Expressive Typography

High Quality Dynamic and Responsive Typography in the Electronic Environment

by

David Small

Bachelor of Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 1987

Submitted to the Media Arts and Sciences Section in Partial Fulfillment for the Requirements of the Degree

Master of Science in Visual Studies
at the
Massachusetts Institute of Technology

February, 1990

Author _____
David Small
Media Arts and Sciences
January 19, 1990

Certified by _____
Muriel Cooper
Professor of Visual Studies
Thesis Supervisor

Accepted by _____
Stephen Benton
Department Committee for
Graduate Students

Expressive Typography

High Quality Dynamic and Responsive Typography in the Electronic Environment

by

David Small

Submitted to the Media Arts and Sciences Section on January 19, 1990 in Partial Fulfillment for the Requirements of the Degree of Master of Science in Visual Studies at the Massachusetts Institute of Technology

Abstract

This thesis develops a methodology for handling complex typographic information. Tools for creating and modifying complex typography, in both the static and dynamic cases, form a platform for greater expression. Dynamic text is created using a spring- based physical simulation model. Type can be animated with sound and voice. The quality of screen fonts is emphasized, and a scheme for real-time filtering of fonts is discussed. Simulation of pigments on a massively parallel computer enables complex simulation of pigment and paper based fonts. The thesis exists as a dynamic, interactive experiment.

Thesis Supervisor: Muriel Cooper
Title: Professor of Visual Studies

The work reported herein was supported in part by NYNEX, Hewlett Packard, IBM and Bitstream.

Acknowledgements

I would like to thank the following people:

Mike McKenna, the best friend a boy could have.

Muriel Cooper, for never tolerating the ugly, the incoherent, the poorly designed, or the kludgy.

Ron MacNeil, Patrick Purcell, and Walter Bender for their words of encouragement.

Russell Greenlee, Suguru Ishizaki, and Sylvain Morgain for showing me how to get it done.

Bob Sabiston, for teaching me how to program and for making the complex so simple.

Laura Robin and Ming Chen for their help and friendship.

Wave for showing me how not to fear the Connection Machine.

Anne Russell and Pascal Chesnais, for giving me the courage to go on, when all seemed hopeless.

Marie Crowley, for all those words that she

knows how to spell correctly.

Jacqueline Casey, for reminding me that designing is supposed to be fun, and for her fresh perspective on everything.

Mom and Dad, for putting up with me for so many years and for always being interested, even when I can't explain what it is that I do.

Sergio Canetti and Nathan Felde, of NYNEX, for their kind support and interest and for wanting me to do what I wanted to do.

Nicholas Negroponte and the Media Laboratory for giving me the opportunity to grow here.

Given 1. the waterfall

2. the illuminating gas,

one will ~~determine~~
we shall determine the conditions
?
for the instantaneous State of Rest (or allegorical appearance)
of a succession [of a group] of various facts
seeming to necessitate each other
under certain laws, in order to isolate the sign
the
of accordance between, on the one hand,
all ~~the~~ (?)
the State of Rest (capable of innumerable eccentricities)
and, on the other, a choice of Possibilities
authorized by these laws and also
determining them.[1]

Marcel Duchamp
Door of Given: 1. The Waterfall,
2. The Illuminating Gas. 1946-1966

Under the sea
Under the sea
Thats why its hotter
Under the water
Take it from me

Sebastian
The Little Mermaid
Walt Disney

Contents

Abstract
Acknowledgments
Quotes
Contents

1 Introduction
1.1 Overview 7
1.2 Motivation 9
1.3 Related Work 9
1.4 Example 11

2 Dynamics
2.1 Dynamic Simulation 13
2.2 Distorting Matrices 16
2.3 Springy Fonts 19

3 Sound
3.1 Sound and The Workstation 20
3.2 Keyboard Type Control 21
3.3 Voice Type Control 23

4 Quality
4.1 The Font Pipeline 24
4.2 Filtering on the Fly 25

5 Wet Fonts
5.1 The Connection Machine 27
5.2 Simulation of Pigment and Water 28
5.3 Type as Pigment - parrallel fonts 32

6 Conclusion
6.1 Conclusion 35
6.2 About this thesis 37

7 Appendices
7.1 Dynamics 40
7.2 Solving a 4x4 matrix 42
7.3 The MIDI Server 43
7.4 The Dispersion Algoritm 44
7.5 Parallel Display of Type 47

8 Bibliography 49

1 Introduction

1.1 Overview

Typography is used to express ideas, style and emotion. It is used to enhance the display of information, especially where spatial clues are important. This thesis looks into some of the issues raised by the use of typography in the electronic environment. The computer requires us to rethink much of what has been learned about type in the print domain. We will look at four interrelated topics: dynamics, sound, quality, and wet fonts. Each of these topics requires us to look at type in a different way. In doing so we are forced to develop a robust and flexible model for type, which allows us to have a greater range of typographic expression than was previously possible.

wet fonts is the term I am using to refer to the simulation of fonts as areas of ink and water responding to an active paper medium.

The computer enables us to handle the vast amount of computation that is now required. It can track multiple objects, render graphics, compute simulations and manufacture high-quality letterforms. Perhaps the most exciting property of the computer is the ability to create images that evolve and change over time.

The computer used is a Hewlett-Packard 835 equipped with a 1280x1024 32 bit display with hardware assisted 3-d rendering, including solid modeling and shading.

Dynamics refer to the movement of graphical objects on the CRT, usually in response to changing information, user interaction, or physical constraints. This thesis explores the use of dynamic simulation to animate

the shape of individual letterforms. By so doing we can create type which is responsive to physical events. Springy constraints allow the designer to modify letterforms quickly and intuitively.

Sound and sight together help us to navigate through complex environments. Audio cues can be an integral part of simulations, the user interface, and expressive typographic communication. The spoken word and the written word have evolved along a parallel course. Now we have the potential to bring these two aspects of language closer together.

The need to maintain the integrity of type on computer displays cannot be overstated. We increasingly consume information displayed on computers, and while many people have become accustomed to low resolution type and graphics, it is both counter-productive and technologically outdated. A fast-filtering method allows us to maintain the quality that is essential to typographic communication while retaining the greatest flexibility for the designer.

The use of the Connection Machine, a massively parallel supercomputer, provides us with a radically different way of thinking about the nature of graphics. It is possible to think of the screen, not merely as a canvas on which images can be pasted, but as an active, computationally rich medium. For example, we can simulate an active paper by assigning an entire processor to each pixel on the screen. This allows us to think of type less as a pure abstraction of shape, and more as an area of changing

pigment density, which can change and evolve over time.

1.2 Motivation

The interface to the computer will be dynamic, flexible, personalized, and responsive. Designing for that kind of rich environment is much more complex than for a flat page of paper. Design is both an analytical and an exploratory, playful activity. The computer should be able to support a rich, unconstrained play environment without sacrificing ease-of-use. Access to the appropriate level of detail should be available without difficulty.

This thesis explores new territory in the use and representation of type. More importantly, it lays the foundation on which others can explore on their own. This thesis tries, not only communicate what has been done, but to explore the nature of that communication and ways in which the computer could be used to enhance it.

1.3 Related Work

This thesis draws from a wide variety of research topics - typography, dynamic simulation, music and parallel computing. This meant synthesizing material from a wide variety of sources.

Avi Naiman's work on rectangular convolution of fonts provided the basis for a fast-filtering algorithm. Using his work, and work

Avi Naiman, *Rectangular Convolution for Fast-filtering of Characters.*

done in the VLW by Russell Greenlee we are now able to filter characters at the rate of six per second. Naiman's algorithm breaks up the character into smaller parts which are easy to filter. Work by Walter Bender and others in the Electronic Publishing Group, MIT Media Lab, on the evaluation of typefaces and the combined influence of design and anti-aliasing on legibility was very important in defining the type model that was used. He showed that by combining automatic filtering techniques with adjustments made by a type designer a superior display typeface could be made.

Walter Bender, et. al. *CRT Typeface Design and Evaluation.*

Jane Wilhelm's paper on dynamics provided a good overview of the steps involved in creating a dynamic simulation. Work done in the Computer Graphics and Animation Group, MIT Media Lab by David Zeltzer, Mike McKenna and Peter Schröder on collision detection, springy constraints and everyday physics was essential to creating a simulation that worked accurately. Much of their work is covered in their recent Masters' theses.

Jane Wilhelm, *Dynamics for Everyone.*

Mike McKenna, *A Dynamic Model of Locomotion fro Computer Animation.*

Peter Schroeder, *The Virtual Erector Set.*

Work done in the Speech Group, MIT Media Lab, especially by Janet Cahn, on the emotional qualities of speech demonstrates that distinct emotional states can be determined from voicing alone.

Janet Cahn, *Generating Expression in Synthesized Speech.*

In addition to the work done in the technological community, there is of course a long and rich history of experimental typography from the design community. Although not often put in quantitative terms, there is a great deal to be learned from this work. One such example of

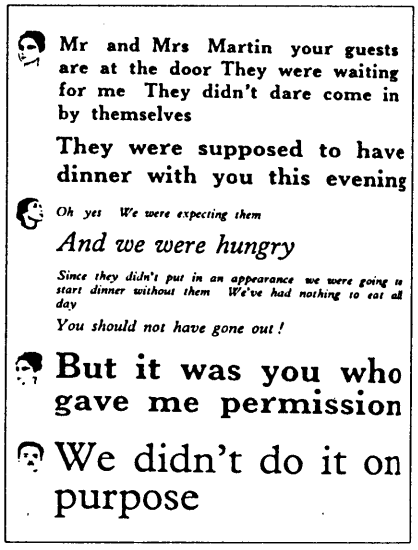
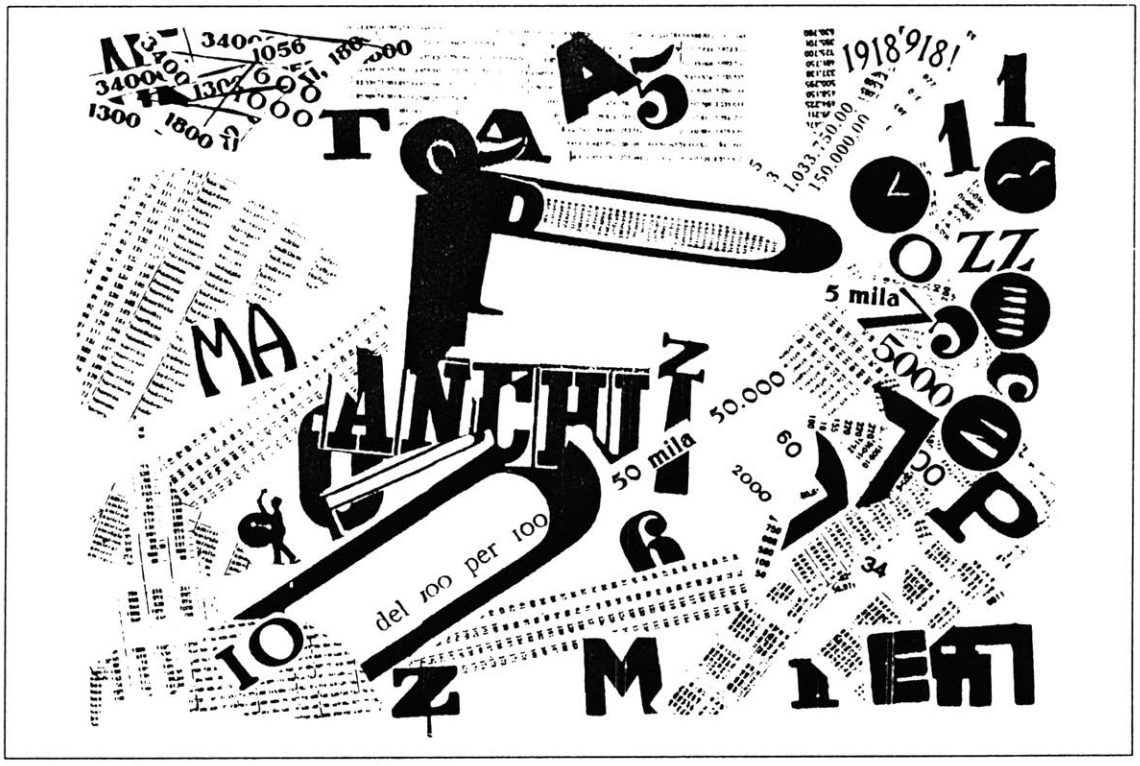


Figure 1. Eugene Ionesco and Massin. *The Bald Soprano.*

typography that imitates the expressiveness of the spoken word is The Bald Soprano, a play written by Eugene Ionesco and designed by Massin. In *Les Mots en Liberte* by F.T. Marinetti the form embodies the explosive and revolutionary nature of Futurism.

This work depends heavily on software written in the Visible Language Workshop and the Media Laboratory. Bob Sabiston's window system, Russell Greenlee and Curtis Eubanks tools for Bitstream fonts, Eric MacDonald's sound server, and work by Shaun Kaneshiro and Tim Kukulski which greatly simplified the complexities involved in working with type.

Figure 2. F.T.Marinetti. *Les Mots en liberte futuristes.* 1919



1.4 Example

In order to illustrate the various typographic tools that have been created, the pieces are integrated in an on-line example of the work. The result is both an explanation and an example of the use of dynamic and responsive type. In addition it will serve as an environment in which designers can sit down and explore the different tools.

The final component of this thesis is an electronic manuscript in which text is supplemented by working, interactive demonstrations. The reader of the electronic thesis will have access to a working environment in which ideas can be sketched out and tested. This sort of document or manuscript will evolve over time, as readers add their own notes and examples. I hope that it will be a valid model for how typography can begin to be used in electronic texts.

2 Dynamics

2.1 Dynamic Simulation

Dynamic simulations have often been used to demonstrate physical properties or make realistic animation. We are using dynamics to control graphics which have no real-world correlates. In doing so, we create something which is both realistic and abstract. For example, the word breakdown could break into pieces. It is important that the computation is fast enough that the user can interact with the simulation in close to real time. Direct manipulation is then easy, because the computation matches our expectations.

See Jane Wilhelms, *Dynamics for Everyone,* Fourth USENIX Computer Graphics Workshop, for a complete treatment of the physics involved.

The study of the physics of bodies in motion is called dynamics. Forces act on bodies, causing accelerations. Given accelerations over time, it is possible to compute a body's velocity and position. This computation involves solving the Newton's Second Law,

$$f = m\,a \qquad \text{or} \qquad f = m\,dv/dt$$

where the force applied to a body equals the mass times the acceleration, and the equations of motion :

$$\frac{\partial v}{\partial t} = a \quad \text{or} \quad v = \int_t^0 a \qquad\qquad \frac{\partial p}{\partial t} = v \quad \text{or} \quad p = \int_t^0 v$$

Although it is possible to solve the differential equation for very simple cases, in general one must use a numerical integration method to solve for the velocities and positions from the acceleration, over time. This is especially true if the bodies are subject to real time user control.

The numerical integration method works by breaking down the solution into many small discrete time steps. All the forces acting on each body are computed and summed, the accelerations are solved, and then new values are computed for the bodys' velocity and position. As long as the accelerations or velocities do not become large, this will give an accurate approximate solution. The forces acting on a body are gravity, spring forces, and collision reactions. The force of gravity is simply the body's mass times a constant *g.* The force exerted by a spring is:

$$k * (\text{length} - \text{rest length})$$

where *k* is the spring constant, *length* is the distance between the two endpoints of the spring, and the *rest length* is the distance between the endpoints of the spring when it exerts no force.

When the mass collides with the floor, the component of the velocity perpendicular to the floor is scaled by the coefficient of restitution (σ) and reflected back. This is due to the conservation of momentum.

$$v_{y \text{ after collision}} = - v_{y \text{ before collision}} * \sigma$$

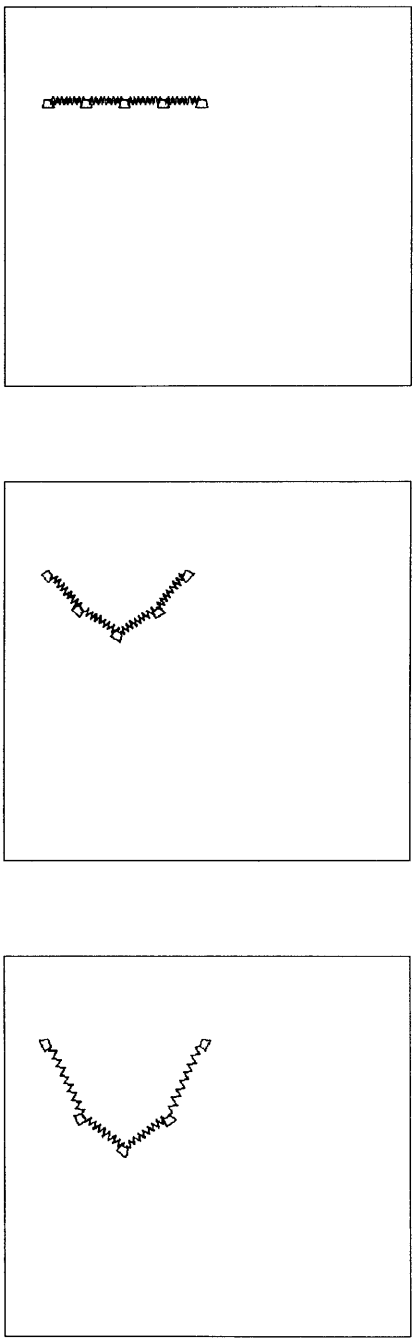


Figure 3. Bodies connected by springs falling.

Another force is due to damping. This force acts to slow down the bodies in motion, creating a "viscous" medium. The force it exerts is equal to the inverse of the velocity times the damping constant, b:

$$F_{new} = F_{old} - Velocity * b$$

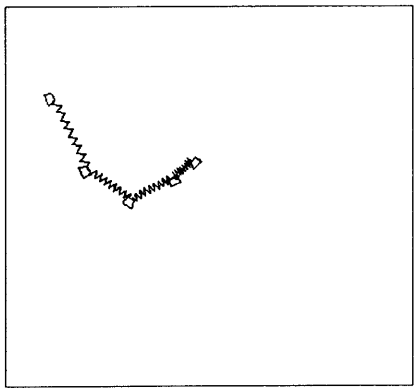
Given the total force acting on each body, it is possible to compute the instantaneous acceleration of each body. For some small amount of time Δt it is possible to compute new velocities and positions.

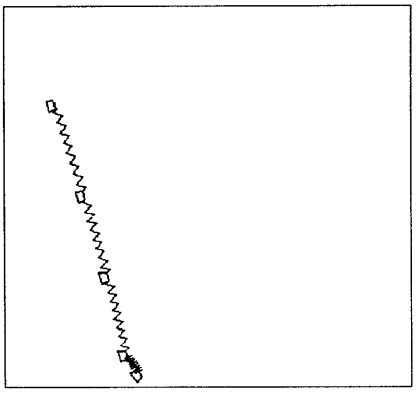
$$A = F \div Mass$$

$$V_{new} = V_{old} + A * \Delta t$$

$$P_{new} = P_{old} + V*\Delta t + 0.5*A*\Delta t^2$$

This is the essence of the dynamic simulation. It is possible to create any number of masses and springs and to connect them in any manner. The user can interactively modify any of the parameters of the simulation (spring constants, masses, gravity, damping, etc.) In addition, it is possible to grab masses with the stylus and move them around. There are two ways to move a mass. One can pick up a mass and pull it lightly; when the stylus is released the mass will then move according to the simulation. If however the user presses down harder, the mass will "stick" to that location. A clicking sound is made to indicate that the mass has





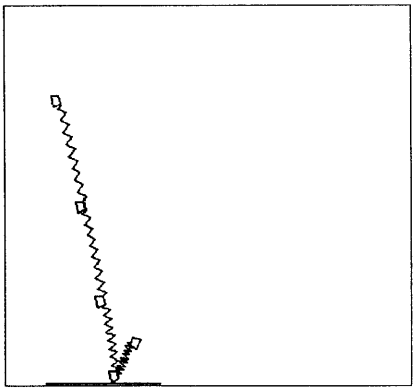


Figure 4. Collision with floor and collision response.

been tacked down. Also, when a mass collides with the floor, a clanking sound is made. The greater the mass's velocity, the louder the clanking sound. The integration of graphics and sound is very important to interactive simulations, and is discussed in greater detail in chapter three.

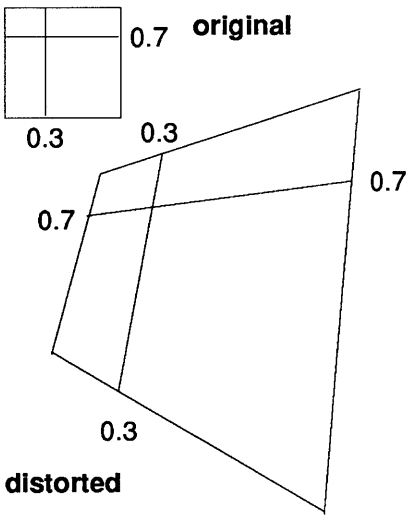


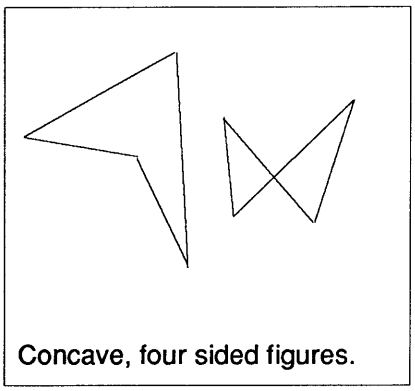
Figure 5. Bilinear interpolation method.

2.2 Distorting Matrices

In order to connect typographic objects to the dynamic simulation it is necessary to be able to distort the type. Given any four sided figure, one must be able to distort the character such that its em-box conforms to the figure. This could be accomplished by performing a bilinear interpolation for each point in the character. Thus if a point's coordinates are (0.3, 0.7), the new point would lie on the intersection of the line from three-tenths of the way along the bottom of the figure to three-tenths of the way along the top and the line seven-tenths of the way along the left edge to seven-tenths of the way along the right edge. This operation would have to be performed on every point in the character.

The em-box is the bounding box of a capitol M. This is commonly used as a scale of reference for the typeface.

Another way to solve this problem is to create a four by four matrix which transforms points in the original space into the distorted space. The matrix would only have to be calculated once and every point in the character can then simply be multiplied by the matrix. This is implemented to take advantage of a hardware matrix solver for displayed graphics. We need



Concave, four sided figures.

Figure 6. For some figures, the solution for the matrix **M** is undefined.

to compute the matrix **M** such that the points of the unit square [(0,0), (0,1), (1,1), (1,0)] when multiplied by **M** give the distorted figure [($P1_x$,$P1_y$), ($P2_x$,$P2_y$), ($P3_x$,$P3_y$), ($P4_x$,$P4_y$)]. We have sixteen unknowns and eight equations to be solved in terms of eight variables. However, because we are only dealing with two dimensional points, some of the unknowns in the matrix can be set to zero.

$$\mathbf{M} = \begin{bmatrix} a & b & 0 & d \\ e & f & 0 & h \\ 0 & 0 & 0 & 0 \\ m & n & 0 & p \end{bmatrix}$$

Now we only have nine unknowns and eight equations. If we set the homogeneous coordinate (p) to one we can now solve the equations. We must be constrained, however, to convex four-sided figures.

The problem breaks down into four separable matrices; translation, rotation, shear, and perspective transformations. It was then possible to solve for these matrices and then multiply them to get the matrix **M.**

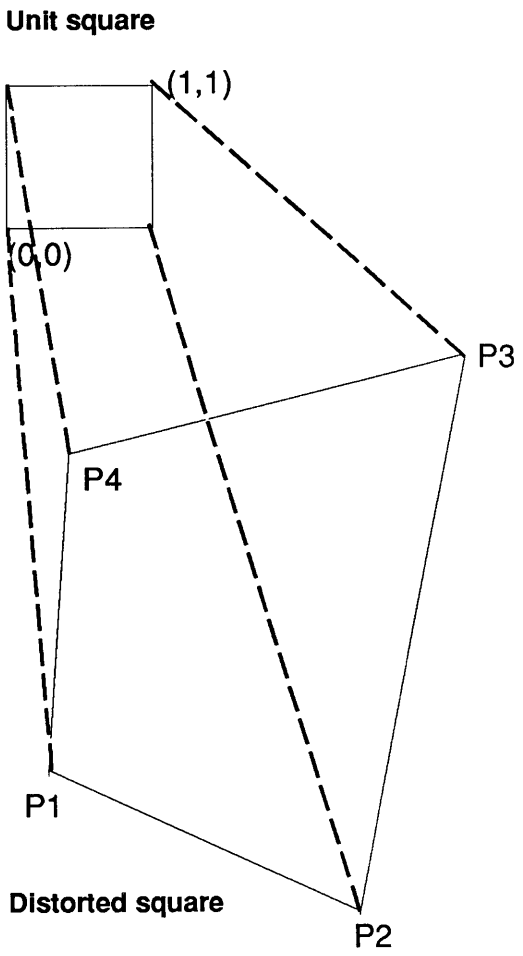


Figure 7. The matrix **M** transforms the unit square to the desired figure, given the four points P1 to P4.

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ P1_x & P1_y & 0 & 1 \end{bmatrix}$$

$$\mathbf{Sc} = \begin{bmatrix} P2_x - P1_x & 0 & 0 & 0 \\ 0 & P4_y - P1_y & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$shear_x = \frac{P4_x - P1_x}{P2_x - P1_x}$$

$$shear_y = \frac{P2_y - P1_y}{P4_y - P1_y}$$

$$\mathbf{Sh} = \begin{bmatrix} 1 & shear_x & 0 & 0 \\ shear_y & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$h_x = \frac{\frac{P3_x - P1_x}{P2_x - P1_x} - \frac{shear_x \cdot (P3_y - P1_y)}{P4_y - P1_y}}{1 - shear_x \cdot shear_y}$$

$$h_y = \frac{P3_y - P1_y}{P4_y - P1_y} - shear_y \cdot h_x$$

$$\mathbf{P} = \begin{bmatrix} \frac{1 - h_y}{h_x + h_y - 1} + 1 & 0 & 0 & \frac{1 - h_y}{h_x + h_y - 1} \\ 0 & \frac{1 - h_x}{h_x + h_y - 1} + 1 & 0 & \frac{1 - h_x}{h_x + h_y - 1} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{M} = \mathbf{P} \times \mathbf{Sh} \times \mathbf{Sc} \times \mathbf{T}$$

2.3 Springy Fonts

Once the dynamic simulation works and it is possible to distort characters, one can combine these two techniques to create interactive typographic animation. A network of springs is designed such that they define a four- sided figure for each character. Two springs form the space in between characters. The springs form the em-box of the character and the rest length of the connecting springs is set to the correct spacing. The dynamic simulator is used to determine the positions of each corner. Then, given the four corner points, a matrix **M** is found. This matrix is concatenated to the stack of matrices in the graphics pipeline. Then the character is drawn as if it were at the origin and on the scale such that it has an em-box one unit wide. The resulting image automatically conforms to the shape of the springs. This is done for each character in the network.

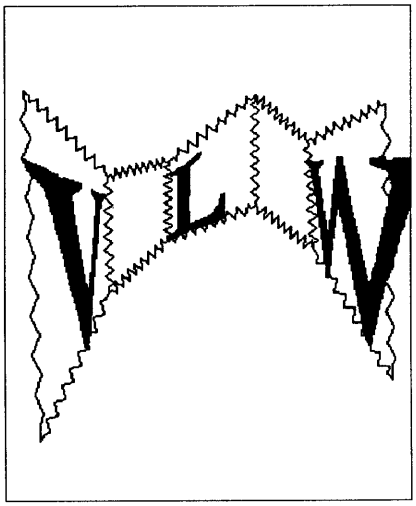


Figure 8. This figure illustrates a network of springs used as letterform constraints.

See Alvy Ray Smith *The Viewing Transformation*, for discussion of the view pipeline.

By selectively clamping and positioning some points, while leaving other points free, it is possible to constrain the shape of a word easily and interactively. In addition, it is possible to create animations of words. Because dynamic simulation is used, the animation tends to look as if it is based on real physical objects, despite the fact that the letterforms themselves are quite simple. Words are abstractions, yet the form words take can suggest images to the reader. The computer enables us to make moving images with words.

3 Sound

3.1 Sound and the Workstation

Sound can and should be an integral part of the computer interface. People can process sound information at the same time that they are absorbing visual information. In fact, it is possible to pay attention to many completely unrelated streams of information in this way (watch MTV for a couple of hours). Because computer hardware engineers for the most part have neglected sound (the exception being NeXT and, to some extent, Apple), it can often be quite difficult to have even the simplest sound control. We have chosen MIDI as our means of interfacing with musical instruments and samplers.

MIDI - Musical Instrument Digital Interface, See *MIDI Specification 1.0* for a detailed decription of the interface. *C programming for MIDI* by Jim Conger is a good overview of computer interfacing for MIDI.

Two-way MIDI communication between the computer and musical devices provides a rich environment for exploring the integration of sound and graphics. The communication is two-way because we not only want graphics to be able to create or initiate sounds, we also want to be able to shape graphics with musical instruments.

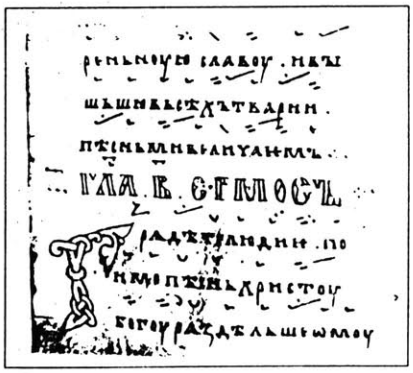
MIDI has become a de-facto standard in the music industry and a wide variety of musical devices can be controlled with MIDI. It is fast enough so that there are potentially very short lags between when the computer triggers a

sound and when it is heard. We are currently using MIDI as a way of controlling both a digital sampler (an AKAI 950) and a keyboard (for user input). In order for the computer to communicate with MIDI we need an interface that will allow the computer to reads and write MIDI data. This interface is discussed in detail in Appendix 7.3. Once a platform is established for computer interaction with musical instruments we can perform experiments which link graphics and sounds.

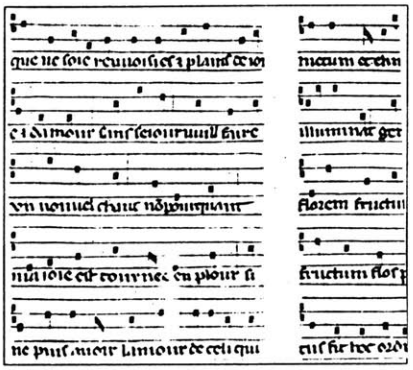
3.2 Keyboard Type Control

There are well established conventions which have allowed people to represent music (or sound) with graphics. We can look at medieval musical scores which represented some of the music but still left much unspecified. This worked because the music was well known throughout the culture. Perhaps also people in non-literate societies were more able to remember and embellish simple scores. As music became more sophisticated, the conventions became stricter. This reduced confusion, but also required that the amount and kind of information be limited. Starting in the twentieth century, people began to experiment with other ways of representing a wider variety of sounds. (See Masens' score for Duchamp's Band).

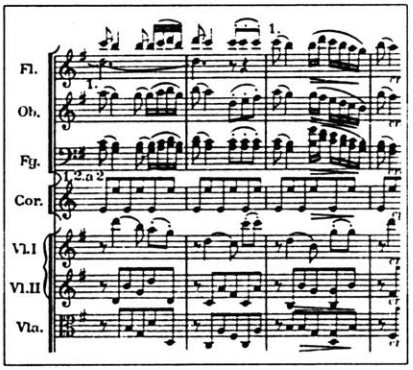
What we have done is allow the manipulation of graphics, directly by playing musical instruments. The keyboard sends information when a key is pressed (note, channel, key



A



B



C

Figure 9 (A,B,C). Three examples of musical scores. A is from the Hirmologium of the Codex Monasterii Chiliandarci, 308 AD, B is from a collection of thirteenth century French motets, and C is from Beethoven's Ninth Symphony.

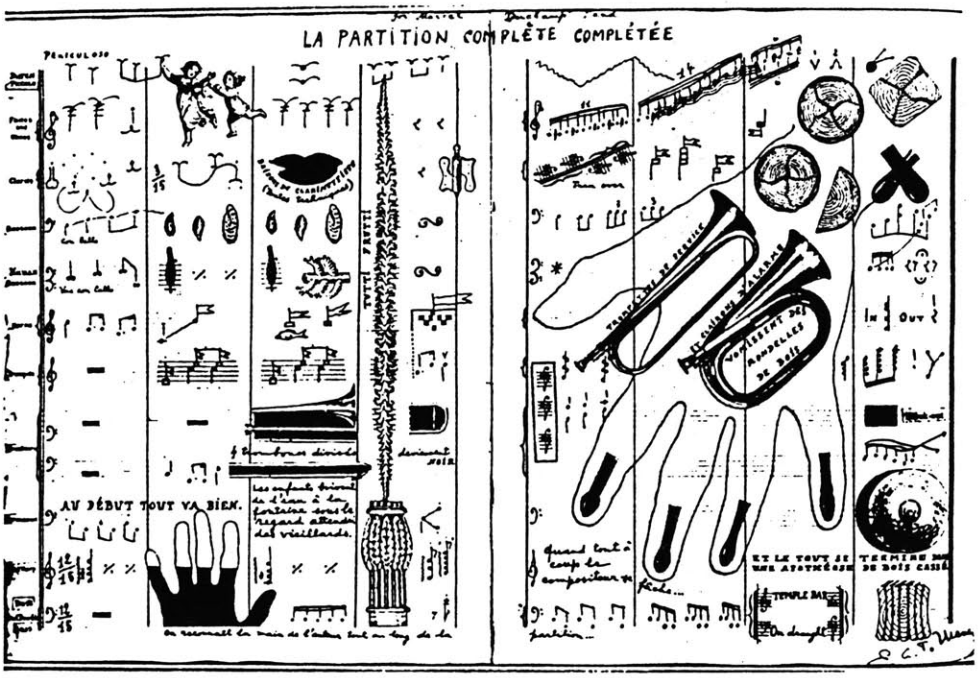


Figure 10. E.L.T.Mesens. *The Complete Score for Marcel Duchamp's Band Completed.* 1945

velocity), after it is pressed (aftertouch) and when it is let up. We can connect the information from a particular key on the keyboard (for example middle C) to a dynamic graphic (say the letter C). Initial velocity and aftertouch can be used to control the size of the graphic.

Although this experiment is very limited, it allows us to begin to think of type and dynamic graphics as being performances of a sort. This is certainly not a new idea (The Bald Soprano, by Ionesco and designed by Massin being a wonderful example) however the integration of elements by the computer gives us the opportunity, not only to automate parts of the process, but to have much finer control and perhaps most importantly to add the element of time to a typographic piece.

3.3 Voice Type Control

Of course there are many aural experiences besides music and perhaps the most interesting of these for our purposes is speech. If speech is digitized and analyzed it is possible to get some information about the aural qualities of the speech. This information can be linked to graphics by using the same sort of paradigms described above.

Because our ability to analyze speech is currently very limited, it was decided to try and link a single sound property (volume) to a single typographic element (point size). The cry of a baby was recorded and digitized, and then broken down into a graph of energy over time. The energy (or volume) of the cry was then used to control the size of as string of letters ("Aagh") in real time. The result was a visual representation of the sound of the baby crying. When seen in conjunction with the sound it gives the "impression" of the cry.

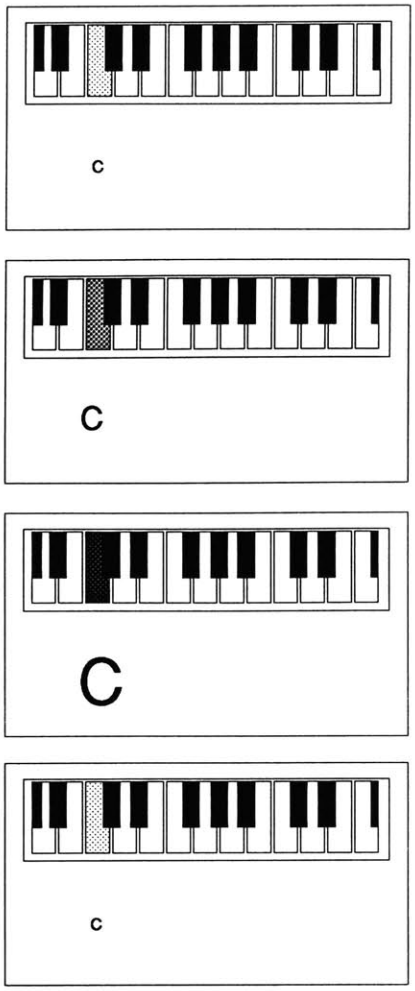


Figure 11. The letter C being scaled by pressing the note C on the keyboard.

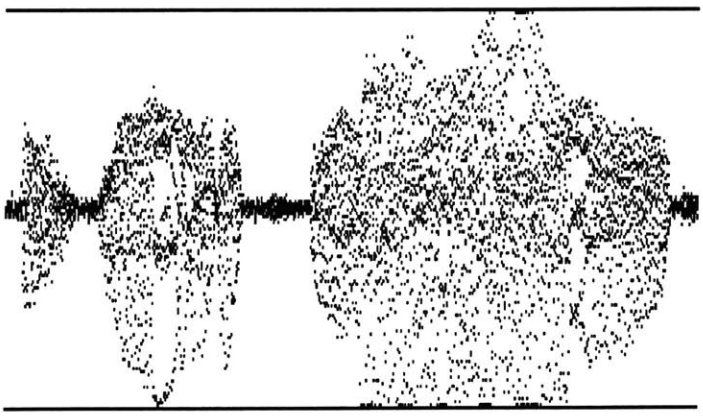


Figure12. Digitized cry.

Figure 13. Sequence of cry images.

4 Quality

4.1 The Font Pipeline

One of the goals of this research is to provide more flexibility without sacrificing quality. By using efficient filtering techniques, it is possible to create anti-aliased fonts on the fly. In the past, designers have depended on WYSIWIG (What You See Is What You Get) as a model when working with computers. This meant that the image on the screen was an "accurate" representation of the printed output. Display Postscript is the latest example of this trend, where the screen is thought of as nothing more than a low resolution printer which can only produce black dots. In fact, by using the gray levels of the computer screen, and filtering properly, it is possible to create much higher apparent screen resolution. This emphasis on printed output ignores the vast amount of consumed text that is never printed at all, but lies totally within the electronic environment. It is not enough that the image on the screen reminds one of a typeface, it must be that typeface; What You See Is All You Get.

Speed is of the essence in an interactive system, especially when animation is considered. In order to provide the user with as much speed as possible, while still maintaining flexibility, I am using several levels of representation

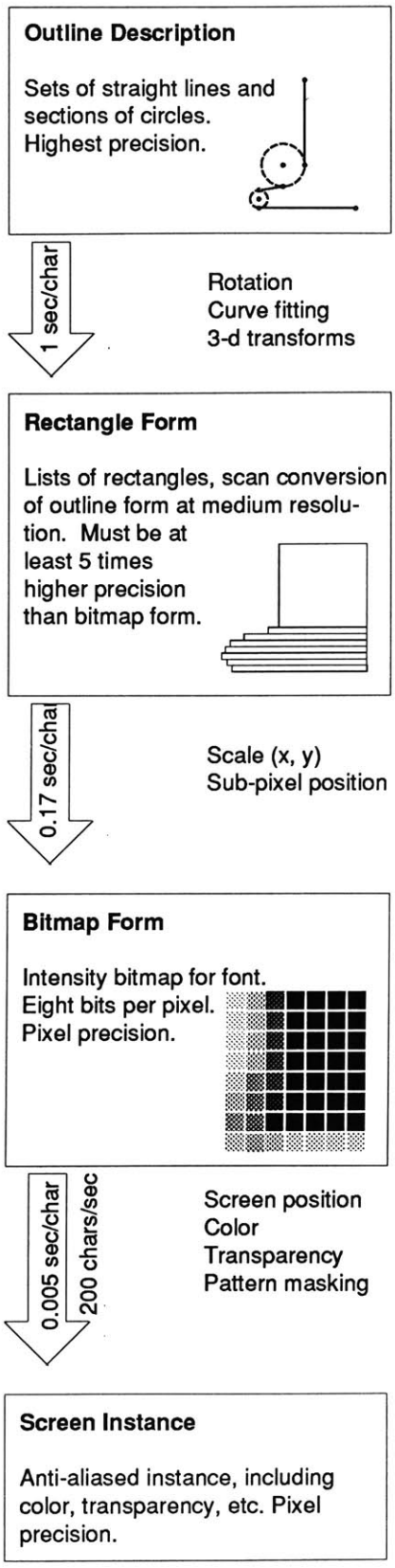


Figure 14. The Font Pipeline

for the font. Certain kinds of transformations can occur at different levels, so for example, it is possible to make some changes without always going back to the font outline. The system can be thought of as a pipeline which connects different representations: data flows from higher levels of representation to lower ones. The user can interrupt this flow, or make changes at any level. Unless the designer wants to make changes, the pipeline remains transparent.

4.2 Filtering on the Fly

Filtering is an important and well recognized way of improving the legibility of screen fonts. It has never become a widely used technique partially because anti-aliased typefaces are more difficult to produce, require greater storage space, and are rendered slower than single-bit typefaces. The simplest way to generate an anti-aliased typeface is to start with a high-resolution single-bit master and then filter it down to the desired size. The larger the master, the smoother the filtered version will be. Unfortunately this process can be quite time consuming (typically, it might take an hour to generate an entire typeface). If one needed a size which was unavailable, it would be very inconvenient to make it.

For further information on anti-aliased text see Bender, et. al. *CRT Typeface Design and Evaluation.*

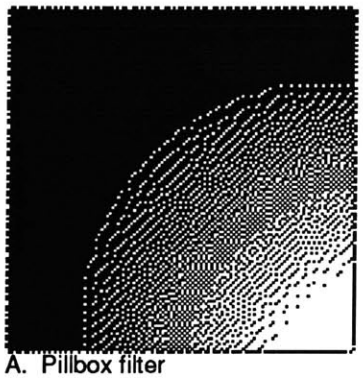
To get around some of these problems we have implemented a version of Avi Naiman's rectangular convolution method for fast filtering of characters. Essentially, this method breaks

Avi Naiman, *Rectangular Convolution for Fast Filtering of Characters.*

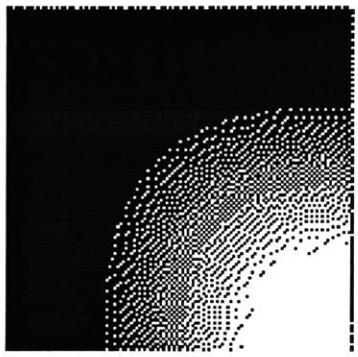
the high-resolution master font into rectangular parts. Although there are many of these component rectangles (three to four hundred per character on average), there exist very efficient algorithms for filtering rectangles (especially when the orientation is along the axes of the display). Russell Greenlee has implemented such an algorithm that uses a pre-computed filter table, rather than computing the filter for each element separately.

Using this method, we can filter a typeface in approximately fifteen seconds (240 fold improvement). This is fast enough that it is now possible to create typefaces on-the-fly rather than load them off disk. This saves disk space, but more importantly, gives the designer the flexibility to make any size face whatsoever, including fractional sizes (i.e. 12.25 point). Also, it is possible to create rotated or distorted type without any loss in quality.

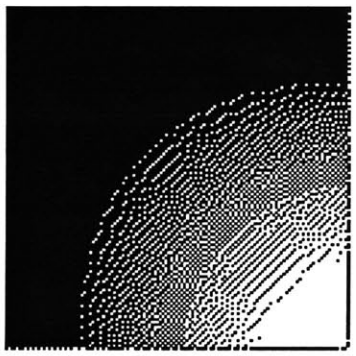
One important advantage to this method is that it uses a table look up for the filters. There are many different kinds of filters and there are trade-offs associated with each. Different filters are required depending on the size of the smallest details in a character, and on the characteristics of the CRT which is being used. By using a table to do the filtering it is possible for the user to swap in a variety of filters, or even for the computer to choose a filter which best matches the display and the typeface.



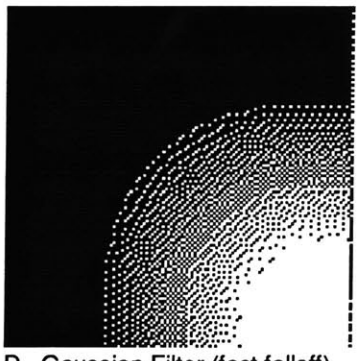
A. Pillbox filter



B. Conical Filter



C. Gaussian Flter (slow falloff).



D. Gaussian Filter (fast falloff)

Figure 15 (A-D). Several different filters which can be used to make anti-aliased fonts.

5 Wet Fonts

5.1 The Connection Machine

In this chapter current work on the Connection Machine that simulates the actions of water and pigments on a paper substrate will be discussed. The Connection Machine is a computer with a massively parallel architecture. The computer is made up of 16,000 processors, each with its own memory. All processors execute the same instructions, but they can have different values to compute with. The processors are controlled by a serial computer (also called the front-end computer) and can interface with a frame-buffer and a parallel data storage unit (the data vault). In addition to communicating with the front-end and these peripherals, individual processors can also access information stored by other processors. This architecture is ideal for simulating systems in which there are many individual components which function in similar ways. Although each processor is relatively slow, together they can process vast amounts of data.

For further information on the Connection Machine see Daniel Hillis' book *The Connection Machine*

To simulate the actions of paper, it is necessary to compute the movement of pigment and water into and out of millions of paper fibers. Although the rules governing individual fibers are quite simple, the aggregate behavior is quite complex. Each fiber needs only to know

its own state, the state of its neighbors, and some global information. This sort of simulation, with a million active fibers (1024x1024, one fiber for each pixel), would be extremely difficult to perform on a serial machine, not only because of the numbers of computations involved, but also because of the large amount of data involved (about 45.8 megabytes). It is ideally suited, however, to compute on the Connection Machine.

Memory usage

348 bits per fiber
÷ 8 bits per byte
* 1024 by
* 1024 fibers
= 45.613 megabytes

Figure 16. Memory needed to simulate fibers.

5.2 Diffusion of Pigment and Water

The goal of this project is to create an interactive simulation of paper, water, paint and brushes. Various pigments and water can be applied to the paper in a variety of ways- with brushes, geometric stamps, and photographic or typographic screens. Once applied, the pigment and water will begin to diffuse into the paper according to the physics of diffusion and transport mechanisms. By varying the way in which diffusion occurs, different kinds of paper can be simulated. In fact, it is possible to create a paper with properties that cannot exist in the real world.

The basic element of the simulation is the paper fiber. There is one fiber for each pixel on the display and one processor for each fiber. Each fiber knows its own color, the amount of each pigment it contains, how wet it is, how absorbent it is, its x and y location, and the color it computes to send to the display. The simulation can be divided into three main parts: creat-

Code Summary

- initialize CM
- create display
- create VP set
- allocate memory
- set fiber color and absorbency
- put pigment and water on paper

while (time)

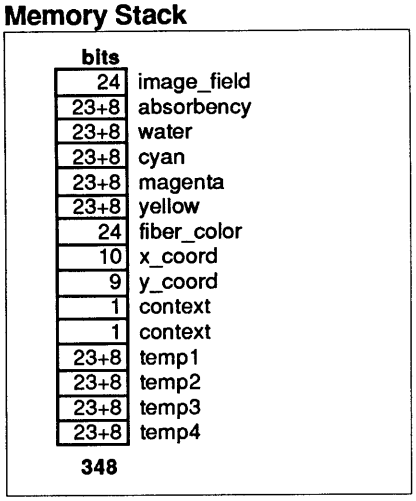
    disperse water
    disperse pigment
    create image
    send image

ing and initializing the paper, applying pigments and water, and performing diffusion cycles.

The first step requires that a display and a two-dimensional virtual processor set is created with the same width and height as the desired piece of paper. Then memory is allocated on each VP for all of the data it will need during the simulation. Memory is allocated on a stack. This means that if any field is de-allocated, any bits above it in the stack will be lost. In order to avoid any potential problems, all fields, including temporary variables, are allocated at the start of the program. Once this is done there is real memory allocated for every field on each VP. Given the current memory size on the CM the largest display that can be allocated is 1024 x 512 pixels Manipulations of these fields will take place in all active processors simultaneously. In addition to these fields there are a number of global variables. These exist in memory which is shared by all of the processors and they are not duplicated.

• VP = virtual processor

Memory Stack



Once the memory is allocated, some initial conditions are set. Fiber color and absorbancy are initialized and then randomized slightly for each pixel. Random fiber colors give the paper a speckled appearance. Random absorbency is used to generate a "noisy" dispersion function.

The second step is to apply pigments and water. In addition to its own color, each fiber can contain and be tinted by pigments. Initially there are only cyan, magenta and yellow pigments. This allows the generation of any color

mixture; however there will be situations in which one will need to use spot colors. This is a simple extension from the three pigments already used, however it will require more memory, which is limited. There are several functions which allow pigment and water to be added to the paper. A specific amount of each pigment can be added to fibers within a specified rectangle. In addition, a certain amount of water can be added. These functions depend on a property of the Connection Machine called context. Each virtual processor has one bit called the context bit. Any functions (addition, multiplication, etc) which are performed on the VP set will only be executed on those processors whose context bit is on.

A more complex method to apply images is to take a color bitmap and convert it to pigment, which is then placed on the page. The bitmap can also be used as a water mask. The function CM_(read/write)_to_news_array is provided for transferring data arrays on the front end to VP fields in the CM. Type can be applied in a manner almost identical to bitmaps, and is discussed in greater detail in Appendix 7.5.

news = north, east, west, south. Of course, if the VP set is higher dimentional, news refers to the processor neighbors in each dimention.

One more method which could be implemented is to use a brush made up of bristles and controlled with a pressure sensitive stylus. The brush works in the same manner as the fibers themselves, able to transfer pigment and water along gradients both into and out of the paper.

Steve Strassmann's paper *Hairy Brushes* is a good treatment of the issues involved in simulating brushes.

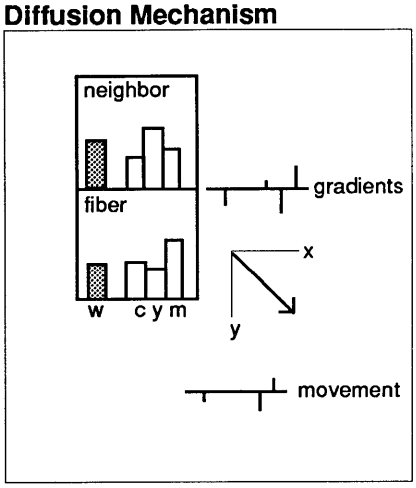
The last step is to simulate time with the diffusion cycle. This is nothing more than a loop

of dispersing water and pigment, creating a display image from the pigment content and color of each fiber, sending the display image to the frame buffer, and finally removing some water from all fibers relative to the current humidity.

The dispersion function is the heart of the program. Modifying the dispersion algorithm can create drastically different results. The first step is to disperse the water. For each fiber a water gradient is calculated relative to its neighbor. In the most simple case, one-half of the gradient (or difference) between the two fibers is subtracted from the wetter fiber and added to the drier fiber. Noise can be added to the dispersion by multiplying the gradient by the fibers' absorbency. Adjusting the amount of variation of absorbency between fibers changes the noisiness, or roughness, of the dispersion. Gravity is also taken into account by modifying the gradients. Gradients are increased along the gravity vector and decreased in the opposite direction.

Pigment diffusion is slightly more complicated. It begins the same way as the water diffusion by computing a pigment gradient and modifying it with absorbency and gravity. In addition, the gradient is multiplied by the water content of the fiber; the wetter the fiber, the easier the pigment will move. The gradient is also multiplied by the molecular mass of the pigment (a constant). Some pigments will diffuse faster than others, creating colored fringes.

Diffusion Mechanism



See Appendix 7.4 for complete description of the dispersion algorithm.

After new pigment values have been determined for all of the fibers, we can calculate the displayed color for each fiber. This color is simply the fiber color minus any pigment. This is done in three channels (rgb). The red part of the image is the red part of the fiber color minus any cyan pigment. This is similar for both green and blue. If there were any spot pigments, they would affect more than one channel. I have also written a function which computes a gray value for each pixel based on the fiber's water content. This is very useful for visualizing the water flow in the paper.

Finally, we have a 24-bit image that can be sent to the frame buffer. Because news ordering is different from frame-buffer ordering, the image must be shuffled and twiddled before it is sent to the frame buffer. This operation takes about 0.5 seconds and is the main factor in limiting interaction with the simulation.

Shuffling moves data between physical processors. Twiddling moves data around to different VP's on the same physical processor.

5.3 Type as Pigment

One of the interesting things about simulated watercolor is the ability to precisely locate pigment on the paper. This means that it is possible to create images with fine detail (as on a silkscreen) but still be able to have the pigment flow freely (as on a wet sheet of paper). This is particularly interesting with regards to typography. It is possible to create crisp type and selectively modify it as if it were made of ink. Traditionally, type has moved away from

See Appendix 7.5 for parallel font display algoritm.

ink. Technology has enabled us to remove more and more of the artifacts associated with ink, and in the case of electronic text, to remove the ink entirely. Type is thought of more as a shape, an outline, an abstract delineation between letterform and space.

Why then should it be interesting to think of type as a spot of ink on paper? Why indeed! This way of thinking broadens the possibilities of the electronic typographic image. The character becomes an active, moving, physical experience.

By thinking of a glyph, not as some pure mathematical construct, but as a loose area of continuous pigment, we are freed from the constraints of treating it as an abstraction. We can use surface tension to "round up" shapes, or use diffusion to soften them. Bold type could be made simply by using more ink. Words can be integrated into images, because now they are made from the same material. All this is possible because of this radically different way of thinking about the letterform.

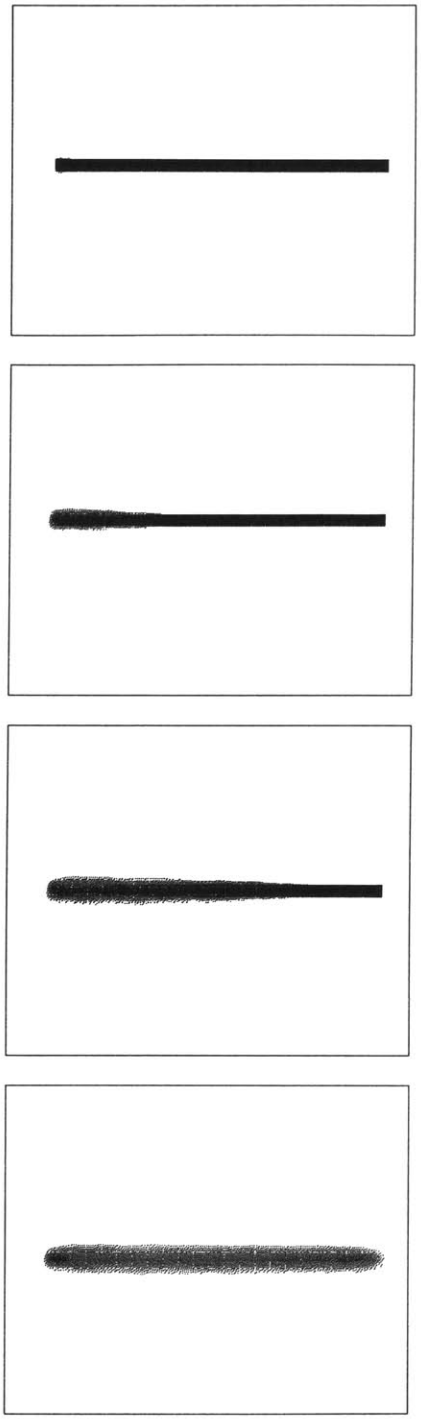
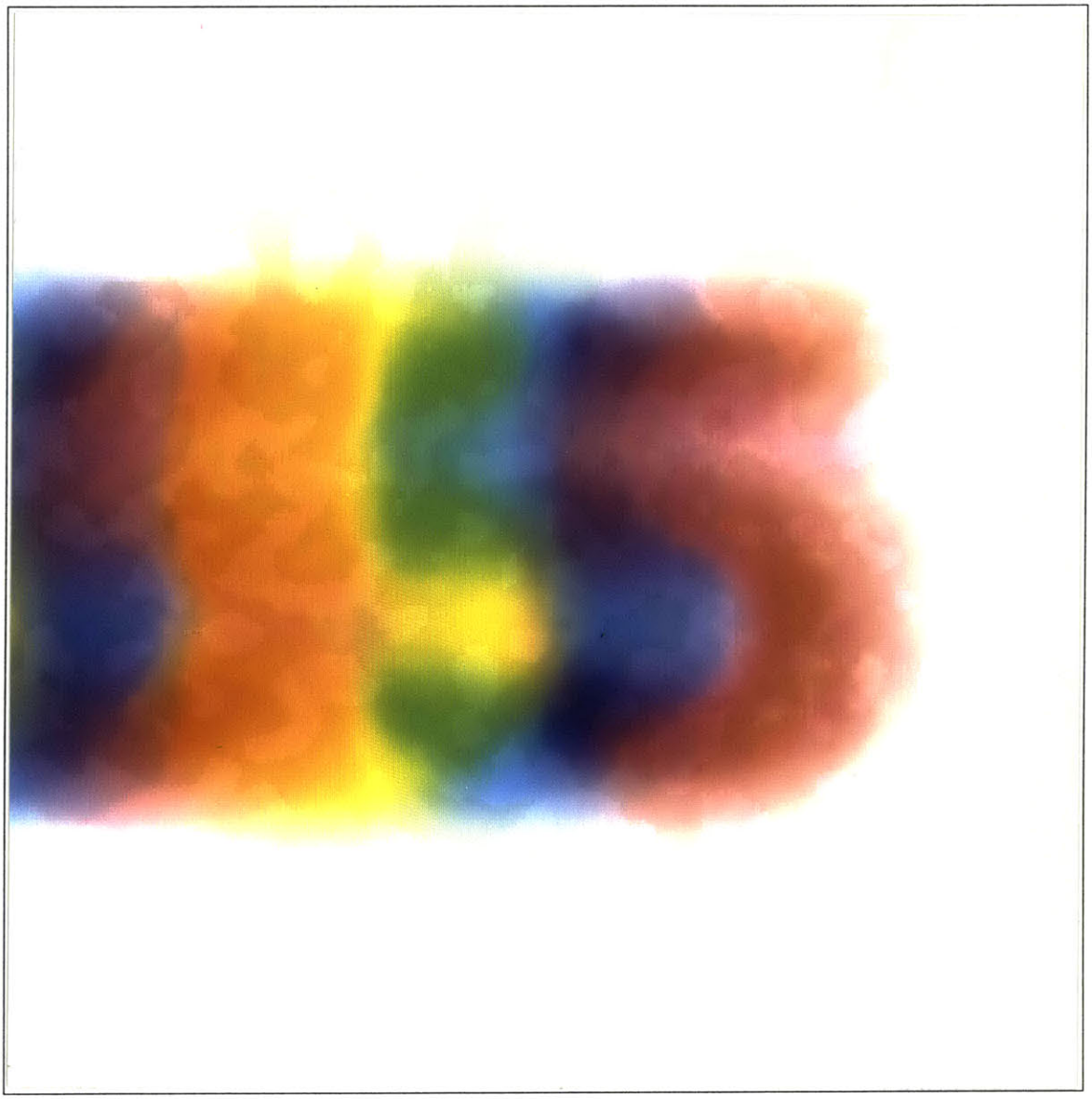


Figure 17. Series of images demonstrating the diffusion of a line of pigment. The water travels from left to right.

Figure 18. **Fives**, design used for the fifth anniversary of the Media Laboratory. The image was the result of a collaboration between David Small and Jacqueline S. Casey, Visiting Design Scholar at the Visible Language Workshop. The varnish shows where the paper is most wet.



6 Conclusion

6.1 Conclusion

This thesis sets forth a variety of ways to think about and use typography in the electronic environment. There are many issues that have been raised which deserve further investigation. There will probably never be one unifying approach to the use of typography to convey information in the electronic environment. Rather, there will be a wide selection of related ways to use type, each suited to a particular task. The integration of various methods into a coherent whole will be a difficult but essential task.

Future directions to explore include translucency, editing, sub-pixel positioning, bristly brushes and dynamic linking. Translucency can be used to expand the dimensions of graphics in the Z-axis. Some experiments have been done, such as using translucency to fade from one piece of text to another. This technique could be used in a more general way to handle linked information or annotations.

Editing abilities can be built into the system so that text can be added or modified in real time. Building a robust text editor with anti-aliased fonts poses several interesting problems. The editor needs to be able to find words in text that may contain several different type-

faces. Rendering has to be fast enough so that text can be inserted at any point, without locking out the user. Also, text should be able to perform some designing on its own, such as identifying and flowing around other objects.

With the fast filtering techniques discussed earlier, it should be possible to position text on a sub-pixel basis without causing an unacceptable loss of speed. More accurate positioning of text means that the designer has a much finer control over typesetting operations. For certain applications (such as subscripts or justified text) the benefits will outweigh the computational difficulties.

The addition of bristly brushes to the watercolor simulation will greatly enhance the interactivity of the system. One will be able to apply ink or water directly to the paper surface in a rational manner. Coupled with a model of surface tension, it will result in a more accurate and flexible simulation.

Dynamic linking of comments and annotations to the text is an important way to expand the generality of the system. Although footnotes currently perform some of the functions associated with hypertext, a more robust and general system needs to be developed. It should be able to handle an arbitrary number of layers and types of links. This will require some kind of automatic layout, so that the screen space can be dynamically allocated in such a way as to follow design rules and avoid losing the user in a sea of complexity.

6.2 About this Thesis

In addition to the paper copy filed with MIT libraries, this thesis exists as an interactive electronic text. The entire text of the thesis, as well as interactive simulations, can be traversed on-line. The purpose in doing this was to break down the separation that occurs between the text of the thesis and the subject matter being presented. Also, it was a good opportunity to explore certain design issues associated with electronic texts.

There are several key components to this implementation of the thesis: interaction, reconfigurability, annotation, a visual overview, scrolling, and linked "footnotes". Of course, being an electronic text, this thesis raises a host of issues - about linking, editing, and personalization - which have not been addressed at this time. My criteria in designing the system was that it be intuitive and as simple as possible. Interaction should be quick, and the design should not require a lot of explanation to use. At the same time, I made the software as modular and flexible as possible so that improvements could be made gradually.

Footnotes refer not only to text, but also to graphics or sketches.

Interaction is one of the key elements in the thesis. It is important that the user can explore in his or her own way the concepts that are being discussed in the writing. In making the demonstrations I wanted the reader to be able to go through the same learning process that I had (albeit with slightly less pain). In the future, I hope that I can make it easier to include

simulation objects as well as recompilable code fragments.

Reconfigurability is another important attribute of an electronic text. The layout should be abstracted as much as is possible from the specific content of each page. Users can interactively reconfigure the layout, which acts as a template for all of the pages. Footnotes stay attached to the correct part of the text, even if the font changes, or the text is reformatted. I am using a grid to simplify the design, and most of the layout is determined in grid rather than absolute coordinates. Colors should also be abstracted out of the code as much as possible and easily edited.

Included is a graphical annotation system. This allows the user to make color, pressure-sensitive, translucent, anti-aliased sketches over the thesis. These sketch objects can be moved around, saved and played back. In this way, it is possible to build up layers of annotations and still maintain the kind of quality that people expect from paper and pencil.

One constant problem with electronic texts is that it is difficult to get a quick overall view. With a book, it is easy to pick it up, riffle through the pages, and get a lot of information about the book. This feeling can be approximated electronically by allowing the user to scan through filtered miniatures of the "pages" that make up the book. Although the image quality is somewhat poor, it is easy to get a quick gestalt of what each page is about. This sort of fast browsing is a very useful and important

paradigm, especially since that computer gives you access to such a vast amount of information.

In addition to being able to move quickly through pictorial data, it is also good to be able to move through textual information smoothly and quickly without sacrificing typographic quality. One common way to move through a large body of type which will not all fit on the screen at one time is to scroll it inside of a window. A scrolling function has been implemented which maintains full anti-aliasing and still moves quite quickly.

Of course, no thesis would be complete without footnotes. Footnotes are divided into several classes (citations, explanatory notes, images, sketches, etc.). This allows us to have a looser, less constrained definition of a footnote object, as well as leaving the door open for other types of footnotes later. It was also important to be able to attach the footnote to a particular part of the text, regardless of how the text was formatted. This was done by having the text body send a message to each footnote object whenever it was reformatted specifying the new location of the link.

Scrolling Algoritm

to scroll down:

- block move all but the last line down one line.
- compute which part of the text will now fall on the first line.
- render just that line.
- repeat for as many lines as you wish to scroll.

to scroll up:

- same as above, except in reverse.
- slightly slower because it takes longer to find out what part of the text will fall on the last line.

7 Appendices

7.1 Dynamics

This code computes the forces, accelerations, velocities and new positions for objects.

```
/******** compute forces ************************/

  /******* spring forces ************************/
  for (i = 0; i < params->springs; i++) {
    TheSpring(i)->ForceX = cos(TheSpring(i)->Theta) *
TheSpring(i)->k *
        (TheSpring(i)->Distance - TheSpring(i)->rest_length);
    TheSpring(i)->ForceY = sin(TheSpring(i)->Theta) *
TheSpring(i)->k *
        (TheSpring(i)->Distance - TheSpring(i)->rest_length);
  }

  /******* forces acting on mass ************************/
  /***** initialize forces ******************************/
  for (i = 0; i < params->masses; i++) {
        TheMass(i)->ForceX = 0.0;
        TheMass(i)->ForceY = 0.0;
   }
  /***** spring forces **********************************/
  for (i = 0; i < params->springs; i++) {
    n = TheSpring(i)->end[0];
    if (!TheMass(n)->sticky || !(params->selected == n)) {
        TheMass(n)->ForceX -= TheSpring(i)->ForceX;
        TheMass(n)->ForceY -= TheSpring(i)->ForceY;
    }
    n = TheSpring(i)->end[1];
    if (!TheMass(n)->sticky || !(params->selected == n)) {
        TheMass(n)->ForceX += TheSpring(i)->ForceX;
        TheMass(n)->ForceY += TheSpring(i)->ForceY;
    }
  }
  for (i = 0; i < params->masses; i++) {
        if (!TheMass(i)->sticky || !(params->selected == i)) {
          /*** gravity ****/
          if (TheMass(i)->Wy > 0.5)    /* if it is touching the floor,
```

First, all of the forces are computed.

Spring forces.

Spring forces (with the appropriate sign) are attached to the masses.

Gravity.

```
gravity is counteracted */
            TheMass(i)->ForceY -= params->gravity *
TheMass(i)->mass;

          /*** damping ****/
          TheMass(i)->ForceX -= (TheMass(i)->VelX * params-
>damping);
          TheMass(i)->ForceY -= (TheMass(i)->VelY * params-
>damping);

        }
        /**** Compute new position from force *******/
        if (TheMass(i)->sticky || params->selected == i) {
          TheMass(i)->AccelX = 0.0;
          TheMass(i)->AccelY = 0.0;
          TheMass(i)->VelX = 0.0;
          TheMass(i)->VelY = 0.0;
        }
        else {
          TheMass(i)->AccelX = TheMass(i)->ForceX /
TheMass(i)->mass;
          TheMass(i)->AccelY = TheMass(i)->ForceY /
TheMass(i)->mass;
          TheMass(i)->VelX += TheMass(i)->AccelX*params-
>dTime;
          TheMass(i)->VelY += TheMass(i)->AccelY*params-
>dTime;

          if (TheMass(i)->Wy < 0.5)     /* if it hits the floor,
bounce */
            if (TheMass(i)->VelY < 0.0) {
              TheMass(i)->VelY = -(params->floor) * TheMass(i)-
>VelY;
              if (TheMass(i)->VelY > 0.2) { /* make a clank sound
*/
                    velocity = (int) (TheMass(i)->VelY * 20);
                    if (velocity > 127) velocity = 127;
                    clank(velocity);
              }
            }
          TheMass(i)->Wx += TheMass(i)->VelX * params-
>dTime +
            0.5 * TheMass(i)->AccelX * params->dTime *
params->dTime;
```

Damping forces.

If the mass is fixed or held by the cursor, its acceleration is zero.

Acceleration = force / mass.

Velocity = acceleration * dt.

Collision detection:
If there is a colision with the floor, the Y component of the velocity is reflected back scaled by the floor's damping factor.

Make a sound when the mass hits the floor.

Compute new positions for each mass. Position += velocity * dt + 1/2 acceleration * dt^2

```
            TheMass(i)->Wy += TheMass(i)->VelY * params-
>dTime +
                0.5 * TheMass(i)->AccelY * params->dTime *
params->dTime;
            }
    }
```

7.2 Distortion with Matrices

This code demonstrates how to compute the 4x4 matrix which will transform the unit rectangle to the general quadrilateral {(x1, y1), (x2, y2), (x3, y3), (x4,y4)}.

```
 Tx = x1;
 Ty = y1;
```

Compute translation.

```
Translate(M, Tx, Ty, 0.0, FALSE);
concat_transformation3d(screen, M, PRE, PUSH);

ScaleX = x2 - Tx;
ScaleY = y4 - Ty;
```

Compute scale.

```
Scale(M, ScaleX, ScaleY, 1.0, FALSE);
concat_transformation3d(screen, M, PRE, REPLACE);

ShearX = (x4 - Tx)/ScaleX;
ShearY = (y2 - Ty)/ScaleY;
```

Compute shear.

```
Shear(M, ShearX, ShearY, FALSE);
concat_transformation3d(screen, M, PRE, REPLACE);

 /*********
          Hx = (x3-Tx)/ScaleX - ShearX*Hy;
          Hy = (y3-Ty)/ScaleY - ShearY*Hx;
Solving for Hx, we get:
***********/

Hx = ((x3-Tx)/ScaleX - ShearX*(y3-Ty)/ScaleY) /
                (1 - (ShearX*ShearY));
Hy = (y3-Ty)/ScaleY - ShearY*Hx;
```

Compute homogenous coordinate.

```
Homo(M, (1 - Hy) / (Hx + Hy - 1), (1 - Hx) / (Hx + Hy - 1),
FALSE);
 concat_transformation3d(screen, M, PRE, REPLACE);

 draw_rect_char2(screen, params->rfont, params->string[i] - 33,
                                250, 250, 250, 0, 0);
```

Draw character.

```
pop_matrix(screen);
```

Pop all matrices off of the matrix stack.

```
Shear(M,x,y,c)
float M[4][4],x,y;
int c;
{
 float temp[4][4];

   MakeIdentity(M);
   M[1][0] = x;
   M[0][1] = y;
}
```

Subroutines:
Shear.

```
Homo(M,a,b,c)
float M[4][4],a,b;
int c;
{
 float temp[4][4];

   MakeIdentity(M);
   M[0][3] = a;
   M[0][0] = a + 1;
   M[1][3] = b;
   M[1][1] = b + 1;
}
```

Homogenous.

7.3 The MIDI Server

The work described below was done in large part by Eric MacDonald.

The hardware device we are using is the Hinton Box. It provides us with real-time interfacing between RS-232 (a common communications standard used by computers) and MIDI. In addition to this hardware, software must be written to provide transparent access to the Hinton Box from windows running on several different workstations. Let's trace a signal from the keyboard to the window environment and then back to the sampler. First, someone

presses the key on the keyboard. It figures out which key has been pressed and how quickly and sends a three byte long packet to its MIDI out port. The packet goes out onto the MIDI loop, travelling through the AKAI and into the Hinton Box. Here, the data is read and passed through several optional filters (the filters can strip out extraneous information such as after-touch). They are then stored in a buffer until the server process reads from the serial port. The server reads the bytes in and immediately writes them out to the pipe (same as stdin) and flushes the buffer. (It is important that sanitary conditions are maintained to prevent lags). The window system is running a loop during which it samples all of the input devices and then sends messages to the appropriate windows. Every cycle it checks the MIDI input buffer. If it finds any data it sends a data received message to the current MIDI window. The window reads the data, figures out which key has been pressed and how hard, and then performs some action (for example, highlighting a letter). Now let us suppose the a window wants to trigger a sound. It simply sends some data backwards through the same elements until it reaches, for example, the AKAI and triggers a sample.

Using a seperate process for the server simplifies communications and allows many machines to have access to the sound equipment without any re-cabling.

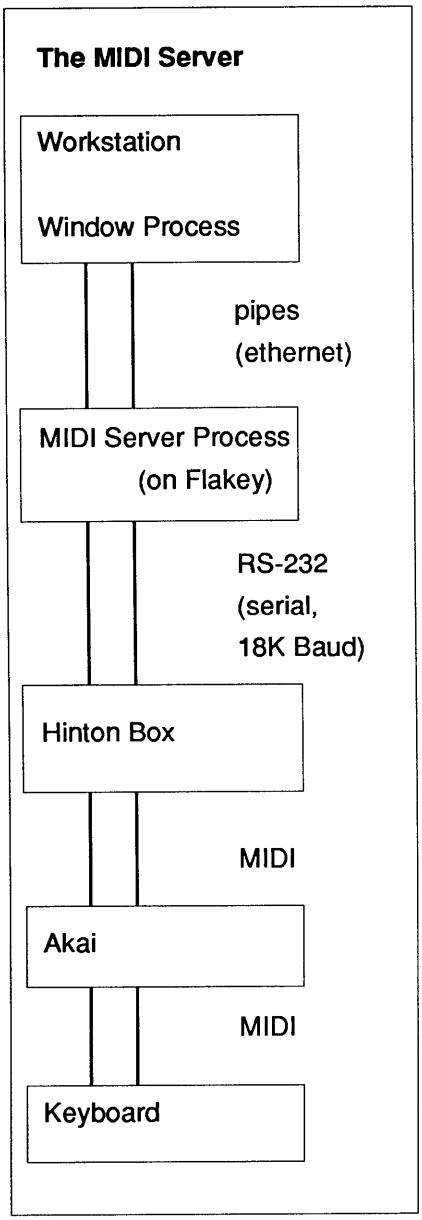


Figure 19. The MIDI server.

7.4 The Dispersion Algorithm

This code demonstrates a simple dispersion algoritm implemented in C/Paris.

```
disperse_pigment_and_water(Page)
    struct FiberStruct *Page;
{
  /*** EAST/WEST ****/
  disperse_to_neighbor(Page, X);
  /*** NORTH/SOUTH ***/
  disperse_to_neighbor(Page, Y);
}
```

Dispersion occurs in each dimension seperately.

```
disperse_to_neighbor(Page, axis)
    struct FiberStruct *Page;
    unsigned int  axis;
{
  CM_field_id_t inward_gradient;
  CM_field_id_t outward_gradient;
  CM_field_id_t inflow;
  CM_field_id_t outflow;
  CM_field_id_t pigment[3];

  ... initialization of variables...

  Turn_on_context(Page, Page->context, 2, 2, Page->width - 2,
Page->height - 2);

  /****** calculate gradients *********/
CM_get_from_news_1L(inflow, Page->water, axis, 1, s + e + 1);
CM_f_subtract_3_1L(inward_gradient, inflow, Page->water,s, e);
CM_f_multiply_2_1L(inward_gradient, Page->absorbency, s, e);

  /***** adjust gradients for gravity ********/
  CM_store_context(Page->old_context_store);
  CM_f_gt_zero_1L(inward_gradient, s, e);
  CM_logand_context_with_test();

  CM_f_multiply_constant_2_1L(inward_gradient, gravity, s, e);
  CM_load_context(Page->old_context_store);

  CM_store_context(Page->old_context_store);
  CM_f_lt_zero_1L(inward_gradient, s, e);
  CM_logand_context_with_test();
```

CM_field_id_t refers to a parrallel data structure.Turn on only those processors not on the edge of the paper. This prevents wrap-around.

gradient = absorbency * (neighbor - self)

Adjust gradients to account for gravity.

```
 CM_f_divide_constant_2_1L(inward_gradient, gravity, s, e);
 CM_load_context(Page->old_context_store);

 CM_get_from_news_1L(outward_gradient, inward_gradient,
axis, 0, s + e + 1);

 /**** calculate new water ********/
 CM_f_add_2_1L(Page->water, inward_gradient, s, e);
 CM_f_subtract_2_1L(Page->water, outward_gradient, s, e);
 CM_f_min_constant_2_1L(Page->water, 1.0, s, e);
 CM_f_max_constant_2_1L(Page->water, 0.0, s, e);

 for (i = 0; i < 3; i++) {
  /**** calculate new pigment ********/
   /****** calculate gradients *********/
   CM_get_from_news_1L(inflow, pigment[i], axis, 1, s + e + 1);
   CM_f_subtract_3_1L(inward_gradient, inflow, pigment[i], s, e);
   CM_f_multiply_2_1L(inward_gradient, Page->absorbency, s,
e);
   CM_f_multiply_2_1L(inward_gradient, Page->water, s, e);
   CM_f_multiply_constant_2_1L(inward_gradient,
molecular_weight[i], s, e);

   /***** adjust gradients for gravity ********/
   CM_store_context(Page->old_context_store);
   CM_f_gt_zero_1L(inward_gradient, s, e);
   CM_logand_context_with_test();

   CM_f_multiply_constant_2_1L(inward_gradient, gravity, s, e);
   CM_load_context(Page->old_context_store);

   CM_store_context(Page->old_context_store);
   CM_f_lt_zero_1L(inward_gradient, s, e);
   CM_logand_context_with_test();

   CM_f_divide_constant_2_1L(inward_gradient, gravity, s, e);
   CM_load_context(Page->old_context_store);

   CM_get_from_news_1L(outward_gradient, inward_gradient,
 axis, 0, s + e + 1);
```

water = water + inward gradient - outward_gradient

For each pigment calculate gradients and new values.

pigment_gradient = molecular_weight * absorbency * (neighbor - self)

Adjust gradients to account for gravity.

```
  /**** calculate new pigment ********/
  CM_f_add_2_1L(pigment[i], inward_gradient, s, e);
  CM_f_subtract_2_1L(pigment[i], outward_gradient, s, e);
  CM_f_min_constant_2_1L(pigment[i], 1.0, s, e);
  CM_f_max_constant_2_1L(pigment[i], 0.0, s, e);
 }

 /*** restore context ******************************/
 CM_load_context(Page->context);
}
```

pigment = pigment + inward_gradient - outward_gradient

Restore context.

7.5 Parallel Display of Type

```
Read_image_from_disk_and_screen_ink_on_paper(Page, file-
name, width, height, x_off, y_off, cyan, magenta, yellow, water)
    struct FiberStruct *Page;
    char            *filename;
    int             width, height, x_off, y_off;
    float           cyan, magenta, yellow, water;
{
  ... initialize vectors associated with
CM_u_write_to_news_array_iL ...

  if ((fp = fopen (filename, "r")) == NULL)
  { printf ("error opening %s, exiting.\n", filename);
    perror ("");
    exit (0);
  }

  if ((buff = (unsigned char *) malloc (width * height )) == NULL)
  { perror ("couldn't malloc buff, exiting");
    exit (0);
  }

  if ((test = fread (buff, 1, width * height, fp)) != width * height)
  { printf ("error reading %s, exiting\n", filename);
    perror ("");
    exit (0);
  }

  CM_u_move_zero_1L(Page->image_field, Page-
>bits_per_pixel);
```

This code demonstrates a parallel algoritm for the display of type.

Open file containing font bitmap.

Malloc buffer.

Read in font bitmap.

Clear image field.

```
 CM_u_write_to_news_array_1L(red, offset_vector, start_vector,
end_vector, axis_vector, Page->image_red, 8, 2, dimention_vector,
1);

 fclose (fp);
 free (buff);

 /*** turn on context x1, y1, x2, y2 only **********/
 Turn_on_context(Page, Page->context, x_off, y_off, x_off + width -
1, y_off + height - 1);

 CM_f_u_float_2_2L(Page->temp1, Page->image_red,
CHANNEL_SIZE, Page->s, Page->e);

 CM_f_divide_constant_2_1L(Page->temp1, 255.0, Page->s, Page-
>e);

CM_f_multiply_constant_3_1L(Page->temp2, Page->temp1, cyan,
Page->s, Page->e);
CM_f_add_2_1L(Page->cyan, Page->temp2, Page->s, Page->e);
CM_f_min_constant_2_1L(Page->cyan, 1.0, Page->s, Page->e);
CM_f_max_constant_2_1L(Page->cyan, 0.0, Page->s, Page->e);

CM_f_multiply_constant_3_1L(Page->temp2,Page->temp1,
magenta, Page->s, Page->e);
CM_f_add_2_1L(Page->magenta,Page->temp2,Page->s,Page->e);
CM_f_min_constant_2_1L(Page->magenta,1.0,Page->s,Page->e);
CM_f_max_constant_2_1L(Page->magenta,0.0,Page->s,Page->e);

CM_f_multiply_constant_3_1L(Page->temp2, Page->temp1, yellow,
Page->s, Page->e);
CM_f_add_2_1L(Page->yellow,Page->temp2,Page->s,Page->e);
CM_f_min_constant_2_1L(Page->yellow,1.0,Page->s,Page->e);
CM_f_max_constant_2_1L(Page->yellow,0.0,Page->s,Page->e);

CM_f_multiply_constant_3_1L(Page->temp2, Page->temp1, water,
Page->s, Page->e);
CM_f_add_2_1L(Page->water,Page->temp2,Page->s,Page->e);
CM_f_min_constant_2_1L(Page->water, 1.0,Page->s,Page->e);
CM_f_max_constant_2_1L(Page->water, 0.0,Page->s,Page->e);

CM_load_context(Page->context);
}
```

Write font bitmap to red bank of image.

Convert to float and store in temp1.

Compute new cyan,

magenta,

yellow,

and water.

Restore context.

8 Bibliography

Beethoven, Ludwig Van. *Symphonie IX.* Novello, London.

Bender, Walter, et. al. *CRT typeface design and evaluation.* Proceedings of the Human Factors Society 31st Annual Meeting; October 19-23, 1987:New York. Santa Monica, CA: Human Factors Society; 1987; 2: 1311-1314.

Cahn, Janet. *Generating Expression in Synthesized Speech.* Master's thesis, MIT. 1989.

Conger, Jim. *C Programming for MIDI.* M&T Publishing, Redwood City, California. 1988.

d'Harnoncourt, Anne and McShine, Kynaston, eds. *Marcel Duchamp.* Museum of Modern Art, New York. 1973.

Fragmenta Chiliandarica Palaeoslavica, a photoreproduction of the B. Hirmologium Codex Monasterii Chiliandarica 308. Monumenta Musicae Byzantinae, Volume V. Ejnar Munksgaardd, Copenhagen. 1957.

Hillis, Daniel. *The Connection Machine.* MIT Press, Cambridge MA. 1987.

Ionesco, Massin, and Cohen. *The Bald Soprano.* Grove Press, Inc. New York, 1956.

McKenna, Michael. *A Dynamic Model of Locomotion for Computer Animation.* Master's thesis, MIT. February 1990.

MIDI Musical Instrument Digital Interface: Specification 1.0.

Naiman, Avi and Fournier, Alain. *Rectangular Convolution for Fast Filtering of Characters.* Computer Graphics, Volume 21, number 4. Published by ACM Siggraph. July, 1987.

Pietgen, H.-O. and Saupe, D. *The Science of Fractal Images.* Springer-Verlag, 1988.

Rubin, William S. *Dada and Surrealist Art.* Henry Abrams, New York.

Schröder, Peter. *The Virtual Erector Set.* Master's thesis, MIT. February, 1990.

Spencer, Herbert. *Pioneers of Modern Typography.* MIT Press, Cambridge, MA. 1983.

Strassmann, Steve. *Hairy Brushes.* Computer Graphics, Volume 20, number 4. Published by ACM Siggraph. August, 1986.

Toffoli, Tommaso. *Cellular Automata Machines.* MIT Press, Cambridge, MA. 1987.

Uubry, Piere. *Cent Motets du XIIIe Siecle.* Broude Brothers, New York. 1964.

Wilhelms, Jane. *Dynamics for Everyone.* Fourth USENIX Computer Graphics Workshop, 1987.

Wolfram, Stephen. *Theory and Applications of Cellular Automata, Including Selected Papers.* World Scientific, Singapore. 1986.