Customized Data Visualization Using Structured Video

by **Kathleen Lee Evanco**

Bachelor of Science, Computer Science and Engineering
Massachusetts Institute of Technology
May 1993

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

Master of Science in Media Arts and Sciences at the
Massachusetts Institute of Technology
February 1996

Author: _____

Program in Media Arts and Sciences
November 10, 1995

Certified by: _____

V. Michael Bove, Jr.
Associate Professor of Media Technology
Program in Media Arts and Sciences
Thesis Supervisor

Accepted by: _____

Stephen A. Benton
Chairperson, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

Customized Data Visualization Using Structured Video

by **Kathleen Lee Evanco**

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
on November 10, 1995
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences

Abstract

Structured video describes a video sequence in terms of its component structural parts and a set of instructions describing how to recombine them. Structured video research has primarily focused on entertainment applications such as creating and displaying movies. In contrast, the thesis presented in this paper emphasizes the advantages of structured video as a tool for visual communication of information. This thesis expands the current field of structured video research by using structured video as a data visualization technique in a weather information system. Using a structured video approach to data visualization allows users to customize the format of the visual presentation of information. Issues addressed include collection and parsing of information, automatic composition of video sequences, and forms of customization and interaction with the resulting video sequence.

Thesis Supervisor: V. Michael Bove, Jr.
Title: Associate Professor of Media Technology

This work was supported in part by the Television of Tomorrow consortium

Customized Data Visualization Using Structured Video

by **Kathleen Lee Evanco**

The following people served as readers for this thesis:

Reader:

Walter Bender
Associate Director for Information Technology
MIT Media Laboratory

Reader:

Glorianna Davenport
Associate Professor of Media Technology
Program in Media Arts and Sciences

Acknowledgments

This probably looks like a thesis to you, but when I look at it, I see much more. I see two years of my life spent in a haze of not understanding who I am or what makes me tick. This paper is proof to myself that I have emerged from that haze with a new life, a grounded sense of identity, and the knowledge that I can do anything I set my mind to. Of course, I didn't find my way alone, and I would like to thank the people who made all of this - the thesis and the deeper achievements it represents - possible:

Brian - thank you for always being there. You gave me strength when I had none, you gave me motivation when I ran out, and you gave me patience when I lost my own. You are probably the only reason that I managed to find my way at all -thank you for wanting this for me as much as (if not more than) I wanted it for myself.

My advisor, Mike Bove - thank you for showing confidence in me even when I did not deserve it. Thank you for being human, and for realizing that I am *only* human.

Readers of this thesis, Glorianna and Walter - thank you both for your patience and your help, even though I waited till the last minute to ask.

My dear friends, Beth, Nancy, Lisa, Kacey, Jim and Roger - thank you for just being there - always. I absolutely could not have done this without your constant reassurance.

My co-worker, Stefan - for providing endless support for my work, and for making my thesis a reality.

Jon Orwant and Dan Gruhl for helping me learn to write perl scripts even though half of what you said sounded like it was in another language.

The gardeners, JSheena, Araz, Karrie, Jill, Shawn, Michelle and Chris and previous gardeners, Brett, Mark, and Cris - Thanks for being there during the long days and even longer nights. Thank you for being a part of my life. You've all given me more than you will ever know.

My family, Mom, Dad, Nicky, Chris, Colleen, Lance, Brennan, Katelyn, Mom and Dad Brown and Paul - Thank you for your support and confidence. I know that lately it seemed like I dropped off the face of the earth. I just wanted to let you know I'm back.

Table of Contents

List of Figures

Chapter 1

Introduction

Structured video describes a video sequence in terms of its component structural parts and a set of instructions describing how to recombine them. Structured video research has primarily focused on entertainment applications such as creating and displaying movies. In contrast, the thesis presented in this paper emphasizes the advantages of structured video as a tool for visual communication of information. This thesis expands the current field of structured video research by using structured video as a data visualization technique in a weather information system. Using a structured video approach to data visualization allows users to customize the format of the visual presentation of information. Issues addressed include collection and parsing of information, automatic composition of video sequences, and forms of customization and interaction with the resulting video sequence.

1.1 Data Visualization

"Whatever relates to extent and quantity may be represented by geometrical figures. Statistical projections which speak to the senses without fatiguing the mind, possess the advantage of fixing the attention on a great number of important facts."
Alexander von Humboldt, 1811

Alexander von Humboldt was one of the earliest scientists to realize the advantages of representing information graphically. In 1817 he prepared the first isothermal chart showing equal lines of temperature. Surprisingly, examples of graphical representations of data can be traced back to the 1600s. While the concept of data

visualization is not new, the type of data we are able to represent graphically is increasingly more complex, because of the introduction of computers and computer graphics to the field. From its humble beginnings with simple maps and charts, data visualization has grown to become one of the most widespread and technologically advanced fields [Ear93].

Computer-aided data visualization began in the late 1960s, when the computer allowed scientists to calculate answers to large, complex problems which until that point had been too computationally intensive to solve. With this significant increase in the complexity of data came a renewed need for methods of understanding that data. A surge of interest in the field of data visualization occurred in 1987 after the release of the SIGGRAPH panel report *Visualization in Scientific Computing* [McC87]. Researchers from a wide variety of fields began looking for theoretical models and display techniques to achieve a greater understanding of their research. Most of the data visualization research today focuses on modeling very complex systems and almost always involves data in more than two dimensions. However, both a simple bar chart and an abstract graphical representation of n-dimensional space are all examples of data visualization.

In the simplest sense, visualization is making pictures out of data. Jose Encarnacao points out in *Animation and Scientific Visualization* that there are essentially two problems to face in creating a data visualization system:

> "Initially, visualization may be considered as a simple technical problem of how to map information into visual forms. In fact, it involves a communication problem of great significance. The mapping process needs to be adapted to the specific goals of the visualization. Moreover, mechanisms must be provided to support the exploration of generated visual forms." [Ear 93]

This quote suggests that in order to determine how numerical information in a data visualization system will be mapped to a graphical representation, one must first determine the goals of the visualization. For this thesis, the goals of visualization were very simple. First, the visualization should be a video sequence, where a video sequence refers to one or more sequential video frames. Second, the visualization should be modeled after a visualization currently available in a traditional video format in order to show that structured video can achieve similar results. Third, the visualization should be easily understandable.

The above quote also mentions the importance of exploring the generated visualization. The issue of providing mechanisms to support the exploration of the resulting visualization is a focus of this thesis; this thesis will show that structured video is an excellent way of providing interaction that allows exploration of the data space.

1.2 Structured Video

The Information and Entertainment Systems Group of the MIT Media Laboratory has been exploring an image representation known as "structured video" for the past few years. Structured video (also known as model-based video [Har89] or analysis-synthesis video [Mus89], [Bov94b]) represents moving images in terms of component parts such as backgrounds and actors instead of sequences of frames. These parts may be objects with uniform depth (2-D), objects with corresponding surface depth values (2 1/2-D), or particle databases (3-D). These components, along with a set of instructions describing how they should be assembled, are used to produce video sequences.

Structured video provides advantages over traditional image-based coders. For example, structured video allows complex video sequences to be reduced to simple parametric expressions which leads to high compression rates. Using a traditional coder, a 1 sec-

ond video sequence at NTSC resolution would require transmission rates on the order of 40 megabytes per second. Assuming that the structured video receiver has enough memory to locally store the component objects (see Section 2.1), and that the objects can be transmitted in advance, then only the instructions describing how to construct the sequence need to be transmitted. Depending on the language used for the instructions (see Section 2.2), the required transmission rates could drop to the order of kilobytes per second.

Another advantage of structured video is that it uses knowledge of the structure of the scene being represented to allow enhanced user interaction [Bov 94b]. In current commercial applications, "customized video" can mean anything from video on demand to interactive television. However, because of the limits of the frame-based video representations used in these applications, once the video sequences have been produced, they cannot be significantly altered. This limits the interaction in such systems to high-level choices between different predetermined video selections as opposed to low-level manipulation of the video's structural components. Because structured video creates video sequences by compositing a set of objects, the sequence can be altered at any time, including during display, thus allowing much higher levels of interaction. Past research has shown that such manipulation is achievable with reasonable processing power (see Section 2.1). In a structured video system, the level of interaction is not dictated by the video representation, but instead by the method of interaction used.

A third advantage of structured video is its ability to adapt to different display environments. Again, because the video sequence is represented by its component parts and a set of instructions defining how to reconstruct it, the video can be adapted for a variety of different displays. For example, one script could describe how to construct the sequence for a wall size projection while another could describe how to construct the sequence for a wrist watch size display. Because traditional frame-based representations do not have this versatility, they have prevented researchers from focusing

on how the display environment should affect the creation of the video sequence. This is one of the most interesting questions introduced by such a flexible video representation and it is a focus of current research.

1.3 A New Direction for Structured Video

While the Entertainment and Information Systems group of the MIT Media Laboratory has developed a structured video decoder, it is currently only being used to create movies. This focus on entertainment applications overlooks a valuable use for structured video as a method of visually displaying information. This thesis expands the breadth of the current research by examining how structured video principles can be applied to a data visualization system.

Structured video is useful for a data visualization application in two ways. First, the main objective of such a system is to allow a user to present data in a coherent graphical fashion. Because structured video uses component video objects and a script describing how to assemble them, it is conducive to assembling data. Each datum can be represented by a video object, while the instructions explain when, where, and how to place that datum in the overall data space. Second, data visualization systems require a means of interacting with the visual product. Structured video easily provides this functionality by allowing direct interaction with the video sequence. Because structured video takes into account the structure of the scene, it is advantageous for a system where the structure of the scene conveys information.

1.4 Why Weather?

Despite the large quantity of weather information available through television, radio, newspapers and the World Wide Web, finding the answers to personal questions is still very difficult. For example, imagine yourself preparing to embark on a trip. You would like to fly from Boston to San Francisco and you are given the option of

making a connection in either Denver or Chicago. Which do you choose? Both airports are notorious for delays due to weather. You would be more able to made an educated decision if you could instantly find out what the weather is like in both cities.

Or, perhaps you are planning on driving from Boston to Cleveland. and there are two routes you could take. One is shorter but takes you straight through the snow belt caused by the Great Lakes. The other is a longer, more southerly route that takes you through the Pocono Mountains. Which do you choose? If you had access to a weather system that could show you the weather along the two routes, you could plan the safest, most efficient trip. This thesis attempts to implement a customizeable weather information system that will allow users to answer very user-specific weather related questions.

Weather data was selected for three reasons. First, while customization and interaction are important in all data visualization systems, they are particularly desirable in an information system that is used by many different people from many different backgrounds, like a weather information system. When many people are using an information system, it must provide a large range of capability in contrast to a highly specialized data visualization system that can cater to a small group of people in a specific field. The diversity among users of a general weather system will require a great deal of flexibility to accommodate the user's customization requests. Structured video can satisfy this need.

Second, while meteorological modeling and forecasting is an extremely complicated science, there are already a large number of systems that perform these functions. One such system is the "Weather" program used by the Earth, Atmospheric and Planetary Sciences Department at MIT. This program uses sophisticated models to predict weather patterns, but the resulting data is at best readable text and at worst, cryptic sequences of alphanumerics. The weather information system in this thesis transforms the data gener-

ated by the "Weather" program into a graphical representation. Weather data was selected for this thesis because of the existence and widespread availability of systems like "Weather." By using another system to perform the complicated modeling, this thesis was free to focus on how data created by these systems could be visualized.

Finally, a weather system was selected because of the existence of video based weather visualization systems: television weather forecasts. The current method of using video to visualize weather information provides a framework for creating similar visualizations using structured video, and for showing the advantages of structured video over traditional frame based video representations.

1.5 Thesis Overview

This thesis examines the use of structured video as a form of data visualization. It shows how video sequences can be automatically generated to represent data provided by an information system, specifically a weather information system. It also shows how the video stream produced by the system can be customized through user interaction.

Chapter 2 will describe the structured video decoder that this system uses. It begins with an explanation of the hardware and software of the Cheops Imaging System which is the platform of the decoder. It also addresses two scripting languages that have been developed as methods of describing how to composite video objects.

Chapter 3 details the implementation of the system. This chapter discusses how the system collects data, how users request information, how the system parses the data, how the system generates the scripts, and how the user interacts with the system.

Chapter 4 describes the different kinds of video objects and how they are created.

Chapter 5 concludes with the results and evaluation of the system and ideas for future work.

Chapter 2

A Structured Video Decoder

The challenge of implementing a structured video decoder lies in the large variety of data representations that must be supported. For example, the types of video objects supported by the decoder should include, but not be limited to, arrays of pixels with constant depth (2-D), arrays of pixels with associated depth values (2 1/2-D), computer graphics objects (3-D), and layered objects with associated intensity, velocity and opacity maps (2-D or 2 1/2-D). In addition, the decoder should be extensible enough to support new object representations as the need arises.

A generic structured video decoder must also support a large range of instructions. For example, it must be able to function both as a three dimensional interactive graphics rendering system and as a standard hybrid decoder. Figure 2.1 shows a proposed pipeline for such a generic structured video decoder. [Bov94a] A more detailed explanation of these issues can be found in "Real-Time Decoding and Display of Structured Video" by Bove, Granger and Watlington. [Bov94a]

2.1 The Cheops Imaging System

One example of a structured video decoder has been implemented on the Cheops Imaging System. Cheops is a compact, modular platform for acquisition, processing, and display of digital video sequences and model-based representations of moving scenes [Bov93]. Cheops is both a prototype architecture for programmable video decoders and a tool for research which requires real-time video manipulation [Bov93, Bov91].

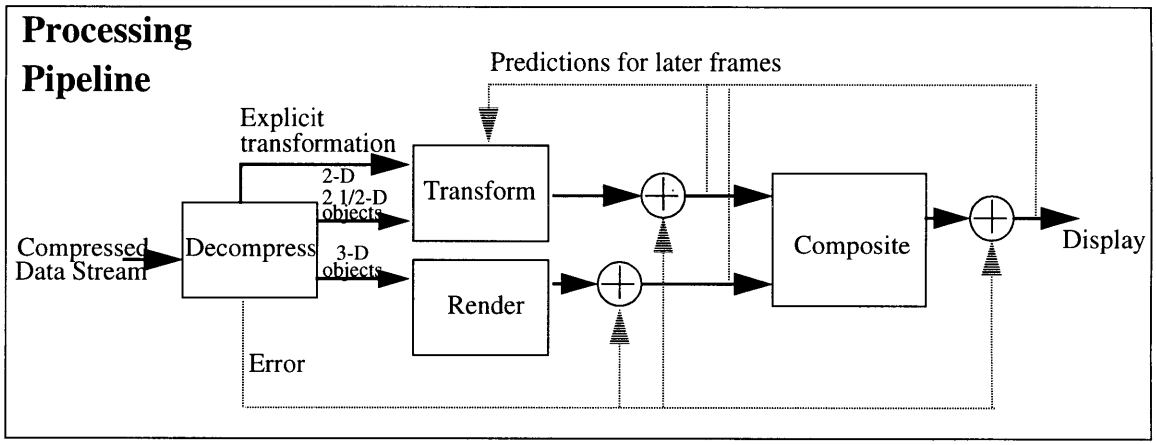

Figure 2.1 Processing pipeline of a generic structured video decoder.

Implementing a structured video decoder presents two main processing challenges. First, the decoder must be able to interpret a set of instructions and composite video objects as specified by those instructions in real time. In some cases, the decoder must also support real-time interaction. Second, the decoder must be able to store the large quantity of video objects and retrieve them quickly enough for real-time display.

Cheops' design makes it ideally suited for acting as a structured video decoder. Cheops overcomes the challenge of compositing objects in real-time by implementing a set of basic, computationally intensive operations in specialized stream processors. These processors perform functions like convolution, correlation, matrix algebra, block transforms, and spatial remapping. In addition, Cheops' modular design allows the machine to be reconfigured as technology or computational needs change. Finally, a general purpose processor in conjunction with a software resource manager [She92] allows the system to execute stream operations in parallel whenever possible. These attributes enable Cheops to perform real time video processing that is currently impractical on general purpose computers. This ability is essential for a structured video decoder that has to interpret a set of instructions and composite

video objects in real time. In addition, this real time processing ability allows applications to provide user interaction options that are not feasible on other systems.

With regard to the problem of accessing large quantities of video objects, Cheops can easily store and manipulate the video objects required for the composited sequence because it can be configured to have up to 32 gigabytes of RAM. This, in conjunction with the high bandwidth Nile Buses provides Cheops the ability to store and retrieve the video objects in a timely fashion.

For information about the Cheops imaging system, see [Bov91], [Bov93a], and [Bov95].

2.2 The Isis Scripting Language

Isis is a platform independent scripting language which can be used for developing multi-media applications. It can be used either as a stand-alone programming environment or as an interface to C functions that perform a set of specialized operations. [Aga95b] Two examples of specialized operation sets are the structured audio and video packages described in [Aga95c] and [Aga95d]. Isis can best be described as having three levels. The highest level is the platform independent Isis interpreter. The second level consists of the built in Isis functions implemented in the structured audio and video packages. Finally the packages rely on machine specific libraries to produce the video sequence for the platform being used.

The two most noteworthy attributes of the implementation of Isis are that Isis data structures are based on arrays rather than linked lists and that the language core of Isis is small compared to other programming languages. The fact that it is based on arrays makes Isis an efficient scripting language for many structured video applications. The small language core makes it easy to learn and use. This paper will not address the syntactical implementation of Isis except to say that it resembles the syntax of Scheme. Instead, it will

focus on the functionality provided by Isis and how Isis is used to create the structured video application implemented in this thesis. Complete documentation of the Isis scripting language and its syntax can be found in [Aga95b], [Aga95c], and [Aga95d].

In general, Isis provides all of the basic functionality needed in any programming language including data types, variables, constants, lists, conditionals, expressions, and input/output operations. Three of the basic functions that were particularly useful in implementing this thesis were user-defined types, user-defined procedures and file loading operations. User defined types were useful because weather-specific data types (like longitude and latitude pairs) could be defined. This made generating the scripts easier because the C code did not have to convert the data from a weather representation to an Isis representation. For example, a "LongLat" type and a procedure that converts a "LongLat" from longitude and latitude to x, y pixel coordinates were implemented in Isis. Then, all the C code has to do is generate a script that loads that type and procedure; it does not have to know anything about the coordinate system that Isis uses. Thus, by being able to define types, write procedures and load files that contain these predefined types and procedures into the generated Isis scripts, the amount of work required to generate the scripts was significantly reduced.

In addition to the general functionality, Isis provides one particularly innovative data structure: the timeline data structure. A timeline structure is initially empty, but can have values at any real numbered time on the timeline. The values can be anything from boolean values to user-defined types and do not have to be uniform throughout the timeline. This data structure was especially useful in implementing a weather system that deals with large sets of time series data. For example, a set of forecasted temperatures starting at the current time and extending for 57 hours at 3 hour intervals can easily be implemented as a timeline. It is important to note that timelines do not necessarily have to be correlated to physical time; they can be correlated to any ordered list of numbers.

At the second level, the structured video package provides a framework for creating structured video applications. This package allows the manipulation of 6 kinds of entities: *cameras*, *environments*, *actors*, *objects*, *windows*, and *engines*. [Aga95c] The *camera*, in essence, is the viewer. The *camera* information specifies where the viewer is in relation to the scene being displayed. In the weather system, this information controls what part of the map is visible in the window and at what scale the map is drawn. The *environment* provides information about how the viewer perceives the scene being displayed. Lighting information is one example of information that would be stored in the *environment*. Again, none of this information was needed for the weather information system.

The *actors* are the things in the production. For the weather system implemented in this thesis, each datum for each city is represented by an *actor*. *Actors* can be positioned and transformed in variety of ways including translation, rotation, warping, and scaling. Each *actor* must be associated with an *object* which is a graphics data file. An *object* can be used by multiple *actors* simultaneously, but an *actor* can only be associated with one *object* at a time.

The *window* describes how the scene will be translated to the window on the display. The *window* information includes window size and position which, in the weather system, is provided by the user (see Section 3.2.1). Finally, the *engine* is the entity that performs all of processing and compositing. An *engine* refers to a *camera*, a *window*, an *environment*, and a set of *actors*. An application can have multiple *engines* which offers the capacity to control several presentations at once.

The weather system implemented in this thesis is the largest structured video application implemented in Isis to date. Two other structured video productions, "The Museum Movie" and "Yellow Wallpaper" have been implemented in Isis. Each of those applications uses a maximum of 25 actors. The weather system uses a minimum of 128 actors and can easily exceed 1000 actors. In addition,

the Isis scripts created by the weather system can be as large as 25,000 lines of code, much larger than the Isis files for either of the other two applications. This application establishes Isis' robustness as a scripting language that can define a variety of structured video applications.

Chapter 3

Implementation

The implementation of the weather system described in this thesis can be divided into four steps. The first step is to download weather data from a weather database. The second step is to provide a mechanism for requesting information. The third step is to parse the weather data files and extract the relevant information. The final step is to generate Isis scripts describing how to composite the video objects in relation to the data. The Isis scripts can then be executed on a structured video decoder where the compositing, display and interaction takes place. The next sections of this chapter will describe each of these steps in detail.

The current implementation of the system is limited to 8 states: Colorado, Connecticut, Maine, Massachusetts, Minnesota, New Hampshire, Rhode Island, and Vermont. This is not a factor of the availability of data, but instead a factor of the number of video objects required to support all 50 states. However, the organization of the system is versatile enough to add more states should the need arise (See Section 3.6).

3.1 Collecting Weather Data

There are many sources of weather data available to the general public in a large variety of formats including text and hypertext, audio, and video. In video format, there are weather forecasts on every major television network and an entire cable television channel devoted to presenting weather information. In addition, the audio portion of a televised weather forecast can often stand alone as an audio presentation of weather information. Other audio pre-

sentations include telephone numbers providing recorded information and radio station forecasts. Radio is such a popular medium for distributing weather information that manufacturers build radios that only tune to the weather station. In written format, weather data is available in every newspaper, and with the growth of the World Wide Web, there are literally hundreds of weather information sites on line.

Despite the proliferation of weather information, on closer examination, the data set is actually quite small; the same information is being presented in many different ways. Most weather data originates from the National Weather Service (NWS). The National Weather Service collects weather observations from around the world, and then generates a variety of forecasts using different models. The results of the models are then passed to vendors who package the data and distribute it. The large number of vendors accounts for the enormous variation in presentation and format.

3.1.1 Data Set

The weather system implemented in this thesis only uses a small subset of the information available for two reasons. First, the sheer quantity of information available would be too much to parse. Second, there is so much overlap of information in the various formats that parsing more data does not necessarily add information to the system. The data files come from MIT's Department of Earth, Atmospheric and Planetary Sciences (EAPS) "Weather" program. "Weather" provides access to EAPS' extensive database of United States weather data. Of all the data available from the "Weather" database, three data sets are used for each city. These data sets were selected for their regularity of format, which aids in parsing, and for their combined quantity of information with as little overlap as possible.

3.1.1.1 Surface Observations

The first of the three sets of data is the surface observation data (sa) which provides information about observed cloud cover and weather type, temperature in degrees Fahrenheit, dewpoint in degrees Fahrenheit, humidity by percentage, wind direction in degrees, wind speed in miles per hour, gust speed in miles per hour, pressure in millibars, altimeter in inches of mercury and visibility in miles at hourly intervals. The surface observations are provided in the textual format shown in Figure 3.1

```
weather -c safulldecode BOS 1

The weather observed at BOSTON (BOS) at 03:50 PM EDT was:
The skies were thinly scattered with clouds.
Temperature: 57F (14C)  Dewpoint: 27F (-3C)   Relative Humidity: 31%
Winds from the NW (310 degs) at 18 mph gusting to 24 mph.
Pressure: 1023.7 millibars.  Altimeter:30.23 inches of mercury.
Visibility: 20.0 miles.
```

Figure 3.1 An example surface observation data file.

3.1.1.2 Nested Grid Model

The second data set is the Nested Grid Model (ngmmos). This data file provides forecast information that is created by a combination of interpolation and regression estimation. Nested Grid refers to the method used for interpolating values between observed weather data points. MOS stands for Model Output Statistics and refers to a regression estimation technique based on past weather developments. [Par88] This data provides forecast information about minimum and maximum temperature in degrees Fahrenheit, temperature in degrees Fahrenheit, dewpoint in degrees Fahrenheit, cloud cover, wind direction in tens of degrees, wind speed in miles per hour, probability of precipitation for 6 and 12 hour periods by percentage, precipitation type, and snowfall at three hour intervals for 54 hours. An example nested grid model data file is shown in Figure 3.2.

```
weather -c ngmmos BOS 1
-BOS  ESC  NGM MOS GUIDANCE  10/17/95  1200 UTC
DAY /OCT  17 /OCT  18                 /OCT  19                  /
HOUR   18 21 00 03 06 09 12 15 18 21 00 03 06 09 12 15 18 21 00
MN/MX                    45          68          53          66
TEMP   55 55 51 49 48 47 50 60 66 67 62 59 56 55 56 61 64 63 59
DEWPT  29 29 31 34 36 37 39 41 42 44 47 48 49 49 48 49 48 49 49
CLDS   CL CL CL CL CL SC SC SC SC BK BK SC SC BK OV BK BK OV OV
WDIR   29 29 27 24 23 22 21 22 22 22 24 24 25 25 26 27 15 15 14
WSPD   16 15 09 06 05 07 09 12 16 16 11 09 08 06 06 06 07 08 05
POP06        0     2     6     0     0     6     6     7    12
POP12                    5           0          10          15
QPF        0/    0/    0/0   0/    0/0   0/    0/0   0/    0/0
TSV06     0/ 5  0/ 2  2/ 7  2/ 5  5/ 7  6/ 5  3/ 1  5/ 2  4/ 2
TSV12           0/ 4        3/10        9/ 9        7/ 2
PTYPE  R  R  R  R  R  R  R  R  R  R  R     R     R     R     R
POZP   1  1  1  0  0  0  0  0  0  0  1     0     0     0     0
POSN   0  0  2  0  0  1  0  0  1  0  0     0     0     0     0
SNOW       0/    0/    0/0   0/    0/0   0/    0/0   0/    0/0
CIG    7  7  7  7  7  7  7  7  7  7  7     7     7
VIS    5  5  5  5  5  5  5  5  5  5  5     5     5
OBVIS  N  N  N  N  N  N  N  N  N  N  N     N     N
```

Figure 3.2 An example nested grid model data file.

3.1.1.3 Climatological Report

The third data set is the climatological report (climo) which is created once per day. Climatological data files are usually only available for large cities, and their format is not as standardized as the formats of the other two files. The climatological report provides information about the previous day's minimum and maximum temperatures in degrees Fahrenheit, and the mean temperature and departure from the normal mean temperature in degrees Fahrenheit. It also provides the current day's normal high and low temperatures in degrees Fahrenheit, record high and low temperatures in degrees Fahrenheit, and the years those records were set. The climatological data includes heating and cooling information for the previous day, month and season in degrees Fahrenheit, and the precipitation for the previous day, month and year, as well as the normal precipitation amounts for the month and year in inches. In months when

snow occurs, this file also provides snowfall information for the previous day, month and season in inches. Information about the previous day's fastest two minute wind and peak gust in miles per hour as well as the sunrise and sunset times for the current day and the next day can also be found in this file. See the example climatological data file in Figure 3.3.

3.1.2 Data Retrieval

The weather information system implemented in the thesis requires access to each of the three data files described in Section 3.1.1 for each city in the system. As mentioned at the beginning of Chapter 3, only the New England states, Colorado, and Minnesota have currently been implemented. However, these 8 states have 94 cities with National Weather Service stations. Thus the system needs access to 282 data files. In order to parse the data, the data files need to be locally accessible to the weather system. A perl script is used to download the most recent data files from the EAPS database to a local data directory.

The script named "weatherfiles.pl" was written in perl which is "a language for easily manipulating text, files, and processes." [Wal91] "Weatherfiles.pl" must be executed inside a directory named "Data" which has the subdirectory structure shown in Figure 3.4. The first level of subdirectories under the "Data" directory are named after the two letter abbreviations of the implemented states. Within each state's directory, there is a subdirectory for each city in that state with a National Weather Service station. These city directories are named after the three character NWS station codes. Within each city directory there are three data files named "sa," "climo," and "ngmmos" as described in Section 3.1.1. "Weatherfiles.pl" traverses the directory structure of "Data" to compile a list of NWS station codes for which information is requested. It also relies on this directory structure to know where to put the data files as they are being created.

```
weather -c climo BOS 1
CSUS2 KBOS 170519
CLIBOS

CLIMATOLOGICAL REPORT (DAILY)
NATIONAL WEATHER SERVICE BOSTON MA
121 AM EDT TUE OCT 17 1995

...TEMPERATURE...
HIGH YESTERDAY... 58
LOW YESTERDAY.... 46
MEAN TEMPERATURE. 52      DEPARTURE FROM NORMAL...MINUS 3

NORMAL HIGH FOR TODAY....62
NORMAL LOW FOR TODAY.....46
RECORD HIGH FOR TODAY....89 SET IN 1947
RECORD LOW FOR TODAY.....27 SET IN 1886

...DEGREE DAY DATA...
HEATING
YESTERDAY...     13
MONTH.......     75
SEASON......    191        DEPARTURE.....MINUS  17

COOLING
YESTERDAY...      0
MONTH.......     15
SEASON......    846         DEPARTURE....PLUS  168

...PRECIPITATION IN INCHES...
YESTERDAY..............TRACE
TOTAL FOR THE MONTH......4.29
NORMAL MONTH TO DATE.....1.60
TOTAL FOR THE YEAR......24.64
NORMAL YEAR TO DATE.....31.58

...WIND DATA...
FASTEST 2-MIN WIND YESTERDAY.........32 MPH FROM THE W
PEAK WIND GUST YESTERDAY.............41 MPH FROM THE W

...ASTRONOMICAL DATA...

SUNRISE TODAY.......659 AM EDT
SUNSET TODAY........600 PM EDT
SUNRISE TOMORROW....700 AM EDT
SUNSET TOMORROW.....558 PM EDT

...END...
```

Figure 3.3 An example climatological data file.

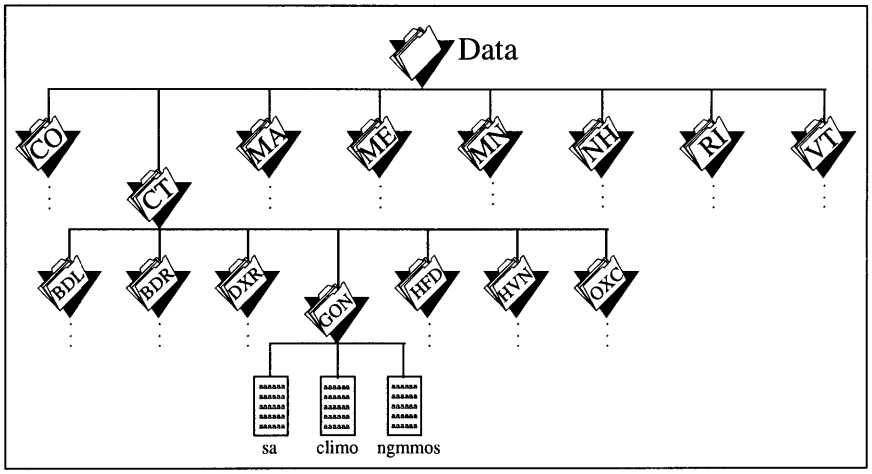

Figure 3.4 Diagram of Data directory tree structure.

After logging into the EAPS computer, "weatherfiles.pl" sends commands of the syntax "weather -c safulldecode BOS l" to the EAPS server. "Weather" is the name of the EAPS program that provides access to the weather database in either interactive or command line modes. The "-c" command line option specifies which type of data to retrieve (in this case, surface observations; the other two data files are created using "-c ngmmos" and "-c climo"). The next three letters are the National Weather Service code for the city (in this case, Boston). Finally, the "l" asks for the last or most recent data set. The script then takes the text returned by these commands and writes them to the appropriate local file as determined by the directory structure described above (in this case, Data/MA/BOS/sa). Once all 292 commands have been executed (one for each data file) the necessary weather files are locally accessible to the weather system.

While "weatherfiles.pl" does serve its purpose, it does not necessarily do it in the most efficient fashion. The drawback of this implementation is that it takes approximately two hours to download all of the data files. This is caused by two problems. First, the EAPS computers are exceptionally slow; this problem cannot be fixed within the scope of this thesis. The second problem is that not each

of the three data files is available for every city in the system. When the EAPS "weather" program is asked for information that does not exist, it spends a lot of time searching for the information before returning. The script execution time could be reduced by creating a more sophisticated perl script that determines when the EAPS computer is taking too long to respond, and then kills that command and moves to the next one. This is one suggestion for future enhancements to the system.

3.2 Requesting Information

One function of any information system is to provide a method of requesting a data set. In a weather application, users need the ability to request information for cities or regions of interest. In current weather applications data is selected in a variety of fashions ranging from simple text based interfaces where the user enters the National Weather Service code for each city (for example, the weather program on Athena), to more sophisticated graphical interfaces where the user can click on a map to select a city (for example, the University of Michigan Interactive Weather Browser at http://rs560.cl.msu.edu/weather/graphicalinteractive.html). Regardless of how sophisticated the interface is, all weather systems provide the same functionality: allowing the user to select a data set to view.

3.2.1 Request Files

The mechanism for requesting information about cities in the weather system implemented in this thesis is to create a request file. Request files must be in the format shown in Figure 3.5. The first line of the request file is the name of the Isis script file to be generated. The second line must have four real numbers separated by spaces. These numbers represent the coordinates of the lower left corner and upper right corner of the display window. The third line indicates the display mode and can currently be one of two values, VGA or DEFAULT, both of which must be typed in capital letters.

The display mode information affects the placement of the display window on the screen. DEFAULT centers the display window on a high resolution monitor, while VGA centers the window for display on an NTSC device, such as a television monitor, or projection screen. In general, the DEFAULT setting should be used. The VGA option was implemented in order to facilitate demonstrating the system at the Media Lab. Finally, the request file contains a list of cities (listed as CITY, STATE in capital letters) for which information is desired.

```
Connecticut
-256.0 -256.0 256.0 256.0
DEFAULT
BRADLEY FIELD, CT
BRIDGEPORT, CT
DANBURY, CT
GROTON, CT
HARTFORD, CT
NEW HAVEN, CT
WATERBURY, CT
```

Figure 3.5 An example request file.

3.2.2 Region Request Files

One interesting ability of the weather information system implemented in this thesis is that users can define regions. A region is a named list of cities. When a region name is included in the list of cities in a request file, as in Figure 3.6, the program first looks for the region name in the list of cities it knows about. If the region name is not found in that list, the program looks for a file named "Data/*regionname*/request," where *regionname* is the name of the region as typed in the request file. If a file by that name exists, the list of cities inside the file "Data/*regionname*/request" is substituted into the list of cities in the request file. Note that region request files are different from standard request files in two ways. First, there is no header information in a region request file; it simply contains a list of cities or regions. Second, the name of the region must be a

subdirectory of the "Data" directory, and the region request file must be in that subdirectory. If no request file is found under these circumstances, the systems treats the name as an invalid request and will print a message that there is no data available for that city.

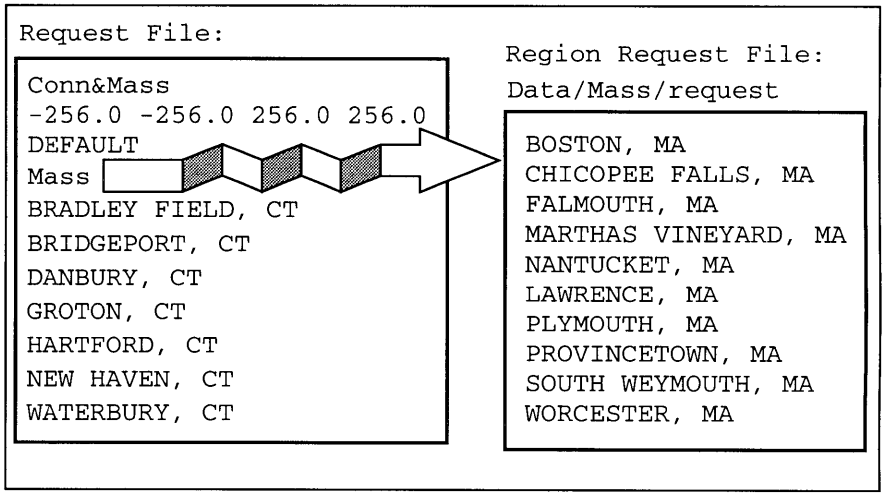

Figure 3.6 The name Mass is expanded to be the list of cities included in the region request file Data/Mass/request.

The limitation of this implementation is that it creates errors when "weatherfiles.pl" is executed. Because states and regions are both implemented as subdirectories of the "Data" directory, "weatherfiles.pl" assumes that the region name is a state, and that "request" is the NWS code for a city. Of course, when the command is passed to the EAPS "weather" program asking for information about the city "request" "weather" returns errors. Likewise, when "weatherfiles.pl" tries to write the empty data files in the subdirectory "request," the script generates errors because "request" is not a directory. While these errors may be bothersome, they do not impair the function of "weatherfiles.pl." The errors could be eliminated by making "weatherfiles.pl" more intelligently traverse the tree structure of the "Data" directory by looking only for three character NWS codes. In that implementation, "weatherfiles.pl" would know that a region request file is not a city and would not try to request that information.

3.3 Parsing the Weather Data

Once the weather data files are downloaded, and the request file has been created, the next step is to parse the data for the requested cities into a format that will facilitate the association of data with video objects. The parsing routines for the weather system implemented in this thesis were written in the C programming language. Each data file is parsed by a separate function and these functions can be found in files named "sa.c," "ngmmos.c," and "climo.c." This modular design facilitates extending the system to parse more data files should the need arise.

Each of the three parsing functions uses a different parsing technique based on the structure of the data file. For example, because the nested grid model data file is in a tabular format, the lines are all processed simultaneously and each consecutive number is read off and placed in the appropriate C structure. On the other hand, because the surface observation data file is in more of an english format, "sa.c" searches for key words and then reads the number associated with that key word. Climatological information is the most difficult to parse because there is no standard format for climatological data files. "Climo.c" was written by examining a large data set of climatological data files and determining what format they had in common. The parsing routine was then based on that intersection of formats. This has two repercussions. First, climatological data files often have more information than "climo.c" is able to parse because "climo.c" is limited to parsing information common to all climatological data files. Second, some climatological data file formats are so different from the other formats that they have to be discarded all together. When this happens, the system will print the message "Nonstandard file format *command*" where *command* is the command sent to the EAPS server that created the nonstandard data file. Finally, not all cities have all three types of data. When data does not exist, the data file will be empty and the

system will print the message "Empty file *filename*" where *filename* is the name of the empty file.

3.4 Creating the Isis Scripts

Once the data has been parsed, the weather system can use that information to create Isis scripts that will describe how to composite video objects to represent the data. The functions that create the scripts were also written in the C programming language and can be found in the file "script.c."

"Script.c" creates three different kinds of Isis files, each of which is created by a separate function. The three files and their creation are described below.

3.4.1 City Isis Files

The function in "script.c" named "CityScript" creates an Isis file for each city for which data has been requested. Each city script is named after the three character National Weather Service code for that city with the extension ".isis." For example, assuming the information system is given the request file shown in Figure 3.5, "CityScript" would create the following files:

- BDL.isis
- BDR.isis
- DXR.isis
- GON.isis
- HFD.isis
- HVN.isis
- OXC.isis

The first line in the city Isis file indicates the longitude and latitude of the city. As shown in Appendix B, a set of standard weather Isis functions have been defined, one of which converts longitude and latitude to screen coordinates based on the range of longitudes and

latitudes being displayed. Next, the city Isis script sets up one actor per set of information (i.e. temperature, dewpoint, wind speed, wind direction, etc.). Then, using the Isis timeline structure, the script defines what the value of each data set is for all the given times. If there is no value for a particular data set a a particular time, the value is set to a dummy value and the visibility is set to false. In general, each city Isis file sets up 120 actors and is approximately 450 lines of Isis code. An example city Isis file is shown in Appendix B.

3.4.2 Data Isis Files

There is only one data Isis file created for each script generated by the weather system and it is created by the function "AllCitiesScript." The data Isis file is named after the script with the extension "data.isis" appended. For example, given the request file in Figure 3.5, the data Isis file would be named "Connecticutdata.isis." The first function of the data Isis file is to load each of the city Isis files. The only other function of the data Isis file is to set up the timeline structure that is used for the data knob during display. As the user turns the data knob, a different piece of information is being displayed for each city (i.e. temperature, dewpoint, wind direction, etc.). This is accomplished by setting up a timeline that determines which actors should be displayed at each value of the data knob. For example, when the value of the data knob is 0, the system displays the cloud cover symbols for each city. Because there is one actor for each city's cloud symbol, the value of the data knob timeline at 0 is the list of all of the city's cloud symbol actors. Because there are 38 different pieces of information contained in the system, the data knob timeline has 38 entries, each of which is a list of actors. Obviously, the size of the data Isis file depends on the number of cities requested; the more cities, the larger the file. The data file for a script that implements all of the cities in New England is 650 lines.

3.4.3 Map Isis Files

The function "MapsScript" creates the map Isis file for each script created by the weather system. The map Isis file is named after the script with the extension "map.isis" appended. For example, given the request file in Figure 3.5, the map Isis file would be named "Connecticutmap.isis." In creating the map Isis file, the weather system loops through each of the requested cities and determines a list of state maps that are needed. For the request file in Figure 3.5, only the Connecticut maps are needed, but for the request file in Figure 3.6, both the Connecticut and Massachusetts maps are needed. The map Isis file loads the necessary state map graphics objects. There are at least five maps per state as described Figure 4.1. After loading the maps, the map Isis file sets up actors for each state map and places them at their proper positions according to the longitude and latitude of the center of the state and the longitude and latitude range of the entire area being displayed. Again, the size of the map Isis file depends on the number of state maps needed, but the map Isis file for all of New England is 450 lines.

3.4.4 Main Isis Files

The script that ties everything together is the main Isis file. This file is created by the function "Script" in "script.c." It is named after the script with the extension ".isis" appended. For example, given the request file in Figure 3.5, the main Isis file would be named "Connecticut.isis."

The main Isis file begins by loading the standard weather Isis definitions. Then it initializes the system. These commands are the same for every main Isis script file created by the weather system.

The main Isis file then defines the scale information that allows the structured video decoder to map longitude and latitude to screen coordinates. This includes setting the position of the display window as defined by the DEFAULT and VGA settings discussed in Section 3.2.1. This file also loads the other Isis files and sets up the

list of actors. These commands depend on information gathered during the parsing step.

Another important function of the main Isis script is that it defines the user interaction with the system. The main Isis file loads the standard weather knob Isis files that set up each of the 8 knobs as follows:

- Knob 1: Controls horizontal position
- Knob 2: Controls vertical position
- Knob 3: Controls the scale
- Knob 4: Changes the map
- Knob 5: Moves through the data sets
- Knob 6: Moves through the time
- Knob 7: Dummy knob
- Knob 8: Exits

The main loop of the main Isis file then reads the input from the knobs and adapts the display accordingly. The average size of a main Isis file is 350 lines. The equation shown in Figure 3.7 shows why automating the generation of these scripts is a necessity for this type of system.

$$53 \text{ cities} * 450 \frac{\text{lines}}{\text{city file}} + 650 \frac{\text{lines}}{\text{data file}} + 450 \frac{\text{lines}}{\text{map file}} + 350 \frac{\text{lines}}{\text{main file}} = \mathbf{25{,}300} \frac{\text{lines}}{\text{script}}$$

Figure 3.7 Lines of Isis code required to implement a script for all of New England.

3.5 Using the Weather System

The previous sections describe the logical steps in implementing the weather system, but they do not discuss how these steps work together. When the system is executed, the four steps described above are actually reduced to three steps. The first steps are the

same: the files are downloaded using the "weatherfiles.pl" script and a request file is generated. However, because the generated scripts depend heavily upon the parsed information, these two steps are tied together in the C program "makeweather" described below.

3.5.1 Makeweather

"Makeweather" is a simple program that takes two arguments: a request file, and a directory to write the scripts to. It must be executed in the directory that contains the "Data" directory. "Makeweather" first sets up the database of implemented cities which includes information about the longitude and latitude of each city and the city's NWS code. It then sets up a database of implemented states that includes information about the longitude and latitude ranges of the states. These structures provide the necessary information that cannot be obtained from parsing the weather data files.

"Makeweather" then parses the request file specified by the first command line argument and creates a linked list of cites for which information is desired. Then, for each of these cities, "makeweather" reads the three data files and calls each of the three parsing routines described in Section 3.3. Finally, "makeweather" calls each of the script generating functions described in Section 3.4. The scripts are written in the directory specified by the second command line argument.

One limitation of "makeweather" is that it only runs on DECstations as opposed to DEC 3000/AlphaStations. Unsuccessful attempts were made to compile the code on a Dec Alpha. This presents a problem when attempting to combine the script generation and script execution into one step because the script generation only works on DECstations while Isis only runs on Dec Alphas.

3.5.2 Weather: A Graphical Interface

Although the weather information system can be executed by running all of these scripts and programs individually, it would be much easier, and much more appealing if the system had a nice, simple graphical interface that tied everything together in one step. An interface was developed that allows a user to perform all of the necessary tasks, creating request files, downloading data files, and generating the scripts. The interface shown in Figure 3.8 was implemented in Tcl/Tk which is "a programming system for developing and using graphical user interface applications."[Ous94] It is executed by typing "weather" in a directory that has a copy of the executable, and that has subdirectories named "Scripts" and "Data." The Data directory must contain the directory tree structure described in Section 3.3.

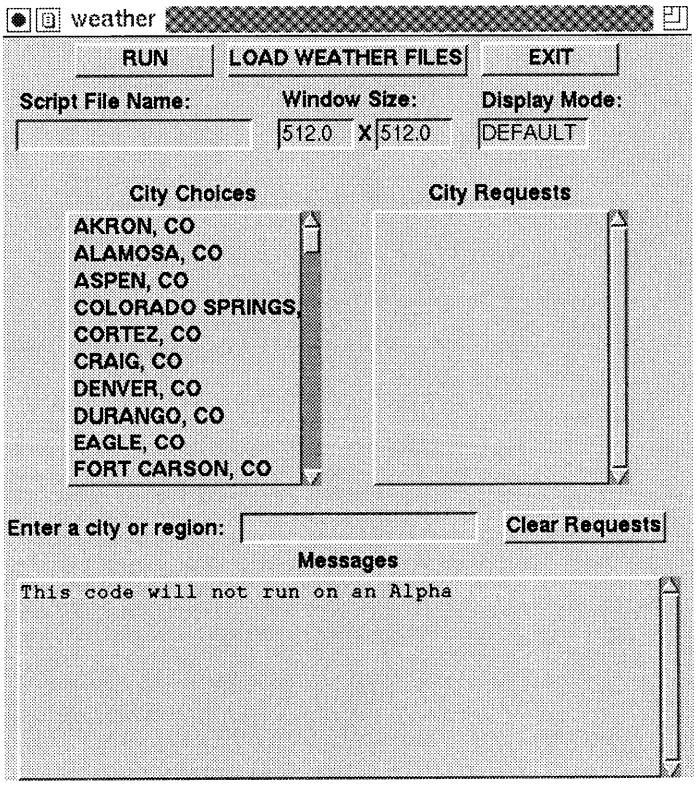

Figure 3.8 The graphical interface to the weather system.

"Weather" makes the system very easy to use. To functions of the buttons, entry boxes and list boxes are described below.

- City Choices: The city choices list box provides a list of all of the cities implemented in the system. The scroll bar to the right of the list box is used to scroll through the list. To request information for a city, double click on the city name in the city choices list box.
- City Requests: The city requests box display the current selections. When a city or region is selected it will appear in the city requests list box. The scroll bar to the right is used to scroll through the list.
- Enter City or Region: This entry box allows the user to type the name of a city or user defined region. A city must be typed in as CITY, STATE in all capitals. A region must be typed as it appears in the "Data" directory. To add the city or region to the city requests list box, press <enter>.
- Clear Requests: This button clears the entire city requests list box. There is no way to delete only one city.
- Script Name: This entry box allows the user to specify the name of the Isis script to be created.
- Window X and Y size: These entry fields allow the user to specify the size of the display window. The default values are 512 by 512.
- Display Mode: This entry box allows the user to specify the display mode which can be either DEFAULT or VGA. The default is DEFAULT.
- Run: This button performs two functions. First it writes the script name, window x and y size, display mode, and city requests to a request file. The request file is written in the directory "Scripts/*scriptname*/" where *scriptname* is the script name provided. Next is executes "makeweather" with that request file and the directory "Scripts/*scriptname*/." After clicking on the run button, the script can be executed on cheops.

- Load Weatherfiles: This button executes "weatherfiles.pl" from inside the "Data" directory, thus downloading the data files. Because the script takes a long time to execute, a message box is used to ask for a confirmation.
- Exit: This button exits the "weather" Tcl script.
- Messages: This box displays the messages printed by any of the programs or scripts executed by "weather."

3.5.3 Executing the Programs and Isis Scripts

Currently, the weather system is implemented in the directory "/cheops/demos/weather/." This directory has the necessary "Data," "Objects," and "Scripts" subdirectories. Also, regions have been defined for each state that include every city for that state. The regions are named "Colo," "Conn," "Main," "Mass," "Minn," "NewH," "Rhod," and "Verm" after the first four letters of each state.

"Weatherfiles.pl" should be executed from the directory "cheops/demos/weather/Data." "Makeweather" and "weather" should be executed from "/cheops/demos/weather/." "Weather" will write the resulting scripts in the directory "/cheops/demos/weather/Scripts/." In order to preserve convention, if "makeweather" is used manually it should be given the directory "Scripts/*scriptname/*," where *scriptname* is the desired script name, as the directory to write the scripts to.

Once the scripts have been generated, either by the "weather" script or by executing "makeweather" manually, they can be executed on cheops. Cheops must be running RAM version of "m7-src." To execute a script from the directory "/cheops/demos/weather/", type "chex ../isis-files/cheops-isis-iv Scripts/*scriptname/scriptname*.isis" where *scriptname* is the name of the script to be executed. Figure 3.9 shows examples of the display produced by scripts when they are executed on Cheops.

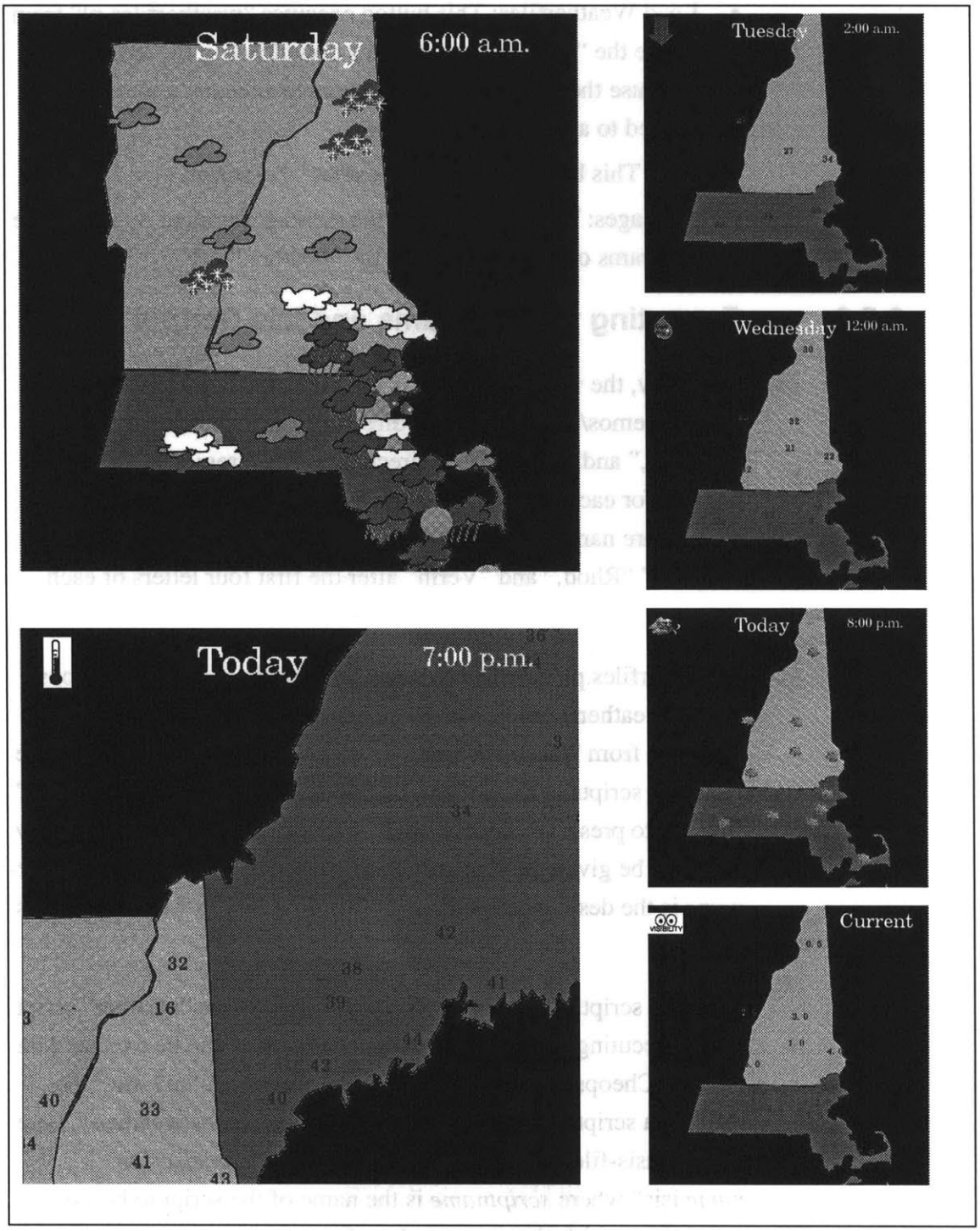

Figure 3.9 Examples of the display created by the generated Isis scripts when they are executed on cheops.

3.6 Extending the System

Because only 8 states are currently implemented in the weather system, it was designed to be extensible. One way in which it is extensible is that it allows users to define regions. However, ultimately the remaining states should be implemented. The following sections describe how to extend the system to include more cities and states.

In order to add more cities and states, the city and state databases need to be changed. The city database can be found in the file "cities.c" and is implemented as an array of structures that maintain the information for each city. The maximum number of cities in the system is defined by the constant "CITIES." To add a city, increment "CITIES" and add another array element with the proper city information: city name, state, national weather service code, longitude, latitude, and elevation. If the city added is in a state other than one of the eight implemented, the state database will also need to be updated. The process is very similar to the process described above. The state database can be found in the file "state.c" and is implemented as an array of structures that maintain the information for each state. The maximum number of states in the system is defined by the constant "STATES." To add a state, increment "STATES" and add another array element with the proper state information: the two letter abbreviation, maximum and minimum longitudes, maximum and minimum latitudes, width and height in pixels, and the number of maps the state has (i.e. Massachusetts has only one while Maine has two because it had to be tiled). Explicitly defining the city and state databases makes adding cities and states to the system more difficult than necessary. Section 5.2 proposes a converting the databases to read from files which would make this process much easier.

Once these changes have been made, "makeweather" should be recompiled. Before executing "makeweather" the appropriate state and city directories should be added to the "Data" directory, and the

data files should be downloaded by running "weatherfiles.pl." Before the scripts can be executed on Cheops, the map objects for the new states have to be created and added to the "Objects/Maps/" directory. Finally, "weather" relies on a file named "cities" to create the list of cities that appears in the city choices list box, so the new cities should be added to that file as well

Chapter 4

Video Object Database

The final step in the implementation of the weather system described in this thesis is to relate the information parsed by the system to video objects. This actually takes place when the scripts are being written. The script generation functions take as input the raw data parsed from the data files and output scripts that composite the video objects. Thus, the association of data and video objects takes place in the script generation functions. Once the data has been parsed, collecting the necessary video objects is quite simple. The difficult part is creating the set of video objects to choose from. The next sections describe the different types of video objects available and the methods used to create them.

4.1 Maps

The maps used in this system were obtained from a software package named "MapArt" made by Cartesia. "MapArt" provides maps of all 50 states, drawn to the same scale. For each state, there are 4 maps displaying counties, zip codes, rivers, and highways and cities. These maps are in PICT format and can be edited by a variety of drawing programs on a Macintosh including "MacDraw Pro" by Claris. These maps and "MacDraw Pro" were used to create 5 map video objects for each state implemented in the system. Video object maps for the remaining states can be created by following the steps explained in this section.

The process of converting maps for use in this weather system begins by opening the counties, rivers, and highways and cities maps in "MacDraw Pro." The zip code maps are not used because

they have little correlation with weather information. In "MacDraw Pro," the first step is to select all of the items and group them together. Then, scale the image by 250%. This scale factor needs to be consistent in order to have the maps line up properly on the display. For the rivers map, this is the only step. Once that map has been scaled up it can be saved again in PICT format.

For the counties map, the counties have to be colored by hand, being careful not to color adjacent counties with the same color. Making sure that adjacent counties are different colors can be tricky when dealing with counties that border other states. After the counties have been scaled and colored, the counties map can also be saved in PICT format.

The highways and cities map has to be separated into three different maps: highways, cities and state. The highways map is created by deleting everything but the highways and the outline of the state. This file can then be saved in PICT format as the highway map. The city map is created by deleting everything but the city markers, city names, and the state outline. Some city names have to be repositioned to fall completely within the state boundaries. Then the cities map can be saved in PICT format. Finally, the state map is created by deleting everything but the state border. Once again, the map has to be colored being careful to differentiate it from neighboring states. Once this is completed, the state map can be saved in PICT format.

Once all five PICT files have been created, they are opened as anti-aliased PICTs in Adobe "Photoshop." This step serves three purposes. First, the maps need to be converted to tiff format and "MacDraw Pro" does not support tiff. This is done using the "save as" command in "Photoshop." Second, alpha channel tiff files need to be created for each map image. The easiest way to do this is to use the "invert" command for black on white maps and a combination of the "threshold" and "invert" commands for the color maps. Finally, if the maps are bigger than 512 pixels in either direction,

they will have to be tiled because they will exceed the size allowed by Isis. When a map set is tiled it is important that all of the maps be tiled in exactly the same manner so that they will line up properly on the display. The "fixed size marquee" in "Photoshop" is a useful tool for uniformly tiling the maps.

Once all of the maps have been converted to tiff files, they are transferred from the Macintosh to the network. There they are converted to datfiles which are directories containing one or more related files. Datfiles generally have a data file which contains the raw image data, and a descriptor file which describes the data in the data file. For example, the descriptor file specifies the x and y dimensions, number of channels, and number of frames of the image contained in the datfile. To be used with the weather system, datfiles must have a few special properties. First, the datfiles must have 4 channels: three for RGB values and one for the alpha channel. Second, because of hardware constraints of Cheops, the width of the datfiles must be a multiple of 16. Third, the maps must be centered in the datfiles. A perl script named "tiffs2datalphacenter" is used to facilitate this lengthy process.

Finally, once a state's maps have been converted to datfiles, they are moved to the directory "Objects/Maps/*state*/" where *state* is the two letter abbreviation of the state represented by the maps. In this directory the maps are named "state.dat," "cities.dat," "highways.dat," "counties.dat," and "rivers.dat." Also in this directory is a "tiff" subdirectory where the original tiff files are stored in case they are needed.

4.2 Graphics

The weather graphics were much easier to create. They were modeled after the graphics used in current media such as television and newspaper. After looking at a variety of different weather formats, it was remarkable how universal weather graphics actually are.

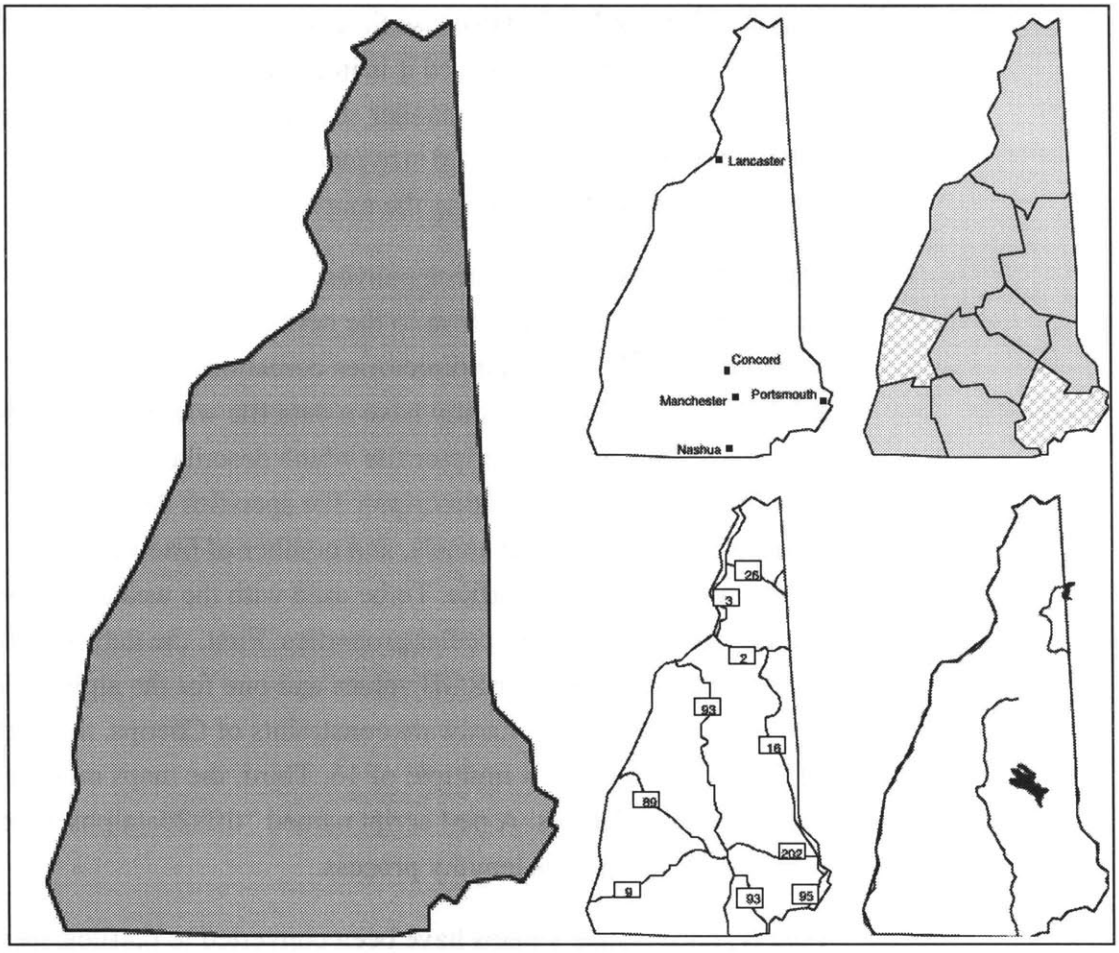

Figure 4.1 Examples of the 5 state maps used for each state in the system.

People instantly understand the message conveyed by these symbols.

Because the graphics were hand drawn, it was necessary to test their effectiveness before implementing them in the system. For each graphic, 5 people were asked what the symbol meant. If all 5 people responded correctly the graphic was added to the system. If any one of the 5 responded incorrectly or was unsure, the graphic was redesigned until it could pass the test.

The process of creating the graphics is similar to the process for creating the maps. First the graphics are drawn in "MacDraw Pro" and saved as PICTs. Then they are opened in "Photoshop," cropped to a uniform size of 64 by 64 pixels and saved as tiff files. From these tiff files, alpha channel tiff files are created. Then the graphics are transferred to the network where they are converted to datfiles using a perl scrip named "tiff2datalpha." Once they are converted to datfiles they are moved to the directory "Objects/Graphics/." Again the tiff files are moved to a "tiffs" directory in the "Objects/Graphics/" directory in case they are needed in the future.

Structured video allows objects to be either static or dynamic. The only dynamic object used in the system is the wind speed symbol. The wind speed symbol is a leaf that flutters at a speed related to the wind speed. Other examples of dynamic objects that could be useful in this system are excerpts of televised weather forecasts or satellite imagery that shows the progression of clouds (see Section 5.2). Figure 4.2 shows the graphics used in the system. The weather type graphics show what kind of weather is either occurring or being forecasted. The data set graphics are used in the upper left corner of the display window to show the user what piece of information is being displayed. The television network graphics were implemented to allow the use of digitized video clips of televised weather forecasts. However, because the digitized video clips are not functional in the current implementation, these graphics are not used (see Section 5.2).

4.3 Text

There are two types of text objects used in the weather system implemented in this thesis. The first is a 24 point Century Schoolbook datfile font that is used for numbers that change like temperatures, dewpoints, etc. The datfile font was already available but it had to be modified to have an alpha channel, to have a width that is a multiple of 16, and to be red so that it would be visible on a vari-

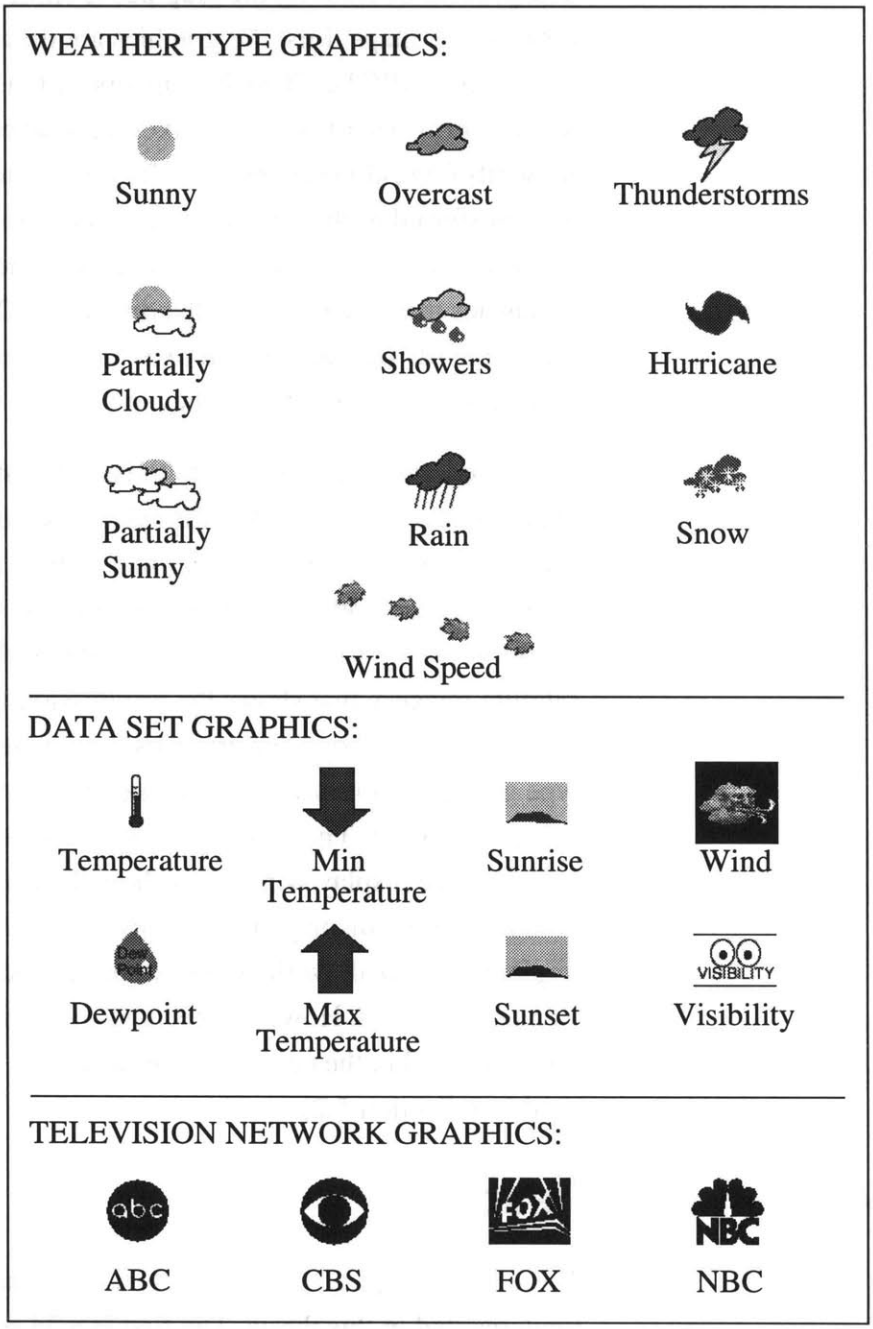

Figure 4.2 The weather graphics.

ety of colored backgrounds. This datfile was then moved to the directory "Objects/Text/."

The second type of text objects are the static text objects. Static text objects are implemented separately for efficiency reasons. In the Isis system, if text needs the ability to change in response to user interaction (for example, the text data being displayed), then it must be implemented with one actor per character. If however, the text is constant (for example the words that appear in the upper corners of the display), it can be implemented as one datfile and thus would only require one actor. Of course, once the datfile has been created, the user cannot interactively change the text.

The static text objects were typed into "FrameMaker" (made by Frame Technology Corp.) and then grabbed using "xv." They were saved as tiff files and converted to datfiles like the other graphics objects. The words implemented in this fashion include "Today," "Yesterday," "Tomorrow," "Current," and "Wind." There are also multi-frame datfiles for the days of the week, the hours ("1:00 p.m.," etc.) and the wind direction arrows (0-360 degrees in 10 degree intervals). These files are stored in the directory "Objects/ Text."

Chapter 5

Conclusions

5.1 Evaluation

The focus of this thesis was to show how structured video can be used to produce a customizeable form of data visualization. In relation to that goal, the weather system is a success. The implementation of the weather system on the structured video decoder embodied by the Cheops Imaging System and the Isis scripting language demonstrates that structured video can be used in this capacity. However, successfully implementing the weather system does not answer the question "How effective is structured video as a tool for data visualization?" This question is addressed here.

In general, structured video is an effective tool for data visualization when the data being represented can be decomposed into a set of small graphical units. For example, the weather system is most effective, and most interesting, when it is displaying the weather symbols or the leaf animations overlaid on the appropriate maps. Unfortunately, as the implementation of the weather system progressed, it became apparent that only a small subset of weather data is really graphical in nature. Much the data is displayed as numbers on top of maps. If this is truly the graphical extent of the data being displayed, then structured video is neither the most efficient, nor the most effective tool for achieving visualization.

Structured video is also effective when the resulting visualization requires interaction, or the ability to make static video information into a dynamic presentation. There are many examples of how interaction makes the video sequence produced by the weather

information system dynamic. The most basic examples are the translation and scale knobs. The only example of a how a video sequence can be made dynamic by adding dynamic objects is the display of wind speed. The animated leaf that moves faster or slower in relation to the wind speed not only makes the video sequence dynamic, but also transforms a static piece of information into a dynamic presentation.

In terms of the specific structured video decoder used to implement this thesis, structured video is most effective when the data being displayed can be represented by 2-D or 2 1/2-D graphics objects and when the user needs the ability to interact with and modify the display in real-time. Although Cheops does have the ability to render full 3-D graphics objects in real-time, these objects slow the frame rate of the resulting display thus decreasing the impact of real-time interaction. The slower the frame rate, the longer the system takes to respond to user interaction. However, despite the negative effects on interaction, using Cheops as a structured video decoder to perform full 3-D data visualization is an interesting concept that should be pursued (see Section 5.3).

The structured video scripting language used to implement this system (Isis), was, for the most part, very simple and effective. However, the current implementation of the weather system does not use all of the functionality of such a powerful scripting language. For example, the system does not use the scripting language to define a continuous dynamic video sequence although this would be an interesting addition (see Section 5.2). The most valuable contribution of the scripting language to this thesis was that it allows for real-time user interaction which is essential to any information system.

Only two problems were encountered with Isis. The first was its method of handling text. Isis requires each character to be represented as an actor which is inconvenient and expensive in terms of memory required to execute the scripts. If this were improved such

that entire strings could be implemented with one actor, the number of actors required by the system could be reduced by at least one order of magnitude. The second was that the size of the Isis script generated by the weather system is rather unwieldily. Over half of the lines of Isis code are used to set values for each datum, for each city, for each time, in Isis timeline structures. Because of this, the size of the script grows exponentially and quickly becomes unmanageable. However, while the files are large, they only require kilobytes of storage space as opposed to the megabytes of storages space that would be required to store the video sequence created by the script if it were represented in a more traditional frame based format.

The method of interaction provided by the structured video decoder used in this thesis is most effective for series data. The knob interface is well suited for the time series nature of weather forecasts because scrolling through information is a natural way to interact with series data. If, however, the information system being implemented requires the ability to make selections or traverse a tree structure, as in a hypertext like visualization system where data are linked and the user needs the ability to traverse a path of links, the knob interface fails miserably.

Finally, when the weather system is viewed as simply a weather system, and not as a structured video application, it is only slightly better than average. The output is more understandable and entertaining to the average person than the amalgamation of complicated meteorological symbols created by most other weather systems. In addition, the system allows users to customize the display of information more than other weather systems. However, because of the current limitations of the weather system in its present state, it does not compare well to systems that can provide both more data and more supplementary information (like satellite photos, radar maps, etc.) in the same amount of time.

5.2 If I Had More Time.....

The list of things I would add to this weather system if I had more time seems innumerable. However, it is important to note that while there is room for improvement, the system did accomplish its goal of showing that structured video could be used as a form of data visualization.

The first feature I would add is a larger variety of knobs. For example, there are already knobs that move through time and data set; a knob that moves through the cities and shows all of the information for one city at a time would be interesting. Other ideas for knob functions include a knob that controls the granularity of data being displayed and a knob that allows the user to view video clips of televised weather forecasts. More knobs would enhance the user's ability to customize the weather system.

In addition to adding knobs, the system should be enhanced to offer continuous scripted dynamic sequences, rather than having motion imposed by user interaction. One interesting example would be to create a script would loop through the time while allowing the user to control the other aspects of the display. This would further emphasize the usefulness of the scripting language component of a structured video decoder for visualizing data.

Second, I would add live video clips to the system. This step was partially implemented. The graphics and audio objects for three television news weather forecasts were created. Adding these to the system would require defining a knob to control how and when these video clips are displayed.

Third, a very interesting suggestion was provided by Glorianna Davenport that involves using an audio weather forecast and using the weather maps generated by the weather system to supplement what the weather forecaster is saying. This would show how the

structured weather system could augment traditional weather broadcasts.

The system also needs satellite and radar images. A lot of effort was put into trying to implement these graphics. However, none of the radar or satellite images available over the Web was able to be adapted for use in the system. The problem was separating the radar or satellite information from the background of the image so that they could be composited on top of the maps in the weather system.

One simple but useful improvement to the system would be to set up the city and state databases from files, rather than hard coding them as they are done now. This would allow changes to be made and cities and states to be added without recompiling the system. In addition, "weather" could use the same "cities" file so that any changes made to the system would automatically be reflected in the cities request box.

Of course, implementing all 50 states is on this list of improvements, but it is much more interesting to show different features for a few states than to show the same features for a lot of states, so this is not a high priority.

5.3 Future Work

Section 5.2 discusses how I would extend the current system. However, this is not where the future of this work lies. This thesis has shown that it is possible to use structured video as a form of data visualization. However, instead of focusing on this one application, further research should consider how structured video could be used for other forms of data visualization. Two main factors point to this conclusion.

First, in the process of looking for background information on data visualization I was consistently confronted with example of scientific visualization or systems that deal with very difficult and complicated natural phenomenon. Such a system was not chosen for

this thesis because implementing it would require knowledge about what was being modeled. However, scientific visualization is the focal point of a substantial amount of research in the data visualization field and it would be interesting to see if structured video could be useful as a form of data visualization in that context.

Second, when Cheops is better able to handle full 3-D rendering, it could be useful for information systems that require real time access to 3-D data. In fact, this is the most likely way in which structured video could outperform other visualization systems. Many people asked why the current system did not use fancy three dimensional weather graphics. The main reason is based on Tufte's argument that information should be presented visually in the same number of dimensions that it requires. For example, using a three dimensional graphic to display two dimensional data (like a time series plot) only serves to unnecessarily embellish the data, and often either misconstrue or detract from it. However, the fact remains that real-time rendering is one ability currently unique to Cheops and a few other select (and expensive) platforms, so implementing a system that uses three dimensional data, would show off the benefits of both structured video and Cheops as tools for data visualization.

Appendix A

Isis Files

This section includes examples of the isis script files generated by the weather system. These files were created by executing the "makeweather" program with the request file shown in Section B.1. These files are included for reference purposes and do not have detailed accompanying explanations.

A.1 The Request File

```
mycities
-256.0 -256.0 256.0 256.0
DEFAULT
BOSTON, MA
AUGUSTA, ME
```

A.2 The Main Isis File

```
#Scripts/mycities/mycities.isis
(load "Scripts/Standard/strvid.isis")
(load "Scripts/Standard/strtext.isis")
(load "Scripts/Standard/weatherfunc.isis")

#_____________________________________________________________
# Iniialize

# Set these higher for debugging purposes
(set-video-verbose 0)
(set-interface-verbose 0)

(initialize-interface)
(initialize-video-system)

#_____________________________________________________________
# Set the scale information
# This is used to map longitude and lattitude to
# screen coordinates.

(set windowsize (RealRect -256.0 -256.0 256.0 256.0))
```

```
# Set the number of x and y pixels of the map
(set pplong 104.469)
(set pplatt 131.515)

# Set the geographical center of the map in pixels
(set centlong 70.235)
(set centlatt 44.340)

(load "Scripts/Standard/weatherdef.isis")

(vid-update window win-location (Loc 588 512))

#_____________________________________________________________
# Register graphics objects with the system

(load "Scripts/mycities/mycitiesmaps.isis")
(load "Scripts/Standard/graphics.isis")
(load "Scripts/mycities/mycitiesdata.isis")

#_____________________________________________________________
# Text Actors

(load "Scripts/Standard/corner.isis")
#_____________________________________________________________
# The other definitions are in weatherdefs.isis

(vid-update videng
     vid-actors (append (AddrList center-txt left-corner-txt right-corner-txt

MAs1 MAr1 MAc1 MAh1 MAt1 MEs1 MEr1 MEc1 MEh1 MEt1 MEs2 MEr2 MEc2 MEh2 MEt2 )
(AddrList BOS-clds AUG-clds )
BOS-temp-str AUG-temp-str
BOS-max-str AUG-max-str
BOS-min-str AUG-min-str
BOS-dew-str AUG-dew-str
(AddrList BOS-wspd AUG-wspd )
(AddrList BOS-wdir AUG-wdir )
BOS-gusts-str AUG-gusts-str
BOS-vis-str AUG-vis-str
BOS-press-str AUG-press-str
BOS-humidity-str AUG-humidity-str
BOS-altimeter-str AUG-altimeter-str
BOS-sunrise-str AUG-sunrise-str
BOS-sunset-str AUG-sunset-str ))

#_____________________________________________________________
# Create the user interface

(set knobs (create-interface "Knobs"))

# X position knob
(load "Scripts/Standard/Knobs/x_pos_knob.isis")

# Y position knob
```

```
(load "Scripts/Standard/Knobs/y_pos_knob.isis")

# Scale knob
(load "Scripts/Standard/Knobs/scale_knob.isis")

# Map selection knob
(load "Scripts/Standard/Knobs/map_sel_knob.isis")

# Data knob
(load "Scripts/Standard/Knobs/data_knob.isis")

# Time knob
(set start-time 19)
(set end-time 72)
(load "Scripts/Standard/Knobs/time_knob.isis")
# City knob
(load "Scripts/Standard/Knobs/city_knob.isis")

# Exit knob
(load "Scripts/Standard/Knobs/exit_knob.isis")

#_____________________________________________________________
# Start the movie loop

(print newline
"Knob one controls the horizontal position"
       newline)
(print "Knob two controls the vertical position"
       newline)
(print "Knob three controls the zoom"
       newline)
(print "Knob four controls the maps being displayed"
       newline)
(print "Knob five controls the data being displayed"
       newline)
(print "Knob six controls the time being displayed"
       newline)
(print "Knob eight exits"
       newline
       newline)

(start-video-system)
(start-interface knobs)

(set leaf-frame 0)
(set start-day 4)

(set makeframe
     (proc ()
          (begin
             (update-interface knobs)
             (set map-sel (/ (check map-sel-input) 20))
             (vid-update MAs1 ac-visibility (svis-tl map-sel))
             (vid-update MAc1 ac-visibility (cvis-tl map-sel))
```

```
(vid-update MAr1 ac-visibility (rvis-tl map-sel))
(vid-update MAh1 ac-visibility (hvis-tl map-sel))
(vid-update MAt1 ac-visibility (tvis-tl map-sel))
(vid-update MEs1 ac-visibility (svis-tl map-sel))
(vid-update MEc1 ac-visibility (cvis-tl map-sel))
(vid-update MEr1 ac-visibility (rvis-tl map-sel))
(vid-update MEh1 ac-visibility (hvis-tl map-sel))
(vid-update MEt1 ac-visibility (tvis-tl map-sel))
(vid-update MEs2 ac-visibility (svis-tl map-sel))
(vid-update MEc2 ac-visibility (cvis-tl map-sel))
(vid-update MEr2 ac-visibility (rvis-tl map-sel))
(vid-update MEh2 ac-visibility (hvis-tl map-sel))
(vid-update MEt2 ac-visibility (tvis-tl map-sel))

(set thescale (valid-scales (/ (check scale-input) 20.0)))
(vid-update camera
     cam-ref-point
     (Pos (* (check xpos-input) -1) (check ypos-input) 0.0)
     cam-viewport
     (RealRect (/ -256.0 thescale) (/ -256.0 thescale)
               (/ 256.0 thescale) (/ 256.0 thescale)))
(set data (/ (check data-input) 20))
(set time (/ (check time-input) 20))

(vid-update left-corner-txt ac-object (lcorner-tl data))
(if (< time start-time)
     (vid-update right-corner-txt
           ac-frame 0
           ac-object yesterday)
     (if (= time start-time)
         (vid-update right-corner-txt
              ac-frame 0
              ac-object current)
         (vid-update right-corner-txt
               ac-frame (% time 24)
               ac-object times)))
(if (<= time start-time)
    (vid-update center-txt
           ac-visibility 0.0)
    (if (< time 24)
        (vid-update center-txt
                ac-visibility 1.0
                ac-frame 0
                ac-object today)
        (if (< time 48)
            (vid-update center-txt
                    ac-visibility 1.0
                    ac-frame 0
                    ac-object tomorrow)
            (vid-update center-txt
                    ac-visibility 1.0
                    ac-frame (+ start-day (floor (/ time 24)))
                    ac-object days))))
```

```
            (set data-elems (data-tl data))
            (set data-strings (str-tl data))

            (let ((elems (length data-elems)) (count 0))
               (while (< count elems)
                      (begin
                         (let ((city-str (data-strings count))
                               (city-data (data-elems count)))
                            (let ((cd (city-data time)))
                                (if (= data 0)
                                      (begin
                                         (vid-update city-str
                                                ac-visibility (cd 1)
                                                ac-object (cd 0))
                                         (vid-update left-corner-txt
                                                 ac-visibility 0.0))
                                   (if (= data 5)
                                           (vid-update city-str
                                              ac-visibility (cd 1)
                                              ac-frame leaf-frame
                                              ac-object (windspd-tl
                                                                (cd 0)))
                                      (if (= data 6)
                                          (vid-update city-str
                                             ac-visibility (cd 1)
                                             ac-frame (/ (cd 0) 10))
                                          (begin
                                             (setstring city-str (cd 0))
                                           (stringvisible city-str (cd 1)))))))))

                        (set count (+ count 1)))))

          (realize-video videng)
          (set leaf-frame (+ leaf-frame 1))
          (if (> leaf-frame 11) (set leaf-frame 0) Null)

          (let ((elems (length data-elems)) (count 0))
            (while (< count elems)
                   (begin
                     (let ((city-str (data-strings count)))
                       (if (or (= data 0)(= data 5) (= data 6))
                           (begin
                             (vid-update city-str ac-visibility 0.0)
                             (vid-update left-corner-txt ac-visibility 1.0))
                           (stringvisible city-str 0.0)))
                     (set count (+ count 1)))))

          )))

(while (= (check exit-input) 0) (makeframe))

(interactive)
```

A.3 The Data Isis File

```
#Scripts/mycities/mycitiesdata.isis

(load "Scripts/mycities/BOS.isis")
(load "Scripts/mycities/AUG.isis")
# Set up datalines of pieces of data for each city
(set data-tl (new-timeline))

(key data-tl 0 (TimeLineList BOS-clds-tl AUG-clds-tl ))
(key data-tl 1 (TimeLineList BOS-temp-tl AUG-temp-tl ))
(key data-tl 2 (TimeLineList BOS-max-tl AUG-max-tl ))
(key data-tl 3 (TimeLineList BOS-min-tl AUG-min-tl ))
(key data-tl 4 (TimeLineList BOS-dew-tl AUG-dew-tl ))
(key data-tl 5 (TimeLineList BOS-wspd-tl AUG-wspd-tl ))
(key data-tl 6 (TimeLineList BOS-wdir-tl AUG-wdir-tl ))
(key data-tl 7 (TimeLineList BOS-gusts-tl AUG-gusts-tl ))
(key data-tl 8 (TimeLineList BOS-vis-tl AUG-vis-tl ))
(key data-tl 9 (TimeLineList BOS-press-tl AUG-press-tl ))
(key data-tl 10 (TimeLineList BOS-humidity-tl AUG-humidity-tl ))
(key data-tl 11 (TimeLineList BOS-altimeter-tl AUG-altimeter-tl ))
(key data-tl 12 (TimeLineList BOS-norm-max-tl AUG-norm-max-tl ))
(key data-tl 13 (TimeLineList BOS-norm-min-tl AUG-norm-min-tl ))
(key data-tl 14 (TimeLineList BOS-rec-max-tl AUG-rec-max-tl ))
(key data-tl 15 (TimeLineList BOS-rec-maxy-tl AUG-rec-maxy-tl ))
(key data-tl 16 (TimeLineList BOS-rec-min-tl AUG-rec-min-tl ))
(key data-tl 17 (TimeLineList BOS-rec-miny-tl AUG-rec-miny-tl ))
(key data-tl 18 (TimeLineList BOS-sunrise-tl AUG-sunrise-tl ))
(key data-tl 19 (TimeLineList BOS-sunset-tl AUG-sunset-tl ))
(key data-tl 20 (TimeLineList BOS-temp-dep-tl AUG-temp-dep-tl ))
(key data-tl 21 (TimeLineList BOS-heating-tl AUG-heating-tl ))
(key data-tl 22 (TimeLineList BOS-heatingm-tl AUG-heatingm-tl ))
(key data-tl 23 (TimeLineList BOS-heatings-tl AUG-heatings-tl ))
(key data-tl 24 (TimeLineList BOS-heatingd-tl AUG-heatingd-tl ))
(key data-tl 25 (TimeLineList BOS-cooling-tl AUG-cooling-tl ))
(key data-tl 26 (TimeLineList BOS-coolingm-tl AUG-coolingm-tl ))
(key data-tl 27 (TimeLineList BOS-coolings-tl AUG-coolings-tl ))
(key data-tl 28 (TimeLineList BOS-coolingd-tl AUG-coolingd-tl ))
(key data-tl 29 (TimeLineList BOS-precip-tl AUG-precip-tl ))
(key data-tl 30 (TimeLineList BOS-precipm-tl AUG-precipm-tl ))
(key data-tl 31 (TimeLineList BOS-precipm-norm-tl AUG-precipm-norm-tl ))
(key data-tl 32 (TimeLineList BOS-precipy-tl AUG-precipy-tl ))
(key data-tl 33 (TimeLineList BOS-precipy-norm-tl AUG-precipy-norm-tl ))
(key data-tl 34 (TimeLineList BOS-snowfall-tl AUG-snowfall-tl ))
(key data-tl 35 (TimeLineList BOS-snowfallm-tl AUG-snowfallm-tl ))
(key data-tl 36 (TimeLineList BOS-snowfalls-tl AUG-snowfalls-tl ))
(key data-tl 37 (TimeLineList BOS-fastwind-tl AUG-fastwind-tl ))
(key data-tl 38 (TimeLineList BOS-peak-tl AUG-peak-tl ))

(set str-tl (new-timeline))
(key str-tl 0 (AddrList BOS-clds AUG-clds ))
(key str-tl 1 (StringList BOS-temp-str AUG-temp-str ))
(key str-tl 2 (StringList BOS-max-str AUG-max-str ))
```

```
(key str-tl 3 (StringList BOS-min-str AUG-min-str ))
(key str-tl 4 (StringList BOS-dew-str AUG-dew-str ))
(key str-tl 5 (AddrList BOS-wspd AUG-wspd ))
(key str-tl 6 (AddrList BOS-wdir AUG-wdir ))
(key str-tl 7 (StringList BOS-gusts-str AUG-gusts-str ))
(key str-tl 8 (StringList BOS-vis-str AUG-vis-str ))
(key str-tl 9 (StringList BOS-press-str AUG-press-str ))
(key str-tl 10 (StringList BOS-humidity-str AUG-humidity-str ))
(key str-tl 11 (StringList BOS-altimeter-str AUG-altimeter-str ))
(key str-tl 12 (StringList BOS-norm-max-str AUG-norm-max-str ))
(key str-tl 13 (StringList BOS-norm-min-str AUG-norm-min-str ))
(key str-tl 14 (StringList BOS-rec-max-str AUG-rec-max-str ))
(key str-tl 15 (StringList BOS-rec-maxy-str AUG-rec-maxy-str ))
(key str-tl 16 (StringList BOS-rec-min-str AUG-rec-min-str ))
(key str-tl 17 (StringList BOS-rec-miny-str AUG-rec-miny-str ))
(key str-tl 18 (StringList BOS-sunrise-str AUG-sunrise-str ))
(key str-tl 19 (StringList BOS-sunset-str AUG-sunset-str ))
(key str-tl 20 (StringList BOS-temp-dep-str AUG-temp-dep-str ))
(key str-tl 21 (StringList BOS-heating-str AUG-heating-str ))
(key str-tl 22 (StringList BOS-heatingm-str AUG-heatingm-str ))
(key str-tl 23 (StringList BOS-heatings-str AUG-heatings-str ))
(key str-tl 24 (StringList BOS-heatingd-str AUG-heatingd-str ))
(key str-tl 25 (StringList BOS-cooling-str AUG-cooling-str ))
(key str-tl 26 (StringList BOS-coolingm-str AUG-coolingm-str ))
(key str-tl 27 (StringList BOS-coolings-str AUG-coolings-str ))
(key str-tl 28 (StringList BOS-coolingd-str AUG-coolingd-str ))
(key str-tl 29 (StringList BOS-precip-str AUG-precip-str ))
(key str-tl 30 (StringList BOS-precipm-str AUG-precipm-str ))
(key str-tl 31 (StringList BOS-precipm-norm-str AUG-precipm-norm-str ))
(key str-tl 32 (StringList BOS-precipy-str AUG-precipy-str ))
(key str-tl 33 (StringList BOS-precipy-norm-str AUG-precipy-norm-str ))
(key str-tl 34 (StringList BOS-snowfall-str AUG-snowfall-str ))
(key str-tl 35 (StringList BOS-snowfallm-str AUG-snowfallm-str ))
(key str-tl 36 (StringList BOS-snowfalls-str AUG-snowfalls-str ))
(key str-tl 37 (StringList BOS-fastwind-str AUG-fastwind-str ))
(key str-tl 38 (StringList BOS-peak-str AUG-peak-str ))
```

A.4 The Map Isis File

```
#Scripts/mycities/mycitiesmaps.isis

# Register Maps

(set MA-state1
     (new-video-object "Objects/Maps/MA/state1.dat"
     "m1-MA-state1" "non-cacheable"))
(set MA-rivers1
     (new-video-object "Objects/Maps/MA/rivers1.dat"
     "m1-MA-rivers1" "non-cacheable"))
(set MA-counties1
     (new-video-object "Objects/Maps/MA/counties1.dat"
     "m1-MA-counties1" "non-cacheable"))
(set MA-highways1
```

```
     (new-video-object "Objects/Maps/MA/highways1.dat"
     "m1-MA-highways1" "non-cacheable"))
(set MA-cities1
     (new-video-object "Objects/Maps/MA/cities1.dat"
     "m1-MA-cities1" "non-cacheable"))

# First set up actors for the maps
(set MAs1 (new-actor))
(set MAc1 (new-actor))
(set MAr1 (new-actor))
(set MAh1 (new-actor))
(set MAt1 (new-actor))

(vid-update MAs1
     ac-do-zbuffer True
     ac-position-in-3d True
     ac-visibility 0.0
     ac-position (+ (conv (LongLatt 71.71 42.06)) (Pos 0.0 0.0 -1.5))
     ac-object MA-state1)
(vid-update MAc1
     ac-do-zbuffer True
     ac-position-in-3d True
     ac-visibility 0.0
     ac-position (+ (conv (LongLatt 71.71 42.06)) (Pos 0.0 0.0 -1.5))
     ac-object MA-counties1)
(vid-update MAr1
     ac-do-zbuffer True
     ac-position-in-3d True
     ac-visibility 0.0
     ac-position (+ (conv (LongLatt 71.71 42.06)) (Pos 0.0 0.0 -1.0))
     ac-object MA-rivers1)
(vid-update MAh1
     ac-do-zbuffer True
     ac-position-in-3d True
     ac-visibility 0.0
     ac-position (+ (conv (LongLatt 71.71 42.06)) (Pos 0.0 0.0 -1.0))
     ac-object MA-highways1)
(vid-update MAt1
     ac-do-zbuffer True
     ac-position-in-3d True
     ac-visibility 0.0
     ac-position (+ (conv (LongLatt 71.71 42.06)) (Pos 0.0 0.0 -1.0))
     ac-object MA-cities1)

(set ME-state1
     (new-video-object "Objects/Maps/ME/state1.dat"
     "m1-ME-state1" "non-cacheable"))
(set ME-state2
     (new-video-object "Objects/Maps/ME/state2.dat"
     "m1-ME-state2" "non-cacheable"))
(set ME-rivers1
     (new-video-object "Objects/Maps/ME/rivers1.dat"
     "m1-ME-rivers1" "non-cacheable"))
(set ME-rivers2
```

```
     (new-video-object "Objects/Maps/ME/rivers2.dat"
     "m1-ME-rivers2" "non-cacheable"))
(set ME-counties1
     (new-video-object "Objects/Maps/ME/counties1.dat"
     "m1-ME-counties1" "non-cacheable"))
(set ME-counties2
     (new-video-object "Objects/Maps/ME/counties2.dat"
     "m1-ME-counties2" "non-cacheable"))
(set ME-highways1
     (new-video-object "Objects/Maps/ME/highways1.dat"
     "m1-ME-highways1" "non-cacheable"))
(set ME-highways2
     (new-video-object "Objects/Maps/ME/highways2.dat"
     "m1-ME-highways2" "non-cacheable"))
(set ME-cities1
     (new-video-object "Objects/Maps/ME/cities1.dat"
     "m1-ME-cities1" "non-cacheable"))
(set ME-cities2
     (new-video-object "Objects/Maps/ME/cities2.dat"
     "m1-ME-cities2" "non-cacheable"))

# First set up actors for the maps
(set MEs1 (new-actor))
(set MEc1 (new-actor))
(set MEr1 (new-actor))
(set MEh1 (new-actor))
(set MEt1 (new-actor))
(set MEs2 (new-actor))
(set MEc2 (new-actor))
(set MEr2 (new-actor))
(set MEh2 (new-actor))
(set MEt2 (new-actor))

(vid-update MEs1
     ac-do-zbuffer True
     ac-position-in-3d True
     ac-visibility 0.0
     ac-position (+ (conv (LongLatt 69.03 45.51)) (Pos 0.0 0.0 -1.5))
     ac-object ME-state1)
(vid-update MEs2
     ac-do-zbuffer True
     ac-position-in-3d True
     ac-visibility 0.0
     ac-position (+ (conv (LongLatt 70.59 43.33)) (Pos 0.0 0.0 -1.5))
     ac-object ME-state2)
(vid-update MEc1
     ac-do-zbuffer True
     ac-position-in-3d True
     ac-visibility 0.0
     ac-position (+ (conv (LongLatt 69.03 45.51)) (Pos 0.0 0.0 -1.5))
     ac-object ME-counties1)
(vid-update MEc2
     ac-do-zbuffer True
     ac-position-in-3d True
```

```
     ac-visibility 0.0
     ac-position (+ (conv (LongLatt 70.59 43.33)) (Pos 0.0 0.0 -1.5))
     ac-object ME-counties2)
(vid-update MEr1
     ac-do-zbuffer True
     ac-position-in-3d True
     ac-visibility 0.0
     ac-position (+ (conv (LongLatt 69.03 45.51)) (Pos 0.0 0.0 -1.0))
     ac-object ME-rivers1)
(vid-update MEr2
     ac-do-zbuffer True
     ac-position-in-3d True
     ac-visibility 0.0
     ac-position (+ (conv (LongLatt 70.59 43.33)) (Pos 0.0 0.0 -1.0))
     ac-object ME-rivers2)
(vid-update MEh1
     ac-do-zbuffer True
     ac-position-in-3d True
     ac-visibility 0.0
     ac-position (+ (conv (LongLatt 69.03 45.51)) (Pos 0.0 0.0 -1.0))
     ac-object ME-highways1)
(vid-update MEh2
     ac-do-zbuffer True
     ac-position-in-3d True
     ac-visibility 0.0
     ac-position (+ (conv (LongLatt 70.59 43.33)) (Pos 0.0 0.0 -1.0))
     ac-object ME-highways2)
(vid-update MEt1
     ac-do-zbuffer True
     ac-position-in-3d True
     ac-visibility 0.0
     ac-position (+ (conv (LongLatt 69.03 45.51)) (Pos 0.0 0.0 -1.0))
     ac-object ME-cities1)
(vid-update MEt2
     ac-do-zbuffer True
     ac-position-in-3d True
     ac-visibility 0.0
     ac-position (+ (conv (LongLatt 70.59 43.33)) (Pos 0.0 0.0 -1.0))
     ac-object ME-cities2)
```

A.5 A City Isis File

```
#Scripts/mycities/BOS.isis

# Set position
(set BOS (LongLatt 71.03 42.37))

# Set up objects
(set BOS-clds (new-actor))
(set BOS-wspd (new-actor))
(set BOS-wdir (new-actor))

(set BOS-temp-str (newstring 2 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
```

```
(set BOS-max-str (newstring 2 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-min-str (newstring 2 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-dew-str (newstring 2 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-gusts-str (newstring 2 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-vis-str (newstring 4 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-press-str (newstring 7 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-humidity-str (newstring 2 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-altimeter-str (newstring 5 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))

# Set up today objects
(set BOS-norm-min-str (newstring 2 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-norm-max-str (newstring 2 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-rec-min-str (newstring 2 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-rec-miny-str (newstring 4 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-rec-max-str (newstring 2 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-rec-maxy-str (newstring 4 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-sunrise-str (newstring 4 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-sunset-str (newstring 4 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))

# Set up yesterday objects
(set BOS-temp-dep-str (newstring 3 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-heating-str (newstring 2 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-heatingm-str (newstring 3 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-heatings-str (newstring 3 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-heatingd-str (newstring 4 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-cooling-str (newstring 2 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-coolingm-str (newstring 3 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-coolings-str (newstring 3 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-coolingd-str (newstring 4 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-precip-str (newstring 3 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-precipm-str (newstring 4 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-precipm-norm-str (newstring 4 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-precipy-str (newstring 5 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-precipy-norm-str (newstring 5 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-snowfall-str (newstring 3 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-snowfallm-str (newstring 5 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-snowfalls-str (newstring 5 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-fastwind-str (newstring 2 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))
(set BOS-peak-str (newstring 2 cs24 (conv BOS) (Pos 10.0 0.0 0.0)))

(vid-update BOS-clds
        ac-do-zbuffer True
        ac-position-in-3d True
        ac-visibility 0.0
        ac-position (conv BOS)
        ac-object sunny)
(vid-update BOS-wspd
        ac-do-zbuffer True
        ac-position-in-3d True
        ac-visibility 0.0
        ac-position (conv BOS)
        ac-object calm)
(vid-update BOS-wdir
        ac-do-zbuffer True
```

```
        ac-position-in-3d True
        ac-visibility 0.0
        ac-position (conv BOS)
        ac-frame 0
        ac-object degrees)

# Set up a timeline for each object
(set BOS-clds-tl (new-timeline (ValVis sunny 0.0)))
(set BOS-temp-tl (new-timeline (ValVis 0 0.0)))
(set BOS-max-tl (new-timeline (ValVis 0 0.0)))
(set BOS-min-tl (new-timeline (ValVis 0 0.0)))
(set BOS-dew-tl (new-timeline (ValVis 0 0.0)))
(set BOS-wspd-tl (new-timeline (ValVis 0 0.0)))
(set BOS-wdir-tl (new-timeline (ValVis 0 0.0)))
(set BOS-gusts-tl (new-timeline (ValVis 0 0.0)))
(set BOS-vis-tl (new-timeline (ValVis 0 0.0)))
(set BOS-press-tl (new-timeline (ValVis 0 0.0)))
(set BOS-humidity-tl (new-timeline (ValVis 0 0.0)))
(set BOS-altimeter-tl (new-timeline (ValVis 0 0.0)))
(set BOS-norm-max-tl (new-timeline (ValVis 0 0.0)))
(set BOS-norm-min-tl (new-timeline (ValVis 0 0.0)))
(set BOS-rec-max-tl (new-timeline (ValVis 0 0.0)))
(set BOS-rec-min-tl (new-timeline (ValVis 0 0.0)))
(set BOS-rec-maxy-tl (new-timeline (ValVis 0 0.0)))
(set BOS-rec-miny-tl (new-timeline (ValVis 0 0.0)))
(set BOS-sunrise-tl (new-timeline (ValVis 0 0.0)))
(set BOS-sunset-tl (new-timeline (ValVis 0 0.0)))
(set BOS-temp-dep-tl (new-timeline (ValVis 0 0.0)))
(set BOS-heating-tl (new-timeline (ValVis 0 0.0)))
(set BOS-heatingm-tl (new-timeline (ValVis 0 0.0)))
(set BOS-heatings-tl (new-timeline (ValVis 0 0.0)))
(set BOS-heatingd-tl (new-timeline (ValVis 0 0.0)))
(set BOS-cooling-tl (new-timeline (ValVis 0 0.0)))
(set BOS-coolingm-tl (new-timeline (ValVis 0 0.0)))
(set BOS-coolings-tl (new-timeline (ValVis 0 0.0)))
(set BOS-coolingd-tl (new-timeline (ValVis 0 0.0)))
(set BOS-precip-tl (new-timeline (ValVis 0 0.0)))
(set BOS-precipm-tl (new-timeline (ValVis 0 0.0)))
(set BOS-precipm-norm-tl (new-timeline (ValVis 0 0.0)))
(set BOS-precipy-tl (new-timeline (ValVis 0 0.0)))
(set BOS-precipy-norm-tl (new-timeline (ValVis 0 0.0)))
(set BOS-snowfall-tl (new-timeline (ValVis 0 0.0)))
(set BOS-snowfallm-tl (new-timeline (ValVis 0 0.0)))
(set BOS-snowfalls-tl (new-timeline (ValVis 0 0.0)))
(set BOS-fastwind-tl (new-timeline (ValVis 0 0.0)))
(set BOS-peak-tl (new-timeline (ValVis 0 0.0)))
(key BOS-temp-tl 17 (ValVis "36" 1.0))
(key BOS-max-tl 17 (ValVis "40" 1.0))
(key BOS-min-tl 17 (ValVis "31" 1.0))
(key BOS-temp-dep-tl 17 (ValVis "-11" 1.0))
(key BOS-heating-tl 17 (ValVis "29" 1.0))
(key BOS-heatingm-tl 17 (ValVis "269" 1.0))
(key BOS-heatings-tl 17 (ValVis "599" 1.0))
(key BOS-heatingd-tl 17 (ValVis "-15" 1.0))
```

```
(key BOS-cooling-tl 17 (ValVis "0" 1.0))
(key BOS-coolingm-tl 17 (ValVis "0" 1.0))
(key BOS-coolings-tl 17 (ValVis "846" 1.0))
(key BOS-coolingd-tl 17 (ValVis "168" 1.0))
(key BOS-precip-tl 17 (ValVis "0.01" 1.0))
(key BOS-precipm-tl 17 (ValVis "2.88" 1.0))
(key BOS-precipm-norm-tl 17 (ValVis "1.74" 1.0))
(key BOS-precipy-tl 17 (ValVis "29.65" 1.0))
(key BOS-precipy-norm-tl 17 (ValVis "34.94" 1.0))
(key BOS-snowfall-tl 17 (ValVis "0.20" 1.0))
(key BOS-snowfallm-tl 17 (ValVis "0.20" 1.0))
(key BOS-snowfalls-tl 17 (ValVis "0.20" 1.0))
(key BOS-fastwind-tl 17 (ValVis "15" 1.0))
(key BOS-peak-tl 17 (ValVis "20" 1.0))
(key BOS-wdir-tl 17 (ValVis 180 1.0))
(key BOS-norm-min-tl 19 (ValVis "39" 1.0))
(key BOS-norm-max-tl 19 (ValVis "53" 1.0))
(key BOS-rec-min-tl 19 (ValVis "16" 1.0))
(key BOS-rec-max-tl 19 (ValVis "71" 1.0))
(key BOS-rec-miny-tl 19 (ValVis "1905" 1.0))
(key BOS-rec-maxy-tl 19 (ValVis "1993" 1.0))

(key BOS-clds-tl 19  (ValVis rain 1.0))
(key BOS-temp-tl 19 (ValVis "47" 1.0))
(key BOS-max-tl 19 (ValVis "-100" 0.0))
(key BOS-min-tl 19 (ValVis "-100" 0.0))
(key BOS-dew-tl 19 (ValVis "43" 1.0))
(key BOS-wdir-tl 19 (ValVis 70 1.0))
(key BOS-wspd-tl 19 (ValVis 37 1.0))
(key BOS-gusts-tl 19 (ValVis "48" 1.0))
(key BOS-vis-tl 19 (ValVis "2.8" 1.0))
(key BOS-press-tl 19 (ValVis "1012.00" 1.0))
(key BOS-humidity-tl 19 (ValVis "86" 1.0))
(key BOS-altimeter-tl 19 (ValVis "29.89" 1.0))
(key BOS-sunrise-tl 19 (ValVis "633" 1.0))
(key BOS-sunset-tl 19 (ValVis "424" 1.0))

(key BOS-clds-tl 12  (ValVis overcast 1.0))
(key BOS-temp-tl 12 (ValVis "36" 1.0))
(key BOS-max-tl 12 (ValVis "-100" 0.0))
(key BOS-min-tl 12 (ValVis "-100" 0.0))
(key BOS-dew-tl 12 (ValVis "31" 1.0))
(key BOS-wdir-tl 12 (ValVis 60 1.0))
(key BOS-wspd-tl 12 (ValVis 10 1.0))
(key BOS-gusts-tl 12 (ValVis "-100" 0.0))
(key BOS-vis-tl 12 (ValVis "-100.0" 0.0))
(key BOS-press-tl 12 (ValVis "-100.00" 0.0))
(key BOS-humidity-tl 12 (ValVis "-100" 0.0))
(key BOS-altimeter-tl 12 (ValVis "-100.00" 0.0))
(key BOS-sunrise-tl 12 (ValVis "-100" 0.0))
(key BOS-sunset-tl 12 (ValVis "-100" 0.0))

(key BOS-clds-tl 15  (ValVis overcast 1.0))
(key BOS-temp-tl 15 (ValVis "43" 1.0))
```

```
(key BOS-max-tl 15 (ValVis "-100" 0.0))
(key BOS-min-tl 15 (ValVis "-100" 0.0))
(key BOS-dew-tl 15 (ValVis "32" 1.0))
(key BOS-wdir-tl 15 (ValVis 50 1.0))
(key BOS-wspd-tl 15 (ValVis 13 1.0))
(key BOS-gusts-tl 15 (ValVis "-100" 0.0))
(key BOS-vis-tl 15 (ValVis "-100.0" 0.0))
(key BOS-press-tl 15 (ValVis "-100.00" 0.0))
(key BOS-humidity-tl 15 (ValVis "-100" 0.0))
(key BOS-altimeter-tl 15 (ValVis "-100.00" 0.0))
(key BOS-sunrise-tl 15 (ValVis "-100" 0.0))
(key BOS-sunset-tl 15 (ValVis "-100" 0.0))

(key BOS-clds-tl 18  (ValVis overcast 1.0))
(key BOS-temp-tl 18 (ValVis "46" 1.0))
(key BOS-max-tl 18 (ValVis "-100" 0.0))
(key BOS-min-tl 18 (ValVis "-100" 0.0))
(key BOS-dew-tl 18 (ValVis "36" 1.0))
(key BOS-wdir-tl 18 (ValVis 70 1.0))
(key BOS-wspd-tl 18 (ValVis 17 1.0))
(key BOS-gusts-tl 18 (ValVis "-100" 0.0))
(key BOS-vis-tl 18 (ValVis "-100.0" 0.0))
(key BOS-press-tl 18 (ValVis "-100.00" 0.0))
(key BOS-humidity-tl 18 (ValVis "-100" 0.0))
(key BOS-altimeter-tl 18 (ValVis "-100.00" 0.0))
(key BOS-sunrise-tl 18 (ValVis "-100" 0.0))
(key BOS-sunset-tl 18 (ValVis "-100" 0.0))

(key BOS-clds-tl 21  (ValVis overcast 1.0))
(key BOS-temp-tl 21 (ValVis "47" 1.0))
(key BOS-max-tl 21 (ValVis "-100" 0.0))
(key BOS-min-tl 21 (ValVis "-100" 0.0))
(key BOS-dew-tl 21 (ValVis "40" 1.0))
(key BOS-wdir-tl 21 (ValVis 60 1.0))
(key BOS-wspd-tl 21 (ValVis 21 1.0))
(key BOS-gusts-tl 21 (ValVis "-100" 0.0))
(key BOS-vis-tl 21 (ValVis "-100.0" 0.0))
(key BOS-press-tl 21 (ValVis "-100.00" 0.0))
(key BOS-humidity-tl 21 (ValVis "-100" 0.0))
(key BOS-altimeter-tl 21 (ValVis "-100.00" 0.0))
(key BOS-sunrise-tl 21 (ValVis "-100" 0.0))
(key BOS-sunset-tl 21 (ValVis "-100" 0.0))

(key BOS-clds-tl 24  (ValVis overcast 1.0))
(key BOS-temp-tl 24 (ValVis "47" 1.0))
(key BOS-max-tl 24 (ValVis "48" 1.0))
(key BOS-min-tl 24 (ValVis "43" 1.0))
(key BOS-dew-tl 24 (ValVis "44" 1.0))
(key BOS-wdir-tl 24 (ValVis 50 1.0))
(key BOS-wspd-tl 24 (ValVis 25 1.0))
(key BOS-gusts-tl 24 (ValVis "-100" 0.0))
(key BOS-vis-tl 24 (ValVis "-100.0" 0.0))
(key BOS-press-tl 24 (ValVis "-100.00" 0.0))
(key BOS-humidity-tl 24 (ValVis "-100" 0.0))
```

```
(key BOS-altimeter-tl 24 (ValVis "-100.00" 0.0))
(key BOS-sunrise-tl 24 (ValVis "634" 1.0))
(key BOS-sunset-tl 24 (ValVis "423" 1.0))

(key BOS-clds-tl 27  (ValVis overcast 1.0))
(key BOS-temp-tl 27 (ValVis "46" 1.0))
(key BOS-max-tl 27 (ValVis "-100" 0.0))
(key BOS-min-tl 27 (ValVis "-100" 0.0))
(key BOS-dew-tl 27 (ValVis "45" 1.0))
(key BOS-wdir-tl 27 (ValVis 40 1.0))
(key BOS-wspd-tl 27 (ValVis 23 1.0))
(key BOS-gusts-tl 27 (ValVis "-100" 0.0))
(key BOS-vis-tl 27 (ValVis "-100.0" 0.0))
(key BOS-press-tl 27 (ValVis "-100.00" 0.0))
(key BOS-humidity-tl 27 (ValVis "-100" 0.0))
(key BOS-altimeter-tl 27 (ValVis "-100.00" 0.0))
(key BOS-sunrise-tl 27 (ValVis "-100" 0.0))
(key BOS-sunset-tl 27 (ValVis "-100" 0.0))

(key BOS-clds-tl 30  (ValVis overcast 1.0))
(key BOS-temp-tl 30 (ValVis "45" 1.0))
(key BOS-max-tl 30 (ValVis "-100" 0.0))
(key BOS-min-tl 30 (ValVis "-100" 0.0))
(key BOS-dew-tl 30 (ValVis "45" 1.0))
(key BOS-wdir-tl 30 (ValVis 50 1.0))
(key BOS-wspd-tl 30 (ValVis 21 1.0))
(key BOS-gusts-tl 30 (ValVis "-100" 0.0))
(key BOS-vis-tl 30 (ValVis "-100.0" 0.0))
(key BOS-press-tl 30 (ValVis "-100.00" 0.0))
(key BOS-humidity-tl 30 (ValVis "-100" 0.0))
(key BOS-altimeter-tl 30 (ValVis "-100.00" 0.0))
(key BOS-sunrise-tl 30 (ValVis "-100" 0.0))
(key BOS-sunset-tl 30 (ValVis "-100" 0.0))

(key BOS-clds-tl 33  (ValVis overcast 1.0))
(key BOS-temp-tl 33 (ValVis "47" 1.0))
(key BOS-max-tl 33 (ValVis "-100" 0.0))
(key BOS-min-tl 33 (ValVis "-100" 0.0))
(key BOS-dew-tl 33 (ValVis "46" 1.0))
(key BOS-wdir-tl 33 (ValVis 70 1.0))
(key BOS-wspd-tl 33 (ValVis 17 1.0))
(key BOS-gusts-tl 33 (ValVis "-100" 0.0))
(key BOS-vis-tl 33 (ValVis "-100.0" 0.0))
(key BOS-press-tl 33 (ValVis "-100.00" 0.0))
(key BOS-humidity-tl 33 (ValVis "-100" 0.0))
(key BOS-altimeter-tl 33 (ValVis "-100.00" 0.0))
(key BOS-sunrise-tl 33 (ValVis "-100" 0.0))
(key BOS-sunset-tl 33 (ValVis "-100" 0.0))

(key BOS-clds-tl 36  (ValVis overcast 1.0))
(key BOS-temp-tl 36 (ValVis "46" 1.0))
(key BOS-max-tl 36 (ValVis "-100" 0.0))
(key BOS-min-tl 36 (ValVis "-100" 0.0))
(key BOS-dew-tl 36 (ValVis "45" 1.0))
```

```
(key BOS-wdir-tl 36 (ValVis 70 1.0))
(key BOS-wspd-tl 36 (ValVis 14 1.0))
(key BOS-gusts-tl 36 (ValVis "-100" 0.0))
(key BOS-vis-tl 36 (ValVis "-100.0" 0.0))
(key BOS-press-tl 36 (ValVis "-100.00" 0.0))
(key BOS-humidity-tl 36 (ValVis "-100" 0.0))
(key BOS-altimeter-tl 36 (ValVis "-100.00" 0.0))
(key BOS-sunrise-tl 36 (ValVis "-100" 0.0))
(key BOS-sunset-tl 36 (ValVis "-100" 0.0))

(key BOS-clds-tl 39  (ValVis overcast 1.0))
(key BOS-temp-tl 39 (ValVis "47" 1.0))
(key BOS-max-tl 39 (ValVis "-100" 0.0))
(key BOS-min-tl 39 (ValVis "-100" 0.0))
(key BOS-dew-tl 39 (ValVis "46" 1.0))
(key BOS-wdir-tl 39 (ValVis 160 1.0))
(key BOS-wspd-tl 39 (ValVis 17 1.0))
(key BOS-gusts-tl 39 (ValVis "-100" 0.0))
(key BOS-vis-tl 39 (ValVis "-100.0" 0.0))
(key BOS-press-tl 39 (ValVis "-100.00" 0.0))
(key BOS-humidity-tl 39 (ValVis "-100" 0.0))
(key BOS-altimeter-tl 39 (ValVis "-100.00" 0.0))
(key BOS-sunrise-tl 39 (ValVis "-100" 0.0))
(key BOS-sunset-tl 39 (ValVis "-100" 0.0))

(key BOS-clds-tl 42  (ValVis overcast 1.0))
(key BOS-temp-tl 42 (ValVis "49" 1.0))
(key BOS-max-tl 42 (ValVis "-100" 0.0))
(key BOS-min-tl 42 (ValVis "-100" 0.0))
(key BOS-dew-tl 42 (ValVis "46" 1.0))
(key BOS-wdir-tl 42 (ValVis 190 1.0))
(key BOS-wspd-tl 42 (ValVis 16 1.0))
(key BOS-gusts-tl 42 (ValVis "-100" 0.0))
(key BOS-vis-tl 42 (ValVis "-100.0" 0.0))
(key BOS-press-tl 42 (ValVis "-100.00" 0.0))
(key BOS-humidity-tl 42 (ValVis "-100" 0.0))
(key BOS-altimeter-tl 42 (ValVis "-100.00" 0.0))
(key BOS-sunrise-tl 42 (ValVis "-100" 0.0))
(key BOS-sunset-tl 42 (ValVis "-100" 0.0))

(key BOS-clds-tl 45  (ValVis overcast 1.0))
(key BOS-temp-tl 45 (ValVis "47" 1.0))
(key BOS-max-tl 45 (ValVis "-100" 0.0))
(key BOS-min-tl 45 (ValVis "-100" 0.0))
(key BOS-dew-tl 45 (ValVis "42" 1.0))
(key BOS-wdir-tl 45 (ValVis 220 1.0))
(key BOS-wspd-tl 45 (ValVis 18 1.0))
(key BOS-gusts-tl 45 (ValVis "-100" 0.0))
(key BOS-vis-tl 45 (ValVis "-100.0" 0.0))
(key BOS-press-tl 45 (ValVis "-100.00" 0.0))
(key BOS-humidity-tl 45 (ValVis "-100" 0.0))
(key BOS-altimeter-tl 45 (ValVis "-100.00" 0.0))
(key BOS-sunrise-tl 45 (ValVis "-100" 0.0))
(key BOS-sunset-tl 45 (ValVis "-100" 0.0))
```

```
(key BOS-clds-tl 48  (ValVis overcast 1.0))
(key BOS-temp-tl 48 (ValVis "44" 1.0))
(key BOS-max-tl 48 (ValVis "51" 1.0))
(key BOS-min-tl 48 (ValVis "36" 1.0))
(key BOS-dew-tl 48 (ValVis "39" 1.0))
(key BOS-wdir-tl 48 (ValVis 220 1.0))
(key BOS-wspd-tl 48 (ValVis 18 1.0))
(key BOS-gusts-tl 48 (ValVis "-100" 0.0))
(key BOS-vis-tl 48 (ValVis "-100.0" 0.0))
(key BOS-press-tl 48 (ValVis "-100.00" 0.0))
(key BOS-humidity-tl 48 (ValVis "-100" 0.0))
(key BOS-altimeter-tl 48 (ValVis "-100.00" 0.0))
(key BOS-sunrise-tl 48 (ValVis "-100" 0.0))
(key BOS-sunset-tl 48 (ValVis "-100" 0.0))

(key BOS-clds-tl 51  (ValVis partialcloudy 1.0))
(key BOS-temp-tl 51 (ValVis "43" 1.0))
(key BOS-max-tl 51 (ValVis "-100" 0.0))
(key BOS-min-tl 51 (ValVis "-100" 0.0))
(key BOS-dew-tl 51 (ValVis "35" 1.0))
(key BOS-wdir-tl 51 (ValVis 230 1.0))
(key BOS-wspd-tl 51 (ValVis 17 1.0))
(key BOS-gusts-tl 51 (ValVis "-100" 0.0))
(key BOS-vis-tl 51 (ValVis "-100.0" 0.0))
(key BOS-press-tl 51 (ValVis "-100.00" 0.0))
(key BOS-humidity-tl 51 (ValVis "-100" 0.0))
(key BOS-altimeter-tl 51 (ValVis "-100.00" 0.0))
(key BOS-sunrise-tl 51 (ValVis "-100" 0.0))
(key BOS-sunset-tl 51 (ValVis "-100" 0.0))

(key BOS-clds-tl 54  (ValVis partialcloudy 1.0))
(key BOS-temp-tl 54 (ValVis "41" 1.0))
(key BOS-max-tl 54 (ValVis "-100" 0.0))
(key BOS-min-tl 54 (ValVis "-100" 0.0))
(key BOS-dew-tl 54 (ValVis "30" 1.0))
(key BOS-wdir-tl 54 (ValVis 240 1.0))
(key BOS-wspd-tl 54 (ValVis 12 1.0))
(key BOS-gusts-tl 54 (ValVis "-100" 0.0))
(key BOS-vis-tl 54 (ValVis "-100.0" 0.0))
(key BOS-press-tl 54 (ValVis "-100.00" 0.0))
(key BOS-humidity-tl 54 (ValVis "-100" 0.0))
(key BOS-altimeter-tl 54 (ValVis "-100.00" 0.0))
(key BOS-sunrise-tl 54 (ValVis "-100" 0.0))
(key BOS-sunset-tl 54 (ValVis "-100" 0.0))

(key BOS-clds-tl 57 (ValVis sunny 1.0))
(key BOS-temp-tl 57 (ValVis "39" 1.0))
(key BOS-max-tl 57 (ValVis "-100" 0.0))
(key BOS-min-tl 57 (ValVis "-100" 0.0))
(key BOS-dew-tl 57 (ValVis "28" 1.0))
(key BOS-wdir-tl 57 (ValVis 260 1.0))
(key BOS-wspd-tl 57 (ValVis 16 1.0))
(key BOS-gusts-tl 57 (ValVis "-100" 0.0))
```

```
(key BOS-vis-tl 57 (ValVis "-100.0" 0.0))
(key BOS-press-tl 57 (ValVis "-100.00" 0.0))
(key BOS-humidity-tl 57 (ValVis "-100" 0.0))
(key BOS-altimeter-tl 57 (ValVis "-100.00" 0.0))
(key BOS-sunrise-tl 57 (ValVis "-100" 0.0))
(key BOS-sunset-tl 57 (ValVis "-100" 0.0))

(key BOS-clds-tl 60  (ValVis partialcloudy 1.0))
(key BOS-temp-tl 60 (ValVis "37" 1.0))
(key BOS-max-tl 60 (ValVis "-100" 0.0))
(key BOS-min-tl 60 (ValVis "-100" 0.0))
(key BOS-dew-tl 60 (ValVis "26" 1.0))
(key BOS-wdir-tl 60 (ValVis 230 1.0))
(key BOS-wspd-tl 60 (ValVis 11 1.0))
(key BOS-gusts-tl 60 (ValVis "-100" 0.0))
(key BOS-vis-tl 60 (ValVis "-100.0" 0.0))
(key BOS-press-tl 60 (ValVis "-100.00" 0.0))
(key BOS-humidity-tl 60 (ValVis "-100" 0.0))
(key BOS-altimeter-tl 60 (ValVis "-100.00" 0.0))
(key BOS-sunrise-tl 60 (ValVis "-100" 0.0))
(key BOS-sunset-tl 60 (ValVis "-100" 0.0))
```

Bibliography

[Aga95a] Agamanolis, S. "High-level Scripting Environments for Interactive Multi-Media Systems." SM Thesis Proposal. Massachusetts Institute of Technology, August 1995.

[Aga95b] Agamanolis, S. "Isis: A Multi-Purpose Interpretive Scripting Language Reference Manual and Tutorial." Internal Memo. MIT Media Laboratury, November 1995.

[Aga95c] Agamanolis, S. "Structured Video with Isis." Internal Memo. MIT Media Laboratury, November 1995.

[Aga95d] Agamanolis, S. "Structured Audio with Isis." Internal Memo. MIT Media Laboratury, November 1995.

[Bov91] Bove, V., Jr. and J. Watlington. "Cheops: A Modular Processor for Scalable Video Coding." *Proceedings of SPIE Visual Communications and Image Processing 1991*, Vol. 1605, pp. 886-893. SPIE, November 1991.

[Bov93a] Bove, V., Jr. and J. Watlington. "Cheops: A Data-Flow System for Real-Time Video Processing." Internal Memo. MIT Media Laboratory, June 1993.

[Bov93b] Bove, V. Jr. "Hardware and Software Implications of Representing Scenes as Data." *Proceedings of ICASSP - 1993*, pp. I-121-I-124. ICASSP, 1993

[Bov94a] Bove, V., Jr., B. Granger and J. Watlington. "Real-Time Decoding and Display of Structured Video." *Proceedings of IEEE ICMCS 1994*, pp. 456-462. IEEE, May 1994.

[Bov94b] Bove, V., Jr. "Object-Oriented Television." Paper presented at the *SMPTE 136th Technical Conference and World Media Expo.* SMPTE, October 1994.

[Bov95] Bove, V., Jr. and J. Watlington. "Cheops: A Reconfigurable Data-Flow System for Video Processing." *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 5, No. 2, pp. 140-149. IEEE, April 1995.

[Cha95] Chang, T. "Real-Time Decoding and Display of Layered Structured Video." SM Thesis. Massachusetts Institute of Technology, June 1995.

[Cor95] Correia, N., I. Oliveira, J. Martins, and N. Guimaraes. "WeatherDigest: an experiment on media conversation." *Proceedings of SPIE Integration Issues in Large Commercial Media Delivery Systems 1995*, Vol. 2615, in press. SPIE, November 1995.

[Ear93] Earnshaw, R. and D. Watson eds. *Animation and Scientific Visualization Tools and Applications.* Academic Press, Ltd.: San Diego, CA, 1993.

[Elo96] Elo, S. "PLUM: Contextualizing News for Communities Through Augmentation." SM Thesis. Massachusetts Institute of Technology, February 1996.

[Enc93] Encarnacao, J., et al. "Graphics and Visualization: The Essential Features for the Classification of Systems." *IFIP Transactions of Graphics, Design and Visualization*, Vol. B-9, pp. 3-18. IFIP, February 1993.

[Feh88] Fehrle, T., T. Strothotte and M. Szardenings. "Generating Pictorial Presentations for Advice-Giving Dialog Systems." *Lecture Notes in Computer Science, Visualization in Human-Computer Interaction*, Vol. 439, pp. 27-36. Springer-Verlag: New York, May 1988.

[Gra95] Granger, B. "Real-Time Structured Video Decoding and Display." SM Thesis. Massachusetts Institute of Technology, February 1995.

[Har89] Harashima, H., et al. "Model-Based Analysis Synthesis Coding of Videotelephone Images - Conception and Basic Study of Intelligent Image Coding." *Transactions of the IEICE,* E72(5), pp. 452-459. IEICE, 1989.

[Ing95] Inguilizian, A. "Synchronized Structured Sound." SM Thesis. Massachusetts Institute of Technology, September 1995.

[Kun88] Kunkel, K., T. Strothotte. "Visualization and Direct Manipulation in User Interfaces: Are we Overdoing it?" *Lecture Notes in Computer Science, Visualization in Human-Computer Interaction*, Vol. 439, pp. 183-193. Springer-Verlag: New York, May 1988.

[Mar93] Martin, J., "Visualization for Telecommunications Network Planning." *IFIP Transactions of Graphics, Design and Visualization*, Vol. B-9, pp. 327-334. IFIP, February 1993.

[McC87] McCormick, B., T. DeFanti and M. Brown. "Visualization in Scientific Computing." *Computer Graphics* 21(6). 1987.

[McL91] McLean, P. "Structured Video Coding." SM Thesis. Massachusetts Institute of Technology, May 1991.

[Mus89] Mussmann, H., M. Hotter and J. Ostermann. "Object-Oriented Analysis-Synthesis Coding of Moving Images." *Signal Processing: Image Communication I*, pp. 117-138. May 1989.

[Ous94] Ousterhout, J. *Tcl and the Tk Toolkit.* Addison-Wesley: Reading, MA, 1994.

[Par88] Parker, S. *Meteorology Source Book*. McGraw-Hill: NewYork, NY, 1988.

[She92] Shen, I. "Resource Manager foir a Video Processing System." SM Thesis. Massachusetts Institute of Technology, May 1992.

[Tho94] Thomas, E. Jr. "You May Not Know Your Weatherman is in Jackson, Miss." *The Wall Street Journal*, sec. 1: 1, 10. November 2 1994

[Tuf83] Tufte, E. *The Visual Display of Quantitative Information*. Graphics Press: Cheshire, CT, 1983.

[Tuf90] Tufte, E. *Envisioning Information*. Graphics Press: Cheshire, CT, 1990.

[Wal91] Wall, L. and R. Schwartz. *Programming perl*. O'Reilly & Associates, Inc.: Sebastopol, CA, 1991.

[Wat95] Watlington, J. and V. Bove, Jr. "Stream-Based Computing and Future Television." 137th SMPTE Technical Conference, September 1995.