Journal of Digital Information, Vol 5, No 1 (2004)

A Cosmology for a Different Computer Universe: Data Model, Mechanisms, Virtual Machine and Visualization Infrastructure

Theodor Holm Nelson
The Oxford Internet Institute, Project Xanadu and Operation ZigZag*
Email: tandm@xanadu.net; Web: http://xanadu.com/zigzag

* "ZigZag" is a registered trademark in the USA for zzstructure-based software of Project Xanadu. Other trademarks are claimed, including "zzphone". There is also one US patent.

Abstract

The computing world is based on one principal system of conventions -- the simulation of hierarchy and the simulation of paper. The article introduces an entirely different system of conventions for data and computing. zzstructure is a generalized representation for all data and a new set of mechanisms for all computing. The article provides a reference description of zzstructure and what we hope to build on it.

From orthogonally connected data items (zzcells) and untyped connections (zzlinks), we build a cross-connected fabric of data (zzstructure) that is visualizable, interactive, and programmable.

zzstructure does not have a canonical string representation, as is usual. It is essentially spatial. It is based on criss-crossed lists of cells which are assigned to dimensions. Along these dimensions the cells are viewable, traversible, and subject to operations.

This leads to programming mechanisms built on this fabric; a virtual interactive machine (zzvim) built on these mechanisms; new visualization techniques built on the data fabric and mechanisms; and proposed new formats for the general representation of documents and arbitrary structure -- perhaps less biased than XML.

1 Preliminary remarks

Science is supposedly about universals. Yet much of "computer science" is about the ramifications of two conventions: the simulation of hierarchy and the simulation of paper, carefully developed in a variety of directions. For example, file constructs -- files, directories, their operations and their naming-spaces -- are built around lumps of data arranged in hierarchies. Files must all be named, however irrelevant such naming may be to your concerns, how intrusive or how difficult. (If you wish to have nameless data pieces in a user environment, then they must be inside an "application" which keeps track of these pieces. Data units must be either free-standing named files or members inside free-standing named files.)

Even Web-style hypertext is the simulation of paper and hierarchy: specifically, the Web is simulated paper (composed rectangular arrangements of fonted text), often in hierarchical structures (hierarchical URLs) which in turn are often exposed hierarchical directories on their servers. The new XML-based Web often combines both simulated paper and hierarchy in each file. "Relational databases" are rectangular tables (paper simulation again) which map particular relations into rows and columns.

Text files are generally a form of paper simulation. Even system files are frequently divided into lines and sections, with other concessions to paper visualization.

"Programming languages", too, are often deeply married to paper simulation. In their familiar forms, they are methods of commanding computers based on parsing long strings made of "lines" (the program source code) concatenated sequentially on virtual paper; these are then compiled into shorter strings (target code) which inherit the same sequential organization of the virtual paper. Even though a program branches in complex ways, the program must be manipulated and viewed in a line-based, section-based sequence rather than in ways that accentuate and clarify its nonsequential branching structure. (The so-called "graphical computer languages" generally provide a graphical front-end mask to this unchanging structure.)

To my way of thinking, the great disadvantages of these traditions are hardly recognized, and so few alternatives have been considered. Most computer people see no plausible alternatives that are not merely permutations or coverings of the present paradigm.

I propose an alternative cosmology that appears to be radically different in every aspect.

We begin with a very simple set of rules for building blocks and their assembly-- one type of unit, one type of connection-- and then assemble them into a variety of mechanisms. This in turn we intend to set up as an interactive conglomerate for doing anything with data or programmable machines, the zzvim and ZigZag Personal Environment.

1.1 Starting over

I propose to go back to the beginning and branch in another direction. Let's go back to 1945, say, when anything seemed possible, and when conventional wisdom could be challenged more easily because it had only been made up a few months before, perhaps by another 20-year-old in the same lab.

The evolving conventions of those days involved lump files with short names, hierarchical directories for storing them, programs as long strings, databases as tables stored in files. (These would later be packaged into "folders" and "applications".)

But now imagine that a few guys had snuck off to another lab and created a whole new system of conventions, completely different, for everything. Such an imaginary team, unfettered by years of courses, journal articles, meetings, ads and handouts, just might have come up with the following ideas.

1.2 Why change things at all? And why this particular structure?

Many people feel great discomfort and discontent with today's computer world, but are offered -- and imagine -- no alternative pathway,

We think our system is better and much simpler -- conceptually and implementationally -- than the computer world as

generally known. For instance, we believe it will do much of what a relational database will do more simply than the existing methods. (Efficiency and scaling are issues we postpone.)

We see the advantages of this system as:

- minimalism
- simplicity
- elegance
- ease of use
- generative power

The design I am about to present is not arguable. There is no right or wrong about such a design, except that it is good and usable. But it must be seen and touched to be understood. Enthusiasts say, "Once you understand it, everything else seems so complicated!"

1.3 Relation of this system to hypertext

Many ask me how this system relates to hypertext, and to my earlier and ongoing work on Project Xanadu® (the original hypertext project) from 1960 to the present (Nelson 1965, Carmody *et al*. 1969, Nelson 1981, Nelson 1987). The relation is quite indirect.

zzstructure is not hypertext.* It is a generalized representation for all data and a new set of mechanisms for all computing, not particularly related to hypertext. On the other hand, Project Xanadu, and my work on text systems in general, are about a far different kind of text system from the Web. What unites them is that I now believe that zzstructure inside and behind hypertext systems is by far the best way to represent and implement new textual forms. (See discussion later.)

*While it is composed of nodes and links (like the common hypertext forms), by itself it would make very bad hypertext.

1.4 Psychological resistances

Paradigm confrontation is rarely fun for people; new ideas tend to be unpleasant and threatening. This is true of zzstructure: people find many reasons not to like or understand it. We frequently hear such remarks as "You could do it in relational database", or simply "I don't get it."

We tell people you have to use it hands-on to understand it, yet our technically-minded friends often don't want to touch the system itself, but rather perform in their minds some kind of logical analysis. This is essentially useless. Very few can understand the system by analysis. Without hands-on, no one sees what it does or means.

It also helps to have a certain liking for spatial structures. Architects and fighter pilots -- people with a special aptitude for spatial relations -- appear to like it especially. (A former fighter pilot said, "You've found a sweet spot between representation and visualization.")

1.5 Benefits and powers of the system

I will be enumerating six parts of this whole design: zzstructure (consisting of zzcells, zzlinks, and zzdimensions); mechanisms built from zzstructure; the ZigZag virtual machine; and a generalized visualization infrastructure that emerges.

This overall system has a number of beneficial and powerful aspects. (I call what follow "aspects" rather than "features" because I think of features as separable and severable aspects of software, such as a word count routine or colored fonts, which could be removable from a program without altering the rest of it, whereas *aspects* of a system are intrinsic, inseparable and omnipresent therein -- pervasively present in all its parts.)

- the system is intrinsically non-hierarchical;
- all information, whether simple or compound, may be represented in a compact and visualizable way;
- there are intrinsic visualizations for everything, viewable in rows and columns, to considerable benefit;
- operations are defined in rows and columns, to considerable benefit;
- the system is highly minimalist, some say elegant:
  - there are very few fundamental elements;
  - everything is represented compactly with surprisingly few units;
- everything appears to be easier to do:
  - database is easier;
  - programming is easier;
  - interfaces are easier (new views + key reassignments);
  - most problems and "application areas" -- at least that we have found so far -- can be solved easily, with only a few new views, operations, cell types or other simple recourses
- the system is truly integrated (the term "integrated software" was popular in the mid-80s):
  - viewing operations are closely related to program operations;
  - instead of "applications", separated zones of function and usage connected by the narrow channels of clipboard and file export/import, we have "applitudes" which are deeply interconnected to the whole, amongst themselves, and amongst their parts;
- certain complex things can be easily seen and worked with:
  - the equivalent of some complex programs in conventional database become simple input;

- some complex database queries become simple views.

However, zzstructure and ZigZag violate all conventional paradigms.

- hierarchy is optional. Einstein said: "Everything should be as simple as possible, but no more so." To this we add: "Everything should be as hierarchical as necessary, but no more so."
- nothing corresponds to paper, except very indirectly.
- this violates everything in the modern interface (introduced to the public in 1984 -- what most people call misleadingly the Graphical User Interface or GUI, but we call the PARC User Interface or PUI).
- it is not WYSIWYG (what would "get" mean?)
- it offers a different programming model (small programs, encapsulated in zzcells, which accept zzcells as input and generate zzcells as output)
- there are no "applications", meaning separable programs for separate purposes.
- there are a number of paradoxes (violations of expectation):
  - **spatial paradoxes**: spatial inconsistencies compared to more familiar spaces.
  - **structural paradoxes**: for example, more than one cell may be at an intersection of two ranks.
  - **dimensional paradoxes**: ZigZag dimensions can be tricky and surprising, and aren't like dimensions we have seen elsewhere.

(For more counterintuitive aspects, see Appendix 1.)

These preliminary remarks finished, let us discuss the structure itself.

2 zzstructure explained as a kind of spreadsheet

A conventional spreadsheet is an arrangement of connected cells in two discrete dimensions. The connections are symmetrical but may be said to have a direction, since you can select a direction to step in.

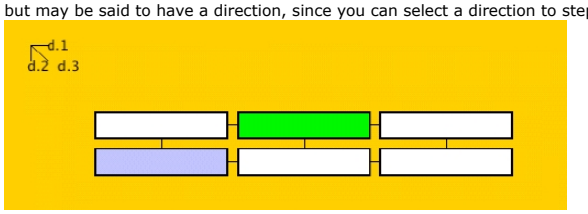

Figure 1. Conventional spreadsheet of two dimensions (d.1 horizontally and d.2 vertically). Default (understood) directions: rightward is posward in d.1, downward is posward in d.2

Here are the spreadsheet properties we are generalizing or adapting:

1. A spreadsheet has two dimensions
2. Each cell has at most two neighbors horizontally and at most two neighbors vertically
3. The cursor sits on a cell, selecting that cell
4. You may move the cursor horizontally or vertically
5. You may edit the contents of a cell selected by the cursor
6. You may create new rows or columns
7. You may delete rows or columns
8. You may rearrange rows or columns
9. You may view a local portion of a spreadsheet when the whole cannot be seen all at once.

All of these spreadsheet facts continue on into the zzstructured world. The generalizations are:

1. A spreadsheet has two dimensions
   => In zzstructure, you may have N dimensions -- that is, a largeish finite number.
2. Each cell has at most two neighbors horizontally and at most two neighbors vertically
   => SAME. We generalize this to at most two neighbors in every dimension.
3. The cursor sits on a cell, selecting that cell
   => SAME; except "sits on a cell" is now assimilated to connection on a specific dimension, d.cursor.
4. You may move the cursor horizontally or vertically
   => SAME; except you may move it in more dimensions as well.
5. You may edit the contents of the cell selected by the cursor
   => SAME
6. You may create new rows or columns
   => SAME; except a row is generalized to a "rank" in any dimension you like, which you generally create one cell at a time.
7. You may delete rows or columns
   => SAME; except generalized to "ranks" in as many dimensions as you like.
8. You may rearrange rows or columns
   => SAME; except generalized to "ranks" in as many dimensions as you like.
9. You may view a local portion of a spreadsheet when the whole cannot be seen all at once.
   => We generalize this: any structure, or substructure, may be viewed locally in 2D as if it were a spreadsheet; but as it is likely to be irregular, its connections may go in various directions.

2.1 Explanation of zzstructure based on the spreadsheet

I would like to give a constructive explanation of zzstructure, meaning that rather than start with a description, we consider

how it is made. However, this explanation is also ***de*** constructive, in that we are making this new world by breaking up certain traditional relationships.

zzstructure deconstructs the spreadsheet by making it into smaller, freer parts.*

*Note that it also generalizes and deconstructs my original design of 1965, as described in my earliest refereed paper (Nelson 1965). This was a design for side-by-side lists with visible criss-crossing connections, called at that time "zippered lists" (later "zipper lists" and "zips") (Figure 2). Zipper lists are closely related to the structures presented here. Note that structures and views identical to 1965 zipper lists could be built with zzstructure; thus zzstructure generalizes zipper lists as well.

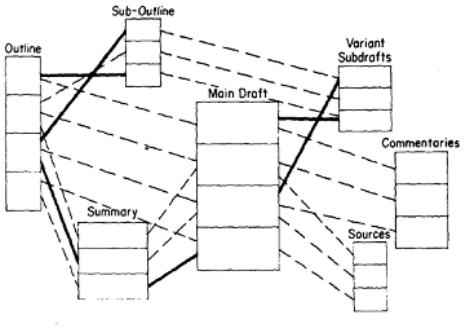

Figure 2. Zipper lists (Nelson 1965)

Let us take a spreadsheet and set its individual cells free, allowing them to be connected individually to other cells in arbitrary structures (see Figure 3)

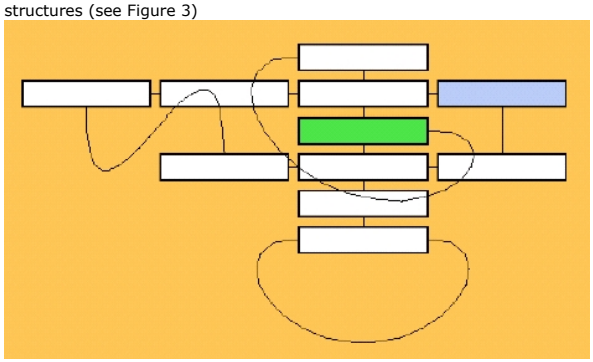

Figure3. Loosened structure of connected cells (in dimensions 1 and 2)

We set each cell free to have its own connections, regardless of its neighbors' connections. But we retain the spreadsheet rule: each cell may have no more than two connections in each dimension-- one positive and one negative. And we may in addition create other dimensions at any time. That is zzstructure.

The cells are simple and the connections are simple (about which more later). The dimensions, which we call zzdimensions or zzdims for short, have puzzling aspects which take a good deal of effort to understand fully; but these puzzlements need not interfere with preliminary understanding. Just think of them as generalizations of the two dimensions of the spreadsheet.

In what follows, we will go through the primitives of zzstructure; then compound structures, mechanisms and visualization primitives.

3 The primitives of zzstructure

We will consider the primitives of zzcell, zzlink and zzdim.

3.1 Primitive 1: the zzcell

The cell, or zzcell, is the principal unit of the system. We may visualize it as a box, sphere, or any other simple object.

The zzcell is a first-class object, meaning it is independently addressable and has a persistent name, in principle referentially available from anywhere. This means each cell has a unique identifier, which is a way of permanently and reliably addressing the cell and/or its contents. The identifier does not have to be chosen by the user.

3.1.1 Containing cells, positional cells

A cell may or may not be a container for contents, such as text, graphics, audio, etc. Some cells have only a positional or topographical function.

3.1.2 Cell types

Cells can have types, either based on their function or the type of data they contain:

- Various types of data cells. A cell may contain, or virtually contain, only one datum. More specifically, a primitive data cell, the fundamental building block of ZigZag, can contain only one unit of data of one type. Examples of these include: span of text, span of sound, span of movie, .jpeg image. (Note: only text has been implemented so far.)
- Various operational types of cell. These may have a variety of different properties and functions, such as:
  - executable program or script ('progcell')
  - label

- trefl (Text REFerential celL, pointing to external text)

3.1.3 Directionality

Cells have only one positive and only one negative side in each given dimension (Figure 4). These designations are used in linking (see links, section 3.2).

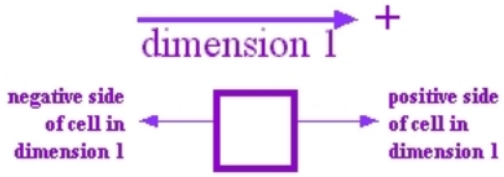

Figure 4. Cells in ZigZag have two sides, and hence no more than two connections in any given dimension. Here shown for d.1

3.1.4 Composite units and structures, and their representation by single cells

Composite structures, such as those needed for a captioned image -- a unit which groups an image and its caption -- are compositions of primitive cells. However, these may be in turn be represented for reference by single cells.

A commonly asked question is, "Can a cell contain more than one type of data?" In the case of such referential cells, the answer is, "Yes, sort of"; the referential cells represent the package of different cells.

3.2 Primitive 2: the link, or zzlink

The second fundamental unit of zzstructure is the link or zzlink, which may be thought of as generalizing the link of the spreadsheet.

A link is a connection between two cells in a specific discrete dimension (zzdim, discussed in section 3.3).

3.2.1 Links are anonymous and untyped

While the cell has contents and an identifier and a type, the link is anonymous.

- Unlike the zzcell, the zzlink is not first-class (it is nameless and not addressable)
- Unlike the cell, the zzlink is allowed to have no type.

3.2.2 zzlinks are symmetrical but directional

- zzlinks are intrinsically symmetrical in representation and mechanism
- zzlinks are directional, so that we may refer to posward and negward cells, movements and operations.

3.2.3 A zzlink connects two cells

A zzlink is only one-to-one between two cells. Links are not *n*-ary, many-to-one or one-to-many. All these relations can be represented, but by compositions of zzstructure.

3.2.4 A cell has only two links per dimension

Any cell can only have two links, at most, per dimension, respectively posward and negward. A cell's positive side can only connect to a negative side of a cell in any given dimension, as illustrated in Figure 4, or indeed to its own negative side. Accordingly, a cell may connect to itself in any given dimension.

3.2.5 Directions: posward and negward

"Posward" and "negward" are the directions of zzstructure. (Note that vertically positive in zzstructure is usually by convention downward, as on a spreadsheet, rather than upward, as in Cartesian coordinates.)

"Posward", of course, means in the positive direction, and "negward" means in the negative direction. The coordinate directions of "posward" and "negward" apply to:

- links
- cells
- ranks
- ringranks

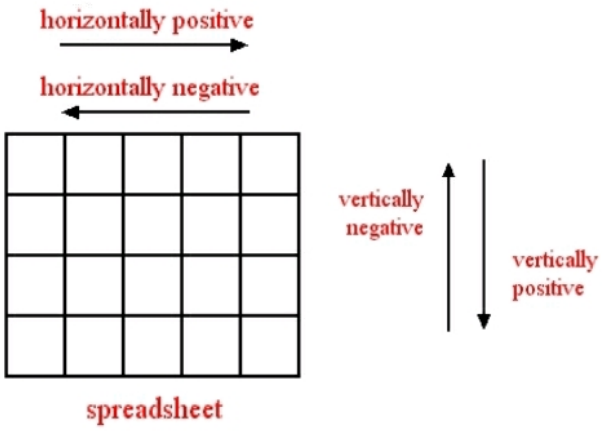

Figure 5. Posward and negward connections in zzstructure derive from spreadsheet directions

3.2.6 Directions are conventional

Directions are assigned meaning by convention, either the general conventions of zzstructure, ZigZag or some local system.

3.3 Primitive 3: zzdimensions, or zzdims

Every zzlink is in a particular dimension (zzdimension or zzdim). These are not dimensions in a continuous space, but have curious properties of their own.

The best way to state this abstractly has not been found. The zzdimension is the hardest concept to explain in zzstructure, though I believe it is exactly the same kind of dimension that spreadsheets have two of. Mathematically-minded individuals have had a particularly difficult time trying to express it in some way more familiar to them.

3.3.1 zzdim is discrete in two senses

The zzdimension is discrete in two senses:

- it holds only discrete or enumerable elements (cells)
- zzdims are separate:
    - there is no such thing as a diagonal

3.3.2 Counterintuitive fact: all cells exist in all dimensions

All cells exist in all dimensions. Conversely, each new dimension embraces every cell. However, this only means that as soon as a dimension is created, connections can be made along that dimension to any cell in the system.

Although a cell may not have any links in a given dimension, a new link may be made in that dimension at any time.

3.3.3 Meanings of particular dimensions

General zzstructure (as an abstraction) assigns no built-in semantic meaning to zzdimensions. This too is like a spreadsheet: what do "horizontal" and "vertical" mean in a spreadsheet? It all depends on what conventional meaning they are assigned, generally relationships of various kinds. The same is true of general zzstructure: any meaning may be assigned to a dimension in a given local system of zzstructure.

However, as a matter of convention, a number of dimensions have been reserved and given meanings in our system. In section 3.4 we depart from abstract zzstructure. And, in the virtual machine (section 9), we also delegate specific operative dimensions (such as d.cursor and d.clone) to specific mechanical functions.

3.4 Explaining zzstructure in more conventional terms

Unfortunately, few people find the previous explanation (deconstructed spreadsheet) satisfying. Many people would rather ignore the previous explanation, thinking they can translate ZigZag into more familiar or conventional terms. Here madness lies, because every translation to other terms is misleading.

As with other new paradigms, attempting to state zzstructure in prior terms loses its integrity, uniqueness, inner meaning, and magic. Such new paradigms as the atomic bomb, DNA, object-oriented programming and hypertext are very misleading when expressed in any previous terms*, and this system just as much so.

---

*For decades I endeavoured to explain hypertext as the obvious next stage of writing and literature, to no one's understanding whatsoever.

Moreover, once someone has found conventional terms to describe it they can easily dismiss it with no sense of its power, since they think it is just a special case of something else, such as "linked lists." We believe that

1. considering zzstructure as a generalization of the spreadsheet, deconstructed to individual cells, seems the clearest way to go;
2. hands-on experience with zzstructure almost always necessary to understand it.

3.4.1 Alternative brief description

With that warning, let us compactly state zzstructure in other terms.

zzstructure can be described as a list-based system where each list belongs to a class called a "dimension".

Lists in the same dimension cannot share cells (Figure 7), but lists in different dimensions may share cells and be viewed at right angles (Figure 8 and Figure 9). Any number of lists may intersect, so that a zzcell can be on many lists at once.

Semantically, a relation may be assigned to a particular dimension; connections in these dimensions may then represent these relations applying between cells.

This approach facilitates many views and operations, and greatly simplifies many forms of information management.

4 Emergent concepts of zzstructure: the space, and configurations in it

So far

- everything is a zzcell or a link between zzcells
- all links are of one standard kind: bidirectional, untyped, between pairs of individual cells
- zzdims ("dimensions") are classes of links that may be assigned relational meanings, shown at right angles and operated on by new operations.

From these primitives of zzcell, zzlink and zzdim emerge a number of different structures.

4.1 Emergent structure 1: zzspace, a pseudospace without coordinates

zzstructure creates a kind of a space, a paradoxical pseudo-space, which both simplifies and provides broad cognitive access and understanding. It is both visual and operational. Explored with cursors, it may appear to be a kind of a familiar space, but it has a number of special properties:

- It exists only around the cells themselves; there is actually no "space", just the structures within.
- Its paradoxical characteristics can be managed and understood.

zzspace is a pseudospace without coordinates. It is simply the visualizable and operational system of constructed relations of cells and links. But it is not just visualizable, it is operational, since the connections and turns (zigs and zags) in the space are significant and consistent.

Much of the system's benefit comes from the intuitive understandability of this space.

4.2 Emergent structure 2: rank or list

"Rank" is what we call a series of cells connected sequentially in any dimension. We adopt Iverson's (1962) term "rank" as the generalization of viewable row and column, rather than "list", which has no sense of dimensionality. As in Iverson's APL language, a rank is a connected series of cells along a specific dimension, like a row or a column in a spreadsheet.

It would be simpler if we only had two dimensions: we might call a series of cells a "row" in one dimension and a "column" in the other dimension, as we do on the spreadsheet.

However, ZigZag has potentially many dimensions, but there are no terms in English beyond "row" and "column" for the rest. Moreover, to say "row" and "column" in ZigZag could be misleading even for two dimensions, since we may choose which to see vertically, or even choose angles to view them that are neither vertical nor horizontal.

Thus we drop the terms "row" and column," and simply say rank.

4.2.1 A rank is not a dimension

A rank is not a dimension; a rank is *in* a particular dimension. This fact causes confusion for many beginners but is fundamental. Many ranks can be in the same dimension (Figure 7).

4.3 Emergent structure 3: headcell

We often want to refer to the end, or commanding cell, of a rank. For this we use the headcell, which is the negmost cell in a rank, (e.g. at the top of a column or vertical rank).

4.4 Emergent structure 4: ringrank

The rules of connection allow a rank to be a ring (Figure 6). This has many uses.

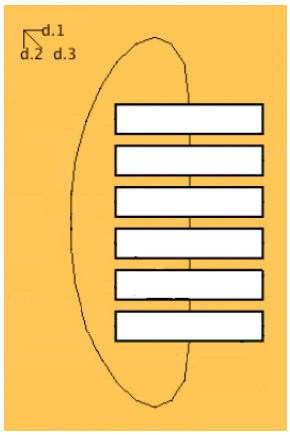

Figure 6. Ringrank in d.2

(A ringrank may have a headcell; ideally, this is the cell most desirable to jump to quickly. It may be chosen by the user or by some algorithmic method.)

4.5 Emergent structure 5: parallel ranks (on the same dimension)

Ranks in the same dimension are considered parallel (Figure 7). They cannot intersect, and cannot have cells in common.

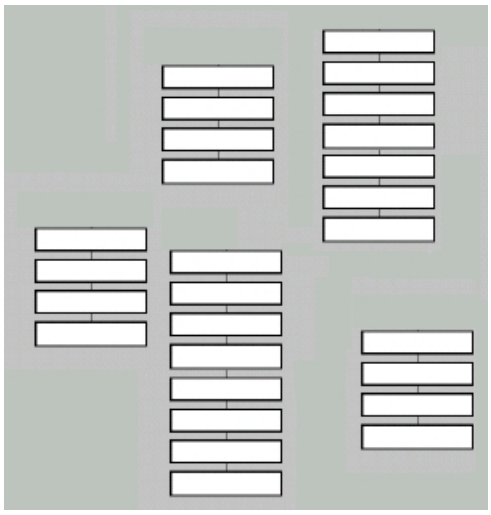

Figure 7. Parallel ranks in some dimension, seen vertically

Ranks can be parallel whether or not they are ringranks, since a ringrank and a straight rank can be in the same dimension.

4.6 Emergent structure 6: intersecting ranks (on different dimensions)

Ranks in different dimensions may intersect, or have cells in common. We may view them as crossed, or perpendicular to one another.

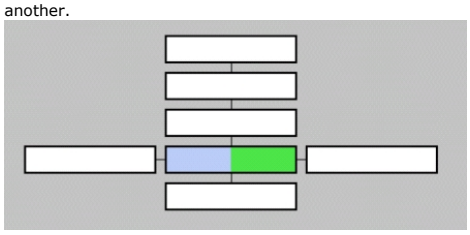

Figure 8. Two perpendicular ranks on two dimensions with one cell in common

Paradoxically, however, two ranks may have several cells at their intersection, although it may be difficult to comprehend from looking at the view (Figure 9)

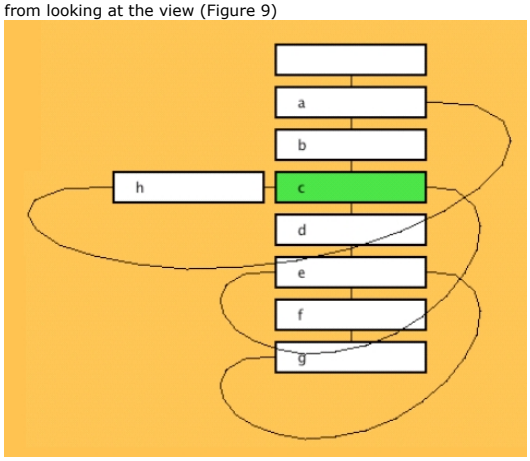

Figure 9. Ranks perpendicular on two dimensions with four cells in common

In Figure 9, a horizontal and vertical rank share cells a, c, e and g.

5 zzstructure conventions and enactments

So far we have defined a general but meaningless system, abstract zzstructure. Anyone might have taken the ideas of zzstructure and developed them in different directions; what follows is our own approach to building it into a generalized computer world.

To colonize it and make it useful, we must decide a large number of conventions. This is where generalized zzstructure becomes a particular package. In what follows I will describe the dimensions and structures of ZigZag software and (later) the ZigZag Virtual Interactive Machine.

5.1 Dimensional assignments

The first conventions that must be decided are the meanings of dimensions.

Some dimensions are passive and nominal, merely receiving and presenting data. This can be useful by itself (see genealogy example, section 8.2). Other dimensions may be operational, programmed to monitor changing zzstructures and events, and calculate and present results automatically.

5.1.1 Passive dimensions: d.1, d.2, d.3

For example, by convention the first three dimensions (available by default) are named d.1, d.2 and d.3. These can be conveniently thought of as corresponding somewhat to paper. d.1 can correspond to text written continuously from right to left, like an old-fashioned ticker-tape. d.2 can correspond to a page of text, in which that tape has been cut into segments and pasted onto paper (as on an old-fashioned telegram). d.3 can correspond to successive pages. In principle, these could be built into in rectangular sheets or even stacked cubes. However, the many possible interweavings of zzstructure make sparse, irregular and criss-crossed structures much more common.

5.1.2 Operational dimensions: d.cursor, d.clone, programming dimensions

So far we have just described passive dimensions. However, other dimensions may be programmed to make automatic calculations and presentations. Two examples are d.cursor and d.clone

- d.cursor is an operative dimension which connects a cursor to the cell it selects (its "accursed cell"). A cursor moves among cells by attaching itself successively to one cell after another along d.cursor. The connection on d.cursor is fundamental to viewing. After each operation of the system (cursor move or change of zzstructure), a view centered on the accursed cell is calculated. The system looks negward on d.cursor to the accursed cell, and then refreshes the local view starting at that accursed cell. (Views are discussed in section 7.)
- d.clone is an operative dimension which connects all clones of a given cell as a single rank. The headcell holds the contents; each clone, when displayed, shows the contents of the headcell.

5.1.3 Avoiding dimensional collisions

It is possible, in planning ZigZag applitudes, to have in mind too few dimensions, not recognizing that there are already plans for the use of a given dimension which might cause interference. Therefore, in designing zzdims and planning zzstructure functionality, it is important to keep dimensional functions separate. A dimensional collision is an inconsistent use of the same dimension for two purposes, causing an ambiguity or incorrect reference.

It is far better to have too many dimensions than too few -- where "too many" means "more than necessary".

6 zzstructure mechanisms built from primitives: the ZigZag hyperstructure toolkit

zzstructure by itself has no mechanisms; it provides only a representational fabric. But from zzcells and zzlinks we can build mechanisms for computer operations of all kinds. Having begun with an extremely simple set of rules for one building block and its specific connections -- one unit structure, one connection structure -- we can assemble them into mechanisms for all computer operations. We may call this a hyperstructure toolkit.

Our desire is to build a complete new computer world, so these are the mechanisms we have chosen as steps toward a complete system for arbitrary data representation and arbitrary computer operations, with all mechanisms contained in these selfsame structures.

6.1 The main mechanisms of ZigZag

We have found a number of zzstructure mechanisms which work very well.

6.1.1 Mechanism 1. Pointers

Pointers are cells. A pointer need not be of a special type, since it is the mechanism that counts. Any routine or progam may create a pointer directly connecting a routine to its current operand cell(s). Sometimes this requires creating a special dimension to avoid colliding with other dimensions.

6.1.2 Mechanism 2. Cursors and their view

A cursor is a pointer with an associated view. The cursor is directly attached to the highlighted cell ("accursed cell") in a reserved dimension (d.cursor). The view mechanism then uses the cursor as the starting center for refreshing a current view of the cells around that cursor.

6.1.3 Mechanism 3. Wheels

A wheel is a ringrank of cells that effectively turns, operationally, as one or more pointers step around it.

Ringranks with stepping pointers are used for a number of repetitive operations or data:

- 'next dimension'
- 'next view'
- 'next operation' in zzstructured languages [hypothetical]

A pointer can be directly attached to such a wheel of cells and made to step around it. This is how we do a repeating list of views; this is how we manage repeating lists as ringranks.

6.1.4 Mechanism 4. Cells as programs

Cellular programs, in this system, have a number of properties:

- programs and routines are in cells
- a routine's incoming arguments are cells
- a routine's outgoing results are cells

This can lead to new kinds of programming languages organized on zzdims. These are unlike conventional languages, where programming is segmental in a long string. A number of zzstructured languages have been proposed; two have been implemented to some extent:

- Kaijanaho and Fallenstein's (2001)"Flowing Clang" in Gzz, organized on d.xeq and d.1
- Nelson's (2003) "Nuzzl" (NU ZigZag Language), implemented by Jeremy Smith

6.1.5 Mechanism 5. Clone cells

Often we want to provide the same content in two or more places, i.e. of providing multiple reference to a particular datum from different cells.

Clones are a special mechanism within ZigZag specifically designed for transclusion (the knowable identity of more than one

thing), in this case putting a cell understandably in two or more places. Because many-to-one connections on a dimension are forbidden in the cosmology, there are two alternatives:

- use of additional dimensions
- mimicking many-to-one connection by other structures
- allowing distributed live copies (clones) of a cell.

How then can you get the same piece of information in two places? The answer is that you clone the cell that contains it. A clone is a cell that contains a transclusion (in this case, a dynamic reference) of the cell it represents.

The clone mechanism is very simple, and also shows how we integrate zzstructure with other intended functions. Clones are connected along d.clone to their source cell, which is the headcell of that rank in d.clone. By interpretation, all these cells have the same value, and changing any one changes them all. The current value of the original cell is instantly present in all clones, allowing consistency and integrity to be maintained.

(Multiple reference to structures of multiple cells is more complex and will not be discussed here.)

6.1.6 Mechanism 6. Operative units

Often multiple cells work together as one unit (either data or programs). We may call this an operative unit.

6.1.7 Mechanism 7. Maincells

In an operative unit, we let one cell stand for the whole -- denominating one cell as the cell to refer to when we wish to refer to the whole unit. This is an operative unit's maincell. The maincell may be the headcell or negend of its rank. A maincell is expected to be connected directly to its supporting cells.

Example: representing a human individual. Ordinarily a person is represented in a database by a given name and family name. However, in zzstructure we wish to have one cell to refer to each entity; therefore by convention we take the given name as the maincell representing that person.

6.1.8 Mechanism 8. Compound cell

A compound cell represents one level of indirection greater than a maincell. A maincell is a single cell interpreted to represent its neighboring cells in the operative unit. Compound cells can go farther, however.

Various types of compound cells, where one cell represents a structure of more than one cell type, have been proposed and a few have been implemented.

Example: in later Azz versions (see Table 1), there were compound cells using a containment mechanism, whereby one cell contains others by interpretation (on d.contain and d.contained-set).

6.1.9 Other mechanisms

A number of other structures have already been discovered and propounded, which are not yet fully documented.

Edited to here.

7 zzspace and zzviews

As already explained, zzspace is simply the pseudospace generated by zzstructures, which may be viewed in various ways. Its usefulness is in

- its visualizabilitiy
- its ability to represent relations visually
- the ease of designing, creating and managing structured operations

A view is a presentation of some portion of zzspace, seen in some way. A view is presented by a view program, which presents

- a region around a particular cursor
- a pattern of cells around that cursor explored in a certain sequence ("view raster")

There may be other kinds of zzview in the future, but this describes all that have been created so far.

7.1 Pseudospace: views of this structure

zzstructure generates a pseudospace that is somewhat comprehensible visually. We try to make views that make the space -- that is, the relations contained -- as clear as possible. There is no canonical visualization; we need to try to imagine zzstructure any way we can. This means looking at it in any way that may be useful, and often simplifying it by viewing specific dimensions two or three at a time.

In a given view, many of the connections may not be shown; but the user is free to rotate the view, move the cursor to another cell, or switch to another view, showing the same region in a different way. Any view will accentuate some features and leave out others. What best shows a particular conglomerate of cells for a given purpose is up to the user.

It is important to note that in ZigZag all views are valid (and WYSIWYG is meaningless). But new views may be programmed with relative ease in order to see and understand the space better.

7.2 Viewspaces (canvases, projection spaces, presentational fields, viewing tanks)

There is no canonical viewing mechanism for viewing zzstructures. We may in principle use any graphical system as a canvas or projection space, presentational field or viewing tank. For instance, the Azz prototype used "curses" (a classic Unix utility extracted from Bill Joy's "vi" editor) as a canvas to show zzstructure as arrays of colored characters and spaces. In the Zzz development system, we are using OpenGL as a 3D projection space or viewing tank.

7.3 Typical view structures

Typically, a view begins at an accursed cell and works its way outward.

Two views have been built into all ZigZag implementations: the I, or row view and the H, or column view. These are so-called because of what connections are displayed on the screen. In the I view, horizontal ranks are shown emanating from a central column (rather like the capital letter I); in the H view, vertical ranks are shown emanating from a central row (rather like the capital letter H).

Many more views are possible. (Lukka's group has programmed more than a dozen, some of which -- such as the "MindSunDew" view-- are wonderfully expressive; see Figure 10.)

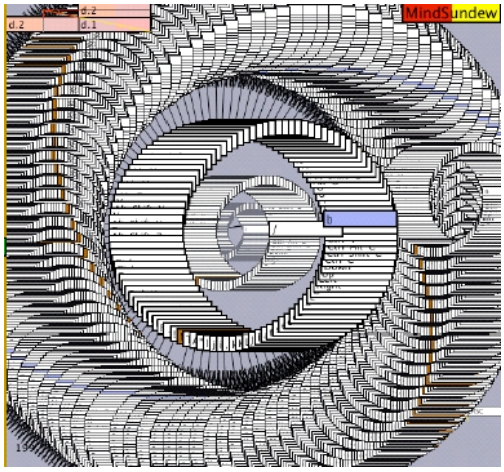

Figure 10. "MindSunDew" view in ZigZag-Gzz, by Tuomas J. Lukka's group. System lists are shown, viewed both horizontally and vertically in d.2

Our recent experimental version, Zzz, presents primitive viewable units (vunits) in OpenGL, a standard three-dimensional viewing system (Figure 11).

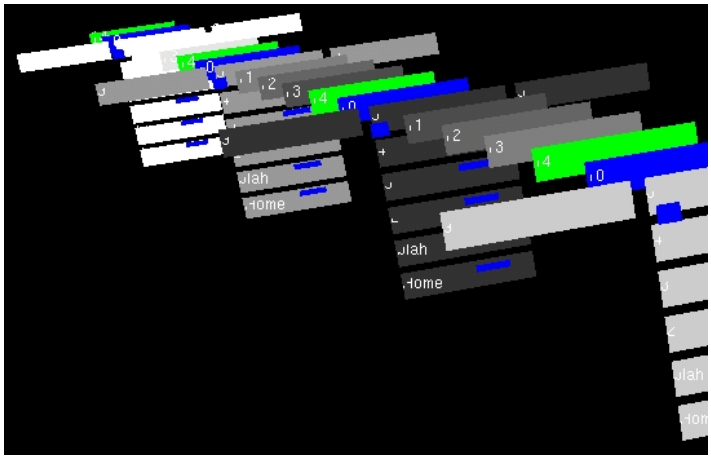

Figure 11. OpenGL cellviews in Zzz. A forward-to-back ringrank is shown, repeating without unifreshment

7.4 Views and presentations for the visually impaired

While all the zzviews so far use graphical output, there is no reason that other forms of presentation cannot show the same structures. For some time we have been interested in auditory presentations, where voices automatically read the directions of stepping and the contents of cells. This remains on our list of things to try.

7.5 Unifreshment

The initial views (H and I) look quite confusing, since a cell may turn up in several places at once without being easily recognizable as the same cell. A problem is how to recognize a cell appearing simultaneously in several different parts of a view (because it is on several ranks you are viewing).

There are a number of possible views that clarify this, called collectively "unifreshment" views. These are refreshment methods that unify different screen instances of a cell. They may only show each cell once, or otherwise unify the view so that the identities of cells are recognizable.

For instance, Tuomas Lukka's Gzz (Table 1) uses a system of unifreshment that we may call collapsed unifreshment. This shows each zzcell in the view only once, and presenting its other connections as 2D spline-lines.

Another method of unifreshment would show separate instances of a cell, refreshed in different locations in the same view, by the same color, by blinking, or by user interrogation. We hope to explore new 3D methods for this in OpenGL.

7.6 zzspace in 3D

zzspace is simply the pseudospace generated by viewing zzstructures. Not only is it desirable to view zzstructures in 3D, but it should be possible easily to view more than three zzdims at once. This would be done by assigning a 3D angle to each dimension.

Suppose we are refreshing straight ranks on the screen. Each straight rank in a specific dimension is refreshed as a series of cells laid out in that dimension's 3D angle from that rank's origin of refreshment. (Ringranks would also have a hoop angle.)

Thus, given a specific point of origin for the accursed cell generating the view, each 3D view steps outward on ranks, presenting each cell in the 3space position generated by that 3D view.

It should be simple to generate such 3D views for four, five or even more dimensions at a time; the question is how comprehensible such a view may be to the eye and mind. (Particularly if the issue of unifreshment is addressed as well.)

7.7 Vunits in Zzz, and their control cells

In Zzz (our current experimental system), a "vunit" is any viewable unit shown in three dimensions. Vunits include:

- the zzcell
- the cell ID of the accursed cell
- the flying text which shows the cell's content
- the visible connectors between cells.

Thus a cell in viewed OpenGL 3space is represented as three vunits (a rectangular solid, the cell ID and the cell's text contents) plus the connectors shared by each pair of cells within the view.

In accordance with ZigZag philosophy, each vunit should have its control cells exposed within the zzspace conglomerate. Thus the parameters for position, sizes and color of each vunit are accessible as cells. This allows all of them to be changed programmatically by cellular programming. However, since each control cell may also be a vunit, and each of these vunits may in principle have control cells, there is a recursion problem which needs to be stopped by some means. The ideal method for this is not yet known.

8 Combined methods for doing real tasks (applitudes)

In zzstructure systems, we do not speak of "applications" (deeply segregated zones of separate functionality) but of "applitudes" (relative neighborhoods of connectivity where things are accomplished in certain ways).

8.1 Simplicity of applitudes so far

So far only a few people have worked with ZigZag, attempting only a few things. However, we have found each endeavor to be surprisingly simple.

There are basically four alternatives when building some applitude.

- new dimensions
- new cell types
- new operations (encapsulated in a clonable cell)
- (most important) new conventions

In our experiments so far, we have found each objective easier than we expected for each problem.

In most cases so far, two or three of these alternatives have tended to suffice -- say, two new dimensions, or one new dimensions and a new cell type. (Most of these have been *Gedankenexperiments*, but we believe from results so far that this will hold.)

8.2 Example: genealogy

By the judicious construction of dimensions, we can do a surprising amount. For instance, in our classic demo, we demonstrate what would be a conventional "genealogy application" simply with a number of ZigZag cells representing individual people and their family connections. Connecting these cells on four zzstructure dimensions -- d.1, d.2, d.marriage and d.children -- accomplishes much of what would conventionally be a genealogy application requiring both programming and database management. Instead of creating such a program and working with such a database, the user merely inputs the information correctly, creates the two special dimensions (d.marriage and d.children) and the work is done and ready to explore. Stepping through cells and views shows a great deal of what you'd want to know and see from a genealogy program. (Additional functions of course may be created indefinitely.)

8.3 Example: paint applitude

We have been asked what a "paint application" would look like in ZigZag.

First of all, it would not be a separate application, but an applitude, combinable with other functions. Separate cells would represent all the parts, both visible and hidden. Whether it would look like a separate application, or appear to merge visibly with the rest of the user's ZigZag environment, would be up to the designer(s).

Second, all its visible parts would be cells:

- the paint canvas would be a cell;
- the menus and selections would be individual cells.

Third, its interior mechanisms would be cells -- a clockwork of cellular connections among reusable and combinable mechanisms. Separate cells would represent all the parts:

- The visible mechanisms -- say, a "brush" or "pen" or color selection window -- would be represented internally by cellular mechanisms.
- The internal structures of the system would be represented by cellular mechanisms. This would include graphical buffers and lists of current state -- color, brush size...
- The connections between internal structures would be represented by cells.
- The operands would be represented by cells. For example, the current x and y positions of the brush would each be maintained in a specific cell.

If designed within the ZigZag philosophy, these functional structures would be designed for flexibility -- ideally, to be mixed

and matched with functions developed for other purposes.

9 The virtual interactive machine (zzvim)

We generally view and manipulate the universe of zzcells through what we call the zzvim, or ZigZag Virtual Interactive Machine. The design of the basic zzvim appears to have become stable, as instantiated by the several zzvims that have already been programmed (listed in Table 1). The zzvims sharing this structure have been in particular Azz, Ezz, Gzz and Zzz; Lzz appears to be a borderline case.

The mechanisms of the zzvim are highly connected and integrated, like clockwork. The zzvim already has many conventions.

Two windows. A zzvim is expected to have at least two windows, each with its own cursor at the center.

Power-user interface. The zzvim receives keystroke commands in Kblang (the KeyBoard LANGuage of ZigZag). (section 9.1.1)

Menus. Menus are built into the system as executable cells; the left cursor (Action or Control cursor) may be put onto an executable cell and fired (with the Enter key). The result occurs at either cursor depending on the command; some results only occur at the right cursor (Event or Data cursor).

System lists. Software generally uses system lists; ours are ranks built of exposed zzcells.

User data fields. The stuff of conventional databases, user data fields may be entered into the system as individual cells in particular locations, like the cells of spreadsheets or relational databases. However, they are not constrained to be arranged in rectangles. Our standard views already make them visible to some degree; we expect to develop particular views and operations for database regions of user data.

9.1 The deep zzvim

The extended or deep zzvim is still under selective definition. The following are still under study:

- what canonical dimensions to include
- cell ID conventions
- underlying dimensional ID conventions (to be renamed and nicknamed by the user)
- cell types
- anatomy of system cell area

The following general conventions of the zzvim are definite.

9.1.1 The Kblang keyboard language

Kblang (KeyBoard LANGuage) is a set of conventions for manipulating both windows at once -- stepping the cursor, rotating the dimensions of the specific view, creating and manipulating cells and connections. Designed for touch-typists, Kblang allows the user rapidly to move both cursors independently without "shifting focus", and to rotate either view quickly into whatever three dimensions the user wants to see at a given moment.

However, Kblang is not for beginners. It could be classed as a Power User interface, somewhat scary to users; and ZigZag applitudes need not expose Kblang to the simple user.

9.1.2 Everything, big and small, is in a zzcell

There is an accessible cell for every object or entity in the system. Cells represent objects, subobjects, individual records, and whole databases (or the equivalent).

9.1.3 All parameters are manifest in zzcells

Not only is there an accessible cell for every object or entity, but for every variable or parameter inside every entity. This is the opposite of the 'encapsulation' philosophy.

9.1.4 All programs are in cells

A program is not a string but a series of cells (progcells). Every significant macro or block is in a progcell. When executed, such a progcell takes in cells as operands and creates new cells as output. Pointers used by such programs are also cells.

Each progcell may be cloned, thus providing another executable cell that does the same thing -- but in a different place, with different operands.

9.1.5 System lists are ranks or ringranks

- The list of views is a ringrank.
- The list of dimensions is a ringrank.
- Lists of subdimensions associated with particular views and applitudes are separate ringranks with clones of the dimension names from the main dimension ringrank.

Later, windows and cursors, applitudes and data slices will all be ringranks.

9.1.6 Combinable slices

- Blocks of cells (zzslices) may be separately brought into RAM and merged, their cells connecting appropriately

9.1.7 Keystroke handler

- Raw keystrokes may be brought into the zzvim, so that the ZigZag environment may dynamically redefine all keys, as well as creating its own classes of keystrokes as software constructs (just as the "alt" keystrokes are a software construct).

9.2 The zzvim was interactive first, programmed second

Other virtual machines -- for example, the Lisp Machine in the 1980s, and more recently the Java virtual machine -- have been defined at first as string-based languages with keyboard interfaces, then usually combined with interaction afterward.

Unlike the Lisp and Java machines, ZigZag was built as a spatial interactive system first, not a system of string programming. The ZigZag operations, built first for keyboard exploration as a data system, became the basis for the other operations and for ZigZag programming methods; thus ZigZag began as an interactive system and is still being extended to its other functions.

This is the opposite of how Lisp and Java evolved, and that may be one of its particular strengths. These languages have no intrinsic relation to their graphics and interactive properties, and I believe this has disadvantaged them.

10 A new visualization infrastructure

10.1 Problems with "virtual reality"

Three-dimensional interactive visualization, often glamorized under the name "virtual reality", seems to many people the ultimate visualization, but it has a number of problems:

- Euclidean perspective shrinks objects to vanishing smallness rather quickly.
- Metaphorical space assignment is generally necessary.
- VR systems are generally built around a single space, therefore designing that space is crucial -- yet it is indeterminate, laden with conflicting objectives and possibilities, and highly arguable.
- Many aspects are not represented easily in three dimensions.

I believe VR really needs more coefficients -- and more dimensions -- to escape these problems. There is no difficulty finding a place for them in zzspace. We will consider this below.

10.2 Problems with multidimensional visualization

Those who have worked on multidimensional Euclidean-type visualization are challenged by the lack of standardized methods, the fact that most software stops at three dimensions, and the fact that there is no place to put the extra coordinates in conventional software.

(I say "Euclidean-type" to refer to continuous dimensions which may or may not be subject to conventional perspective visualization and transformation.)

By contrast, ZigZag allows the easy generalization of any Euclidean-type graphics to as many dimensions as you like. This is because there is always room for extra continuous coordinates. For each continuous coordinate, a holding cell is created to hold that coordinate.

10.3 What to call the spaces (shorthand)

"Virtual reality" and realistic graphics may be thought of as taking place in a three-dimensional space where each point has three real numbers as coordinates. The space itself is considered mathematically to be the cube of the real line. This is mathematically referred to as R-cubed (unfortunately, in this article it must be written as R**3 because of the character-set).

With ZigZag methods, we can easily expand R**3 to R**N (a large number of real numbers, as limited by space capacity) by the following method. A point is represented by a cell. Its coordinates are held by additional cells adjacent on respective zzdims; each of these zzdims is named for the coordinate being held.

Suppose you want to position objects not just in the standard set of three coordinates (x,y,z), which I will abbreviate xyz, but sometimes in another coordinate-set, pqr. Suppose further that you would like to position the objects in some new combination of the coordinates, such as xqr. Thus you need to save all six coordinates in association with every point being so presented.

To avoid problems with existing coordinates, it will be possible to create, for each point having an additional coordinate, a new holding cell for that coordinate. Such a holding cell will be connected to the cell representing that point on a new zzdim.

Example: suppose you have a point on the tip of a virtual actor's nose, which is one of many points required to produce the surface of the actor. Let us call it point #1375. In ordinary software, that point would be in a table; the entry for point #1375 would be associated with three entries xyz. In ZigZag, we represent the point as a cell. To that cell we connect not just coordinates xyz but coordinates pqr, as follows:

- its holding cell for its x-coordinate on d.holding-x
- its holding cell for its y-coordinate on d.holding-y
- its holding cell for its z-coordinate on d.holding-z
- its holding cell for its p-coordinate on d.holding-p
- its holding cell for its q-coordinate on d.holding-q
- its holding cell for its r-coordinate on d.holding-r

In one approach, ZigZag could be a storage engine allowing conventional 3D applications to be used with multidimensional data. We could store the current values for these Euclidean coordinates in these holding cells but take them out for each use and put them back after modification.

In a more efficient, fully ZigZagular method, we would simply keep the coordinates in those cells and perform all operations from the coordinates as stored in those cells.

10.4 The combined visualization space

N-dimensional real space (R**N) does not exhaust the graphical possibilities. We can also intermingle the N-space coordinates with positions derived from 3D views of zzcells. When we show zzviews in OpenGL or other 3D spaces, each cell gets a derived position in 3space, as already mentioned, where they happen be put by a particular 3-dimensional zzview. Such zzdim positions can be used to hang, or position, 3D objects (such as airplanes, synthespians, etc.) Thus we combine zzdim positions with 3space positions from any sources.

This gives us a very large space indeed, which may be expressed as (R**N) (Z**N). Meaning "as many real coordinates as you like times as many zzdims as you like". The ramifications of this are still being considered.

11 The ZigZag personal environment

All these considerations lead to many places and possibilities. Our central focus, however, is to create from all this a portable personal environment with the basic software that people need most generally. We intend to build a new, cross-platform personal-computer system. Not a whole operating system, mind you, but a system easily loaded on any popular machine. Thus we call it a co-operating system. It should behave the same whether running under Linux and running under the Macintosh and running under Windows. By eschewing the temptations of each of these environments and creating rather a hermetically clean internal world, we can start over.

This means that a user should be able to carry his or her entire operating environment from machine to machine on a thumbdrive, with all settings remaining the same.

For the simple beginner we hope to provide, in place of "applications," a pre-supplied system of LifeDimensions(tm). These will have the effect of applications, with simple integrated functions -- for example, managing personal contacts. It should be easy to use such functions as: add relative, add appointment, delete appointment, change appointment. All these are unified by the spatial structure of the system. The LifeDimension package is intended as a unified conglomerate structure so that making these changes is conceptually unified. Genealogy is not separate from address book or appointments; we intend that all these will be a unified package you do not have to leave.

Of course, the power beginner has the option of seeing all the internal cells and orthogonal connections, and building database-like structures of his or her own.

12 New document formats

At last we may speak of hypertext.

ZZ structure, in its principled generality and relative neutrality, opens the possibility of principled alternatives to today's document structures, which are in many ways deeply limiting. Instead of embedded angle-bracket markup ("thornyscript"), we seek cellular representation of structure, links and decoration in additional layers independent of the document content (Nelson 2004).

13 Conclusion

The computer world, and software design, have always been to some considerable extent about the design of imaginary constructs and their ramifications. Such concepts as "desktop" and "clipboard" are not reality, but imaginary constructs that become familiar and come to seem like reality. The same holds for textfiles and directories, imaginary constructs of an earlier generation. There is no right or wrong about imaginary software constructs, save for such criteria as usability (the pragmatic aspect), comprehensibility (the cognitive aspect) and aesthetics (the art aspect).

Consider again the spreadsheet. If we begin a new spreadsheet, but commit one column and one row, we have divided the area irrevocably into quarters. We run out of places to put things, not because we run out of space in the computer, but because we run out of space in the construct.

It's very like game design. Note that Conway's Game of Life and the familiar game of chess appear more playable and enjoyable than their variations -- more payout for the user, in other words. That is essentially the claim I make for the system I have described -- a wholly new computer system for everything, departing from all conventional methods, which enthusiasts believe makes everything far simpler -- and like these games, it must be tried to be understood, because mental analysis does not tell you much about playability.

What we need is better constructs, and that means the explicit study and exploration of constructs in all their orneriness and arbitrariness, and teaching of construct design, and the design of their ramifications, as an independent subject -- partly so students will not mistake constructs for reality.

Computer science, too, is often largely about imaginary constructs and their exploration. Those selfsame directories and textfiles have become sacrosanct, regarded as fundamental; the same is happening now with "Web pages", perhaps the most artificial and arbitrary construct that has ever been taught. But the field is quick to adopt constructs and apply them without question.

In this respect computer science can be more like metaphysics than physics: assigning unprovable ideas and models to all parts of the universe, without technical justification.

The research done by computer scientists generally assumes such unprovable and unjustifiable mechanisms as hierarchical structures (embedded in almost every application and research model), hierarchical directories, lump file documents, one-way links, and the Web -- templates and representations which strongly and invisibly affect and bias what is done. Whereas not enough thought has been given to the arbitrariness, freedom and possibilities of construct design.

When you select a construct you select ramifications; reworking the construct to improve the ramifications is extremely hard. This is where philosophy, game design, screen writing and science fiction meet.

Unlike these traditional structures of the computer world, which expanded outward from simple ideas, zzstructure and ZigZag have been designed with an emphasis on construct ramifications. Our cycle has been a continuous loop of expanding premises into ramifications, then redesigning the premises. Thousands of decisions went into its present design, but they have disappeared into the resulting simple structure. Simplicity is not cheap, and simple design is very difficult.

Acknowledgements

The author would like to thank Adam Moore (University of Nottingham) for transcriptions and contributions to this manuscript; Helen Ashman and members of the Web Technologies Group, University of Nottingham, for hospitality and many interesting conversations; and Marlene Mallicoat for substantial support over more than a decade. During some of the preparation of this manuscript, the author was also Leverhulme Visiting Professor at the School of Computer Science and Information Technology, University of Nottingham, UK.

References

Bush, V. (1945) "As We May Think". *The Atlantic Monthly*, 176(1), 101-108 http://www.theatlantic.com/unbound/flashbks/computer/bushf.htm

Carmody, S. *et al*. (1969) "A Hypertext Editing System for the 360". In *Pertinent Concepts in Computer Graphics*, edited by M. Faiman and J. Nievergelt (Urbana: U. Illinois Press)

Iverson, K. (1962) *A Programming Language* (New York: Wiley)

Kaijanaho, Antti-Juhani and Fallenstein, Benjamin (2001) "Totally Different Structural Programming - Programming Languages

in ZigZag". *First International ZigZag Conference*, part of ACM Hypertext Conference 2001, Aarhus, Denmark, August http://www.mit.jyu.fi/antkaij/plinzz.html

Nelson, T. (1965) "A file structure for the complex, the changing and the indeterminate". *Proceedings of the ACM 20th National Conference*, Cleveland, OH, pp. 84-100

Nelson, T. (1981) *Literary Machines* (Mindful Press)

Nelson, T. (1987) *Literary Machines 87.1* (Mindful Press)

Nelson, T. (2003) "Structure, tradition and possibility". *Proceedings of the 14th ACM Conference on Hypertext and Hypermedia*, Nottingham, UK http://www.ht03.org/keynote-ted.html

Nelson, T. (2004) "DeepLit: Multilayered Nonhierarchical Representations for Content Tracking, Side-by-Side Intercomparison, Variant Structuring and Re-Use of Parallel Documents". In preparation

Notes

Table 1: History of ZigZag implementations

Name	Developers	Date	Platform(s)	Notes
?	Mark Miller & Terry Stanley	ca. 1984	Forth	Memory-management module designed as part of larger zzvim that was never built
Azz	Andrew Pam, Operation ZigZag	1997	Perl (for Linux)	First full zzvim implementation; row and column views only
Azz boot floppy	Andrew Pam, Operation ZigZag	1998	Perl + minimal Linux on floppy	Runs stand-alone on standard Wintel machine (Linux not accessible) Not upgraded to later Azz features, e.g. d.contain.
Azz on Windows	Andrew Pam, Operation ZigZag	1998	Perl under Windows	Azz running under Windows95, starting from shortcut
Gzz076	Tuomas J. Lukka and associates, U. Jyväskylä & elsewhere	2000	Java	Offers several excellent views, achieving unifreshment with 2D splines
Lzz	Les Carr, U. Southampton	2001	HTML/XML/XSLT	Works client-side (with varying results in different browsers)
Gzz	Tuomas J. Lukka and associates, U. Jyväskylä & elsewhere	2001	Java (1.2 or 1.3; subsequent Javas spoil performance)	Wonderful variety of views; some excellent extensions; outputs and accepts zzstructure via XML
Ezz	Commission work by Ejobshop Inc., programmed by Mikhail Seliverstov	2003	Java; runs on Mac and Windows from shortcuts	Re-implements Azz (row and column views only); puts out and accepts XML (Gzz format)
Zzz (current experimental system)	Jeremy Alan Smith	2003	C with Python. Separate binaries for Windows, Linux, Mac OSX	Routine-by-routine conversion of Azz, extensible in Python. Some 3D views in OpenGL.
zzphone	James Goulding, Operation ZigZag	In progress	Certain telephones under Symbian operating system, esp. Sony-Ericsson P600	Unification of most functions under ZigZag; thanks to kind assistance of Symbian UK

Besides the implementations mentioned above, several partial and related software projects have been carried out by Grant Blaha, Jan Galkowski and others.

Appendix 1. Some counterintuitive facts and precepts about zzstructure

zzstructure, and the spaces that it generates, have many curious and seemingly paradoxical aspects. These paradoxes, however, simply register our cognitive consternation over unexpected ramifications of data structure and space in this system. They violate many expectations from the paper and spreadsheet world, and indeed from the everyday world of physical objects as well.

A1.1 Counterintuitive fact: all links are independent (different from spreadsheet)

A cell's connections on a given dimension are not necessarily similar to its neighbor's connections (see Figure 3).

These are important differences from the spreadsheet, where the connections of one row are repeated identically in the next row, and the connections of one column are repeated identically in the next column.

A1.2 Counterintuitive fact: individual cells may represent small units, large units, regions and conglomerates

An individual zzcell may be simply a small data field, such as a first name or the number in an address. But a cell may also represent a large file. It may also represent a structure of other cells, by some convention. The breadth of this scale variation may confuse people. (Organizing it spatially is the design problem.)

A1.3 Counterintuitive fact: it is easy to add dimensions

It is easy always to add another dimension. We call this "headroom".

A1.4 Counterintuitive precept: more dimensions are better

It is counterintuitive for most people, but adding dimensions appears to make most operations much simpler to implement. The adding of dimensions proves a surprisingly simple solution to many issues. Many operations and "applications" are simply reduced to a few dimensions (see genealogy example).

A1.5 Counterintuitive fact: new structure does not change the old

Because we can add new dimensions, we don't have to change existing structures to add new aspects or features.

This is quite different from most conventional methods. For example, in a conventional database, adding new fields or tables requires changing the table definitions and the programs around them.

A1.6 Counterintuitive precept: no action at a distance (recommended)

Conventional computer systems arrange action at a distance. Software frequently refers to elements by references from various places -- for instance:

- Programs: conventional programs constantly refer to variables throughout the data and program space.
- Tables: conventional tables constantly refer to variables throughout the data and program space.

These often use strings as a mechanism for finding files, selecting subprograms, etc.

Instead we use connected cells wherever possible. It is easy to create a direct connection to any point in the system. While it is possible to refer to a cell from elsewhere, our default method is to refer to a cell by touching it -- with a pointer cell which connects to it in some dimension.

This has two special advantages:

- it is robust;
- it is visible, and can be watched in its functioning. This can help in debugging.