To the Graduate Council:

I am submitting herewith a thesis written by Eric S. Evers entitled "Gestural Interfaces Suggested by Field Computation". I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Dr. Bruce MacLennan
Major Professor

We have read this thesis
and recommend its acceptance:

Michael W. Berry

Brad Vender Zanden

Accepted for the Council:

Associate Vice Chancellor
and
Dean of the Graduate School

GESTURAL INTERFACES SUGGESTED BY FIELD COMPUTATION

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Eric S. Evers
May 1995

Acknowledgments

First I would like to thank Dr. Bruce MacLennan. With his example I have learned to question fundamental assumptions as a way to arrive at new understanding.

A general thanks is extended to the Computer Science Department of the University of Tennessee at Knoxville for challenging me and expanding my interests.

I am grateful for the support of my wife, Kristen Reynolds, without whom this thesis would have been much more difficult to complete.

Last and not least, I would like to thank my parents, Sharon and Wesley Evers, from whom I received all that I needed to start this thesis: self-discipline, self-confidence and curiosity.

Eric S. Evers

Table of Contents

Page

1 Introduction 1
1.1 Computers are Tools 1
1.1.1 Previous Research on Gestures 1
1.1.2 How This Thesis Is Related to Previous Research 3
1.2 Field Computation 4
2 Problems with Interfaces 5
2.1 Expressiveness 5
2.2 Parallel Speedup 6
2.3 Scalability 6
2.4 Consistency 6
2.5 Robustness 7
3 Motivations Toward a General Solution to Problems with Interfaces 8
3.1 Expressiveness from Closeness of Fit 8
3.2 Field Computational Interfaces are Robust and are Compatible with VR 9
4 A General Solution to Interface Problems: Tactilization 10
4.1 Dimensional Analysis of Interfaces 10
4.2 The Syntax of Fields is Differential 13
4.3 The Semantics of Fields are Continuous 15
4.4 Classification of Constraints 15
4.4.1 Hard Constraints: Static and Moving 16
4.4.2 Soft Constraints: Static and Moving 17
4.4.3 Informational Constraints 17
4.4.4 Relativistic Constraints 18
4.5 Continuous Mathematical Models of Gestures 19
4.5.1 Differential Geometry 20
4.5.2 Differential Flows 21
5 Implementations 24
5.1 Menu Interfaces 24
5.1.1 Nearly Continuous Menus 24
5.1.2 Continuous Menus 26
5.2 Continuous Gesture-Operator Spaces 29
5.2.1 Selection/Encirclement of an Icon 30
5.2.2 The Log-Shell Space 33
5.2.3 Conformal Maps of Response Regions 33
5.2.4 Toroidal Spaces 37
5.2.5 Hyperbolic Spaces 38

6 Creating and Editing Fields 40
6.1 Encoding Fields by the Placement of Minima and Maxima 40
6.2 Drawing Energy Surfaces 46
7 Conclusions 50
7.1 Field Computation Derived Interfaces 50
7.2 Designing Field Interfaces 51
7.3 The Meta-Design of User Interfaces 51
8 Directions for Future Work 52
8.1 Field Computation Interface Standards 52
8.2 Semidirect Interfaces Via Dynamical Systems 52
8.2.1 Soft Constraints as Semi-Direct Interfaces 53
8.2.2 Single User Visio-Tactile Dialogs 54
8.2.3 Multi User Visio-Tactile Dialogs 55
8.2.4 Force Interfaces 55
8.2.5 Momentum Interfaces 56
8.2.6 Acceleration Interfaces 57
8.2.7 Groupware 58
8.2.8 Advantages of Semi-Direct Interfaces ... 60
REFERENCES 63

List of Figures

		Page
Figure 4.1	Dimensional Analysis of Gestures	12
Figure 4.2	Integral Operator for GestureRecognition	13
Figure 4.2.1	Gestures as a Discrete Language	14
Figure 4.2.2	Gestures as a Continuous Language	14
Figure 4.5.2.1	Definition of Flow	21
Figure 4.5.2.2	Details of Flow Definition	22
Figure 4.5.2.3	A Saddle Flow	23
Figure 5.1.1.1	Semi-Continuous Menu Selection	27
Figure 5.1.1.2	Nearly Continuous Menu Selection	27
Figure 5.1.2	A Branch Cut Menu	28
Figure 5.2.1.1	Encirclement Equations	31
Figure 5.2.1.2	Encirclement Program Screen	32
Figure 5.2.2	The Log-shell Space	34
Figure 5.2.3.1	User Drawn Radial Response Curve	35
Figure 5.2.3.2	Conformal Mapping Function of a Star to a Circle	36
Figure 5.2.3.3	Conformal Map of a Star to a Circle	37
Figure 5.2.4	A Toroidal Space	38
Figure 5.2.5	A Hyperbolic Space	39
Figure 6.1.1	Flow Lines of a Flow Field	41
Figure 6.1.1.1	Field Computer Flow Equations	42
Figure 6.1.2	Emission Probability Editor	44
Figure 6.1.3	Flow Field of Three Dipoles	45
Figure 6.1.4	Flow Near a Dipole Pair	46
Figure 6.2.1	Air Brush Flow as a Function of Time	47
Figure 6.2.2	Continuous Competitive Learning	48
Figure 8.2	Semidirect Interfaces	53
Figure 8.2.1	Soft Constraints as a Gestural Interface	55
Figure 8.2.7	Data Processing Tactio-Visualization	60

ABSTRACT

This paper describes how to construct gestural interfaces from building blocks derived from field computation. Field computational building blocks are combined to produce recognizers of oriented gestures in a plane. The topics discussed include a unification of pull-down menus with the desktop using a branch cut type topology and an investigation of ways to automate drop and drag methods via the flow fields of autonomous differential equations. This paper also discusses how to measure primitive qualities of gestures such as encirclement in a mathematically consistent way using path integrals. When these building blocks are combined, gestural interfaces and methods of visual programming result.

1 Introduction

1.1 Computers are Tools

Tools exist to help convert human will into action. Computers are tools. There are a number of metrics that are used to describe the quality of the conversion of will into action using computers: language levels, user friendliness, learning curves, transparency, and productivity. Human computer interfaces are important because they impact the friendlyness of computers. This thesis explores methods of designing human computer interfaces. In particular it looks at ways to design gestural interfaces which are suggested by a field model of computation. Human gestures are the inputs to human computer interfaces and sound and graphics are outputs. Human gestures will be the focus but this paper also discusses topics such as visualization and visual programming.

1.1.1 Previous Research on Gestures

A good discussion of gesture recognition can be found in the 1991 doctoral thesis of Dean Harris Rubine of Carnegie Mellon University, entitled The Automatic Recognition of Gestures. Rubine describes a classical neural network method of recognizing gestures [Rubine 1991]. Gestures are mapped into feature vectors. Feature vectors are mapped to various discrete classes of gestures. Once a gesture is mapped to a class, that class evokes some action. The set of gesture

classes is discrete and finite, typically smaller than 100 classes.

Zorpette [1994] describes the use of particle animation, which is an application of field computation to the generation of visual effects for animation video. Fields are used to direct the particles of light which form the images of an animated video.

A large body of research has been devoted to robot vision based on optical field flows. By analyzing the optical field flow in a sequence of video images seen by a robot, one can deduce the shape and motion of objects in the environment. Optical flows generated by gradations of light on objects due to shadow and perspective can inform one about the shape of objects. Limitations exist in the use of optical field flow analysis due to the secondhand nature of the information, as described in an article titled "Inherent Ambiguities in Recovering 3-D Motion and Structure from a Noisy Flow Field":

> [Optical] Flow fields generated by existing techniques are noisy and partially incorrect especially near occlusion or motion boundaries... Two problems may arise due to the presence of noise in the flow field. First, motion parameters of the sensor or a moving object may be extremely difficult to estimate because there may exist a large set of significantly incorrect solutions which induce flow fields similar to the correct one. The second problem, which is closely related to the first one, is in the decomposition of the environment into independently moving objects. Two such objects may induce optical flows which are compatible with the same motion parameters, and hence, there is no way to refute the hypothesis that these flows are generated by one rigid object. These ambiguities are inherent in the sense that they are algorithm-independent [Adiv 1989].

In "Navigation Templates: Mediating Qualitative guidance and Quantitative Control in Mobile Robots" Slack has

discovered a method of directing mobile robots. His approach is qualitative and is a good example of field computation [Slack 1993]. Slack takes obstacle objects and goal objects and associates with each an abstract force field of an appropriate geometry. The normalized vector sum of the object fields generate a flow field which guides the robot.

Krozel discusses an interesting algorithm which is directly related to my thesis. "Intelligent Path Prediction for Vehicular Travel" develops a continuous algorithm to predict the destination of a vehicle moving smoothly on a manifold [Krozel 1993].

1.1.2 How This Thesis Is Related to Previous Research

Classical neural networks as applied to gesture recognition map a continuous input to a finite discrete set of output classes [Rubine 1991]. This paper does not model gestures as a discrete or finite language but models gestures as a continuous language [MacLennan 1994]. Often the gesture takes its meaning from the structure of the space around the gesture. The structure of the space is in turn generated by objects in the space.

The purpose of this thesis is to solve the generalized inverse problem of Slack's robot navigation problem. Slack starts with a set of obstacle and goal objects. He wishes to generate a gesture of robot motion towards a goal. Inversely, my thesis starts with a gesture in progress, and an environment full of goal objects. This paper addresses the problem of determining the goal or meaning of the gesture in progress. It is important for advanced interfaces to be able to recognize gestures in a more global context because the user is gesticulating in a space created by the placement of many objects.

My thesis uses a variation of Krozel's equation for a measure of path correlation to fields. Measuring path correlations to helps to measure properties of gestures. I rediscovered this equation and improved it by changing the scalar field to a complex valued field. This method is developed in the section of this paper on implementations.

1.2 Field Computation

Historically there are two types of recordable knowledge. The first is knowledge that can be written down in discrete symbols. The second type of knowledge cannot be described discretely.

Continuous knowledge can be stored and manipulated in a continuum computer(also known as a field computer). Continuum computer beavior is seen in a conputer as the number of processors approaches a million. These processors are connected in some topology, a type of connected compact manifold. In general the majority of memory is local and the topology is locally and simply connected [MacLennan 1987].

The data types which field computers handle are fields which exist in continuous topological spaces. Operators change these fields, sometimes creating new fields in new spaces of a different dimension.

2 Problems with Interfaces

What approach should programmers use to design interfaces which allow humans to communicate with computers that meets the requirements of parallel speedup, scalability, consistency, robustness and expressiveness?

2.1 Expressiveness

Maintaining a highly responsive virtual world is a problem for three reasons: the ability of the device to take in large amounts of interface data, the ability to output vast quantities of data, and the abillity to make the latter a function of the former in real time. If interface systems are not designed properly, the expressiveness interface suffers. Expressiveness is the output data flow rate as a function of input data flow rate. The reason I choose the word expressiveness is because it reflects the abillity of the user to express herself quickly in a virtual world without needing to worry that the virtual world representation will update in real time. Slowness in updating a virtual world representation can cause very serious problems such as nausea for users. Expressiveness also refers to the level of spatial and auditory resolution of the inputs and outputs.

2.2 Parallel Speedup

From a computational point of view, how does one manage to quickly recognize, represent and generate responses to multiple human bodies moving in a three dimensoinal space? Requirements for computational power demand the use of parallel hardware. Even if the hardware used by each person is of large granularity, when it is connected to the hardware of other users to create a large virtual world, the effective networked computer will be of small granularity. The problem of parallel speedup is how to get 10 processors working on the same problem to achieve nearly 10 times the solution speed.

2.3 Scalability

Scalability refers to the number of users involved in a virtual world. If more than one user makes the same gesture the system response should be increased. An example of multiple action would be an auditorium full of people making a wave with their arms. The system response to 100 clapping would be similar but greater in magnitude to the system response of 10 people clapping. A similar increased system response would be expected if one user makes the same gesture with more than one appendage, such as pounding a table top and stomping their feet compared with the system response of stomping their feet alone.

2.4 Consistency

Consistency refers to the possibility of having consistency of meaning assigned to gestures independent of

the type of human appendage used to make the gesture, or the type of input device used (a mouse vs. a data glove).

2.5 Robustness

Any interface algorithms should be robust and not brittle, meaning that it should be difficult to make the interface algorithms malfunction when extending or contracting the virtual world dynamically.

3 Motivations Toward a General Solution to Problems with Interfaces

3.1 Expressiveness from Closeness of Fit

This section will explore arguments for field computation derived interfaces from closeness of fit. Human beings are highly parallel information processors. The function of parts of the human brain can be approximated by field computers. Indeed, Pribram claims that parts of the visual cortex actually are field computers[Pribram 1991]. How do humans communicate with one another? Generally they prefer a communication channel with a high degree of parallelism.

Photographic images are examples of fields. Images are the most efficient method to communicate with a field computer. The quickest way for father to give a coworker an impression his child is to show a photograph of the child.

Fields are not only visual images. Any type of sensory image of high dimension quallifies as a field. A human has billions of pressure, pain, and temperature sensitive nerve cells in the manifold of its epidermis. In order to learn how to knead bread, which is faster: read a book on the subject, or have a friend demonstrate how to knead bread? A demonstration is the most timely method. Discrete strings of symbols can generate only a limited approximation to the sensation of kneading bread compared to the actual million-neuron somatosensory stimulation available from kneading real bread dough.

Recent models of massively parallel computation by MacLennan et al. have uncovered an interesting problem. The

new model of parallel processing called field computation involves processing information in continuous n-dimensional fields. Massively parallel computers can be programmed as field computers. An example might be the use of n-dimensional lattice of processors to model weather patterns. The output of such a simulation would require the use of visualization software to construct color images that represent meaningful results.

On the one hand, humans which are partly field computers, and on the other hand, massively parallel computers which can behave as field computers. There exist means to communicate fields from the computer to the human user, namely visualizations. There is no way to communicate fields from a human field computer to the artificial field computer.

To fill this void, an area of study called tactilization suggests itself. Tactilization is a type of visualization, which allows the computer to feel the gesture of the human in a way that is efficient and natural for human and computer. Tactilization retains the high dimensionally of gestures through the interface.

3.2 Field Computational Interfaces are Robust and are Compatible with VR

One wishes that an interface model be robust. Field computational interfaces are robust because the logic is fuzzy rather than hard.

Fortunately, field computation is also compatible with the continuous dynamical systems exhibited by virtual worlds and gestures.

4 A General Solution to Interface Problems: Tactilization

Tactilization refers to a way of designing a kinesthetic interface to a computer system. Tactilization includes field computation derived interfaces which recognize gestures.

Tactilization includes both programs and devices whose behavior and structure are suggested by field computation. Because the syntax of fields is continuous and the information carried by gestures is potentially large, it is reasonable to posit that this type of interface could be more efficient than current interfaces if one assumes that the user and/or the computer is field computational.

Human appendages are limited to moving at a finite speed continuously through space. The continuous path of a point located on a human appendage moving through space can be modeled perfectly by sampling the location at a high enough rate. (The Nyquist Theorem states that sampling the position of an object at a rate twice the highest physical vibration allows for the perfect reconstruction of a movement).

One can rate a gestural interface in terms of the number of dimensions in which it communicates information to the machine, and the resolution of the space in which the field resides. The following subsections of this paper are an interpretation of tactilization.

4.1 Dimensional Analysis of Interfaces

Figure 4.1 is a simple form of the dimensional analysis of interfaces developed by [MacKinlay 1990]. There are four levels in the MacKinlay chart. 1, 10, 100, and ∞, which is interesting from an anthropological standpoint. Here one can see the primitive tribal number system of {one,two,three,many}, rendered in exponential notation. It would seem that this number system is sufficient to describe qualitatively the range of dimensionally in input devices. If a device has a range much greater than 100 then it is perceived as a continuous interface. This is compatible with statements by MacLennan which set practical continuum behavior near a million.

Gestures can be thought of as timeless one dimensional fields imbedded in a higher dimensional space as seen from the future. Simple gestures like mouse drags are 1-D fields imbedded in a two dimensional space. Looked at from the present a gesture is a time varying field.

computer key board	(80 x 1)	
potentiometer	(1 x $\Re$)	
mouse cursor windowing environment	(3 x $\Re$ x $\Re$) + (3 x 1)	x and y coordinates three mouse buttons
data glove	$\Re^{(3 \times 5)}$	(5 fingers by three joints)

a potentiometer,

an axial or linear variable resistor.
some of these devices involve a digital simulation a continuum which uses 128 or greater discrete values.
A potentiometer is a tactile interface that involves real access to a 1-D continuum: $\Re$

a mouse cursor

a mouse cursor is a tactile interface that involves real access to a 2-D continuum: $\Re^2$
The visual feedback is a megapixel(discrete) simulation of the tactile continuum.

windowing environment $\Re^2$

a mouse cursor in a windowing environment is a tactile interface that involves a realistic access to a 2-D continuum: $\Re^2$
the mouse has buttons that generate Boolean values plus poor visually simulated access to depth

Figure 4.1 Dimensional Analysis of Gestures

4.2 The Syntax of Fields is Differential

Gestures are connected paths in a space. If one views a gesture as a continuous language then the syntax of a gesture is differential. In other words, gestures are composed continuously. The simplest continuous language is the set of real numbers. One way to map the space of gestures which do not cross themselves into the reals is by means of the integral operator in Figure 4.2.

$$\int_g f(g(t),t) \bullet \dot{g}(t)$$

Figure 4.2 Integral Operator for Gesture Recognition

This integral operator maps gestures into the reals by means of a path integral of the inner product of the tangent space of the gesture and a field over the space. This is similar to the way a physicist would calculate the work needed to move a particle in a potential field.

f is the time dependent field, and g is the time dependent gesture. Of course, $\dot{g}$ is the time derivative of gesture g.

Figure 4.2.2 shows a space of gestures being mapped to a set which contains a continuous language. Figure 4.2.1 shows the space of gestures being mapped to a set which is a discrete language.

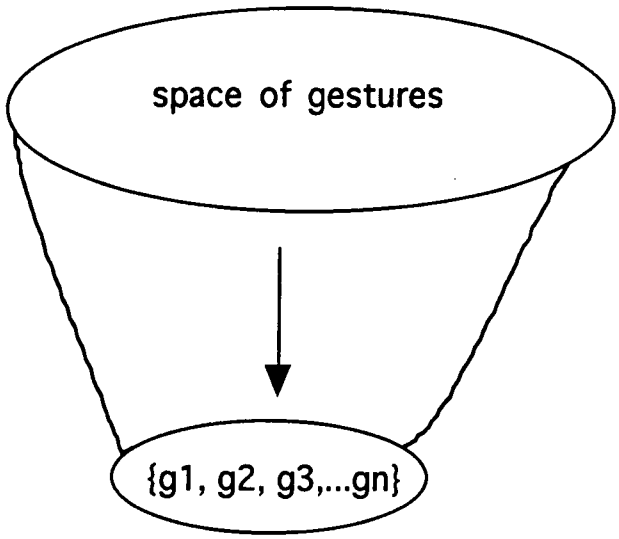


Figure 4.2.1 Gestures as a Discrete Language

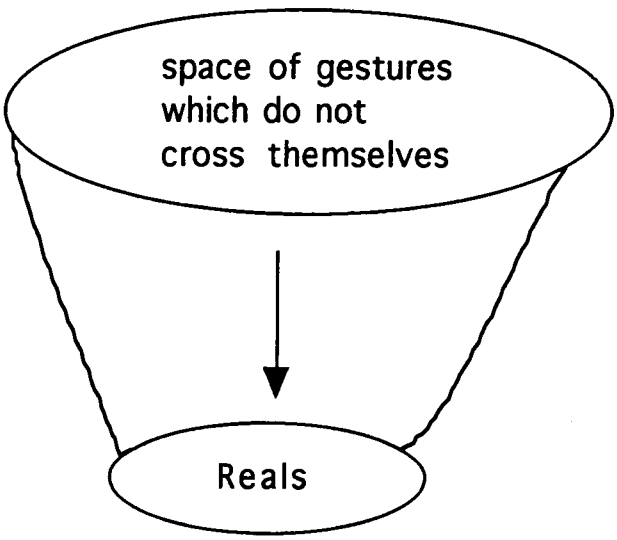


Figure 4.2.2 Gestures as a Continuous Language

Because fields are continuously deformable into one another and the operator output is continuous, this scheme qualifies as a continuous language. This integral operator produces a fuzzy landscape with the help of a field that distinguishes gestures continuously. The operator generates a real value output which measures the continuous response to each gesture.

4.3 The Semantics of Fields are Continuous

Since the syntax of fields is differential, the associatedsemantics are continuous. Therefore, one measures the semantics of field derived interface inputs on continua. For example, 2π encirclements is clockwise and -2π encirclements is counterclockwise. Continuous semantics will be discussed in the implementations section 5.2.

4.4 Classification of Constraints

The recognition of gestures in a virtual world should have the ability to be context dependent. There exist four types of constraints on gestures: hard, soft, informational, and relativistic.

Real human gestures do not exist in a vacuum. Human gestures exist in a context of constraints created by the physical environment. A table, a wall, and a vase all present different types of constraints on gestures.

4.4.1 Hard Constraints: Static and Moving

A wall is a hard constraint. Any gesture is totally constrained to exist in the space outside a solid wall.

Hard constraints include moving solid objects. A person crossing the street must avoid moving cars. When one is hitting a tennis ball, the motion of the racquet is constrained by the time dependent position of the ball. Physical objects have a number of continuous physical properties, including velocity, acceleration, position, and angle of rotation. If one wishes to recognize gestures in a virtual world, one needs to be aware of the constraints on motion. These constraints change continuously in time. Such moving physical systems can be modeled by field computers.

Field computation is a good way to model continuous dynamical systems. Parts of these systems can constrain the motion and state of other parts of these systems. Fortunately the constraints generally occur in a local region of space-time which lends itself to the local nature of field computation. In a virtual world, one needs to model the constraints of the environment on the user's gestures and model the constraints of the user's gestures on the environment. There also exist gesture-gesture constraints. One virtual can not pass through another virtual hand.

Again modelling continuous systems compares well with field computation because data and program are often indistinct in field computation. Gestures are operators on fields but are themselves fields. If two people share a virtual space then the gestures of the first operate on the body of the second and the gestures of the second opperate on the body of the first. Such physical systems can be modeled

well by field computation, in which fields are generated by the placement of objects in the environment to compute responses to gestures.

4.4.2 Soft Constraints: Static and Moving

A body of water is an example of a soft constraint. A gesture can continue in a fluid such as water, but the viscous friction bounds the velocity of the gesture. If too much torque is applied to an object in a fluid, a condition called cavitation is produced. Cavitation involves the temporary destruction of the continuum of the fluid and the creation of a vapor phase.

A soft constraint might include a moving fluid like air or water. When a hand is extended out the window of a moving car, the motion of the hand is constrained in various ways. Certain gestures are impossible, such as holding the flat of your palm facing the fluid source. Also, the moving fluid can impart forces such as lift if you form a properly oriented air foil with your hand. Such moving physical systems can be modeled by field computers.

4.4.3 Informational Constraints

Informational constraints are not physical constraints; rather, they are constraints caused by the semantic structure of the context of a gesture [Kugler 1987]. Kugler also discusses informational constraints. Gestures can gain partial or whole meaning from the orientation to their context. The simplest example is pointing. The existence of

possible objects of interest constrains the directions of pointing, not in an absolute physical sense but in an informational sense. Imagine a person is an unoriented dark space with one dim light bulb. There are only a pair of meaningful directions in which to point: at the light and at the darkness.

There are fuzzy gradations between the light and the dark. However in general, such a total assignment of meaning to the pointing space would be imprudent, because it would be too easy for the gesture recognizer to mistake a stretch or the swing of a hand while waiting for a meaningful command.

Imagine the gesture of a hand waving. This gesture has a number of possible parameters. Examples might be the waving frequency and an average distance of the hand from your body. Its meaning is highly dependent on its orientation. If a person is waving at a blank wall they might be testing their eyesight; a wave oriented at a person is a gesture of parting or greeting.

Consider waving a hand oriented at a moving car. One might be attempting to stop the cars advance. In general one can assume that the orientation of the gesture tracks the motion of an object of interest. Here one has two dynamical systems, the car and the hand. The hand motions are some function of the car's motion. This correlation is meant to communicate some action or information to the computer system. This paper explores the recognition of oriented gestures in a plane.

4.4.4 Relativistic Constraints

A fourth possible type of constraint on gestures is a relativistic constraint. The relativistic constraint on

gestures and the information exchange rate in virtual reality is the speed of light, which is a hard velocity constraint. In other words, interacting objects must exist in each other's light cones.

Relativistic constraints may turn out to be the most important of all constraints in virtual worlds because any network delay in a highly parallel field computer will be limited by the geometry of 4-D light cones radiating from local computational events in the processor space.

Just as human gestures are constrained by gravity in everyday reality, it may well be that the dominant "physical" constraint on gestures in virtual reality will be special relativity.

This speed of light will be in general some fraction of the actual speed of light. The virtual reality speed of light will be a combination of the interprocessor and network delays.

For example, a person working in a virtual world with someone in another continent would be keenly aware of whether the computers were connected by submarine optical fiber or by geosynchronous satellite, the latter causing a much greater delay.

4.5 Continuous Mathematical Models of Gestures

There are two major areas of mathematics that can be used to model gestures: differential geometry and flows. Flows deal with the solution to autonomous differential equations.

This paper will use both methods, but will concentrate on flows. Differential geometry covers nearly every gesture one would like to consider. Sadly, such breadth leaves it little to say about gestures as a whole.

Flows deal with a restricted set of gestures. The gestures that fit a flow model include gestures which do not cross themselves. Even though these flows miss many of the gestures programers would like to study, much can be said about those gestures which do fit. Many gestures which do not fit can be shoe-horned into flows so that more powerful statements can be made. Flows are often described as functions of a complex variable.

4.5.1 Differential Geometry

Humans wish to communicate with machines using gestures. Gestures can be simple mouse drags, or more complicated curves in 3-space. One-dimensional gestures or mouse drags can be described by continuous-one dimensional curves. Simple planar gestures can be described by a curve with a starting point and an end point. This curve can be totally described as one function: the curvature as a function of curve length. For a curve in three space one needs to add one more function to curvature, the torsion as a function of curve length. It is well known from differential geometry that "...two curves with the same curvature and the same torsion are identical with each other." [Aleksandrov, 1963].

Suppose one is given two curves. How can one know if they are the same curves. Assume the curves may be rotated relative to one another in three space. Let s be the arc length. One could approach the exact recognition of gestures(curves) in the following difficult way:

> If we move along the curves to various distances A from our initial points, we will arrive at points where the curvature k and the torsion t will have various values, depending on s. Thus k(s) and t(s) will be certain well-defined functions of the arc length s [Aleksandrov, 1963].

One simply matches the torsion and curvature as a function of arc length from one curve to the other. If the torsions and curvatures always match then the curves match.

Of course, no match in curvature is exact and one would need some compute intensive means of matching curvatures within some range of acceptable values. Most simply, one could minimize the integral of the square of the error in curvature over the length of the gesture.

The problem with cuvature matching is that one needs a finite alphabet of gesture classes with which to compare the input, thus, one is limited to a discrete language of gestures. Because of these disadvantages, this method of gesture recognition was abandoned.

4.5.2 Differential Flows

Flows are the solutions to autonomous differential equations. The Figure 4.5.2.1 is a quote from [Beck 1974].

> Especially important are the cases in which X is a topological linear space or an analytic manifold and in which the function
>
> $$\dot{\varphi}(x) = \lim_{t \to 0} \frac{\varphi(t,x)-x}{t}$$
>
> exists at each point. These are the autonomous differential equations, which we see from the vantage point of dynamical theory as differential flows.

Figure 4.5.2.1 Definition of Flow

One should realize when looking at the equation in Figure 4.5.2.1 that the statement in Figure 4.5.2.2 if also true.

$$\begin{array}{c} \text{if } t = 0 \\ \text{then} \\ \varphi(t,x) = x \end{array}$$

Figure 4.5.2.2 Details of Flow Definition

because at time zero, flow has not had time to move a point to a point different than its starting place.

One can think of a differential flow as follows. If one takes a surface with varying elevation and pours syrup over the surface. Because of the high viscosity, the resulting flow has effectively zero linear and angular momentum. One has low velocities which match the negative of the gradient of the surface. Such a flow is an example of a differential flow. Gradient flows are not the only kind of flows. The following is a formal definition of algebraic flow from [Beck 1974]:

> Definition.
> Let X be a set. A function φ from R x X onto X will be called an algebraic flow in X if it satisfies the following group property:
> $\varphi(s+t,x)=\varphi(s,\varphi(t,y))$ for every s,t in R, and x in X.

Flows are useful in the generation and recognition of gestures. The section about creating and editing fields describes how to generate gestures with flows. Often in differential flows, there exist stagnation points. These are places where the flow is zero and thus there exists a stationary point or point-like orbit.

Def: The orbit of a point p, $O(p)$, is the set of all points that flow can take a particle from point p [Beck 1974].

It is assumed that $O(p) = O^{+}(p)$, where the superscript plus sign refers to positive time evolution. Stationary points are special because they are their own orbit. $O(s) = \{s\}$. Since stationary points are points of zero flow velocity, for gradient systems, the gradient is zero where the surface is either a maximum, minimum, or a saddle point.

At the center of the differential flow in Figure 4.5.2.3 is a stationary saddle point. In non-gradient systems, stationary points can be found at the center of curl fields.

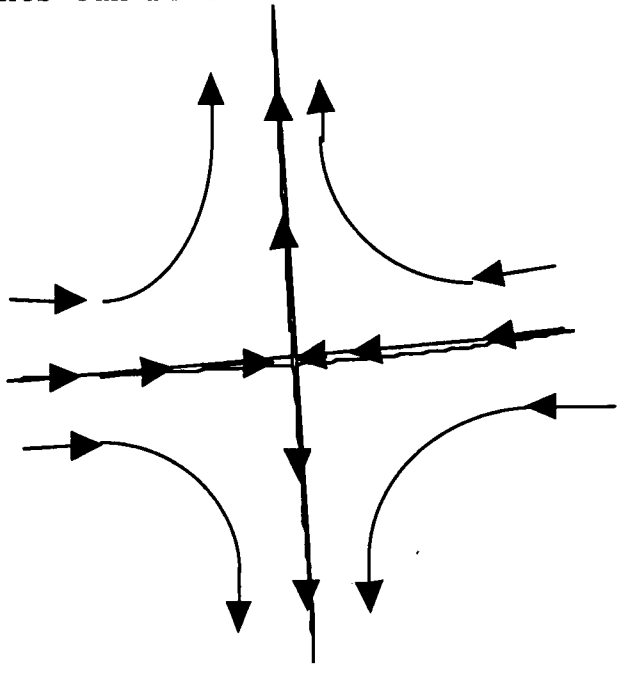


Figure 4.5.2.3 A Saddle Flow

5 Implementations

This paper will apply field derived interfaces to existing types of interface objects before attempting to construct a purely field computational interface. The first subsection discusses both semi-continuous and continuous implementations of menu interfaces.

5.1 Menu Interfaces

This section will look at various continuous gesture-operator spaces including:

1) nearly continuous menus
2) continuous menus

A computer program [Evers 1994] was constructed to demonstrate the menu interface in section 5.2.

5.1.1 Nearly Continuous Menus

Keyboards are discrete interfaces. Menus are semi-continuous (see earlier section on the dimensional analysis of input devices). The segments of menus are continuous rectangular regions. But these regions have hard boundaries. The word "hard" here refers to the hardness of the logic used, not the hardness of a constraint on gesture motion.

This is in contrast to the fuzzy interface that humans have at stop lights. Red, Green, and Yellow, traffics lights present the Stop, Go and We-are-changing-between-stop-and-go-so-change-your-perceptions-accordingly indicators. When a

user pulls on a pull-down menu, the menu selections change discontinuously between one state and the next.

[MacLennan 1994] has described basins of attraction in a field in which there always exist a region: possibly of differential width, but a border which is a non-empty set, where a continuous valued function changes from region A to C through a region which is not A or C. In other words: in between every Green light and Red light there is a Yellow light.

Humans can be sloppy when they use interfaces. It would be nice if the continuum of human gestures was carried through the interface to the computer. Consider the case were a menu selection "green" is catastrophic. Green deletes a file which contains the only copy of a thesis. Red is the choice that the user really wishes: not to delete the file. Given a hard boundary between red and green, a very small jiggle of the hand during a menu choice can cause catastrophe. The chances of catastrophe are lessened by the more continuous gestural interface. If there is a yellow region of finite area in between each region that invokes some action, then accidents can be avoided. This interface is analogous to someone carrying a object, they can feel if their grip is slipping. Likewise, if a yellow region is inserted between red and green, the user can more easily get a grip on what it is she is doing. The appearance of a yellow border bar between each menu selection would indicate to the user that they did not have a good grip on their menu choice. It so happens that the more continuous gestural interface is suggested by field computation. If indeed the human brain relies heavily on field computation, it makes sense that one should make efforts to respect these approaches in the design of human-computer interfaces. With a yellow menu boundary,

the user and the computer know when selections are safe and when they are not. See Figures 5.1.1.1 and 5.1.1.2.

Current menu interfaces often measure the amount of time spent in an item, in order to determine the confidence in that selection. If a minimum amount of time is not spent in the item before release, then the item is not selected.

The field computation compatible menu described above exchanges space for time in the determination of confidence, thus increasing productivity.

5.1.2 Continuous Menus

Now our analysis will move the behavior of menu evocation to the continuum. Since the syntax of fields is differential, one need to construct our menu behavior differentially. One can think of the popping up of a menu as the movement of the gesture(cursor) in the z direction(out of the screen) to a new window whose depth is less than the space that evoked it. It would be nice to confine motion in the z direction to regions of reduced dimension imbedded in the space of evocation.

Branch cut menus work as follows. The menu popping-up is not caused by a mouse click. The mouse moves to the very top of the screen (where the menu bar resides on a Macintosh).

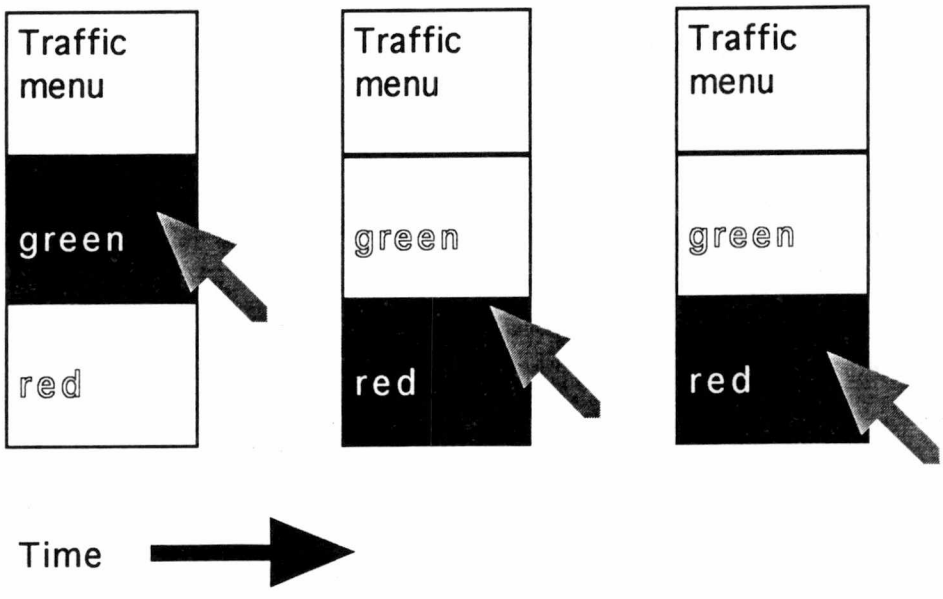


Figure 5.1.1.1 Semi-Continuous Menu Selection

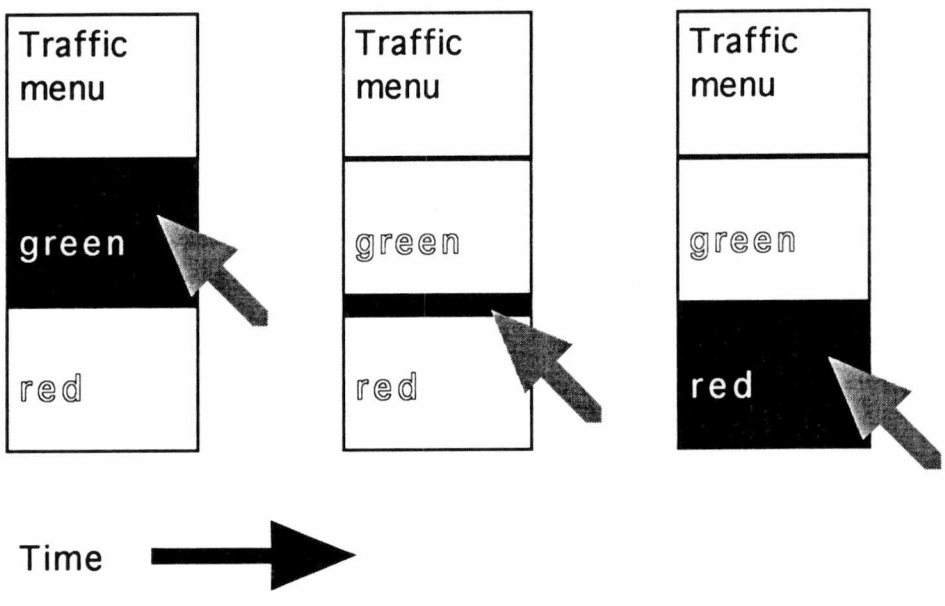


Figure 5.1.1.2 Nearly Continuous Menu Selection

The very top line of the screen is considered to be the location of the linear defect (Figure 5.1.2). While moving upward, the cursor is restricted by the top of the screen. Motion stops on the topmost rasterline. The menu then pops up because the hot spot of the cursor is on the lineal defect. Motion continues downward now on the menu. No mouse button has been pushed. The mouse cursor moves down the menu. At the appropriate item, the mouse exits to the right of the menu, selecting that item, or the mouse exits to the left of the item, selecting nothing.

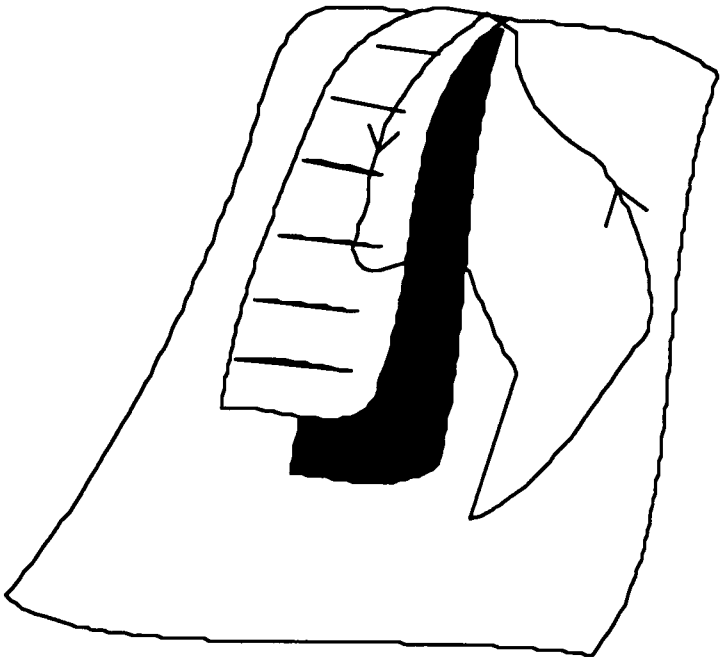


Figure 5.1.2 A Branch Cut Menu

Upon leaving the menu rectangle, the menu disappears and the mouse drops back onto the desktop and moves regularly.

All flow is strictly into the top of the menu and out of the bottom and the sides. A branch cut menu can be seen in Figure 5.1.2.

The menu has been constructed out of linear defects(folds) in the underlying space of evocation. Section 5.2.1 of this paper describe how icon type behaviors can be created with point like topological defects in a space of evocation.

Flows can be well defined because of the differential structure of the lineal defect. Once interfaces are described in terms of flows, gestural interfaces can be more easily built from various topological building blocks. One can also more easily automate (construct macros for) such interfaces. The automation of user gestures allows for automated testing of interfaces and programs.

5.2 Continuous Gesture-Operator Spaces

This section will look at various continuous gesture-operator spaces including:

1) Selection/Encirclement of an icon
2) The log-shell space
3) Conformal maps of non simple response regions
4) Toroidal spaces
5) Hyperbolic spaces

A computer program [Evers 1994] was constructed to demonstrate operator spaces described in sections 1, 2, and 5.

This section describes an approach to the design of visio-tactile human-computer interfaces from a topological point of view. It considers various screen topologies. Usually screen topologies are flat Euclidean rectangular

bounded spaces. The mouse cursor moves in a rectangular space but is constrained from moving out of the screen boundary. Humans kinesthetically think in terms of topology, such as holes, edges, and folds. Why not make use of this natural inclination in the design of human-computer interfaces?

5.2.1 Selection/Encirclement of an Icon

One can combine traditional desktop interfaces filled with icons with field computation derived gestural interfaces in what appears at first to be an impure way. Some of the nice properties of fields fail when fields contain actual singularities. Sometimes the properties of singularities in a field are to our advantage when one is trying to bridge the world of the discrete to the world of the continuous. What I will show is that one can use a finite valued field to simulate a field with singularities and hence get the best of both worlds: singularity type properties from finite valued fields.

An icon is located somewhere in the complex plane (the desktop). The icon is at location p. One now constructs a field, f, over the complex plane, which has a pole at p.

There are a number of tricks that one can use to deal with point p. The simplest is to add some very small irrational number (like $\pi/10\hat{}6$) to the coordinates of p, moving p in between the cracks of any chosen mesh which approximates the continuous field f. Now the problem of real infinities causing a possible floating point error is avoided.

There exists an integral operator, ψ, which measures encirclement locally. ψ is the imaginary part of the integral of any gesture over the field f, which has a first order pole

[Kreyszig 1983]. There is a continuous map between the imaginary part of the path integral of g in f and the number of encirclement about p. The gesture, g, starts at a mouse down, d, and ends at mouse up, u. The reason that this scheme works is that a topology has been constructed which is measurable, a punctured plane. The first order pole in the complex plane creates a Riemann surface which can be used to build an operator that measures encirclement. Figures 5.2.1.2 show computer screen of the program that uses the integral operator described in Figure 5.2.1.1.

$$f(z) = \frac{1}{(z-p)}$$

$$\text{encirclements} = \psi(f(z),g(t))$$

$$\psi(f(z),g(t)) = \text{imag}\left(\int_g f\, dz\right)$$

Figure 5.2.1.1 Encirclement Equations

In this experimental interface, p was placed at the origin. Successively dragging the 'z' icon around the origin caused the imaginary part of the path integral to increase (see Figure 5.2.1.2). This encirclement quantity is represented by the position of the circular icon climbing the imaginary axis.

The best part of this whole scheme is that even though the field does not have a "real" pole anywhere, the path integral behaves properly because path integrals are local computations. The field "tells the truth" almost everywhere in the sense that the Labegue measure of the set of points in the field that lie about the true function value is 0

and the set has a counting measure of 1. The field combined with the operator measures what one wishs to measure about the gesture, encirclement. Encirclement is a natural way to select things [Spencer-Brown 1974].

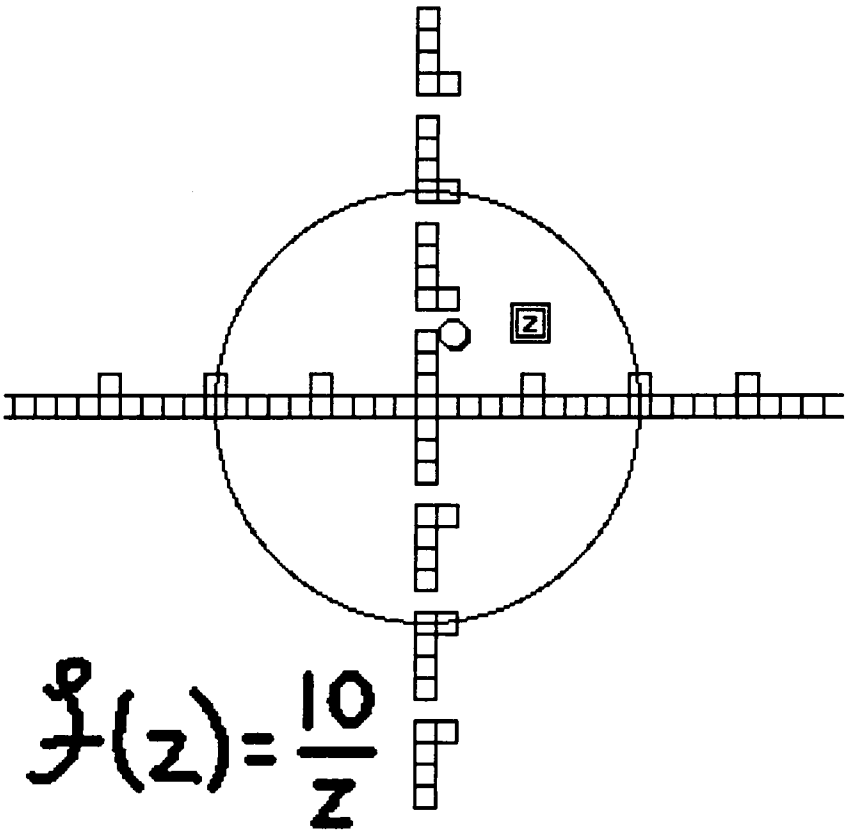


Figure 5.2.1.2 Encirclement Program Screen

Another attribute of this scheme is that it scales well. One can add any number of processors with their own batch of icons(poles). The input device can broadcast the stream of differential gestures to all processors. Each processor (knows) how much each icon has been selected and can send

relevant visualization information to the video output stream. A computer program was constructed to demonstrate this operator space.

5.2.2 The Log-Shell Space

Consider a circular logarithmic space shown in Figure 5.2.2. The author calls it a log-shell space.

In this interface [Evers 1994] the user drags an object to be modified. The object could be an icon of a scalar, or a field. The log-shell is an oriented topological space. One drags the object around the space to change some attribute of the object continuously.

Assume one is in a base-10 log-shell space. If one wishes to divide the values contained in an object by ten one does the following:

1) Drag the object into the space radially.
2) Circle the center of the space once clockwise.
3) Drag the object out of the space.

Fortunately, encirclement is easy to compute via path integrals around first order pseudo-singularities in the complex plane. A computer program was constructed to demonstrate this operator space.

5.2.3 Conformal Maps of Response Regions

Using conformal maps of the complex plane, one can define complicated planar regions which respond to gestures with high precision and without using digital regions [Apple 1985].

If one picks a rectangular region in the complex plane and applies a conformal map to it, as long as the function one applies is analytic, the resulting region will have non zero area. Because conformal maps are continuous, it is safe to use them to convert one continuous language-recognizing field into another.

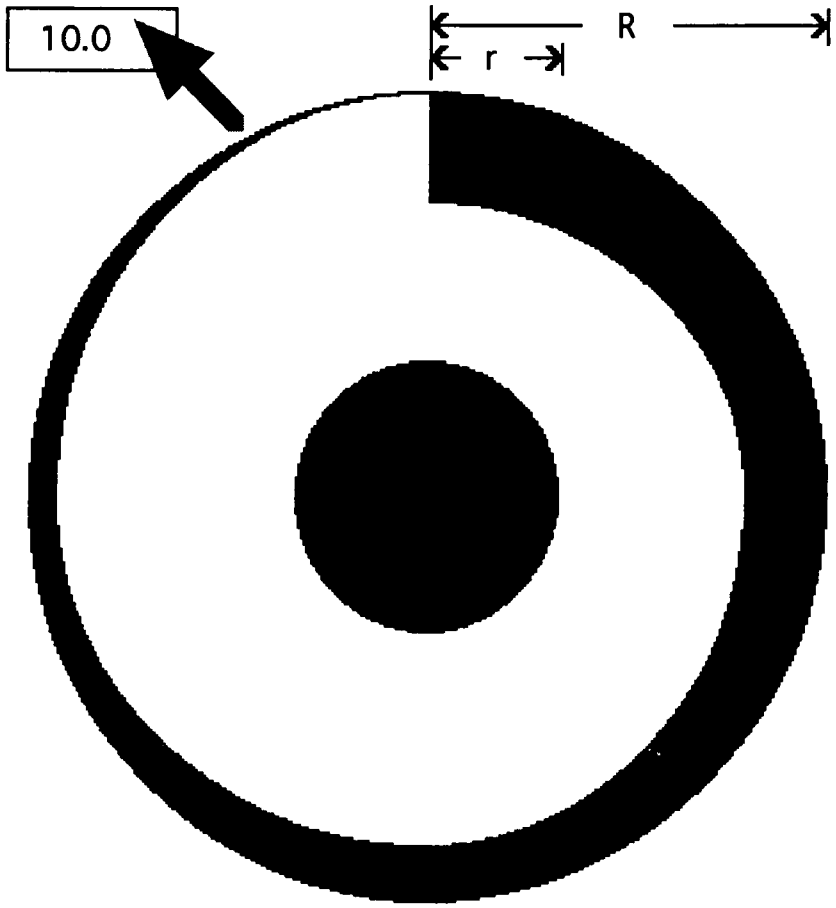


Figure 5.2.2 The Log-shell Space

The Riemann mapping theorem implies that any connected shape with area can be mapped into a circle by a conformal map. In practice it is often hard to find such maps. In terms of gestural interfaces, polygons, curvilinear polygons and

nearly circular domains are sufficient for almost any need one could conceive. Fortunately, polygons, curvilinear polygons and nearly circular domains can all be mapped into either a circle or a rectangle by means of well known conformal maps [Nehari 1952].

These maps are useful because it is computationally simple to check for mouseEnter, mouseLeave, and mouseNear (distance) events for circular regions?

For example, say one has a region in space shaped like a star. How does one program the response of the system upon the approach of a gesture towards the star shaped region.

First, the user needs to describe the system response of a gesture approaching a circular object. To do this the user can draw the response of the system, I(r) on a graph of I(r) vs. r (See Figure 5.2.3.1). r is the radial distance from the center of the circular response

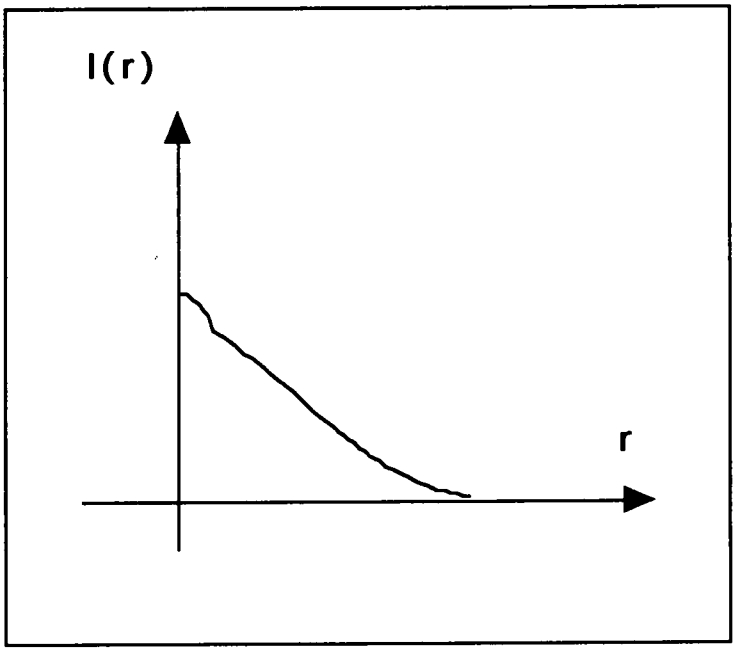


Figure 5.2.3.1 User Drawn Radial Response Curve

Second, the user can select from some previously defined shapes. Ultimately, one would like a visual programming environment that allows the user to draw a response region of any shape constructed of non-overlapping straight lines or circular arcs. Suppose that the user selects a star shaped response region. The interface is now complete.

The system applies a conformal map, F, to the gesture, g(t), which maps stars to circles (see Figure 5.2.3.3). Now the system can directly compute the response of the system, which is a function of the distance of the gesture to the star's(circle's) center. The conformal map from a star shaped region to a circular region, F(z) is shown in Figure 5.2.3.2.

where

$$F(z) = \int_0^Z \frac{(1-z^5)^{2/5}}{(1+z^5)^{4/5}}\,dz$$

Figure 5.2.3.2 Conformal Mapping Function of a Star to a Circle

If the response of a circular region is I(r) at radius r then the response of the system to a gesture is:

$$R(t) = I(|F(g(t))|).$$

Note that the response of the system, R(t), is a function of time.

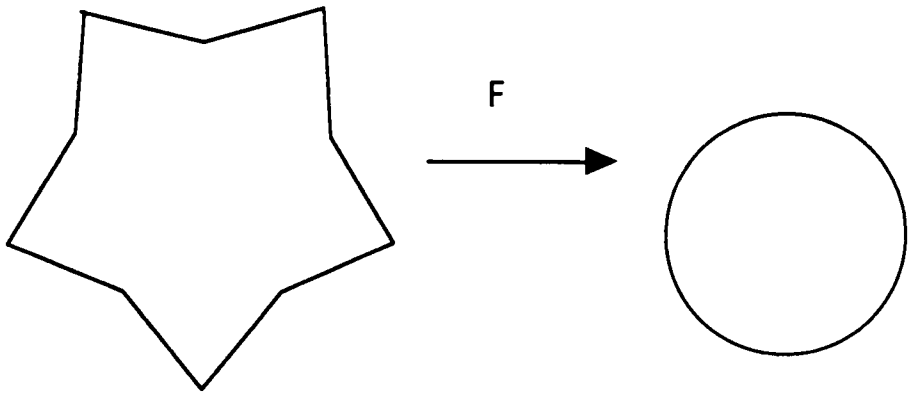


Figure 5.2.3.3 Conformal Map of a Star to a Circle

5.2.4 Toroidal Spaces

A toroidal space is another example of a topological gesture space. The left side of the screen is connected to the right and the top to the bottom, forming a torus. A torus is nearly as easy to implement as the bounded rectangle and has advantages. It takes only half the time to drag an object to the other side of the screen on a torus as shown by the movement of the black ovoid in Figure 5.2.4. The movement of the white ovoid takes twice as long in a bounded rectangular space.

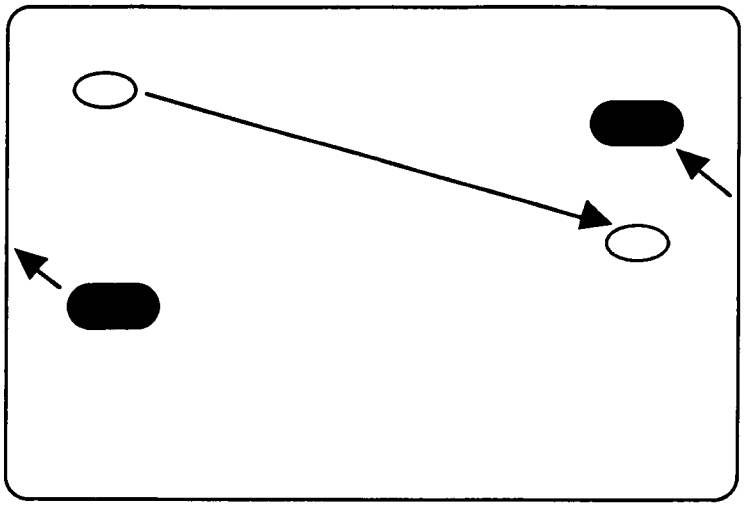


Figure 5.2.4 A Toroidal Space

5.2.5 Hyperbolic Spaces

There are other stranger topologies which could be useful as visio-tactile interfaces. Consider a hyperbolic space used as a desktop. This is a space that is bounded but infinite. Such a space makes possible a visio-tactile dialog with infinite data structures in a finite space in a way that is intuitive to humans. It is also natural for a field computer because the space has a well-defined metric. Such data representations also parallelize well.

Objects shrink as they are dragged towards the edge of the screen. Here one can see a field computational way to implement continuous iconification.

A computer program [Evers 1994] was constructed to demonstrate this operator space. A screen dump of "hyperbolic" can be seen in Figure 5.2.5.

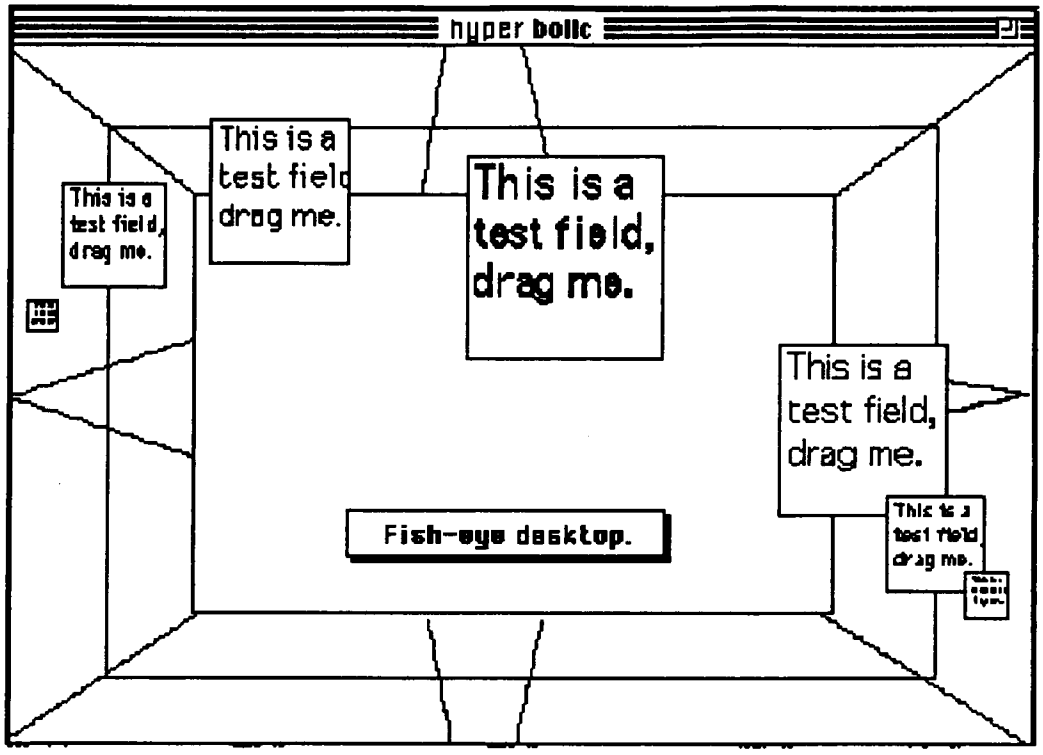


Figure 5.2.5 A Hyperbolic Space

6 Creating and Editing Fields

One would like to create and define energy surfaces by some type of visual programming. Various methods suggest themselves:

1) Trace equal energy contours.
2) Position energy surface minima and maxima,
   (a simplified version of method 1)
3) Draw shadows
4) Draw highlights
5) Draw paths that minimize or maximize certain path integrals(distance, energy, time average velocity etc.)
6) Sculpt the surface directly if one has a data glove or another 3-D rendering surface manipulator.
7) Sculpt by dragging a volume integral surface over the target surface, convolving as one goes.

The first section of this chapter explores an implementation of method 2. The second section discusses method 7. A computer program was written to show method 2 [Evers 1994].

6.1 Encoding Fields by the Placement of Minima and Maxima

In order to define energy surfaces by visual programming, one can position energy surface minima and maxima. This can be accomplished by using icons to represent the position of pseudo-singularities in a potential field.

Some of these singularities are positive valued(sources), and others are negative valued(sinks).

In Figure 6.1.1 lines show the flows of autonomous differential equations in the plane.

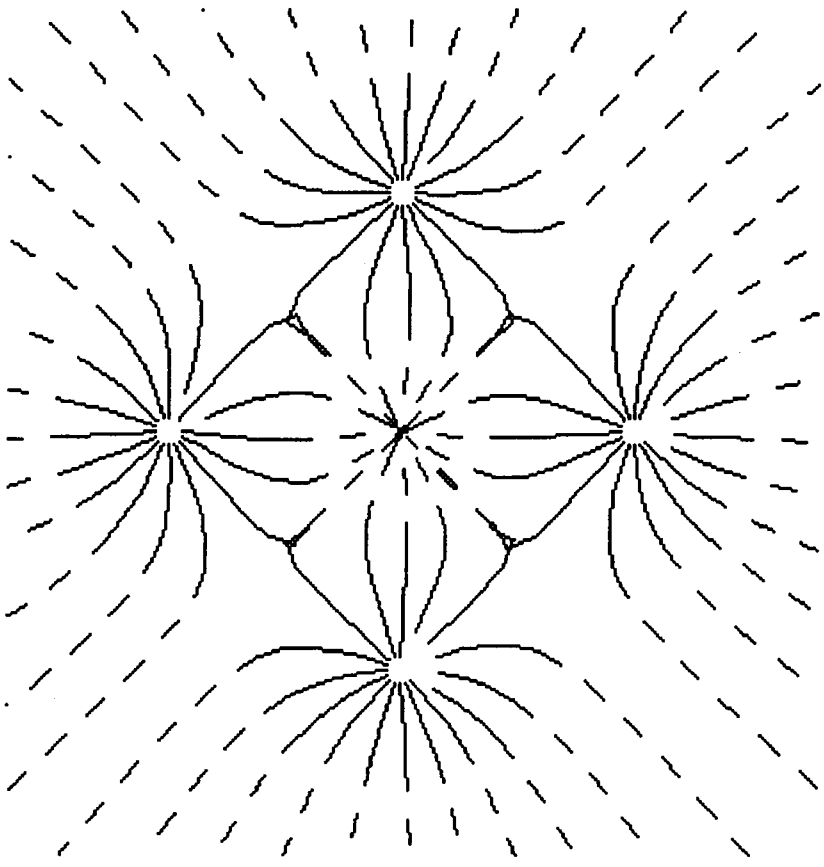


Figure 6.1.1 Flow Lines of a Flow Field

The solid lines show positive field potential and the dashed lines show negative field potential. At the center is an attractor. The four surrounding centers are repellers(poles). Time positive flows send particles from repellers towards attractors. One can think of the four repellers as islands and flow as the movement of water off the islands into the sea, solid lines being above sea level and dashed lines being below sea level. Time negative flow is

the reverse. Equations that describe the flow is shown in Figure 6.1.1.1.

$$F = \frac{\prod_{n=1}^{1} (z - q_n)}{\prod_{m=4}^{1} (z - p_m)}$$

$$\text{flow: } V = \text{grad}(\log(\text{abs}(F)))$$

$$v = \frac{V}{|V|}$$

Figure 6.1.1.1 Field Computer Flow Equations

The q's are the complex values of the coordinates of the zeros of the potential field F. The p's are the poles. One takes the absolute value to make all flow towards the zeros and away from the poles. The log then converts the zeros into negative poles but leaves the positive poles unaltered qualitatively. Next one takes the gradient to construct the actual autonomous flow. Then one normalizes. Normalizing actually destroys some of the nice mathematical properties that V had, but is necessary to make the flow useful on an computer that does not actually compute with continua. One of the nice properties destroyed by normalization is the smooth convergence of trace particle motion.

A field programmable planar data flow computer can easily be implemented with the building blocks of poles and zeros and a little extra code, including a probability field editor.

In the experimental interface program, the poles and zeros can be dragged around the space with the mouse. There is a trace particle which can be inserted into the flow. Time

can be reversed. Also the zeros (sinks) can be programmed to "absorb" the trace particles within a certain neighborhood and then force the emission at some angle from a pole(source) to continue flowing. This is a way to visually program a field computer to automate drag and drop type interfaces.

In figure 6.1.3 one can see poles and zeros grouped into dipoles. q1 is above p1, q2 is above p2, and q3 is above p3. The emission properties are shown in the programmable interface seen in Figure 6.1.2. One can see the way flow is connected in a single dipole pair in Figure 6.1.4. This program shows how one can program a field computer with topological defects which is reminiscent of the branch cut menu.

The emission probability editor is used to program the exit angle of trace particles(Figure 6.1.2). The effector on the right, the radial dirac delta function dial, sets a single deterministic exit angle. The editor on the left of the emission editor is more subtle. It allows one to sculpt the probability distribution of the exit angle in greater detail. In Figure 6.1.2 one can see that a user has used the editor on the right to choose an angle of $\pi/4$ which is reflected in the probability distribution on the left.

- help button: gives you this help text,
  click this text to make it go away
- round button hanging under the pn: shows direction
  of flow out of the bottom of pole pn
- square buttons in graph: define the probability
  distribution of the flow out of the bottom of pn
- finished editing distribution button: sends you
  back to the dipole flow screen [Evers 1994]

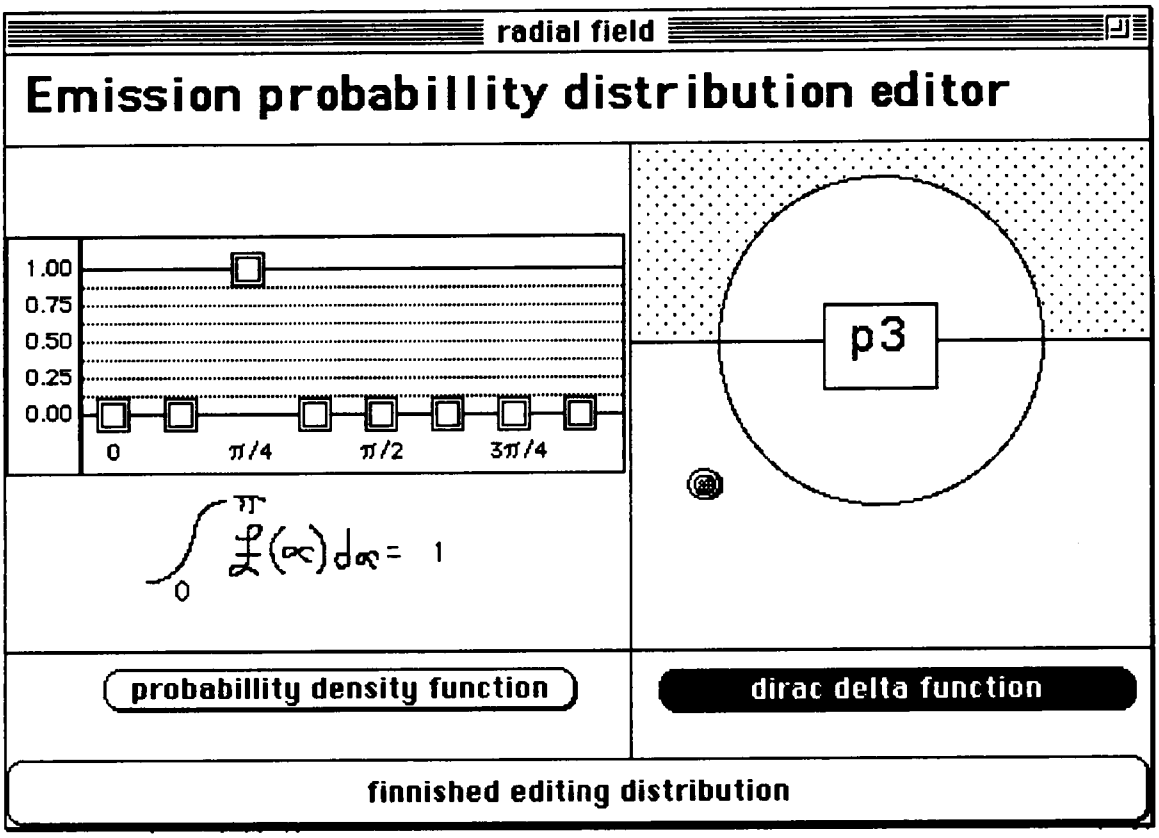


Figure 6.1.2 Emission Probability Editor

- help button: gives you this help text, click this text to make it go away
- reverse time button: reverses the flow
- animate button: toggles the motion of the trace particle
- send update button: tells the trace particle to move once
- find trace particle: places the trace particle on the screen (in case it gets lost off screen)
- double click on one of the "p" icons to go to its probability editor screen [evers 1994]

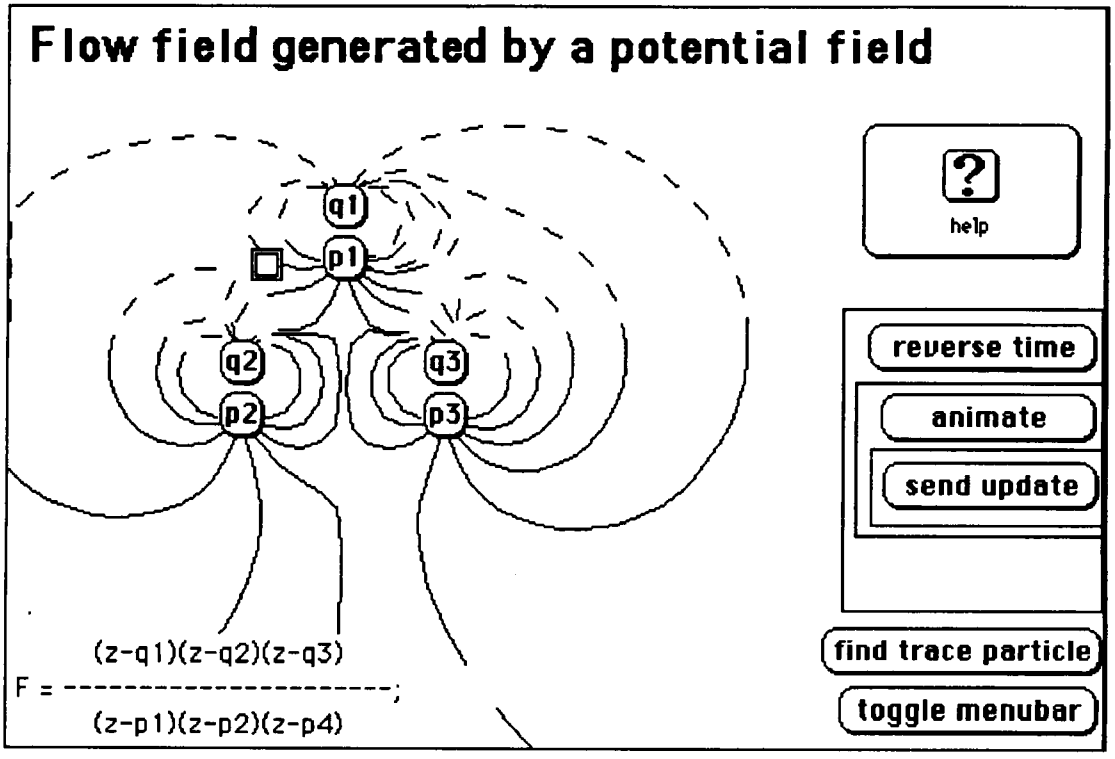


Figure 6.1.3 Flow Field of Three Dipoles

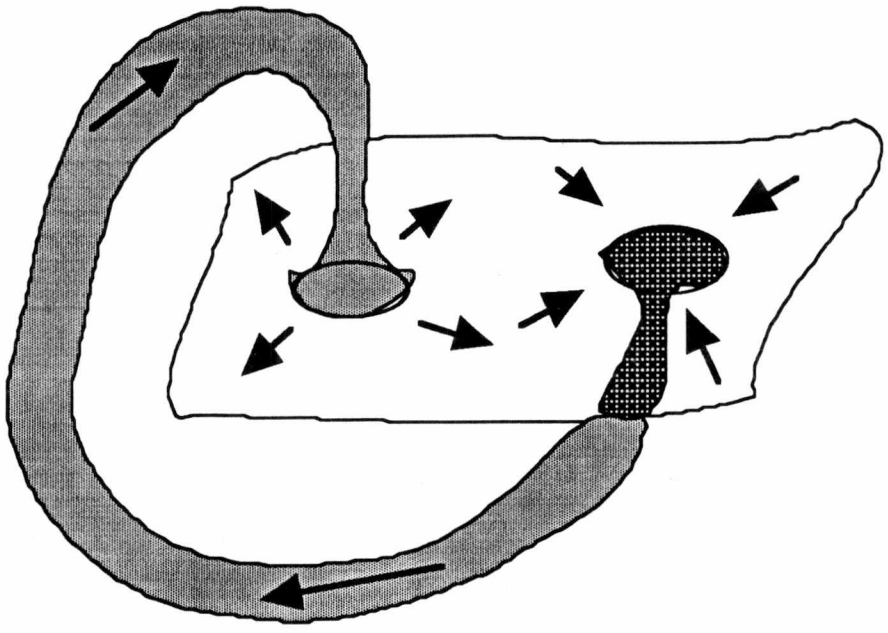


Figure 6.1.4 Flow Near a Dipole Pair

6.2 Drawing Energy Surfaces

Encoding or programming a field computer requires the ability to sculpt an energy surface. Consider a blank flat surface. To sculpt this surface according to the laws of field computation requires operators which are continuous. The sculpting tool of choice is an air brush-like device.

A differential gesture is an example of a continuous field operator. "Pen down" can simply be a command to focus on a coordinate rather than a whole field and then increase the air brush volumetric flow of "ink" from an asymptote of zero to some non-zero value. See Figure 6.2.1 for a graph of the volumetric flow of ink after the initial mouseDown. The mouseUp happens at the maximum of the curve.

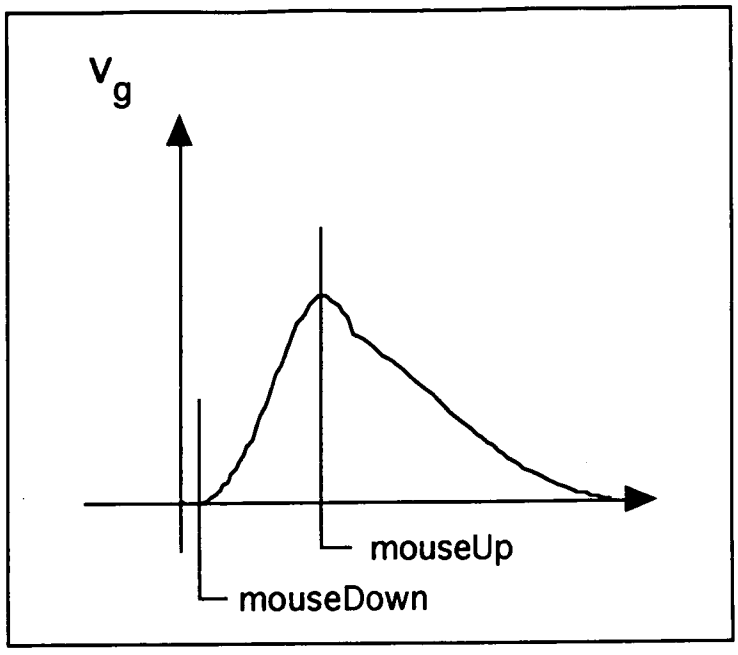


Figure 6.2.1 Air Brush Flow as a Function of Time

The air brush can not be turned off instantaneously because an instant cessation of flow is not continuous function. But its volumetric flow can be brought to 0 in a finite amount of time asymptotically.

A mathematically well behaved differential gesture surface operator would be a function with a total area equal to 0.

One can view differential gestures as differential field operators and therefore as differential fields. These fields need to be of the same dimension as the field one is operating on so that the operation is continuous.

Such a function (as seen in Figure 6.2.2) is an example of a continuous version of competitive learning. Neural connection weight is shifted around but the total weight, c, over the surface, S, remains constant. This is wave encoding

or wave sculpting because it is the sweeping of a zero area disturbance in the memory surface.

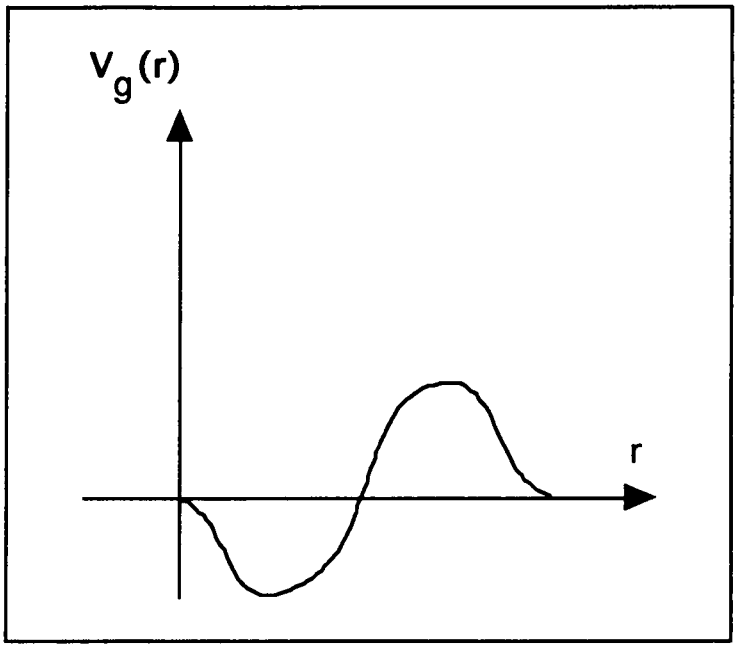


Figure 6.2.2 Continuous Competitive Learning

Therefore,

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} S \, dx \, dy = c$$

is true before and after all differential gestures.

Of course, it might be nice to make the memory surface itself a zero integral surface, which allows one to unify the mathematics at two different levels:

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} S \, dx \, dy = 0.$$

Here S is the energy surface that one is sculpting with our waves. However one moves the wave through the surface or scales it, the integral of the energy surface is unaltered. Also, because of the choice of a continuous wave shape, and the choice of a continuous amplitude as a function of time, the resulting energy surface will be continuous.

One can think of the wave as an abstract airbrush spray distribution, where half of the spray deposits ink and the other "half" removes ink from the surface. The ink is "deposited" in a continuous fashion. The nice thing about using 0-integral energy surfaces is that they can in turn be used as "air brushes" to sculpt other energy surfaces. One can add various 0-integral fields to create new ones for new purposes.

7 Conclusions

To build a gestural interface derived from field computation requires three things: a topology (a manifold), a field over that manifold, and field operators. As a gesture moves over the manifold, the gesture and the field are combined by the operator to compute the continuous system response. It has been shown that all three can be combined to construct gestural interfaces.

This paper shows examples of useful topological manifolds in interface design.

This paper explains ways to generate useful fields from desktop objects and describes ways to edit fields by operating on fields with other fields.

MacLennan has previously shown that one can define operators by composing operators [1979].

All the elements necessary to build field computation derived gestural interfaces have been either explained or demonstrated with computer programs. It is hoped that these interfaces are likely to be used to solve the serious software problems that are likely to frustrate the construction of large virtual worlds.

7.1 Field Computation Derived Interfaces

Field computer derived interfaces allow humans to communicate with computers in ways that meets the requirements of parallel speedup, scalability, consistency, robustness, and absolute rapid response.

These points have been argued and example interfaces provided. In particular, a real number editor utilizing an

encirclement interface was prototyped. Also, a spatiality continuous data flow field computer was prototyped and discussed.

7.2 Designing Field Interfaces

In general, topology represents types of gestural constraints. Continuous topology deals with connectedness and gestures are connected curves. If one knows the types of constraints one wishes to impose and compute, one can design a suitable manifold-topology. Once one have the correct manifold-topology, one designs fields over those manifolds with a suitable structure which can measure the semantics of continuous gestures.

7.3 The Meta-Design of User Interfaces

Most humans have severe limitations on the number of things they can keep in mind at the same time. Assuming one can construct an expressive fast interface, one would like to make possible the maximal externalization of human thought. Direct interfaces which use tactilization and visualization in a unified fashion can help externalize human thought. Given the goal of externalization one designs interface manifold-topologies, fields and field operators accordingly.

8 Directions for Future Work

8.1 Field Computation Interface Standards

The software for field computation derived interfaces needs sets of standards associated with them so that they can be consistently combined with various types of input devices and operating systems.

8.2 Semidirect Interfaces Via Dynamical Systems

There exist ways to improve the match between the field computers models and interface models. Often field computers model dynamical systems. Dynamical systems are not totally passive things. If left in a state that has non-minimal potential energy, the system continues to evolve.

Unless a ball has zero kinetic energy, and is resting on a locally flat area, it will continue to roll on a curved surface until it is in such a state. One could use the mouse cursor position relative to the ball as a means to apply a force to the ball. The ball would roll over a continuous surface, perturbed by the shape of the surface and the input force generated by the cursor. One might have multiple balls (roughly simulating a fluid) in which this same interface applies. One could let the force be inversely proportional to the distance to the mouse cursor. This would have the effect of the balls being like steel ball bearings and the mouse cursor being a magnet. In this way one sculpts the field of ball bearing density and velocity. In a different mode, one

might sculpt the field of hills that support the ball bearing. See Figure 8.2.

In field computer terms, the surface is a field computer program. The ball is a discrete approximation to a fluid field and is the data. But as one can see, program and data and rather indistinct in field computation. The attractive force as a function of distance to the mouse cursor is just another continuous 2-D field which is radially symmetric. The attractive force, which is a continuous function of the radial distance from the cursor, can be visually input by drawing on a graph.

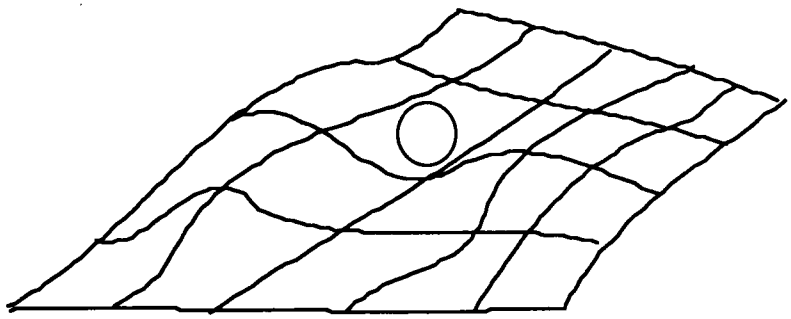


Figure 8.2 Semidirect Interfaces

Such a dynamical system model of human-computer interaction is interesting for a number of reasons. It approximates real life experiences closely, which implies user friendliness. A rapid feedback loop exists between user and computer, which aids user concentration and comprehension of cause and effect during interaction with the program.

8.2.1 Soft Constraints as Semi-Direct Interfaces

One can encode soft constraints into a potential field. If a surface was scored with various types of valleys and basins, it is a model that generates a system of soft

constraints. Soft constraints (see section 4.4.2) are tactile pathways that are not absolute constraints. See Figure 8.2.1. A surface molded thusly makes it easy for the user to drag along the valleys of a surface. Only with great effort can the valleys be escaped. This type of valley following could be useful at constructing computer-aided instruction. Valley following can be used to train humans for various skilled activities that require materials processing and involve specific continuous two or three dimensional gestures. Particular examples include the training of potters, athletes or welders.

One must realize that in a fully field computational model, the machine state changes continuously. Continuous state changes may add time to task completion, however it also adds robustness to the interface, since a task that takes more time allows the user more time to consider aborting the task, or making necessary preparations for the task to be completed safely.

8.2.2 Single User Visio-Tactile Dialogs

The best attribute of a system of soft constraints is that the computer and the user can carry on a continuous visio-tactile dialog. Information is being continuously exchanged between the user and the field computer. If the surface changes its shape as the user navigates the surface, there exists a means of communicating a 2-D field continuously to the user in real time. Ultimately one should like to have a viscosity interface in which one has 3-D vectored force feed back to ones dataglove from the computer. This could be effected by means of magnetic fields imparted on a thin, ferrous, chain mail data glove. Hardware

implementations include a magnetic suspension system or having puppet-like strings on the human appendages to apply torques to surfaces.

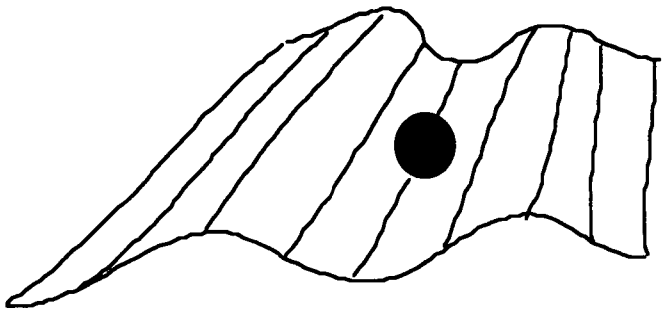


Figure 8.2.1 Soft Constraints as a Gestural Interface

8.2.3 Multi User Visio-Tactile Dialogs

Continuous visio-tactile interfaces are also useful for virtual worlds with multiple users. Current interfaces force the user to drag an object around in a strict sense. A mouse down on an icon makes that icon move strictly with the velocity of the cursor until a mouseUp event. But if forces are exerted by cursors at a distance from the icon in a way that is a function of the distance, then multiple users can exert simultaneous forces on the same virtual object, thereby allowing these users to carry on a visio-tactile dialog with one another. Also, force at a distance helps keep the object of interest from being obscured by multiple cursors.

8.2.4 Force Interfaces

For the Macintosh, and other systems, the mouse cursor responds not only to position but also acceleration.

Acceleration (Newton's first law) is proportional to force and force is the way humans tend to interpret acceleration because of the inertia of the arm, hand, and mouse.

$$F = ma$$

Purely accelerational interfaces exist. Examples include video games like "Gravitar" and "Crystal Quest". Acceleration based interfaces can be more useful because they communicate signed real numbers rather than the absolute real numbers of position and distance. For Instance "Crystal Quest", maintains a velocity vector for the motion of the spaceship in a gravitationally flat vacuum. The force applied can change the direction of motion by imparting acceleration. Such an interface can simultaneously register various combinations of two vectors: velocity and acceleration. Such a system is potentially more expressive because more information is retained.

Acceleration/force systems also allow an intuitive interface for the perception of path and time integration, which can be of use in the design of interactive scientific visualizations.

8.2.5 Momentum Interfaces

Traditional mouse systems allow only the input of velocity. In other words, the velocity of the cursor is mapped to the velocity of the window scroll. It might be nice to give the scroll tile of a text window a certain amount of momentum. Once pushed, the scroll tile would coast in either a dissipative or frictionless manner, thus giving the user full attention to browse at low or high speed. If one makes the mouse impart a force to the scroll tile, the mouse

velocity would control the acceleration of the scroll tile. Needless to say, smooth scrolling of text in a window is more compatible with continuous field computation derived interfaces.

8.2.6 Acceleration Interfaces

Gestural interfaces can be literal in the sense that mouse motions are linearly mapped into cursor motions. Often a linear mapping is at first intuitive, but other mappings suggest themselves. On the Macintosh, the location of the mouse cursor is not a linear mapping for high mouse velocities. It is a mapping of both acceleration and velocity into velocity. In other words, a high mouse acceleration will add to the mouse cursor velocity in a non-linear way.

If the mouse acceleration is above a certain threshold, the cursor acceleration is doubled. If $dv/dt > T$, effective cursor displacement dv/dt is replaced by $2\ dv/dt$. One can demonstrate this on a Macintosh and other systems by moving the mouse slowly from the left of the mouse pad to the right and noticing the screen distance traversed. Now return to the starting position and move with a large acceleration. The mouse acceleration is added to the cursor velocity in a nonlinear manner.

There have been experiments with Newtonian types of cursors. An example of such for the Macintosh was constructed in 1989 by Charlie Reiman, aptly named "AutoCruise". Reiman's experiment gave the mouse cursor a fixed mass, and mouse velocities imparted forces on the free floating cursor. Interesting aspects of this type of interface include the ability to impart a particular velocity to the cursor so it continues to travel in a non-dissipative fashion. For instance, when operating the scroll

bar of a text window, a small velocity combined with a mouse down is sufficient to slowly browse a large piece of text without extra mouse motions. Of course it is difficult to make the initial cursor velocity perfectly vertical, leading to the inevitable wandering of the mouse cursor away from the sliding browse tile menubar. Consider a solution where the browse tile rests in a trough, previously described in this paper as a soft constraint to motion on a potential surface. The mouse cursor may be confined to the tile trough like a bowling ball is confined to a gutter.

Of course, the object oriented possibilities are endless. One could simply give the browse tile itself Newtonian behavior, then one is free to do other things with the mouse cursor while the text is rolling in a window.

8.2.7 Groupware

In terms of designing groupware, Newtonian gestural interfaces have some advantages over other types of interfaces. Such interfaces allow realistic interactions between cursors controlled by different users in the same virtual space.

Scenario #1:

There exists a **file** icon, which user Jane wishes to drag and drop onto some type of **sort** icon, perhaps shaped like a sausage grinder because it does a destructive sort of the input. But user Albert does not wish this file processing to happen. He can drag his cursor with greater velocity (force) on the **file** icon thus preventing Jane's efforts long enough for Jane to realize that someone somewhere does not wish the action. The tactile dialog just described between Jane and

Albert suggests that forces should be continuous vector additive in Newtonian interfaces.

Scenario #2:

Also consider that Albert is new to virtual reality and is clumsy with his data glove. Jane can lend a hand by steadying the file cabinet as Albert attempts to move it. There also exists a way for the machine to guide Albert's hand. One can use 3-dimensional potential streams or gutters to guide the file cabinet.

Scenario #3:

One wishes multiple users be able to visualize and interact with the processing of a data file. Let gutters exist in a potential surface. These gutters can be a way to tactio-visualize the processing of data. Like a river in a river valley which frees silt as it flows, a series of (icons) balls roll down the gutter. These gutters are a very quick and natural visualization describing how much progress has been made, because the human visual system is particularly sensitive to motion. Best of all one has a unified visio-tactile interface. Objects to be processed are placed in the input mouth of the trough. As computation proceeds on the data object, the object moves down the trough. Upon the completion of computation the object exits the trough. See Figure 8.2.7.

Of course gutters can be of a higher dimension. Flow can be unidirectional along a curve in three space. The curve itself changes shape as time passes. Once an object is pushed into the gutter, it moves at a rate which is some function of the arc length. A high dimensional gutter is able to communicate much information to the user.

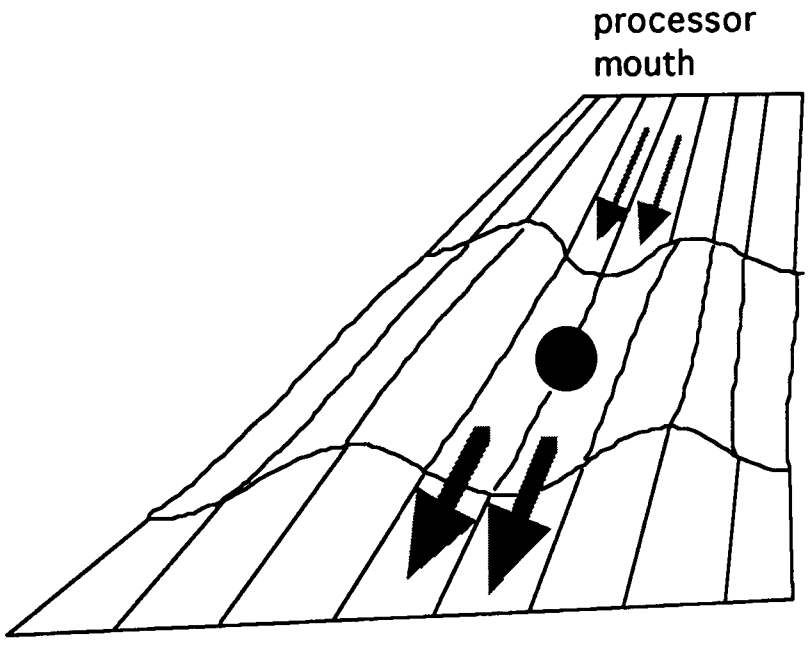


Figure 8.2.7 Data Processing Tactio-Visualization

8.2.8 Advantages of Semi-Direct Interfaces

This section describes a nontraditional interface for moving a rectangular object on the screen. Clicking the object that one wishes to move, one selects the object. Once it has been selected, one moves the pointing device away from the object. Now object movement is generated by a force which is proportional to the distance from the pointer, just as if the pointer and the object were connected by a linear spring.

This, in a number of situations, is a superior method to direct manipulation. For example, if one is running a program remotely over a network, then delays in the update of the rectangle position cause the cursor to leave the rectangle, causeing a frustrating disruption of a direct interface.

Soft effectors such as linear springs and gravitational wells degrade gracefully in the face of network applications

whose synchronous and asynchronous responsiveness depend on network traffic. Semidirect interfaces can also facilitate particular types of popular interface techniques, such as drag and drop.

Soft constraints allow humans to interact in a quite natural manner. Humans attempt to maintain homeostasis. Rapid visual feedback allows the closing of the feedback control loop, which in turn encourages the taking of risks and the subsequent completion of more complex tasks. [Thimbleby 1990] Users seek homeostasis with regard to the risk of operating on data in a computer through an interface. There is homeostasis (balance) between risk and productivity.

Dynamical system type interfaces provide continuous feedback and allow humans to build a more intuitive understanding of the processes which exist through the interface. The discrete knowledge of the process is united with the continuous skills of operating the interface, leading to more user confidence in the process. According to [Thimbleby 1990], greater confidence results in greater risks taken with the data, resulting in greater productivity.

REFERENCES

REFERENCES

Doctoral Thesis and Technical Reports

(MacLennan, 1987) Bruce J. MacLennan, Field Computation and Nonpropositional Knowledge. pp. 14-17.

(Rubin, 1991) Dean Haris Rubin, The Automatic Recognition of Gestures, Carnegie Mellon.

Monographs

(Aleksandrov, 1963) A.D. Aldksandrov, A. N. Kolmogorov,, M. A. Lavrent'ev, Mathematics: Its Content, Methods, and Meaning, pg 75.

(Apple, 1985) Apple Computer, Inc., Inside Macintosh Vol. 1. pg. 141.

(Beck, 1974) Anatole Beck, *Continuous Flows in the Plane,* Springer-Verlag, New York, NY, 1974.

(Kreyszig, 1983) Erwin Kreyszig, Advanced Engineering Mathematics p 559-753.
John Wiley & Sons, Inc. 1983

(Kugler, 1987) Peter N Kugler, Information, Natural Law, and the Self-assembly of Rhythmic Movement, Lawrence Erlbaum Associates, Inc. Hillsdale, New Jersey 1987.

(MacLennan, 1994) Bruce J. MacLennan, Image and Symbol. pg. 8. University of Tennessee Computer Science Department. (monograph in progress).

(Nehari, 1952) Zeev Nehari, Conformal Mapping, McGraw-Hill, New York, NY, 1952, pg. 174-175, 196-198, 263-264.

(Pribram, 1991) K. H. Pribram, Brain and perception: Holonomy and struction in figural processing. Hillsdale NJ: Lawrence Erlbaum.

(Spencer-Brown, 1974) G Spencer-Brown, *Laws of form,* E.P. Dutton, New York, NY, 1979., pg 69.

(Thimbleby, 1990) Harold Thimbleby, User Interface Design, Adison-Wesley Publishing Company, New York, 1990, pg. 42. (homeostasis)

Periodicals

(Adiv, 1989) Gilad Adiv, Inherent Ambiguities in Recovering 3-D Motion and Structure from a Noisy Flow Field. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 11, No. 5, May 1998. pg. 477.

(Krozel, 1993) Jimmy Krozel and Dominick Andrisani, II, Intelligent Path Prediction for Vehicular Travel in *IEEE Transactions on Systems, Man, and Cybernetics*, March/April 1993, Volume 23, Number 2, pp. 478-487.

(MacKinlay, 1990) Jock Mackinlay, Semantic Analysis of the Design of Input Devices in *Human Computer Interaction*, 1990, Volume 5, pp. 145-190.

(Slack, 1993) Mark G.Slack, Navigation Templates: Mediating Qualitative Guidance and Quantitation Control in Mobile Robots. in *IEEE Transactions on Systems, Man, and Cybernetics*, March/April 1993, Volume 23, Number 2, pp. 452-466.

(Zorpette, 1994) Glenn Zorpette, Going to pieces. in *IEEE Spectrum*, April, 1994, pg. 8

Software

(Reiman, 1989) Charlie Reiman, **AutoCruise**
Experimental interface software.
System extension for Macintosh operating system.

(Evers, 1994) Eric Evers, **evers thesis demo,**
experimental interface software.
It has five major sections:
1) hyperbolic desktop interface
2) encirclement interface
3) log-shell interface
4) dipole flow field computer interface
5) branch cut menu interface

Vita

Eric Scott Evers was born in Hamond Indiana on January 4, 1966. He moved many times growing up. Guam, California and Mississippi are some of the places where he spent his childhood. Having lived in Knoxville Tennessee longer than any other place Eric now calls it home.

Eric graduated from the University of Tennessee with a B.S. in electrical engineering in 1990, with a concentration in optics and image processing. He graduated from the same university with a M.S in computer science in 1995. Currently he writes multimedia educational software for the Department of Social Work at the University of Tennessee.

His interests include user interfaces, educational software, visualization, and virtual reality. In his spare time he practices yoga, plays tennis, and loves to cycle and swim in lakes, rivers and pools. Eric also sings at weddings.