect        dRect;

    DrawBeveledFrame(top,left,bottom,right);

    GetForeColor(&curFore);

    /* draw dark rect */
    dColor.red = 10000;
    dColor.green = 5000;
    dColor.blue = 10000;
    RGBForeColor(&dColor);
    SetRect(&dRect,left,top,right,bottom);
    PaintRect(&dRect);

    /* restore fore colors to ensure proper drawing */
    RGBForeColor(&curFore);
}

void DrawBeveledBar(int left,int top,int right, int bottom)
{
    Rect        dRect;
    CIconHandle dCIcon;

    SetRect(&dRect,left,top,right,bottom );
    dCIcon = GetCIcon(BAR_CICN);
    PlotCIcon(&dRect,dCIcon);
    SetRect(&dRect,left,top,left+32,bottom );
    dCIcon = GetCIcon(LBAR_CICN);
    PlotCIcon(&dRect,dCIcon);
    SetRect(&dRect,right-32,top,right,bottom );
    dCIcon = GetCIcon(RBAR_CICN);
    PlotCIcon(&dRect,dCIcon);
    DisposCIcon(dCIcon);
```

```
}

void DrawBeveledFrame(int top,int left,int bottom,int right)
{
    Rect        dRect;
    CIconHandle dCIcon;

    /*** draw bin frame */
    /* draw left */
    SetRect(&dRect,left-OFFSET_B,top,left,bottom);
    dCIcon = GetCIcon(RIGHT_B);
    PlotCIcon(&dRect,dCIcon);
    DisposCIcon(dCIcon);
    /* draw right */
    SetRect(&dRect,right,top,right+OFFSET_B,bottom);
    dCIcon = GetCIcon(LEFT_B);
    PlotCIcon(&dRect,dCIcon);
    DisposCIcon(dCIcon);
    /* draw top */
    SetRect(&dRect,left,top-OFFSET_B,right,top);
    dCIcon = GetCIcon(BOTTOM_B);
    PlotCIcon(&dRect,dCIcon);
    DisposCIcon(dCIcon);
    /* draw bottom */
    SetRect(&dRect,left,bottom,right,bottom+OFFSET_B);
    dCIcon = GetCIcon(TOP_B);
    PlotCIcon(&dRect,dCIcon);
    DisposCIcon(dCIcon);
    /* draw topleft */
    SetRect(&dRect,left-OFFSET_B,top-OFFSET_B,left,top);
    dCIcon = GetCIcon(BR_B);
    PlotCIcon(&dRect,dCIcon);
    DisposCIcon(dCIcon);
    /* draw topleft */
    SetRect(&dRect,left-OFFSET_B,top-OFFSET_B,left,top);
    dCIcon = GetCIcon(BR_B);
    PlotCIcon(&dRect,dCIcon);
    DisposCIcon(dCIcon);
    /* draw topright */
    SetRect(&dRect,right,top-OFFSET_B,right+OFFSET_B,top);
    dCIcon = GetCIcon(BL_B);
    PlotCIcon(&dRect,dCIcon);
    DisposCIcon(dCIcon);
    /* draw bottomleft */
    SetRect(&dRect,left-OFFSET_B,bottom,left,bottom+OFFSET_B);
    dCIcon = GetCIcon(TR_B);
    PlotCIcon(&dRect,dCIcon);
    DisposCIcon(dCIcon);
    /* draw bottomright */
    SetRect(&dRect,right,bottom,right+OFFSET_B,bottom+OFFSET_B);
    dCIcon = GetCIcon(TL_B);
    PlotCIcon(&dRect,dCIcon);
    DisposCIcon(dCIcon);
```

```
}

void HiliteBinButton(short binID)
{
    Rect        dRect;
    CIconHandle dCIcon;

    /* draw the hilited bin */
    dRect = (*((*gBins[binID])->button))->contrlRect;
    dCIcon = GetCIcon(BUTTON_HILITED_ID);
    PlotCIcon(&dRect,dCIcon);
    DisposCIcon(dCIcon);
}

void InvalSequencerBody(void)
{
    Rect        bodyRect;

    bodyRect = gSequencerWin->portRect;
    EraseRect(&bodyRect);
    InvalRect(&bodyRect);
}

void ActivateSequencer(void)
{

}

void DeactivateSequencer(void)
{

}

/* sequencer storyboard routines ******************************************************
/*
   Notes:
     - note that we refer to "screen-frame indices" and actual "shot indices."  Screen
       frame indices refer to the fact that we can only physically display a certain
       number of frames on the screen, up to NUMBER_FRAMES.  However, we may have more
       actual shots in our sequence than we can actually display.  So while we always
       have 1-to-NUMBER_FRAMES shots displayed on the screen, those shots may not
       actually be shots #1-to-#NUMBER_FRAMES, but may be #(1+gStoryScrollOffset)-to-
       #(NUMBER_FRAMES+gStoryScrollOffset)
     - note that we have two shot arrays.  One is our actual array of shots, "gShots",
       which is our actual collection of shots.  gShots however is not ordered in any
       way.  We instead use a "translation" array that takes a certain ordinal
       positional, like 5 (shot #5), and returns the index into the actual gShots
       array to the data we need.  So to access the 3rd shot, we specify:
       gShots[gShotRankToIndex[3]].
*/

void InitStoryboard(void)
{
    short           x,y;
    Rect            movieRect;

    /*** init for NUMBER_FRAMES shots */
    for (x=0;x<NUMBER_SHOTS;x++)     {
        if (!(gShots[x] = (ShotHandle) NewHandle(sizeof(struct Shot)))) {
            ErrorDisplay(OOM_STORY,NIL);
            return;
        }
        /* init values */
        for (y=0;y<NUM_STORYLINES;y++) ShotClip(x,y) = NIL_POINTER;
        for (y=0;y<NUM_CLIPS_PER_SHOT;y++) ShotMoovNode(x,y) = NIL_POINTER;
        ShotPicon(x) = NIL_POINTER;
        ShotNumberClips(x) = 0;
        ShotEnabled(x) = TRUE;
        gShotRankToIndex[x] = x;
    }
    /*** prepare the display window */
    if (!(gStoryMovie = NewMovie(newMovieActive)))   {
        ErrorDisplay(STORY_MOVIE,NIL);
        return;
    }
    SetMovieGWorld(gStoryMovie,NIL,NIL);
    SetRect(&movieRect,MOVIE_LEFT,MOVIE_TOP,MOVIE_RIGHT,MOVIE_BOTTOM);
    SetMovieBox(gStoryMovie,&movieRect);

    /* init other vars */
    gStoryScrollOffset = 0;
}

void DisposeStoryboard(void)
{
    short x;

    for (x=0;x<NUMBER_SHOTS;x++)     {
        DisposHandle(gShots[x]);
    }
}

void ComposeStoryboardMovie(void)
{
    short         clipIndex,shotIndex,x;
    Movie         shotClip;
    TimeValue     movieDuration,storyMovieTime,
                  oldSelectionTime,oldSelectionDuration;
    Rect          theRect,oldRect;
    RGBColor      dColor;

    movieDuration = GetMovieDuration(gStoryMovie);

    /* clear movie */
    DeleteMovieSegment(gStoryMovie,0,movieDuration);
    if (gStoryController) DisposeMovieController(gStoryController);
    /* iterate through sequencer and add movie */
    SetRect(&theRect,MOVIE_LEFT,MOVIE_TOP,MOVIE_RIGHT,MOVIE_BOTTOM);
    for (x=0;x<NUMBER_SHOTS;x++)    {
        /* get the clip for shot x */
        shotIndex = ShotRankToIndex(x);
        if (!ShotNumberClips(shotIndex)) continue;
        if (!ShotEnabled(shotIndex)) continue;
        clipIndex = ShotActiveClip(shotIndex);
        shotClip = MoovNodeMovieID(ShotMoovNode(shotIndex,clipIndex));
        /* do the paste */
        movieDuration = GetMovieDuration(shotClip);
        storyMovieTime = GetMovieDuration(gStoryMovie);
        GetMovieBox(shotClip,&oldRect);
        SetMovieBox(shotClip,&theRect);
        InsertMovieSegment(shotClip,gStoryMovie,0,movieDuration,storyMovieTime);
        SetMovieBox(shotClip,&oldRect);
    }

    SetRect(&theRect,MOVIE_LEFT,MOVIE_TOP,MOVIE_RIGHT,MOVIE_BOTTOM);

    /* restore fore/back colors to ensure proper drawing */
    RGBForeColor(black);

    /* SOUND HACK */
    if (gSoundClip) {
        AddMovieSelection(gStoryMovie,gSoundClip);
    }

    /* create associated controller */
    SetRect(&theRect,theRect.left,theRect.top,theRect.right,theRect.bottom+MC_HEIGHT);
    gStoryController = NewMovieController(gStoryMovie,&theRect,NIL);
    if (!gStoryController)      {
            ErrorDisplay(STORY_MOVIE,NIL);
            return;
    }
    MCSetControllerBoundsRect(gStoryController,&theRect);
}

/* given a mouseloc, will find which shot contained the loc, else NIL */
short FindShotRank(Point mouseLoc)
{
    short   x,frameLeft;
    Rect    shotFrame;

    for (x=0,frameLeft=FRAME_LEFT; x<NUMBER_PHYSICAL_FRAMES; x++,frameLeft+=FRAME_WIDT
        SetRect(&shotFrame,frameLeft,FRAME_TOP,frameLeft+FRAME_WIDTH,FRAME_BOTTOM);
        if (PtInRect(mouseLoc,&shotFrame))
            return(gStoryScrollOffset+x);
    }
    return(NO_SHOT);
}
```

```
/* will add the passed movie clip (specified by a moovnode created in the visual
   bins) to the specified shot's clip list array */
void AddClipToShot(short shotRank,MoovHandle moovNode)
{
    short shotIndex;

    /* translate from the "nth shot" index to the actual, unordered shot index */
    shotIndex = gShotRankToIndex[shotRank];

    if (ShotNumberClips(shotIndex)==NUM_CLIPS_PER_SHOT) return;
    /* add 1 to the total number of clips */
    ShotNumberClips(shotIndex) += 1;
    /* add the associated moovhandle to the shot's moov array */
    ShotMoovNode(shotIndex,ShotNumberClips(shotIndex)-1) = moovNode;
    /* specify the newly-added clip (the last clip) as the active clip */
    SetShotActiveClip(shotRank,ShotNumberClips(shotIndex)-1);
    /* increment the # of references */
    MoovNodeStoryRefs(moovNode) += 1;
}

/* will make the specified clip the active clip for the specified shot, used by
   AddClipToShot() */
void SetShotActiveClip(short shotRank, short clipIndex)
{
    short         shotIndex;
    PicHandle     picon;
    Movie         movieID;
    MoovHandle    moovNode;

    /* translate from the "nth shot" index to the actual, unordered shot index */
    shotIndex = gShotRankToIndex[shotRank];
    picon = ShotPicon(shotIndex);

    ShotActiveClip(shotIndex) = clipIndex;
    /* update the shot picon */
    if (picon) KillPicture(picon);
    moovNode = ShotMoovNode(shotIndex,clipIndex);
    if (moovNode) {
        movieID = MoovNodeMovieID(moovNode);
        if (movieID) ShotPicon(shotIndex) = GetMoviePosterPict(movieID);
    }
}

void SetActiveStoryline(short storyline)
{
    short         shotRank,shotIndex,clipIndex;
    PicHandle     picon;
    Movie         movieID;
    MoovHandle    moovNode;

    gActiveStoryline = storyline;

    /* update sequence list */
    for (shotRank=0; shotRank<NUMBER_SHOTS; shotRank++) {
        shotIndex = gShotRankToIndex[shotRank];
        clipIndex = ShotActiveClip(shotIndex);
        picon = ShotPicon(shotIndex);

        if (picon) KillPicture(picon);
        moovNode = ShotMoovNode(shotIndex,clipIndex);
        if (moovNode) {
            movieID = MoovNodeMovieID(moovNode);
            if (movieID) ShotPicon(shotIndex) = GetMoviePosterPict(movieID);
        }
    }

    /* remake movie and redisplay frames on sequencer */
    ComposeStoryboardMovie();
    InvalStoryboardDisplayFrames(0,7);
    InvalStoryboardAnalysis();
}

/* this function will reorder a shot. */
void MoveShot(short oldShotRank,short newShotRank)
{
    short x,oldShotIndex;

    oldShotIndex = gShotRankToIndex[oldShotRank];
    /* swap shots and move intermediate shots */
    if (newShotRank<oldShotRank)    {
        for (x=oldShotRank;x>newShotRank;x--)
            gShotRankToIndex[x] = gShotRankToIndex[x-1];
    }
    else    {
        for (x=oldShotRank;x<newShotRank;x++)
            gShotRankToIndex[x] = gShotRankToIndex[x+1];
    }
    gShotRankToIndex[newShotRank] = oldShotIndex;
}

/* will inval the contents of a series of storyboard picon display frames,
   where the passed shot ranks MUST BE of shots ranks that are currently being
   displayed */
void InvalStoryboardDisplayFrames(short startRank,short endRank)
{
    short   startFrameNum,endFrameNum;
    short   screenShotPos,frameLeft,shotIndex;
    Rect    shotFrame;

    startFrameNum = startRank-gStoryScrollOffset;
    endFrameNum = endRank-gStoryScrollOffset;
    for (screenShotPos=startFrameNum,frameLeft=FRAME_LEFT+startFrameNum*(FRAME_WIDTH+1
            screenShotPos<=endFrameNum;
            screenShotPos++,frameLeft+=FRAME_WIDTH+10)   {
        SetRect(&shotFrame,frameLeft,FRAME_TOP,frameLeft+FRAME_WIDTH,FRAME_BOTTOM);
        shotIndex = gShotRankToIndex[gStoryScrollOffset+screenShotPos];
        /* clear the picon frame */
        EraseRect(&shotFrame);
        InvalRect(&shotFrame);
    }
}

/* will display the shot picons for all the visible shots in the storyboard */
void DisplayStoryboardPicons(void)
{
    short       frameNum,frameLeft,shotIndex;
    Rect        shotFrame;
    Pattern     dimPat;
    RGBColor    curFore;

    Rect        numFrame;
    CIconHandle dCIcon;

    GetForeColor(&curFore);

    for (frameNum=0,frameLeft=FRAME_LEFT;
            frameNum<NUMBER_PHYSICAL_FRAMES;
            frameNum++,frameLeft+=FRAME_WIDTH+10)    {
        SetRect(&shotFrame,frameLeft,FRAME_TOP,frameLeft+FRAME_WIDTH,FRAME_BOTTOM);
        /* we have NUMBER_FRAMES display available on the screen, but we may have
           a larger number of actual shots we need to display.  gStoryScrollOffset
           gives us the number of the shot that corresponds to display frame #1.
           So if we have 8 physical display frames, but 10 shots, with shot 3
           as the first shot shown on the screen, then gStoryScrollOffset = 3,
           and the actual shots shown on the screen are gStoryScrollOffset+
           frameNum, where 0<frameNum<NUMBER_FRAMES. */
        shotIndex = gShotRankToIndex[gStoryScrollOffset+frameNum];
        /* display the picon, if there is one */
        if (ShotPicon(shotIndex))    {
            /* draw picon */
            DrawPicture(ShotPicon(shotIndex),&shotFrame);
            /* dim out if not an active clip */
            if (!ShotEnabled(shotIndex))    {
                RGBForeColor(dkGray);
                GetIndPattern (&dimPat,0,26);
                PenPat(dimPat);
                PenMode(patOr);
                PaintRect (&shotFrame);
                RGBForeColor(&curFore);
            }
            /* draw shot num */
            SetRect(&numFrame,shotFrame.left+1,shotFrame.top+1,shotFrame.left+1+SMALL_
                                shotFrame.top+1+SMALL_NUMS_SIZE);
            if (gShowShotNumbers)    {
                dCIcon = GetCIcon(SMALL_NUMS_OFFSET+gStoryScrollOffset+frameNum+1);
                PlotCIcon(&numFrame,dCIcon);
                DisposCIcon(dCIcon);
            }
        }
        else    {
            RGBForeColor(gray);

            GetIndPattern (&dimPat,0,26);
            PenPat(dimPat);
            PaintRect (&shotFrame);

            RGBForeColor(&curFore);
        }
    }

    /* restore default pen pattern and xfer mode */
    GetIndPattern (&dimPat,0,1);
    PenPat(dimPat);
}

void ToggleShotNumberDisplay()
{
    if (gShowShotNumbers)   {
        gShowShotNumbers = FALSE;
        InvalStoryboardDisplayFrames(gStoryScrollOffset,
                                gStoryScrollOffset+NUMBER_PHYSICAL_FRAMES);
        InvalStoryboardAnalysis();
    }
    else {
        gShowShotNumbers = TRUE ;
        InvalStoryboardDisplayFrames(gStoryScrollOffset,
                                gStoryScrollOffset+NUMBER_PHYSICAL_FRAMES);
        InvalStoryboardAnalysis();
    }
}

void ToggleCamDisplay()
{
    if (gShowShotCam)   {
        gShowShotCam = FALSE;
        InvalStoryboardAnalysis();
    }
    else {
        gShowShotCam = TRUE ;
        InvalStoryboardAnalysis();
    }
}

void  InvalStoryboardAnalysis(void)
{
    Rect    theRect;

    SetRect(&theRect,ANALYSIS_LEFT,ANALYSIS_TOP,ANALYSIS_RIGHT,ANALYSIS_BOTTOM);
    InvalRect(&theRect);
}

void DisplayStoryboardAnalysis(void)
```

```
{
    short         clipIndex,shotIndex,x;
    MoovHandle    moovNode;
    Movie         shotClip;
    TimeValue     clipDuration,clipTimeScale,clipSeconds,
                  totalTimeScale,totalDuration,totalSeconds;

    int           left,right;
    Rect          theRect;
    RGBColor      theColor,curFore,curBack;
    Pattern       dimPat;

    Rect          dRect;
    int           colorIconID;
    CIconHandle dCIcon=NIL_POINTER;

    StringHandle    cameraMotion;

    GetForeColor(&curFore);
    GetBackColor(&curBack);

    /* set pattern so that we draw w/ the background color */
    GetIndPattern (&dimPat,0,20);
    PenPat(dimPat);

    /* calculate length of entire sequence */
    totalDuration = GetMovieDuration(gStoryMovie);
    totalTimeScale = GetMovieTimeScale(gStoryMovie);
    totalSeconds = totalDuration/totalTimeScale;    /* convert to seconds */

    /* iterate through sequencer and calculate times.  NOTE: all times converted
       to seconds since there can be different time scales */
    for (x=0,left=right=ANALYSIS_LEFT-1;x<NUMBER_SHOTS;x++,left=right+1)    {
        /* get the clip for shot x */
        shotIndex = ShotRankToIndex(x);
        /* if there are no clips, skip to next shot */
        if (!ShotNumberClips(shotIndex)) continue;
        if (!ShotEnabled(shotIndex)) continue;
        /* get the movie of the active clip */
        clipIndex = ShotActiveClip(shotIndex);
        moovNode = ShotMoovNode(shotIndex,clipIndex);
        shotClip = MoovNodeMovieID(moovNode);

        /* calculate length */
        clipDuration = GetMovieDuration(shotClip);
        clipTimeScale = GetMovieTimeScale(shotClip);
        clipSeconds = clipDuration/clipTimeScale;   /* convert to seconds */

        /* calculate rectangle */
        right = left+((clipSeconds*(ANALYSIS_RIGHT-ANALYSIS_LEFT))/totalSeconds);
        if (right>ANALYSIS_RIGHT) right = ANALYSIS_RIGHT;
        SetRect(&theRect,left,ANALYSIS_TOP,right,ANALYSIS_BOTTOM);
        /* draw rectangle */
        colorIconID = AssignIconToMovieByDuration(shotClip);
        dCIcon = GetCIcon(colorIconID);
        PlotCIcon(&theRect,dCIcon);
        DisposCIcon(dCIcon);
        MoveTo(theRect.right,theRect.top);
        LineTo(theRect.right,theRect.bottom);

        /* draw num id */
        if (gShowShotNumbers)    {
            SetRect(&theRect,theRect.left+1,theRect.top+1,theRect.left+1+SMALL_NUMS_SI
            dCIcon = GetCIcon(SMALL_NUMS_OFFSET+x+1);
            PlotCIcon(&theRect,dCIcon);
            DisposCIcon(dCIcon);
        }

        /* draw cam motions */
        if (gShowShotCam)    {
            cameraMotion = GetMovieCameraMovement(shotClip);
            dCIcon = NIL;
            /* figure out which camera movement and assign the proper cicn */
            if (EqualString((char *) *cameraMotion,LEFT_STRING,FALSE,FALSE))
                dCIcon = GetCIcon(ARROW_LEFT_CICN);
            else if (EqualString((char *) *cameraMotion,RIGHT_STRING,FALSE,FALSE))
                dCIcon = GetCIcon(ARROW_RIGHT_CICN);
            else if (EqualString((char *) *cameraMotion,UP_STRING,FALSE,FALSE))
                dCIcon = GetCIcon(ARROW_UP_CICN);
            else if (EqualString((char *) *cameraMotion,DOWN_STRING,FALSE,FALSE))
                dCIcon = GetCIcon(ARROW_DOWN_CICN);
            else if (EqualString((char *) *cameraMotion,ZOOM_STRING,FALSE,FALSE))
                dCIcon = GetCIcon(ARROW_ZOOM_CICN);
            /* show the cicn */
            if (dCIcon) {
                SetRect(&theRect,left,ANALYSIS_TOP,right,ANALYSIS_BOTTOM);
                SetRect(&theRect, (theRect.left+theRect.right-ARROW_SIZE)/2,
                                  (theRect.top+theRect.bottom-ARROW_SIZE)/2,
                                  (theRect.left+theRect.right+ARROW_SIZE)/2,
                                  (theRect.top+theRect.bottom+ARROW_SIZE)/2);
                PlotCIcon(&theRect,dCIcon);
                DisposCIcon(dCIcon);
            }
        }
    }
    /* cleanup any error */
    if (left<ANALYSIS_RIGHT) {
        SetRect(&theRect,left,ANALYSIS_TOP,ANALYSIS_RIGHT,ANALYSIS_BOTTOM);
        EraseRect(&theRect);
    }

    /* restore fore/back colors to ensure proper drawing */
    GetIndPattern (&dimPat,0,1);
    PenPat(dimPat);
    RGBBackColor(&curBack);
```

```
}

void ArrangeStoryline(short baseShotRank)
{
    WindowPtr    arrangeWin,theWindow;
    RGBColor     curFore,curBack;

    register short   shotRank;
    short            shotIndex,clipIndex,numClipsShots,
                     leftPos,topPos,delta,x;
    Rect             frameRect,dRect,oldRect;
    short            newRankedShotActiveClip[NUMBER_SHOTS];
    Boolean          newEnabledFlags[NUMBER_SHOTS];

    PicHandle        picon;
    Pattern          dimPat;

    EventRecord theEvent;
    Boolean     done;
    Point       mouseLoc;

    long             buttonID;
    ControlHandle    OKButton,XButton,theControl;
    CIconHandle      dCIcon;

    Boolean inButton;

    PointerCursor();
    /* allocate new sequencer window */
    arrangeWin = GetNewCWindow(STORY_FORMATTER_ID, NIL_POINTER, MOVE_TO_FRONT);
    if (!arrangeWin)     {
        DisposeHandle(arrangeWin);
        ErrorDisplay(OOM_ARRANGER,NIL);
        return;
    }

    /* set as active grafport, show, and select window */
    ShowWindow(arrangeWin);
    SelectWindow(arrangeWin);
    SetPort(arrangeWin);

    /* set fore/background colors to ensure clean copybits operation */
    GetForeColor(&curFore);
    GetBackColor(&curBack);
    RGBForeColor(dkGray);

    /* draw cosmetics */
    DrawBeveledBar(BAR2_LEFT,BAR2_TOP,BAR2_RIGHT,BAR2_BOTTOM);
    /* draw circles */
    dCIcon = GetCIcon(CIRCLE_CICN);
    for (shotRank=baseShotRank,leftPos=LEFT_POS+(MICON_SMALL_H/2)-(CIRCLE_WIDTH/2);
            shotRank<baseShotRank+NUMBER_PHYSICAL_FRAMES; shotRank++,leftPos+=MICON_S
        shotIndex = ShotRankToIndex(shotRank);
        if (ShotEnabled(shotIndex))
            dCIcon = GetCIcon(CIRCLE_CICN+shotRank);
        else
            dCIcon = GetCIcon(CIRCLE_DIM_CICN+shotRank);
        SetRect(&dRect,leftPos,BAR2_TOP-CIRCLE_HEIGHT-5,leftPos+CIRCLE_WIDTH,BAR2_TOP-
        PlotCIcon(&dRect,dCIcon);
        DisposCIcon(dCIcon);
    }

    /* draw buttons */
    OKButton=GetNewControl(FOK_BUTTON,arrangeWin);
    XButton=GetNewControl(FX_BUTTON,arrangeWin);
    if (OKButton)   {
        Draw1Control (OKButton);
        Draw1Control (XButton);
    }
    else
        return;

    /* draw a bevel and micon for each clip */
    for (shotRank=baseShotRank,leftPos=LEFT_POS;shotRank<baseShotRank+NUMBER_PHYSICAL_
            shotRank++,leftPos+=MICON_SMALL_H+DELTA_H)  {
        shotIndex = ShotRankToIndex(shotRank);
        numClipsShots = ShotNumberClips(shotIndex);
        for (clipIndex=0,topPos=TOP_POS;clipIndex<numClipsShots;clipIndex++,
                                                                topPos+=MICON_SMALL_V+
            /* draw beveled frame for micon */
            SetPort(arrangeWin);
            DrawBeveledFrame(topPos,leftPos,topPos+MICON_SMALL_V,leftPos+MICON_SMALL_H

            picon = GetMoviePosterPict(MoovNodeMovieID(ShotMoovNode(shotIndex,clipInde
            SetRect(&frameRect,leftPos,topPos,leftPos+MICON_SMALL_H,topPos+MICON_SMALL
            DrawPicture(picon,&frameRect);
            if (picon) KillPicture(picon);

            /* grey out the non-active clips */
            if (ShotActiveClip(shotIndex)!=clipIndex)   {
                GetIndPattern (&dimPat,0,7);
                PenPat(dimPat);
                PenMode (patOr);
                PaintRect ( &frameRect );
                PenMode (patCopy);
            }
        }
    }

    /* init shot active clip list */
    for (shotRank=baseShotRank; shotRank<baseShotRank+NUMBER_PHYSICAL_FRAMES; shotRank
        shotIndex = ShotRankToIndex(shotRank);
        newRankedShotActiveClip[shotRank-baseShotRank] = ShotActiveClip(shotIndex);
    }
    for (shotRank=baseShotRank; shotRank<baseShotRank+NUMBER_PHYSICAL_FRAMES; shotRank
        shotIndex = ShotRankToIndex(shotRank);
        newEnabledFlags[shotRank-baseShotRank] = ShotEnabled(shotIndex);
    }

    SetPort(arrangeWin);
    /* handle events */
    done = FALSE;
    while (!done)   {
        /* get event */
        if ((NGetTrapAddress(WNE_TRAP_NUM,ToolTrap) !=
            NGetTrapAddress(UNIMPL_TRAP_NUM,ToolTrap)))
            WaitNextEvent(everyEvent, &theEvent, SLEEP, NIL_MOUSE_REGION);
        else
        {
            SystemTask();
            GetNextEvent (everyEvent, &theEvent);
        }
        /* handle event */
        switch (theEvent.what)
        {
            case mouseDown:
                mouseLoc = theEvent.where;
                GlobalToLocal(&mouseLoc);
                FindWindow (theEvent.where, &theWindow);
                if (theWindow!=arrangeWin) return;
                /* click in a control? (the X or OK buttons */
                FindControl(mouseLoc,theWindow,&theControl);
                if (theControl!=NIL)    {
                    buttonID = GetCRefCon (theControl);
                    if (buttonID==FOK_ID)   {
                        HiliteControl (OKButton,HILITED );
                        /* update sequence list */
                        for (shotRank=baseShotRank; shotRank<baseShotRank+NUMBER_PHYSI
                                                                                     s
                            SetShotActiveClip(shotRank,newRankedShotActiveClip[shotRan
                        /* update enabled shot list */
                        for (shotRank=baseShotRank; shotRank<baseShotRank+NUMBER_PHYSI
                                                                              shotRank++
                            shotIndex = ShotRankToIndex(shotRank);
                            ShotEnabled(shotIndex) = newEnabledFlags[shotRank];
                        }
                        /* recompose movie */
                        ComposeStoryboardMovie();
                        /* redraw graphics */
                        InvalStoryboardDisplayFrames(0,7);
                        InvalStoryboardAnalysis();
                        done = TRUE;
                    }
                    else    {
                        HiliteControl (XButton,HILITED );
                        done = TRUE;
                    }
                }
                /* check to see if mouse clicked in a picon frame */
        for (shotRank=baseShotRank,leftPos=LEFT_POS;shotRank<baseShotRank+NUMB
                                                        leftPos+=MICON_SMALL_H
            shotIndex = ShotRankToIndex(shotRank);
            numClipsShots = ShotNumberClips(shotIndex);
            for (clipIndex=0,topPos=TOP_POS;clipIndex<numClipsShots;clipIndex+
                                                        topPos+=MICON_SMALL_V+
                SetRect(&frameRect,leftPos,topPos,leftPos+MICON_SMALL_H,topPos
                if (PtInRect(mouseLoc,&frameRect)) {
                    /* mouse was clicked in this frame! */
                    if (delta=newRankedShotActiveClip[shotRank-baseShotRank]-c
                        /* activate this frame */
                        picon = GetMoviePosterPict(MoovNodeMovieID(ShotMoovNod
                        SetRect(&frameRect,leftPos,topPos,leftPos+MICON_SMALL_
                        DrawPicture(picon,&frameRect);
                        if (picon) KillPicture(picon);
                        /* dim out old active frame */
                        SetRect(&frameRect,leftPos,topPos+delta*(MICON_SMALL_V
                                            leftPos+MICON_SMALL_H,topPos+delta
                                                                +MICON_SMA
                        GetIndPattern (&dimPat,0,7);
                        PenPat(dimPat);
                        PenMode (patOr);
                        PaintRect ( &frameRect );
                        PenMode (patCopy);
                        /* update new active clip list */
                        newRankedShotActiveClip[shotRank-baseShotRank] = clipI
                    }
                }
            }
        }
        /* check to see if mouse clicked on a shot number circle */
        for (shotRank=baseShotRank,leftPos=LEFT_POS+(MICON_SMALL_H/2)-(CIRCLE_
                shotRank<baseShotRank+NUMBER_PHYSICAL_FRAMES; shotRank++,leftP
            SetRect(&dRect,leftPos,BAR2_TOP-CIRCLE_HEIGHT-5,leftPos+CIRCLE_WID
            if (PtInRect(mouseLoc,&dRect)) {
                if (newEnabledFlags[shotRank])      {
                    newEnabledFlags[shotRank] = FALSE;
                    dCIcon = GetCIcon(CIRCLE_DIM_CICN+shotRank);
                }
                else    {
                    newEnabledFlags[shotRank] = TRUE;
                    dCIcon = GetCIcon(CIRCLE_CICN+shotRank);
                }
                PlotCIcon(&dRect,dCIcon);
                DisposCIcon(dCIcon);
            }
        }
        for (shotRank=baseShotRank,leftPos=LEFT_POS;shotRank<baseShotRank+NUMB
                                                        leftPos+=MICON_SMALL_H
            shotIndex = ShotRankToIndex(shotRank);
            numClipsShots = ShotNumberClips(shotIndex);
            for (clipIndex=0,topPos=TOP_POS;clipIndex<numClipsShots;clipIndex+
                                                        topPos+=MICON_SMALL_V+
                SetRect(&frameRect,leftPos,topPos,leftPos+MICON_SMALL_H,topPo
                if (PtInRect(mouseLoc,&frameRect)) {
                    /* mouse was clicked in this frame! */
                    if (delta=newRankedShotActiveClip[shotRank-baseShotRank]-c
                        /* activate this frame */
                        picon = GetMoviePosterPict(MoovNodeMovieID(ShotMoovNod
                        SetRect(&frameRect,leftPos,topPos,leftPos+MICON_SMALL_
                        DrawPicture(picon,&frameRect);
                        if (picon) KillPicture(picon);
                        /* dim out old active frame */
                        SetRect(&frameRect,leftPos,topPos+delta*(MICON_SMALL_V
                                                  leftPos+MICON_SMALL_H,topPos+delta
                                                                        +MICON_SMA
                        GetIndPattern (&dimPat,0,7);
                        PenPat(dimPat);
                        PenMode (patOr);
                        PaintRect ( &frameRect );
                        PenMode (patCopy);
                        /* update new active clip list */
                        newRankedShotActiveClip[shotRank-baseShotRank] = clipI
                    }
                }
            }
        }

        break;
    case updateEvt:
        mouseLoc = theEvent.where;
        GlobalToLocal(&mouseLoc);
        inButton = FALSE;
        /* check to see if mouse within a shot number circle */
        for (shotRank=baseShotRank,leftPos=LEFT_POS+(MICON_SMALL_H/2)-(CIRCLE_
                shotRank<baseShotRank+NUMBER_PHYSICAL_FRAMES; shotRank++,leftP
            SetRect(&dRect,leftPos,BAR2_TOP-CIRCLE_HEIGHT-5,leftPos+CIRCLE_WID
            if (PtInRect(mouseLoc,&dRect))
                inButton = TRUE;
        }
        /* see if mouse w/in a picon frame */
        for (shotRank=baseShotRank,leftPos=LEFT_POS;shotRank<baseShotRank+NUMB
                                                        leftPos+=MICON_SMALL_H+DEL
            shotIndex = ShotRankToIndex(shotRank);
            numClipsShots = ShotNumberClips(shotIndex);
            for (clipIndex=0,topPos=TOP_POS;clipIndex<numClipsShots;clipIndex+
                                                            topPos+=MICON_SMALL_V+
                SetRect(&frameRect,leftPos,topPos,leftPos+MICON_SMALL_H,topPos
                if (PtInRect(mouseLoc,&frameRect))
                    inButton = TRUE;
            }
        }
        if (inButton)
            HandCursorPoint();
        else
            PointerCursor();
```

```
                break;
            case keyDown:
                switch ((theEvent.message)&charCodeMask)
                {
                    case 0x0d:  /* [Return] pressed or … */
                    case 0x03:  /* … [Enter] pressed     */
                        /* hilite OK key
                        GetDItem(theDlg,OK_ITEM,&iType,&button,&iRect); */
                        HiliteControl (OKButton, HILITED ); .
                        /* update sequence list */
                        for (shotRank=baseShotRank; shotRank<baseShotRank+NUMBER_PHYSI
                                                                            shotRank++
                            SetShotActiveClip(shotRank,newRankedShotActiveClip[shotRan
                        /* update enabled shot list */
                        for (shotRank=baseShotRank; shotRank<baseShotRank+NUMBER_PHYSI
                                                                            shotRank++

                            shotIndex = ShotRankToIndex(shotRank);
                            ShotEnabled(shotIndex) = newEnabledFlags[shotRank];
                        }
                        /* recompose movie */
                        ComposeStoryboardMovie();
                        /* redraw graphics */
                        InvalStoryboardDisplayFrames(0,7);
                        InvalStoryboardAnalysis();
                        /* exit */
                        done = TRUE;
                        break;
                    case '.':
                        if (theEvent.modifiers & cmdKey) done = TRUE;
                        HiliteControl (XButton, HILITED );
                        break;
                    case 0x1b:  /* [Esc] pressed */
                        /* hilite Cancel key
                        GetDItem(theDlg,CANCEL_ITEM,&iType,&button,&iRect); */
                        HiliteControl ( XButton, HILITED );
                        /* return cancel key */
                        done = TRUE;
                        break;
                    default:
                        break;
                }
                break;
            default:
                break;
        }
    }

    /* Dispose of data structures */
    CloseWindow(arrangeWin);

    /* restore original state */
    RGBForeColor(&curFore);
    RGBBackColor(&curBack);
```

```
}

/* movie information (user-data & inherent info) routines **************************/

/* routine will get the camera movement string associated with a movie */
StringHandle GetMovieCameraMovement(Movie movieID)
{
    UserData        dataList;
    StringHandle    camMove;

    camMove = NewString("\p<No Entry>");
    SetHandleSize(camMove,256);

    dataList = GetMovieUserData(movieID);
    if (CountUserDataType(dataList,MOTION_DATA))    {
        GetUserDataText(dataList,(Handle) camMove,MOTION_DATA,1,0);
    }

    return(camMove);
}
void SetMovieCameraMovement(Movie movieID,StringHandle camMove)
{
    UserData    dataList;
    OSErr       error;

    dataList = GetMovieUserData(movieID);
    AddUserDataText(dataList,(Handle) camMove,MOTION_DATA,1,0);
}

/* function expects the address of a 9-byte unsigned char array "char durString[9]" */
void GetMovieDurationString(Movie movieID,char *durationString)
{
    TimeValue   movieDuration,movieTimeScale;
    short       durationHours,durationMinutes,durationSeconds;
    char        hours[3],minutes[3],seconds[3];

    movieDuration = GetMovieDuration(movieID);
    movieTimeScale = GetMovieTimeScale(movieID);
    durationSeconds = (movieDuration/movieTimeScale)%60;
    durationMinutes = (((movieDuration/movieTimeScale)-durationSeconds)%3600)/60;
    durationHours = ((movieDuration/movieTimeScale)-durationSeconds-durationMinutes)/3
    NumToString(durationHours,hours);
    if (durationHours<10) {
        hours[2] = hours[1];
        hours[1] = '0';
    }
    NumToString(durationMinutes,minutes);
    if (durationMinutes<10) {
        minutes[2] = minutes[1];
        minutes[1] = '0';
    }
    NumToString(durationSeconds,seconds);
    if (durationSeconds<10) {
            seconds[2] = seconds[1];
            seconds[1] = '0';
        }
        /* set up final duration string */
        durationString[0]=8;
        durationString[1]=hours[1];
        durationString[2]=hours[2];
        durationString[3]=durationString[6]=':';
        durationString[4]=minutes[1];
        durationString[5]=minutes[2];
        durationString[7]=seconds[1];
        durationString[8]=seconds[2];
}

/* sequencer clip bin routines ********************************************************
   these clips constitute the visual bins library **********************************/

/* add a movie, from a passed movie file, to the moov list of the current bin */
void AddClip(StandardFileReply fileInfo)
{
    OSErr           error;
    FSSpec          fileSpec;
    MoovHandle      moovNode;
    short           resRefNum,resID;
    Movie           theMovie;

    /* vars for setting up micon frame */
    Point           centerWindow;
    Rect            modelFrame,defaultFrame,frame;

    /**** init vars */
    fileSpec = fileInfo.sfFile;
    BusyCursor();

    /**** add a new moov node to current bin, and assign it to moovNode */
    if (!AddMoovNodeToCurBin()) {
        ErrorDisplay(OOM_CLIPS,NIL);
        return;
    }
    moovNode = BinMoovNodeList(gCurrentBin);

    /**** open movie file & update moovNode */
    OpenMovieFile(&fileSpec,&resRefNum,0);
    if ((error=GetMoviesError())!=noErr)    {
            /* RemoveMoov(padNode,KILL_FIRST_MOOV,IGNORE_MICON); */
            ErrorDisplay(NIL,error);
            return;
    }
    MoovNodeResRefNum(moovNode) = resRefNum;
    MoovNodeFileInfo(moovNode) = fileInfo;

    /* assign it to movie */
    resID = 0;
    NewMovieFromFile(&theMovie,resRefNum,&resID,NIL,0,NIL);
    if ((error=GetMoviesError())!=noErr)     {
            /* RemoveMoov(padNode,KILL_FIRST_MOOV,IGNORE_MICON); */
            ErrorDisplay(NIL,error);
            return;
    }
    MoovNodeMovieID(moovNode) = theMovie;
    MoovNodeResID(moovNode) = resID;
    /* CloseMovieFile(resRefNum); */

    /* setup miconlocation */
    GetMovieBox(theMovie,&defaultFrame);
    MoovNodeDefaultFrame(moovNode) = defaultFrame;
    SetRect(&frame,BIN_LEFT,BIN_TOP,BIN_RIGHT,BIN_BOTTOM);
    RectToCenterPoint(frame,&centerWindow);
    SetRect(&modelFrame,0,0,MICON_SMALL_H,MICON_SMALL_V);
    CenterRectAboutPoint(centerWindow,modelFrame,&frame);
    MoovNodeMiconFrame(moovNode) = frame;

    /* set as micon */
    SetActiveMicon(moovNode);

    /* clean up */
    PointerCursor();
}

/* THIS IS PART OF MY SOUND HACK */
void SpecifySoundClip(StandardFileReply fileInfo)
{
    OSErr           error;
    FSSpec          fileSpec;
    MoovHandle      moovNode;
    short           resRefNum,resID;
    Movie           theMovie;
    /* vars for setting up micon frame */
    Point           centerWindow;
    Rect            modelFrame,defaultFrame,frame;

    if (!fileInfo.sfGood) {
        gSoundClip = NIL_POINTER;
        return;
    }

    /**** init vars */
    fileSpec = fileInfo.sfFile;
    BusyCursor();

    /**** open movie file & update moovNode */
    OpenMovieFile(&fileSpec,&resRefNum,0);
    if ((error=GetMoviesError())!=noErr)     {
            /* RemoveMoov(padNode,KILL_FIRST_MOOV,IGNORE_MICON); */
            ErrorDisplay(NIL,error);
            return;
```

```
	}

	/* assign it to movie */
	resID = 0;
	NewMovieFromFile(&theMovie,resRefNum,&resID,NIL,0,NIL);
	if ((error=GetMoviesError())!=noErr)	{
		/* RemoveMoov(padNode,KILL_FIRST_MOOV,IGNORE_MICON); */
		ErrorDisplay(NIL,error);
		return;
	}
	/* CloseMovieFile(resRefNum); */

	/* set as micon */
	gSoundClip = theMovie;
	ComposeStoryboardMovie();

	/* clean up */
	PointerCursor();
}

/* add a moovnode to the beginning of the current bin's moov list; return
	TRUE if successful, else FALSE */
Boolean AddMoovNodeToCurBin(void)
{
	MoovHandle	newNode;

	newNode = (MoovHandle) NewHandle(sizeof(struct MoovNode));
	if (!newNode)	{
		ErrorDisplay(OOM,NIL);
		return(FALSE);
	}

	/* place in list */
	MoovNodeNext(newNode) = BinMoovNodeList(gCurrentBin);
	MoovNodePrev(newNode) = NIL_POINTER;
	MoovNodePrev(BinMoovNodeList(gCurrentBin)) = newNode;

	/* field initialization */
	MoovNodeMovieID(newNode) = NIL_POINTER;
	MoovNodeStoryRefs(newNode) = NIL;

	BinMoovNodeList(gCurrentBin) = newNode;
	return(TRUE);
}

/* get the moov node of the current page w/ the passed movieID */
MoovHandle FindMoovNode(Movie movieID)
{
	MoovHandle	moovNode;

	moovNode = BinMoovNodeList(gCurrentBin);
	while (moovNode)
		if (MoovNodeMovieID(moovNode)==movieID)
```

```
            break;
        else
            moovNode = MoovNodeNext(moovNode);

    return(moovNode);
}

MoovHandle LocateMoovNode(Point mouseLoc)
{
    MoovHandle  moovList;
    Rect        boundsRect;

    moovList = BinMoovNodeList(gCurrentBin);

    while (moovList)    {
        boundsRect = MoovNodeMiconFrame(moovList);
        if (PtInRect(mouseLoc,&boundsRect))
            break;
        moovList = MoovNodeNext(moovList);
    }
    return (moovList);
}

/* function will kill the moovnode associated with a movie, or if passed NIL
   as a movie, will kill the first moovnode of the passed bin */
void RemoveMoov(Movie movieID,short binNum,Boolean eraseMicon,Boolean checkReferences
{
    WindowPtr   curPort;
    MoovHandle  moovNode;
    Rect        miconFrame;
    OSErr       error;

    /* if not passed a specific movieID, then just take the first moovnode of the
       current bin, else find the right moovnode that matches the passed movieID */
    if (!movieID)   {
        moovNode = BinMoovNodeList(binNum);
        movieID = MoovNodeMovieID(moovNode);
    }
    else
        moovNode = FindMoovNode(movieID);

    /* couldn't find any moov node? */
    if (!moovNode) {
        ErrorDisplay(MOOVNODE,NIL);
        return;
    }

    /* if the storyboard references this moovnode, then we can't remove the node cuz
       we need to keep the associated movie file open */
    if (checkReferences)    {
        if (MoovNodeStoryRefs(moovNode))    {
            ErrorDisplay(REF_CONFLICT,NIL);
            return;
        }
    }

    /*** detach node from list */
    if (MoovNodePrev(moovNode)!=NIL_POINTER)
        /* case: deleting a moov node that is not the first  */
        MoovNodeNext(MoovNodePrev(moovNode)) = MoovNodeNext(moovNode);
    else
        /* case: deleting the first moov node */
        BinMoovNodeList(binNum) = MoovNodeNext(moovNode);
    if ((*moovNode)->next!=NULL)
        /* case: deleting a moov nod ethat is not the last */
        MoovNodePrev(MoovNodeNext(moovNode)) = MoovNodePrev(moovNode);

    /**** disable micon (if necessary) and erase it */
    if (movieID==gActiveMicon)
        SetActiveMicon(CLEAR_MICON);
    if (eraseMicon) {
        GetPort(&curPort);
        SetPort(gSequencerWin);
        miconFrame = MoovNodeMiconFrame(moovNode);
        EraseRect(&miconFrame);
        InvalRect(&miconFrame);
        SetPort(curPort);
    }
    /**** delete it */
    DisposeMovie(movieID);
    if ((error=GetMoviesError())!=noErr)    {
            ErrorDisplay(NIL,error);
            return;
    }   /*
    CloseMovieFile(MoovNodeResRefNum(moovNode));    */
    DisposHandle(moovNode);
}

/* removes all the moov nodes associated with a bin */
void RemoveAllBinMoovNodes(short binNum)
{
    MoovHandle  moovNode;

    moovNode = BinMoovNodeList(binNum);
    while (moovNode)    {
        moovNode = MoovNodeNext(moovNode);
        RemoveMoov(NIL,binNum,NIL,NIL);
    }
}

/* set/reset the active micon */
void SetActiveMicon(MoovHandle moovNode)
{
    Rect        piconFrame,oldFrame;
    OSErr       error;
    TimeValue    previewTime,previewDuration,prevTimeScale,prevDurationSeconds;

    /*** check if special request */
    /* check if moovNode==NIL, which means no active movie, and stop previous
       active movie if any */
    if (moovNode==CLEAR_MICON)  {
        if (gActiveMicon)   {
            SetMovieActive(gActiveMicon,FALSE);
            StopMovie(gActiveMicon);
            ShowMoviePoster(gActiveMicon);
            SetMoviePreviewMode(gActiveMicon,FALSE);
            SetMovieVolume(gActiveMicon,kFullVolume);
            gActiveMicon = NIL;
        }
        return;
    }
    if (moovNode==SUSPEND_MICON)     {
        if (gActiveMicon)
            SetMovieActive(gActiveMicon,FALSE);
        return;
    }
    if (moovNode==RESUME_MICON) {
        if (gActiveMicon)
            SetMovieActive(gActiveMicon,TRUE);
        return;
    }

    /**** set new active micon */
    if (gActiveMicon==MoovNodeMovieID(moovNode)) return;
    piconFrame = MoovNodeMiconFrame(moovNode);

    /* stop old micon */
    if (gActiveMicon!=NIL)  {
        /* inval old movie frame so it is redrawn */
        GetMovieBox(gActiveMicon,&oldFrame);
        InvalRect(&oldFrame);
        /* disable old active movie (micon) */
        SetMovieActive(gActiveMicon,FALSE);
        StopMovie(gActiveMicon);
        SetMoviePreviewMode(gActiveMicon,FALSE);
        SetMovieVolume(gActiveMicon,kFullVolume);
    }
    gActiveMicon = MoovNodeMovieID(moovNode);

    /**** enable new active movie (micon) */
    /* resize */
    SetMovieBox(gActiveMicon,&piconFrame);
    error = GetMoviesError();
    /* set preview if necessary */
    GetMoviePreviewTime(gActiveMicon,&previewTime,&previewDuration);
    if (previewTime==0) {
        previewDuration = GetMovieDuration(gActiveMicon);
        prevTimeScale = GetMovieTimeScale(gActiveMicon);
        prevDurationSeconds = previewDuration/prevTimeScale;     /* convert to seconds
        if (prevDurationSeconds>4)
            previewDuration = prevTimeScale*4;                   /* cap at 4 seconds */
        SetMoviePreviewTime(gActiveMicon,previewTime,previewDuration);
            previewTime = 0;
    }
    SetMoviePreviewMode(gActiveMicon,TRUE);
    /* start it */
    SetMovieVolume(gActiveMicon,kNoVolume);
    StartMovie(gActiveMicon);
}

/* display the picon of the passed moovNode */
void DisplayMoovNodePicon(MoovHandle moovNode)
{
    WindowPtr   oldPort;
    Movie       theMovie,activeMicon;
    Rect        piconFrame;
    OSErr       error;
    Boolean     movieActive;

    /* init vars */
    theMovie = MoovNodeMovieID(moovNode);
    piconFrame = MoovNodeMiconFrame(moovNode);
    movieActive = GetMovieActive(theMovie);

    /* if the requested movie is already a micon, and it is in the
       front window (meaning it is actually being serviced and running), then exit */
    if ((theMovie==gActiveMicon) && (FrontWindow()==gSequencerWin))
        return;

    /* setup */
    GetPort(&oldPort);
    SetPort(gSequencerWin);
    SetMovieGWorld(theMovie,NIL,NIL);
    SetMovieBox(theMovie,&piconFrame);
    SetMovieActive(theMovie,TRUE);

    /* show it */
    ShowMoviePoster(theMovie);

    /* restore old settings */
    SetMovieActive(theMovie,movieActive);
    SetPort(oldPort);
}

/* display a text label under the moovnode's movie */
void DisplayMoovNodeTextLabel(MoovHandle moovNode,Boolean clear)
{
    WindowPtr       oldPort;
    Rect            piconFrame,textFrame;
    RgnHandle       visRegion,oldVisRgn;
    RGBColor        dColor;
    char            duration[9];
    char            *displayString;
    StringHandle    movieCopy;
    UserData        dataList;

    if (gTextTag==NO_TAG) return;

    /* set port and clip region */
    GetPort(&oldPort);
    SetPort(gSequencerWin);
    visRegion = NewRgn();
    SetRectRgn(visRegion,BIN_LEFT,BIN_TOP,BIN_RIGHT,BIN_BOTTOM);
    oldVisRgn = gSequencerWin->visRgn;
    gSequencerWin->visRgn = visRegion;

    /* set pen color to dark grey */
    dColor.red = 10000;
    dColor.green = 5000;
    dColor.blue = 10000;
    RGBForeColor(&dColor);

    /* init vars */
    piconFrame = MoovNodeMiconFrame(moovNode);
    switch (gTextTag)
    {
        case NAME_TAG:
            displayString = (char *) MoovNodeFileInfo(moovNode).sfFile.name;
            break;
        case DURATION_TAG:
            GetMovieDurationString(MoovNodeMovieID(moovNode),duration);
            displayString = duration;
            break;
        case COPYRIGHT_TAG:
            dataList = GetMovieUserData(MoovNodeMovieID(moovNode));
            movieCopy = NewString("\p<No Entry>");
            SetHandleSize(movieCopy,256);
            if (CountUserDataType(dataList,COPYRIGHT_DATA)) {
                GetUserDataText(dataList,(Handle) movieCopy,COPYRIGHT_DATA,1,0);
            }
            displayString = (char *) *movieCopy;
            break;
        default:
            break;
    }

    /* define text box */
    textFrame.left = ((piconFrame.right+piconFrame.left)/2)-StringWidth(displayString)
    textFrame.right = textFrame.left+StringWidth(displayString);
    if (gColorTag==NO_TAG)
        textFrame.top = piconFrame.bottom+1;
    else
        textFrame.top = piconFrame.bottom+6;
    textFrame.bottom = textFrame.top+10;

    /* display text in text box, or clear the box */
    if (!clear) {
            MoveTo(textFrame.left,textFrame.bottom);
        DrawString(displayString);
    }
    else {
        MoveTo(textFrame.left,textFrame.top);
        textFrame.bottom += 3;
        InvalRect(&textFrame);
        EraseRect(&textFrame);
    }


    /* restore fore/back colors to ensure proper drawing */
    RGBForeColor(black);

    /* clean up port */
    gSequencerWin->visRgn = oldVisRgn;
    DisposeRgn(visRegion);
    SetPort(oldPort);
}

/* display a color label under the moovnode's movie */
void DisplayMoovNodeColorLabel(MoovHandle moovNode,Boolean clear)
{
    WindowPtr       oldPort;
    Rect            piconFrame,colorFrame;
    RgnHandle       oldVisRgn,visRegion;
    RGBColor        dColor;

    int         colorIconID;
    CIconHandle dCIcon;

    if (gColorTag==NO_TAG) return;

    /* setup port and visrgn */
    GetPort(&oldPort);
    SetPort(gSequencerWin);
    visRegion = NewRgn();
    SetRectRgn(visRegion,BIN_LEFT,BIN_TOP,BIN_RIGHT,BIN_BOTTOM);
    oldVisRgn = gSequencerWin->visRgn;
    gSequencerWin->visRgn = visRegion;

    /* init vars */
    piconFrame = MoovNodeMiconFrame(moovNode);
    switch (gColorTag)
    {
        case DURATION_TAG:
            colorIconID = AssignIconToMovieByDuration(MoovNodeMovieID(moovNode));
            break;
        default:
            break;
    }

    /* define and draw the color box */
    SetRect(&colorFrame,piconFrame.left,piconFrame.bottom,piconFrame.right,piconFrame.
    if (!clear) {
        dCIcon = GetCIcon(colorIconID);
        PlotCIcon(&colorFrame,dCIcon);
        DisposCIcon(dCIcon);
    }
    else {
        InvalRect(&colorFrame);
        EraseRect(&colorFrame);
    }

    /* restore fore/back colors to ensure proper drawing */
    RGBForeColor(black);

    /* clean up port */
    gSequencerWin->visRgn = oldVisRgn;
    DisposeRgn(visRegion);
    SetPort(oldPort);
}

/* passed a movie, will return the ID of a cicn that marks the duration of the
   movie.  in this app, the cicn is a colored box. */
int AssignIconToMovieByDuration(Movie movieID)
{
    TimeValue       movieTimeScale,movieDuration,totalSeconds;

    movieDuration = GetMovieDuration(movieID);
    movieTimeScale = GetMovieTimeScale(movieID);
    totalSeconds = (movieDuration/movieTimeScale);
    if (totalSeconds<((TimeValue) 5))
        return(SPEED_0);
    else if (totalSeconds<((TimeValue) 20))
        return(SPEED_1);
    else if (totalSeconds<((TimeValue) 45))
        return(SPEED_2);
    else
        return(SPEED_3);
}

void AddPlayoutWindow(void)
{
    MoovHandle  moovNode;

    moovNode = FindMoovNode(gActiveMicon);
    AddMovieWin(MoovNodeFileInfo(moovNode),MoovNodeMiconFrame(moovNode));
}

/* sequencer file routines ********************************************************
   these routines show the "movie playout" library ********************************/
```

```
/* given file information, will mark a padnode to that file */
void SetFileInfo(StandardFileReply fileInfo)
{
    /* update name of window */
    SetWTitle(gSequencerWin,(fileInfo.sfFile).name);
}

/* moov controller related routines ************************************************/
/* note: functions centerRectAboutPoint, RectToCenterPoint, & ConvertRect are also
   required and should be duplicated if this section is separated as an independent
   file */

struct MovieWin {
    struct MovieWin     **next;
    struct MovieWin     **prev;
    MovieController     mcID;
    Movie               movieID;
    WindowPtr           windowID;
};
typedef struct MovieWin **MovieWinHandle;

/* linked list of standard QT movie playout windows */
MovieWinHandle gMovieWins=NIL_POINTER;

/* function prototypes */
static Boolean BaseMovieWin(MovieWinHandle);
static Boolean IsMovieWindow(WindowPtr);
static void RemoveMovieWin(WindowPtr);
static void ZoomRect(Rect,Rect,int);

/* programs that wish to use this playout library should first call this function */
void InitMovieWinList(void)
{
    gMovieWins = (MovieWinHandle) NewHandle(sizeof(struct MovieWin));

    (*gMovieWins)->next = NULL;
    (*gMovieWins)->mcID = NULL;
    (*gMovieWins)->movieID = NULL;
    (*gMovieWins)->windowID = NULL;
}

static Boolean BaseMovieWin(movieWinNode)
MovieWinHandle movieWinNode;
{
    if ((*movieWinNode)->next)
        return(FALSE);
    else
        return(TRUE);
}

static Boolean IsMovieWindow(windowID)
WindowPtr   windowID;
```

```
{
    MovieWinHandle  movieWinNode;

    movieWinNode = gMovieWins;
    while (!BaseMovieWin(movieWinNode)) {
        if ((*movieWinNode)->windowID==windowID)
            return(TRUE);
        movieWinNode = (*movieWinNode)->next;
    }
    return(FALSE);
}

/* programs that wish to use this playout library should insert this call into
   their main event loop.  the function will return TRUE if it acted on the
   event, in which case the calling program can ignore the event.  if FALSE
   is returned, the calling program should continue to process the event normally */
Boolean HandleMovieWinEvent(theEvent,dragRect)
EventRecord theEvent;
Rect        dragRect;
{
    WindowPtr       windowID;
    MovieWinHandle  movieWin;
    Movie           movieID;
    int             windowCode;

    /* check if any active movie wins generated by this library */
    if (!(movieWin = gMovieWins)) return;

    windowID = (*movieWin)->windowID;
    /* handle any controller actions */
    while (!BaseMovieWin(movieWin)) {
        if (MCIsPlayerEvent((*movieWin)->mcID,&theEvent))
            return(TRUE);
        movieWin = (*movieWin)->next;
    }
    /* handle any window actions event */
    switch (theEvent.what)
    {
        case mouseDown:
            windowCode = FindWindow(theEvent.where,&windowID);
            switch (windowCode)
            {
                case inDrag:
                    if (IsMovieWindow(windowID))    {
                        DragWindow(windowID,theEvent.where,&dragRect);
                        return(TRUE);
                    }
                    else
                        return(FALSE);
                case inGoAway:
                    if (IsMovieWindow(windowID))    {
                        if (TrackGoAway(windowID,theEvent.where))
                            RemoveMovieWin(windowID);
                            FixMenus();
                            return(TRUE);
                        }
                        else
                            return(FALSE);
                    default:
                        return(FALSE);
                }
            default:
                /* we didn't do nothin! */
                return(FALSE);
        }
}

/* programs wishing to playout a movie should call this routine
   note: sourceRect is the rect to zoom from */
void AddMovieWin(movieFileInfo,sourceRect)
StandardFileReply   movieFileInfo;
Rect                sourceRect;
{
    FSSpec          fileSpec;
    short           resRefNum;
    Movie           movieID;
    Rect            movieFrame,controllerFrame;
    WindowPtr       movieWindow;
    MovieController mcID;
    OSErr           error;
    MovieWinHandle  oldMovieWindows;

    fileSpec = movieFileInfo.sfFile;

    /**** get a qt movie */
    OpenMovieFile(&fileSpec,&resRefNum,0);
    if ((error=GetMoviesError())!=noErr)    {
            ErrorDisplay(NIL,error);
            return;
    }
    NewMovieFromFile(&movieID,resRefNum,NIL,NIL,0,NIL);
    if ((error=GetMoviesError())!=noErr)    {
            CloseMovieFile(resRefNum);
            ErrorDisplay(NIL,error);
            return;
    }
    CloseMovieFile(resRefNum);

    /* reset movie box origin to 0,0; convert source rect to global */
    GetMovieBox(movieID,&movieFrame);
    OffsetRect(&movieFrame,-movieFrame.left,-movieFrame.top);
    SetMovieBox(movieID,&movieFrame);
    sourceRect = ConvertRect(sourceRect,GLOBAL);

    GoToBeginningOfMovie(movieID);
    /**** prepare a new window */
    movieWindow = GetNewCWindow(MOVIE_PLAYOUT_ID,NIL,MOVE_TO_FRONT);
    SetWTitle(movieWindow,movieFileInfo.sfFile.name);
    SizeWindow(movieWindow,movieFrame.right,movieFrame.bottom,TRUE);
    if (!movieWindow)    {
            ErrorDisplay(NIL,error);
            return;
    }

    /*** prepare movie */
    SetPort(movieWindow);
    SetMovieGWorld(movieID,NIL,NIL);
    SetMovieActive(movieID,TRUE);
    MoviesTask(movieID,NIL);

    /**** prepare a new controller */
    mcID = NewMovieController(movieID,&(movieWindow->portRect),
                mcTopLeftMovie && mcWithBadge);
    if (!mcID)      {
            ErrorDisplay(NIL,error);
            return;
    }

    /* resize window */
    MCGetControllerBoundsRect(mcID,&controllerFrame);
    UnionRect(&movieFrame,&controllerFrame,&movieFrame);
    SizeWindow(movieWindow,movieFrame.right,movieFrame.bottom,TRUE);

    /* zoomrect */
    movieFrame = movieWindow->portRect;
    movieFrame = ConvertRect(movieFrame,GLOBAL);
    ZoomRect(sourceRect,movieFrame,4);
    movieFrame = ConvertRect(movieFrame,LOCAL);
    /* show the window */
    ShowWindow(movieWindow);
    SelectWindow(movieWindow);

    /* update linked list */
    oldMovieWindows = gMovieWins;
    gMovieWins = (MovieWinHandle) NewHandle(sizeof(struct MovieWin));
    if (!gMovieWins)         {
            ErrorDisplay(NIL,error);
            return;
    }
    (*gMovieWins)->next = oldMovieWindows;
    (*gMovieWins)->prev = NULL;
    (*oldMovieWindows)->prev = gMovieWins;
    (*gMovieWins)->mcID = mcID;
    (*gMovieWins)->movieID = movieID;
    (*gMovieWins)->windowID = movieWindow;
}

static void RemoveMovieWin(windowID)
```

```
WindowPtr   windowID;
{
    MovieWinHandle  movieWinNode,tempNode;
    Rect            windowFrame,zoominFrame;
    Point           windowCenter;

    /*** find the movie win node */
    movieWinNode = gMovieWins;
    while (!BaseMovieWin(movieWinNode)) {
        if ((*movieWinNode)->windowID==windowID)
            break;
        movieWinNode = (*movieWinNode)->next;
    }
    if (BaseMovieWin(movieWinNode)) return;

    /* init vars (before trashing the window) */
    windowFrame = ((*movieWinNode)->windowID)->portRect;
    RectToCenterPoint(windowFrame,&windowCenter);
    SetRect(&zoominFrame,windowCenter.h-1,windowCenter.v-1,windowCenter.h,windowCenter
    windowFrame = ConvertRect(windowFrame,GLOBAL);
    zoominFrame = ConvertRect(zoominFrame,GLOBAL);

    /**** dispose of the stuff */
    HLock(movieWinNode);
    CloseComponent((*movieWinNode)->mcID);
    DisposeMovie((*movieWinNode)->movieID);
    DisposeWindow((*movieWinNode)->windowID);
    HUnlock(movieWinNode);

    /**** remove the movieWin node */
    if ((*movieWinNode)->prev)
        (*(*movieWinNode)->prev)->next = (*movieWinNode)->next;
    else
        /* we're deleting the first node, so reassign gmoviewins */
        gMovieWins = (*movieWinNode)->next;
    (*(*movieWinNode)->next)->prev = (*movieWinNode)->prev;
    DisposHandle(movieWinNode);

    /* zoom in */
    ZoomRect(windowFrame,zoominFrame,4);
}

static void ZoomRect(startRect,endRect,numRects)
Rect    startRect,endRect;
int     numRects;
{
    CGrafPtr    desktopPort,curPort;
    PenState    curPen;
    Point       startPoint,endPoint,curPoint;
    Rect        curRect;
    int         delx,dely,i;
    long        delayTick;
    /**** save state */
    GetPort(&curPort);
    GetPenState(&curPen);

    /*** setup new state */
    desktopPort = GetDesktopCGrafPort();
    SetPort(desktopPort);
    PenMode(patXor);

    /**** draw the rects */
    /* init vars */
    RectToCenterPoint(startRect,&startPoint);
    RectToCenterPoint(endRect,&endPoint);
    delx = (endPoint.h - startPoint.h)/(numRects);
    dely = (endPoint.v - startPoint.v)/(numRects);
    OffsetRect(&startRect,-startRect.left,-startRect.top);
    OffsetRect(&endRect,-endRect.left,-endRect.top);

    /* draw rects */
    for (i=1,curPoint=startPoint;i<=numRects;i++)    {
        /* setup new point */
        curPoint.h = startPoint.h + (delx*i);
        curPoint.v = startPoint.v + (dely*i);

        /* draw new rect 1 */
        SetRect(&curRect,0,0,startRect.right+(endRect.right-startRect.right)*i/numRect
                    startRect.bottom+(endRect.bottom-startRect.bottom)*i/numRects);
        CenterRectAboutPoint(curPoint,curRect,&curRect);
        FrameRect(&curRect);

        /* delay */
        delayTick = TickCount()+1;
        while (TickCount()<=delayTick)
            ;
    }
    /* erase rects */
    for (i=1,curPoint=startPoint;i<=numRects;i++)    {
        /* setup new point */
        curPoint.h = startPoint.h + (delx*i);
        curPoint.v = startPoint.v + (dely*i);

        /* draw new rect 1 */
        SetRect(&curRect,0,0,startRect.right+(endRect.right-startRect.right)*i/numRect
                    startRect.bottom+(endRect.bottom-startRect.bottom)*i/numRects);
        CenterRectAboutPoint(curPoint,curRect,&curRect);
        FrameRect(&curRect);

        /* delay */
        delayTick = TickCount()+1;
        while (TickCount()<=delayTick)
            ;
    }
    /**** cleanup & restore state */
    DisposHandle(desktopPort);
    SetPort(curPort);
    SetPenState(&curPen);
}
```