Final Report to the National Endowment of the Arts

Thomas A. DeFanti and Daniel J. Sandin
University of Illinois at Chicago Circle
Box 4348 Chicago Il 60680 (312) 996-4621

General Statement of the Problem

Analog computers have been much more successful as tools for the creative artist than digital computers. Analog computers are quite sucessfully used in music and more recently, in video art. Most of the current work in digital computer art suffers from a real lack of fluidity in control, which is why the work appears so sterile and mechanical. What this project has attempted is the creation of dynamic, flexible, user-programmable input devices and requisite control structures. This report will attempt to communicate the actual progress we have made in humanizing digital computing for aesthetic ends.

The problem in general arises from the reliance on linguistic/numeric control of digital computers. Computers were invented primarily to do accounting and highly numerical scientific tasks. If they had been invented with the purpose of doing art, they would probably be quite different in structure. Analog machines, on the other hand, rely mostly on sensing and processing continuous events and data collected via well-tuned input devices. This orientation makes analog computers much like musical instruments, video cameras and other artistic tools that transform continuous phenomena into continuous phenomena. Language in general and computer languages in particular do not

easily perform this type of function. The solution, then, is to provide rich non-linguistic inputs to the digital system and simulate the workings of an analog system but retain the advantages of working with computers.

We had done quite a bit of preliminary investigation before applying for the NEA grant. We have been using slide potentiometers and dials as well as joysticks and data tablets for roughly eight years. Nevertheless, dials, slide potentiometers and the like are very inefficient in their connects to humans. One can only control two of these devices at once, getting a maximum of four dimensions of coherent control. The situation is worse with computer keyboards and function button arrays. Yet the amount of independent degrees of freedom the human body can exert is quite fantastic, perhaps as much as one hundred, but the problem of wiring oneself up to a computer has yet to receive enough attention, to say the least.

So the goals of this project were to provide convenient multi-dimensional controls laden with constructive feedback. Other self-imposed constraints were adopted as well: the controls had to be easy to construct by individuals without access to an elaborate machine shop, the documentation had to be usable by someone with only moderate electronic skills and the construction materials had to be affordable by people without the resources of a major grant.

What we developed falls into four categories:

1. Pressure sensitive keyboards
2. Sensing clothing
3. Force feedback dials

4. Time-based computer variables

Pressure Sensitive Keyboards

The basic idea behind the pressure sensitive keyboard is to provide a mechanism to translate touch (i.e. force) to voltage. One person, then, could control as many inputs as one has parts to push with. Five or ten inputs could obviously be easily manipulated. The concept is seductively simple but, as usual, the actual implementation required several passes. (It should be noted that normal keyboards are binary in nature, that is, all on or all off.)

The easiest way to build a pressure sensitive keyboard is with a type of silicon plastic foam that changes resistance with pressure. Its trade name is DYNACON C (available from Dynacon 14 Bisset Drive, West Milford N.J. 04780). It is cheap ($25./sqft.), can be trimmed and custom fit easily and has a tremendous dynamic range (about nine orders of magnitude versus three for linear dials and sliders).

However, our experience with DYNACON C has uncovered several severe problems that the general user should take note of, and we have some partial solutions. The manufacturer claims that the new product (DYNACON D) has better qualities but we were unable to verify this at the time of writing this report.

First, the performance of DYNACON C is best around 100 billion ohms. This is an impractical level of resistance to design circuitry for--one gets severe noise and bandwidth problems. The partial solution here is to develop circuits running at 10 million ohms, which we did, but the performance is degraded.

Second, the material, not being a solid but a foam, tends to flow away from the area of force after a while. In other words, it forms dimples. This is not a problem if one uses large pads which the user can poke anywhere to even out the dimpling. However, this solution runs afoul of the next problem.

Third, the material is not quite sensitive enough. One cannot easily maintain an even pressure and not get a very tired finger quickly. The solution to this problem involves building force concentrators (e.g. the levers of telegraph keys or large ball bearings) but this solution causes dimpling.

Fourth, it is hard to electronically connect the foam in a way that does not deteriorate with time (due to oxidation). If one uses conductive paint or rubber, for example, the force is spread out too much and the second problem is exacerbated. Thus, the devices built with this foam seem to work at first but do not continue to work very long. A partial solution is to have replaceable pads for the devices, but this requires constant maintenance.

The last problem was unexpected. A linearly applied force tends to produce steps and spikes in the voltage. In other words, if you connect the output of the foam to a microphone and push, a very noisy sound would be produced. We solved this one by using the foam to add to the voltage rather than set the voltage. One then arranges pads in pairs, one to add to the accumulated voltage and the other to subtract from it. The amount added or subtracted is proportional to the force applied, of course, but the noise is not evident because the output is filtered. Note that this solution also gets around the problem of

maintaining even pressure to keep a constant voltage: one simply pushes down until the proper voltage level is reached and then lets go. When not pushed, the foam adds zero to the voltage. It also has no noticeable lag in returning to zero.

We then gave the problem to an industrial design class (AD 224, Professor Dimmitt). The students devised several human oriented implementations (see photos). The best designs had the palm resting on something that allowed the fingers to be curved so force could be applied in a grasping rather than pushing motion.

The conclusion, then, is that pressure sensitive keyboards are a good idea but one that needs stability and sensitivity refinements. We have, however, developed a substitute for the foam by implementing an idea of Rich Sayre's (from the University of Chicago). The next section describes the Sayre Glove, and it should be noted that individual sensors from the glove can be arranged into pressure sensitive keyboards that one can push quite well and get smooth continuous voltages out.

Sensing Clothing

The idea of putting your hand into a glove to control something is not new. There exist handlers of radioactive materail that let humans remotely pick up eggs and crush bricks. Computer Image Corporation has developed an "anthropometric harness" to allow an actor's movements to control the parts of a computer-generated cartoon figure. Both of these systems are quite expensive, though.

The Sayre Glove is an expensive way to turn hand movements into control voltages (see photo). It is easy to build and fits

many hand sizes. Furthermore, the sensor is usable on many parts of the body and in many different situations. The basic idea is to put a light source in one end of a flexible tube and a photo cell in the other end. As the tube is bent or compressed, the amount of light hitting the photocell decreases evenly. The photocell, of course, turns light into voltage. (Note that this is different from light pipe technology (e.g. fiber optics) in which the light is not diminished by bending the conductor.)

Much refinement of Rich Sayre's original idea had to be done to actually make the glove functional. We had to find a glove, for example, that fit various sized hands tightly but comfortably. It also had to be strong enough to attach tubes to but not be stiff. We found that a glove designed for arthritis patients works perfectly.

We also needed a way to attach the rubber tubes in a way that they could be easily repositioned. Rubber 'O' rings sewn to the glove work well and they even hold down the wires. We also found that clear laboratory tubing (R 3603 Tygon tubing) painted black to shut out the room lights works better than black or amber tubing. The internal shiny surface of the clear tubing helps smoothen the effect because the multiple internal reflections still allow light through even when the tube is bent quite a bit. Black tubing and amber tubing simply cut the light off when the tube is not straight.

Some minor disadvantages were discovered and corrected. First, it was difficult to accomodate different finger lengths so we cut the tips of the glove fingers off. Second, the response becomes jumpy when the tube is almost straight so one either

keeps the fingers slightly bent or a bump is placed between the knuckle and the tube. Another solution we found to work is to use logarithmic electronics to process the signal.

In conclusion, the Sayre Glove provides multi-dimensional control which is very smooth and has low noise characteristics. Its disadvantages arise from the user being tethered to the computer by wires (although it is quite possible to have a cup of coffee while the glove is on). One also has to be somewhat graceful to take advantage of such rich control.

Tactile Feedback Devices

The concept of communicating with a computer through tactile feedback is also not new. The remote handlers of radioactive material also have feedback built in. A. Michael Noll built a 3-D servo-controlled joystick many years ago and Kent Wilson has more recently developed a variation he calls the "touchstone." Professor Wilson has also proposed a variation using compressed air to provide the feedback forces. Again, these devices are reasonably difficult to construct so we have attempted a simplification more appropriate to the means of visual artists.

By simplifying, we were able to generalize as well. Our graphics language (GRASS) allows the user to control anything that can be controlled by turning a dial or programming a variable or any combination of the two. Our tactile control devices, the motor dials (see photos), lets one share control with the computer in several meaningful ways. Let us first clarify the control structure a bit.

Say we want to change the size of a picture object on the

graphics screen. Assume its name is "star." One can specify that it should change size by checking the value of variable A sixty times a second by typing "Scale star,A" once. It can change on dial 1 by indicating "Scale star,D1." A mixture of the two is easy as well: "Scale star,(A+D1)/2." The last example is a case of sharing control but it is not one in which the user has overall control since he can only bias the value of variable A. However, by typing "Scale star,M1," the program can control motor dial 1 by writing to it but the user can grab hold of the dial, delay it going to its goal, turn it past its goal, make it go slower of faster, and so on. The interactive control is much more intense.

Fighting the computer, even for aesthetic purposes is not the only use of motor dials. One can use them to preset dial positions while setting up for a performance, for example. But the computer keeps track of the position of these dials very accurately so by clever programming, the user can set detents (detents are found on TV channel selectors and aperature rings on camera lenses). It is then easy to have the stiffening of the knob as one is turning it signify that something is important about a particular value. Kent Wilson used his touchstone to simulate the interactive forces of atoms. We use ours to help join pieces together in 3-space.

One can also program these motor dials to simulate momentum, acceleration and viscous damping. Such simulations require polling of the motor dials at faster speeds than sixty times a second, however.

Time Based Variables

One of the concepts that most frustrates the novice computer graphics user is the complicated looping structures that must be coded for even simple animation sequences. Graphics users who are not particularly interested in learning how to program cleverly are nevertheless often the people who have the most interesting concepts to animate. Much work here has concerned eliminating the need for program loops and explicit iterative procedures for the more common animations. This work now includes implementing variables that are based on time and motion rather than on position, as well as providing methods for using analog control inputs heavily, and allowing easy parallel programming.

Programmers are paid, more or less, on the basis of how well they code loops. Coding loops, especially nested ones, is a skilled task and one that has little direct relation to the real world. We, as humans, do not usually solve our everyday problems by iteration unless we are using a computer.

Computer graphics, of course, involves using a computer. The focus of the work in the Chicago Circle Graphics Habitat has been on eliminating the constant need for programming iterative loops for several types of common animations. This has been done by incorporating various primitives which are discussed in detail below. There are programming languages (Csuri's ANIMA II, for example) which allow the user to type such things as "Move the duck from 1000,0 to -1000,-1000 in 24 frames" but this approach suffers from a lack of generality since the user is restricted to simple motions and cannot use the power of the computer to design motions that obey equations, simulations and so on. In addition, the approach we have taken here allows one to combine timing

functions in a truly great variety of ways.

Now, keep in mind that these concepts are being developed for use by artists who have little programming background, and that the sequences they choose to animate are often rather sophisticated in terms of time and motion. Second, note that this system with its video component (Dan Sandin's Image Processor) is not sophisticated enough to operate in anything but real time (non-real-time video recording devices like video disks are in the $60,000 to $100,000 range). Of course, the real-time operation allows the user to make aesthetic judgments and corrections which, when working with non-real-time systems are much too time and material consuming. Besides, all systems that are fun to use or instruments which have recognized virtuosos (motorcycling, conducting, etc.) are real-time instruments. So the task at hand is to improve the speed of programming for real-time graphics.

A Trivial but Problematic Loop Situation

Say the user's script requires that a sub-picture move from position x1,y1 to x2,y2 in n seconds. How do we normally (i.e. in BASIC or FORTRAN with a refresh CRT) program that sequence? Surely any first quarter programming student could solve the problem--but with what effort? First one calculates x2-x1 and y2-y1 and divides each by the number of steps for the proper increments. But, what is the number of steps? In real-time programming, you have to time the sequence with a watch once it is coded to find out how long it will take. Eventually, by successive approximation, the proper timing of this trivial sequence is achieved. We have made several assumptions so far, however. First, to eliminate integer round-off errors, one uses floating

point calculations, mindful of the fact that floating point is often too slow for much real-time activity unless a hardware floating-point unit is available. Second, we assume that nothing else is supposed to happen while this object is moving from x1,y1 to x2,y2.

The second assumption is far more serious because the task of adding a second object to go from x3,y3 to x4,y4 in m seconds where n and m overlap is non-trivial. The coding is far more complicated than simply writing another loop like the one above because BASIC, FORTRAN and the like do not permit parallel processing (although GRASS, SMALLTALK, and simulation languages do). Consequently, a single iterative loop must be written to encompass the two motions and the timing gets relatively tricky. Again, the first-quarter student could handle this one, with effort, of course.

One solution to this problem is to control subpictures with interactive analog devices like tablets, dials and joysticks. Such devices have problems in that one person can only control so many dials and the movements are not very crisp and precise. Analog controls do, however, allow very subtle movements that would be difficult to program. Since we use dials heavily for prototyping graphic sequences, we feel instant graphic paralysis when the analog-to-digital converters are down. In fact, we have learned a good deal about graphic programming by using dials. The time-based variables were developed with the dials as a model.

The Idea Behind Time-Based Variables

First, let's review why vector generators were invented.

The basic problem with point plotting displays was that they were too slow and the vectors they could draw were prone to digital stepping (both, of course, are still problems with frame buffers and plasma panels today). So hardware was developed to make line drawing a primitive hardware function of these systems. Now, the obvious extension of the vector generator for graphics programming is providing primitive arithmetic functions whose value integrates over time to control motions, scaling, rotations, and other transformations. With a CRT like the Vector General, one can smoothly change 3-D rotations, single axis scaling, scissoring, translation, intensity, depth cueing and still have time left over to change modify endpoints as desired. Time-based control of these primitive (but high-level) functions is what most of our users spend most of their time doing. We have now made it vastly simpler to coordinate time-based changes, so that our users who are not programmers can control a whole new class of animations easily and our users who are programmers can program faster and sloppier without affecting the quality of the final sequences.

Part of adding a primitive to a programming language is trying to make it conveniently fit into the syntax and maintain conceptual clarity. The time-based variables (TBV's) in GRASS do the former well--only a few dozen lines of assembly language code were required to implement the TBV's in both the interpreter and the compiler. The TBV's are conceptually neat because they are treated like any other integer variable (except at interrupt level) and their meaning is clear. Let's look at the details.

All arithmetic variables in GRASS have fixed names. The

type of variable is given by the first letter. The user cannot have arbitrarily-named variables, a shortcoming which has the benefit of both eliminating the need for declaration statements and simplifying the parsing. BASIC, of course, discovered this long ago. GRASS has a dozen different variable types: 64 channels of dials, integer, floating point, local integer and floating, arrays, digital-to-analog, sine and cosine, and so on. It is clumsy to have declaration statements in a language that is playable like a piano.

At any rate, the TBV's are in two sets of 26 pairs each. MA,MB,...,MZ and NA,NB,...,NZ, the first pair, are the linear TBV's. The user sets an M variable to the number of ticks (one tick = 1/60 second) that the motion is supposed to take to complete then gives corresponding N variable the final value wanted (the goal). If the M variable is zero, the N variable is immediately updated, otherwize, the system decrements the M variable every 1/60 second during an interrupt generated by the line clock and the N variable is re-calculated. As an example, say we want to move an object SAM from -1000,1000,500 to 1000,500,-200 in three seconds. We type in:

```
MOVE SAM,NA,NB,NC
MA=0;NA=-1000;MB=0;NB=1000;MC=0;NC=500
MA=180;MB=MA;MC=MA;NA=1000;NB=500;NC=-200
```

The MOVE command sets SAM to translate according to whatever values NA,NB and NC contain every 1/60 second (the updates of subpictures are handled during the interrupt mentioned earlier). So, whenever the variables change, SAM moves accordingly. The next two lines in the example set the original values of the N's

and then specify three seconds are to be taken to reach the goals. Clearly, to set another object in motion, say, after 1.25 seconds to a different place in four seconds, we type:

```
MOVE SAM,NA,NB,NC
MOVE DAN,ND,NE,NF
NA=-1000;NB=1000;NC=500
ND=200;NE=300;NF=400
MA=180;MB=MA;MC=MA
NA=1000;NB=500;NC=-200
WAITFOR MA LE 180-75
MD=240;ME=MD;MF=MD
ND=-500;NE=700;NF=-200
```

The time necessary to interpret these statements is negligible so the timing is accurate. The important thing to note is WAITFOR with which the user gets directly at the system timing mechanism in a simple way. Of course, rotations, scaling, scissoring and so on can be controlled just as easily. One can even control the Image Processor through the digital-to-analog channels. The reader should be able to imagine a set of commands that cause an object to rotate ninety degrees over 1.2 seconds, shrink during one second to half size after two seconds, and translate along the legs of a triangle at the rate of one second per leg.

The interpolation method used for the linear TBV's is efficient in terms of storage and execution time. An N variable is updated by the following formula if the corresponding M variable is non-zero:

```
new=Nold+(Goal-Nold)/Mnow
```

This method compensates for binary roundoff so the calculations

may be done in integer, and it allows the user to change the goal by resetting the N variable before the M variable has gone to zero with no jumping as in the more obvious:

Nnew=Noriginal+(Goal-Noriginal)/Moriginal*Moriginal-Mnew

This latter formula also requires more storage and is prone to binary roundoff errors with integer arithmetic.

Sinusoidal Time-Based Variables

One of the defects of linear interpolation for object transformations is the apparent 'banging' at the start and end of the interpolation. This perceptual phenomena is due to the unnatural instant acceleration and deceleration possible in computer simulations but unusual, to say the least, in our everyday experience. The technique to avoid this banging is to start the object slowly, speed it up smoothly and end it slowly, just like the motion of a pendulum. GRASS has an internal 256-element sine/cosine table which is used for updating the rotation matrices. By mapping this table on arbitrary distances, it was possible to easily implement the sinusoidal TBV's QA,QB,...,QZ and RA,RB,...,RZ. The sinusoidal TBV's are used just like the linear ones, but the effect and interpolation are different.

The level of this primitive is somewhat higher than the linear case. It is unusual that a user of this system would spend the time to figure out a cosine mapping like this to improve the motion. The sinusoidal TBV's are currently used more than the linear ones.

Earlier it was mentioned that the TBV's were developed based on observations of dial usage. At one point, Dan Sandin put 25 microfarad capacitors across the dials to make the movements less

jittery and more slushy. When a dial is turned quickly, it takes about a quarter-second to get to the new value. (This delayed response makes it murder to play PONG so we also have dials with no capacitors.)

Use of Time-Based Variables to Build other Primitives

Keyframing or linear interpolation has been a computer graphical technique for many years. The goal of keyframing has usually been to get from one subpicture to another in a smooth fashion, an approximation of conventional cartooning inbetweening. The algorithm is trivial, but the organization of the data is not. Yet, given that the point-to-point matching is done, the trivial interpolation algorithm can benefit by being controlled by time-based variables or dials.

Our keyframing (called BLEND) was once implemented with the same algorithm used for the linear TBV's. This algorithm, however, has its own peculiar roundoff errors which cause a wobble when interpolating from a circle to a triangle, for example. The TBV method was originally chosen to save space but it turned out to be inflexible as well as wobbly. Our artists wanted to be able to blend on a dial, blend back and forth at the flick of a wrist or under program control, blend in subsets of the range, and so on. They even wanted to blend from one object to another which, in turn, is blending to a third, all in real-time, of course. Consequently, BLEND was rewritten in the following command format:

BLEND OLD,NEW,VARIABLE,EXPRESSION

where variable can be an integer variable, dial, TBV or value and the EXPRESSION indicates the range over which the

variable should pass.

The simplest use of BLEND for the case of normal keyframing is done by

```
MA=180; BLEND OLD,NEW,MA,180
```

The range is set from 180 to 0 and MA is decremented by the system every 1/60 second. Also possible is:

```
BLEND OLD,NEW,D0,WZ
```

WZ is set to 32767 so users do not have to remember such a peculiar number. The intermediate 'frame' of old on its way to NEW is controlled by the position of dial 0. You can crank back and forth at will to discover interesting visuals. By setting the expression to 10000, say, twisting dial 0 between 0 and 10000 will produce the normal progression from OLD to NEW, but going past 10000 will cause the interpolation to progress past the subpicture called NEW. Setting dial 0 negative will back the interpolation through OLD. This extension, especially on a dial, allows the discovery of new images based on the differences between already existing images.

Subpictures with several hundred vectors can be blended in real-time, that is, with no noticeable stepping. Many subpictures may be blending at the same time. When too many vectors are blended, the response at the system console terminal (VT05) suffers and the interpolations start to step. Flicker is not increased by an increase in interrupt-level calculations since vectors are shipped to the Vector General ar a higher interrupt level.

Some Further Observations

First, since our interfaced 16mm movie camera is interrupt-driven at about one frame-per-second, any program witten with

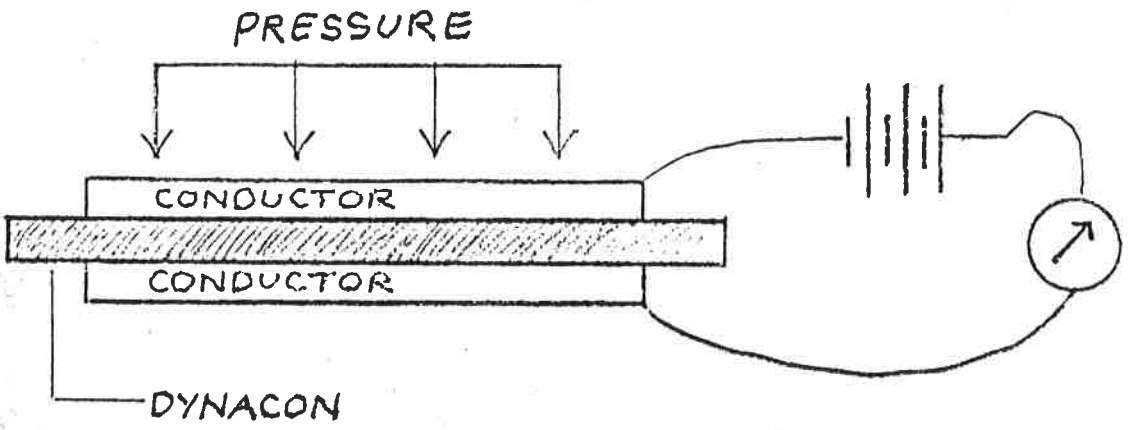
TBV's automatically slows down to the camera's time. In this case, ticks are taken from the camera's shutter opening, not a line clock. We occasionally use film when enough vectors to cause flicker are necessary to complete the animation, when too much blending or calculation is indispensible, or the application requires film (there are still a lot more 16 mm projectors than video projectors in use). Of course, filming off a large CRT is not comparable in ease or quality to using a microfilm recorder.

Second, and more important, the first eight of the time based variables also go out to the digital-to-analog converters. The values can be used to change motor dial positions over time, control the Image Processor to programatically change colors, do keys, wipes, fades and so on. The richness of this coordinated interaction is easily available to the artist and it boggles the mind.

Conclusion

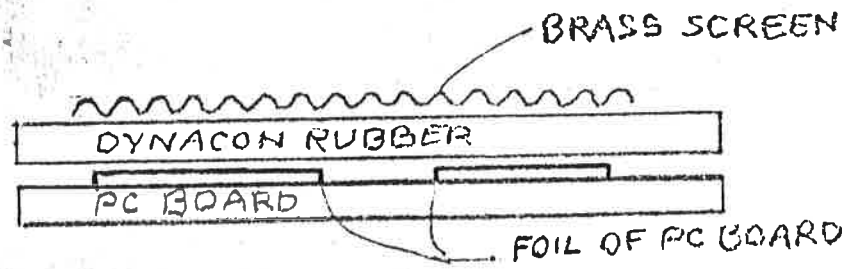
We feel real progress has been made in user control. We have simplified the concepts of feedback control, we have designed and built several new multi-channel input devices that artists can afford to build for their home systems and we have generalized the concept of computer controlled motion. We have also made it possible to mix modes of control in very flexible ways. Our task at hand now is to disseminate this information through the normal channels of computer art/design conferences and technical papers.

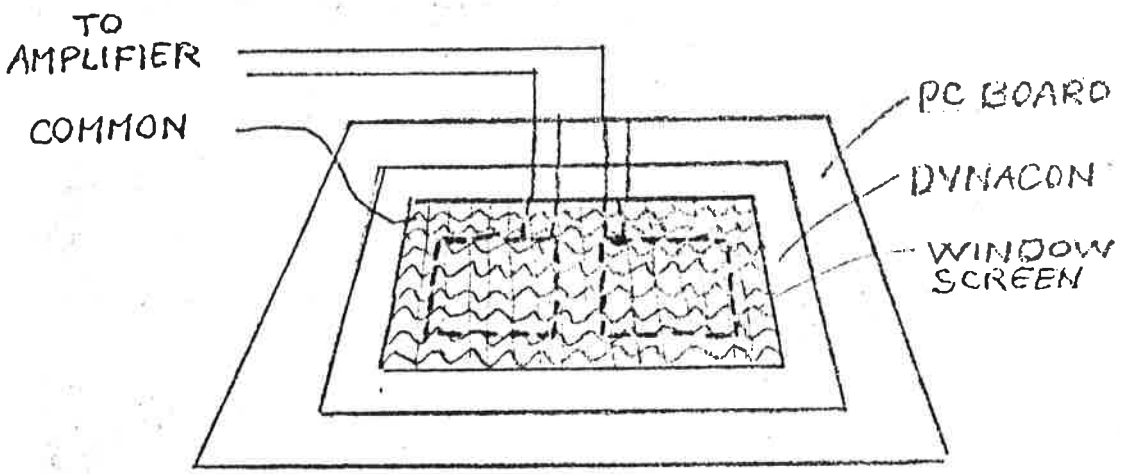
fig 1



Current through Dynacon rubber is a function of pressure.

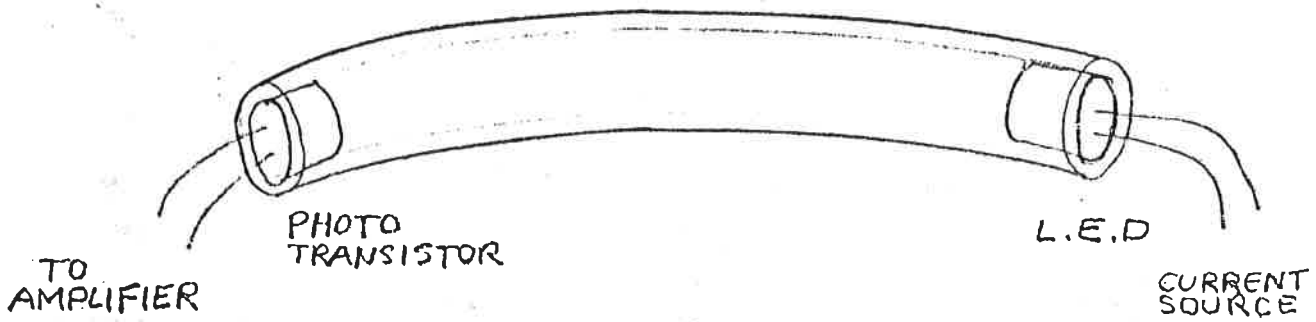
fig 2





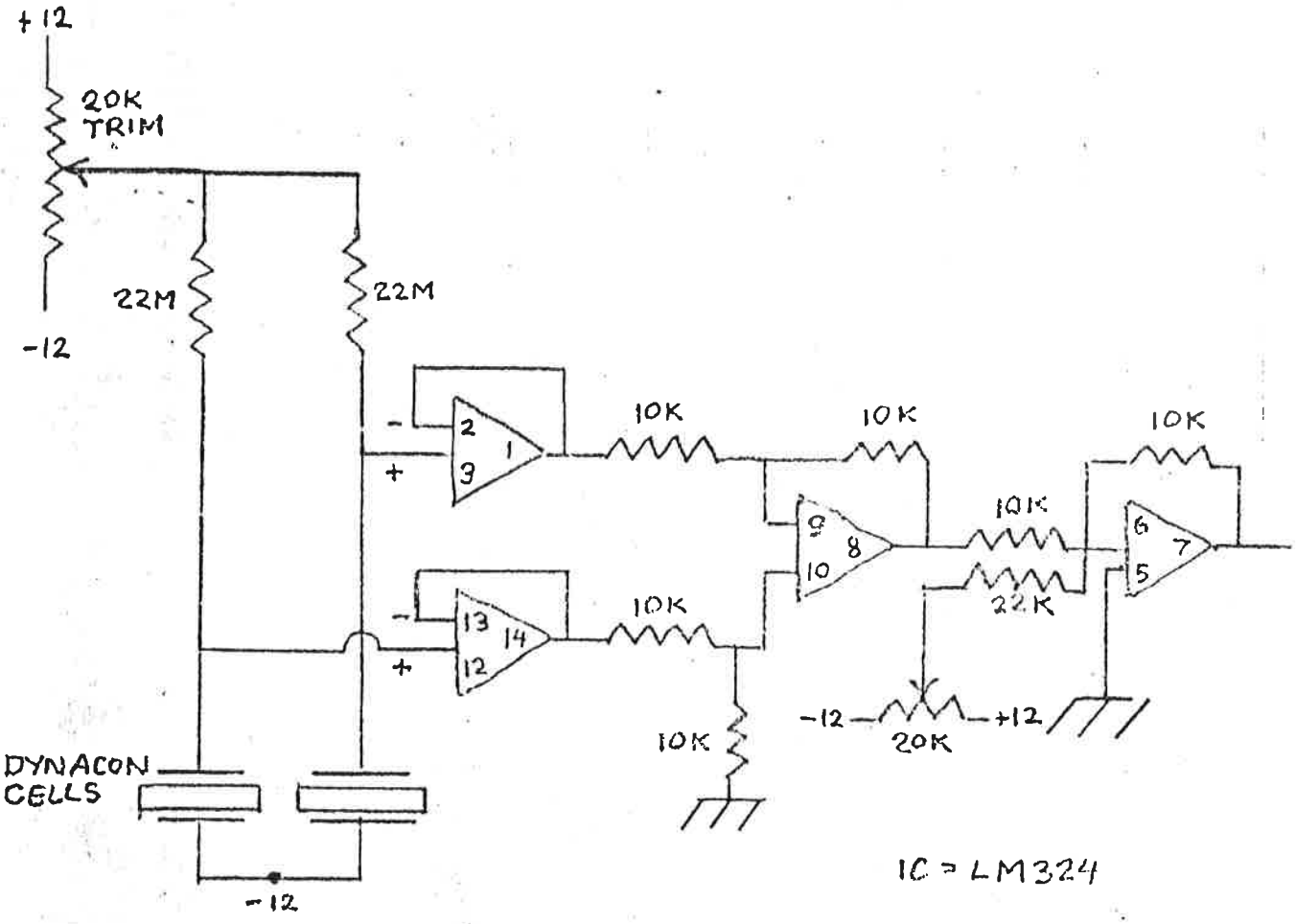
A practical pressure transducer can be made with printed circuit card material, brass window screen and Dynacon foam. The areas that are to be made sensitive along with conductive paths to the edge are masked off with contact paper. The board is then etched in the same manner as for printed circuits. The tape is removed and a sandwich is assembled with Dynacon foam and screen. The screen forms the common conductor for all the cells. As with the other designs, people who wish to construct copies of these devices should contact Dan Sandin or Tom DeFanti for more detail.

fig 3



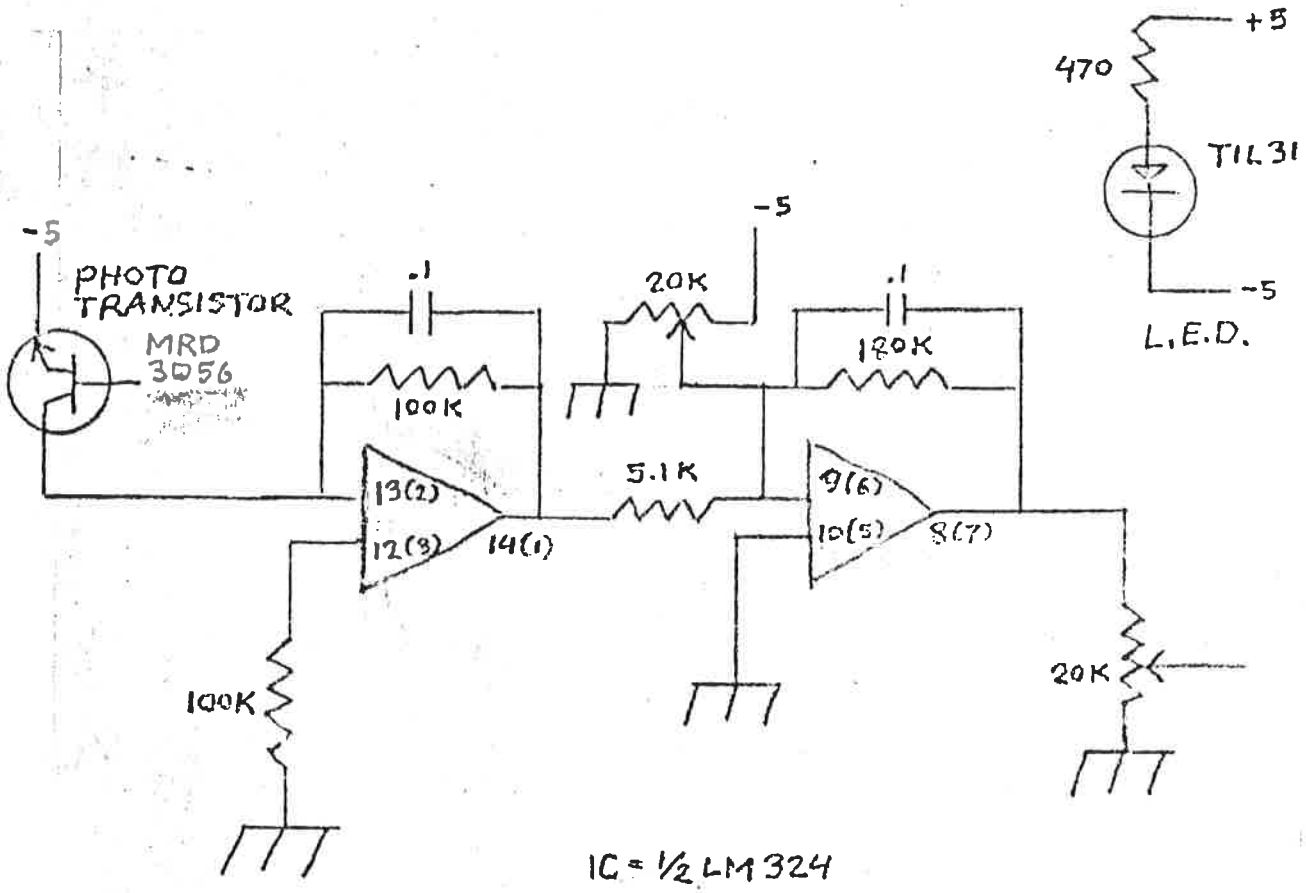
An angle transducer can be created by press fitting an L.E.D. and photo-transistor into opposite ends of a clear plastic tube. As the tube is bent, the amount of light falling on the photo-transistor is lowered. This technique was suggested by R. Sayre at the University of Chicago.

fig 4



This schematic represents the ciruit required for one opposed pair of pressure sensors of a pressure sensitive key board. A single quad operational amplifier implements the entire circuit which consists of two high impedence buffers, a differential amplifier and a summing amplifier. The supply voltage is positive and negative 12 volts.

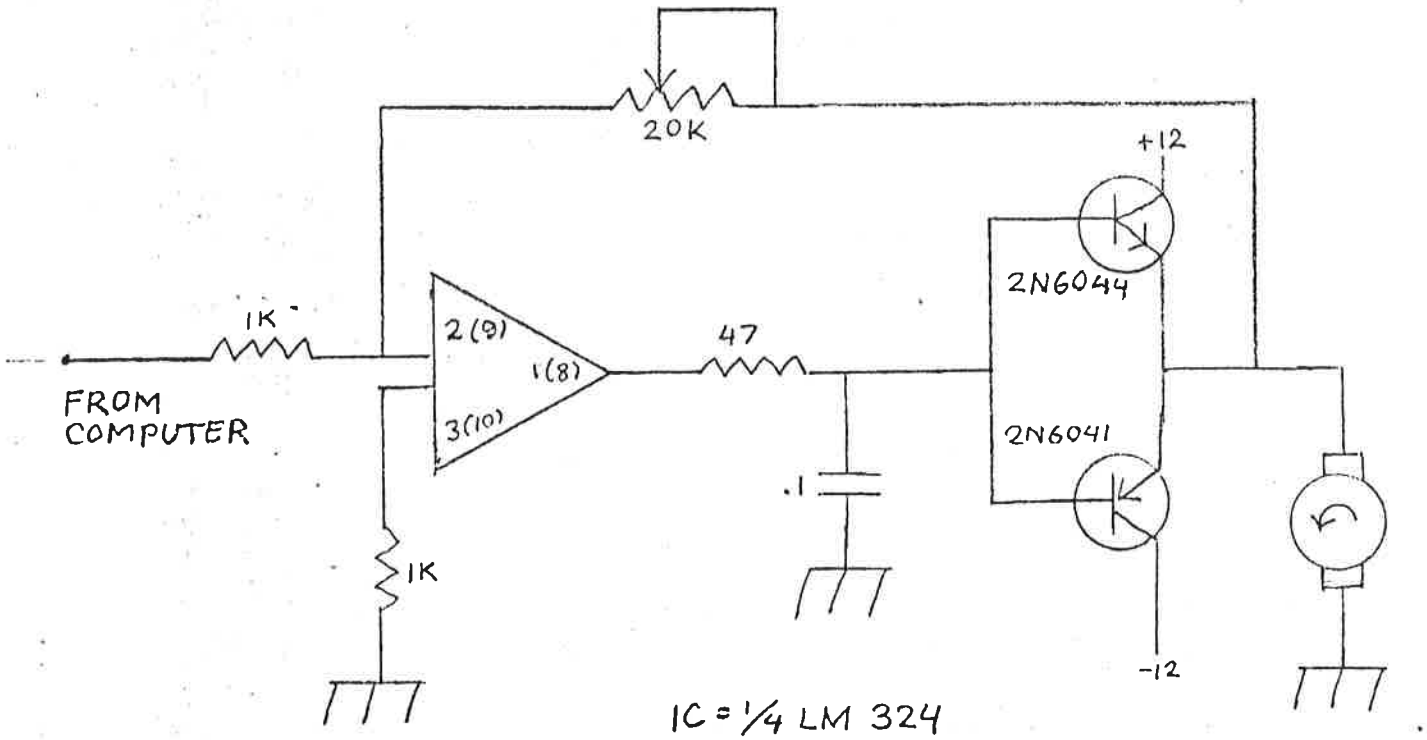
fig 5



This is the schematic for the amplifier for one channel of an angle transducer for the Sayre glove. It consists of a current to voltage converter and summing amplifier. It uses supply voltages of positive and negative 5 volts.

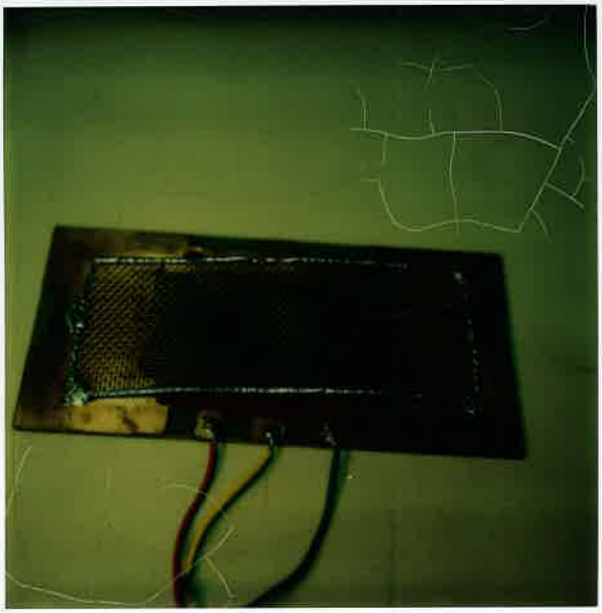
The number in parentheses are the pins of the other half of the I.C.

fig 6



This is the schematic of the amplifier for the force feedback dials. It consists of a single operational amplifier with a current booster inside the feedback loop. It is important to keep the power and ground wires associated with the power transistors and motor separate from the power and ground wires of the operational amplifier to prevent oscillations. Each motor requires about 3 amps peak current, but good regulation is not necessary. The number in parentheses are another operational amp in the same I.C. Because the current required to drive the current booster are significant, I recommend using only 2 out of the 4 op-amps in one chip.

fig 7



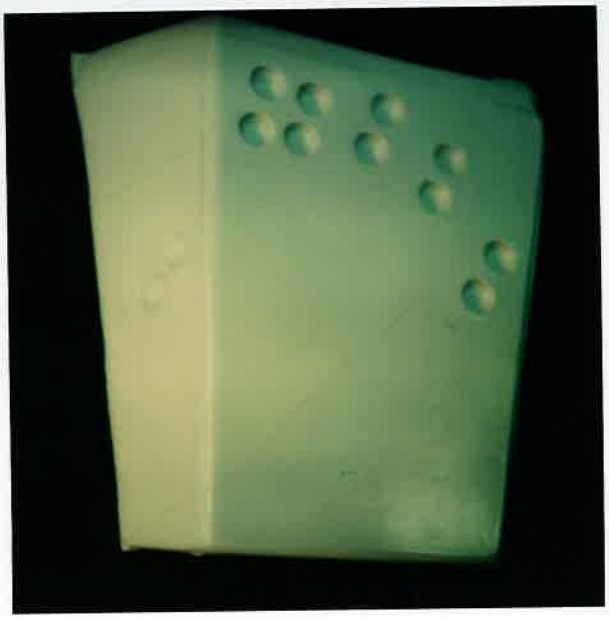
Basic two-key keyboard constructed with printed circuit material.

A keyboard design for either hand which allocates 3 functions to the thumb. The thumb is underutilized by most keyboard designs.

fig8

A bowl shaped keyboard which allows the hand to be comfortably bent.

A keyboard design to be held in both hands which allows simultaneous control of 5 variables.



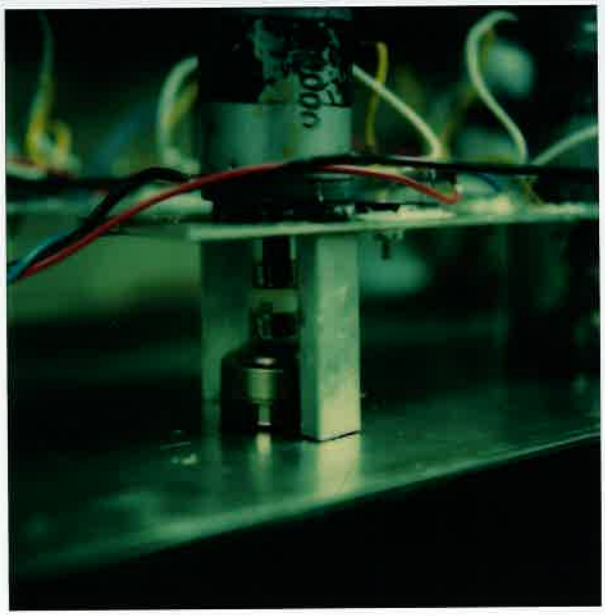
A keyboard design based on two planes which accomplishes the goal of allowing the hand to bend comfortably.

A keyboard design based on a rocking motion to control a variable in opposite directions.

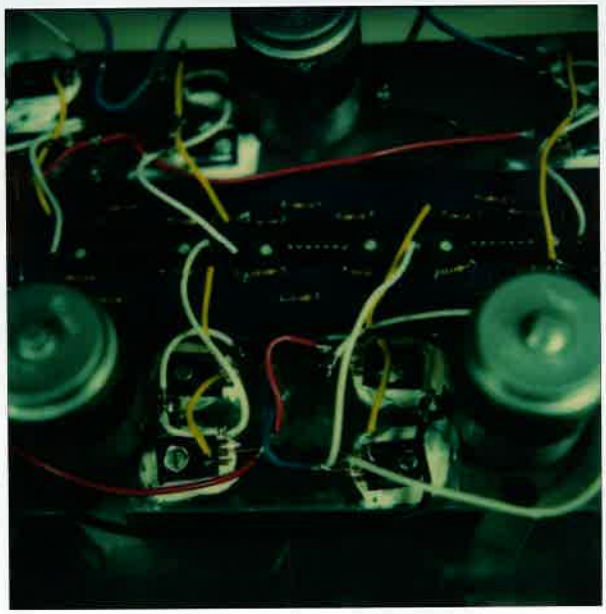
fig 10

The finished force feedback dials, in foreground, look very much like the normal dials, in background, but they feel different.

The Sayre glove. The black tubes are the angle transducers which register how much each joint is bent.



Detail of force feedback dial. Motor on top is connected via a flexible coupling to a potentiometer.



Bottom view of force feedback dials. Note that the power and ground wiring associated with the power transistors and motor is heavy guage wire and is kept separate from the operational amplifier's wiring.

The rectangular box is a versatile input box that allows a variety of analog signals to be conveniently connected to the computer without modifying anything.