Implementation of a Temporal Typography System

by Douglas Soo

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science at the Massachusetts Institute of Technology

[illegible]

Signature of Author
Department of Electrical Engineering and Computer Science
February 4, 1997

JOHN MAEDA

Certified by
John Maeda
Assistant Professor of Design and Computation
MIT Media Laboratory
Thesis Supervisor

[illegible]

Accepted by
F. R. Morgenthaler
Chairman, Department of Graduate Theses

Implementation of a Temporal Typography System

by Douglas Soo

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science at the Massachusetts Institute of Technology

February 4, 1997

Abstract

Temporal typography is the dynamic treatment of written language. This thesis offers an analysis of the design and implementation of systems used for authoring temporal typography. It presents a broad overview of the issues involved in authoring temporal typography, and proposes solutions to improve authoring, and evaluates these solutions through the use of experiments. Experiments are performed to evaluate the viability of these solutions. These experiments includes experimentation with transparency and three-dimensional graphics in a browsing system, the use of electromagnetic field sensors as an input device and an automated design system used in a networked messaging application. It also describes the real-time rendering temporal typography rendering engine and scripting language implemented for use in these experiments.

Thesis Supervisor: John Maeda
Title: Assistant Professor of Design and Computation, MIT Media Lab

This work was supported in part by the MIT Media Laboratory News in the Future Consortium

Acknowledgements

Thanks to all those at the Media Lab who have helped me in my endeavors. My thesis advisor, John Maeda, whose energy and creativity inspire all those around him. Ron MacNeil, who first entrusted work to me as an undergraduate researcher. Yin Yin Wong, whose work inspired this thesis project, and whose confidence in my work gave me the push I needed. Walter Bender, who always made sure that I had funding, even though I saw him far too rarely. Of course, all of the other graduate students at the Visible Language Workshop and Aesthetics and Computation Group, who created a relaxed atmosphere, and never let me get too bogged down in my own work.

More thanks go to my family, who always encouraged me. And thanks also go to all my various friends and roommates, who were always entertaining, who always egged me on, but who were always supportive in times of trouble.

Table of Contents

1. Introduction

As the computer has become present everywhere in today's world, from the office to the home, it has become increasingly important as a medium for communications. Especially with the advent of local networks in the work environment and on-line services at the home, an increasing amount of communications is being done with computers.

Most of this communication is being done using electronic mail or chat systems that use unenhanced text as their medium. While recently, e-mail support has started to include basic text formatting available in word processors by using HTML and other text formats, this kind of text is a difficult medium for expressing emotion, and conveying subtleties that are often transmitted almost subconsciously in face to face or vocal communication. While people have used novel and inventive ways to increase the depth of textual conversation by using mechanisms such as smileys [Bruckman 93], these mechanisms are often cryptic, and often not understood universally. One solution to this kind of problem is the use of new technologies, such as audio and video conferencing, which can be used to simulate a face to face conversation.

However, this approach is not entirely satisfactory. Audio and video are high-throughput media, and make it difficult to attain acceptable quality except by using high bandwidth network connections. Also, it is unlikely that computer technology will be able to provide a satisfactory simulation of face to face conversation in the near future. An approach that

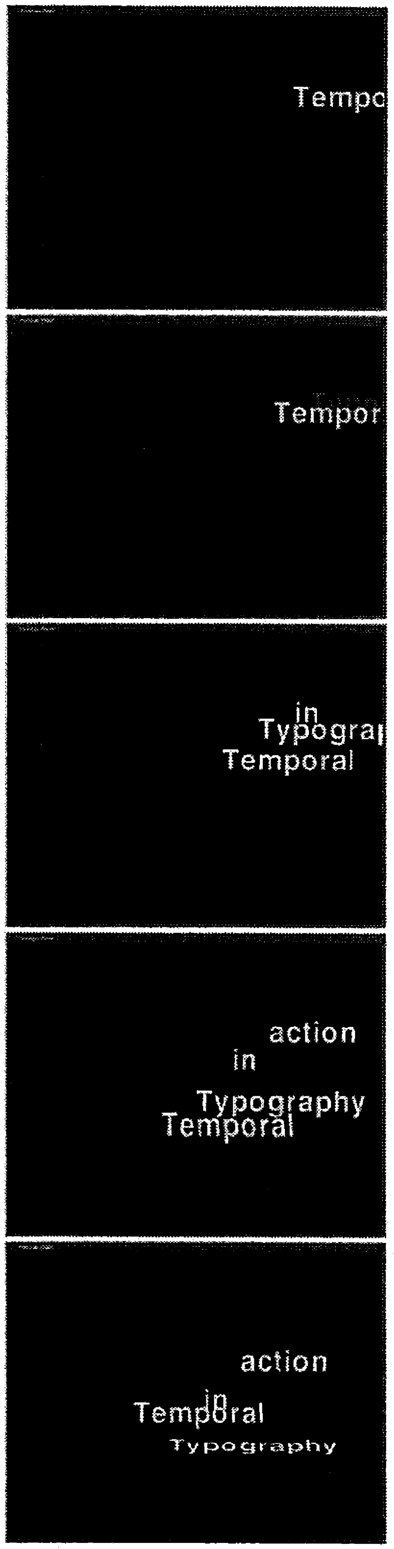


Figure 1-1:An example of temporal typography, viewed from top to bottom.

took advantage of the various abilities of the computer would be a more interesting approach.

One approach that can be taken is the use of temporal typography as a communications medium. Temporal typography is a dynamic treatment of written text - in essence, text that changes its form over time.. Common examples of temporal typography include "flying text" that is seen in commercials, and introductory sequences in films and television.

Temporal typography is able to take advantage of the qualities of the computer that differentiate it from paper - namely, an interactive dynamic medium, which allows for much more expressive capability. With the increasing power and decreasing cost of computer systems, the quality and cost of such graphics is good enough that such experiments can be performed using reasonably inexpensive hardware. In addition, temporal typography maintains some of the qualities of text that are advantages over audio and video mediums, such as the ability to do off-line composition, and relatively low bandwidth requirements.

While temporal typography addresses the issues of expressivity and off-line composition, there is currently a major problem with using it for communications - that of authoring. Like most other dynamic mediums, such as music and animation, authoring is a very time consuming process because of the complexity of the medium. Adding the dimension of time to typography greatly increases the amount of data that needs to be dealt with.

This thesis examines the issues that are involved in the creation of tools to author temporal typography, and outlines several possible solutions to these issues. These solutions include improving browsers and editors by the use of transparency and three-dimensional graphics, the use of new types of input devices such as electromagnetic field sensors in order to make control more intuitive, and the use of automated design to reduce the complexity that a user has to control directly.

In order to examine these issues, several tools and examples were created. These include:

- A real-time temporal typography rendering engine, and its scripting language.

- A browser and editor to the rendering engine, which uses transparency and three-dimensional graphics.

- Two examples of the use of electromagnetic field position sensors in the creation of temporal typography pieces.

- A network messaging system, TypoTalk, which uses a simple automated design template language.

Organization of this thesis

This thesis is organized into several sections. Chapter two contains background information, including previous research into temporal typography and automated design, as well as examples of expressive typography. Chapter 3 offers a broad overview of the general design issues involved in temporal typography authoring tools. Chapter 4 covers the design of the rendering framework and scripting language that were used to render temporal typography in the design experiments. Chapter 5 describes the various design experiments - the browser/editor interface, alternative input devices, the template language and the TypoTalk temporal typography chat system. Chapter 6 draws conclusions from the research done, and provides insight into future research in the area. The appendices include more detailed technical descriptions of the tools used in the design examples, primarily the syntax of the scripting language and template language.

2. Background and Related Work

Traditional typography is the form of static text on a page of paper. There have been many experiments in the past with the use of typography to convey emotion and meaning. This area of typography is called "Concrete Poetry". One example of this is the work "The Bald Soprano" (Figure 2-1), where type is used to communicate the emotion and nature of communication between characters in a play [Massin 1956].

The study of typography and graphic design has resulted in a formalized language, which makes it easier to describe a piece, and what elements lend it its appearance. However, because dynamic typography has not until recently been a medium in which formal research has been done, there has not been a similar language in which to describe temporal typography.

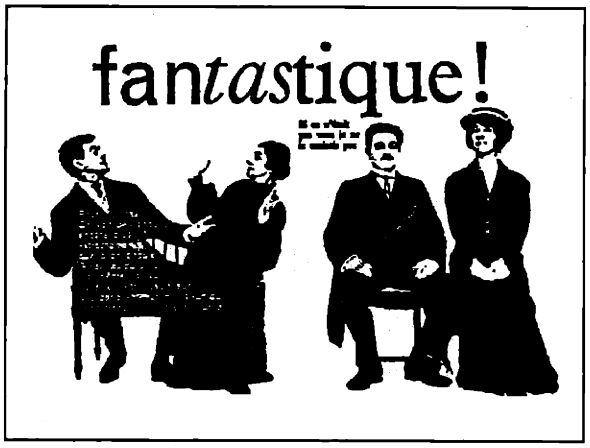


Figure 2-1: An example of expressive typography taken from "The Bald Soprano" [Massin 1956]

This is the area which Yin Yin Wong, in her thesis entitled *Temporal Typography: Characterization of time-varying typographic forms* [Wong, Y. 1995] researched. In it, she describes a formal structure for the characterization of temporal forms of typography. The characterization that she developed is based on the notion of dimension. Dimensions are attributes of form that can be changed over time, such as position, size and color.

The description of a temporal form is done by describing the modifications of the dimensions of an object. She describes a hierarchical structure, starting at the lowest level with a *visual technique*. She defines a visual technique as "a primitive unit which describes how a specific typographic dimension changes over a period of time." Each conceptual layer is composed of elements of the layer below it. For example, a *visual action* is a group of visual techniques that is combined together to form a complex expressive behavior, and an *expression* is composed of a sequence of several visual actions. At the highest level is a *presentation*, which is "a composition of expressions presented in an integrated manner for the purpose of visual communication." [Wong, Y.

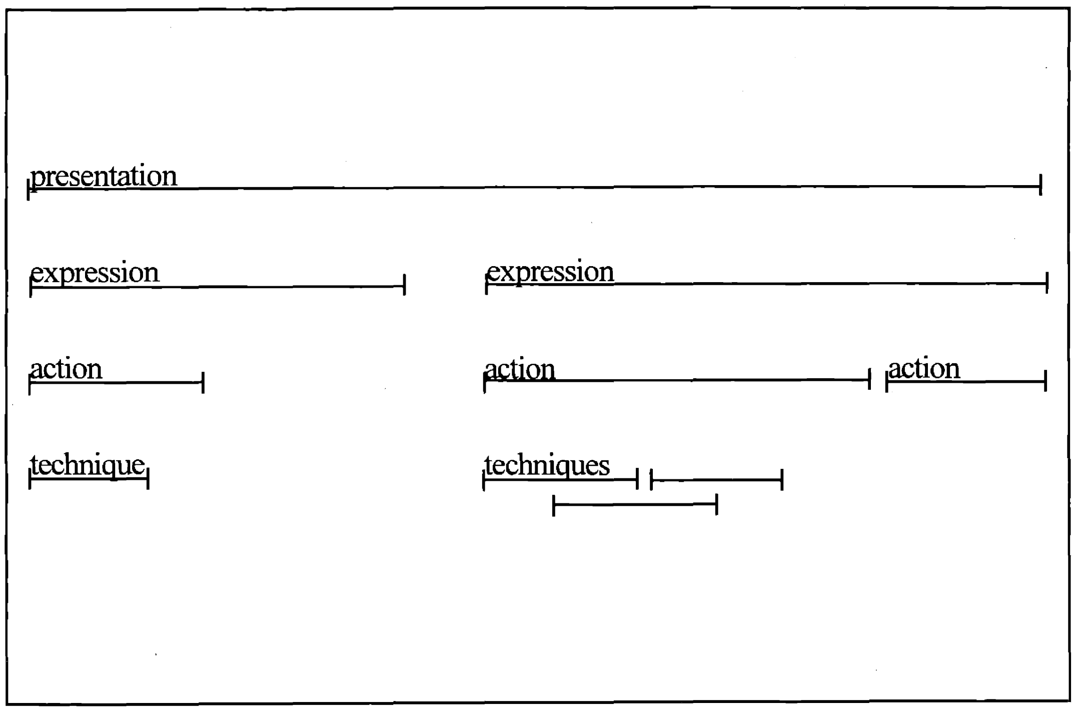


Figure 2-2: A diagram showing a sample hierarchy of a visual presentation.

1995] This hierarchical structure is the basis for the temporal typography tool that is described in Chapter 4.

Automated design is an important approach in improving any kind of authoring system. In particular, automated design which is able to adapt to changing information and visual spaces is important. In Louis Weitzman's thesis, *The Architecture of Information* [Weitzman 95], a large part of his thesis revolved around the use of grammatical systems in order to create automated visual design that adjusts itself to different forms.

An important concept that he uses is that of "meta-design", where flexible descriptions of designs are made. These descriptions are capable of responding to the environment and the information being represented. Meta-design is an extension of the use of design templates and rules, allowing an increased amount of flexibility in relation to both the information being presented, as well as the medium being used to present it.

Weitzman implements meta-design through the use of Relational Grammars, which are able to describe the visual relationship between objects, and which describe the design of the system in a manner that allows for changes in the medium that the information is

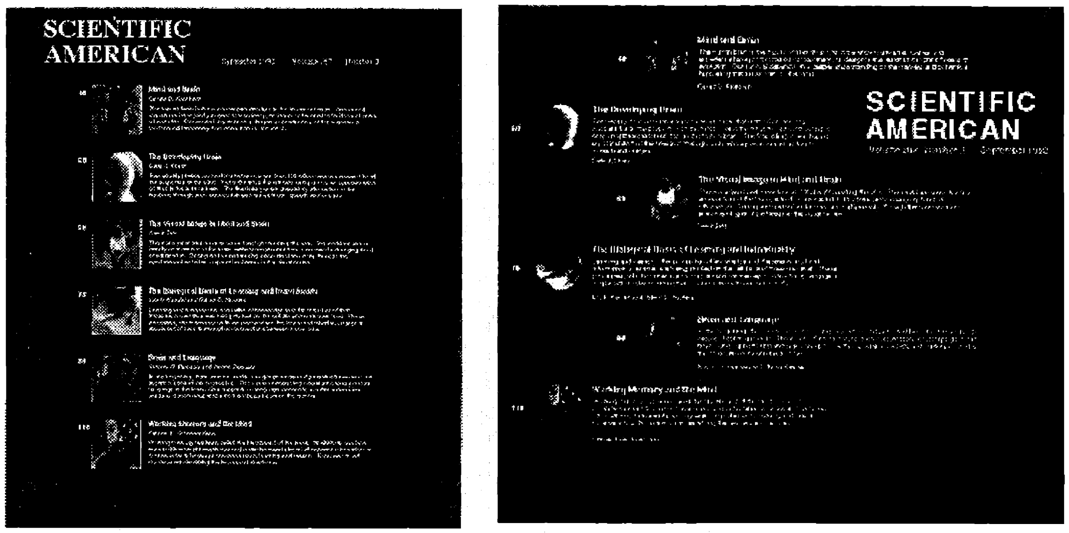
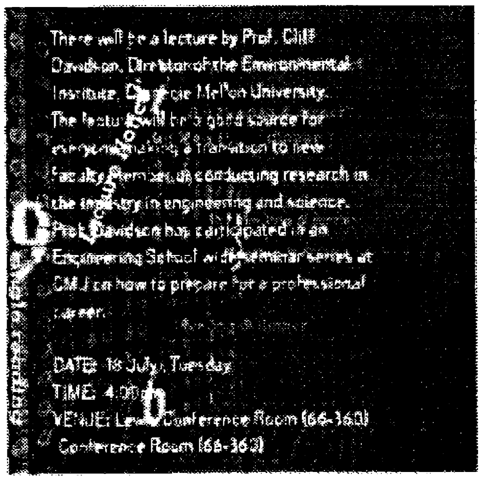


Figure 2-3: Weitzman's VIA system automatically redesigned information to fit different styles using Relational Grammars.

being displayed in. Relational grammars are an extension to traditional string languages, and function by specifying constraints between the different members.

Another related approach is taken by Suguru Ishizaki, in his thesis *Typographic Performance* [Ishizaki 1995]. Ishizaki also examines the ability to create designs that are able to deal with changing information content. However, instead of approaching this with the use of relational grammars, Ishizaki proposes a model of dynamic design with multiple agents, which a visual designer uses in order to create designs that continuously change over time.

He uses a decentralized theory of design that is based on temporally based performing arts such as dance and music, as well as on multiagent systems. In his model, each particular segment of the design is represented by a design agent, which communicates with the other agents in order to create a design solution.


There will be a lecture by Prof. Cliff Davidson, Director of the Environmental Institute, Carnegie Mellon University. The lecture will be a good source for [illegible] a transition to new faculty members conducting research in the industry in engineering and science. Prof. Davidson has participated in an Engineering School [illegible] seminar series at CMU on how to prepare for a professional career.

DATE: 18 July, Tuesday
TIME: 4:00
VENUE: [illegible] Conference Room (66-360)
Conference Room (66-360)

Figure 2-4: Ishizaki's e-mail example uses design agents to automatically redesign itself whenever a new piece of mail arrives.

One example that was implemented using his agent system was that of an electronic mail reader. As a new piece of electronic mail arrives, various agents are told of the mail's arrival, and the Sender agent positions the piece of mail. If the user clicks on a piece of mail, the sender agent informs a message agent, and the message agent brings the message to the foreground, while other agents defocus their text or render their text translucent in order to make it easier to read the message. The ability of the various elements of the design to react to each other based on the information that is presented to them allows the system to adapt to different circumstances, making it much more flexible, and maintaining the quality of the design.

Examples of Temporal Typography

Before the advent of computer technology, temporal typography was primarily used in the film and television industry. Temporal typography was most often used in title sequences to make the name of the movie as memorable as possible. Several good examples of temporal typography have been done by Saul Bass - who used temporal typography extensively in the title sequence in the Alfred Hitchcock movie *Psycho* (Figure 2-5), as well as in *North By Northwest*. In the introduction to *Psycho*, the text entering as the horizontal lines crossed the screen gave the words a jarring feel, that caused the viewer to feel the unease associated with the word "psycho" [Bass 1960].

Interactive temporal typography pieces are not as common, because the technology necessary to create these pieces did not exist until recently. In *Flying Letters*, John Maeda uses dynamic typographic experiments in an attempt to amuse the user, and inspire new ways of thinking about the use of text on computers [Maeda 1996].

Other interactive work includes that by Peter Cho, in which he takes the letterform of a single letter - A - and distorts it

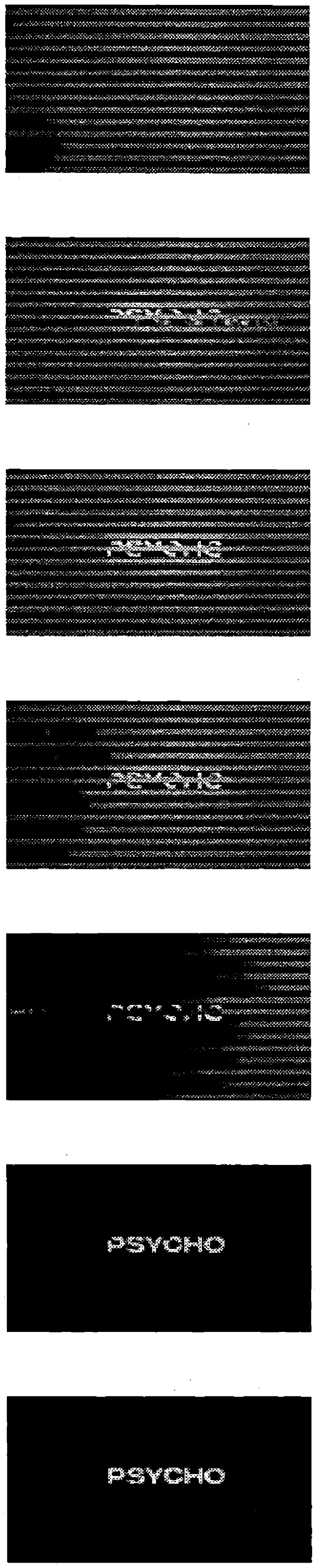


Figure 2-5: Sequence from credits of Hitchcock's Psycho

in a playful and interactive way [Cho 96]. By moving the mouse around the screen, it appears almost as if the letter is dancing and smiling at you, because of the way the crossbar bends into a curve. His work is an excellent example of work where the letterform itself is distorted in order to give a sense of life to the letter.

Another area where temporal typography has been used is in video and computer games. One example of a computer game that uses temporal typography is *You Don't Know Jack*, a trivia game [Berkeley Systems 1996]. Temporal typography is used within the game to introduce questions, and impart a sense of excitement. While the quality of the temporal typography is reasonably good - anti-aliased and at a reasonable frame rate - it is non-interactive. The temporal typography is played back in movie form. Incorporating interactive temporal typography that would interact with the user, and work with not only the introductory sequences but with all the text throughout the game would help a great deal with the atmosphere of the game.

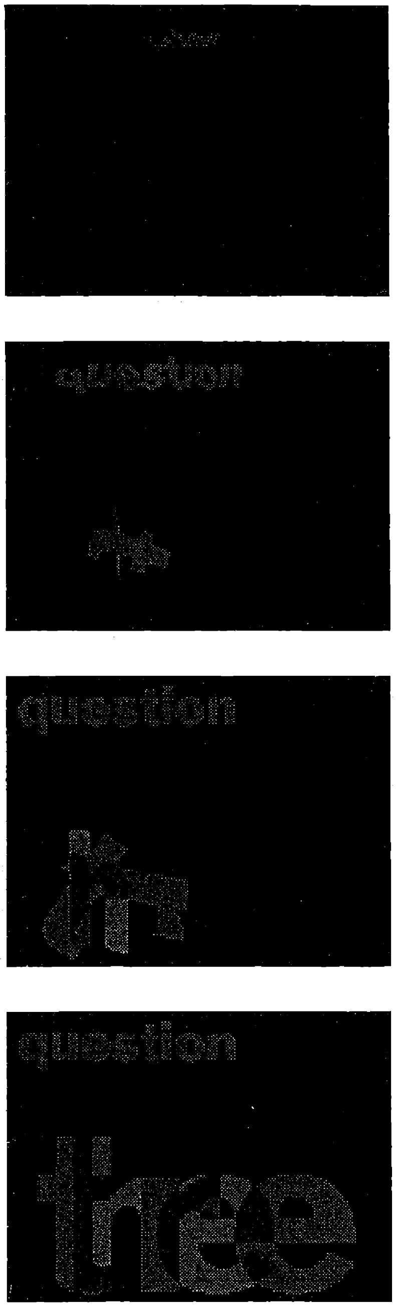


Figure 2-6: You Don't Know Jack uses temporal typography to introduce its questions.

Existing Temporal Typography Tools

There are currently few tools which are designed to explicitly work with temporal typography. The ones that do exist tend to be high end tools used for video and movie production.

One of the biggest problems with these tools is that they are unable to render their output in real time - in other words, it is not possible to view the temporal typography in its final form as it is created - there is usually a rendering stage that is necessary, in order to see the results of manipulation. Thus, none of these tools are suited to any kind of system where content is generated in real time - a chat system, for example, would be impossible to implement.

In addition to these high end typography tools, tools for use in multimedia productions such as Macromedia Director can be used to create temporal typography. However, these tools are not easy to use in such a manner because they were not designed to be used for temporal typography, but for other authoring purposes. Adapting them to produce high quality temporal typography output has been difficult. Again, it would be difficult, if not impossible, to use these tools to create a high quality real time temporal typography tool.

3. Design Overview

Temporal Typography Authoring Issues

Temporal typography suffers from the difficulty of representing time (and rhythm), like all other mediums that have time as one of their primary dimensions, such as music, dance, and cel-based animation. In addition to the traditional properties of text (size, rotation, font, etc.), the addition of mutability of change over time significantly increases complexity. Thus, one of the primary problems involved in authoring is attempting to find a representation for a multidimensional space that is easy to comprehend and manipulate.

The design experiments that are described in this thesis attempt to address the problems of temporal typography authoring by using three different approaches - improved editing tools, input devices/methods and automated design. These approaches try to improve authoring by focusing on two different problems - by visualizing large amounts of information, and by reducing the amount of information needed to author a temporal typography piece.

Designing improved temporal typography editing tools that can enhance the presentation of all the information needed to edit a temporal typography piece is at most the simplest, but most difficult approach. This approach requires the least technological development, but requires a good understanding of the way that people can visual large quantities of data and not become overwhelmed. In this thesis, an attempt is made to improve the presentation of information by taking advantage of high quality transparency and three-dimensional graphics.

The second approach is the use of nontraditional input devices and methods in order to make authoring more intuitive. The use of a keyboard and mouse is a very limiting method of creating a temporal typography piece, especially when the use of a mouse is generally reduced to pointing at a certain location and clicking. In face-to-face human communication, much of the information is being conveyed not in the words, but in the gestures that are being used by the hands of the person that is communicating. In the experiments outlined in this thesis, input devices that are able to track hand position in a much less limiting fashion are used to examine the viability of such an approach.

A third approach to improving temporal typography authoring is that of automated design. By reducing the amount of information that needs to be handled in a temporal typography piece, it would be easier to deal with the scope of a large temporal typography piece. One way to do this would be by simply using higher level temporal typography styles, and not editing low-level techniques - in other words, using dumb templates. However, this approach has problems. Much of the time spent in creating a temporal typography piece comes not at the high level design of general expressions and motions, but from the low level tweaking - making words appear at the proper rhythm, at the proper place, making a smooth flow. A dumb template language would be fast, but would be unable to do the tweaking necessary, resulting in a much lower quality design. By using much more "intelligent" automated design that was able to properly deal with these low level details, the high quality that is important in temporal typography would be maintained.

Description of Experiments

Initial work done to improve temporal typography authoring involved the development of an experimental browser which attempted to take advantage of high quality three-dimensional graphics in order to make it easier to browse an object hierarchy representing a temporal typography piece. This system was used on top of the temporal typography rendering framework and scripting language which is described in the next chapter.

Another set of systems experimented with using different input devices. These experiments (TTT Poem and Fable Table) both incorporate electric field sensors, which are able to determine the position of a user's hand in space. TTT Poem involved using sensors which were placed underneath the surface of a desk, while the Fable Table used sensors embedded into a wooden table, with a projector displaying images on the surface of the table.

The last experiment, TypoTalk, was done in order to test the use of an intelligent template system to automatically generate temporal typography. The TypoTalk system is a networked intercomputer messaging system, where the user selects a style for the temporal typography piece. The template system also includes a dictionary, which can parses the text looking for words with associated styles, and render the text using those styles.

Chronology of Research

All of the experiments were done over a period of two years. The initial work was done as an undergraduate researcher at the Visible Language Workshop under Yin Yin Wong, for her thesis on creating a formal characterization of temporal typography. Work then continued on independently.

Initial design and implementation of the temporal typography rendering system was done during January of 1995. Further development of the scripting language occurred during the spring and summer of that year, and development of the editing tools occurred at that time began to occur at that time as well. Development of the template language began during the spring of 1996, and in the fall of 1996 work on input devices and the TypoTalk system was done.

4. Basic Rendering Framework

In order to perform experiments on temporal typography authoring, it was first necessary to develop a system that would be able to render real time temporal typography. In addition, this rendering tool needed to have a scripting language to control it that would be simple, but flexible enough. This chapter contains a description of the tool that was originally designed for Yin Yin Wong's experiments with temporal typography, but was adapted for use in experimenting with authoring.

Scripting Language Design

In designing a temporal typography scripting language, there were two key design goals: simplicity and flexibility. This was necessary because the goal was to have a simple language that could be used for the creation of multiple different sample temporal typography pieces and experiments, each of which could be vastly different.

The scripting language and system that were designed closely parallel the characterization of temporal form that was developed by Yin Yin Wong [Wong, Y. 1995]. The two main concepts behind this language are its use of *visual techniques* and *key variables*.

A *visual technique* is essentially an action which affects one or more of the properties of an object (called a *plan*). For example, a visual technique could rotate a piece of text around its center, or change its color. A visual technique's behavior is controlled by that

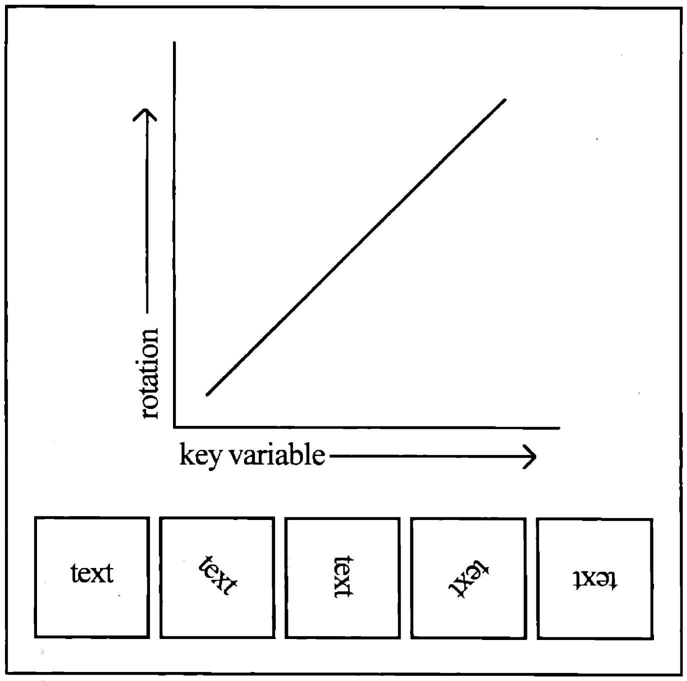


Figure 4-1: A diagram showing the relationship between a key variable and the parameter of a visual technique

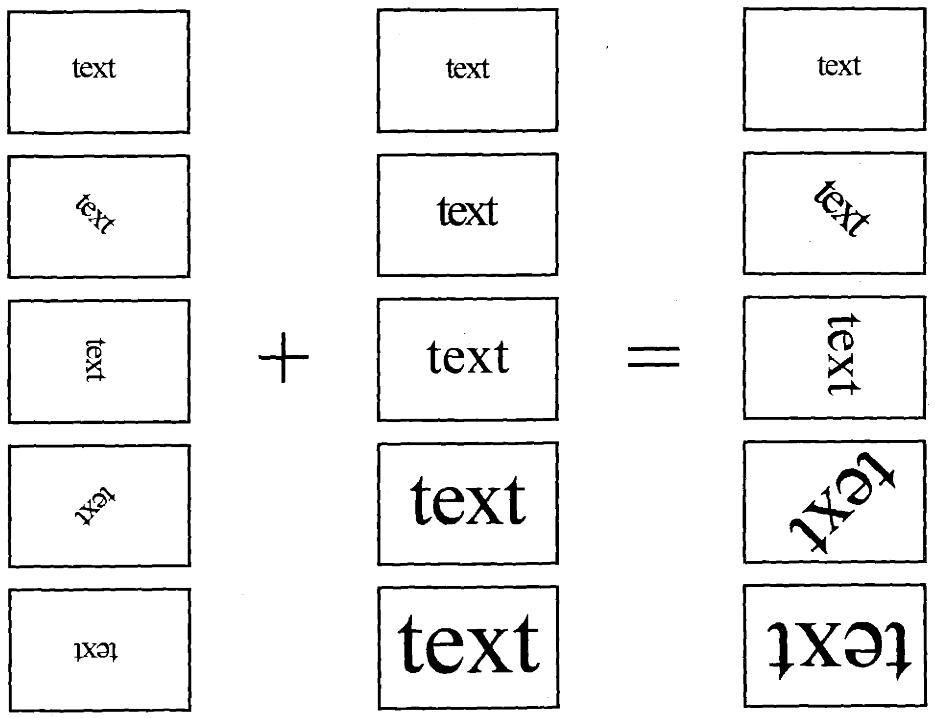


Figure 4-2: The effect of composing multiple visual techniques on a single item.

the *key variable* that it is associated with.

A *key variable* is a parameter that affects the behavior of objects. It can be time, mouse position, strength of an electromagnetic field sensor, or any other property that can be characterized as a numeric value. Key variables allow for flexibility in the design of pieces - in this way, a piece can be designed for a conventional time-based playback, or playback based on the position of the user in a three-dimensional space.

Each visual technique is associated with a single key variable. Depending on the value of the key variable, the parameter that is being affected by the visual technique will be interpolated. For example, if the key variable is time, the parameter being affected by the visual technique will transition smoothly with the passage of time over the range in which it is active.

An important property to note about visual techniques is that they have a cumulative effect on the plan that they are associated with. Thus, multiple visual techniques can overlap, and affect the same property of the object. Also, visual techniques have different interpolation methods. These interpolations are used to modify the behavior of

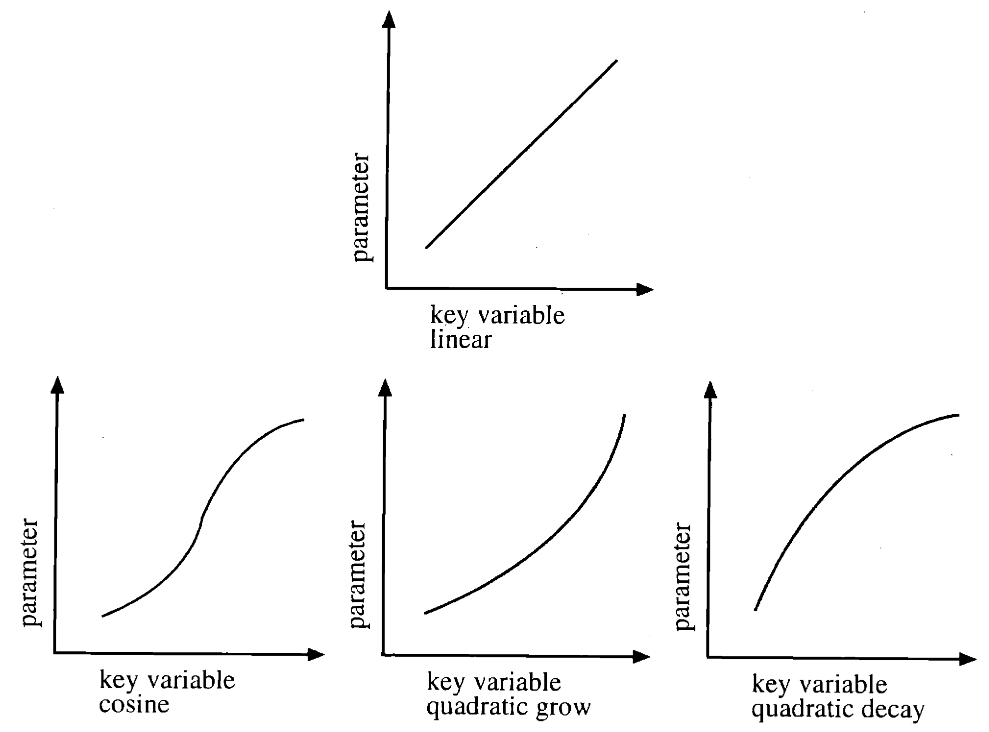


Figure 4-3: Examples of different types of interpolation methods.

the visual technique over the range in which they take place - for example, a cosine technique could be used to stop and start an action smoothly, resulting in an action that appears to ease in and ease out. There are currently four types of interpolation methods (Figure 4-3), with the possibility of adding more.

However, while the scripting language and renderer function worked, it suffered from several limitations. The largest limitation was the fact that their was no way of organizing either the visual techniques or the plans into any form of hierarchical structure. Thus, as the complexity of a presentation grew, it became very difficult to organize the structure. Also, since there was no way of grouping visual techniques, there was no easy way of reusing techniques that had been used before.

Thus, another version of the scripting language was designed, in which there was a hierarchical structure. Each visual technique was allowed to have multiple visual techniques as its children, and each plan was allowed to have multiple plans and visual techniques as well. In addition, each of these objects was given the ability to have a names, so each item could be copied and reused.

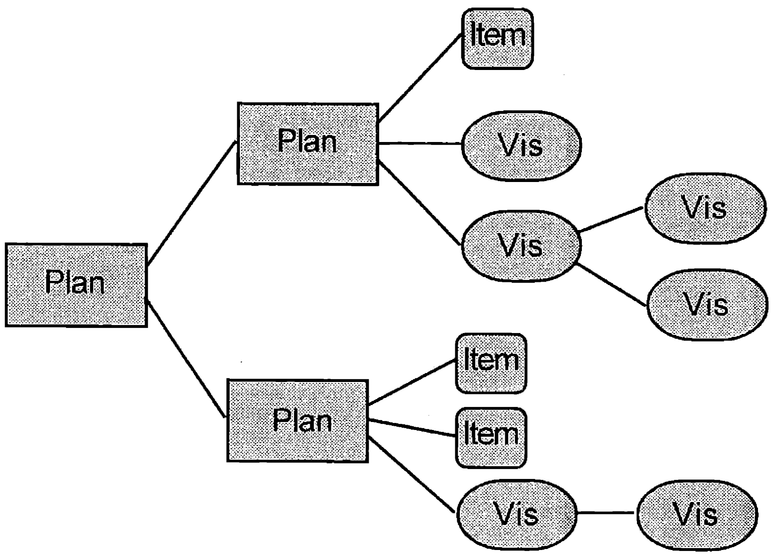


Figure 4-4: Example of hierarchical presentation structure.

Another problem which hindered the original scripting language was that there was no local scope. Composing visual techniques in a global scope would be very difficult, as it would be necessary to determine when in global time each individual action was occurring, and then whenever you wanted to change when an action was occurring, it would be necessary to edit the values of each of the individual element. Also, in order to change the duration of a complex visual action, it would be necessary to manipulate all of the children composing the visual action.

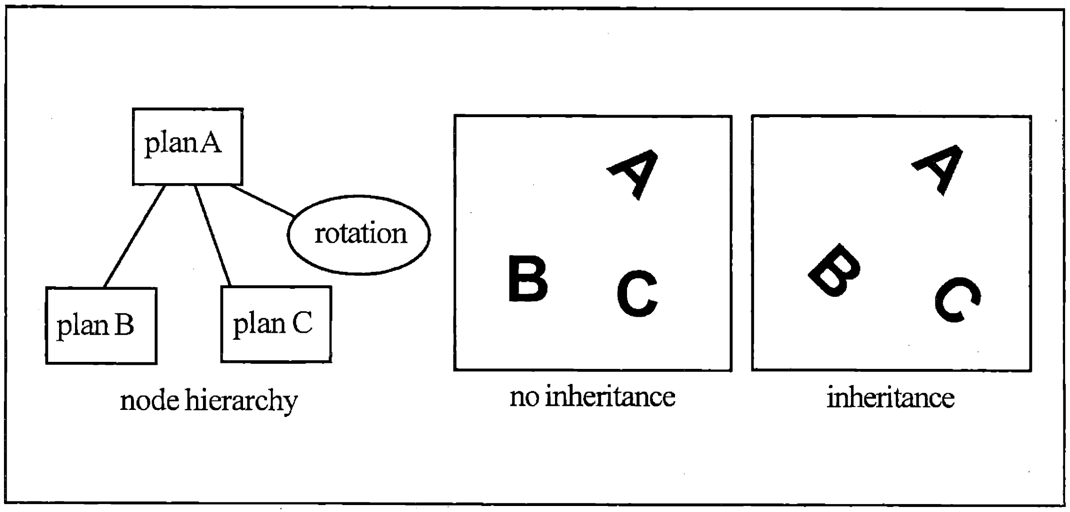


Figure 4-5: Example of the effects of inheritance of visual techniques

In order to solve the scoping problems, *key offsets* and *key scales* were added to every visual technique and plan. A key offset shifts the value of each of the key variables, so it is possible to move a visual technique by just changing the offset. This makes it simpler to move complex actions. A key scale expands or shrinks the value of the key variables. For example, a key scale of 2 would multiply the value of the key variable by 2, so if the key variable were time, things would happen twice as fast.

The hierarchical structure also resulted in another concept that was added to the scripting language - that of inheritance. Since it was now possible to have nested plans, there was an issue of whether the visual techniques of the parent plan should affect the child plan. It was decided to have a flag in the visual technique that would toggle if it was inherited or not. This is a useful capability, as it makes it simpler for different items which have a common visual technique (such as text which all have the same font) to share it, by simply including them under a common plan with an inherited visual technique.

Rendering Engine

Designing the scripting language and the functionality of the rendering engine took place simultaneously, by necessity. The basic rendering framework used in the experiments described in this thesis needed certain important functionality beyond that determined by the scripting language. It was important that animation of the temporal typography occur in real-time. In addition, the spacial quality of the text had to be high, necessitating the use of anti-aliased text, and the use of three-dimensional graphics.

In addition to maintaining the quality of the text, there were other issues that had to be addressed in the system. The system needed to be flexible, but simple. In order to be able to accommodate the various different experiments, it was necessary that it be relatively simple to add functionality such as new input devices, new types of visual techniques, and other things of a similar nature. The time required for implementation of these new pieces had to be relatively short. Thus, it was necessary for the system to be extremely modular in nature, with as little interdependency on various types of software and hardware.

In addition, there was a desire to make the system fairly easy to port from one graphics technology to another. This was important because of the desire to possibly use the system for communications on multiple different platforms. Again, modularization simplified this goal. Platform specific parts of the system were abstracted away into a core which could be easily and quickly replaced, surrounded by platform independent code.

Implementation

Initial implementation of the rendering framework occurred in January of 1995, concurrent with the development of the scripting language. It was done using Silicon Graphics hardware, using C++ and the Iris Performer graphics toolkit, as well as an anti-aliased bitmapped font library. This allowed for the manipulation of text in three-dimensional space in real-time, as well as the use of important features such as transparency.

The new version of the renderer and scripting language was then developed, again in Performer and C++. Later this was ported to the Open Inventor toolkit, but the basic framework (with the exception of the specific graphics toolkit commands) remained essentially the same. It is this version that will be discussed in depth here.

In rendering a single frame of a temporal typography presentation, the rendering engine takes the following steps:

1. The values of all of the key variables are determined. This is done either by polling system calls, such as in getting the current time or mouse position, or by getting the state of a device, such as the fish sensor.

2. The values of the key variables are then passed to the base plan of the node hierarchy. At this point, the program begins a traversal of the node hierarchy, starting with the plan at the base.

2a. If the node is a plan, the first thing that is done is to modify the key variables by their offset and scales. This is done in order to place the key variables in the local context of the plan. Then, the plan calls the update methods for each of the child visual techniques, which results in a stack of currently active visual techniques. Then, it initializes the state of each of its child items, returning it to a default state (translation, rotation, scale, etc.). The visual techniques in the stack are then applied to each item, modifying its state. It then removes visual techniques that are not to be inherited by its child plans, and then calls the update methods for each of its children. Afterwards, it removes its visual techniques from the stack.

2b. If the node is a visual technique, as with plans, the key variables are modified by their offsets and scales. Then, the visual technique determines if it is active by checking the persistence properties and the value of the key variables.. If it isn't, it stops traversing the tree. If it is, it adds itself to the stack of visual techniques, and then does an interpolation to determine what its behavior should be.

3. The frame is rendered.

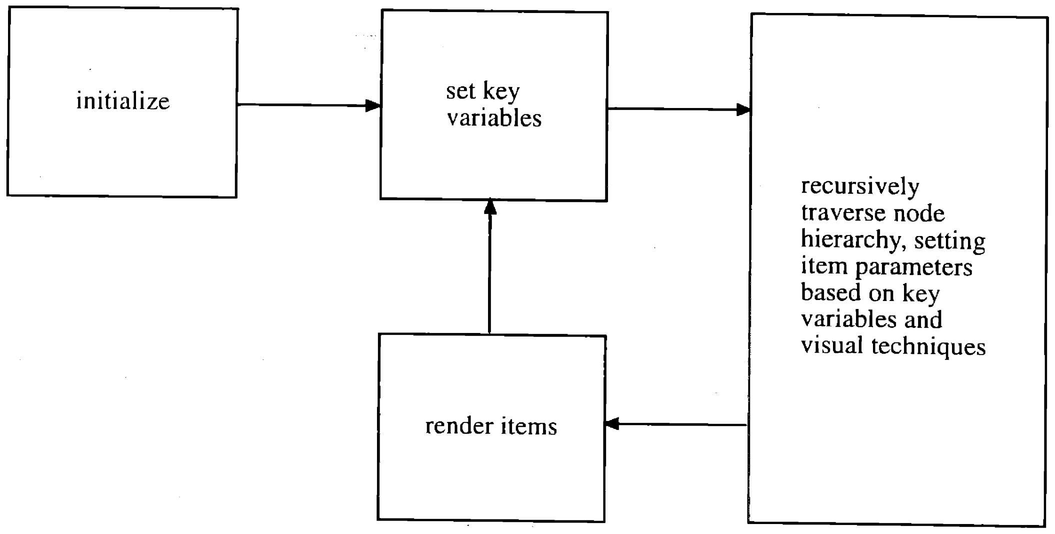

initialize
set key variables
recursively traverse node hierarchy, setting item parameters based on key variables and visual techniques
render items

Figure 4-6: Rendering engine flow chart.

5. Design Experiments

5.1 Browser/Editor

Browser/Editor Design

One of the problems with the scripting language was that by just reading the scripting language in text form it was very difficult to determine what a particular script would do and what effect changes to it would have. In order to be able to edit and manage a presentation well, it was necessary to have some sort of browsing and/or editing system that would make it easy to visualize the structure of a temporal typography.

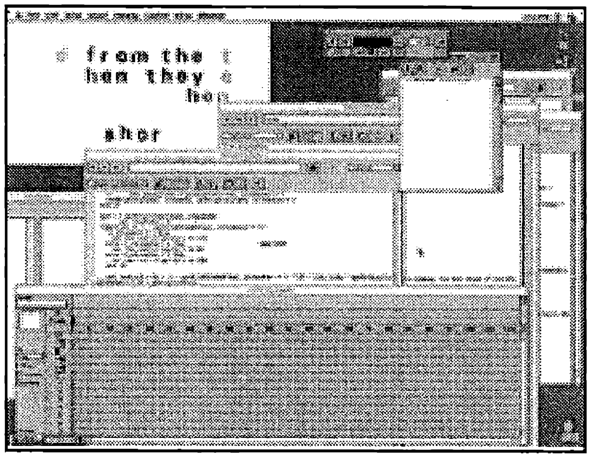


Figure 5-1: An example of a typical Macromedia Director editing screen, courtesy of Reed Kram

Thus, one of the ideas that was developed was that of a browser or editor system for temporal typography, similar to those used to look at the class hierarchies in an object oriented language, or that of a system for scoring music. This system would make it possible to perform real-time editing of temporal typography: As parameters of the temporal typography piece were changed, the effects would be displayed immediately, instead of having to restart the renderer.

However, the approach that commercial tools use approaches for browsing and editing that are not well suited for the development of temporal typography. In order to understand the limitations of these systems, the interfaces used by two different sets of tools will be discussed - the approach used by three-dimensional modeling tools, and the approach taken by multimedia development tools such as Macromedia Director [Macromedia].

Three-dimensional modeling programs often use a combination of a scene graph and key frames in order to represent the movement of an object through space and time. The scene graph is a hierarchical tree which represents the relationships between objects in the scene, usually with orthogonal matrices to represent the position, orientation, and scaling of the object. However, such a structure does not easily lend itself to representing what changes are occurring to an object over time, but only the fact that such a change is occurring. These tools are unable to generate temporal typography in a nonlinear, dynamic manner.

The second group of tools are tools that are used for the development of multimedia programs - programs that incorporate video, sound, and computer animation in a nonlinear way. One such program is Macromedia Director. Director's approach to authoring is taken from that of a movie as well, with cast members and actors performing various actions, where the workspace is called a stage.

Director uses a timeline approach, where there is a timeline window with different frames on the horizontal axis, and the different

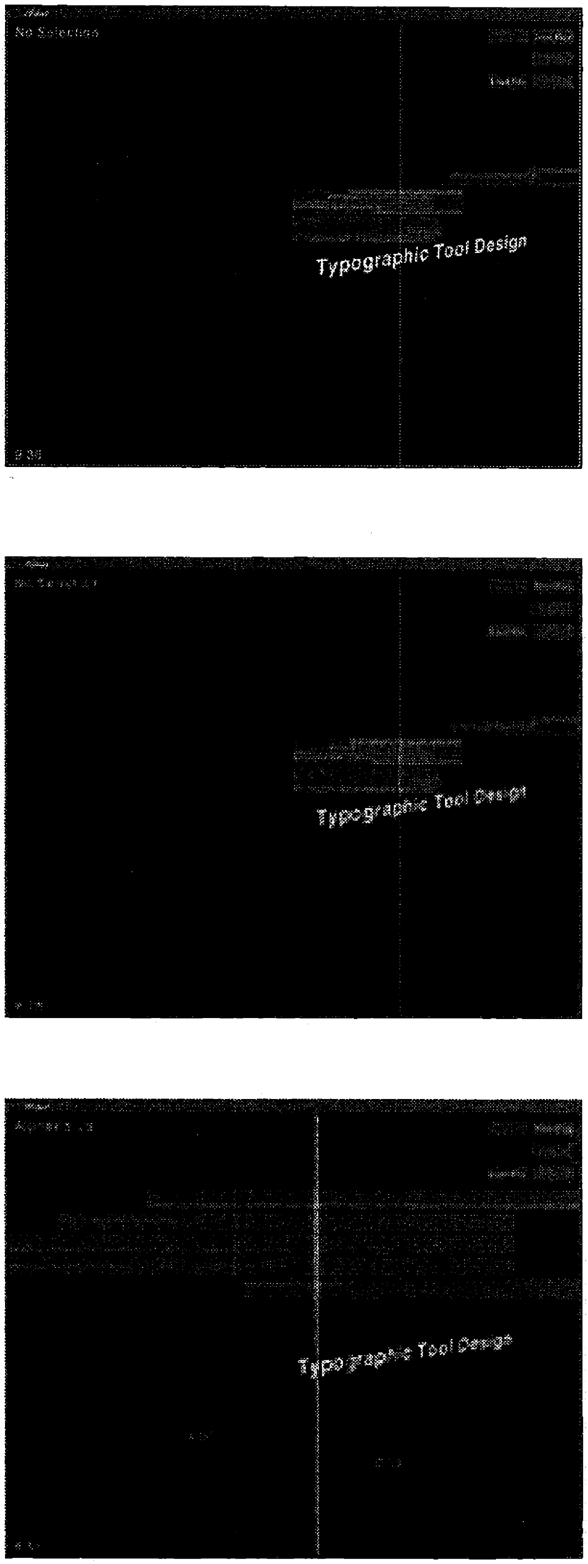


Figure 5-2: Sample screenshots from the experimental browser. Note the superposition of the browser on the presentation being edited, as well as the child nodes inside of the parent nodes.

members of the presentation on the vertical axis. One problem with this approach is that the timeline window takes up a great deal of screen real estate, making working in the space an act of juggling windows. Another problem is that it is very difficult to see what the effect of different actions are, and it is generally not possible to edit effects without having to use some sort of dialog box.

There are also several problems that are not related to the editing environment itself, but with the underlying structure of the program.. One limitation of Director is that the number of objects that are allowed in a presentation is limited - this makes it much more difficult to create temporal typography. This makes it necessary to do some complex programming in Lingo in order to create temporal typography. Unfortunately, as soon as you do this kind of programming, it is no longer possible to display the temporal typography piece on the timeline, resulting in a scripting language which cannot be edited using the timeline.

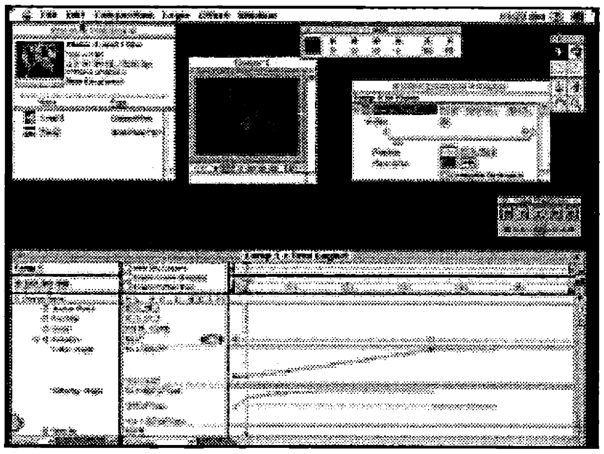


Figure 4-7: A sample screen from Adobe AfterEffects.

Another example of a system that can be used for temporal typography is Adobe AfterEffects. AfterEffects is a tool that is used for the creation of effects on digital video - effects such as lens flare, rotation, and scaling. While AfterEffects has the capability of doing almost all of the temporal typography effects necessary, it suffers from a workspace that is too complicated, like Director's. The main window which shows the effects over time cannot easily display the information for the manipulation of a single item of text: With multiple pieces of text, it becomes far too unwieldy and cumbersome [Adobe].

Thus, in general, there are two major problems with current browsing and editing tools. One of these is the need to switch back and forth between an editing context and a viewing context, generally by having them in two different windows, often making it impossible to work on the piece at the same size that it will appear in the presentation. The other is that there needs to be a good way of simultaneously representing both the geometric and computational structure of the presentation and the time structure.

The approach taken in attempting to resolve these problems was to try to take advantage of the technology that was available, namely fast three-dimensional graphics, and the ability to do high quality transparency. By taking advantage of the ability to see through multiple layers of the hierarchical structure at once, it makes it easier to maintain context. In addition, it is possible to manipulate the presentation while being able see the effect, unobscured by dialog boxes or other windows.

The basic design of the editor uses the horizontal axis of the screen to represent time in a relatively conventional manner. However, the vertical axis as well as the axis that enters the screen are used to represent the hierarchy of nodes in the presentation. While traditionally this would result in objects blocking each other, transparency was used to help alleviate this problem, as well as other problems.

Transparency was used to reduce the problem of obscuring in two ways. The first way that transparency was used was to superimpose the editor on top of the presentation being worked on, much like the heads up displays that are used in military aircraft. This makes it possible to see both the presentation and its representation at the same time, saving both screen real estate. It also avoids the idea of having different editing and viewing contexts, making it easier to keep track of what is happening.

Transparency was also used was in making it possible to see multiple layers of the browser representing different layers of hierarchy at the same time. Each plan or visual technique is represented as a translucent rectangle - tinted blue if it is a plan, tinted red if it is a visual technique. Each plan or visual technique is represented at a different height on the vertical axis. If a plan or visual technique has children, the children are placed "behind" the parent - in other words, further away from the camera. Since the rectangles are transparent, it is possible to see through many levels of the hierarchy - at once seeing the individual leaf nodes, and the top-level structure.

One problem with this representation is that plans and visual techniques do not necessarily have a time or duration - their duration is based on that of their children. So when determining the representation of an object, it depends both on the object's own duration, but also those of its children. This requires the implementation of a tree traversal in order to express the values of the all the nodes in the global time scale.

The representation allows the user to move the camera in and out of the presentation, and pan the camera left and right, or up and down. By zooming in, the user can take a more detailed look at an individual plan or visual technique. It is also possible to turn off layers of the view, so it is possible to view the lower levels of the hierarchy without having to view the higher layers.

When a node is selected by clicking on it, the vertical axis serves a double purpose. Vertical lines appear marking the beginning and end of the visual technique, and horizontal markers appear on these lines which represent the beginning and end value of the property being manipulated. Thus, the vertical axis is transformed to represent the dimension that is manipulated by the particular visual technique selected. By dragging the time markers back and forth, it is possible to change the duration and time of the visual technique, and by dragging the value markers up and down, it is possible to change the effect that the visual technique has.

In sum, the editor design was able to make it much easier to determine what was going on in a specific presentation. However, there was still an overwhelming amount of data, which still eventually would grow out of hand. In addition, it was still not very intuitive to edit temporal typography presentations. The next set of experiments attempted to resolve that problem.

5.2 Input Devices

TTT Poem

The TTT Poem was a piece that examined the possibility of using electromagnetic field sensors as an

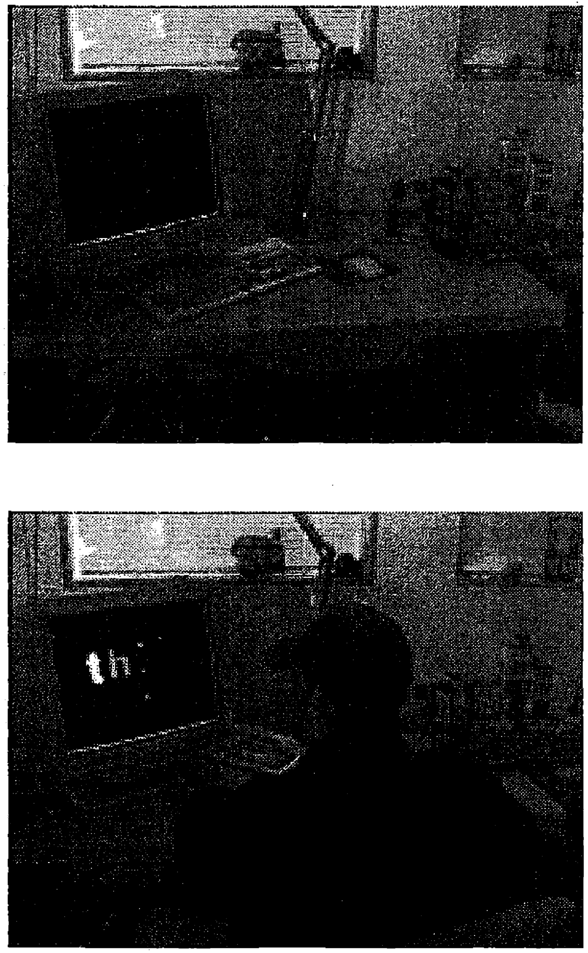


Figure 5-3: Sample screenshots from the TTT Poem. As the user moves their hands over the desk, the properties of the text change.

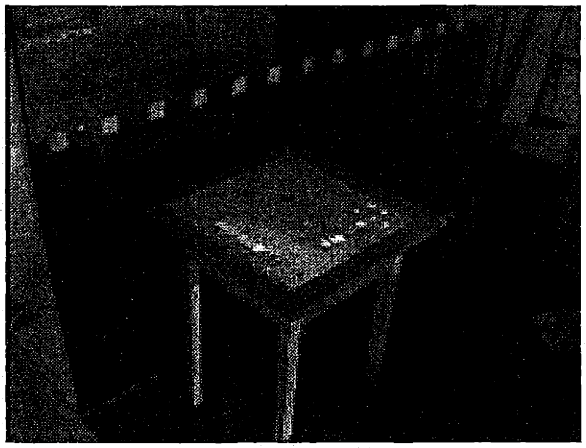
input device. The design consisted of a set of four receivers for the electromagnetic field sensors on the underside of the table, and a transmitter that the user sat on. As the user moved their hands over the table, each sensor detects the strength of the field emanating from their hands, and thus how far away a hand is from the sensor. For example, as a user moves their hand over the sensor, the different letters in the word "things" fades in and out.

By making the relationship between hand motion and the behavior of the object on the screen much more direct, users were able to have a much more intuitive feel over what changes they are making to the presentation. Instead of setting beginning and end points, and then viewing playback, the user was able to almost directly manipulate what is happening.

Fable Table

The Fable Table, a collaborative work with Sawad Brooks, uses technology similar to the TTT Poem, in that it uses electromagnetic field sensors as the primary input device. However, while the TTT Poem essentially keeps the traditional monitor-keyboard/mouse paradigm, except by substituting field sensors for the keyboard and the mouse, the Fable Table attempts to break that paradigm.

The Fable Table is a wood table, approximately two and a half feet square, which has a projector suspended above it, projecting an image onto the surface of the table. In addition to the projector there is also a small display that is embedded into the surface of the



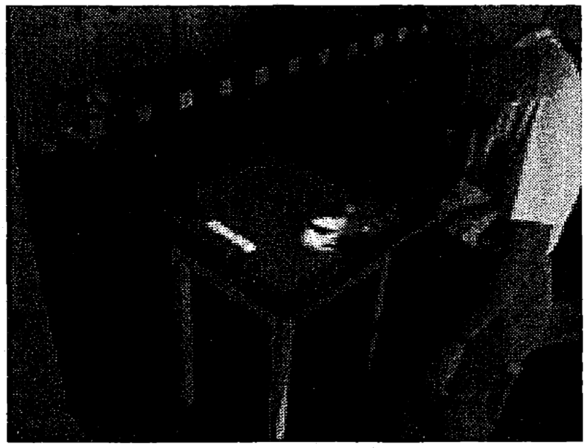


Figure 5-4. As the user places their hand over areas of the Fable Table, actions occur in those areas of the table

table. Electromagnetic field sensors are embedded in the table, and are able to sense the position of a hand in a three by three grid. The longer a person leaves their hand in a position, the higher the value of the variable associated with that space.

As different areas of the table are activated, different things happen. Objects that are on the table are displayed, and fade in and out. Sounds are played. Also, on the small display, words that are associated with the area appear, scrolling across the display. By moving their hands over the table, the user is able to compose a temporal typography piece.

The Fable Table, by placing the display surface on top of the surface used for control, gives a much more immediate sense of control, more so than even the TTT Poem. Also, because of the quality of the wood, and the projection, it appears as if the table itself is glowing, giving the piece a very different feel from any conventional display. By breaking away from the traditional user interface, usability is greatly enhanced.

Template Language/Design Automation

One of the largest problems with temporal typography authoring currently is the issue of dealing with the details of the temporal typography piece. In order to have a temporal typography piece appear right, it is necessary to tweak the timing, position, and all the other attributes. One way to approach the problem, as described earlier, is to make it easier to visualize all of the information. Another approach, however, is to reduce the amount of information, by automating low level design.

However, such automation is not a simple task. While it is very simple to just break things up into different visual expressions, and then allow the user to simply pick different expressions for each word, that would not create a satisfactory design. Each element of the design is affected by what precedes it and what follows it. In order for the design to work, each element must be able to adjust itself based on the surrounding elements.

In order to solve these problems, a template-based language was implemented. This system uses the idea of "intelligent" template that, based on information about the presentation and the other templates around them, adjust themselves in order to provide the best design.

The template language does this by using *fields* and *links*. Fields are values that represent either a property of the presentation, or a value that is set by either the user or another template in the presentation. This allows for information to be communicated between different templates.

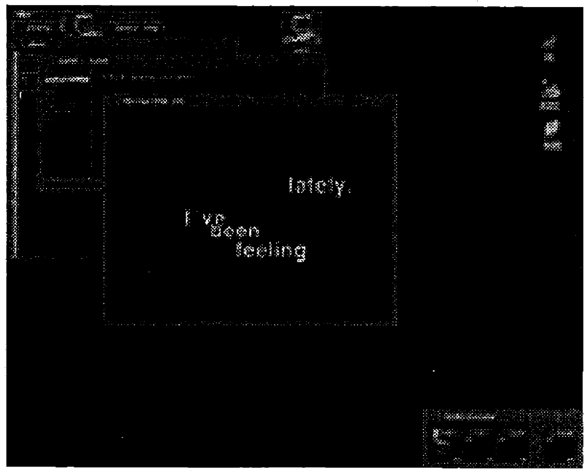
Fields are complemented by links, which connect a field to the value of a specific property in the presentation. For example, a field called "angriness" could be connected to color and size. As a user increased the angriness field, the object the template is attached to could turn red and grow larger.

By building templates which contain links to fields in other template in the presentation that store information about the various properties and qualities of the those templates, the templates are able to adjust to the templates around them, and perform the tweaking that is necessary to produce a seamless presentation

TypoTalk

The TypoTalk system uses temporal typography as the communications medium for intercomputer messaging between multiple users, similar to a chat system. Because of its need to function in nearly real-time, it was a good test bed for the template-based automated design system.

TypoTalk is used by typing a message in the text window, and selecting a template from a list of predetermined general templates. Then, the text that is typed is broken



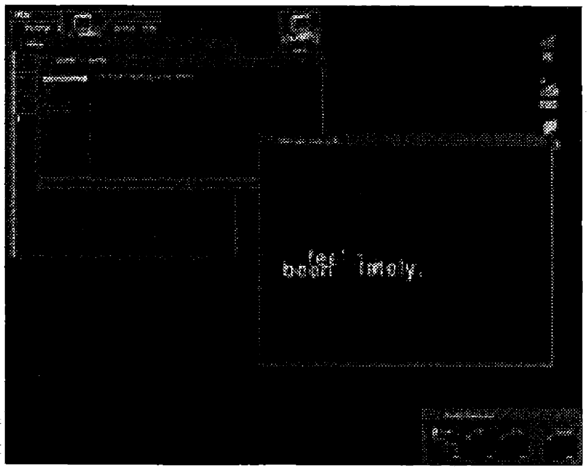


Figure 5-5: Sample screenshots from the TypoTalk system. Messages to be sent are typed into the dialog box, and when they are received, are displayed in pop-up windows by the server.

up into words, and each word has the general template style applied to it.

In addition to using the generic templates, a dictionary feature was added to TypoTalk. Each word in the message is checked in a dictionary to see if there is a more specific temporal typography template associated with it. For example, the word "blue" has a template that automatically turns that word the color blue. Once these templates are gathered for each word, the templates are translated into lower level scripts and sent across the network to a server on the target machine, which then displays it in a window. The use of a feature such as this dictionary feature is a prelude to the possibility of using some kind of automatic language parser, which would be able to automatically understand the meaning of text and apply a temporal typography style accordingly (see Appendix B).

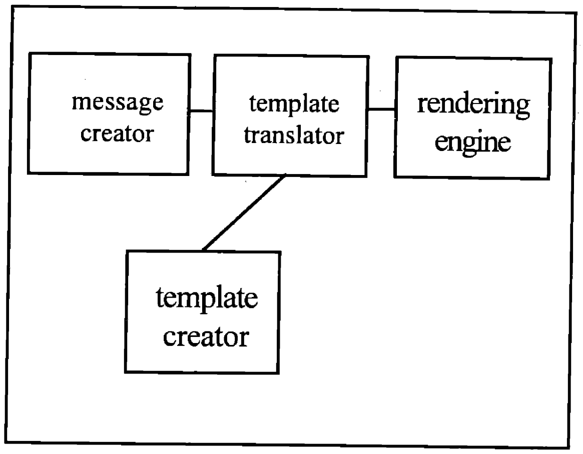


Figure 5-6. A diagram showing the major components of the TypoTalk architecture.

Implementation

The framework of the TypoTalk system is comprised of four major segments:

- The rendering engine draws the text at the receiving end of the message. This was implemented using the rendering framework described in Chapter 4, modified to include a socket interface that allowed clients to connect to the machine.

- The template translator is responsible for taking a template script generated by the user and a template file generated by a template creator, and generating output that is used by the rendering engine. The translator is implemented in Java, and connects to the renderer via a network socket.

- The message creator is the interface that is used to create the temporal typography message to be sent to other users. The user types in a piece of text that they wish to send, and then picks a template that they wish to use with the text. The message is then sent to the template translator.

- The template creator is used to create templates that are used in the template translator and message creator to generate text. Currently, this is not implemented - templates must be created manually...

In order to deliver a message, the following sequence of events occurs:

1. The user picks a template style, and types in a message to be sent, and presses the send button.

2. The template translator breaks up the message into words. For each word, it uses the template style picked by the user. In addition, the translator checks to see if any of the words are in its dictionary. If they are, the translator adds the styles associated with them to the list of styles being used by the template.

3. The translator takes the list of styles, and generates a script file. It connects to the rendering engine, and sends it the file.

4. The rendering engine parses the file and displays the message.

6. Conclusion

Temporal typography has great potential as a new expressive medium, allowing a level of interactivity and feeling that has only been possible in spoken communications. This thesis has demonstrated various different methods that can make temporal typography authoring a much simpler process. Hopefully, this will result in temporal typography becoming less of an experimental curiosity. As the practicality of using temporal typography increases, the difficulty of creating expressive, clear messages using the computer will hopefully disappear.

By performing the experiments and creating the tools outlined in this thesis, the research attempted to outline the difficulties involved in temporal typography authoring, and suggest ways in which these difficulties could be solved. In particular, two major approaches were conceived - those of improved interfaces, and those of automated design. Most approaches towards improving temporal typography can be categorized as one of these two broad approaches.

The experiments with an improved browser/editor interface attempted to take advantage of high quality graphics in order to improve the traditional class browser/ scoring interface. The use of transparency and three-dimensional graphics proved to be able to show a greater amount of information, as well as being able to see local information while viewing from a global context.

Using electromagnetic field sensors demonstrated the possibilities of using hand position and gestures as a way of creating temporal typography. Systems using such controls provide the user with a much higher degree of control over the output of a system, with more intuitive feel. They make using a temporal typography tool more like playing an instrument than programming.

Automatic design is a powerful way of creating temporal typography pieces that need to be created in nearly real time, such as in the TypoTalk system that was developed. It significantly reduces the amount of time necessary to create a piece, while simultaneously maintaining most of the visual quality.

Future work

Due to the scope of the issues involved, it was only possible to perform a very broad investigation. There is need for deeper research into the different approaches, as well as looking at approaches that were not examined.

For example, because of complexity and time constraints, it was not possible to develop a fully functional browser and editor based on the research performed. In order to fully evaluate a design of this nature, it is necessary for it to be used in practice. While the use of transparency and three-dimensional graphics in order to increase the amount of information which could be visualized should prove to be a useful approach, there has not been any kind of evaluation of exactly how much benefit would be obtained. The human mind has some limitations on exactly how much information it can absorb at one time, and it may be that limits in how much information can be comprehended at one time have nearly been reached.

In experimenting with input devices, the extent of the research performed in this thesis was only to show that such an approach was viable, and was much more intuitive than using conventional approaches. There is still much more research to be conducted. In particular, much work has been done with examining the use of gesture in human communication. If that research could be combined with that of temporal typography, a system could be envisioned in which a user would not have to consciously create temporal typography actions, but the computer could automatically recognize gestures and associate them with temporal typography.

Design automation is at the same time the most obvious solution to authoring problems, as well as the most difficult solution. The template system developed in this thesis represents only a crude example of what could be done by using design automation. Approaches using much more complex grammatical, or multi-agent systems would almost certainly be much more flexible, and have much better results, in terms of the quality of output and ease of use.

Of course, these different approaches need not be applied exclusively. A very powerful system could be envisioned that combined some form of gesture recognition and language recognition along with agents. In it, a user could speak a message, and temporal typography would automatically appear, formatted properly.

While the research in this thesis deal only with temporal typography, the results can be applied to a much broader variety of applications. In the process of understanding how to author expressive temporal typography, insight is reached into ways of making the computer perceive human emotion and language. Temporal typography authoring can be seen as just one piece of the larger problem of computers expressing emotion.

Appendix A. Rendering Engine Scripting Language

Basic structure

The structure of the scripting language is basically a hierarchial tree, consisting of three types of nodes -*plans*, *items*, and *visual techniques*.

*Item*s are the simplest objects - they are simply the graphical object that is to be manipulated by the plans and visual techniques. Each of them provides a set of functions that can be used to manipulate various attributes (such as translation, rotation, scale) by the visual techniques.

A *plan* is a container for items that are to be acted on. So, for example, a plan usually contains several items such text or images. The reason that the plan is not the graphical item itself is so the plan can be used as a container for other objects without having its own graphical form.

A plan has four basic parts:

- The item that it contains (if any)
- A list of visual techniques that modify the attributes of the item.
- A list of subplans that can inherit attributes from the plan
- Several variable that affect the timing and time scale of a plan and its children (both visual techniques and subplans).

A visual techniques is basically an action that is performed on the plan. Based on some sort of key variable, such as time, or the volume of sound, it manipulates the attributes of an item. Each visual technique can have subordinate techniques, which inherit the scaling and timing of it's parent.

The actual structure of the program is very simple. Each frame, the program gathers the values of the key variable. It then does a recursive search through the various plans and visual techniques, finds which ones are active, executes the transformations, and redraws the screen.

The Ivshell Scripter

This section describes the usage of the rendering engine, called *Ivshell.* This program can be run on an SGI, running IRIX 5.3 or 6.2

```
Ivshell [-sound] [-pos <x> <y> <w> <h>] <filename>
```

<filename>   This is the script file to be used (usually .prs)

While running the program, there are various keyboard commands you can use.

p pauses and unpauses the program.
r resets the presentation (not always reliable!).

Script File Syntax

The script files that are used by the program are simple ASCII text files. The preferred extension for the names of these files is .prs, because there is an `emacs` editing mode automatically detects the extension.

The syntax of the script file is fairly simple. It is primarily based on the use of several key words. Basically, each plan must begin with the keyword Plan, and end with the keyword EndPlan. Similarly, each visual technique must begin with the keyword Vis, and end with the keyword EndVis.

To create a piece of text saying "Hello", you would type:

```
Plan
   Item: Text
   "Hello
   EndPlan
```

This creates a new plan, places a text item in the plan, with the string "Hello", and ends the plan.

Suppose you wanted to rotate the object around the Z axis. You would then type:

```
Plan
   Item: Text
   "Hello
   Vis
      Type: Rotate
         Keys: 0.0 10.0
       Vals: 0.0 360.0
       Axis: 2
       EndType
      EndVis
   EndPlan
```

Plans

The syntax of a plan is very simple. It consists of the following format

```
Plan
      Item: <type>
      ID: <id>
      Offset: <offset>
      Scale: <scale>
      <subplans and visual techniques>
      EndPlan
```

All arguments are optional except for the keywords Plan and EndPlan. Also, the arguments do not have to be in order. Offset shifts the timescale of the object, and scale scales the time for the presentation by a factor. (Note: the time is first multiplied by the scale, and then the offset is added to it)

Currently, there are 4 kinds of items: text, rectangles, images, and cameras. Here is the syntax for those items:

Text Item:

```
Item: Text
"Put what you want to say here
```

Rectangle Item:

```
Item: Rect
<width> <height>
```

Image Item:

```
Item: Image
<width> <height>
<filename>
```

The image file should be in SGI Iris format(.rgb).

A camera item:

```
Item: Camera
```

There should only be one of these in a presentation. Consistent results are not guaranteed otherwise!

Visual techniques

Here is the syntax for a basic visual technique:

```
Vis
      <type>
      ID: <id>
      Offset: <offset>
      Scale: <scale>
      <subordinate visual techniques>
      Persist: <persistence mode>
      Inter: <interpolation mode>
      Keytype: <key variable type>
      EndVis
```

As you can see, the syntax for a visual technique is very similar to the syntax for a plan. The only important thing is that ID must be set after type, due to the method of parsing. Persistence mode refers to the effect of the visual technique if it is not in the range specified by the keys:

always	The visual technique always takes effect, with the beginning value
n	No persistence. The visual technique has no effect outside of the key value range
p+	The visual technique persists afterwards, using the value begin val + delta val. This is the default setting for most visual techniques.
+p	Same as previous, except for key values lower than the range, and with the begin value.
`+p+`	Combination of the two previous persistence modes.

Interpolation mode refers to the method in which the key values are translated to the values used by the visual techniques.

linear	A one to one correspondence
cos	Follows a curve shaped like an inverse cosine curve - i.e. slow at the beginning, slow at the end.
quaddec	Quadratic decay. Fast at first, then slower later.
quadgrow	Quadratic growth. Slow at first, then faster later.

Key variable type refers to the type of variable that the visual technique is keyed to.

`time` - keyed to the time, in seconds, of the presentation `mousex` - keyed to the mouse's x position, in pixels from the bottom left.

`mousey` - keyed to the mouse's y position, in pixels from the bottom right.

Syntax for different types of visual techniques:

```
Type: Translate
   Axis: <0 1 2>
   Keys: <begin> <delta>
   Vals: <begin> <delta>
   EndType

Type: Rotate
   Axis: <0 1 2>
   Keys: <begin> <delta>
   Vals: <begin> <delta>
   EndType

Type: Alpha
   Keys: <begin> <delta>
   Vals: <begin> <delta>
   EndType

Type: Scale
   Keys: <begin> <delta>
   Vals: <begin> <delta>
   EndType

Type: TransAll
   Keys: <begin> <delta>
   Begin: <x> <y> <z>
   Delta: <x> <y> <z>
   EndType

Type: Stretch
   Axis: <0 1 2>
   Keys: <begin> <delta>
   Vals: <begin> <delta>
   EndType

Type: Color
   Keys: <begin> <delta>
   Begin: <r> <g> <b>
   End: <r> <g> <b>
   EndType

Type: Switch
   Key: <time>
   Dir: <0 1>
   EndType
```

```
Type: Bounce
   Keys: <begin> <delta>
   Floor: <height> <constant>
   Gravity: <gravity>
   Drag: <air resistance>
   EndType

Type: Align
   Key: <time>
   Align: <0 1 2 3>
   EndType

Type: Font
   Key: <time>
   Font: <font>
   EndType
```

In addition to these plans and visual techniques, there are a few other different keywords:

`Include: <filename>` - Includes another presentation file at that location. Not fully tested...

Script file overall format

There is a sample file in ../Data/test.pres

Text in < > are arguments.

All segments are optional except where noted.

```
Plan
      Item: <type of item: Text, etc.>
      ID: <id string - no spaces allowed>
      Plan
            <args for subplan>
      CopyPlan: <id string>   // add a subplan that is a copy of
                              // a prev. plan
      Offset: <key offset>
      Scale: <key scale>
      Vis   // Indicates the start of a visual technique
            Type: <type of vis> // Optional, but put first
                  <args for type>
                  EndType
            KeyType: <type of key: time, etc.>
            Persist: <type of persistence: always, p+, +p, +p+, n>
            Inter: <type of inter: lin, cos, quaddec, quadgrow>
            Offset: <amount of key offset>
```

```
          Scale: <amount of key scaling>
          ID: <id string>
          Vis   // subordinate vis. tech.
                <args for vis>
          CopyVis: <id string>    // Copy of prev. vis. tech.
          EndVis            // Required!
    EndPlan           // Required!

    //    Anything after // is a comment.  Can be put at ends of
    //    lines, or on their own lines.
```

Script File Emacs Mode

To use the editing mode put it into your .emacs file. It will automatically indent your file, to make it easier to detect syntax errors. Commands in pres-mode:

`Esc Ctrl-X`

Reindents the entire file

`Tab`

Reindents the current line.

`Ctrl-C Ctrl-C`

Comments out the marked area.

`Ctrl-C Ctrl-U`

Uncomments the marked area.

Appendix B. Template Language

A template consists of lists of two different types of objects: *fields* and *visual templates*. A field is a variable that can be set either by the user or by a controlling program.

Visual templates consist of three different types of elements: *links*, which connect fields to values in the visual technique, *parameters*, which set generic visual technique options (such as persistence, etc.), and *values*, which set properties specific to a particular type of visual technique. The syntax of the types of fields is as follows:

```
Template:
<name of template>
```

Must be followed by an `EndTemplate.`

```
Field:
<name of field> <min> <max> <default>
```

Min is the minimum value of the field, *max* is the maximum value of the field, *default* is the value that it is normally initialized to.

```
VisTemplate:
<Name of visual technique>
```

Must be followed by an `EndVisTemplate.`

```
Link:
<Visual technique field> <field> <scale> <offset>
```

Visual technique field refers to the name of a variable in the visual technique, *field* refers to the name of a field in its parent template, *scale* is the value the field's value will be multiplied, *offset* is the value that will be added to the field value.

```
Value:
<Line to include in visual technique "Type" section>
```

This paremeter essentially causes the same line to be directly inserted into the script being output by the template translator.

```
Param:
<Line to include in visual technique section after "EndType">
```

This paremeter essentially causes the same line to be directly inserted into the script being output by the template translator.

This is a typical template file that defines a single template called `Spin`.

```
Template:
Spin
Field:
Speed -100 100 0

Field:
Begin 0 100 0

Field:
Delta 0 100 0

VisTemplate:
Translate
Link:
BeginKey Begin 1.0 0.0

Link:
DeltaKey Delta 1.0 0.0

Link:
BeginVal Begin 0.0 0.0

Link:
DeltaVal Delta 1.0 0.0

EndVisTemplate

VisTemplate:
Rotate
Link:
BeginKey Begin 1.0 0.0

Link:
DeltaKey Delta 1.0 0.0

Link:
BeginVal Begin 0.0 0.0

Link:
DeltaVal Speed 360.0 0.0
EndVisTemplate

EndTemplate
```

References

Bass, S. Title sequence for the film *Psycho. 1960.*

Berkeley Systems. **You Don't Know Jack.** 1995.

Bruckman, A., Resnick, M. 1993. **Virtual Professional Community: Results from the MediaMOO Project.** Presented at the Third International Conference on Cyberspace in Austin, Texas.

Cho, P. **Dancing Letters: Movement and expression in typographic elements.** 1996. ACG Memo.

Cooper, M., et al. **Information Landscapes.** Frames. 1994

Ionesco, E., Massin, and Cohen. **The Bald Soprano.** Grove Press. 1956.

Ishizaki, S. **Typographic Performance: Continuous design solutions as emergent behaviors of active agents.** Ph.D. Thesis, Massachusetts Institute of Technology. 1996.

Maeda, John. **Flying Letters.** Digitalogue, Tokyo. 1996.

Thomas, F. Johnston, O. **Disney Animation: The Illusion of Life.** New York, Abbeville Press. 1981.

Tufte, E. R. **Envisioning Information**. Cheshire, Connecticut. Graphics Press. 1990.

Weitzman, L. **The Architecture of Information: Interpretation and presentation of information in dynamic environments.** Ph.D. Thesis, Massachusetts Institute of Technology. 1995.

Wong, W. **Principles of Form and Design.** New York, Van Nostrand Reinhold. 1993.

Wong, Y. **Temporal Typography: Characterization of time-varying typographic forms.** M.S.V.S. Thesis. Massachusetts Institute of Technology. 1995.