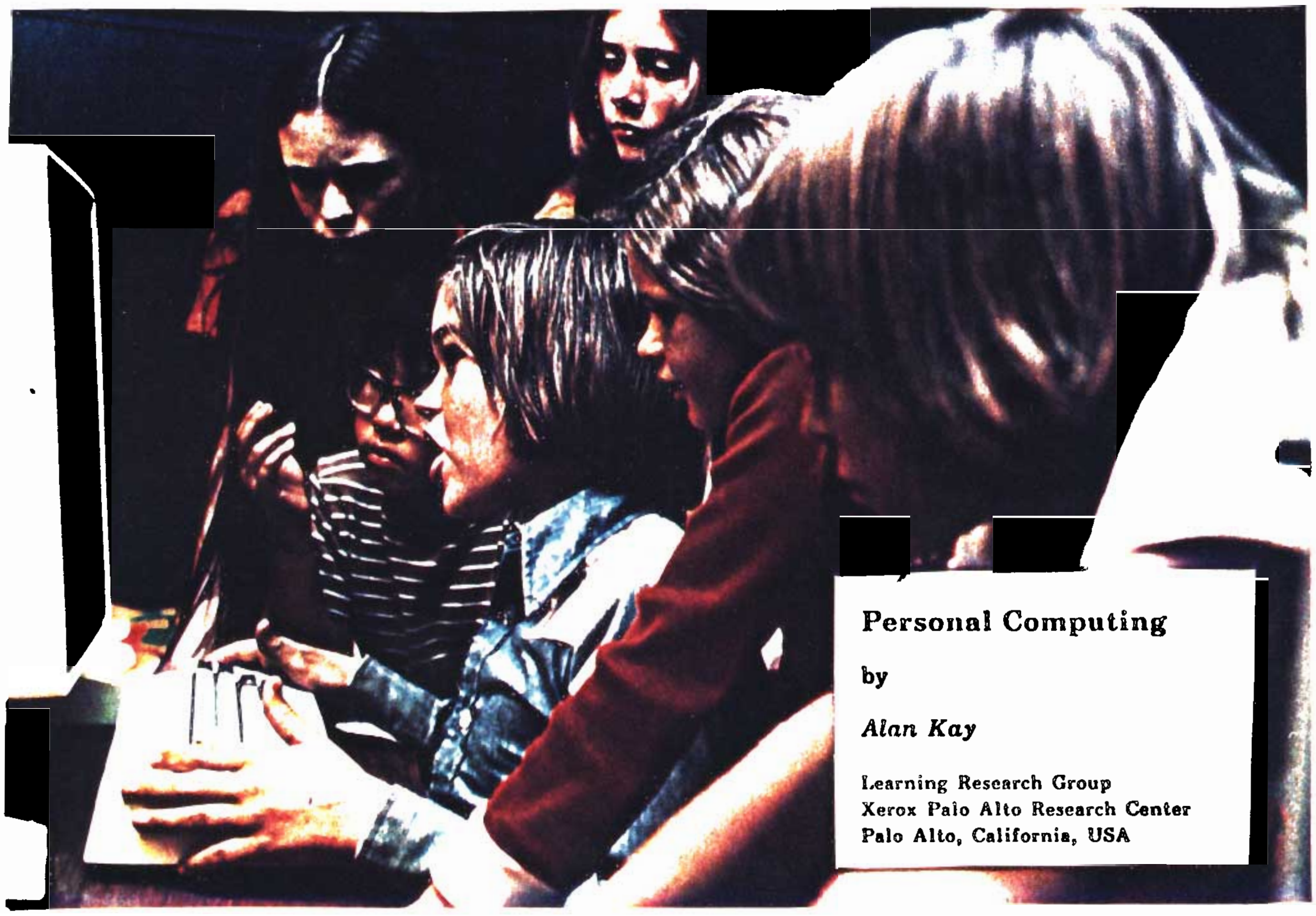
R4348



Personal Computing

by

Alan Kay

Learning Research Group
Xerox Palo Alto Research Center
Palo Alto, California, USA

Invited Paper:
Meeting on 20 Years of Computer Science
Instituto di Elaborazione della Informazione, PISA, ITALY

Invited Paper:
Meeting on 20 Years of Computer Science
Instituto di Elaborazione della Informazione, PISA, ITALY

Personal Computing

by

Alan Kay
Learning Research Group
Xerox Palo Alto Research Center

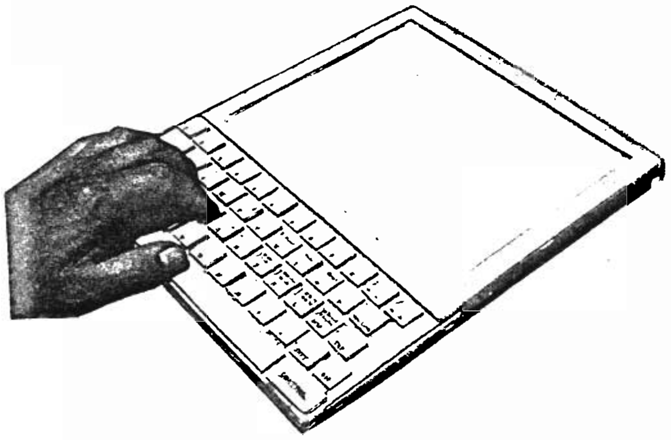
Introduction

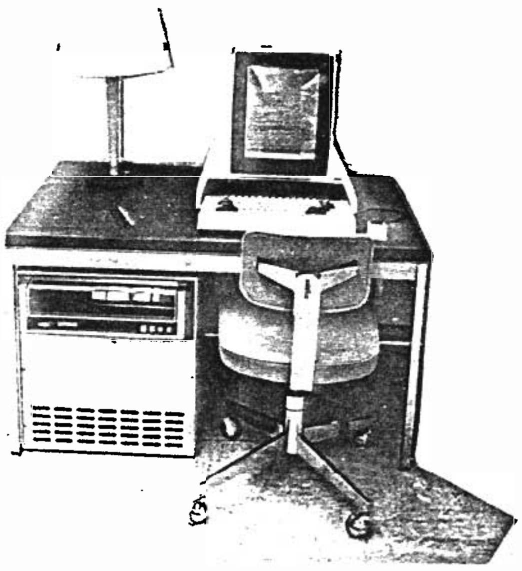
Imagine having your own self-contained knowledge manipulator in a portable package the size and shape of an ordinary notebook. How would you use it if it had enough power to outrace your senses of sight and hearing, enough capacity to store for later retrieval thousands of page-equivalents of reference materials, poems, letters, recipes, drawings, animations, musical scores, waveforms, dynamic simulations, and anything else you would like to create, remember, and change?

Several years ago, we crystallized these long-held desires into a design idea for a personal dynamic medium called the *Dynabook*. We felt that the hardware power for the *Dynabook* would be inevitably available in ten years time, but there was no reason to believe that it would be easily usable by its millions of potential owners if progress in man-computer communication continued at its present rate.

In particular, we wanted our range of users to include children from age 5 or 6 and 'noncomputer adults' such as secretaries, librarians, architects, musicians, housewives, doctors, and so on. We felt strongly that the major design problems of the *Dynabook* lay in the area of communication rather than in new hardware architectures.

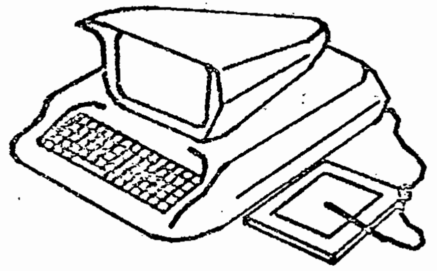
Since it is very difficult to design such an integrated and comprehensive system 'Aristotle fashion' (from one's hammock); we, with others at PARC, designed and built a number of stand-alone self-contained *Interim Dynabooks* in order to have a solid test-bed for our ideas. These machines are the environment for our experimental communications medium, *Smalltalk*. Both the *Interim Dynabooks* and *Smalltalk* have been used by children and adults.

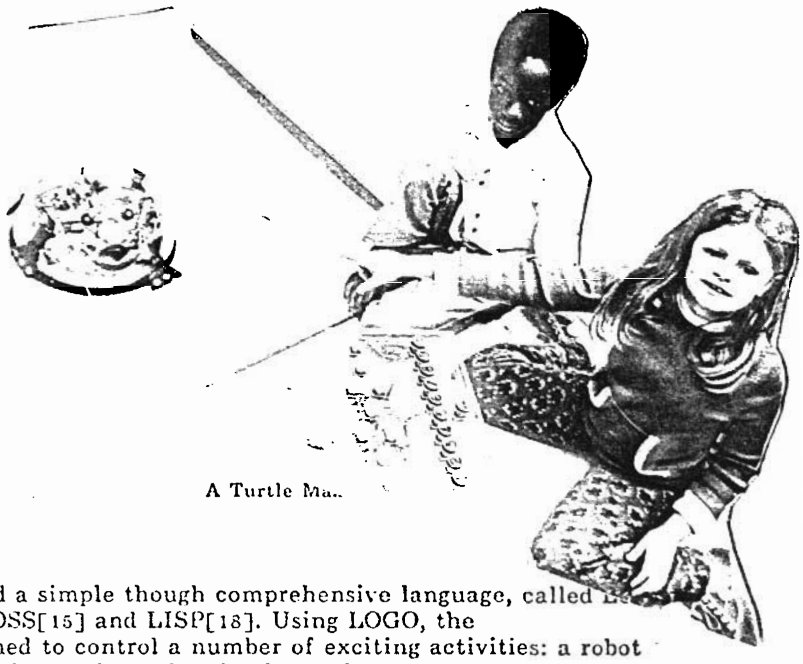

The *Dynabook* Mockup


The *Interim Dynabook*

Design Background

The first attempt at designing this kind of higher-level personal *metamedium* was the development of the FLEX Machine in 1967-69[1,2,3]. Much of the hardware and software was successful from the standpoint of computer science state-of-the-art research but, as is so often the case, lacked sufficient expressive power to be truly useful to an ordinary user. At that time started to appear Papert and Feurzeig's[9,10,11] pioneering work having to do with helping kids learn how to think by giving them an environment in which thinking is both fun and rewarding.


A Drawing of the FLEX Machine Shown On Its Own Display Screen ca. 1968[3]


A Turtle Ma..

They chose a time-shared computer and devised a simple though comprehensive language, called which combined some of the best features of JOSS[15] and LISP[18]. Using LOGO, the children (ranging in age from 8-12 years) learned to control a number of exciting activities: a robot turtle which can draw, a faster CRT version of the turtle, and a simple music generator.

The LOGO work radiates a compelling excitement in several directions.

First, the children really can program the turtle and the music box to do serious things. The programs use symbols to stand for objects, contain loops and recursions, require a fair amount of visualization of alternate strategies before a tactic is chosen, and involve interactive discovery and removal of 'bugs' in their ideas. As Papert points out, the children are performing real *mathematical* acts of a kind, scope, and level not ever achieved by many college graduates.

Second, the kids love it! The interactive nature of the dialogue, the fact that *they* are in control, the feeling that they are doing *real* things rather than playing with toys or working out 'school' problems, the pictorial and auditory nature of their results, all contribute a tremendous sense of accomplishment to their experience. Their attention spans are measured in hours rather than minutes.

After seeing the faces of children suddenly discovering that they are "doers" acting on the world, rather than "things" being acted upon, it was clear that the next attempt to design a personal computer should be done with children strongly in mind.

First, having children as users would throw sharply into focus the expressive communication problems which had caused difficulty with the FLEX Machine. In addition, it might be possible to discover why the LOGO kids had certain difficulties of their own having to do with naming and parameters, with partitioning their problems reasonably, and why they appeared to reach a plateau: they could design and write certain kinds of constructive programs but never quite got to the point where they could do a real *system.*

Second, children really need more computing and expressive power than most adults are willing to settle for when using a time-sharing system; the best that time-sharing can offer directly is slow control of crude wire-frame green-tinted graphics and (even cruder) square-wave 'musical' tones. The kids, on the other hand, are used to high-bandwidth media such as as finger-paints, water colors, color TV, real musical instruments, and hi-fi records. If the 'medium is the message', then the message of low bandwidth time-sharing is 'blah'!

Our Approach

At the outset, we decided to admit that the design of a truly useful dynamic medium for everyday use was a hard but extremely worthwhile problem which would require both many years and several complete interim hardware/software systems to be designed, built, and tested. Approach:

1. *Conceptualize a "Holy Grail" version of what the eventual Dynabook should be like in the future.*

This image will provide a rallying point and goal which will remind us of what we are trying to do while the sometimes grubby spadework of producing intermediate systems is in progress.
An extrapolation (and compression) of the FLEX Machine[4].

2. *Do the research in human factors, psychology of perception, physics, and language design which is prerequisite to any serious attempt at an interim system.*

Very few displays have been designed using any knowledge of the human visual system nor have many artificial languages been developed on non-Indo European models.
An overview of what the Dynabook should be like, including display needs and principles for language design, are found in[4].

3. *Design an interim version of the Dynabook, and build a considerable number of them.*

We felt that this step and the next one are the critical ones in our research. We had to get to the kids and adults as quickly as possible so as not to be led astray by our own assumptions and hopes.

4. *Make the medium of communication as simple and powerful as possible.*

It should be qualitatively simpler and qualitatively more powerful than (say) LOGO. It should be qualitatively more expressive than the best state-of-the-art "grown-up" programming language for serious system design. It should be as "neutral" as possible to all conceivable simulations.

5. *Explore the usefulness of such a system with a large number of short range projects involving many users, ages 4 to 60, from varying backgrounds, and with different needs and goals.*

This phase would involve developing all manner of simulated media, both old and new; finding ways to teach the ideas in the system; do user studies, experiment with peer-group teaching (e.g. 13 year olds teaching 12 year olds), and so on.
Some of the projects we have undertaken are explored in [6,7,8].

6. *Re-extend the system in the light of our previous study and start to think about the next interim version.*

One of our guiding philosophies has been to do many working versions rather than to attempt a long-range 'complete solution' in one fell swoop and risk not ever getting a working system.

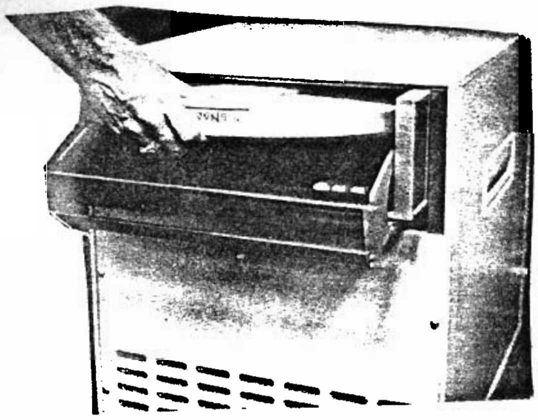
7. *Our current plans are to set up a community resource center containing a number of the interim Dynabook systems for both open- and closed-shop use near school and playground 'traffic' patterns.*

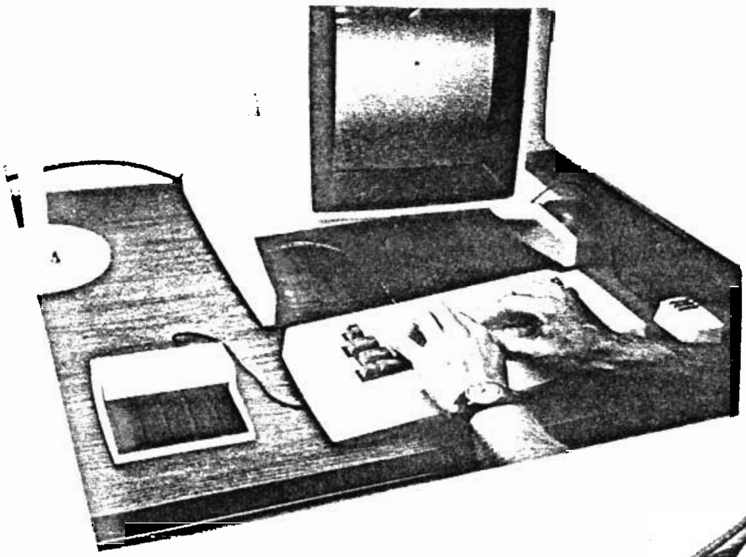
In order to find out some of the things we would like to know about how children and adults think about their world, we really need to conduct a series of longitudinal studies which investigate how daily and casual use of a dynamic medium affect people's way of doing things (and their lives).
We currently expect to start setting up this facility in the very near future.

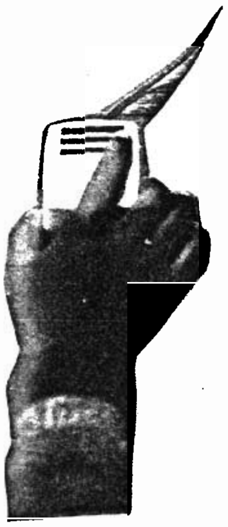
The Interim Dynabook and Smalltalk .

The *Interim Dynabook* is a completely self-contained system, *designed-as-a-whole* from simulations and human factors experiments. To the user, it appears as a small box (in which can be inserted a disk memory containing about 1500 page-equivalents of manipulable storage) connected to a very crisp high resolution B&W CRT or a lower resolution high-quality color display.

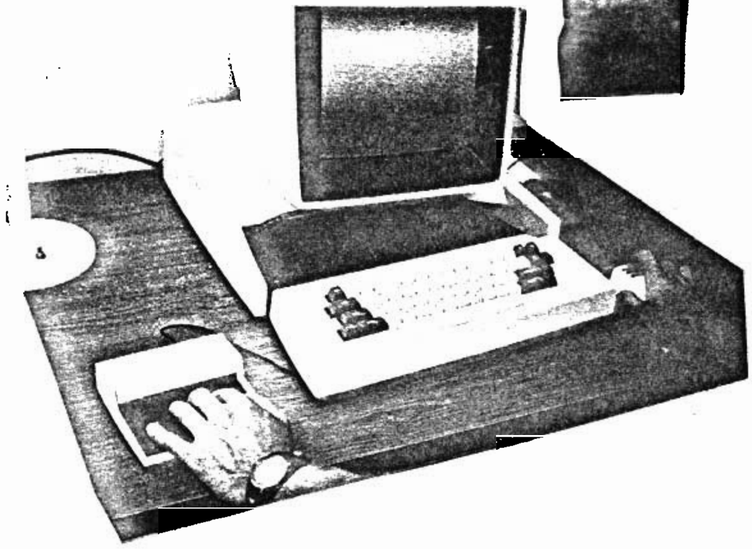
Other input devices include a standard typewriter keyboard, a 'chord' keyboard (for sending simultaneous signals), a 'mouse' (which inputs position as it is moved about on the table), and varieties of organ-like keyboards for playing music. New input devices such as these may be easily attached, usually without building a hardware interface for them.


Inserting A Disk Memory

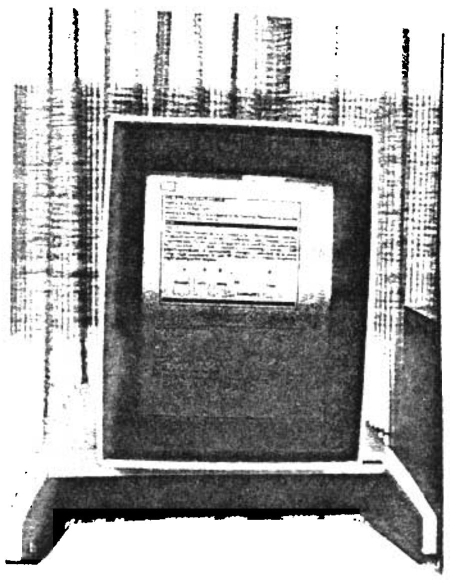

The Typing Keyboard

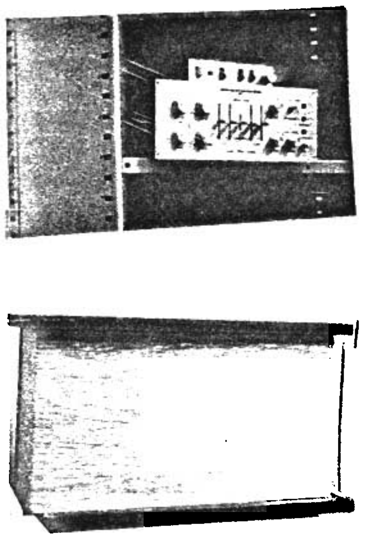

The Mouse

The Music Keyboards


Chord Keyboard on Left, Mouse on Right

Visual output is through the display, auditory output is obtained from a built in digital-to-analog converter connected to a standard hi-fi amplifier and speakers.


High Resolution B and W Display


Amplifier and Speaker

We will see in the next section that the *Interim Dynabook* is powerful enough to be able to produce real-time half-tone video animation, and real-time multiple-voice musical-tone synthesis.

Smalltalk is a very simple, comprehensive way of simulating dynamic models. The built-in primitives of most programming languages (such as numbers, files, data structures, etc.), in Smalltalk, are actually *simulations* built from more comprehensive ideas, including *states-in-process, communication using messages, and classes and instances.*

Two of its basic goals are that *simple things should be very simple*, one should not have to read a manual to do obvious things; and, *complex things should be very possible*, comprehensive interactive systems should be easily programmed without 'hair or prayer'.

As will be seen later in the paper, Smalltalk is a successful 'extensible language' because it focuses squarely on representing the meanings *of things through* descriptions *and gets its syntactic extensibility 'for free'.*

Smalltalk's main lineage can be traced to a combination of new control ideas found in CDL, with FLEX and its precursors: Sketchpad, SIMULA, JOSS, and the B5000 [18,1-3,12-16].

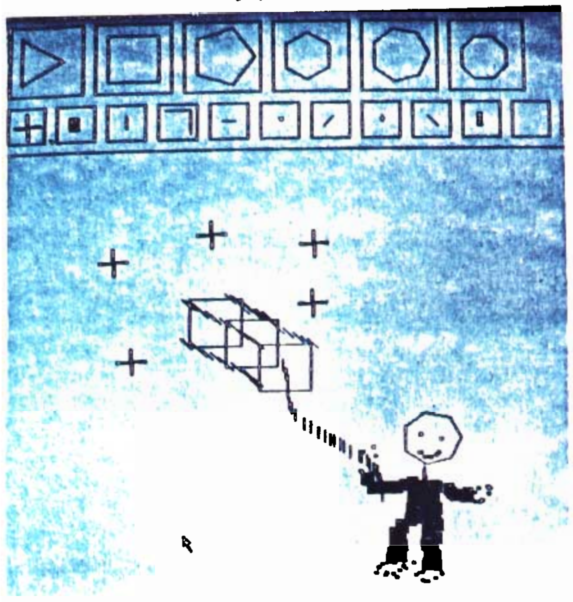
Interim Dynabook Rotogravure

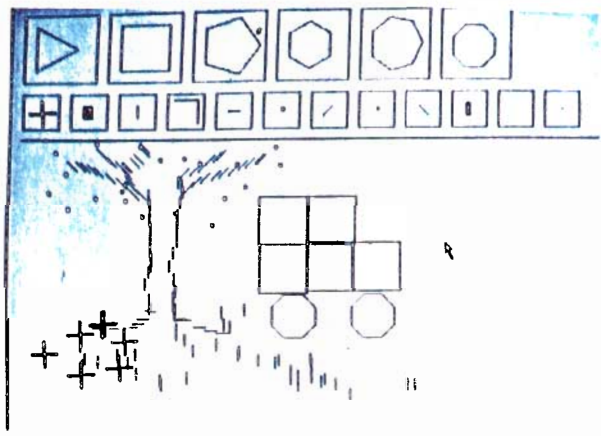
Before we discuss the ideas behind our current system, we present a sample portfolio of some of the projects which kids and adults have created in the past two years.

A Menu Driven Painting Program: *Programmed by a 12 year-old*

This was the first real *system* done by any of our children; there have since been many more. The child did not see any of our painting programs while working on this. She, very early in her introduction to Smalltalk, decided that one *ought* to be able to paint if the *mouse* could tell the *pen* where to go, *over and over*. She came in the next morning, wrote the program, and it worked the first time. After that, she became positively ingenious in finding ways to turn every possible programmatic situation into a way to paint. The idea of having various kinds of brushes and paints was hers; the notion of having a 'menu' of the available repetoire was suggested by us, the method for achieving the menu was designed and programmed by her.

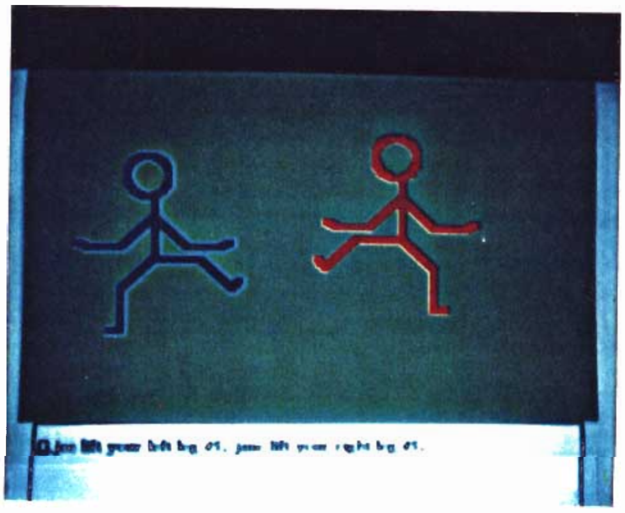
This was the first indication to us that the building blocks of Smalltalk actually were more powerful and easier to use for the naive programmer than the more conventional 'noun/verb' ('data structure/function') primitive ideas of most current programming systems.



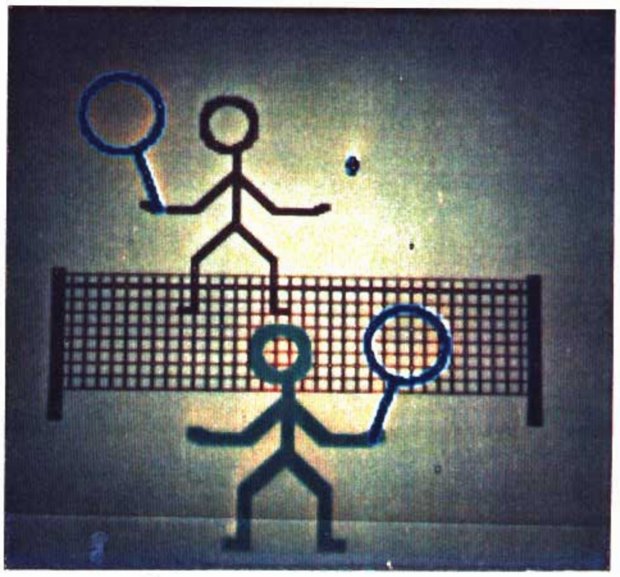


A Choreography/Badminton System

This child next designed a class of dancers which could be controlled singly and together, then choreographed them into a variety of dances. Next, she designed a 'marionette' class which had wrist, elbow, shoulder, ankle, knee, and hip joints. Two of these more sophisticated figures are used for her real-time badminton simulation.

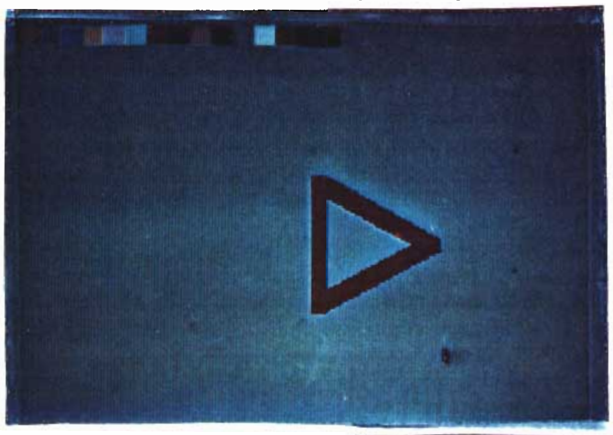


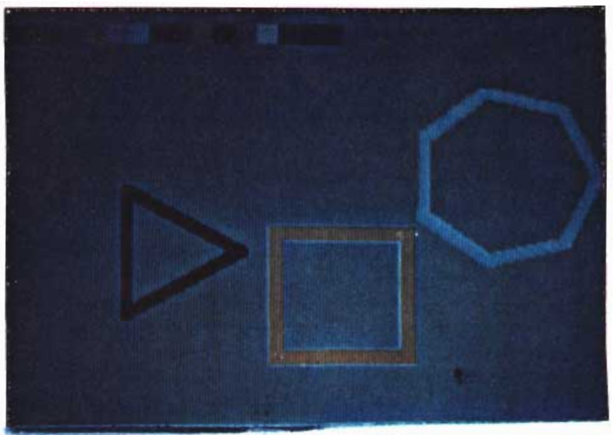
[illegible] lift your left leg 01. [illegible] lift your right leg 01.

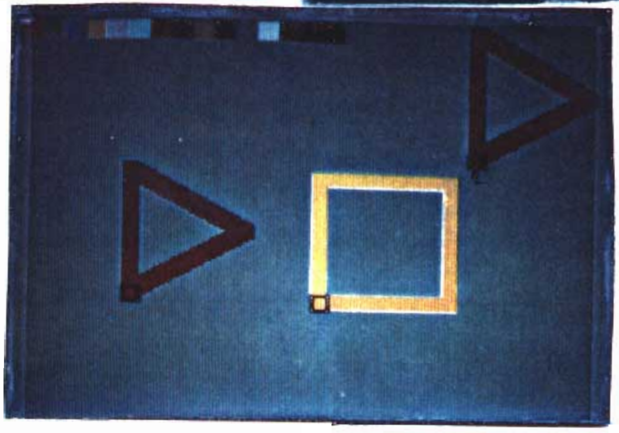


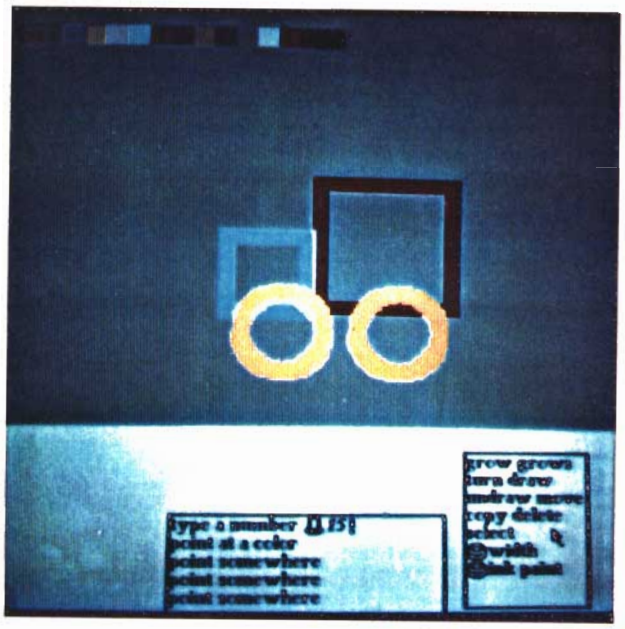
An Illustration System: *Programmed by a 12-year old*

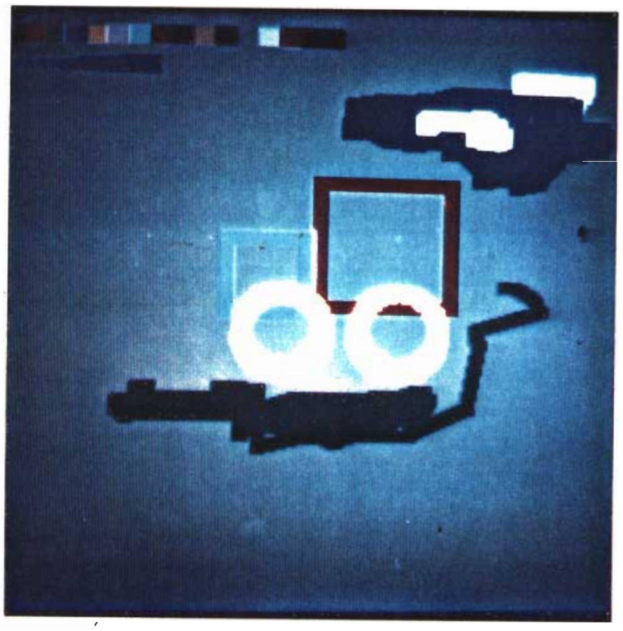
This is one of the most complex systems done by one of our children. The graphic objects can be located, rotated, scaled, can have any number of sides, are any color which the system can produce, and the entire system is controlled by prompting menus. Note also the 'light buttons' on the objects which are sites for 'grabbing' and moving and object with the mouse.





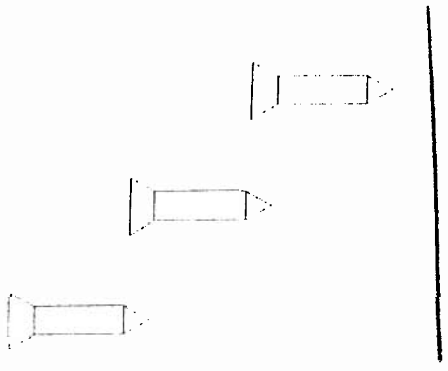


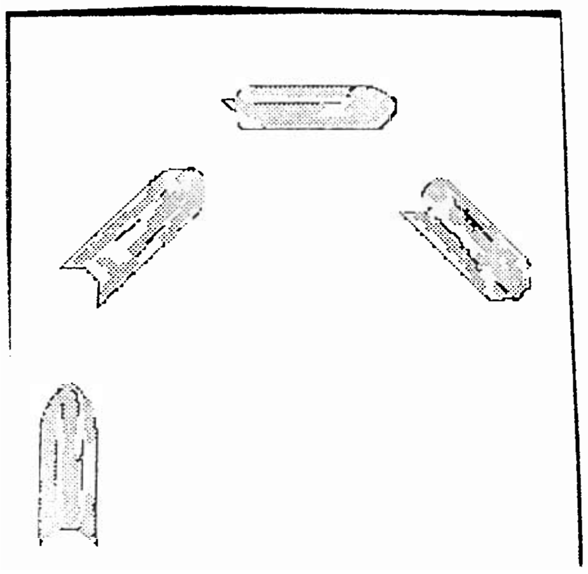


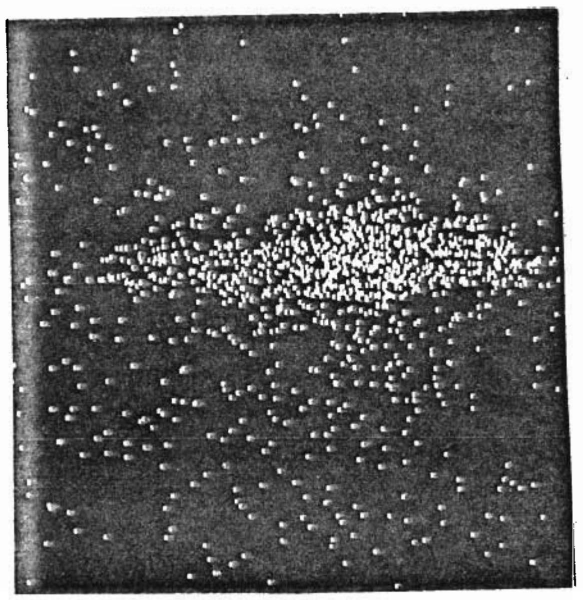


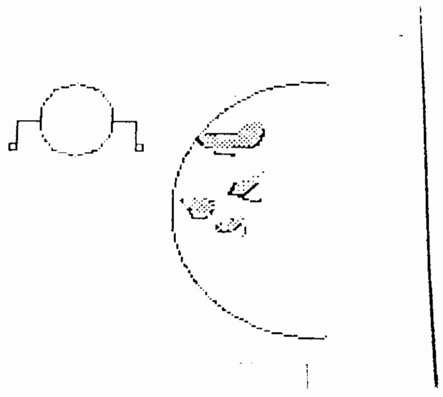
Spacewar: *Programmed by 10-12-year olds*

Spacewar blossoms spontaneously wherever a graphics display is connected to a digital computer. There have been many different versions done by the kids in the past several years, ranging from peaceful moon landings and fleets of cargo ships, to the full fledged game. Star fields (complete with novae and supernovae) form an interesting background.





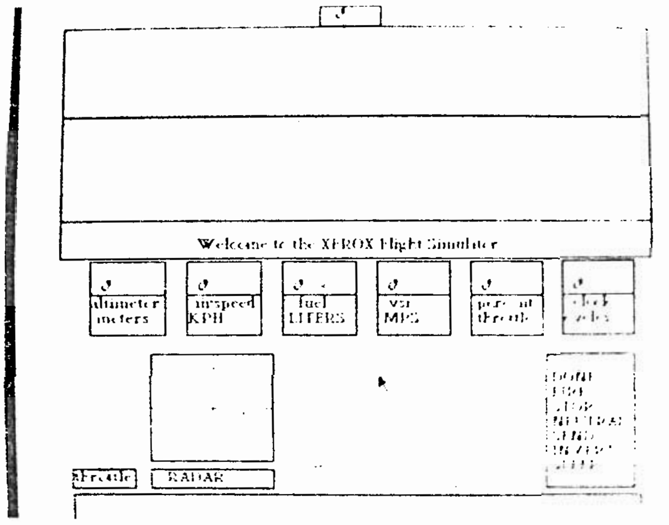


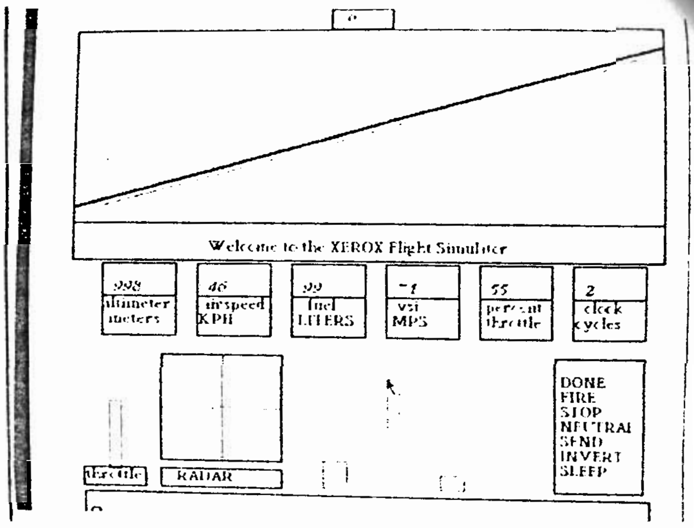


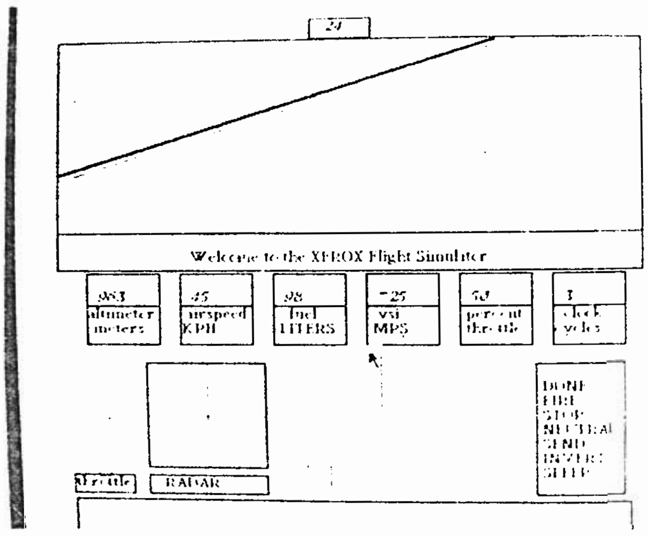
DATE 12-JUN-75 10:19:38

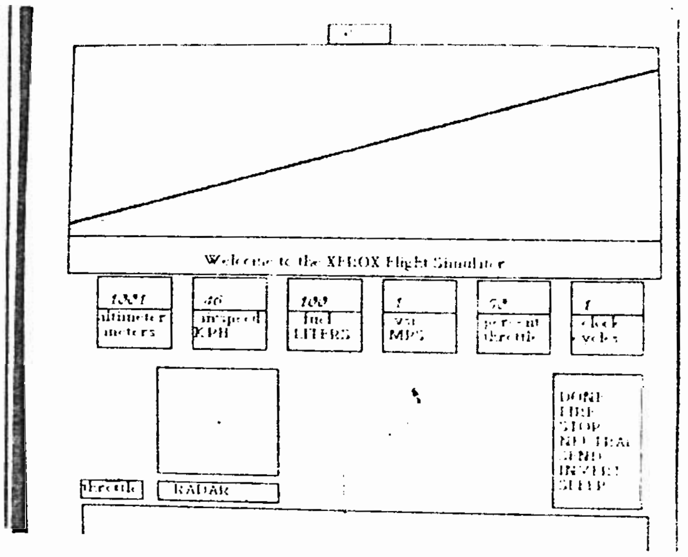
A Flight Simulator: *Programmed by a 15-year old.*

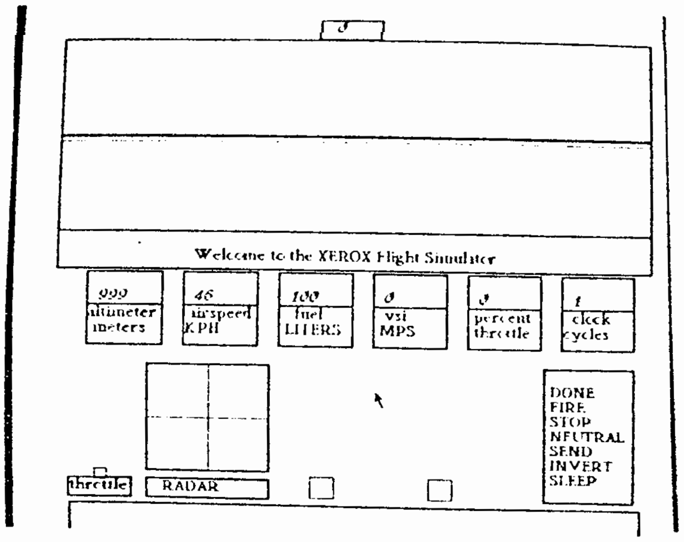
This rather complex system uses an actual flight model which the student found in an introductory book on aeronautics. The artificial horizon is at the top of the panel with the degrees-of-bank over it. The 'stick', pedals, and throttle can all be grabbed by the mouse. The sequence shows the plane in a slow roll which is then corrected by manipulating the stick.







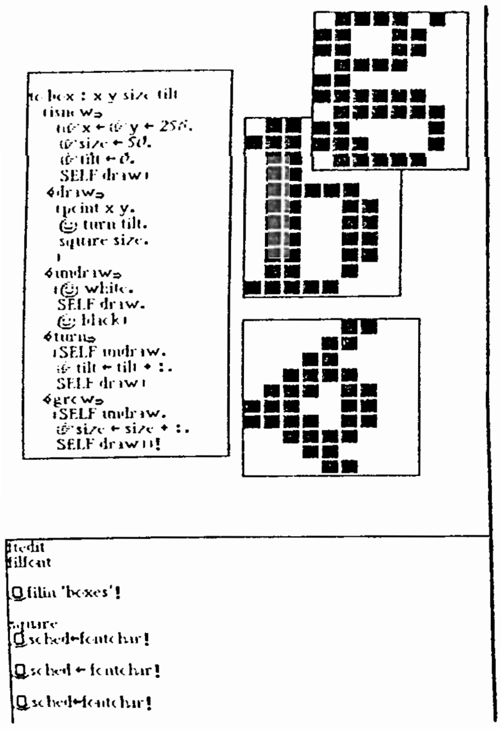




Windows and Different Fonts for Different Effects

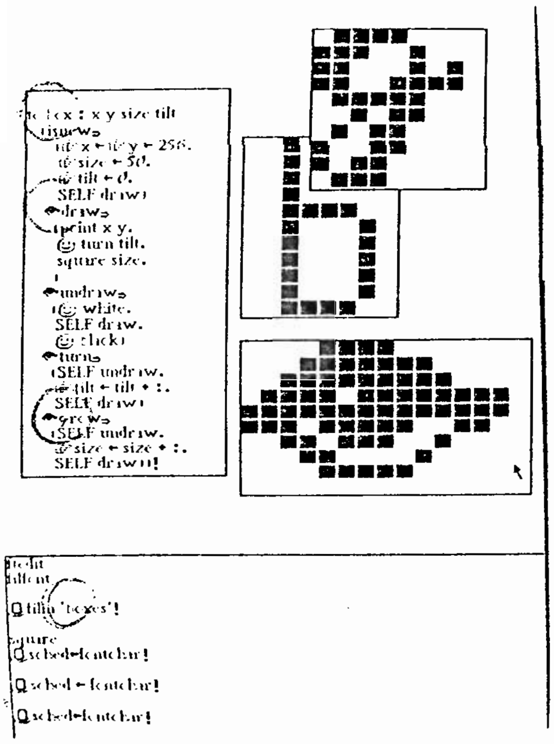
The flight simulator shown previously used Smalltalk *windows* for displaying instrument values and the horizon. Each window has as its content a Smalltalk object; they can be moved, stretched, overlapped, and edited.

One of our goals was not to be worse than paper in any important way. The *Dynabook* is flexible to the point of allowing its owner to describe just how knowledge is to be viewed, including tailor-made printing quality fonts. Any character font can be described as a matrix of black and white dots. The corners of the dots will not be perceived at normal viewing distance if the light level from the display is high enough and the resolution is sufficient. Here we see a font in the process of being created and displays of text in some of the fonts which are available.



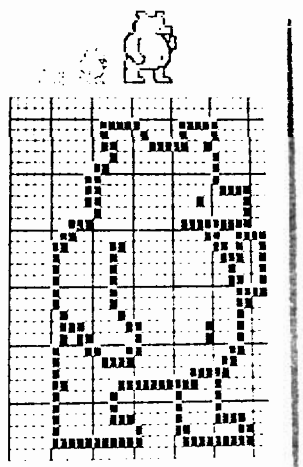
```
to box : x y size tilt
 (isnew ⇒
   (@x ← @y ← 256.
    @size ← 50.
    @tilt ← 0.
    SELF draw)
 ◇draw ⇒
   (point x y.
    ☺ turn tilt.
    square size.
   )
 ◇undraw ⇒
   (☺ white.
    SELF draw.
    ☺ black)
 ◇turn ⇒
   (SELF undraw.
    @ tilt ← tilt + :.
    SELF draw)
 ◇grow ⇒
   (SELF undraw.
    @size ← size + :.
    SELF draw))!
```

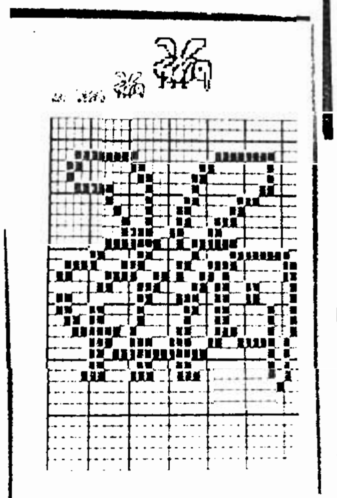
```
tedit
illfont
☐filin 'boxes'!
square
☐sched←fontchar!
☐sched ← fontchar!
☐sched←fontchar!
```

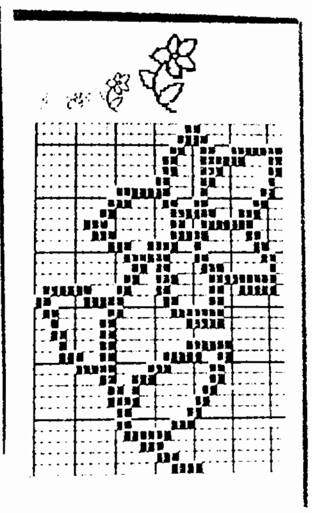


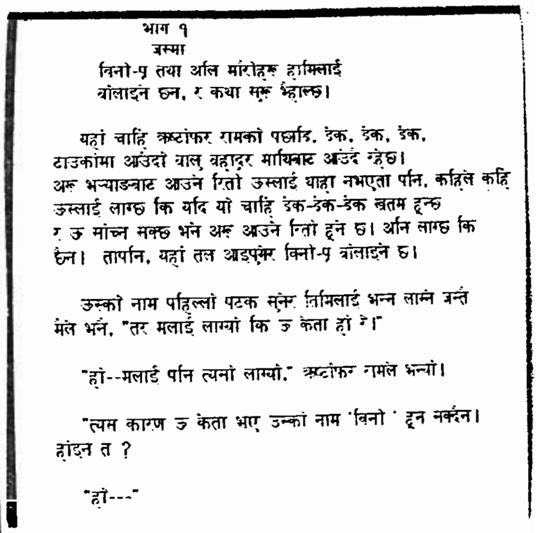
```
to box : x y size tilt
 (isnew ⇒
   (@x ← @y ← 256.
    @size ← 50.
    @tilt ← 0.
    SELF draw)
 ◇draw ⇒
   (point x y.
    ☺ turn tilt.
    square size.
   )
 ◇undraw ⇒
   (☺ white.
    SELF draw.
    ☺ black)
 ◇turn ⇒
   (SELF undraw.
    @ tilt ← tilt + :.
    SELF draw)
 ◇grow ⇒
   (SELF undraw.
    @size ← size + :.
    SELF draw))!
```

```
tedit
illfont
☐filin 'boxes'!
square
☐sched←fontchar!
☐sched ← fontchar!
☐sched←fontchar!
```









भाग १
जस्मा
विनो-प्र तथा अलि मारोहरू हामिलाई
वोलाइने छन, र कथा सुरू भ्हाल्छ।

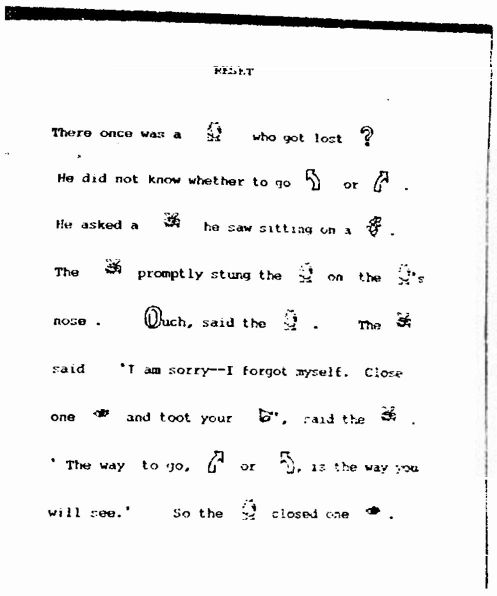
यहां चाहि क्रिस्टोफर रामको पछाडि, डक, डक, डक,
टाउकोमा अउंदो याल् वहादुर मायिबाट अउंद रहेछ।
अरू भन्याङबाट आउने रितो उस्लाई याहा नभएता पनि, कहिले कहि
उस्लाई लाग्छ कि यदि यो चाहि डक-डक-डक खतम हुन्छ
र उ मोच्न सक्छ भने अरू आउने रितो हुने छ। अनि लाग्छ कि
छैन। तापनि, यहां तल आइपुगेर विनो-प्र वोलाइने छ।

उस्को नाम पहिल्लो पटक सुनेर तिमिलाई भन्न लाग्ने जस्तै
मेले भने, "तर मलाई लाग्यो कि उ केता हो ?।"

"हो--मलाई पनि त्यसो लाग्यो," क्रिस्टोफर रामले भन्यो।

"त्यस कारण उ केता भए उस्को नाम 'विनो' हुन सक्दैन।
होइन त ?

"हो---"



RESET

There once was a who got lost ?

He did not know whether to go or .

He asked a he saw sitting on a .

The promptly stung the on the 's

nose . Ouch, said the . The

said 'I am sorry--I forgot myself. Close

one and toot your ", said the .

' The way to go, or , is the way you

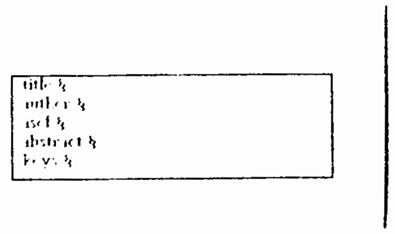
will see.' So the closed one .

Filing and Editing

Every description in the *Dynabook* can be retrieved, shown, and edited in a completely uniform way. Each class of objects (*text, pictures, fonts, movies, choruses*) responds to the message *show* by invoking its own method for showing itself on the screen. The displayed objects actively watch for the the cursor to enter their boundaries, and if it does, the message *edit* is sent to the object. In a manner similar to the *show* message, each class of objects can respond to the message *edit* in its own appropriate fashion.

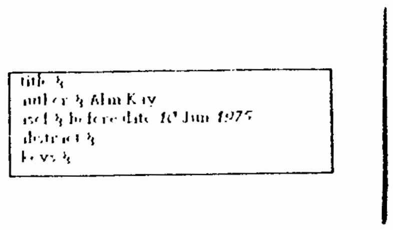
If the object is text, an easy to use and modify 'modeless' text editor with automatic justification is invoked. Drawings and paintings have their own simple editors, as do musical scores, timbres, etc. A *document* is simply a collection of related objects which may be automatically crossfiled by contents including title, author, date range, selected keys, and anything else the owner desires. A new class of objects may be added to a document in a completely independent fashion because no part of the system has to be informed; instead, the new class simply adopts the conventional protocol of being able to respond to the standard messages *show, hascursor, edit* with useful local methods for achieving these goals.

The current version of this system is able to automatically crossfile tens of thousands of objects including textual documents indexed by content, the *Smalltalk* system itself, personal records, books, and so on. As shown in the examples, retrieval is done by simply filling in as much as is known about the document into a blank template; the system will retrieve a collection of documents which fit the description.

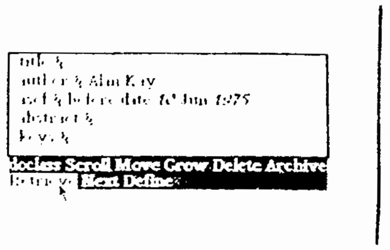


title §
author §
used §
abstract §
keys §

A Blank Document Class Template

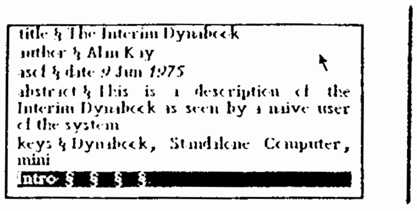


title §
author § Alan Kay
used § before date *10 Jun 1975*
abstract §
keys §



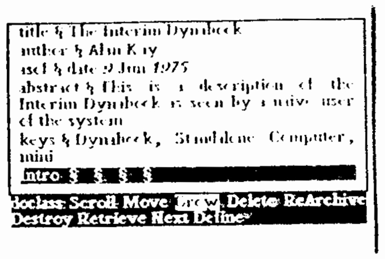
title §
author § Alan Kay
used § before date *10 Jun 1975*
abstract §
keys §

Filling in Identity Fields (any known combination, will do), Invoking a Menu and Asking for a Retrieve



title § The Interim Dynabook
author § Alan Kay
used § date *9 Jun 1975*
abstract § This is a description of the Interim Dynabook as seen by a naive user of the system
keys § Dynabook, Standalone Computer, mini
intro § § § §

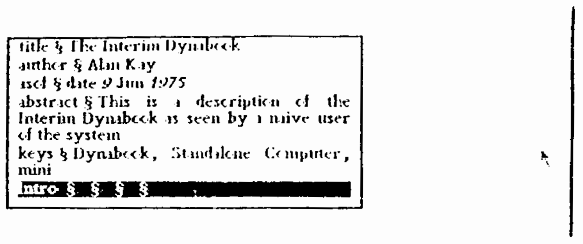
The Document is Found
Reversed Text Indicates Submerged Parts



title § The Interim Dynabook
author § Alan Kay
used § date *9 Jun 1975*
abstract § This is a description of the Interim Dynabook as seen by a naive user of the system
keys § Dynabook, Standalone Computer, mini
intro § § § §

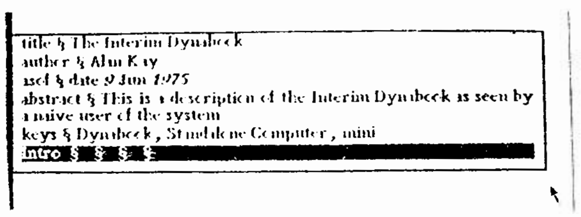
doclass Scroll Move Grow Delete ReArchive
Destroy Retrieve Next Define

A Menu is Invoked at Bottom Margin
We want to make the template wider .....



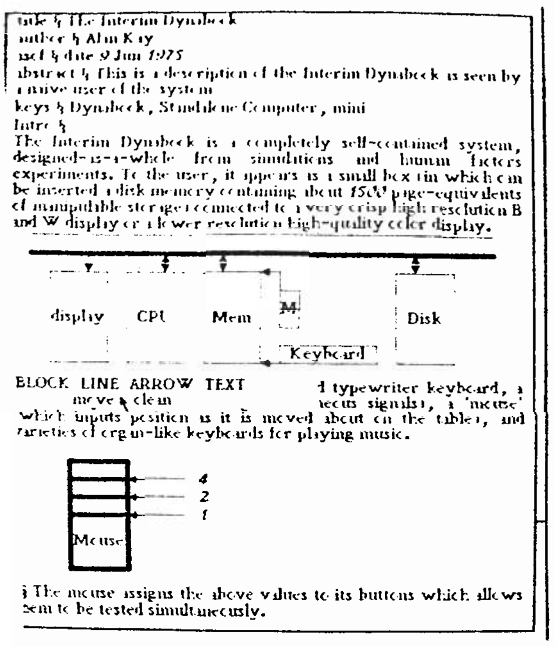
title § The Interim Dynabook
author § Alan Kay
used § date *9 Jun 1975*
abstract § This is a description of the Interim Dynabook as seen by a naive user of the system
keys § Dynabook, Standalone Computer, mini
intro § § § §

Showing the Document Where to Put The Right Margin



title § The Interim Dynabook
author § Alan Kay
used § date *9 Jun 1975*
abstract § This is a description of the Interim Dynabook as seen by a naive user of the system
keys § Dynabook, Standalone Computer, mini
intro § § § §

The Widened Document
Note Automatic Justification



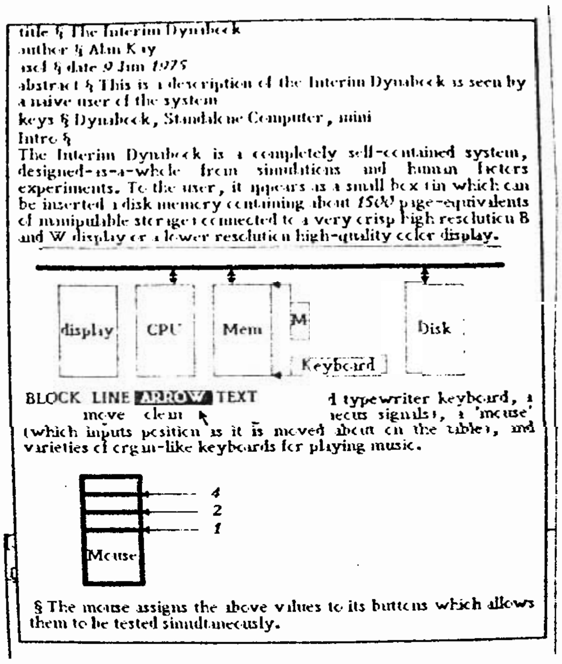
title § The Interim Dynabook
author § Alan Kay
used § date 9 Jun 1975
abstract § This is a description of the Interim Dynabook as seen by a naive user of the system
keys § Dynabook, Standalone Computer, mini
Intro §
The Interim Dynabook is a completely self-contained system, designed-as-a-whole from simulations and human factors experiments. To the user, it appears as a small box (in which can be inserted a disk memory containing about 1500 page-equivalents of manipulable storage) connected to a very crisp high resolution B and W display or a lower resolution high-quality color display.

BLOCK LINE ARROW TEXT
move clean

d typewriter keyboard, a iecus signals), a 'mouse' (which inputs position as it is moved about on the table), and varieties of organ-like keyboards for playing music.

§ The mouse assigns the above values to its buttons which allows them to be tested simultaneously.

Moving Into The First Drawing



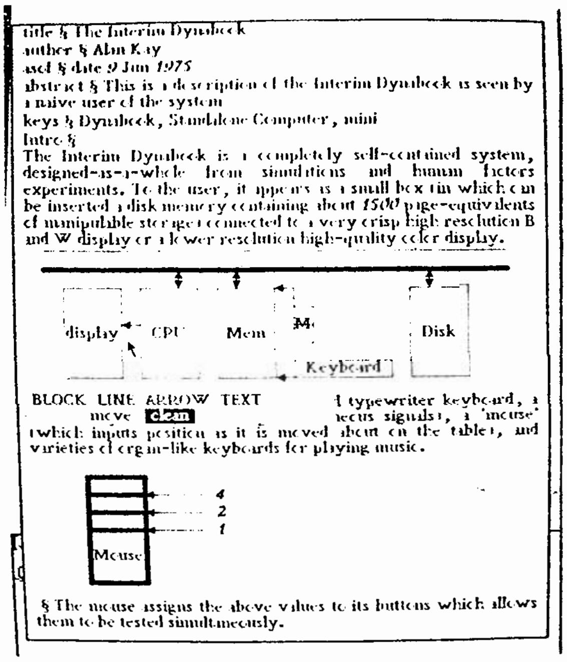
title § The Interim Dynabook
author § Alan Kay
used § date 9 Jun 1975
abstract § This is a description of the Interim Dynabook as seen by a naive user of the system
keys § Dynabook, Standalone Computer, mini
Intro §
The Interim Dynabook is a completely self-contained system, designed-as-a-whole from simulations and human factors experiments. To the user, it appears as a small box (in which can be inserted a disk memory containing about 1500 page-equivalents of manipulable storage) connected to a very crisp high resolution B and W display or a lower resolution high-quality color display.

BLOCK LINE ARROW TEXT
move clean

d typewriter keyboard, a iecus signals), a 'mouse' (which inputs position as it is moved about on the table), and varieties of organ-like keyboards for playing music.

§ The mouse assigns the above values to its buttons which allows them to be tested simultaneously.

Changing The Arrow



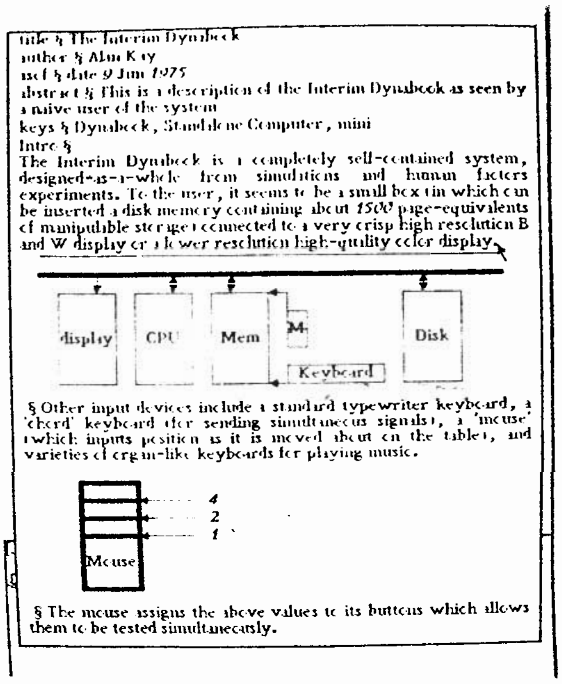
title § The Interim Dynabook
author § Alan Kay
used § date 9 Jun 1975
abstract § This is a description of the Interim Dynabook as seen by a naive user of the system
keys § Dynabook, Standalone Computer, mini
Intro §
The Interim Dynabook is a completely self-contained system, designed-as-a-whole from simulations and human factors experiments. To the user, it appears as a small box (in which can be inserted a disk memory containing about 1500 page-equivalents of manipulable storage) connected to a very crisp high resolution B and W display or a lower resolution high-quality color display.

BLOCK LINE ARROW TEXT
move clean

d typewriter keyboard, a iecus signals), a 'mouse' (which inputs position as it is moved about on the table), and varieties of organ-like keyboards for playing music.

§ The mouse assigns the above values to its buttons which allows them to be tested simultaneously.

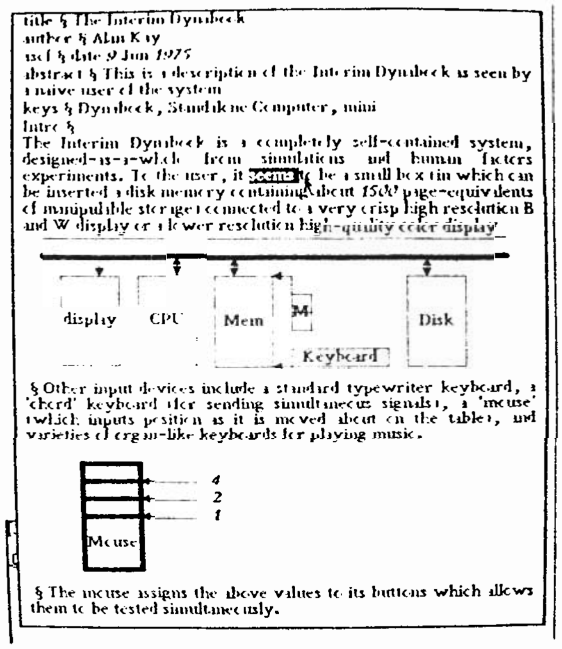
Now The Arrow Points To The CPU Box


title § The Interim Dynabook
author § Alan Kay
ascl § date 9 Jun 1975
abstract § This is a description of the Interim Dynabook as seen by a naive user of the system
keys § Dynabook, Standalone Computer, mini
Intro §
The Interim Dynabook is a completely self-contained system, designed-as-a-whole from simulations and human factors experiments. To the user, it seems to be a small box (in which can be inserted a disk memory containing about *1500* page-equivalents of manipulable storage) connected to a very crisp high resolution B and W display or a lower resolution high-quality color display.

§ Other input devices include a standard typewriter keyboard, a 'chord' keyboard (for sending simultaneous signals), a 'mouse' (which inputs position as it is moved about on the table), and varieties of organ-like keyboards for playing music.

§ The mouse assigns the above values to its buttons which allows them to be tested simultaneously.

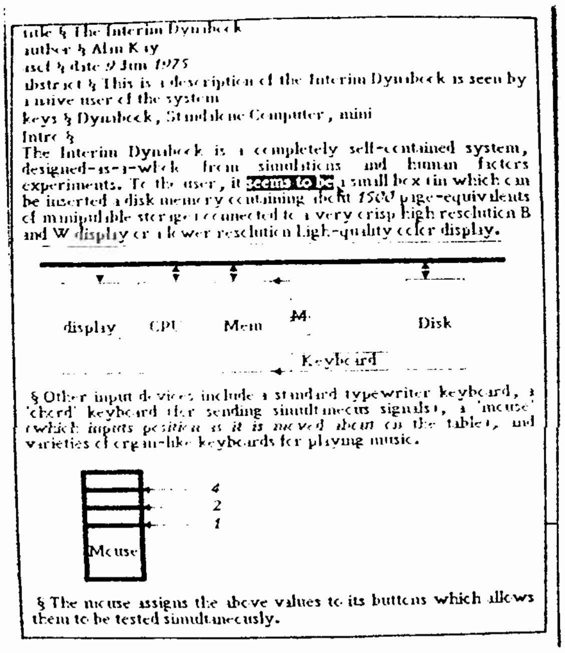
Opening up the Submerged Parts


title § The Interim Dynabook
author § Alan Kay
ascl § date 9 Jun 1975
abstract § This is a description of the Interim Dynabook as seen by a naive user of the system
keys § Dynabook, Standalone Computer, mini
Intro §
The Interim Dynabook is a completely self-contained system, designed-as-a-whole from simulations and human factors experiments. To the user, it seems to be a small box (in which can be inserted a disk memory containing about *1500* page-equivalents of manipulable storage) connected to a very crisp high resolution B and W display or a lower resolution high-quality color display.

§ Other input devices include a standard typewriter keyboard, a 'chord' keyboard (for sending simultaneous signals), a 'mouse' (which inputs position as it is moved about on the table), and varieties of organ-like keyboards for playing music.

§ The mouse assigns the above values to its buttons which allows them to be tested simultaneously.

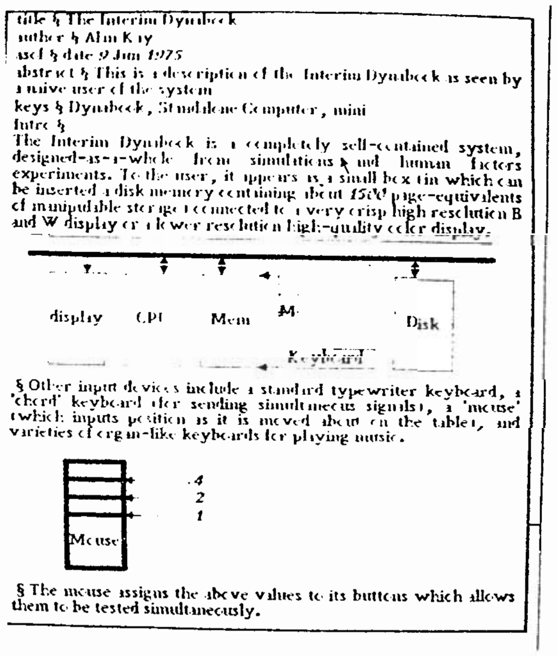
Grabbing Text by
Drawing Through It


title § The Interim Dynabook
author § Alan Kay
ascl § date 9 Jun 1975
abstract § This is a description of the Interim Dynabook as seen by a naive user of the system
keys § Dynabook, Standalone Computer, mini
Intro §
The Interim Dynabook is a completely self-contained system, designed-as-a-whole from simulations and human factors experiments. To the user, it seems to be a small box (in which can be inserted a disk memory containing about *1500* page-equivalents of manipulable storage) connected to a very crisp high resolution B and W display or a lower resolution high-quality color display.

§ Other input devices include a standard typewriter keyboard, a 'chord' keyboard (for sending simultaneous signals), a 'mouse' *(which inputs position as it is moved about on the table)*, and varieties of organ-like keyboards for playing music.

§ The mouse assigns the above values to its buttons which allows them to be tested simultaneously.

Grabbing More


title § The Interim Dynabook
author § Alan Kay
ascl § date 9 Jun 1975
abstract § This is a description of the Interim Dynabook as seen by a naive user of the system
keys § Dynabook, Standalone Computer, mini
Intro §
The Interim Dynabook is a completely self-contained system, designed-as-a-whole from simulations and human factors experiments. To the user, it appears as a small box (in which can be inserted a disk memory containing about *1500* page-equivalents of manipulable storage) connected to a very crisp high resolution B and W display or a lower resolution high-quality color display.

§ Other input devices include a standard typewriter keyboard, a 'chord' keyboard (for sending simultaneous signals), a 'mouse' (which inputs position as it is moved about on the table), and varieties of organ-like keyboards for playing music.

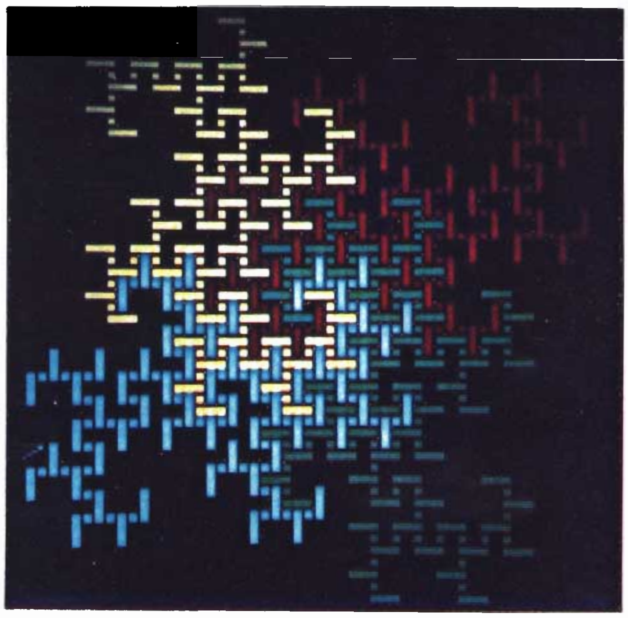
§ The mouse assigns the above values to its buttons which allows them to be tested simultaneously.

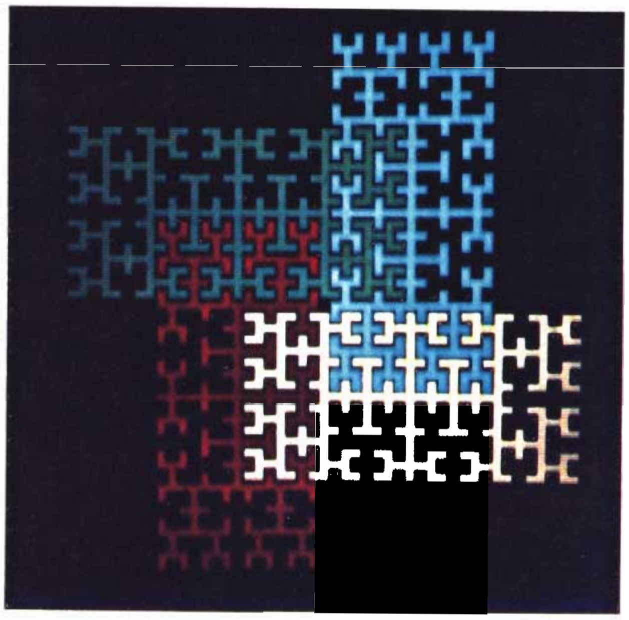
Replacement is Automatic
(without command)

Curves

Smalltalk has a class of *turtles* which can crawl about on the screen leaving (or not) a variable width track of colored ink. A *turtle* has a *location*, a *direction*, a *tip-state(up or down)*, an *ink-color*, a *boundary* (which clips it), and any other properties the owner might like. A *turtle* can be *turned* a relative angle, can *go* a distance in its *direction*, can *goto* a location, and so on.

A powerful idea (borrowed from LOGO) is that the *turtle* is coordinate free, in that *going* and *turning* are completely relative to the *turtle's* current state. The *turtle* thus lives in *curveworld*. A straight line has 0-curvature, a circle has constant-curvature, linearly changing curvature generates beautiful smooth curves. This may be contrasted with the cartesian world in which 0 generates 0, a ramp generates a line, and a quadratic is required for circles and simple curves.



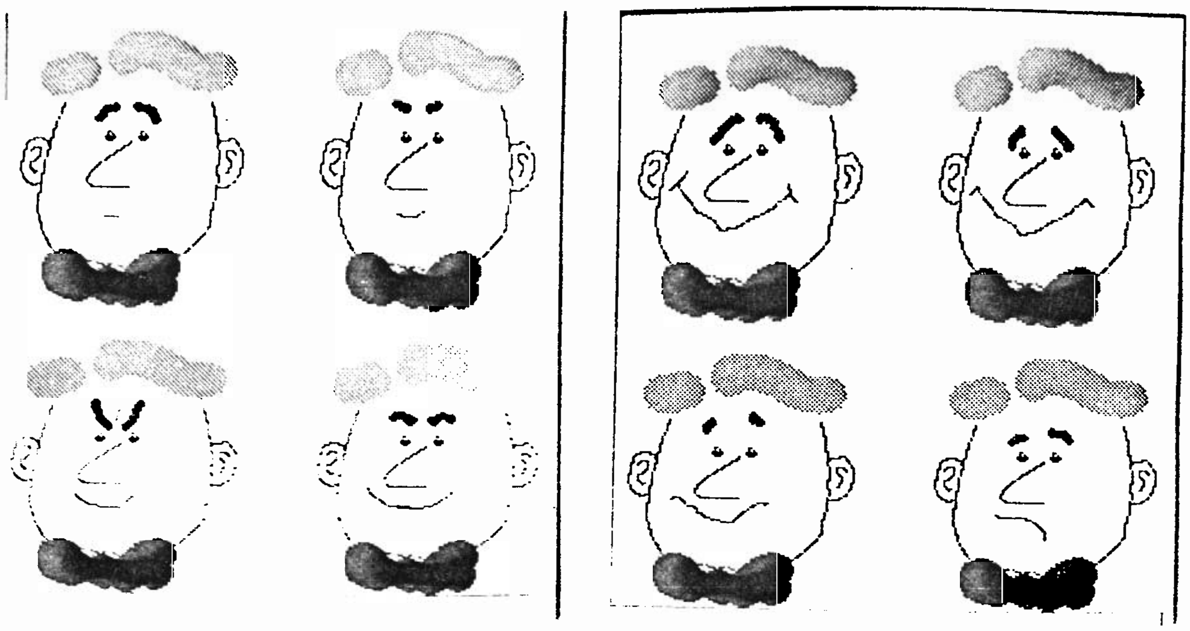


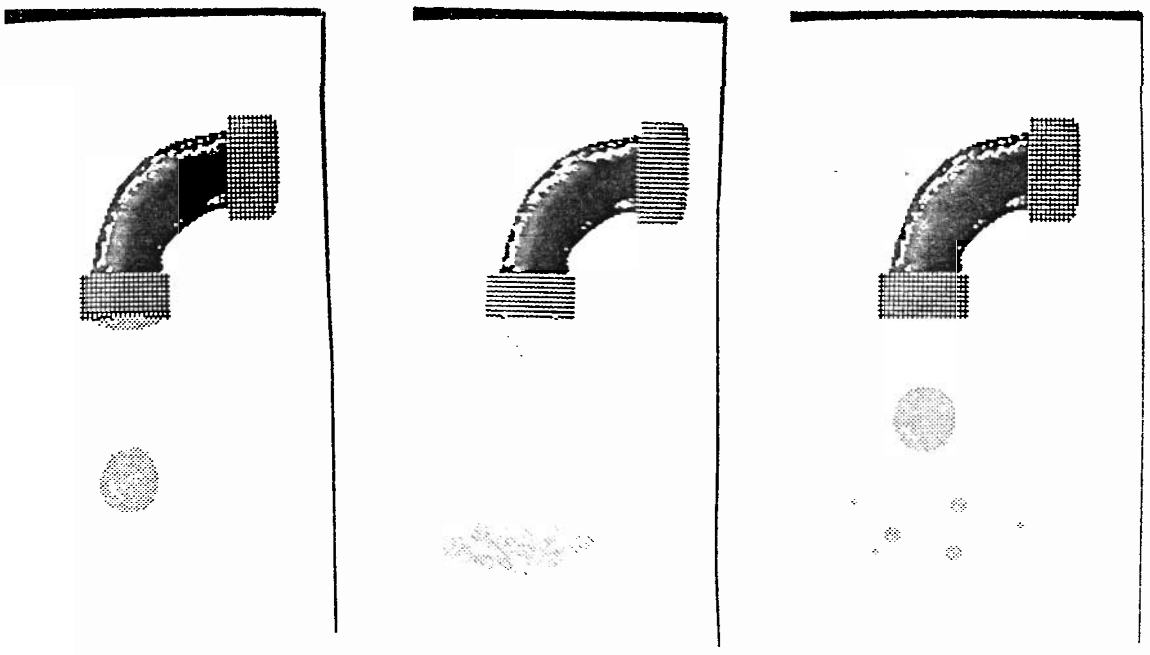
Animation and Music

Animation, music, and programming can be thought of as different *sensory views* of dynamic processes. Their common structural form is apparent in Smalltalk, which provides a neutral framework for expressing these ideas. All of the systems are equally controllable by 'hand' or by program; drawing/painting are directed by the mouse or with curve and area programs; musical events are initiated with a piano-like keyboard or from a dynamic 'score'. Timbres are the 'paint' of musical expression as they contain the quality and mood which different instruments bring to an orchestration.

A Picture Animation System Programmed by Animators

Several professional animators visited us with a long-held dream for a 'magic-slate' which would allow them to create high-quality animations by simple (and literally) "waving their hands". They wanted to paint pictures *into an already running animation* in order to take maximum advantage of the 'phi' effect (which causes the main action of animation to be the *change* between frames).



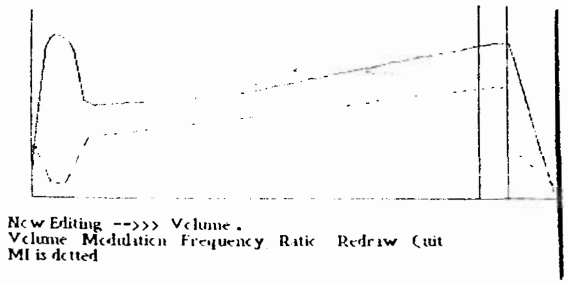


An Audio Animation System Programmed by Musicians

Animation can be considered to be the coordinated parallel control through time of visual images. Likewise, a system for representing and controlling musical images can be imagined which has very strong analogies to the visual world. Music is the design and control of tone images (pitch and duration changes) which can be *painted* different *colors* (timbre changes); it has synchronization and coordination, and a very close relationship between audio and spatial visualization.

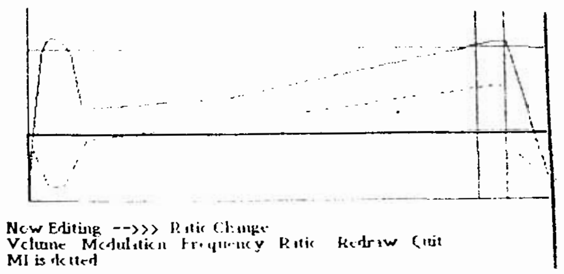
Timbre Synthesis

We use several methods for real-time production of high-quality timbres; both allow arbitrary transients, many independent parallel voices, and are completely produced by programs: no special hardware is used. The most interesting of these was developed [19,20] and allows independent dynamic control of the spectrum, the frequency, the amplitude, and the particular collection of partials which will be heard.


Now Editing -->>> Volume.

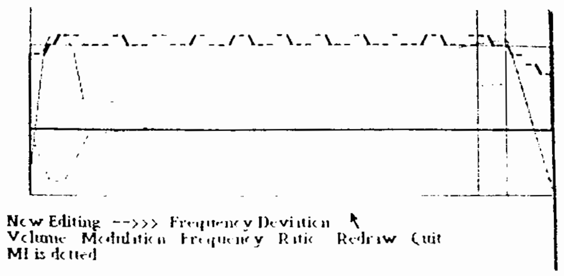
MI is dotted

An instance of a timbre class showing the amplitude change and spectral change (dotted lines) over several seconds of time. The area between the vertical lines will be repeated until the note ceases.


Now Editing -->>> Ratio Change

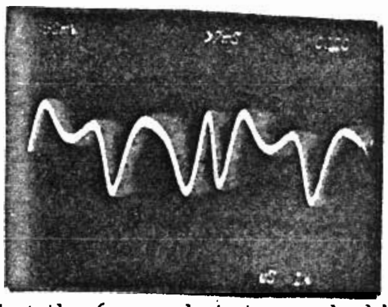
MI is dotted

Now the ratio of modulation is changed to emphasize only the odd harmonics: producing a 'clarinet'

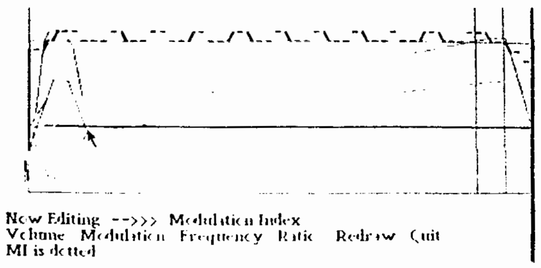

Now Editing -->>> Frequency Deviation

MI is dotted

Now a frequency deviation is drawn to produce an initial portamento followed by a sharp vibrato.

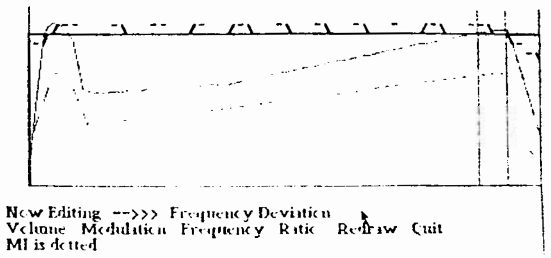


Here is what the (somewhat strange looking but good sounding) wave looks like on a 'scope.'


Now Editing -->>> Modulation Index

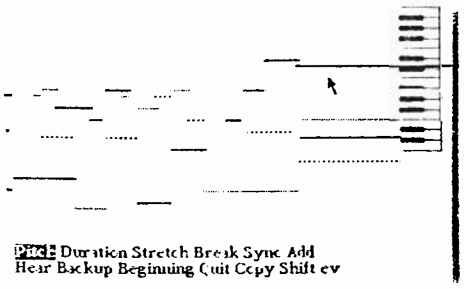
MI is dotted

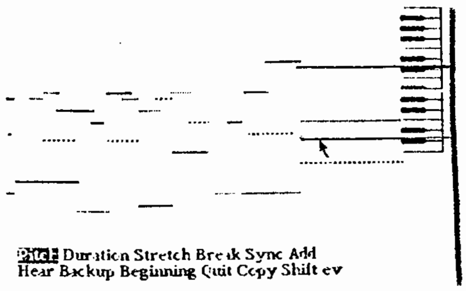
We reach in and grab the spectrum control. We want it to 'bump' in the beginning, and then gradually increase.

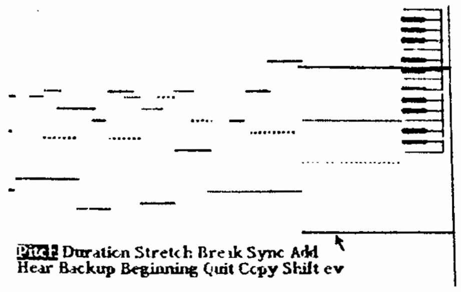

Now Editing -->>> Frequency Deviation

MI is dotted

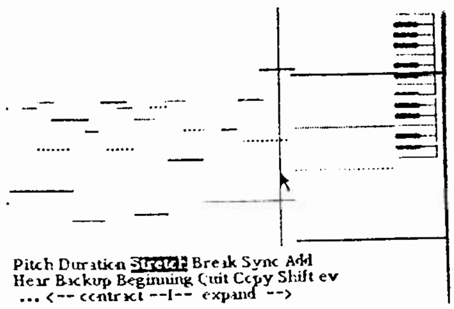
Here, the ratio of modulation is changed to emphasize all harmonic partials: producing a 'trumpet'.

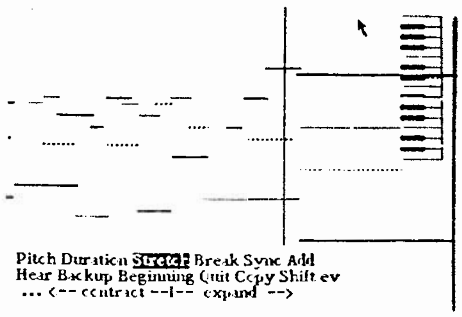

Pitch Duration Stretch Break Sync Add
Hear Backup Beginning Quit Copy Shift ev

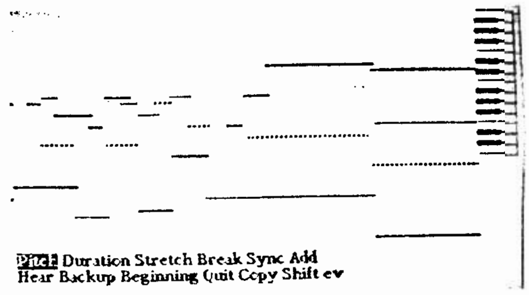

Pitch Duration Stretch Break Sync Add
Hear Backup Beginning Quit Copy Shift ev


Pitch Duration Stretch Break Sync Add
Hear Backup Beginning Quit Copy Shift ev

Move it to 'd'. There are two 'thirds' in chord. Double the octave


Pitch Duration Stretch Break Sync Add
Hear Backup Beginning Quit Copy Shift ev
... <-- contract --|-- expand -->


Pitch Duration Stretch Break Sync Add
Hear Backup Beginning Quit Copy Shift ev
... <-- contract --|-- expand -->


Pitch Duration Stretch Break Sync Add
Hear Backup Beginning Quit Copy Shift ev

'Slice' the score, and... stretch it to there, so ... we get ritard.

These systems have a number of benefits for both children and adults:

The semantics are easy to understand since they intentionally are anthropomorphisms from the real world. The strong similarities between the audio and visual worlds, and between the arts and the sciences, are emphasized because a single vernacular *which actually works* in both worlds is used for description.

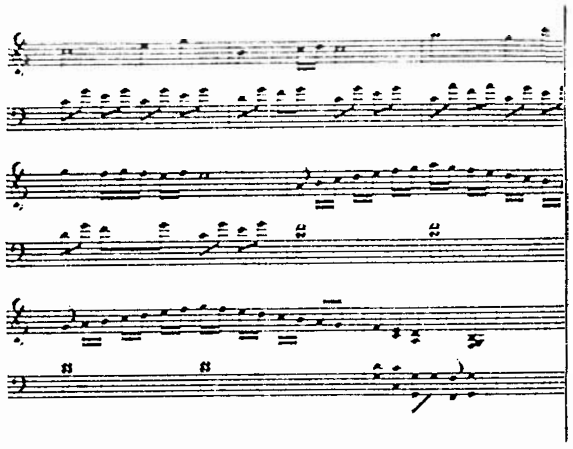
Children can gain skill and coordination by learning how to draw and play. The systems will show and replay for them what they just tried, then allow them to compare their efforts to a more expert model, much in the manner of tennis or skiing instruction.

The arts and skills of composing images changing in time can be learned in parallel with the acquisition of muscular expertise since much of the 'dirty work' is automated. This is not a prosthetic but an *amplifier*.

Score Capture and Editing 1: Dynamic Capture and Score Production.

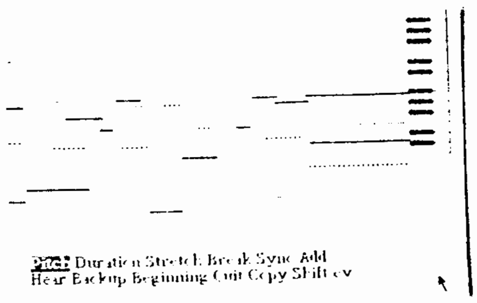
This is a musical score capture system written by a novice programmer who is an experienced musician. The system produces a display of a conventional musical score *directly from data obtained by playing the music keyboard in real-time.*

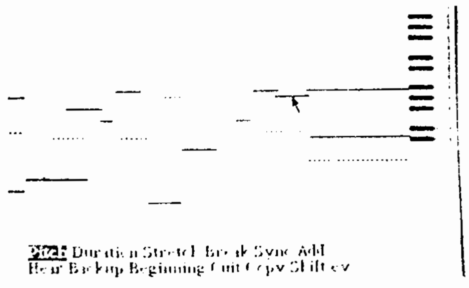
Playing it in.

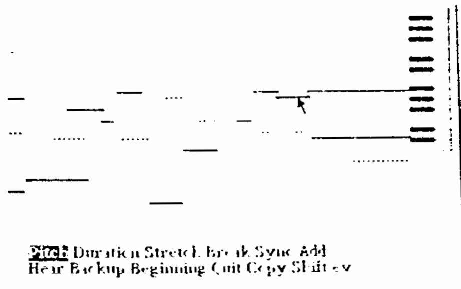

The Captured Score

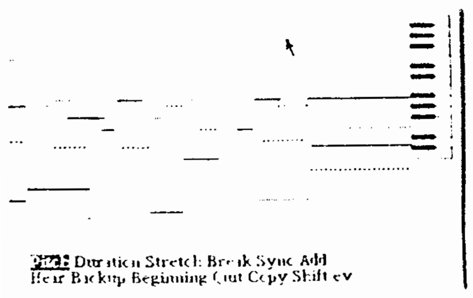
Score Capture and Editing 2: A Comprehensive Music Editor

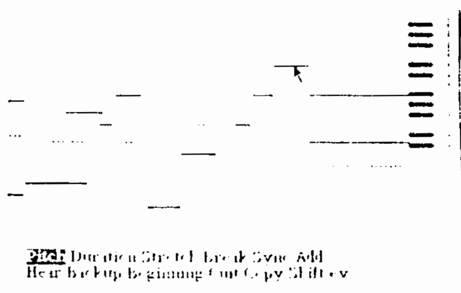
Standard musical notation has a number of drawbacks: a large number of symbols must be remembered, articulation can't easily be shown (without cluttering with rests), precise duration control is very tedious to notate, the staff is biased towards certain keys, and it is hard for kids to see that similar sounding chords are really the same. Accordingly, most of our music editing is done in terms of a simpler notation which uses length to show duration.

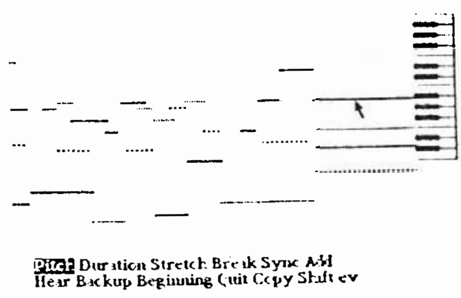

Pitch Duration Stretch Break Sync Add
Hear Backup Beginning Quit Copy Shift ev
A played-in score.


Pitch Duration Stretch Break Sync Add
Hear Backup Beginning Quit Copy Shift ev
We want to change a note.


Pitch Duration Stretch Break Sync Add
Hear Backup Beginning Quit Copy Shift ev
It's grabbed.


Pitch Duration Stretch Break Sync Add
Hear Backup Beginning Quit Copy Shift ev
Playing test notes with mouse


Pitch Duration Stretch Break Sync Add
Hear Backup Beginning Quit Copy Shift ev
This is the one!


Pitch Duration Stretch Break Sync Add
Hear Backup Beginning Quit Copy Shift ev
Grab another!

It will be very difficult, and also rather unesthetic, to develop a *prosthetic* which attempts to understand explanations and descriptions given by people who do not really understand the things they are trying to describe.

It seems much more beautiful to show that people can be amazingly more effective at dealing with their world if they learn some powerful techniques and skills (perhaps with the aid of media) for visualizing, part-and-whole-ing, planning, symbolizing, manipulating, and avoiding debugging.

Some **Principles** We Use **in Teaching**

These ideas are found in many places and many cultures. We came to them from our own experiences, the Suzuki violin method, O. K. Moore, Piaget, Furth, Bruner, Minsky, Papert, and others.

1. Listen to the student.

Since we believe that teaching involves helping a student adapt his knowledge structures to a new situation, we can guarantee ourselves (not to mention the student) an unpleasant journey if we don't try to understand these gossamer schema at the outset.

> Many of the current ways that things are done in Smalltalk come directly from listening to the kids. Smalltalk, as an 'extensible' system, can easily 'be' any kind of tool that we wish. We ourselves have remolded it several times.

2. Never teach anything which has to be unlearned later.

In our experience, humans are very poor at unlearning any kind of skill, whether it be muscular or mental. This principle is well understood by every teacher of music. 'Tempting analogies' which later come back to haunt are especially to be avoided.

> We teach 'straight' Smalltalk, the very same system which adults learn. The very first examples and methods to which the kids are exposed resemble strongly the most sophisticated adult systems.

3. Never pace a student in a way that will require future remediation.

Principle 2. basically says: don't simplify to the point of a lie; Principle 3. is a corollary of this which states: don't put the student into a situation where he will feel dumb and inept because a good enough foundation has not yet been laid. Most kids do not understand the distinctions between skill, structure, and intelligence any better than adults do and are apt to feel *stupid* rather than *unskilled* in new situations.

4. Hook on to existing fruitful structures when possible; if unfruitful concepts exist, don't unteach them, rather supply completely fresh orthogonal concepts.

Most kids know about dictionaries and looking up the meaning of a word. The meaning can be an explanation of a passive relationship or a dynamic act. In fact, every idea in mathematics and in programming can be easily explained in dictionary oriented terms alone; this is a fruitful, useful concept, and it makes sense to use it with kids.

Many other 'natural language' linguistic structures are ultimately deadly and we avoid them. Examples are: 'nouns', 'verbs', 'pronouns', inflections, and their counterparts in most programming languages: data structures, functions and control structures, variables, tagging type to names, etc. Instead, we immediately give children a running example which directly exhibits the more fruitful notions of *states-in-process communicating-with-messages* found in Smalltalk.

5. Do not look over the student's shoulder.

Aside from the obvious reason of avoiding 'putting the student under the gun', there is also a great difference between *performing* and *creating*. In music, this is known as the difference between *improvising* and *composing* (and a greater difference could hardly be found, as any musician will attest). We are much more interested in the design-oriented and planning processes associated with unhurried goal-directed reflection than in the more shallow though flashy effects obtained by virtuoso 'thinking on one's feet'.

We capture every action which a student makes and can replay their session for our eyes later. We tell every student that we do this, but the process of capture is completely invisible and thus rapidly

Our Experience With Naive Users

Smalltalk users have included children, ages 6 -15, and adults, both computer sophisticates and noncomputer specialists. Although we have not followed enough children long enough to determine whether their experience with Smalltalk has qualitatively changed their ways of viewing and thinking about the world, we have reached a number of subjective conclusions from our experience.

First and foremost, the ability to program is an inherent legacy of humankind. Everyone can do it.

Using Radia Perlman's *Button Box*[14] we have seen children of three and four years old consciously plan and 'write' symbolic procedures for drawing pictures. Eight year olds do simulations of moon landings, ten year olds do their own SPACEWAR, twelve year olds design and implement entire systems for illustration, choreography, and simulation. One of our 15 year olds has recently finished an interactive flight simulator of fair complexity (it uses an aerodynamic model for moving from one state to the next). Secretaries have successfully planned and added new 'features' to editing and retrieval systems, artists, who have tried and failed in the past (with BASIC, etc.), have been able to translate some of their yearnings into running simulations of their ideas.

It is only remarkable that some people find this remarkable.

Programming is the act of communicating descriptions of processes symbolically, the very same act as with 'natural' language. We should be able to guess confidently that most of the problems encountered will be those of a linguistic nature (e.g., as with learning French) rather than being due to intellectual problems on the part of the learner.

Second, we have seen a clear separation between the skills associated with programming and debugging, and those having to do with design.

An analogy: Smalltalk can be likened to an extensible Tinker Toy (Thinker Toy?) set. It contains a few simple ideas, some already built struts and connectors, and some moldable plastic for making unanticipated building blocks; *it has been* obvious *to all of our users as to how these may be used to build simple desired structures.* Nor have they had any trouble with the mechanics of molding new structural elements (this is easy and automatic in Smalltalk).

We can imagine children enjoying themselves immensely with a Tinker Toy set (as indeed they do), and, in the process, they learn valuable, more general ideas about how things connect together and can be manipulated in the 3-D world.

Children appear to have the same kind of experience in the 'Thought-D' world provided by Smalltalk.

An adult structural designer may use his Tinker Toy set to test out a new idea for an arched span for a new kind of bridge. The designer, after a few trials, will successfully span a large area with his Tinker Toy model, and there will be no lack of motivation for extending the Tinker Toy set in the process.

Computer systems designers have found Smalltalk to be a exceedingly easy to use, friendly way to bring their (sometimes very complex) ideas to life.

Suppose now that we bring an adult to the Tinker Toy set. After successfully making a few 'kid-type' models the adult decides to make a huge bridge. A massive failure results, and we can confidently assign the reason not to any lack of 'intelligence' on the part of the adult but rather to an innocence of design knowledge having to do with large structures. After all, it took a collection of the 'smartest' designers and builders more than 5000 years to invent the catenary arch and buttress --- ideas any child can read about and understand today.

'Nonprogramming' adults have no trouble programming in Smalltalk. They do run into difficulty when they attempt projects whose structure they really don't understand in any terms.

This parable illustrates one of the leading red-herrings concerning the cult of intellignce in current pedagogical thought: an incredible confusion between performance, skill, and 'innate ability'.

Our method for teaching children and adults about programming, coupled with Smalltalk's simple and very neutral basis for describing dynamic situations, has eliminated many of the usual 'linguistic' problems having to do with vocabulary, place-holding, abstraction, grammar, etc., and has allowed us to get closer to the 'real problems' which concern themselves with the kind of visualizations, structures and skills the fledgling designer brings to this new area.

All of this suggests that the primary goal of a teacher (after learning to listen) *is to find ways to help learners develop* in their heads *a good set of methods centered about the understanding of existing designs and the creation of new ones.*

ignored.

6. Teach and Show Multiple Perspectives of Situations.

A typical problem with fledgling designers of all ages is a strong tendency to commit all of their short term memory to a given perspective of a situation. If it happens to be an unfruitful view it may be very difficult for them to 'bail out' or even tell that it is unproductive. We feel that the Piagetian example of the tilted glass is much more the result of lack of practice in multiple viewing than the result of physiological immaturity.

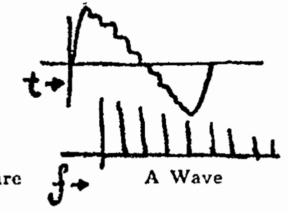
One of the striking things about design methodology is that 'simultaneous' use of a perspective *and its dual* is remarkably more rewarding than using either separately.

A very global example is the duality of wholes-as-collections-of-parts *found in Western science and* wholes-as-wholes *found in Eastern philosophic thought.*

The former has an important dualistic aspect itself: *analytic* (or top-down) vs. *synthetic* (or bottom-up); both of these emphasize *differences* and *boundaries*: a *corpuscular* theory. The Eastern philosophy emphasizes *samenesses* and *connection*: a *field* theory. As more complex systems are studied, the apparent differences between the two schools of thought blur in the underlying sameness that characterizes duals.

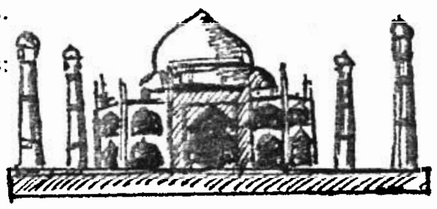
The human nervous system produces reactions in both directions:

A Square



A Wave

Figure/Ground


A Vision

A 'linear' theory, useful for simple models such as walls made from bricks, falls down badly in more interesting domains. A typical reaction of those whose prime methodology is centered about the linear model is to attempt to 'patch' (or add *epicycles* to) the description rather than to recenter their inner vision.

A linear theory of the Ta Maha is that the bricks were brought to the clearing and added together until the building appeared! It is very hard to see how the design process for the Taj Mahal (or of a human embryo) can be fruitfully characterized as a collection of patches on simple assembly notions.

Relativistic philosophy, on the other hand, is much more a grand combination of the two points of view: Every thing *is* every other thing because they are just local geometric states of the 'same' space, and conversely, there is a both a field lag and an attenuation associated with relative distances and speeds which makes the concepts of *objects*, and *parts* reasonable to consider.

We try, partly in the spirit of principle 2, to show children about both aspects of design thought right from the very beginning of their experience with Smalltalk.

Humans and Media

'Devices' which variously store, retrieve, or manipulate knowledge in the form of *messages* embedded in a *medium* have been in existence for thousands of years. People use them to communicate ideas and esthetic feelings both to others *and back to themselves.* Although thinking goes on in one's head, external media serve to materialize thoughts and, through feedback, augment the actual paths which thinking follows. Methods discovered in one medium frequently provide powerful metaphors which contribute new perspectives for notions in other media.

Every message is a *simulation* of some idea. It may be representational or abstract, isolated or in context, static or dynamic. The particular *essence* of a medium is very dependent on the way messages are embedded, changed, and viewed.

Computers were originally designed to do arithmetic computation; a powerful idea was the notion of controlling the computation from a *description* of the algorithm held in a writable store. Even more powerful is this corollary:

> *The* content *of the computer is* descriptions of processes; *the ability of computers to simulate the details of any descriptive model means that the computer, viewed as a medium itself, can be* all other media *if the embedding and viewing methods are sufficiently well provided.*

This 'pocket universe' (a metaphor we like) needs an epistemology if not a metaphysics.

Some Observations Which Led To Smalltalk

The basic principle of recursive design is: make the parts *have the same power and capabilities as the* whole.

The 'whole' is a digital computer, a *black-box* to which we *send messages* and *receive replies*, which contains *state-in-process.*

From the outside, we don't know very much about the methods which the black box uses to send back replies. When we request:

sine 30

we don't know whether the reply is computed by table-lookup, Chebyshev approximation, a summed series, or a combination of these. And, in fact, we don't really care, as long as the expected reply comes back *consistantly, quickly, and without interfering with other things we may also be doing.*

The principle of separating *desire* (a semantic notion) from *method* (a pragmatic notion) is central to Smalltalk.

It is interesting to note that none of the parts *of most programming languages, 'data structures', 'functions', and 'control structures' have the same power as the* whole; *instead they are* dilutions *of the idea of a computer.*

The Five Simple Ideas of Smalltalk

1. There are only objects.

The numbers *3, 4.5, 1.245e14* are each objects; so are the words: *this, identifier, sine, file34*; so is the collection: *(this is a collection of words)*; and the literal piece of text: *'this is a piece of text'*.

2. Each object has memory.

In fact, each object is in charge of its knowledge, how it is represented, and how it may be used. Each object has some way to distinguish itself from other objects. For instance, the object representing the number *3* might have in its memory the magnitude '3' stored in some fashion; the object representing the number *4.5* could use the same or a different technique to remember the magnitude '4.5'.

3. Objects communicate with each other by sending and receiving messages.

The first three ideas constitute a recursion on the notion of a computer. Idea (3.) actually includes idea (2.) but both have been included for clarity.

In Smalltalk, a message is sent to an object by first stating the object, then the message.

3 sign

sends to the object '3', a message consisting of the word 'sign'. The response to a message is entirely up to the receiver. We might hope that a '+' will be sent back consistantly from each number of positive magnitude.

A 'powerful idea' is the notion of grouping objects which have similar properties into *classes* so that they may be discussed in general. In Smalltalk we currently find the additional two ideas:

4. Every object belongs to a class.

A class is thus an object (from 1.); there also must be a class: *class* (from 4.). 'Obvious' classes include *numbers, words, collections, files, text.* More exotic classes include *font characters, pictures, pens, sets, paragraphs, windows, documents, timbres, voices, choruses,* and of course, *class class,* whose members contain the definitions of *numbers, words,* and so on.

Part of the memory of each object in Smalltalk is the knowledge of its class membership.

5. A class is the collection of receivers *for legal messages to objects in the class, coupled with* methods *for producing a reply.*

There are many ways to accomplish idea 5. We have tried several[3,8,9]. Our current approach has a minimum of mystery and requires a minimum of faith in order to understand and use classes.

To define a class description, we use the notion of a dictionary containing entries, each one of which is a *detector* and *replier* for a particular message. A class description is thus just one of the many kinds of documents handled by the information system described previously.

An entry in the class description of number might be:

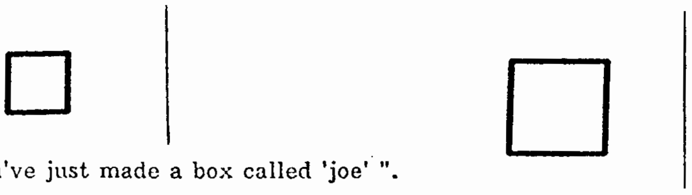
```
        :
        :
    👁sign ⇒ (Self < 0 ⇒ (⇑ '-') ⇑ '+')
        :
        :
```

which means,

If you see	the literal word	then	if the current member of number	is less than zero	then	return to the sender	a literal plus	otherwise return to the sender	a literal minus	
👁	*sign*	⇒ (	*Self*	< 0	) ⇒	(⇑	'+'	) ⇑	'-'	)

Children and Smalltalk

Our basic approach to teaching Smalltalk to both adults and children is to show them a simple, running example of a Smalltalk class and the objects which belong to it. After they 'play' with it for a while, we get them to 'guess' the class definition, and finally they add a new feature to the description. In the process of *editing* an already running program, the new users learn quite a few things about the *form* of a description which we would otherwise have to explain in a mysterious manner.



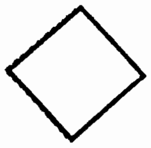
First we get them to type:

☞joe ← box

A square appears on the screen. We say: "You've just made a box called 'joe' ".
Now they try:

joe grow 30

and the box 'joe' grows accordingly. We say: "You've just sent a message to 'joe' that it understands, and it responded accordingly".



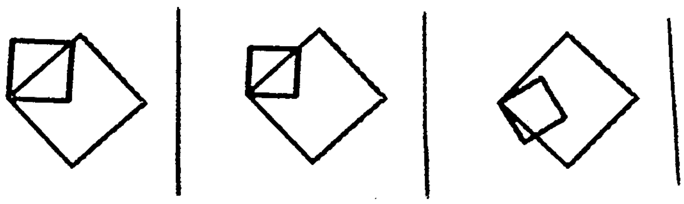
joe turn 45

This is another message that 'joe' understands.

eachtime (joe turn 25)

Now the student has made a simple 'movie' which can be terminated via the 'escape' key.

☞jill ← box



Another square appears on the screen.

jill grow -10

'jill' becomes smaller.

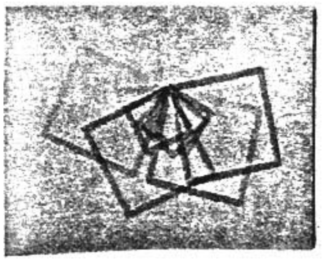
jill turn 50

'jill' turns just as 'joe' did. Evidently 'jill' and 'joe' can receive similar messages and produce similar responses.

jill is ?
box

'jill' sends back the answer 'box' when asked what kind of object she represents.

joe is ?
box



So does 'joe'.

eachtime : joe turn 25. jill turn - 11.

produces a two object animation with a limited plot.

Now we get the children to 'guess' what the general description of the class 'box' might look like. They have seen five example transactions: *make-a-new-one, grow, turn, draw, undraw*, each invokes a different set of actions. Since we have been using the word 'message' it is not difficult for the children to 'guess' that there will be at least five sections to 'box', each dealing with a particular action. The idea of 'looking' at the message to see what's there is introduced. Finally, we ask them to type:

show box

to see what the description of 'box' really looks like:

```
☞box ← class : size tilt positionleft positiondown   properties of a box
     (
      isnew ⇒ (☞positionleft ← 300.                  initial position
               ☞positiondown ← 100.                  initial position
               ☞tilt ← 0.                            initial heading
               ☞size ← 50.                           initial size
                SELF draw.                           cause new object to appear on screen
                ⇑ SELF)                              send back our new object to sender

       👁draw ⇒ (☺ penup                                      tell ☺ to pick the pen up (no marks)
                  goto positionleft positiondown             travel to the new position
                  turn tilt                                  turn to the heading for this box
                  pendn.                             put the pen down (to make marks)
                   do 4 (☺ go size turn 90))         draw a square in new orientation

       👁undraw ⇒ (☺'s ink ← white.                     turn turtle's ink to white
                  SELF draw.                         draw with white to erase shape
                   ☺'s ink ← black.                  reset ink back to black

       👁grow ⇒  (SELF undraw.                        erase ourselves at old position
                   ☞size ← size + ▯.                 increase size by value from message
                   SELF draw)                        redraw ourselves with new size

       👁turn ⇒ (SELF undraw.                         erase ourselves at old position
            ☞tilt ← tilt + ⌷.                increase tilt by value from message
            SELF draw)                       redraw ourselves with new tilt
    )
```

shows the various messages that can be sent to any member of the class *box*.

The various iconic symbols were made up by the first group of children that we taught to program in order to clarify important ideas in their minds.

☞ The kids prefered to use ☞ to indicate a literal symbol since in its typical use:

☞joe

(meaning the literal symbol 'joe' rather than what (or who) 'joe' may stand for), the hand points directly at the symbol itself.

We might guess that:

☞joe ← box

means that we are sending the literal symbol 'joe' the message: ← box. What happens is that 'joe' looks up his meaning and changes it to the reply sent back by 'box' (which, of course, is a new instance of 'box').

: means that the following symbols will be local properties of each created object. In the 'class' human, we might find the property 'eye-color'; each instance of human would have a local value (say) 'blue' or 'brown'.

isnew is a test to see if a new instance is to be created. This a much simpler and less mysterious way to do initialization than with an infinite loop and 'pause'.

⇒ will skip the next object if seen by 'false'. i.e. 3<4 ⇒ ('yesyes') will have value 'yesyes'.

SELF since there are typically many instances, SELF stands for the one currently receiving the message.

⇑ replies back to the original 'sender' of the message.

👁 'looks' in the message to 'see' if a literal word is there. Its reply is true or false allowing it to be tested with '⇒'.

☺ a symbol which stands for one of the many available Smalltalk turtles.

⌷ this symbol means receive the value of the next set of things in the message.

Instead of trying to get the children to understand this definition in any crisp way, we tell them a little bit about *isnew*, ⇒, *SELF*, ⇑, 👁, ☺, ⌷ and almost nothing about ☞ and ':'. Then we ask them to see if they can imagine how they could get any box to *move* to some place on the screen.

They use *grow* and *turn* as models. Their first articulation of the new method is usually very much like:

```
    👁move ⇒ (SELF undraw.
              ...something to change the position....
                   SELF draw)
```

They say:

If ourself sees the word 'move' in the message, then we get ourself to undraw (because we are going somewhere else), we do something to change our knowledge of where we are, then we redraw ourself (just like in 'grow' or 'move').

Now they look closer at the names at the top of the description, the first two of which have already been used for 'size' and 'tilt'. 'positionleft' and 'positiondown' look very hopeful (especially with some hints from us). Now they are ready to understand 'draw' and the turtle (☺). They see that the turtle picks its pen up, goes to 'positionleft' and 'positiondown', then puts its pen down and draws a square (they find out the latter by trying it themself).

It really looks as though they just need to change 'positionleft' and 'postitiondown' by receiving new numbers from the message *exactly analogous to the previous examples*. They write:

```
☜move ⇒ (SELF undraw.
          ☞positionleft ← ⦂.
          ☞positiondown ← ⦂.
          SELF draw)
```

Instead of getting them to learn about the editor at this point, we provide them with a simple class which does the update called: *addto*. So they actually type:

```
addto box (☜move ⇒ (SELF undraw.
          ☞positionleft ← ⦂.
          ☞positiondown ← ⦂.
          SELF draw))
```

This works very nicely when they try:

```
joe move 400 100
```

We take as long as necessary to get to this point with the children (usually 1 or 2 sessions). Although there is quite a strain on their short-term memories, the benefits are tremendous compared to more atomic bottom-up approaches. From here there are many paths that can be followed:

We ask them to try:

```
mouse left
        344
mouse down
        112
```

and their abbreviations:

```
ml
        344
md
        112
```

These are the mouse locations expressed as 'left' and 'down' offsets from the upper right hand corner of the screen. They try:

```
joe move ml md
```

and 'joe' goes where they point! Many children spontaneously 'see' that they can use 'eachtime' to do these operations continuously:

```
eachtime (joe move ml md)
```

and 'joe' follows! Even more interesting, many see that they can 'paint' if they just were to remove the 'SELF undraw.' line from their previous definition. They are now ready for the editor which handles the programs in a structured way and automatically makes sure that parenthetic levels are consistant. When this is done, they have made a controllable 'brush' which can take on various sizes and orientations (by doing 'grow' and 'turn'). Moreover, they know that they can create a large number of tailored brushes and give them names.

The power of classes and instances now starts to make itself felt.

Our children usually spend about 6 to 8 sessions exploring in this fruitful neighborhood. This is important because our experience indicates that learning to program progresses in little leaps with intervening 'plateaus', where, for a while, it is difficult for the students to 'see' new things. Then they do and another leap takes place. This reminds us very much of the typical progression involved in acquiring skill at playing music or sports. Since the plateau areas are very important, it is necessary to make sure that there are plenty of fun things to do while the inner 'coordination' is being built. Otherwise, it is possible for boredom to set in before enough skill is obtained to really make the new area intrinsically interesting.

The children typically will manipulate every part of the 'box' description. They change the 'shape' routine to draw triangles and other polygons and thus learn about 'turtle geometry' by the back door. They may make the 'shape maker' another class entirely and add another parameter, 'shape', to the box description.

After a while, rocket-ship and airplane shapes replace the polygons. New messages are added, like: *fly, bounce, open, close*, and so on. They discover that they only have to continuously add some small constant number to the position holders to get an animation with simple linear motion.

Everything, without exception, *in Smalltalk is represented as active instances of classes, and, the 'box' class is an 'archetype' of most of the already existing Smalltalk classes. Numbers, the turtle, dictionary structures, 'collection baskets', schedulers, all look very much like 'box'. When the children are ready to invent classes of their own, they have not only a bag of tricks but a fairly comprehensive methodology for representing their own ideas.*

We feel this is why they are able to handle many of the notions involved with systems design at such an early age and comparitively low level of sophistication.

Artificial Languages vs. Automatic Programming

This paper is about *Personal Computers for Personal Computing* by noncomputer professionals of all ages. We would characterize our approach to be that of providing very powerful means to express a user's desires and methods through an *artificial language* rather than to supply a system for *automatic programming*[22], although the distinction between the two areas is not as crisp as it might be.

Not too many years ago, FORTRAN and ALGOL, were hailed as 'automatic programming' systems because they automated many nasty assembly code tasks, such as subroutines, arithmetic evaluation, storage allocation, files, and input-output.

In this limited sense, Smalltalk is an 'automatic programming' language since it handles many more of the background needs of a user than 'higher-level' languages of the sixties. However, 'automatic programming' in the seventies has a different connotation which constrasts rather sharply with the 'artificial language' approach to building running programs.

Automatic Programming vs.	*Artificial Language*
Sloppy (natural) discourse	Clear (rather crisp) discourse
'Intelligence': (quasi)-human model	Simplicity, Consistancy, Generality
Knowledge of Problem Domain	Knowledge about the nature of Description
Natural Language Semantics (Nouns, Verbs, ...)	Model-Descriptive Semantics (states-in-process, communication, ...)
Declarative	Some imperative sequences

We like the latter, not only because it works, but because it is a 'convivial tool' (in the sense of Illich): the innards of a system such as Smalltalk are more available and non-intimidating to the user for perusal and modification than that of a Model T Ford in the 20's.

We don't like natural language programming and description very much because we feel that most natural languages lack clarity and focus (they were after all invented to talk about farming and cows).

A remarkable exception to this is Jong.gwo-huah *or Mandarin Chinese, whose features and signs are the most 'orthogonally' organised of any human tongue. In the Indo-European sense of the words, Chinese has no inflections, nouns, verbs, or prepositions. There is a remarkably small basic vocabulary and a simple syntax of great expressive and metaphorical power.*

In a progression starting with Sanskrit, to Latin, to the Romance Languages; English is more like Chinese than any of them and becoming more so every century.

Conclusions

We feel that a promising future for human-human and human-machine communication will be the developement of a simple human-oriented artificial language with great expressive power. It is needed for the same reasons which brought forth mathematics (as an artificial language) in the 16-20 centuries: not only are there new ideas and ways of looking at them afoot in the world, but the old frameworks debilitate their expression and communication.

Smalltalk is nowhere near what is needed; it shows just enough of what the future might be like to motivate further designs.

Thanks

especially to Dan Ingalls who brought the ideas of Smalltalk to robust life, Adele Goldberg who zestfully managed the kids and a thousand other things, Chris Jeffers 'properties person' extraordinary (who alone knows where everything is), and to all the people in the Learning Research Group and elsewhere in PARC who worked hard on Smalltalk and the Interim Dynabook.

From the Learning Research Group

Permanent Staff

Adele Goldberg
Dan Ingalls
Chris Jeffers
Alan Kay
Diana Merry
John Shoch
Dick Shoup

Visitors

Ron Baecker
Dennis Burke (age 12)
Barbara Deutsch
Marian Goldeen (age 13)
Susan Hammett (age 12)
Bruce Horn (age 15)
Tom Horsley
Lisa Jack (age 12)
Ted Kaehler
Kathy Mansfield (age 12)
Eric Martin
Steve Purcell
Dave Robson
Steve Saunders
Bob Shur
Dave Smith
Bonnie Tenenbaum
Steve Weyer

From Other Groups at PARC

Patrick Baudelaire
David Boggs
Bill Bowman
Larry Clark
Jim Cucinitti
Peter Deutsch
Bill English
Bob Flegal
Ralph Kimball
Bob Metcalfe
Ed McCreight
Mike Overton
Alvy Ray Smith
Bob Sproull
Larry Tesler
Chuck Thacker
Truett Thach

References

[1] Kay, Alan, *FLEX: an extensible simulation language which can be directly executed by computer.* Computer Science Note, September, 1967, University of Utah, Salt Lake City.

[2] ---------, *FLEX, a flexible extensible language.* M.S. Thesis. Dept. Com. Sci., May 1968, University of Utah.

[3] ---------, *The Reactive Engine.* Ph.D. Thesis. Dept. Com. Sci., Sept. 1969, University of Utah.

[4] ---------, A personal computer for children of all ages. *Proceedings of the ACM National Conference.* August 1972, Boston.

[5] ---------, A dynamic medium for creative thought. *Proceeding of the National Council of Teachers of English Conf.* November 1972, Minneapolis.

[6] ---------, Goldberg, Adele (editors), and the Learning Research Group, *Dynamic Personal Media.* (in press June 1975), Xerox Palo Alto Research Center.

[7] Goldberg, Adele, *Smalltalk and kids --commentaries.* PARC-LRG-3, June 1974, Xerox Palo Alto Research Center.

[8] ---------, Classroom communication media, *ACM SIGCUE TOPICS in Instructional Computing,* Vol 1, Teacher Education, Jan. 1975, (with Bonnie Tenenbaum).

[9] Feurzeig, W., et. al., *Programming-Languages as a conceptual framework for teaching mathematics.* Final report on BBN Logo Project, June 30, 1971.

[10] Papert, S., Teaching children thinking, *IFIP Conference on Computer Education,* 1970, Amsterdam: North Holland.

[11] Papert, S., Teach children to be mathematicians versus teaching about mathematics, *Inter. Jour. Math. Educ. Sci. Tech.,* Vol 3, 249-262, 1972.

[12] Sutherland, Ivan, *SKETCHPAD: a man-machine graphical communication system,* MIT Lincoln Laboratory TR 296, May 1965.

[13] Dahl, O-J, et.al., SIMULA, an algol based simulation language, *CACM, IX, 9, Sept. 1966.*

[14] ---------, *SIMULA--common base language,* Norwegian Computing Center, Oslo, Norway, 1970.

[15] Shaw, C., JOSS a designer's view of an experimental on-line computing system, *AFIPS Conference Proceedings, XXVI, 1, Fall, 1964.*

[16] Barton, R. S., A new approach to the functional design of a digital computer. Proc. WJCC, 1961.

[17] Fisher, D. A., Control structures for programming languages, Ph.D. Thesis, Carnegie-Mellon University, Pittsburg, 1970.

[18] McCarthy, J., et. al., *LISP 1.5 Programmer's Manual,* Cambridge: MIT Press, 1962.

[19] Chowning, J., The synthesis of comples audio spectra by means of frequency modulation. *J. Audio Eng. Soc.* 21, pp 526-534, 1973

[20] Saunders, S., Improved FM audio synthesis methods for real-time digital music generation. *ACM Comp. Sci. Conference,* 1975, Washington, D.C.

[21] Balzer, R., *Automatic Programming.* USC/ISI, RR-73-1