TRANSCRIPTION NOTES

Source. Transcribed from the 'hin68.tif' file, after fixing mangled X/Y screen resolutions in the original fax scans.The original TIF document is available from http://xanadu.com/REF%20XUarchive%20SET%2003.11.06/hin68.tif or http://web.archive.org/web/20190820162039/http://xanadu.com/REF%20XUarchive%20SET%2003.11.06/hin68.tif

Spelling & abbreviations. I have kept a closely to the author's manuscript as possible, retaining AmE spelling forms (I am using a BrE system). Author's abbreviations such a 'Hthext' and `shd` have been left in place as the expanded meaning should be self-evident ('Hypertext', 'should').

Problem text. In some places the words are indecipherable and cannot be inferred from the run of text. In a few cases the end of le line was cut off in the scan and may be guessed. Harder are a few pages where whole lines are missing at the bottom of the page. Such passages are [highlighted] within square braces. At Ted's specific request these markers are left in pending possible correction at a later date.

Pagination. I have retained the source pagination of the source material, such that one page here refers to one page in the original notes seen in the TIFF file. The source page number is given at the top of each page here below. This document is designed to be read alongside the scans of the original text, which contain the images.

Images & image captions. Due to the quality of the scans I have not attempted to copy across the images. I have indicated where images occur in the run of text. Image numbering is per-par and left/right top/bottom in sequence). Where pertinent I have noted image caption or annotation text. Where there are multiple small captions a ' // ' sequence implies a break between captions.

Transcription. This work was undertaken, with the author's approval, by Mark Anderson (mwra @ mac.com) during August 2019.

[PAGE 1]

WHOEVER GETS A XEROX OF THIS, PLEASE SEND ME A POSTCARD SO INFORMING ME.
T. Nelson, 458 W. 20, New York, N.Y. 10011.
PLEASE DO NOT XEROX AFTER 1968. Write instead to me for most recent intelligence.
[rule]

HIN

Hypertext Implementation Notes. Theodor H. Nelson
6-10 March 1968

Not for publication. Very informal. Cite as "Personal communication", if at all.

This is a rough (examination?) Of the problems that have been on my mind in implementing hypertexts (problems, not applications). The effort here has been to be comprehensive rather than comprehensible. Many things may be unclear. Others have not been specified usefully. Other things have been omitted, presumably.

These notes are attempted to clarify:

1) Basic examples of hypertexts, in greater detail than elsewhere—types and mechanisms.
2) The attempted generalities that have kept coming up causing some confusion.

These things are all being presented candidly here, in the hope of getting across exactly what has been on my mind so that the appropriate implementation details can be handled by then as may henceforward understand them best.

This has been written cold turkey (without notes or revision) so some sections modify earlier ones. Cross-referencing ameliorates this. But these documents, if understood whole, will pass on the burden of seeking overt structures. Perhaps there are none useful.

This is a jigsaw puzzle. Unfortunately, whether or not it makes an overall picture–that is, any unified structure–is unknown.

[PAGE 2]

CONTENTS

This is in no order at all, except that “Types of Hypertext”, p3, might serve as some sort of orientation. The latter three sections are independent.

The only thing which approach is a decent level of specification is “1-DIMENSIONAL CONTINUOUS HYPERTEXT” p14. But the different notes cast shadows on one another and nothing could be implemented except by someone who understood all this [cut off page]

[PAGE 3]

TYPES OF HYPERTEXT

The following types of hypertext (categorized structurally and not editorially) are known to me.

DISCRETE IRREGULAR HYPERTEXT

Individual text sections or chunks joined in a graph structure (one-way arcs or two-way chords)

[IMAGE #1]

A choice, usually visible, lets reader pick the next; though it can be a factual question or a default [UNREADABLE]reading of sections.

DISCRETE REGULAR HYPERTEXT where some repetitive structure is imposed.

[IMAGE #2] (caption left of image: “long sections”)

CONTINUOUS HYPERTEXT where some attribute(s) of the text may be changed by “continuous” degrees (very small increments).

> 1-DIMENSIONAL. The simplest example is Stretchtext, where the attribute that can be changed is length. But it could be be any other attribute, like Humor.
>
> N-DIMENSIONAL. Separate ‘throttles’ or whatever vary the text’s properties separately.

Consider also the following complex texts:

The PROUSTIAN MANUSCRIPT, with a) indexes, b) cross-reference jumping, c) alternative versions.

THE HYPER-MANUSCRIPT, same as above but with alternative hypertexts permissible.

[PAGE 4]

GRAPH DISPLAY

Virtually essential to hypertext construction and complex text editing is a screen display to show (and modify) graph structure. This includes:

[IMAGE #1]
(to right of image #1) PARTS GRAPHS, to show what there is in a given corpus and how it is interconnected.

[IMAGE #2]
(caption image #2) multicoupling

This is essential for Proustian editing and the hyper-library

Naturally the amounts and types of info displayed in graph at any one moment would have to be variable under user control. How graph layout would be selected (from equivalent graphs if modified is an open question.

[IMAGE #2]
(to right of image #2) This would also be an essential display for discrete hypertexts —particularly if the 'variable windows' idea of XANADU (see 'XANADU') is implemented.

FACILITIES GRAPH
[IMAGE #3]
(image #3 captions, clock wise from top) Text Editor // Fortran computer // Information Central // 0S/360 Control // Private Files

This is another XANADU idea. (see 'XANADU') Very important if the hypertext facility is to be linked up to other work, e.g., computer programming.

EVOLUTIONARY GRAPH (For 'Proustian Text Editing' and 'Hyper-Manuscripts', which see)
[IMAGE #4 - truncated at bottom]
(image #4 captions, left to right, top to bottom) First Notes // V1 // Further Nodes // V2

[PAGE 5]

The ELF (a previous unification)

In an earlier paper ('A File Structure for the Complex, the Changing and the Indeterminate') I described the file structure thought to be of general use. The idea was to store documents and text structures with linkages among their sections which would not change if we changed the sequence of a particular one.

[IMAGE #1]

This 1-to-1 relation among user-selected sections may be used to keep track of various changes among versions of a document and corresponding parts of different documents. This latter may be used for creating tables of contents. Thus it is a rather useful and important relation in this problem area.

The ELF (Evolutionary List File) was to be a file structure which incorporated this relation among documents filed, and maintained the relationship through changes. PRIDE was to be the larger language that permitted the changes, plus helping housekeeping.

Neither of these terms is useful right now. But we should consider the connector bundle among documents papers we will call it simply a "multicoupler". It links user-specified sections possibly assuming paragraphs to be the sections as a default assumption, subject to user correction.

Multicouplers may be transitive or nontransitive, hereditary through changes or [deterioraty?], depending on properties the user needs. A variety of types [text clipped by page end]

[PAGE 6]

XANADU (a previous unification)

The Xanadu system was a half-specified setup worked on at Harcourt, Brace & World. It had several interesting and useful aspects:

1) Screen formats for any kind of work were to be quickly reconfigurable. That is, the user could designate the size and shape of ‘windows’ into data, and their positions on the screen, thus creating changeable working formats.

[IMAGE #1]
(captions image #1, left to right) Data // Variable formatter // Screen formats

In much the same way , the use of both virtual pushbuttons on the screen, and quick-setup hardware pushbuttons, were to be reconfigurable.

[IMAGE #2]
(captions image #2, clockwise from top left) Interpretive command Structure // Softbuttons // Hardbuttons // Variable deformatter

2) Particular data structures—that is, relatively simple ones, like business forms and manuscript pages—were to be easily creatable and stored in a common data-base format. The general intent was to experiment with low-level text-editing and business-information systems.

3) Two types of graph structure were to be significant to system behavior/ One was to be a graph among text sections—i.e., a discrete hypertext.

[IMAGE #3]
(caption image #) E.g.

The user, setting up his ‘windows’ any way he wanted, could jump around it by arrows,

with each window looking into the hypertext at a different place, if he wanted.

The second graph structure was to be the set of activities in the user's "workspace".

[IMAGE #1]
(captions, image #1, clockwise from top right) Form 2 // Poetry-writing setup // Form 3 // JOBS TO DO TODAY // Wok on Form 1

Each of these task setups would offer options to switch into the connected task setups. Thus you could work it all day, doing everything on it, supposedly.

The two graph-structure system were supposed to share internal formats, and also to be displayable, as graphs on the screen. (see section 'Graph Displays'.)

[PAGE 8]

POIGNANT (a previous unification)

The incredible jungle of activities, scraps, pointers, and pieces of string to save in these various systems naturally pressed me to think of some fairly general way to handle it all. This was done from 1965 on, taking shape gradually (with the XANADU plan) in a file structure called POIGNANT (because it was mainly concerned with pointers).

Everything had to be capable of being indefinitely long if necessary. However, it was sort of mostly to be divided into big units made up of little ones.

A great variety of pointers , and acknowledgement between them (see 'Hyper-Manuscripts and Hyper-Libraries) had to be possible.

Thus it was decided to have everything in sections of standardized length, threaded into "trains" of possible lengths.

[IMAGE #1]

(Text to right of image #1) This wholly ignored the problem of fast lookup, which which meant nothing to me at the time. Not that I was unaware of it, but I had provisionally in mind corpuses of sizes where this would not be too painful.

It was also decided to separate three different types of item: 1) text, or better matter (like it could be numerical data); 2) pointers into that matter (Inpointers) and, 3) pointers elsewhere , including those connecting something inside with something outside. (Outpointers)

[IMAGE #2]
(captions for image #2, left to right) M // IP // OP

[IMAGE #3]
(Text left of image #3) This meant:

(Text right of image #3) each of the three types of information was to have its own 'train' within the overall file assigned to some particular thing.

[PAGE 9]

In other words, anything in POIGNANT was to be stored in complex ways that would always reduce (though sometimes vestigially) to

[IMAGE #1]

In recent months a few ideas have been added or clarified. One was what the hell, you could have trains for different purposes if convenient all in the same file. For instance, if you were accumulating a manuscript, one train of Matter would be the constituent items and change orders, just as they came in, and another train, obviously part of the same file, would be the updated thing itself, and maybe another train would be screen buffers. So one file could have a lot of different trains. (Pointers, too, might be sorted out into separate trains for [diff?]-purposes.

Other modifications of this idea have to do with diplomatic relations among files; especially ways that one file can know when it is pointed at so it won't mess things up by allowing itself to be changed without taking account of that pointer. Thus we are concerned with such things as change buffers, acknowledgement pointers, and [INDISTINCT] addressing within a file.

One other concession to feasibility was the idea of an executive record telling the location of successive records in a train. But a minor amendment to this was the idea that a if train got too long, the executive record would skip and only point to each nth —to keep the executive small & present.

However, this whole conceptual scheme is shelved, under the present realization that these details can be better worked out by others, once they have the whole picture.

[PAGE 10]

PLAIN DISCRETE HYPERTEXT

Basic screen layout:

1) JUMPING LAYOUT

[IMAGE #1]
(captions to right of image #1) Chink (& question, if any) // Choices or answers

2) THREADED DEFAULT LAYOUT SYSTEM

[IMAGE #2]
(Caption to left of image #2) Continuous movement under some control
(Caption to right of image #2) Jump markers (different little symbols)

The graph structure for either of the above may be alike:

[IMAGE #3]
(caption to right of image #3) etc.

However, in the case of the jumping layout you are brought to a stop till you make the next choice, while with the threaded system the text ‘look continuous’—because a Reader Parameter Vector, actually a second graph, determines the complete (or incomplete) set of default options. If you don’t choose a jump, you are automatically moved on to a next chunk; and this system of default chunks

may be varied from Reader to Reader. The different little jump markers of course need to be conventionally established by an anchor at the beginning of his piece.

For, of storage: Chunks & Graph Structure & Jump Info

(Numbered) Text chunks (say, 128 to 8192 characters)

Graph structure with numbers (representing text chunks) pointing to other numbers (representing other text chunks)

Jump Info. Where the significant jump markers go in the text and what they look like.

[IMAGE #1]
(captions to left of image #1) Note that // and
(captions to right of image #1) may be structurally the same: each is an arrangement of text with light-pen-sensitive (or mouse sensitive, etc.) jump markers.

HOW PROCESS. Obvious.

SCREEN RESPONSE.

You show a thing, then when a jump marker is hit you show another thing.

However, continuous (i.e. up/down 1/1024 increments) up-down movement is also necessary.

This is obvious in the threaded-default case; also needed in the jumping case where one chunk doesn't all fit on the screen.

[PAGE 12]

REPETITIVE DISCRETE HYPERTEXT
(This spec modifies Plain Discrete Htext)

Just like plain discrete hypertext, except you want to have structures that repeat. Polymerize.

[IMAGE #1]
(caption to image #1, left/right top/bottom) Chunk // Chunk // Chunk // Basic stream

This could be like: summary

[IMAGE #2]
(caption to image #2, left/right top/bottom) Exposition // Summary // Extra Clarification // Evidence // Difficulties // Future Research

It seems to me this simply calls for a slight extension of the graph structure system required for Plain discrete Htext. What must be allowed is the pointing, not just to individual chunks, but to Molecules & positions on them. Thus the above could be represented as:

[IMAGE #3]
(caption to image #3, left/right top/bottom) continuous text // A1 // A2 // A3 // B1 // B2 //B3

But further ‘molecular structure’ would have to be variable

[PAGE 13]

(since you can't expect authors to be consistent) and given positions a, b, c on molecules A must be allowed to interpoint irregularly:

[IMAGE #1]

Here we have another variant type of multicoupler (see 'ELF')

[PAGE 14]

1-DIMENSIONAL CONTINUOUS HYPERTEXT
(Especially: Stretchtext.)

Basic screen layout: screen, plus mouse, throttles and altimeter.

[IMAGE #1]
(caption at left of image #1) or
(caption at right of image #1) Preferably actual throttles, e.g. Lionel dual controller

Altimeter can be [IMAGE #2] just some sort of scale, or a two dimensional plot of where you are in the Stretchtext: [IMAGE #3]

Possible 2250 Screen Layout:

[Image #4]
(caption image #4, left to right) Altimeter // Stretch // Movement

(caption image #4, right top) Whether it should be vertical, horizontal or square needs to be determined empirically

(caption in image #4, right bottom) Controls on right (right-handed user). Zapping the arrowheads with the light-pen causes requested action for a little while; continuous zap gets continuous movement (May want some latching screenbutton for “keep moving” both on stretch and movement.

[IMAGE #5]
(caption image #5) Or: a [INDISTINCT] vector whose length and direction indicate desired stretch and movement. Move arrowhead back to location point to stop all motion.

Form of storage: Depends on strategy.

Strategy 1: (theoretical alternative) store as components, reassemble from surface to bottom. [IMAGE #6]

Strategy 2: Store as tree: finished sections which are then revised on basis of change orders.

[PAGE 15]

Under Strategy 2 the following data structure is required:

ALTITUDE M.N
(a two-part number)

CHARACTERISTIC Section M is a chunk of text (say 128 to 2048 characters)
MANTISSA Change orders 1 through N are applied to section M

Change orders are of 3 types:

INSERTION + text (1 to 256 char., say)

DELETION + 2 inter-character pointers [beginning and end of thing to be deleted]

SWITCHEROO + 3 inter-character pointers [beginning and end of sections to be switch[?ed]]

Note that each of thee can be treated and undone as the inverse operation—the deletion only if the deleted text is saved elsewhere, with an aftereffects points to where it came from. Or if text to be deleted is copied in full into the original order when carried out, etc.

GENERAL FILE STRUCTURE OF STRETCHTEXT
(OTHER 1-D Cont. Htexts less clear)

[IMAGES #1]
(captions to left of image #1, top to bottom) Altitude 1 // Altitude 2 // Altitude 3 // Altitude 4
(captions on image #1, top to bottom) Change Orders // Change Orders
(captions to right of image #1) Each file points to those above and below it.

HOW PROCESSED. Let's say Reader starts at A, throttles into B. Then he stretches. Change orders under B are then applied till he gets to A beneath it (call it BA)

[PAGE 16]

Reader keeps stretching. From now on system is referencing file BA directly, supplying change orders to that. Reader keeps stretching, reaches BAC and BACC (which files now become system references in turn). Now Reader goes forward. BACD is added to the buffer:

[IMAGE #1]

What the system does is a tree-retreat:

[IMAGE #2]

Now the Reader shrinks the text. The list of change orders is now undone, each being treated as its inverse operation.

Suppose the Reader is at altitude 3.5 and halfway between BACC and BACD, thus:

[IMAGE #3]

now he keeps going forward. After BACD he slides into BADA, etc.. Each time a record is passed through, the system retreats into the tree to find the next base text section.

SCREEN RESPONSE

If we had throttles, we could give each a ‘neutral’ position. As it is, the two controls must be incremental or semi-increme[tal?]. However, these movements on the screen should be very nearly continuous, Suppose we had

[text] Somewhat worried, he
[change order] INSERT / after char. 16 / about the soldiers /

The “he” should move slowly to the right (& thence to the next line) [TEXT CUT OFF]

[PAGE 17]

Each change order put into effect takes a certain length of time. This should be modifiable under program till we get the things we like. The timing and the combining of the timing should be separately controllable on the basic of “throttle” behaviour.

MOVEMENT Throttle:

[IMAGE #1]

Speed of moving characters:

[IMAGE #2]

Number of change orders being considered by machine under throttle control or number of moving characters

[IMAGE #3]

What I am trying to say is that if you pull hard on the throttle, you should get an overlap (or multiple overlap) of change-order processing; while if you pull softly on the throttle, it processes only a few, or one, at a time.

It should be clear that this exact “feel” is going to be very important and hence must be experimentally variable—especially if it has to be done by light=-pen or even [INDISTINCT].

Probably need a table (program variable):

[IMAGE #4]
(caption to column heads of image #4, left to right) Throttle degree // Change Order Lookahead/Lookback: number to be simultaneously processed) // speed of first c.o. response // speed of #2 resp. // speed of #3 resp.

To determine the [TEXT CUT OFF]

We must complete the change orders currently affecting it (from the table), and the relative speeds they impart to it (from the table). As soon as one change order has been finished, or the throttle setting has been changed, this must be recomputed.

[PAGE 19]

MULTI-DIMENSIONAL STRETCHTEXT
(This spec modifies 1DCH)

This is essentially the same as the 1-Dimensional Continuous Htext, except that provisions must exist to vary several attributes continuously (small change orders, call them Snaps).

Each attribute may have the same tree structure described for the one-dimensional use, except that each also needs to couple to the others. This would be a function of Altitude fall all the different dimensions taken jointly .

LOCATION IN TEXT (expressed as a floating-point number)
Altitude A A A
Dim.1 Dim2 3 4

[IMAGE #1]
(caption to right of image #1) Compound change orders [beyond those decreed separately for diff. dimensions]…
These are specified by the author in terms of relative altitudes of all dimensions, just as regular change orders or specified in terms of individual dimensions.

How to organize this (on disk) I don't know.

[PAGE 20]

PROUSTIAN TEXT EDITING

(Modifies ‘1-Dimensional Cont. Htext’, esp. by permitting chronological forks in change order systems)

Starting from scratch, you may input your text [INDISTINCT], revise it continuously on the screen, having all your revisions remembered with sequence, time and date; go chronologically backwards through revisions, to see the changes; setting it after an earlier point, begin supplying a different set of revisions; declare ‘versions’ (or other parallel structures, such as indexes) at any chronological point in the revision stream or tree; declare correspondences among the versions and structures; and make annotations of any revision, part of version, revision or correspondence between versions.

Indexes of the [INDISTINCT] parts (embracing different versions if desired) may be [INDISTINCT] compiled also as structures ‘corresponding’ to the versions.

Graphs may appear on the screen, reminding the user of his actions and all the material he has now stored. (See ‘Graph Display’)

SCREEN LAYOUT: variable. User shd. be able to move windows and modify his facility graph (ignore for now). (See `XANADU’.)

FILE STRUCTURE. Text segments and change orders, stored as a chronologically branching graph.

[PAGE 21]

These may be stored (see 1-Dim. Cont-Htext) as:

Insertions
Deletions
Switcheroos
AND
Forks or branches, numbered.

Since a common code should probably be used for all chronologically changed material and stretchtext, this modifies that.

2) A further ‘multicoupler’ relation, showing correspondences between discrete units in parallel versions, or structures. (See ‘ELF’’ section)

[IMAGE #1]
(captions above image #1) A1 // A2 // A3
(captions to left of image #1) a // b // c // d
captions to right of image #1) Note: resemblance to a ‘chemical’ structure (see Repetitive discrete Hypertext). Three separate units, here, are coupled with multicoupler A—but the detailed correspondence among sections must be specified for the system by the user.

The multicoupler as required for this editing system has certain odd properties.

[IMAGE #2]

The relation may not be transitive among versions or structures: this is because the elements of one version may become split and dispersed around the screen. Hence the multicoupler must be allowed to fork as changes are made in a particular version. Moreover, several different forms of multicoupler behavior under version change must be possible, depending on what the user wants. The ‘strict’ multicoupler would disconnect from a section once that section was split. The ‘forking’ multicoupler would continue to point to the different parts made from that section. The ‘embracing’ multicoupler would include everything between the split sections, unless the intervening material is pointed at by another part of the [INDISTINCT].

The 1-for-1 property of the multicoupler should also be relaxed [INDISTINCT] optional type 8 multicoupler.

[PAGE 22]

Specialized sequencing subsystem

It is desired to steer the user to sequence materials —his own sections, or structure components—on the screen. This may be done by the usual method of inserting or relocating on a scrollable list; or by method of pair-wise comparisons, where the user couples a present item and then says whether it comes before or after certain other items.

It will be noted that a series of such pairwise comparisons is likely to result in an inconsistent overall graph. This is intended. Given a contradictory set of sequencing choices, he must be showed to undo these choices individually till the overall graph is a satisfactory sequence. We may call this a quasi-sequence facility.

Specialized action pushdown

The user, when in a wildly inspired mood, must be able to Push Down his current activity and skip elsewhere to do something else, then pop up to the dropped activity. This must be possible to a considerable depth.

There must also be a push-away stack, without priority, to which dropped tasks may be relegated for possible return.

[PAGE 23]

Hyper-Manuscripts. Hyper-Libraries

A hyper-manuscript is either an unfinished ordinary text which has ben stored in some complex interconnected form (impossible on paper), or a hypertext with is not yet finished and must be stored in complex forms that include alternatives, undecided. (Continuous hypertexts ignored here).

This is particularly different from Proustian Text Editing. except that it requires treating whole structured texts as the units to be coupled together 'as alternative versions', indexes of one another, etc. This means that the Parts Graph must have graphs as its components: [IMAGE #1](Map of whole hyperscript, where each blob respects the graph of a complete hypertext) (see 'chemical' album under 'Discrete Repetitive Hypertexts')(shd. be recursive molecular [INDISTINCT] ?)

Therefore the multicoupler must couple graph structures as a well as text sections.

[IMAGE #2]
(captions above image #2) Version // Version // Versions // Hypertext Version // Htext Vrsn. // Hthext Vrsn.
(captions below image #2) where [image] represents the multicoupler relation.

It would seem, then, that the notification for graph structures ought to be recursive, so that a specific relation can be notated as glomming into a text section or a graph-structured discrete hypertext. The graph-structure of a a hyper-manuscript includes the graph structures of its component hypertexts.

A hyper-library is a facility which stores hypertexts and makes them available. It has a key problem in common with the hyper-manuscript: if a component hypertext is coupled into (by the writer of another hypertext, or by the student taking notes), this now means that hypertext cannot be changed.

unless there is also provision to save this version of it (coupling it to later versions as well). (while other roles or arrangements might be made, this is the case we have to think about.) Call this a fixating pointer, and a multicoupler into such a version a fixating multicoupler.

This same problem arises with the hyper-manuscript. A component hypertext must somehow be informed that it has been coupled into, so that this version will be saved (or modified perhaps accordingly to some role that preserves the desired part of its content.)

It is further the case that the user (of hyper-manuscript facility or hypertext library) must be able to create pointers, e.g. for annotation, to say any text section, punctuation, type font information, change orders, other pointers, or other recognizable information within the system. And the 'informing' and fixating' mentioned in the above two paragraphs must occur in all these cases.

(Note that we discussed these matters in a a very confused way last June.) This was the purpose of the various acknowledgement backpointers and 'Wilco bit'.)