1

Interactive Systems And The Design Of Virtuality

The Design of Interactive Systems, Tomorrow's Crucial Art Form, Rests On New Philosophical Principles

Ted Nelson

What follows expresses the personal views of the editor.

It should be plain by now to anyone that the future of mankind is at the computer screen. Already hundreds of thousands do their work at screens, working with text or diagrams or maps.

This is only the beginning, and I think it is obvious that virtually every form of non-physical work, and many forms of play and leisure study, will soon migrate to the interactive computer display screen, as suitable systems are designed.

Yet most interactive systems are lousy; and almost every new interactive system I see leaves me aghast.

In this article I will try to set forth very briefly, with examples, what I conceive to be the correct principles of design for interactive systems. There is much more to be said, at both concrete and theoretical levels, but here for now is the brief overview that is so sorely needed.

Interactive screens, as everyone should know, can present anything — text, pictures, maps, the latest information on whatever you need to keep track of.

Real-world control from a screen already occurs in some places. Systems are installed or available for factories and utility companies — and military forces — that allow the user to modify the world at the touch of a picture. By changing a picture, the user can cause the real world to change accordingly — if the system is so set up. Change a valve in a diagram on the screen, and the real valve itself, half a mile away in the refinery, opens or closes.

The potential flexibility of such systems is enormous. Any pointing tools can be used to work in this fashion, if suitably linked into the controlling computer and its program. (You could open that valve with a keyboard, a joystick, a tablet, a trackball, a mouse — or any other mechanical hookup you might prefer.)

Both laymen and computer people mistakenly believe that the design of interactive systems is somehow a "technical" matter. While there are many technical aspects to such design, I believe the conceptual and even artistic problems of such design far outweigh in importance the mere technical details. The designing itself is art, not science at all. (Sometimes, of course, there is a question of whether a thing can actually be done, or how, in which case technical questions loom large.)

PITFALLS

There are many design pitfalls in attempting to build interactive systems. Let us run by these quickly.

1. Cheap AI Systems

One approach that was popular five or ten years ago was the "artificial intelligence" approach, based loosely on the idea that a user would type in English-like sentences, and the program would pretend to be alive, friendly and understanding in responding to what you thought you asked for.

But the typing of input strings is tedious and generally a waste of time. Moreover, the program's masquerading as an intelligent entity is usually misleading and annoying, both for the time wasted in trying to guess what the program *really* does, and for its gratuitous pseudo-social invasion of contemplative privacy. Fortunately, this type of system is proportionally on the wane (except in the personal computing field, where it lives on as the "adventure," or Guess-My-Commands, game).

(Note that these criticisms are not intended to apply to artificial intelligence as a long-term goal, but only to artificial intelligence as a local pretense of contemporary software.)

2. Command Languages

Another common approach is to create a command language for what you want to do — like, say, the "text editors" of old. Each action must be called forth by typing in an input string in code.

But user languages are hard for beginners to learn, and have the "computerish" feel that is so repellent to non-computer people.

Command-language systems are also clumsy and obtuse compared to more highly responsive design. (As one widespread example, more highly interactive forms of word processing have appeared, supplanting the old text editor programs that used to require (for instance) explicit insertion commands.)

Command languages are also dangerous. By permitting many simultaneous options (and variations, and modifications of further variations), command languages make it possible for things to go terribly wrong in a very short time — terribly, terribly wrong; irrecoverably wrong. While language facilities must of course be available to programmers, an environment in which a user thinks *about something other than the computer* should not be tangled by the complications of a command language that forces his attention where it does not belong.

3. Menus and Afterthought Interfaces

The last pitfall is what I call the menu trap. Now, menus are all right, and better than the approaches I have mentioned so far, but for high responsiveness and performance values, I believe the kinds of interactive systems we will describe below are greatly preferable.

The so-called "user-friendly interface" is a variation of the menu trap. The very phrase suggests that the user interface goes on after the system is actually built, like paint. We will speak more about this later.

© 1980 T. Nelson. Trademarks cited are those of the author.

The Ten-Minute Rule

An interactive computer system for most purposes should be learnable in under ten minutes. This criterion shocks many computer people. It has certain vaguenesses. But as a striking statement — a battle cry — mandating high-power interactive design, it is the most concise way I know of saying how easy things should be. They understand the ten-minute rule in the arcades. They understand the general idea of virtuality design in the arcades. The people who don't understand are the "computer professionals" — who sometimes do great damage.

VIRTUALITY

The central concern of interactive system design is what I call a system's *virtuality*. This is intended as a quite general term, extending into all fields where mind, effects and illusion are proper issues.

By the virtuality of a thing I mean the *seeming* of it, as distinct from its more concrete "reality," which may not be important.*

An interactive computer system is a series of presentations intended to affect the mind in a certain way, just like a movie. This is not a casual analogy; this is the central issue.

I use the term "virtual" in its traditional sense, an opposite of "real." The *reality* of a movie includes how the scenery was painted and where the actors were repositioned between shots, but who cares? The *virtuality* of the movie is *what seems to be* in it. The *reality* of an interactive system includes its data structure and what language it's programmed in — but again, who cares? The important concern is, what does it *seem to be?*

A "virtuality," then, is a *structure of seeming* — the conceptual structure and feel of what is created. What conceptual environment are you in? It is this environment, and its response qualities and feel, that master — not the irrelevant "reality" of implementation details. And to create this seeming, as an integrated whole, is the true task of designing and implementing the virtuality. This is as true for a movie as for a word processor.

The virtuality of an interactive system is composed of its conceptual structure and its *feel*. A system should have both a good conceptual structure and the right feel.

The truly interactive system, as in the arcades, needs no carriage returns; each user action creates an instant response — and may not echo what you typed.

(It is amusing to note that the firm most associated with "the computer" in the public mind does not manufacture computers which can be programmed in this way. Yet I believe this is how computers should in general be programmed.)

Here too we see an exact similarity of the interactive system to the motion picture. It is the overall impression, not the component parts or the particular tricks of presentation, that count. In a movie it doesn't matter what kind of camera or form of scenery was used to make a given shot; what matters is the contribution that the scenery makes to the shot (and its feel), and the contribution that the shot makes to the film (and its feel).

The interactive system, I think, may best be thought of as a new kind of movie; but a movie that you control, and a movie that is about something you wish to affect. And it is the imagining of this interactive movie that is the important design task. Never mind the nuts and bolts — what's the dream?

The following brief discussions are intended to highlight some interesting systems and their special features, considered as virtualities. In the final part of the article I will endeavor to tie together some general principles of virtuality design.

A more detailed analysis would get down to exact controls and the way they are merged with presentations, showing the infuriating intricacies of ramification that must be dealt with. I could spend a page comparing the cursor behavior when two different word processors Delete, and much more on the nature of conceptual structure. But not in this overview.

DATALAND

One of the most striking and easy-to-use computer systems in the world is the system called "Dataland," created by Nicholas Negroponte and his associates at the Architecture Machine Group at MIT.

The Dataland user sits at a large screen with various controls. The main controls are essentially joysticks that allow you to pan and zoom.

On the screen you see an unusual collection of pictures, some quite small. You may zoom in on anything (panning the screen to select what you want magnified), and the original pictures or symbols will be enlarged, augmented by more detail, or replaced by other pictures.

In principle this can go on indefinitely. It is like being in satellite orbit with a huge zoom lens, able to look at an overview of Los Angeles, then magnifying it until you can read the fine print on a newspaper on the sidewalk.

Because Negroponte is a great showman, he has dolled the system up with a variety of graphic features, synthesized on the screen by computer, such as ticking clocks.

*The closest other term I can find is "mental environment." My students have urged me to retain the term "virtuality," even though it causes confusion among users of so-called "virtual systems," meaning real systems configured with virtual huge memory.

3

Interactive Systems cont d...

By storing detailed information on any topic in the Dataland at specific levels of magnification, it is possible to put huge arrays of data where the user can essentially pick out what he wants graphically and dive into it. A stunning, sweeping concept, rather obvious if you think about it; yet it makes many of the systems that have been developed for database query look foolish. You can do it all visually with Dataland, at least with hierarchical data. In under ten minutes' training, you are able to search through great hierarchies of stored information, pictorially.

Although the system in its present form requires millions of dollars worth of equipment, I am sure that within a few years we will have access to comparable systems on machines as small as the Apple or Atari.

Enthusiasts have supposed that there is something fundamental and magical about Dataland's two-dimensional array. From the standpoint of virtuality theory, however, the 2D array is only one kind of clear and simple world. This system's power simply shows the power of two-dimensional organization in our thought, especially because of our experience with paper and other 2D viewing situations. But there are many other powerful organizing ideas. While clarity and simplicity (and good examples like Dataland) are desirable, many other screenworlds will also be exciting and useful.

VISICALC

One of the most important new software products is Visicalc, originally designed by Dan Bricklin and now marketed by Personal Software, Inc.

Visicalc (see review in last August's issue of *Creative Computing*) effectively creates a vast dynamic worksheet for bookkeeping, accounting and financial planning.

Now, many people suppose that accounting is an exact science. Many accountants, however, feel differently. A great deal of their time is spent reworking columns of figures and deciding which numbers belong where. An accountant may spend hours creating lists of expenditures, adding them up — then removing items, putting other items in, and adding the column up again in its new form.

This is all perfectly legitimate. Just *why* accountants do this all the time is outside the scope of this article, but they do.

Visicalc consists of *programmable columns*. You may create, for instance, a column of figures, and tell it to keep a sum at the bottom that is always correct; every time you insert or remove a new item, the sum is recalculated.

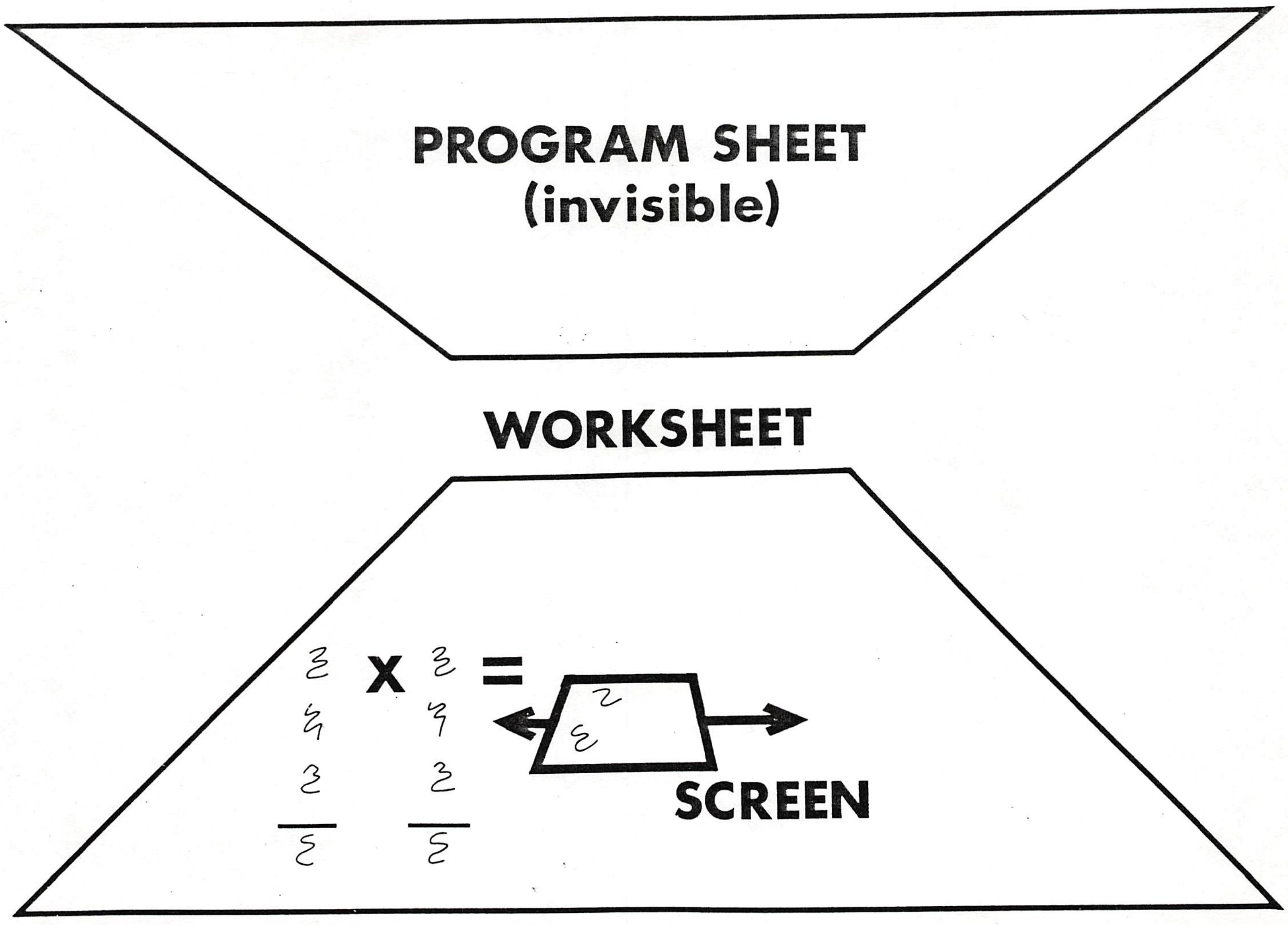
But Visicalc goes much further. You can tell this programmable worksheet, for instance, to take 20% of each figure in Column A and put it in a corresponding position of Column B.

The programming is by example, and somewhat hard to learn, but Visicalc essentially allows the creation of an enormous array of columns and terms, with fairly complex relations among them, all instantly up-to-date every time you change a figure.

Those who work a lot with figures consider it breathtaking, and there are accountants who consider it reason enough to get a computer.

Visicalc might be visualized in any number of ways. I like to think of it as two sheets — the columns of figures, which can be seen by the user, and the overall program sheet, which cannot.

Thus the *world* of Visicalc consists of



A sheet of columns, with instant summing-up and carrying-over, is controlled by an invisible program on an unreachable sheet.

Virtuality, cont'd...

these two sheets; but the *views* allowed of this show only the lower one. (This is something like the division between Heaven and Earth in some religions, the former supposedly being visible and the latter not; but I would hesitate to conclude anything about the religious views of Visicalc's designers from this.)

Considered as a virtuality, then, Visicalc has a serious blind spot: although you create and use the program sheet, you are not allowed to see it.

Pointing up such omissions is a possible benefit of thinking in terms of virtuality. By considering the whole shown world, its views and feel and control structures, we can search out omissions and asymmetries and suggest better structuring. (That this is possible for something as good as Visicalc shows that this approach may be of very general benefit in the future.)

PARKED CARS

An unusual example of an elegant and spare virtuality is a work entitled "Parked Cars" by the Argentinian artist Laszlo Snead. It has not previously been described in print. While the work is still only in the planning stages for the Apple, it can be considered as an interesting virtuality whether or not it is ever completed.

Some might call "Parked Cars" an Adventure game; the artist himself prefers to refer to it as a "work of art." Indeed, it might be considered an X-rated work of art, since it may include both sex and violence, but only if the user so chooses.

"Parked Cars" is to be in the form of a comic strip, showing one panel at a time in Apple lo-res graphics. (This mode on the Apple also allows an independent rectangular panel of scrolling text, which may serve as talk balloon or caption). Snead intends the overlays for the whole of "Parked Cars" to just fill one side of a diskette, so there is to be much doubling-up of text and graphics, in ways the artist hopes will be inspired.

The action of "Parked Cars" takes place at a scenic overlook on a parkway at night. Three cars drive in, in random order. They are inhabited, respectively, by persons that Snead refers to as "The Hot Couple," "The Nervous Couple" and "The Guy With a Knife."

Now, a number of interactive stories already exist for reading from computers. This one may be the first to have extensive graphics. What is especially interesting from the virtuality standpoint, however, is Snead's selection of the user controls, and the way they relate to the world we are watching.

The user may simply rove through the three-dimensional scene as a disembodied spirit, spying on the different characters. But they won't do much; they stay in loops. The user may, however, *enter the psyche* of any character, and read that character's impulses as they pass through his or her mind. The full set of commands is:

ROVE AS SPIRIT
ENTER PSYCHE/LEAVE PSYCHE
THINK (brings forth a thought to read)
ACT ON THOUGHT
GO TOWARD/AWAY FROM OTHER CHARACTER

A little cogitation will sûggest the vivid potential of this small set of commands, playing through a suitable scenario — which Sr.Snead is well advised to supply.

What is less obvious is that this interesting control structure is especially suited to the somewhat raw material that Snead has chosen. While the same control structure could be applied to a more urbane dramatic setting — say, a detective story or political melodrama — the potential tedium on the one hand or wildly varying outcomes on the other would create great difficulties. The choice of a small cast in a highly-charged small setting would seem to be ideal balance for working through the artistic premises of this virtuality.

WORD PROCESSORS

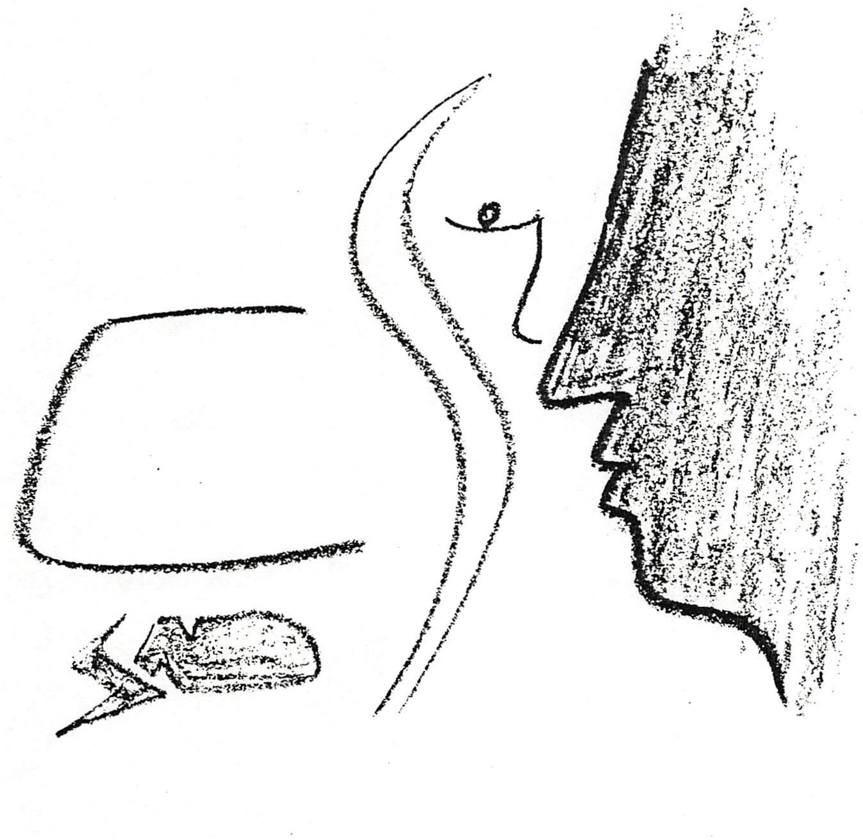
Despite pioneering work by Douglas Engelbart and others, popular opinion has it that "word processing" originated about 1967 with the appearance of IBM's magnetic tape typewriter.

Since then "word processing systems" have become epidemic. These are almost invariably disguised computers with a fixed program that acts only as an interactive text editor — you aren't allowed to run any other program. (Indeed, the salesman will deny ardently that it is a computer).

If you have a personal computer, however, the term's meaning shifts: a "word processor" becomes an interactive text-editing program that you run on your computer. (The marketing of personal computers is already beginning to impinge on the field of fixed-function word processors — the kind that won't let you run any other programs — and this is due to increase.)

Now, for some reason every technical wonk in the world thinks he is capable of writing a good word processing program. But in my opinion there does not exist a single satisfactory word processing program anywhere in the world.

This is not the place to hold forth the argument in detail. My basic view, however, is that a proper word processor should be, like any other interactive system, an artfully constructed virtuality, an integrated system of world, views and controls that is extremely easy to visualize, roam in and change.



Watch from afar, or enter the person's mind. You choose what happens.

Virtuality, cont'd...

Computer people (who are not usually concerned writers) have gotten some fairly twisted notions about the nature of text. The correct units of text are the *word, sentence, paragraph, heading* and *chapter*. Yet for some ungodly reason, computer people have gotten the idea that the units of text are the *character* (including control characters) and the *line* (or *linefeed).* This in turn leads to the unfortunate writers being told by the programmers that they're not thinking logically because they can't keep their minds on the invisible control characters. The programmers' heads have been in the wrong place, implementing the wrong virtuality.

(We will not even discuss here problems of the loss and protection of files, or the atrocity of short filenames required by some popular operating systems.)

Now, a case can be made for baroque word-processing programs that take weeks and months to learn, and allow you to format output in columns, windows and whatnot, but the fundamental problem of word processing is *fast input and revision,* and making the system easy to learn for everyone in the office (including office temps). Thus I think we need simplicity more than we need the baroque.

I have personally been designing word processing programs since 1960, and indeed the theory of virtuality presented here has co-evolved in part with the designs. But I will only inflict one of my text designs on the reader of this article, and that only loosely.

That design is the JOT™ system (Juggler of Text). And its special virtue is its ease of learning and use. In all cases where the system was demonstrated to a fresh user, the user learned to insert, delete and rearrange text in *under seven minutes.* (Unfortunately the implementation died, being in an obscure language on a now-defunct machine, and this figure is for a total number of less than twenty people. However, the system is presently being re-implemented for the Apple.)

Basic Operations

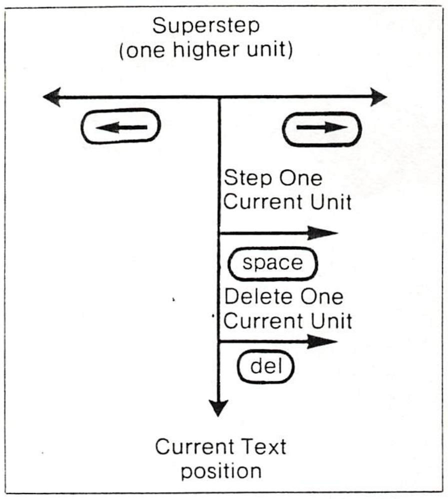
The basic operations of a simple word processor are essentially insert; step forward and back by word, sentence or paragraph; delete units of various sizes; and rearrange.

The JOT system was originally designed to do all this with the Teletype Model 33 as a terminal, capturing full upper and lower case even though that device has no shift key.

There are no command input strings. There are also no menus, a certain minimalist aesthetic having taken over. Half a dozen or so keys have been given new meanings which appear to hang together psychologically.

For instance, in one of the principal modes, left arrow means "move leftward to the beginning of the sentence" and the right arrow means "move rightward to the end of the sentence." The space bar means "step one word."

In the other main mode, the same controls mean "step left to the beginning of the paragraph," "step right to the beginning of the paragraph," and "skip one sentence."



I will not attempt to justify these commands in isolation: it is the whole system whose virtuality we design, and the individual commands stand or fall as part of this whole.

Some programmers consider the command structure of JOT to be detestable and illogical. But the system was not designed for programmers. It is intended to be usable for hours on end by non-computer people who are tense (sometimes frantic), and utterly preoccupied with the words, not with the machine.

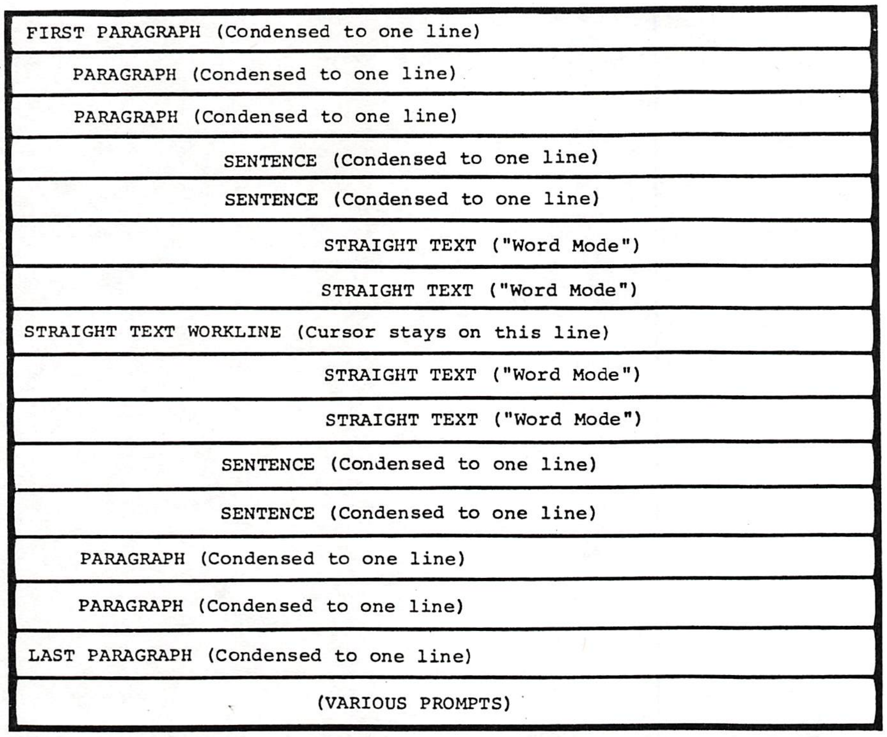
The actual state-diagram flowchart of the system is a hairy mess. (Indeed, the two programmers who implemented it originally agreed between themselves that it was all wrong — until they saw it in operation. Both became ardent virutality-design freaks. The fact that two exceptionally talented individuals were not able to imagine the system's performance from the flowchart shows what we are up against in general.)

I have spent no fewer than five hundred man-hours on the design of the JOT system's virtuality *only,* not counting any implementation. My impression is that most designers spend little or no time on virtuality design. While I make no claim that quality of work is ever proportional to the time spent, I think that the important thing is to design the interactive qualities first, *then* implement.

In essence the design process for JOT involved considering many different desirable features and operations, then cutting them down judiciously to a very small but powerful set that could be easily learned. In addition, it became a challenge to marry the desired control functions to a sparse keyboard (such as that of the Teletype or Apple) while still making it seem natural. Others can judge from the live system whether this has succeeded.

Regardless of the success of this particular design, we will return to these principles, and try to generalize them, later on. □

(TO BE CONCLUDED
IN THE DECEMBER ISSUE)


```
FIRST PARAGRAPH (Condensed to one line)
    PARAGRAPH (Condensed to one line)
    PARAGRAPH (Condensed to one line)
                SENTENCE (Condensed to one line)
                SENTENCE (Condensed to one line)
                          STRAIGHT TEXT ("Word Mode")
                          STRAIGHT TEXT ("Word Mode")
STRAIGHT TEXT WORKLINE (Cursor stays on this line)
                          STRAIGHT TEXT ("Word Mode")
                          STRAIGHT TEXT ("Word Mode")
                SENTENCE (Condensed to one line)
                SENTENCE (Condensed to one line)
    PARAGRAPH (Condensed to one line)
    PARAGRAPH (Condensed to one line)
LAST PARAGRAPH (Condensed to one line)
                          (VARIOUS PROMPTS)
```

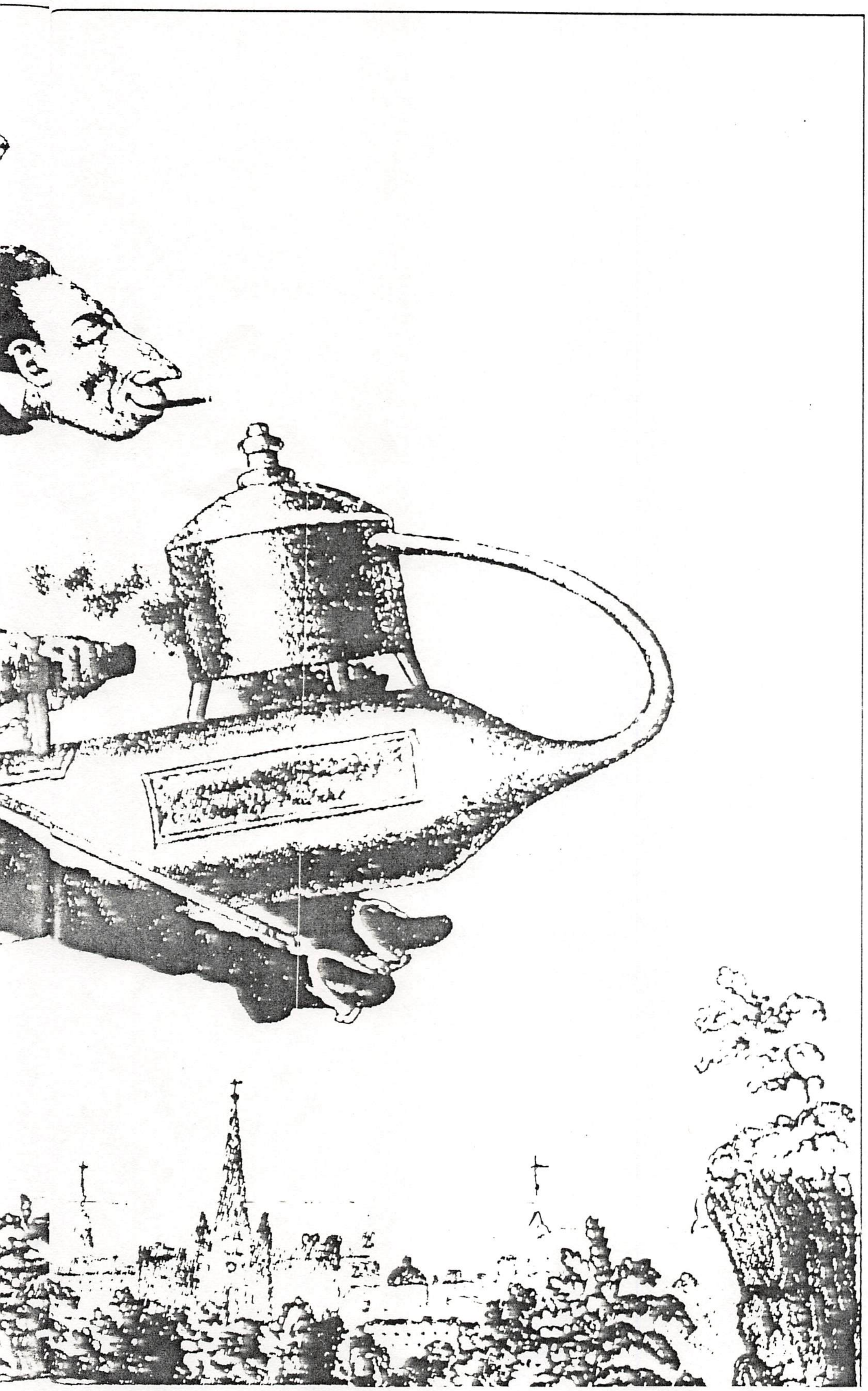
ScreenJot™ — the expanded version of JOT — shrinks an entire document to one screenful, condensing sentences and paragraphs to one line each for orientation and overview. The cursor stays on the midline.

6

Interactive Systems and the D

Design of Virtuality Part Two

Ted Nelson



In Part I, we considered some nice examples of highly responsive systems. The reality of their implementation details is comparatively unimportant. What *is* important is the design of the conceptual structure and feel of a system; we call this its "virtuality" as distinct from the (unimportant) reality.

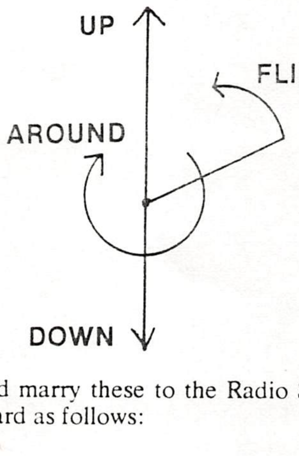
In this concluding section we consider some more design examples, and endeavor to find the right principles on which to base the design of interactive systems in general.

A COMPLETE SYSTEM

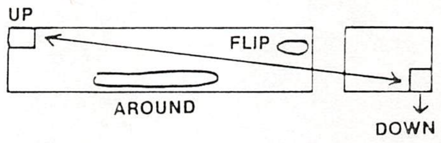
In one design, the Funny-Face Softree™ system, I have endeavored to show that one simple, overarching control structure can be used for a *complete* personal computer system—including word processor, scheduling system, graphics package, bookkeeping package, typesetting and layout programs, etc. (I do not wish to imply, of course, that this is the only way to organize such an integrated system: merely that this one interests me.)

There are four basic controls. These are the *only* controls. They may be understood quickly in a brief demonstration, but in fact the further ramifications of their interaction may become clear gradually.

The controls we call *up, down, around* and *flip*.



I would marry these to the Radio Shack keyboard as follows:

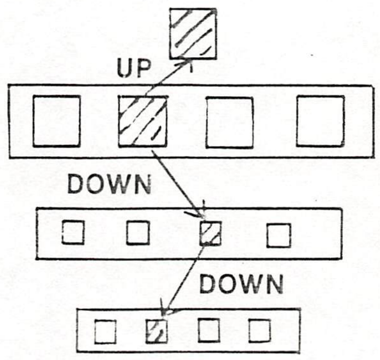


Up and *down* are the easiest. The user

8

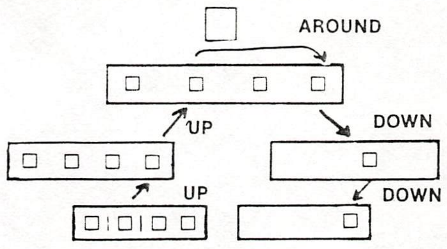
Virtuality, cont'd...

is at all times on a tree of functions. Each node is a particular activity or way-station on the tree. *Up* of course takes you to the node above you on the tree. And on this tree, *down* is always specified at any given moment as one of the specific alternatives below.

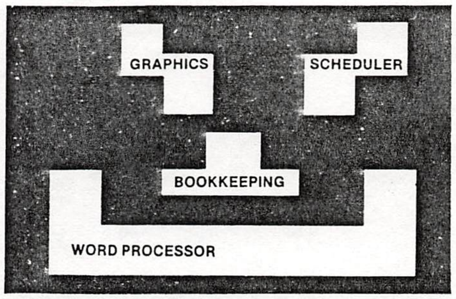


We may call this a *latching tree*. From your current node you may go down or up. If you go up, you get to the top; if you go down, you follow the path of already latched, or chosen, selections.

How do you change the selection of the node which is *down*? You do this by pressing *around*, which selects in turn each of the different alternatives below. (I call such a circular succession of choices a *ringstep*.) Thus to go between any two places on the tree only a few particular steps are required: something like *up, up, around, down, down*.



How do you see where you are and make the choices? Now comes the really unusual part. Each menu is a *jack-o-lantern face*.



You go up and down a tree of menus. Each face has one of its features (left or right eye, nose or mouth) *flashing slightly*. This is the current selection below.

Now a frequent complaint about menus is that you have to take time to read them. In this system that is only true at first; because *every menu has a different facial expression*. So that as you become familiar with the different menus, their scowls and grins tell you where you are at once, and you can make your choices faster and faster.

At the very bottom level of the tree are particular activities; *down* there commands the events themselves.

There are also working faces, however, corresponding to every menu, on which materials may be viewed, scrolled, etc. This working face is the "other side" of the menu. You get to the workface, or back to its menu, by *flip*.

$$\begin{tikzcd}
\text{MENU} \arrow[r, leftrightarrow, "\text{FLIP}"] \arrow[d, leftrightarrow, "\text{UP}" , "\text{DOWN}"'] & \text{WORKFACE} \arrow[d, leftrightarrow, "\text{UP}"', "\text{DOWN}"] \\
\text{MENU} \arrow[r, leftrightarrow, "\text{FLIP}"] & \text{WORKFACE}
\end{tikzcd}$$

That's essentially all there is to it. What you have seen is what the beginner sees. I have left out showing how the different parts combine, so that, for example, the graphics tablet used with the scheduler produces animation, or the scheduler used with the word processor permits a magazine layout.

I would point out certain other features, however. One is that there are very few steps between paired activities, and the user going repeatedly back and forth between them gets into a rhythm. Faster methods would be in reality less simple.

Another aspect is the system's uniformity of replicative structure. You can go anywhere with confidence that the structure will hold. (It does become quite irregular, however, at the bottom or execution level.)

Some people tell me they'd rather have an input-string command language. That's a mater of taste. Other critics say this system lacks generality, which misses the point. It is simple, easy to learn, and integrated. You cannot get lost. And the funny faces are good for a laugh.

THE XANADU ™ HYPERTEXT ENVIRONMENT

The Xanadu ™ hypertext system, toward which I and colleagues have worked for some twenty years now, is intended as a super document library and annotation system, among other things. We may also think of it as a new form of storage and publication.

The Xanadu system is planned as a network of storage computers in McDonald's-like franchised stands around the country. By dialing into your local Xanadu stand, you may get anything on the whole network—to which your local stand is tied by high-speed lines. You must access the system from a fairly powerful terminal—that is, a computer, for reasons which will become clear later.

While most of the Xanadu work has gone into problems of its implementation—especially algorithmic design and analysis—the system's emerging virtuality has acquired an extremely interesting character, which I will now describe.

Everything stored in the Xanadu system we call a *document*. A piece of text, a picture, a movie (someday), a lonesome marginal note—each of these is a document.

Any document you want comes when you ask for it, if you are entitled to it. A document is private or public—that is, published. Any user may call up any public document instantly, as well as his own private documents or any other private documents he has permission to use.

LINKS AND WINDOWS

Links may be put anywhere in any document. Links, like footnotes or marginal comments, permit a user to jump to related material at any time—and come back from that other material when he likes.

Free-form, non-sequential writing of any kind—what we call collectively "hypertext"—is made possible by these links. But the virtuality of general hypertext would take a book in itself.

An important type of link is the window. A window may be thought of as a "hole" in one document through which shows a part of another document.

CHANGES AND VERSIONS

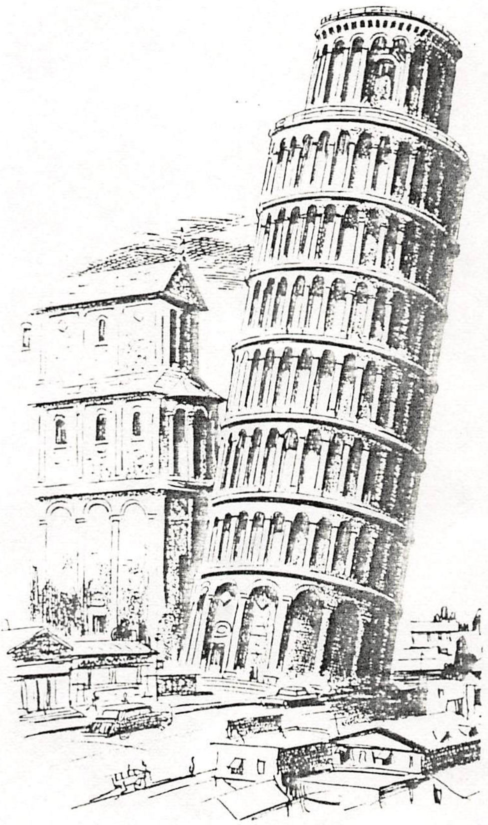
Not only may an author store a document in its present form; he may, if he chooses, write or rework the document on the system, with the changes themselves stored. The Xanadu system does this at a uniquely low incremental cost, since our data structure and algorithms essentially assemble parts of a given version as they are needed—without ever bothering to assemble the full consecutive structure, unless it is asked for.

Thus the user has access, if the materials are saved and open to him, to a reconstruction of any previous version of a document at any previous moment he cares to specify.

Not merely consecutive historical changes, but alternative versions, may be generated at any time. Thus a document may be "rewritten" for different types of readers, and these different versions stored at low overhead.

The user may ask to see any given piece of text (or other information) in any version or at any previous time.

9



A link made to a certain part of one version of a document may be automatically followed through to the same material in any other version of that document or in its previous incarnations or in other public documents that windows it.

We believe that this "versioning" facility, of linkage across backtrack and alternative versions, solves a central problem of text systems — that of cross-referencing any parts still being worked on; a problem which is chopped at and nibbled at everywhere but is often dealt with in ineffectual ways.

FREE LINKING BY ANYBODY

You may create a document that links to any other documents, if they are public (he who publishes must agree to this in advance).

You may create, in your document, windows to anybody else's public documents. (Since they get the royalty when their part shows, they should be pleased.)

This is how we handle marginal notes. If you create a marginal note, it is automatically put in a new companion document, your document, which is permanently linked to the document you have annotated.

(This "companion document" idea also frees you to *alter and rewrite any public document any way you like*—since the alteration is in a private file of your own that points to the intact original.)

COPYRIGHT

"What of the copyright problem?" you ask. Our solution is simple: as you use the system, you are continuously paying small increments of royalties to copyright owners. These are modest amounts, the same for all users: for instance, if we can supply the service for two dollars an hour at 30 characters per second, the fixed royalty runoff will probably be about five cents an hour. This is divided among the copyright holders in proportion to how much you used from each—sliced very finely.

What keeps people from making copies? Nothing, since terminals are under the control of individual users: but since everything is still stored on the system and available instantly, the cost and inconvenience of making and filing private copies will be often seen as superfluous.

OVERVIEW: THE XANADU SYSTEM AS A VIRTUALITY

The above description specifies a general and powerful facility for business, literature, correspondence and digital storage of all kinds.

As such it represents a cohesive and unified virtuality which has been thought about and reworked for years. Its appearance of simplicity and obviousness is the distinctive quality of a carefully wrought design: *There are hundreds of other ways to do these things*, as experienced computer people well know; yet making the parts hold together clearly, complement each other, and *make sense*, takes a very great deal of work.

XANADU FRONT ENDS

Of the functions described above, only a few are actually handled by the Xanadu service network: *put this away* and *give me that in such-and-such a version* are really all that the Xanadu back-end machines do.

The rest has to be done, actually, in your personal computer. Marginal notes, for instance, require making a companion document out of your marginal notes, for instance, and declaring it and putting it away in the network. Most users will also want to keep track of how they have been jumping among various documents and activities. These necessary functions belong in your own computer.

Thus the "full" Xanadu system, as we recommend it be used, entails a cooperating program in your personal machine that acts in these ways.

Thus full Xanadu service has two parts. The "back end" is the proposed Xanadu network; esentially all it does is store and fetch by versions and links.

But a high-powered terminal is needed by the user, to show the documents sent by the back end, to present the possible actions the user may take, and to translate these choices into the proper fetch-and-store instructions for the back end.

This is of course the "front end." There are many possible ways to visualize and control the Xanadu functions—even before graphics or music are stored on the system—and we welcome imaginative front-end programs of any design, even if marketed independently. The Xanadu project will, however, offer guidelines for front-end design.

If you choose to use the back-end network in some other way, that is your privilege as a customer; but in order to encourage what we see as desirable modes of operation, we will be offering various trademarks to software vendors who wish to create cooperating front-end programs.

Given the overall virtuality of the Xanadu system, there are countless possible ways to summon, visualize and control its operations on screen. All of these are valid and welcome. To give some ideas of the possible varieties, I will discuss two very different Xanadu front ends.

(Since these are highlights of the two front ends, no attempt will be made to show all the functions, reconcile their different emphases, or intercompare them.)

THE XANATREK™ FRONT END

The standard Apple computer, laudable as may be its general qualities and capabilities, has a few conspicuous limitations. One is its text screen, only forty characters wide.

However, an Apple strength is fast-action low-res graphics. Two pages of hardware memory are dedicated to either text *or* low-res graphics. We will proceed to use this fact.

The Xanatrek front end has been designed for fast and exciting use of the Xanadu facilities, as well as for invigorating use of its low-res graphics.

The system was, quite frankly, inspired by *Star Wars*, and shows how far you can go in playful and analogous use of graphics.

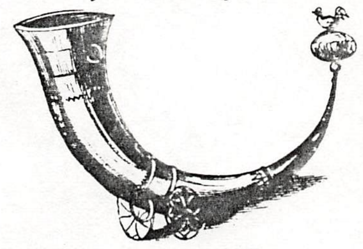
One of the things a Xanadu user must be able to do instantaneously is ask exactly what he is looking at—that is, having jumped to something or wandered by degrees from his original activity, he needs an instant and valid explanation of what he is looking at.

Very well. While reading anything from the Apple screen (from page one of memory) the user may instantly demand a map of what he is seeing. This is continually available and up-to-date in low-res color, on page two. But aha, you say, how can you read it, since the color display disables the character generator? The answer is that the various patches of identifying text are indeed visible on this map as patches of seemingly random color, but since the Apple allows *one text window* on a low-res screen, successive boppings of a particular control will step the various text labels into a readable panel.

The most amusing visualization in the Xanatrek front end has to do with seeing the major features of a document such as chapter breaks and seeing links as well from a companion or other document.

This brings out the "Star Wars" styling. What the user sees looks like a huge passing spaceship or perhaps a packing crate, in the vault of night.

One of its visible sides shows the major parts of the text itself, as streaks of color. The other visible side shows the entrance points provided in your companion document. (You may select any of these places for your next trip.)



Other ships that pass in the night are documents linked to this one. Want to see the links? Hit a button, and animated squares *fire from one ship to the other*, with that p'tew p'tew sound we *Star Wars* fans have grown to know and love as the sound of a laser weapon in a vacuum.

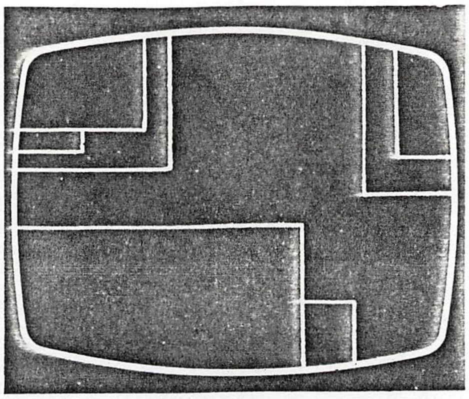
THE CORNERCOPIA™ FRONT END

This Xanadu front end has been thought out for implementation on an actor language on a small Sorcerer computer.

There are many approaches to the design of screen panels. One approach, generally associated with Xerox PARC, strews panels diagonally on the screen. The approach that follows is intended to be a little easier.

Five to ten screen panels are accesible at a given time. They come out of corners of the screen.

Each panel keeps one free corner anchored in a particular corner of the screen. Its opposite corner remains always visible but may be moved by the



user to any position which does not obscure any other panel's free corner. As with PARC panels, any "behind" panel may be instantly brought to the front without moving its borders.

Each panel is labelled with a one-line title (at the top of the lower panels or the bottom of the upper ones).

The text may show and scroll in any panel: naturally dependent or "parallel" text (a standard Xanadu statement) may scroll in any other visible panel with links to the independent text shown by scrolling symbols on the panel borders.

Perhaps this environment seems not to show enough. Very well: some panels themselves represent *other* such environments: when brought to the fore they swell up to become other multipanelled views.

OOF

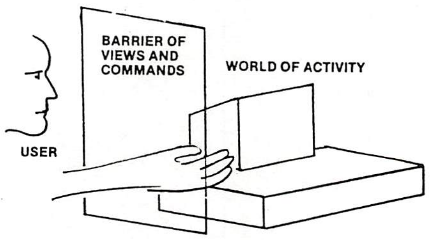
The "office of the future" will consist basically of cabinets for incoming correspondence, printers for outgoing correspondence, and in between, screens, screens, screens.

In this highly competitive area, harried programming managers everywhere are under pressure to work out what *happens* on the screens. But what do they, or *anybody*, know about it? It's *not a technical problem!* It's merely delegated like one.

The problem has nothing to do with technicalities; all of these are squared away. The problem is in the design of virtuality. But I know of few designers at present competent and imaginative enough to make those screens come alive and make working at them a joy. Which is the *real* problem.

WORLD AND VIEWS

An interactive virtuality is essentially a



world created by a programmer or designer. This world has a certain structure which may be easy to understand or hard. This World is visible through different *views* allowed by the designer.

The World is what you're really thinking about; the view is the temporary way you're looking at it.

The distinction between World and View is crucial. The World is what the user is supposed to be acting on and thinking about; the View is all he really gets. (Controls are in a way part of the view.) If Views are good, the World comes to seem real, natural, at hand, under control. Poor Views (or worse, a hard-to envision world) create confusion and poor usability.

The system should have easy-to visualize states and conditions, and, preferably, some kind of spatial orientation that readily becomes a map in the user's mind.

The designer should begin by thinking about visualizing the *World*, not the Views, and let the Views come later.

(Yet designers are always getting seduced by particular views and treating them as the world itself.)

The principles of the World are the central, integral, virtuality; how you see it is secondary. It is important to acknowledge the cruciality of World design, and consequently the importance of the principles you develop for it.

The designer creates a simplification or stylization of the original world. There must always be some reduction or stylization; the important thing is that these reductions or stylizations not detract from the principal things you need to understand and control.

In transposing an old activity, the question is what to retain in the world and what to dismiss as part of the view. (For instance, Text Pages—divisions of text—are part of the View, not the World.)

Anything can be shown, any buttons or sticks or whatever, with any preentational machinery. People are always asking for bigger screens—but actually to ask for a bigger screen is usually a copout. Ask for higher performance. Faster flips and flaps and scrolls and panel pop-ins. Fast action and seething cues. Leave several things on the screen at once, to remind you of what you've been doing, what you might be doing, what else there is to do, and any other current options.

The operations in the user enviroment should feel more and more like operations on the world. As stated above, controls are in a sense part of a view. Anything can be controlled by almost anything, buttons or sticks or keyboards.

An interactive system should have very few controls and these few should have far-reaching and powerful uses.

Marry the available controls and the

Virtuality, cont'd...

desired functions. Menus should be used, rather than input languages or the fictitious "natural language dialogue;" or better, yet, control diagrams.

Actions should be easily reversible and their consequences immediately recognizable so the user can back out of a mistake without being punished. (Compare this with the word-processor horror stories you hear all the time.)

Most important, the overall *principles* you choose for a system should be sweeping and have few or no exceptions. In order to clarify these issues we must consider the issues of both soft principle and soft clarity.

THE PHILOSOPHY OF SOFT PRINCIPLE

The following discussion has to do with the design of principles, which is in fact the essential issue.

We frequently consider something and ask ourselves: *What are the implications of this?* And one of the nice things about science and technology is that the implications tend to be clear and exact.

In many cases, though, implications tend to be less certain. Implications don't follow clearly from premises. Those who want clear-cut answers become edgy or annoyed. The main tradition of Western thought has been to try to find the exact implications of every idea. (Ideas which don't seem to have exact implications, as well as people who *prefer* unclear situations, cluster in the humanities or "fuzzy studies.")

But some things are by their nature unclear in implication. These include both cluster-concepts ("Democracy," "Womanhood") and design ideas ("Let's see, maybe it could fold back onto itself somehow").

By tradition we often tend to talk of such ideas as improperly formulated or incomplete. I would like to put it another way and call a principle whose implications are inexact, a *soft* principle.

This throws things in another light. Rather than suppose the soft principle is just "not finished yet," let us consider it instead *another logical category*—somehow analogous to the conventional hard principle, but not subject to deduction.

If you can't deduce, how can it be logical? The answer may be that we've been looking at the wrong features of logic and have missed the analogy. There are in a sense soft equivalents to implication, contradiction, and other logical configurations. (See table.) I hope to develop these ideas more broadly at a later time.

SOME FAMILIAR IDEAS SOFTENED AND RECONSIDERED

	Hard	*Soft (or mixed)*
IMPLICATIONS, RAMIFICATION	Hard Implication, Consequences	Possibility, Tendency, Expectation, Connotation
PARADOX	Contradiction	Irony, Oxymoron
COMPLICATION	Obstruction, Interference, Countervailing Principle, Something in the Way; Amendment, Modification	Things to be Clarified, Resolved Worked Out

What good is this analysis? At the very least it is suggestive. If a principle is *by nature* soft then we can understand it on its own terms rather than insisting that it hasn't properly hardened "yet."

Or take soft design ideas. A given idea could be worked out into hard form in numerous different ways. Some you may like better than others, and a variety may be valid.

Now take *several* soft design ideas, all at once. How do their ramifications fit together? The answer is *indeterminate*, since the ramifications of each could take many forms. But if you are aware of this, then you can search carefully for the *combinations* of possible workings-out, their variety and their interactions.

The "inspired" design of something final and precise comes, I believe, from sifting many such co-implications of possible hardenings of the ideas.

And the important guideline is: *don't rush it*. Don't take shortcuts. Don't assume that decisively pinning down one aspect of a design will speed things up; it's like nailing your left shoe to the floor.

If we think of design as the search of many possibilites, "soft design" is that which is sensitive to unexpected simplifications, conveyances and harmonies.

In short, *don't be too sure of what you're looking for, and be ready to appreciate the ramifications of surprises.*

Principles in Practice

Eventually, the soft design principles we have tried out lovingly must be hardened into specific hard forms of computer operation. What should the principles be like? Again tradition may ilitate against recognizing the best sign decisions.

The general principles of a system, *once chosen,* should be consistent, but "consistent" according to looser criteria than the designer may be used to. In particular, a design principle may be psychologically clear for people to work with, easily visualized or imagined, yet not reducible to any customary formalism.

Indeed, "consistency" here takes on a strong psychological flavor: a thing is consistent if users think it is consistent and use it consistently—even if we don't like it, like the double negative in Spanish. (We may call this naive consistency or soft clarity.)*

Thus the final chosen principles need not be "logical" in the rigid sense of conforming to somebody's predefined notion of how things should behave. But working out in soft form, we study their fittings-together in great detail.

The designer should eliminate any background notion that the user must be like him. All too many designers reward the user for being like himself, the designer, or punish the user for being different or thinking differently. The objective is to be of service, not to clone yourself.

TECHNICAL TRADITIONS VS. SOFT DESIGNS

The design of virtuality is essentially the design of operating principle. The design of principle, in turn, has to do with the generation and modification and inter-sculpturing of soft principles.

The biggest design problem though, is that the designer tends to freeze too quickly on a particular set of rules and arrangements. Technically-oriented people tend to seize one or two principles and hang onto them through thick and thin, not perceiving when it is time to rework their ideas.

I have learned through bitter experience, indeed, that only a small proportion of technical people are even capable of *listening* to this viewpoint. The soft design of virtuality seems to be totally alien to technical training.

* Mark Miller, who worked on the original JOT system, considers it a consistent virtuality, even though it "corresponds to no known paradigm of program structure."

Virtuality, cont'd...

Those who design interactive systems tend to be technically trained, and technical training generally promotes the background assumption that what you are working on is given and well-defined.

Training in the arts and creative fields, on the other hand, promotes the ideas that a design (or piece of writing or a movie) is fluid, may take many forms, and will be reworked over and over until it reaches a final state that may be wholly unlike its earlier stages. I believe this latter outlook is far more appropriate for the design of interactive systems.

CONCLUSION

Interactive system design is a field in itself, utterly unlike what is taught in any computer science department I know of. If I have not proved this point, I hope the designs and ideas presented here will at least provoke some unease.

(This is no claim that these designs are righter than any others; but rather that these designs are a unified package that feels right and is therefore of interest. They represent local peaks in design space, in the sense that small changes would, I think, detract from their unity and clarity.)

These designs represent hundreds of hours of work, but the difficulties of the decisions and the rough edges don't show. (That's part of good design and art).

The art of designing things in general is very little understood. People think that something is well-designed if it is *sleek*, *stylistically unified*, and *if its controls look as much alike as possible*. (An example is the "designer" audio equipment from Bang and Olufsen, shown at the Museum of Modern Art and copied everywhere, where every control resembles every other control.)

This approach is wrongheaded beyond belief. (I think stereo equipment is poorly designed, and B&O the worst of them.) You do not want controls that look alike. You want controls that look and feel *different*. If you have a big round knob for the volume control, you should have a square knob, or a slider, for the tuning. There should not be a row of similar buttons for different functions, but a row of *different* buttons—or better, not in rows, but some other arrangement contrastively arrayed. Do you need glasses to read what it says above the knobs? Lousy. Can you tell at a glance one control from another? Good. CAN YOU WORK IT IN THE DARK? Terrific.

As a rough guide, *good design is inversely proportional to the probability of a user making a mistake*. And this criteria carries over to interactive computer systems.

To make a system easy to use is extremely difficult and time consuming, in the same way that it takes more work to write a short article than a long one.

You should not "design the system" first, and then put on a "friendly front end", (although this is what must be done in many cases), any more than you should first shoot a movie and decide what it is to be about (although this occasionally works).

An interactive system should become second nature, and become second nature quickly. This is essential for many reasons. One is that we will have to move among many different interactive systems in the future, and there will be no time to savor and adapt to the local complications of each. They will have to spring clearly and straighforwardly at the mind and hand.

Moreover, interactive systems will be used intensely for hours, often by tired, high-strung, frantic people, who are trying to get a job done in a hurry, and who are thinking only of the world they are trying to operate in—not the intervening complications. It is up to us as designers to create fast, safe, elegant systems of view and operation without snags, dangers or complications.

The system designer, or movie director—let's call him *you* —must have a full understanding of what things are easy to do, what things are not, and what is hopelessly impossible. You then make a collection of all the ideas and visualizations (and scraps and parts) you would like to put together in your system. Then your rework them and rework them, and rework them.

THINK OUT THE WORLD

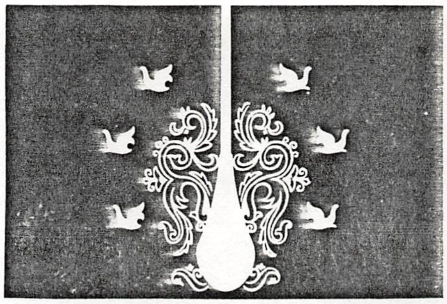
—Its many views and aspects; it s *real* nature (unlike what has been thought of as its nature);

IMAGINE ALL THE CONTROLS AND PRESENTATIONS YOU'D LIKE TO HAVE,
REDUCE THE CONTROLS
AND PRESENTATIONS
TO AN ADEQUATE,
POWERFUL,
EASY-TO-UNDERSTAND SET:

MARRY THEM TO THE AVAILABLE SCREENS, KEYBOARDS AND POINTING TOOLS.

ABOVE ALL, DESIGN THE FULLEST SYSTEM FIRST—THEN CUT IT DOWN, IF YOU HAVE TO. YOU MAY FIND YOU DON'T HAVE TO.

That this is nowhere taught is much worse than regrettable. Because unfortunately the salaried programmer has, in effect, a license to inflict on innocent users anything he likes under the pre-



tense of technical necessity or on the basis of some off-the cuff (or cufflink consultant's) assessment of "user needs."

I regard the decisions involved in designs like those as intricate and interdependent as moves in chess. This kind of design needs a respect and even reverence for the far-flung ramifications of tiny decisions, and the staggering complexity of making things simple.

I hope I have given a sense of this style of design.

I hope, too, that the reader will see it as an art form—somewhere between movies, diagramatics, the design of machinery, the design of games, and the building of philosophical systems.

When done well, it is done with simplicity, consistency, conceptual clarity and vividness. This is not "technical" work in any usual sense. I consider it a form of design and a form of art.

I believe that interactive design is, more than anything else, what the computer field is really about. I find it monstrous and appalling that these general principles are so little understood; that despite all the pompous "computer science curricula," nobody teaches these anywhere; and that innocent customers who want an easy-to-use system—really, is it too much to ask?—are too often led by consultants and tekkies down a primrose path to endless horrors of complication and unnecessary claptrap.

How you feel about all this depends on what you think computers are all about and where the world should be going.

If you want to show off to your family and friends—or financial backers—as a macho master of complicated technicalities, then you don't *want* things to be easily comprehensible. (In that case you should be reading certain other personal computer magazines.

But if you believe that somewhere beyond all the technicalities lies some kind of hope for a better future and a smarter mankind, rich in ideas and knowledge and dreams—as well as gadgets—then the question is how to front-end the gadgets so that they bring us knowledge, and ideas, and dreams, without the technicalities being in the way. □