A Video Editing Tool for Graphical Representation of Story Structures

by
Gideon I. Lee

Submitted to the

Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirement for the Degree of

Bachelor of Science in Computer Science and Engineering

at the Massachusetts Institute of Technology

May 17, 1993

Author [illegible]
Department of Electrical Engineering and Computer Science
May 17, 1993

Certified by [illegible] 5/18/93
Glorianna Davenport
Thesis Supervisor

Accepted by
Leonard A. Gould
Chairman, Department Committee on Undergraduate Theses

A Video Editing Tool for Graphical Representation of Story Structures

by
Gideon I. Lee

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 Computer Science and Engineering

ABSTRACT

For my thesis project, I implemented a video editing tool called the story sketch-book. It allows the user to edit the story structure of a movie graphically. The primary goal of this project is to show that a graphical representation of story structures is indeed useful. The secondary goal of this project is to investigate and resolve some of the user interface issues of an interactive tool that helps the user to create story structures graphically.

Thesis Supervisor: Glorianna Davenport
Title: Assistant Professor, Media Lab

Contents

Table of Illustrations

Acknowledgments

I would like to express my sincere gratitude to Prof. Glorianna Davenport, my thesis supervisor for this project. She provided many valuable suggestions to me in defining and achieving the goals of this project.

I would also like to give thanks to my colleagues in the Interactive Cinema Group, especially to Lee Morgenroth and Erhhung Yuan. They furnished me with much advice in various aspects of software designs.

Finally, I would like to thank Irene Fung, who spent many hours proof-reading this thesis paper for me. Her patience and encouragement are deeply appreciated.

Chapter 1 Introduction

For my thesis project, I built a video editing tool which allows the users (e.g. video editors) to create and edit graphical representations of story structures interactively. The primary goal of this project is to show that a graphical representation of story structures is indeed useful. The secondary goal of this project is to investigate and resolve some of the user interface issues of an interactive tool that helps the user to create story structures graphically.

There are eight chapters in this written account of my thesis project. Chapter 2 and 3 are about the general ideas of story representations using several conventional media. Through these discussions, I seek to illustrate some of the benefits of using a graphical representation of story structures.

In chapter 4, I discuss the functionality of a video editing tool, the story sketch-book, which I implemented for this project. The story sketch-book helps the user to build a graphical representation of story structures. In chapter 5, some sample applications of the story sketch-book are discussed. With these examples, I want to further illustrate the benefits of using graphs to represent story structures. This chapter also serves as an explanation of some of the design decisions I have made in designing the sketch-book.

In chapter 6, I discuss two unresolved issues in the design of the sketch-book. They present two interesting problems for future research. In chapter 7, some other design issues concerning different kinds of constraints(hardware, software and time) are discussed. They also serve as suggestions for future research. In chapter 8, I present a conclusion. Implementation details can be found in Appendix A.

To summarize, I seek to illustrate the benefits of using a graphical representation of story structures in this project. I sincerely hope that this project may stimulate further research endeavors in this area.

Chapter 2 Conventional Media for Story Representations

In this chapter, I discuss several conventional media which can be used for story representations. They are pictures, spoken languages, written languages and motion pictures. I shall observe the advantages and limitations of these media. These observations outline a framework for us to think about story representations on computers. In the next chapter, I shall use this framework to put several research efforts in the IC group, including this project, into a coherent context.

2.1 Three traditional media for story representations: pictures, speech and texts

Ever since people learned to communicate, they have used various representations to tell stories. Some researches reveal that even before languages were invented, people had used various imitative gestures and sounds to tell stories. Later, in order to record and extend the power of their imitative gestures, people invented drawings, most probably carved on stones or wooden blocks. Countless number of those ancient artifacts have been found by archaeologists and anthropologists in the past century. If we look at some of these artifacts, we can see that many drawings were actually sequenced in order to tell stories, e.g. about their hunting trips, or about a religious ceremony. These sequences of drawings are a kind of story representation.

Over time, human beings developed words through the use of the imitative sounds. Singular words ultimately developed into languages. All along the way, people would use these sounds to tell stories. Spoken language was a very powerful narrative medium in ancient times.

Pictures and speech were never mutually exclusive. Ever since the very beginning of the development of spoken language, people might have started to use pictures to represent sounds. In some civilizations like ancient Egypt of China, graphical representations of ideas were deemed important. Hence, their characters were invented based mostly on symbolic pictures. Their written languages are a mapping of symbols to spoken languages. In some other civilizations, like Greece for instance, graphical representations were deemed less successful. This generated a need which would be filled by the invention of alphabets. Thus, in many civilizations textual representations of stories were invented and used extensively.

Long ago humans discovered that these three media have their own unique advantages and disadvantages in representing stories. Drawings are good at describing details of what happened in a relatively short span of time. However, for stories that involve complicated plots, even elaborate sequences of drawings are often inadequate. (In medieval ages there were paintings which strived at telling some biblical stories in one single canvas. Those elaborated pictures, while high in artistic value, really tell you little unless you have heard about the underlying stories before.) Spoken and written languages might never be as good as drawings in terms of representing visual details. But they are much better in conveying overall story structures and details that are invisible, such as human feelings. In addition, spoken narratives have the edge of interactiveness.

2.2 Motion picture as a rich medium for story representation

The medium of motion picture has most of the advantages of the three traditional media. It allows us to incorporate pictures, sounds and texts into one integrated form. Therefore it is a particularly flexible medium for storytelling.

Flexibility comes with a price. Creating movies can be much harder than telling stories using the three traditional media. The story representations can become extremely complicated because they contain so many levels of detail. Without some abstractions, the story representations of movies are almost impossible to deal with.

A natural way to cope with the complexity is to divide the story representations of a movie into three classes of detail. The first class is about the visual details of what happened during a relatively short span of time. The second class is about the story structures. The third class is about the sounds.

One may ask: do the three types of details together completely characterize a movie story? In fact, they cover a large part of it already. Imagine having all the frames, the script and the sound track of a movie with you. Do you have a rough idea of how the movie will look like? I think you should have. Maybe you would not feel the pace or the atmosphere just by reading the text or listening to the sound track, but you should have a very good idea about the story the movie represents.

This classification of detail is also useful when we build representations on computers. In the next chapter, I classify the data representations of motion pictures into three corresponding classes and describe the tools that are required in constructing such classes. The next chapter will also position my current project using the perspective of this classification.

2.3 Some shortcomings of the motion picture medium

The motion picture medium is not the solution to everything. There are things we can do with other media which we cannot do with movies.

Movies are limited by their resolution and dimension. Why are they important in story representations? You will realize it if you go to an old catholic parish and look at the wall paintings. Each picture occupies a special position in the 3-dimensional space. In fact, its 3D position usually has a chronological meaning. Therefore in a metaphorical sense you are flying backward and forward in time and space while you are looking at those pictures! Furthermore, you can study a picture in detail. Watching a movie is not like that. You are locked in step with the time axis of a movie. You are not free to go back to study a frame in detail. Even if you are allowed to do that (e.g. with a VCR), the resolution of the frame is limited for further study.

Spoken narratives are much more interactive than movies. Good storytellers adjust their narratives according to the environment and the audience. The audience may even ask questions! Movies do not allow you to do that.

Fiction can combine many different kinds of background information with the story. The information can range from complex emotions to abstract mathematical formulas. It is harder to incorporate those elements into a movie. In fact, anything that is not narrative in nature or that takes a long time to understand does not lend itself to the movie medium.

The advent of computer technology leads us to the possibility of creating interactive movies. Some of the shortcomings I just mentioned would be partially overcome by interactive movies. Imagine having movies that know how to adapt to the audience, that elaborate on new ideas, that answer "why" and "how" questions... All of these possibilities seem extremely appealing.

However, it doesn't take us too long to realize that interactive movies use even more complicated forms of story representations. For one thing, the story structure may not be linear any more. Finding a way to cope with this additional complexity is essential to the production of interactive movies. In the later sections of the next chapter, I shall discuss some of the current efforts to cope with these problems. I shall also discuss how I have incorporated these ideas into my current project.

2.4 Summary

In this chapter, we discussed briefly some advantages of the motion picture medium for story representations in contrast to other traditional media. The richness of the medium inevitably leads to relatively complicated story representations. Therefore we must find ways to cope with the complexity.

We also looked at some of the shortcomings of movies. We hypothesized some kinds of interactive movies which may overcome those deficiencies. But we realized that such movies require even more complicated story representations. There is an even stronger need to find ways to control the complexity.

Computers are useful tools for controlling complexity. But in order to make use of the computers, we first need to make appropriate representations of movies on computers. This is the subject of the next chapter.

Chapter 3 Representing Motion Pictures on Computers

In section 2.2 I propose that a movie can be characterized by its pictures, its story structure and its sound track. If we want to make a motion picture representation on a computer, these three elements must first be represented on computers.

Therefore, three kinds of data representations are needed. The first kind helps us to represent what happened visually in a short span of time. The second kind helps us to represent story structures. The third kind helps us to represent sounds. Because there are three kinds of representations, three kinds of representation building tools are needed.

3.1 Stratalog, Stratagraph and Framer

The first kind of representation building tools lets us describe the visual details during a relatively short span of time.

Stratalog and Stratagraph developed by the Interactive Cinema Group may be considered examples of this kind of representation building tools. By associating keyword annotations to a small segment of video material (a process we refer to as logging), we essentially represented what happened visually in a short span of time.

Using a database built on Framer [Hasse 1993, Davis 1993], the semantics of keywords can be inferred by analyzing the position of the word in the class hierarchy. For example, suppose the keyword 'cat' is used as an annotation. We may then infer that it is an animal. Thus, a lot of the contextual common sense required to understand pictures may also be incorporated in this kind of knowledge representation system.

Theoretically, the power of such an approach is limited only by 1) the amount of effort we spend on logging and 2) the richness of the corresponding class hierarchy.

3.2 Homer, the Sketch-book...etc.

The second kind of representation building tools lets us establish representations of story structures.

Homer, developed by Lee Morgenroth, provides a script language which we can use to represent story plots [Morgenroth, 1991]. It represents a story plot using a sequence of episodes. Each episode is in turn characterized by keyword annotations, similar to a log of Stratalog. The persons who build the story plot may or may not be familiar with the original footage. In either case, they could specify a story plot and pass it to Homer. Homer will search for the appropriate logs to fit into those episodes. Therefore, as an editing tool, Homer can be used to automate much of the search and select tasks associated with the video editing process. This kind of tools provides a rough cut for final manual editing. Morgenroth suggests that Homer is especially suitable for news stories which several fixed story structures are used in almost every case.

However, the idea embodied by Homer is more general than automatic video editing. Homer essentially defines a keyword representation of story structures. This representation is similar to an outline of a screenplay. Using this representation, many interesting applications can be developed. It is conceivable, for example, that a computer program can use the Homer scripts, together with the logs, to answer many "how" and "why" questions about a movie story interactively.

The story sketch-book I built in this project is based on the same idea. It defines a graphical representation of story structures. The story sketch-book can be used as a video editing tool that allows people to create video sequences graphically. But the representations can also be used in other ways. Some of the sample applications are outlined in Chapter 5.

The most essential feature of the story sketch-book is incremental and interactive development, which makes it a powerful environment for experimenting variations of story structures. The story sketch-book is especially suitable for documentary videos which are shot without a detailed screenplay or even a story line. In those cases, it is the job of the movie editor to make sense out of the materials by creating a story structure which relates different scenes. Using the story sketch-book, we can search for the appropriate video shots by keywords, relate them in different ways and immediately see how effective our story structure is by test playing it back. If we don't like the structure, we can change (using graphical objects and linkages) and review it. We can keep doing this until we are happy with the story structure.

Another advantage of this approach is that it is much more flexible in terms of story structure fine-tuning. Each story structure is developed to maximize the use of available materials.

3.3 Representations of speech and sound

The third kind of representations is more ambiguous. The general idea is that it is about the kind of things we hear while watching a movie. They can be dialogue, narration, song, background music or other sound effects. However these elements are strongly tied to some elements in the first two kinds of representations. Dialogue and songs often appear synchronously with the pictures. Consequently, they are tied to the elements of the first kind. In contrast, narration and background music have close relationship with the story structures. Therefore, they are tied to the elements of the second kind. Due to the overlaps, it may be hard to isolate the elements of the third kind from the other two.

Up to now, there seems to be few research efforts dedicated solely to the representations of the audio aspect of movies. In practice, background music and narration are usually overlaid on the top of a movie only after the picture is locked. I think more can be done to incorporate these elements as organic parts of a movie story.

3.4. A graphical representation of story structures

Let us focus on the representations of story structures again. In the following three sections, I seek to show that a graphical representation of story structures is both intuitive and powerful. The story sketch-book is an initial attempt to realize such a graphical representation. I shall discuss the implementation features of the story sketch-book in the next three chapters.

Representing a story structure by a graph is not the same as simply having a graphical user interface. A graph in this context refers to a mathematical graph, i.e. an object which can be expressed in precise mathematical terms. Therefore, it is really more than a question of visualization. It poses strong restrictions on what we can and cannot represent easily.

As I have shown, there are various kinds of story structure representations. However, few of them are really suitable as computer representations. Narrative texts may be meaningful for people but they are hardly useful for computers today. I sometimes imagine that natural language recognition technology will allow an intelligent computer to understand a screenplay by itself - but this is nothing more than a fantasy now. Given today's technology, we still have to build conventional kind of data representations to describe a story structure.

As a computer representation, graphs have several advantages. First, graph theory is a well developed branch of mathematics. There are standard theorems telling us what we can or cannot do with a graph. For example, a lot of searching and ordering algorithms have been developed. If we devise a script language, we might have to re-invent many of these algorithms. Second, graphs and many useful operations on them can be easily implemented on a computer. There are proven ways to do them. Most importantly, it is natural to think of a story structure in terms of graphs. Many screenplay writers actually work out their story plots using various types of graphs.

A suitable representation must also be easy for people to use. A script language that involves hundreds of keywords may be an adequate computer representation of some story structures. But very few people would be willing to use it. In general, a meaningful story structure involves dozens of keywords. By the time you finish, you might have forgotten the meanings of some keywords already. Worse yet, think about someone who has to use the keyword representation you created. It is extremely hard for him to learn all of them. Therefore, I feel that a keyword representation of story structures could not get us too far; although it may be a powerful representation for short episodes.

Before GUI became a common place for computers, it did not really make sense to use graphs. When expressed in strict mathematical terms, graphs are actually even harder to understand than a script language. With a nice GUI, however, graphs are easy to visualize and edit on computers. (What can be more intuitive than drawing a directed link to represent a causal relationship?) It is easy to learn and easy to understand. Furthermore, a graph can usually summarize a dozen lines of scripts on a single canvas. Thus the efficiency of story structure building is enhanced.

Before I end this section, let me emphasize once again that representing a story structure using a (mathematical) graph is not at all the same as having a graphical user interface. We can design a graphical user interface for a script language in which keywords are represented as graphical symbols. We may, say, design a GUI version of Homer. It might make it easier to use but it does not make such a representation a (mathematical) graph. Conversely, we really don't need a graphical user interface to represent story structures as graphs. But a nice GUI makes working with graphs much more intuitive.

3.5 Dealing with interactive movies

In section 2.3, I observe that the complexity of story structures increases tremendously as we move from conventional movies to interactive movies. Interestingly, the advantages of using graphs actually become more apparent when we build story structures for interactive movies, especially *multi-threaded* movies.

Most of these advantages stem from the fact that a graph can be visualized as a two-dimensional object. No matter how you conceptualize a script in your head, it is still structurally (and visually) a linear object. In contrast, a graph can be mapped to a two-dimensional canvas. (In fact, when virtual reality becomes a possibility, a graph can even be mapped to a three-dimensional space!) This added dimension gives us more space and freedom to layout a story structure. In the following, I list three examples of such flexibility:

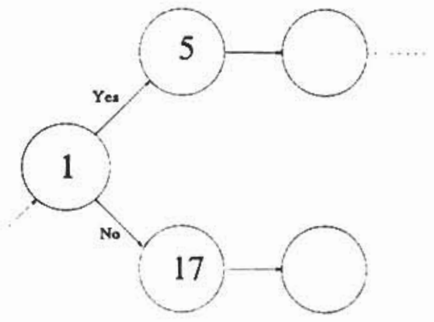
Example 1: Visual locality of closely related episodes

An essential component that we may wish to incorporate into an interactive movie is branching. Suppose that after showing episode #1, we want to show episode #5 if the user enters 'yes'; and we want to show episode#17 if the user enters 'no'. Using a hypothetical script language, we may write something like that:

```
label#1: show "episode #1"
         if yes_entered goto label#5
         if no_entered goto label#7

...[ indefinite number of lines ]...

label#5: show "episode #5"

...[ indefinite number of lines ]...

label#17: show "episode #17"

...
```

A problem pops up immediately: the descriptions of episode#1, episode#5 and episode#17 may be far apart from each other in the script. Consequently, they are not shown on screen at the same time. It is therefore harder to see the relationship of the three episodes quickly. As the number of branchings increases, the script would become harder and harder to follow in this sense. This is, in fact, the infamous "spaghetti" problem of some computer languages such as BASIC.

Using a graphical representation, what we will see instead is:

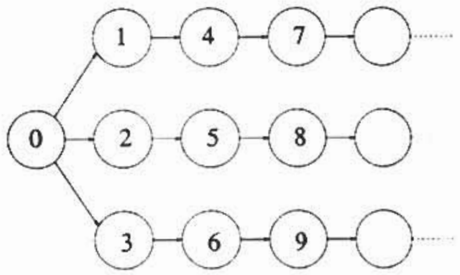


(fig. 3.1 Visual locality of related episodes)

When you look at episode #17, you realize immediately from the graph that it is an alternative to episode #5. And both of them are results of label #1. This kind of information is not apparent from a script. Furthermore, I think you will agree that directed links are much more effective than goto-label pairs in a script. When you see a "goto label#5" statement, you will need to spend a few seconds to scan the script before you will find out where label#5 is. The directed links however point right at the episodes you want to look at.

Example 2: Parallel constructs

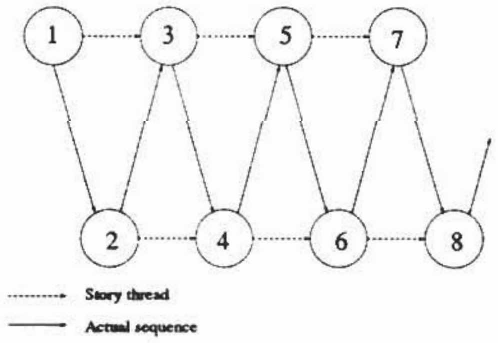
Suppose there are several parallel sequences in a story structure. A graph allows you to represent them explicitly:



(fig. 3.2 Parallel constructs)

We see immediately, for instance, that episode#4 is a parallel to episode#5 and episode#6. In a script, such parallel sequences are harder to visualize.

We should also note that parallel constructs are not useful only in interactive movies. They can also be found in conventional movies when we want to contrast several story threads. For instance,



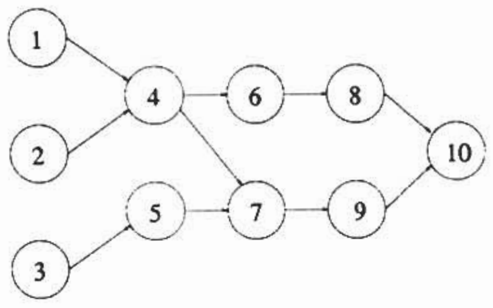
(fig. 3.3 Parallels in a linear movie)

This is basically an interplay of two story threads. The important thing to note is that we can see clearly which episode belongs to which story line. This abstraction is harder to recognize in a script.

Parallel construct is only one special case in which we want to group episodes into larger units. The next example presents another special case, the tree construct.

Example 3: Tree constructs

The tree below traces the causes behind a particular event:



(fig 3.4 Tree constructs)

A tree construct like that is called a semantic net (Ch.2, Artificial Intelligence, Winston). By searching backward (usually referred to as backward chaining), a computer program can use the tree to answer semantic questions, such as,

"Why did event 7 happen?"

Answer: It happened because of event 4 and event 5.

"What are the consequences of event 4?"

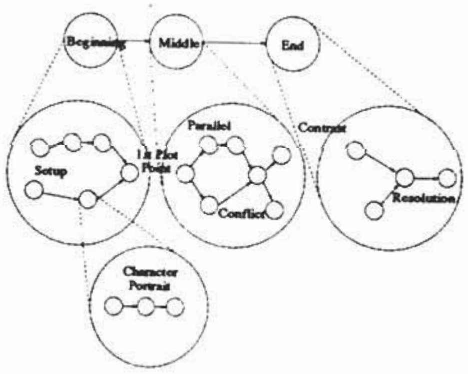
Answer: Event 6 and 8. It is one of the causes for event 7, 9 and 10.

Semantic trees make it possible for the audience to ask questions. They can greatly enhance the power of interactive movies. Using graphs, semantic trees are easy to visualize and create. This is another strong argument for using graphs to represent story structures in interactive movies.

3.6 Representing suppressed details using several levels of graphs

A graph is suitable for showing the overall shape of a story. But it may seem inadequate if the story contains a lot of details. Unless a graph has infinite dimension, there is a limit to the amount of information we can display on it. If the complexity of a story structure is beyond what we can show on a single canvas, we must suppress some of its details. There must also be some ways for us to look at the suppressed details when we need to.

Atlas designers have discovered some solutions to this problem by creating several levels of maps. Similarly, we can break down a story structure into several levels of sub-structures, each represented by another graph. For instance,



(fig. 3.5 Using sub-graphs to abstract details)

This works well initially. But problems arise when we want to cross-relate episodes of two different sub-structures. For example, there is no way to make a causal linkage between episode #1.1 and episode #2.1 because they belong to different graphs. The only way to get around this problem is to bring both of them up to the top level.

Fortunately, cross-relating episodes of two different sub-structures is not a good idea in general. It destroys the very purpose of creating sub-structure abstraction - to suppress the details at a particular level.

3.7 Summary

In the first-half of this chapter, I discussed three kinds of representation building tools for movies. The first kind helps us to build representations for visual details in a short span of time. The second kind helps us to create story structures. The third kind helps us to build representations for sounds. The story sketch-book I built in this project belongs to the second kind.

In the second-half of this chapter, I argued that a graphical representation of story structures is both intuitive and powerful. It helps us to control the complexity of a story structure in a lot of ways. Besides, it takes advantage of today's GUI technology. Finally, it is a well-understood branch of mathematics. Numerous theorems and algorithms are available.

In the following chapters, I shall discuss the story sketch-book I implemented for this project. It is an initial attempt to realize the benefits of using graphs to represent story structures.

Chapter 4 Functionality of the Sketch-Book

In this chapter, I describe the functionality of the story sketch-book, a software module that helps the user to build a graphical representation of story structures.

The story sketch-book was implemented on the X/Motif platform using C. The software was developed on a DEC Station 5000 running the ULTRIX operating system. The DEC Station has a video capture board that displays video in 24-bit and accesses a video laserdisc player (PIONEER 5000).

The story sketch-book is designed to be used together with Stratalog[Lee 1993]. (I developed Stratalog in my UROP position in the IC group under a research project supported by British Telecom.) We use Stratalog to log the video material. Then we use the story sketch-book to build up story structures by relating the logs. In this chapter, I assume the reader understands the functions and the operations of Stratalog.

The story sketch-book is designed to help us create story structures. However, it can also be used as a simple sequencer. That is, we can select a set of shots and link them together into a sequence. In this context, a story structure refers to a representation of the relationships between different episodes in a movie story; while a sequence refers to a particular linear sequence of video clips.

A lot of the GUI ideas presented in this chapter are inspired by two books on representing complex information structures. They are Visual Programming by Nan C. Shu (Van Nostrand Reinhold, New York 1992) and Envisioning Information by Edward R. Tufte (Graphics Press, Connecticut 1990) .

4.1 The menu, the canvas and the cursor

Most of the activities of the story sketch-book take place within a kind of objects called **episode**. An episode can be thought of as a page in the sketch-book. Visually, an episode is a window. Each episode consists of a **menu**, a **canvas**, and a **cursor**. The menu consists of a number of function buttons which you can click on to perform different functions. The canvas is an empty space on which we draw the story structures. The cursor tells us where a new object will appear on the canvas. The cursor shifts its position accordingly when a new object appears.

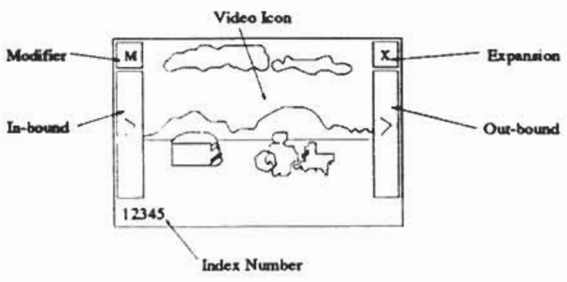
Theoretically, we can use as many episodes as we want. All the episodes are related to the others in a tree-like structure. This allows us to suppress some details of a story structure in an episode window and represent those details in its **sub-episodes**. All the sub-episodes look the same; they all have a menu, a canvas and a cursor.

4.2 Predefined objects

The story sketch-book provides a number of predefined-objects for us to build story structures. They are discussed in this section.

Frame

The **frame** object represents a particular frame on the video. To create a frame object, play the laserdisc until it reaches the desired frame and then click on the **Capture Frame** button in the menu. A frame object representing that particular frame will appear.



(fig. 4.1 A frame object)

There are a number of components in a frame object. At the upper-left corner is a **modifier** ('M') button. At the upper-right corner is an **expansion** ('X') button. In between we see a video icon. Beneath the video icon is an index number. There are also two arrow buttons on the left and right margins of the frame object.

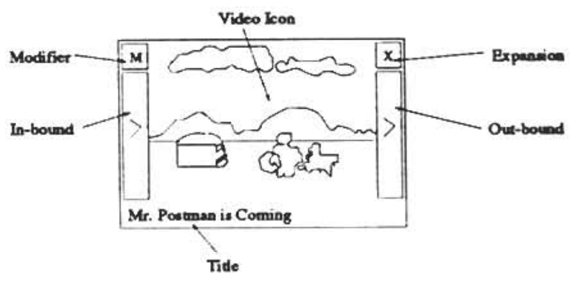
The video icon shows the frame the object represents. The number underneath it is the frame index number. We can modify this frame object to represent another frame any time by clicking its modifier button. The video icon and the frame index number will change accordingly.

It is possible to relocate the frame object on the canvas. First, click on the video icon. The frame object will turn into a rectangle. The rectangle will follow the cursor wherever it goes unless the cursor moves outside the canvas. Lead the rectangle to the desired position and click on the left mouse button again. The frame object will re-appear there.

To delete a frame object, click on its video icon again so that it turns into a rectangle. Click on the delete button on the menu and the frame object will disappear.

Log

A frame is not too useful for building story structures. The building blocks for story structures are usually **logs**. The easiest way to obtain logs is to capture them from Stratalog. First, load the appropriate log file into Stratalog. Next, highlight one or more of the logs in the Stratalog main window just as what we do when we cut or edit logs. By clicking on the **Capture Logs** button of the episode window which we want the selected logs to appear, the logs are imported into the story sketch-book.



(fig. 4.2 A log object)

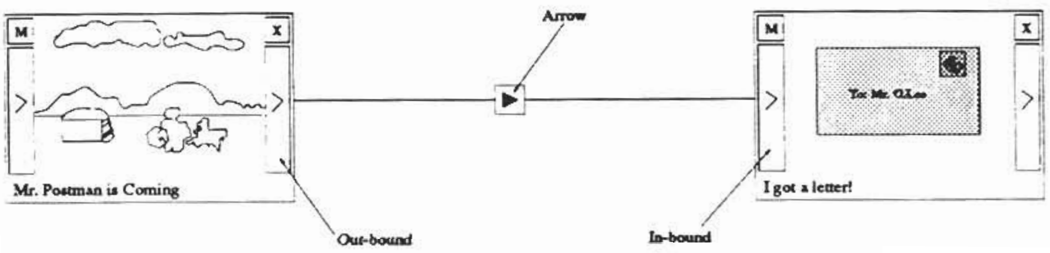
Like a frame object, a log object consists of a modifier button, a video icon, an expansion button and two arrow buttons. There is no frame index number. Instead, the title of the log appears underneath the video icon. The video icon shows the key frame of the log.

The log object can be relocated just like the frame object. If the modifier button is clicked on, the log object will change its content to represent the current selected log in the main window of Stratalog. A log object can be deleted like a frame object.

Linkage

To represent a story structure, we must be able to create **linkages** between log objects. The arrow buttons of the log object are used to create directed links between objects.

The arrows on the left and right margin are called the **in-bound** and the **out-bound** button respectively. Suppose we want to draw a directed link pointing from log#1 to log#2, we first click on the out-bound button of log#1. A linkage object appears and follows the movement of the cursor. Click on the in-bound button of log#2. A directed link has now been established between the two logs.



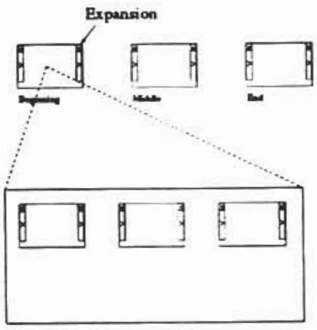
(fig. 4.3 A connection object)

To remove a linkage, click on the arrow button along the linkage.

Sub-episode

Using logs and linkages, a simple story structure can be constructed. However, as a story structure grows in complexity we run into issues of display space and navigation within the story structure istelf. Both of these problems, one physical, one cognitive, suggest the idea of nested hierarchy of episodes. In the story sketch-book, we can use sub-episodes to abstract some of the smaller structures.

Click on the expansion button of a log object. A new episode window will pop up. This new episode window represents a sub-episode to the original episode. It can be used as an elaboration of the expanded log. We can build a story structure in this sub-episode using identical manipulations described above.



(fig. 4.4 A sub-episode object)

We can also have sub-episodes of sub-episodes. There can be as many levels of sub-episodes as we want. Thus, story structures can be made arbitrarily complex.

We can pop down a sub-episode by clicking the **Pop Down Canvas** button in its menu. The sub-episode will then be hidden from the screen until the expansion button of its attached log object is clicked on. We cannot destroy a sub-episode. It is automatically destroyed when we destroy its attached log object.

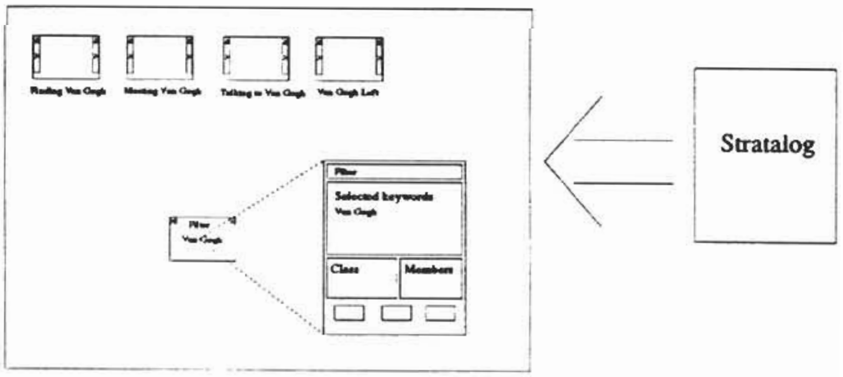
Filter

Obtaining logs using the capture logs button discussed in 4.2.1 can be quite tedious at times. The **filter** object can provide us with efficiency of scale and speed.

Each filter object contains a list of keywords. When a filter object is triggered, it goes through all the logs which are currently loaded into Stratalog and imports all matches to the episode canvas. The filter object is very useful if we want to find, say, all the logs about a character, a place, an object ...etc.

To create a filter object, click on the **Create Filter** button. A filter will appear on the canvas, together with a filter dialogue. We select its keywords by clicking on the appropriate class and the corresponding keywords. To remove a keyword from the list, simply click on the keyword.

Click on the **Done** button when all the desired keywords have been selected. The dialogue will disappear. Click on the **trigger** ('!') button and the filter will take action. All the logs that match the requirements are placed on the canvas.



(fig. 4.5 A filter object and its dialog window)

We can modify the keywords of a filter. Click on the modifier button and the filter dialogue will re-appear. The operation is the same as before.

Filters can be relocated or destroyed like the other kinds of objects. But filters do not support linkages and sub-episode expansions.

4.3 Programmable objects

The predefined objects discussed in section 4.2 may not be sufficient for all representations. Therefore, the story sketch-book allows the user to create new objects to perform customized functions. These objects are known as function objects.

Some programming and interface work are required in order to create new function objects. The details can be found in the appendix. In section 5.2 and 5.3, I shall explore several possible uses of the function objects.

4.4 Test driving...

The test driving function is an essential component of the story sketch-book. Three kinds of test drivings are described here. (Since the procedures discussed here are related to specific kinds of applications, the reader may wish to read Chapter 5 first before coming back to this section.)

Simple linear sequences

To test drive a linear sequence that has no branching or sub-episodes, first click on the **test driving** button. Then click on the first episode of the sequence. A path showing this sequence will be shown. The sequence will start playing when the **Play** button is clicked on. A **Pause** button is also available.

Multi-threaded sequences that involve decision functions

To test drive a multi-threaded sequence that has branching and/or decision functions, we click on the same test driving button as we did with simple linear sequences. We click on the first episode of the sequence. A path showing one possible sequence will be shown. Another path will be shown if the first episode is clicked on again. When the desired path is shown, click on the play button and the sequence will start playing.

Sequences that involve sub-episodes

To test drive a sequence that involves sub-episodes, begin by clicking on the **test drive recursive** button. All of the operations are similar to test driving the previous two kinds of sequences except that all of the different paths - including different paths through the sub-episodes - will be explored.

4.5 Summary

In this chapter, I described the functionality of the story sketch-book. I have tried to make the story sketch-book both easy to use and flexible. Two essential objects that make it flexible to use are the sub-episode object and the function object. In the next chapter, I shall illustrate the power of the story sketch-book by presenting a number of sample applications.

Chapter 5 Sample Applications

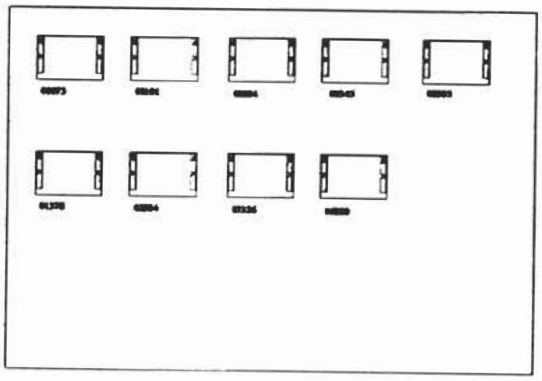
In this chapter, I describe several sample applications of the story sketch-book. The purpose here is to show that this tool (and the more general notion of graphical representations of story structures) is useful in movie production.

Creating a story structure bears some similarity to writing a screenplay or a piece of fiction. Therefore I have placed many of the ideas presented in this chapter in the context of the thought-process of creating stories. There are two books which provided me with many insights of this kind. They are Screenplay: The Foundations of Screenwriting by Syd Field (1982, Dell Publishing Group, New York) and Creating the Story: Guides for Writers by Rebecca Rule and Susan Wheeler (1993, Heinemann Educational Books, Inc., New Hampshire).

5.1 Pre-logging by capturing frames

Logging is usually regarded as the first step in video editing. However, sometimes people want to run through the video once before logging. This allows them to get a rough idea about the contents of the video. While they are watching the video at this stage, they may want to capture some key frames along the way. These key frames help them to gather a first impression of the overall structure of the video so that they can have a better idea of how to log the video.

The frame object is designed for this purpose. To use this object we can start playing the video from the beginning. We then place the cursor on the top of the frame button and click on the button when we see an interesting frame.



(fig. 5.1 Pre-logging by capturing frames)

Soon, we shall have a whole screenful of frames on our canvas. When we decide that we are ready to log the material, we may use these frames to decide what kinds of keywords we need and roughly how the story structure should be.

5.2 A canvas for sketching story structures

The story sketch-book helps us to edit story structures interactively and incrementally. We immediately get a sense of how our story or a portion of it is progressing. This interactiveness gives the story sketch-book an important edge over a script-based editor.

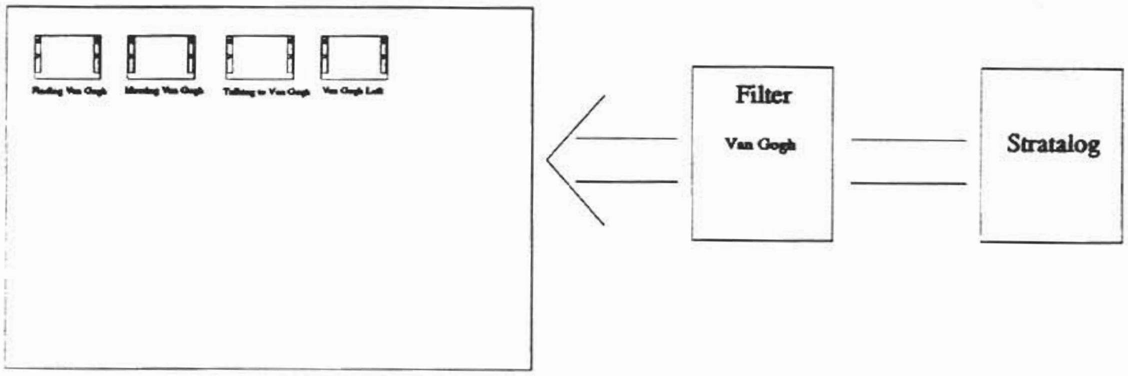
Incremental construction is also facilitated by the video icons. Using the sketch-book to create a story structure is much like arranging photos on a table to tell a story.

In the following, I describe some typical procedures of creating story structures using the story sketch-book. They are by no means the best or the only approaches. But they illustrate how we can take advantage of the power of the story sketch-book.

Sketching story structures for linear movies

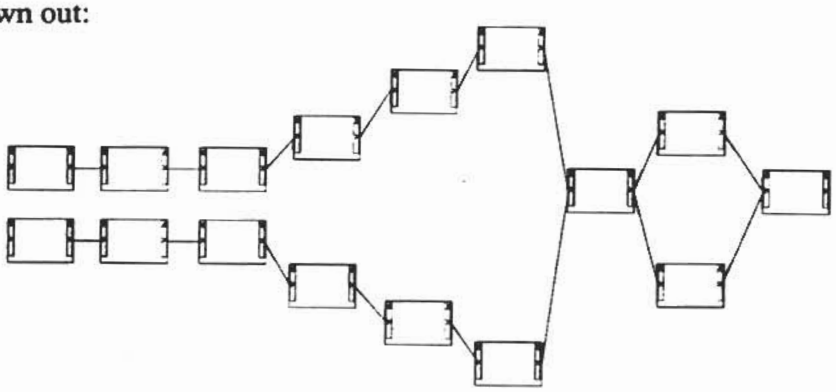
A typical movie story structure has a **beginning**, a **middle** and an **end**. Each of the three might contain a complicated story structure in itself. Using the story sketch-book, we can break down the overall story structure into three corresponding parts, each being represented by a **sub-episode**.

The beginning is often used to **setup** the story; typically it introduces the characters, the place, the situation ...etc. to the audience. Therefore, while we are constructing the setup story line, we may want to locate all the materials about a particular person or a particular place. We can use the **filter** object to locate relevant positions of the materials by generating keyword searches.



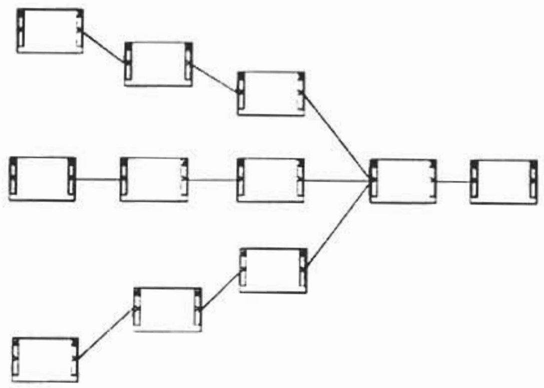
(fig. 5.2 Using filters to search for logs)

There is usually a **plot-point** established at the beginning and linked to a later point in the story. It may be the first encounter of the major characters. It may be some kinds of unusal incidents. Anyway, the plot-point sets the story in motion. What happens usually is that several story threads following different characters will emerge from the plot-point. These threads may take the forms of **parallel action**, **contrast** or **confrontation**. These kinds of story structures are much easier to conceptualize if they are drawn out:



(fig. 5.3 Parallels, contrast and confrontation)

The story threads usually come together at a second plot-point which leads to a **resolution**. Most movies resolve all the conflicts. The story sketch-book allows us to check graphically that they are brought into a common plot-point.

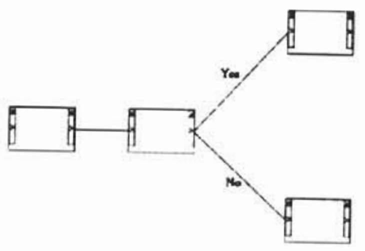


(fig. 5.4 Resolution)

Finally, the end sub-episode *may involve a few flashbacks.* Since those flashbacks are usually fragments of materials, they might not be appropriately logged. In those cases, we might need to go back to the logger to create the appropriate logs. If we have not thrown the **frame** objects away, they come in quite handy. Frame objects can save us a lot of time locating the right frames.

Sketching story structures for interactive multi-threaded movies

The sketch-book provides us with a programmable object, the function object. The function object can be used, for example, as a decision block, which directs the flow of an interactive multi-threaded movie.

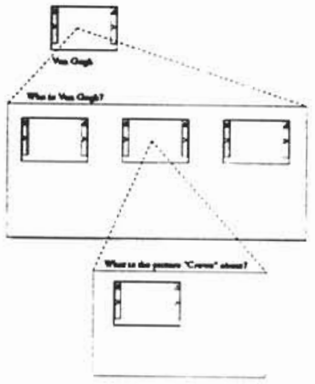


(fig. 5.5 Using decision block)

Using this representation, it is conceivable that a movie player can be programmed to prompt the user for responses when it comes to the decision points. This kind of functions is essential in many interactive movies.

Using sub-episodes to perform hypertext-like functions

It has been shown that a sub-episode can be used to abstract some of the details from the overall story structure. But it is conceivable that sub-episodes can also be used as something like hypertext in an interactive movie.



(fig. 5.6 Using sub-episodes to perform hypertext-like functions)

When such an interactive movie is played, the system will indicate to the user if further background information is available. The user can retrieve the background information if he wants to.

This sort of applications is particularly useful in computer aided education. Each student may watch the instruction video at his own pace. If something is unclear, he can stop and explore the background information.

5.3 A visual representation of relationships between episodes

By representing a story structure in a tree-like fashion, it is possible that we may use the representations to answer some 'how' and 'why' questions. This possibility has been discussed in example 3, section 3.5.

5.4 Summary

In this chapter, I discussed some typical applications of the story sketch-book. It is shown that the story sketch-book can be used to assist the production of many different kinds of movies, from conventional linear movies to interactive multi-threaded movies.

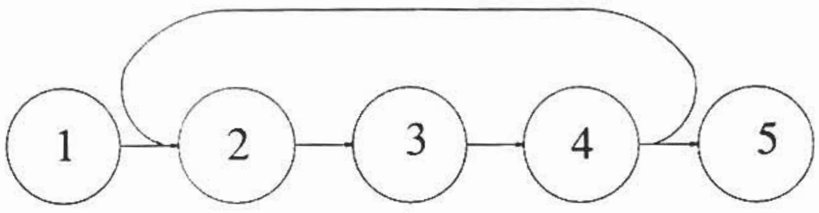
The story sketch-book is only a story structure editor. It must be coupled with a movie player which can make use of the story structure information. A number of these interactive movie players have been developed at the Interactive Cinema group. Most of these movie players, however, either hardwired the story structures in the applications or use some kind of scripts to represent them. The story sketch-book provides a much easier mechanism to build story structures.

Chapter 6 Unresolved Issues

There are two unresolved problems with the test drive function which I described in section 4.4.

6.1 Loops

The test drive function explores all forward directing sequences starting from a log object. This is fine if there is no loop because the number of possible sequences is finite. If there are loops, the number of possible sequences is infinite. This is because a sequence may go through the loop any number of times:

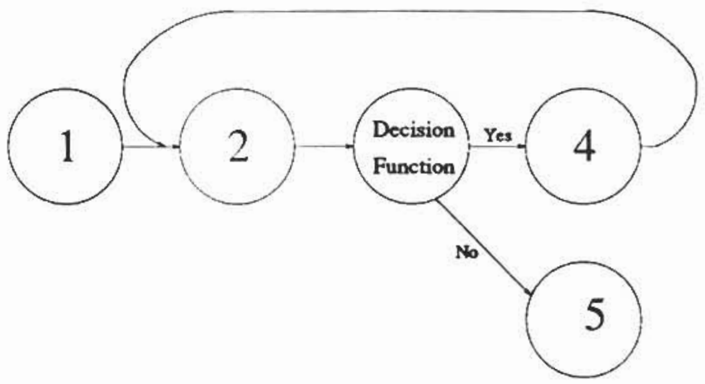


(fig. 6.1 A loop)

The story sketch-book cannot handle this situation. In fact, it is this problem with loops that requires us to use directed links. A bi-directional link is equivalent to a two-node loop.

A partial solution might be achieved by a stop function which kicks in when a loop is detected. That is, the sequence will end at log#4. However, I do not think this is a satisfactory solution. It defeats the very purpose of test driving a loop.

Another solution that seems more satisfactory is to require some kind of decision blocks to exist within a loop. The loop is explored only once. For the second time, it exits at the other branch:

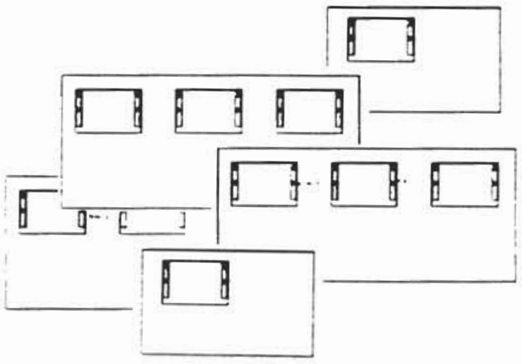


(fig. 6.2 A loop with an exiting decision block)

There are many other alternatives. I leave it to future research to explore the possibilities.

6.2 Exploring sub-episodes

Test driving sequences that involve sub-episodes presents a visualization problem. In the sketch-book, all the sub-episode windows will pop up at once when we explore different sequences. The screen becomes really hard to read if there are many level of sub-episodes:



(fig. 6.3 Too many sub-episodes make sequences unreadable)

However, we do need to show all the sub-episode windows in order to display the differences between sequences. I cannot think of any solution to this problem. This is an interesting problem that we may try to resolve in the future.

Chapter 7 Design Issues

7.1 Design decisions influenced by hardware and software constraints

The design of the story sketch-book is influenced by a number of hardware and software constraints. I discuss some of them here. They are essential technical issues which we need to consider in planning research projects for the future.

Computational speed

In the story sketch-book, an object is represented by a rectangle while we are moving it. This is because redrawing a video frame on screen takes too long for human patience. It is not practical to move an object as it is. The problem is less serious for other kinds of objects such as filters and function blocks. But for consistency, they are also turned into rectangles while we are moving them.

Smooth scrolling of the canvas is impossible with the current hardware. In fact, scrolling a canvas when there are more than ten video frames on it may take as long as half a second. If we increase the maximum size of the canvas, it becomes even slower. For this reason, the default maximum size of a canvas is only set to a modest 1200x1000.

Memory

One of the biggest hardware constraints for me initially seemed to be the memory requirement of the digitized images. A video image of resolution 120x90 with 24-bit color depth takes around 30K of memory. It is very possible that the system will run out of memory quickly as we use more images.

This turned out less of a problem than I anticipated. The program runs perfectly well with as many as 200 images on screen. (That adds up to around 6 megabytes of memory.) It is because in a virtual memory environment like UNIX, the system turns to the hard disk for additional storage when it runs out of local memory. The system will slow down when many images are presented on the canvas. But this does not fundamentally impair the functionality of the program.

Screen size

The decision to have a default image resolution of 120x90 has relatively little to do with the system speed and memory. But rather, it is due more to the screen size and the quality of the video image. I tried different image resolutions, ranging from 80x60 to 200x150. The image quality becomes unacceptable when the resolution is below 100x75. The image quality does not improve very much when the resolution is larger than 160x120. We can place around 20 images on a display-size canvas when the image resolution is 160x120. I think 20 images are not enough. I chose 120x90, which allows us to place around 36-40 images on a display-size canvas. This allows us to sketch a story structure of moderate complexity.

Input devices

I found that the mouse is not a good input device for drawing story structures. It is hard to locate an object precisely at the desired position with a mouse. For some people, it is also awkward to draw linkages with a mouse. We should consider using a drawing tablet or a light pen for future implementations.

Video capture board

The application programmer interface (API) of the video capture board is very comprehensive. It allows the application program to resize and capture the image without much loss of image quality. The story sketch-book would be much harder to implement without such API.

Brightness and contrast are two useful API functions that future implementations should try to make use of. From my experience, when an image is down-sized, it generally looks darker. (It is also true if we dither an image of 24-bit plane to 8-bit plane.) The story sketch-book adjusts the brightness of the original image to compensate for the loss.

X and Motif

I found that it is impossible to implement drag-and-drop easily on the X/Motif platform.

X is a multi-process environment. Another application running on the system can grab the cursor away while we are dragging an object on the sketch-book's canvas. There is no easy way to prevent it.

Besides, Motif organizes its widgets in a tree hierarchy. It is impossible to move an object from one branch of the tree to another. It is impossible therefore to move an object from one canvas to another.

For this reason, instead of drag-and-drop, I implemented what may be called click-and-follow. When we click on an object, the object will start to follow the cursor. When we click on the object again, the object will stop following the cursor. The object will also stop following the cursor if the cursor moves outside the corresponding canvas.

7.2 Time limitation

This project is done in one semester of time (12 weeks). Three aspects of this project can be improved if more time is allowed.

First, the predefined object class can be extended. Some of the customized function blocks I discussed such as AND, OR, CASE..GOTO.. ...etc. can be incorporated into the predefined object class. There are some other possibilities I can think of. Label objects that allow the user to create text illustrations might be useful. Pixmap objects that allow the user to supply a drawn picture to complement the story structures might also come in handy. Finally, the functionality of the predefined objects can be extended, e.g. the filter object can include matching mechanism for words in the titles.

Second, the test drive functions can be made more comprehensive. I only implemented play and pause. Other standard functions such as scan, search, jog...etc. can be added. Besides, the test drive function explores only the sequences in the forward direction. Backward trace can be added. Furthermore, there should be some ways to handle loops as discussed in chapter 6.

Finally, using C++ classes to represent objects would make creating new objects easier. In the current implementation, the objects are independent of each other. But in fact, they could be organized in a hierarchy. Using C++ inheritance, the hierarchy can be made much easier to maintain or extend.

Chapter 8 Conclusion

In this written account of my thesis project, I illustrated the benefits of a graphical representation of story structures. I also discussed some of the user interface issues associated with an editor which helps the user to create story structures graphically.

Both this written account and the implementation of the sketch-book grew far beyond the original estimation. (The sketch-book has around 2000 lines of codes, excluding remarks.) Therefore, this project illustrates some of the unforeseen complications which may occur in similar research projects. On the other hand, I am thrilled that many of these difficulties have been overcome in this relatively short span of time.

Finally, I hope that this project may provide some help in the progress of research in Interactive Cinema.

References

Marc Davis, Media Streams: An Iconic Visual Language for Video Annotation, Media Laboratory, MIT (1993).

Syd Field, Screenplay: The Foundations of Screenwriting, Dell Publishing, New York (1982).

Kenneth B. Hasse, Framer: A Persistent Portable Representation Library, Media Laboratory, MIT (1993).

Gideon I. Lee, A User Guide to Stratalog, Media Laboratory, MIT (1993).

Lee H. Morgenroth, Homer: A Video Story Generator, Undergraduate thesis, Department of EECS, MIT (1992).

Rebecca Rule and Susan Wheeler, Creating the Story: Guides for Writers, Heinemann Educational Books, New Hampshire (1993).

Nan C. Shu, Visual Programming, Van Nostrand Reinhold (1992).

Edward R. Tufte, Envisioning Information, Graphics Press, Connecticut (1990).

Patrick Winston, Artificial Intelligence , Addison-Wesley Publishing (1992).

Appendix A Implementation Details

This section contains the data representation and operations of different objects of the sketch-book.

```
/*************************************************************/
/*  The header file for the Sketch Book                      */
/*  by Gideon I. Lee                                         */
/*  May, 1993                                                */
/*************************************************************/

/* Constants */
#define FREE 0
#define ICON_SELECTED 1
#define RIGHT_ARROW_SELECTED 2
#define LEFT_ARROW_SELECTED 3
#define ICON_X 120
#define ICON_Y 90
#define EPISODE_TYPE 1
#define FUNCTION_TYPE 2
#define FILTER_TYPE 3
#define CURSOR_TYPE 4

/* Everything is an episode in the sketch-book.  It is the root class */
/* of all objects.  Its sub-classes are log, function, frame and      */
/* filter.  A linkage is not represented as an episode                */

typedef struct Episode_struct {
  int episode_type;
  Log log;
  int function;
  LogEditorState log_data;
  Widget frame;
  Widget form;
  Widget menu;
  Widget title_bar;
  Widget iconify;
  Widget maximize;
  Widget left_arrow;
  Widget video_icon;
  Video video;
  Widget right_arrow;
  Widget canvas_shell;
  Widget canvas;
  GC draw_gc;
  GC erase_gc;
  int ref_x;
  int ref_y;

int x1;
  int y1;
  int x2;
  int y2;
  struct Episode_struct *parent;
  struct Episode_struct *hooked_episode;
  int status;
  struct Episode_struct *sources[20];
  struct Episode_struct *targets[20];
  struct Episode_struct *cursor;
  Widget connector[20];
} *Episode, Episode_pointee;

/* A Connection represents a linkage */
typedef struct Connection_struct {
  Episode parent;
  int index;
} *Connection, Connection_pointee;


Episode_pointee root_pointee, child_pointee;

/*********************************************************************/
/* Function prototypes                                               */
/*********************************************************************/


/*******************************/
/* Operations on Episode class */
/*******************************/

Episode Episode_create_root(Widget toplevel);
/* creates a root episode with parent toplevel */

Episode Episode_create_sub(Episode episode);
/* creates a sub-episode with 'episode' being its parent */

void Episode_create_canvas(Episode episode);
/* creates a canvas object within the 'episode' */

Episode Episode_create(Episode parent_episode, Log log);
/* creates an episode with 'parent_episode' being its parent */

Episode Function_create(Episode parent_episode, int fn);
```

```
/* creates a function with 'parent_episode' beings its parent */

Episode LogFilterIcon_create(Episode parent_episode);
/* creates a filter with 'parent_episode' being its parent */

void new_episode_init_eh(Widget w, Episode episode, XEvent *event);
/* initializes a new episode as it appears on the screen */

void track_cursor_eh(Widget w, Episode episode, XEvent *event);
/* tracks the movement of the cursor and reports it to appropriate
functions */

void press_cursor_eh(Widget w, Episode episode, XEvent *event);
/* dipatches to appropriate functions if the mouse button is clicked on
*/

void redraw_connection_cb(Widget w, Episode episode,
                    XmDrawingAreaCallbackStruct *cbs);
/* refreshes the drawn connection */

void title_bar_activate_cb(Widget w, Episode episode,
                  XmPushButtonCallbackStruct *cbs);
/* callback routine for title bar activation */

void menu_activate_cb(Widget w, Episode episode,
                  XmPushButtonCallbackStruct *cbs);
/* callback routine for modifier activation */

void maximize_activate_cb(Widget w, Episode episode,
                    XmPushButtonCallbackStruct *cbs);
/* callback routine for sub-episode expansion activation */

void left_arrow_activate_cb(Widget w, Episode episode,
                      XmPushButtonCallbackStruct *cbs);
/* callback routine for in-bound activation */

void right_arrow_activate_cb(Widget w, Episode episode,
                       XmPushButtonCallbackStruct *cbs);
/* callback routine for out-bound activation */

void cancel_connection_cb(Widget w, Connection connection,
                    XmPushButtonCallbackStruct *cbs);
/* callback routine for linkage removal */

void function_selected_cb(Widget w, Episode episode,


XmPushButtonCallbackStruct *cbs);
/* callback routine for function activation */

void video_icon_selected(Widget w, Episode episode, XEvent *event);
/* callback routine for icon select action */
```

```
void video_icon_move(Widget w, Episode episode, XEvent *event);
/* callback routine for icon movement action */

void video_icon_release(Widget w, Episode episode, XEvent *event);
/* callback routine for icon settle action */

void episode_appear_eh(Widget w, Episode episode, XEvent *event);
/* event handler for new episode appearance */

void grab_button_at(Widget canvas);
/* forces grab-button at a particular canvas */

void dump_button_cb(Widget w, Episode episode, XEvent *event);
/* callback routine for destroy widget */

void capture_frame_cb(Widget w, Episode episode,
                  XmPushButtonCallbackStruct *cbs);
/* callback routine for frame capture action */

void capture_logs_cb(Widget w, Episode episode,
                 XmPushButtonCallbackStruct *cbs);
/* callback routine for logs capture action */

void pop_down_cb(Widget w, Episode episode, XmPushButtonCallbackStruct
*cbs);
/* callback routine for pop down canvas action */

/************************************************/
/* misc. operations on linkages: */
/************************************************/

void draw_track_line(Widget canvas, GC gc, int x1, int y1, int x2, int
y2);
/* draws a linkage */

void draw_forward_line(Widget canvas, GC gc, int x1, int y1, int x2, int
y2);
/* draws forward pointing edge */

void draw_backward_line(Widget canvas, GC gc, int x1, int y1, int x2,
int y2);
/* draws backward point edge */

void draw_line(Widget w, GC gc, int x, int y, int x2, int y2);
/* draws line */

GC get_draw_gc(Widget w);
/* gets a draw graphic context */

GC get_erase_gc(Widget w);
/* gets an erase graphic context */
```

```
void erase_connection(Episode parent, Episode child);
/* erase all connections in a canvas */
void draw_connection(Episode parent, Episode child);
/* redraw all connections in a canvas */

void hook_up(Episode source, Episode target);
/* creates a connection */
void remove_connection(Episode parent, Episode child);
/* removes a connection */

/
******************/
/* video functions /
/******************/

void Video_init_eh2(Widget daW, Video video, XEvent *event);
/* initializes video icon 24 bit-plane */

void Video_redisplay_eh2(Widget video_formW, Video video, XEvent
*event);
/* redraws the images of video icon */

/*******************************/
/* misc. operations on filters */
/*******************************/

void filter_button_cb(Widget w, Episode episode,
                  XmPushButtonCallbackStruct *cbs);
/* callback for filter button */


void LogFilter_done_buttonW_cb(Widget w, Episode filter,
                  XmPushButtonCallbackStruct *cbs);
/* callback for filter done button */

void LogFilter_cancel_buttonW_cb(Widget w, Episode filter,
                  XmPushButtonCallbackStruct *cbs);
/* callback for filter cancel button */

LogEditorState LogFilter_create(Log l, Episode filter, err_id err);
/* creates a log filter */

void filter_trigger_cb(Widget w, Episode filter,
                   XmPushButtonCallbackStruct *cbs);
/* callback for filter trigger action */

void filter_modify_cb(Widget w, Episode filter,
                  XmPushButtonCallbackStruct *cbs);
/* callback for filter modify action */

Episode Cursor_create(Episode parent_episode);
```

```
/* creates a cursor */

int cursor_x(Episode parent_episode);
/* reports the x-position of the cursor */

int cursor_y(Episode parent_episode);
/* reports the y-position of the cursor */

void cursor_shift(Episode parent_episode);
/* shifts the position of the cursor position */
```