

Carving Text at Its Joints

A New Perspective on Writing and Computers

Kevin Graaf

Onward! Essays, 17 October 2025

Outline of talk

- 1 Introduction
- 2 Three Myths Computer Scientists Believe About Text
- 3 Unicode: A Critique
- 4 Prolegomena to Any Future Text Encoding
- 5 Fonts as Programs
- 6 Programmable Typography: The Dubsar Approach
- 7 Onward!

Table of Contents

- 1 Introduction
- 2 Three Myths Computer Scientists Believe About Text
- 3 Unicode: A Critique
- 4 Prolegomena to Any Future Text Encoding
- 5 Fonts as Programs
- 6 Programmable Typography: The Dubsar Approach
- 7 Onward!

100% Human-made

Everything in my paper and presentation is 100% human-made.
No “AI.”

Introduction

- Computer users rightfully expect to be able to input, view, transform, and edit text in their own language and in their own writing system
- It's 2025 (almost 2026!), software &c *must* reflect our multilingual, multiscriptal world!
- English-only, ASCII-only, fixed-width fonts won't cut it anymore!

Introduction

- Computer users rightfully expect to be able to input, view, transform, and edit text in their own language and in their own writing system
- It's 2025 (almost 2026!), software *&c must* reflect our multilingual, multiscriptal world!
- English-only, ASCII-only, fixed-width fonts won't cut it anymore!
- The problem? Human writing is *hard*
- We've come a long way over the past three decades, but much work remains
- Modern text stacks can almost get us there, but...
- Major open problems remain in encoding, interactively editing, programmatically transforming, and displaying text in many of the world's writing systems

Introduction (continued)

- The extreme complexity and difficulty of software internationalization (esp. w.r.t. writing) represents a serious economic disincentivization
- Minority language speaker communities are often left without technological support
- This can further contribute to their marginalization
- As a matter of linguistic justice, there is a particular urgency within our profession to develop better methodologies with respect to programmatic treatment of human languages, writing, numeral systems, and more

To Justify the Ways of Linguists to Computer Scientists

- I argue that the way we have treated text heretofore is wrong
- We must historicize the entire basis of how computers and software have come to handle text
- The apparent complexity of multilingual and multiscriptal computer text is largely *extrinsic*
- The result of bad/no abstractions over writing systems *as a class*
- The particulars of *every* writing system intrude into every part of *every* program
- Linguistic correctness appears locked in an eternal struggle against simplicity

To Justify the Ways of Linguists to Computer Scientists

- I argue that the way we have treated text heretofore is wrong
- We must historicize the entire basis of how computers and software have come to handle text
- The apparent complexity of multilingual and multiscriptal computer text is largely *extrinsic*
- The result of bad/no abstractions over writing systems *as a class*
- The particulars of *every* writing system intrude into every part of *every* program
- Linguistic correctness appears locked in an eternal struggle against simplicity
- I come bearing an offering of peace: tractable and correct text handling is possible, if we are willing to radically rethink what computer text is
- My goal: “to justify the ways of linguists to computer scientists”

A Proper Account of Text

[P]erceiving and bringing together in one idea the scattered particulars, that one may make clear by definition the particular thing which he wishes to explain...[D]ividing things again by classes, where the natural joints are, and not trying to break any part, after the manner of a bad carver.

—Phaedrus 265d-e



What we need is a proper, *philosophical* account of what computer text is!

- ① An alternative style of text encoding which directly and uniformly represents the inherent structure in text
- ② A model of programmable digital typography which is both more general and far less complex than standard, outline-based font tech

Three Aphorisms

I am here guided by three aphorisms:

- 1 There is no such thing as plain text
- 2 Fonts are programs
- 3 Writing is an interface; writing systems are its implementations

Some Definitions

- Grapheme — the smallest semantically indivisible unit of writing
- Logogram — symbol which denotes a word or root instead of a sound or syllable
- Quadrat — blocks some hieroglyphic scripts are visually organized into
- Diacritic — something which modifies a base glyph

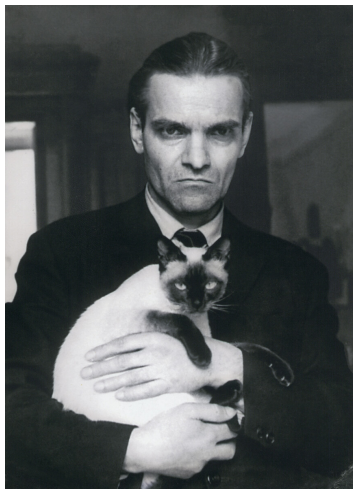
A Brief Introduction to Maya

- First developed in the 3rd Century BC
- One of only *four* known independent inventions of writing!
- Used to write the Classical Maya language
- Suppressed by Spanish colonial authorities in the 16th Century and thereafter forgotten



A Brief Introduction to Maya

- First developed in the 3rd Century BC
- One of only *four* known independent inventions of writing!
- Used to write the Classical Maya language
- Suppressed by Spanish colonial authorities in the 16th Century and thereafter forgotten
- Deciphered by Soviet linguist Yuri Knorozov in the mid-20th Century
- Still not fully understood!



An Example of Maya: Lintel 14, Yaxchilan



Another Example of Maya: The Dresden Codex



Why Care About a Dead Writing System?

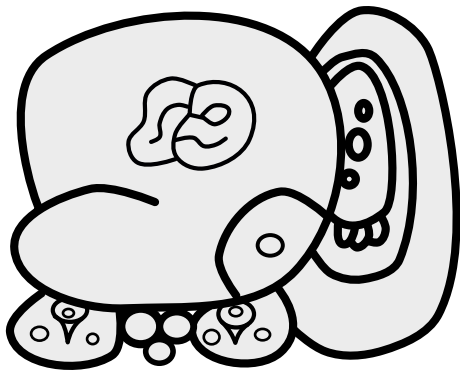
- Maya is a particular interest of *mine*, but why should *you* care?
- Well...

Why Care About a Dead Writing System?

- Maya is a particular interest of *mine*, but why should *you* care?
- Well...
 - ① Periodic interest in revival for contemporary Mayan languages
 - ② Maya is *complex*, good stress test for software
 - ③ Maya is *different*, challenges faulty assumptions about text
 - ④ It's just inherently interesting!

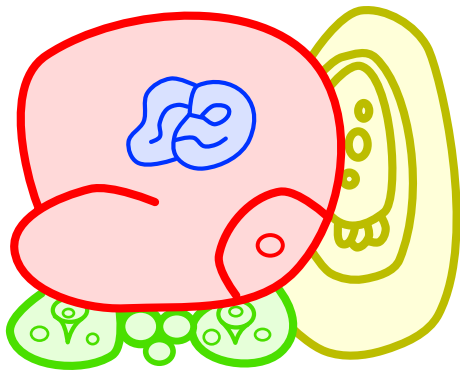
Anatomy of Maya Text

- Let's look at some Maya...
- This says "he/she sat"
- But there's more here than meets the eye!
- Maya text is organized into blocks called quadrats (same term used in Egyptology) but quadrats are NOT graphemes!
- How many components do you think this quadrat has?



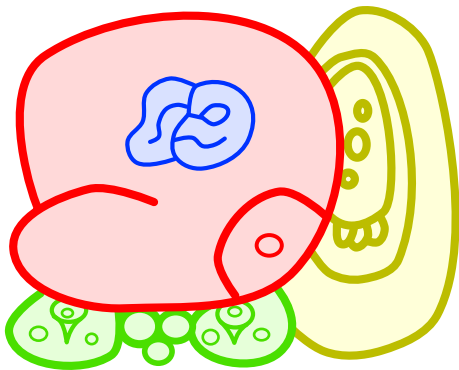
Anatomy of Maya Text

- Let's look at some Maya...
- This says "he/she sat"
- But there's more here than meets the eye!
- Maya text is organized into blocks called quadrats (same term used in Egyptology) but quadrats are NOT graphemes!
- How many components do you think this quadrat has?
- 4!



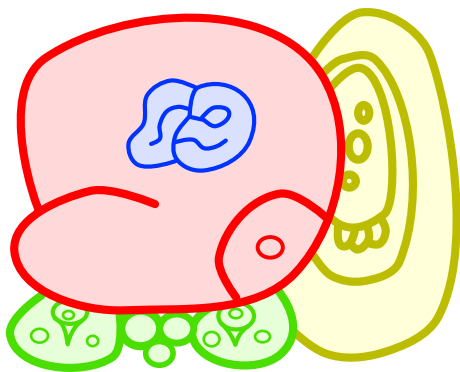
Anatomy of a Maya text (continued)

- Transliteration: **CHUM**[mu]-la-ja
- Pronunciation: *chumlaj*
(N.B. ⟨j⟩ as in Spanish)
- Written Maya has a *complex* relationship with vocalization
- Some glyphs function as *logograms*, they stand for roots (as with **CHUM**—Mayanists transliterate logograms with uppercase)



Anatomy of a Maya text (continued)

- **mu** is a *phonetic complement*—gives a pronunciation hint
- Maya is mostly *syllabic*, Knorozov's great insight
- However, Maya can't easily write closed syllables
- The solution? Fake it!
- This kind of workaround is common cross-linguistically, cf. Linear B



Anatomy of a Maya text (continued)

- Maya layout is *ridiculously* complex
- **mu** is *infix*ed, it's a shrunk version of the normal glyph, overlaying another. Very Common!
- Maya quadrats can have as many as *six* conceptually independent components!
- Again, these aren't diacritics (although those also exist)
- Quadrats frequently include numbers, esp. with dates (we'll see examples)

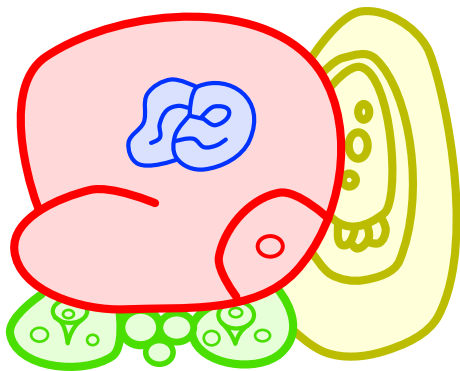


Table of Contents

- 1 Introduction
- 2 Three Myths Computer Scientists Believe About Text
- 3 Unicode: A Critique
- 4 Prolegomena to Any Future Text Encoding
- 5 Fonts as Programs
- 6 Programmable Typography: The Dubsar Approach
- 7 Onward!

Myth 1: Computer Text is Print Transposed to a Screen

- Computer text must be understood as its own modality of writing and not carved, handwritten, etched, or printed text merely transposed onto an electronic display
- Computer text encompasses modes of interaction that have no clear analogue in antecedent forms of writing
- It is not generally possible to reflow, move, delete, or exactly duplicate handwritten text
- There is no way to “backspace” hot-metal typesetting or perform arbitrary insertions on a typewriter

Towards Maya Computerization

- How should backspace work?
- Should the cursor travel *inside* a quadrat?
- Seems that we need sub-quadrat editing!
- Imagine if English could only be edited at the granularity of a word or phrase!
- There are *no* obvious, tidy answers here—we are living in a *Latin world*
- Unfortunately, we can't just ask an ancient Mayan
- Maya also had no print tradition, we can't consult that either

The Bottom Line

- There may be no single fact of the matter about how common human-computer interaction primitives should work that can cover all possible or at least extant writing systems in a consistent and unsurprising manner
- More familiar examples abound
- Backspace deletes a letter, right?
- For European languages, perhaps!
- Some orthographies (e.g. Vietnamese) use multiple, stacking diacritics, should backspace delete a *component* instead?

Myth 2: There Are No New Writing Systems

- In fact, dozens of new writing systems have emerged over the past century
- Particularly common in West Africa
- Every new system brings new challenges to meet

GWY

Cherokee, 1820s

𞤎𞤍𞤏𞤢𞤏𞤢𞤏𞤢𞤏𞤢𞤏𞤢

Vai, 1830s

𞤎𞤍𞤏𞤢𞤏𞤢𞤏𞤢𞤏𞤢𞤏𞤢

N'Ko, 1949

𞤎𞤍𞤏𞤢𞤏𞤢𞤏𞤢𞤏𞤢𞤏𞤢
𞤎𞤍𞤏𞤢𞤏𞤢𞤏𞤢𞤏𞤢𞤏𞤢
𞤎𞤍𞤏𞤢𞤏𞤢𞤏𞤢𞤏𞤢𞤏𞤢
𞤎𞤍𞤏𞤢𞤏𞤢𞤏𞤢𞤏𞤢𞤏𞤢

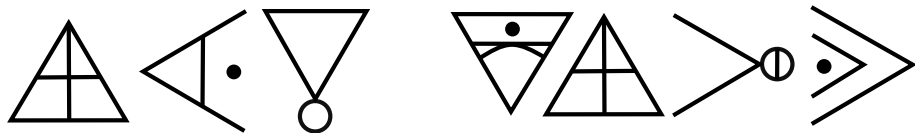
ADLaM, 1989

𞤎𞤍𞤏𞤢𞤏𞤢𞤏𞤢𞤏𞤢𞤏𞤢

Ditema tsa Dinoko, 2010-2015

Ditema tsa Dinoko

- I wanted to draw special attention to Ditema tsa Dinoko
- An attempt to provide a common orthography for many related languages of Southern Africa
- An *emerging* writing system



Myth 3: Writing Is a Linear Sequence of Symbols

[Unicode] inherently assumes that the geometry of 1-dimensional graphetic sequences—as long as there is no 2- dimensional higher structure such as a list or a table—carries no syntactic or semantic information...[but]...[t]here are cases where human creativity has transcended this model...

—Yannis Haralambous and Martin Dürst,
“Unicode from a linguistic point of view”

Let's take a look at some examples...

Nastaliq

- Variant of the Arabic script
- Developed by Persian scribes in the 14th century
- Used to write Urdu—approx. 230 million speakers (roughly as many as German, Italian, and Dutch combined!)—in addition to (classical) Persian and others
- Main conceit: words are written at a diagonal and vertically overlap (*exceptionally* complex layout!)
- From a Western typographical perspective, as many “ligatures” as there are Urdu, &c words!



A ruba'i (or quatrain) of Omar Khayyam

- Everyone knows Hebrew and Arabic are written right-to-left
- Back-borrowed European Arabic numerals (!) are still typically written left-to-right
- As is embedded Latin text
- Still “linear” in a manner of speaking

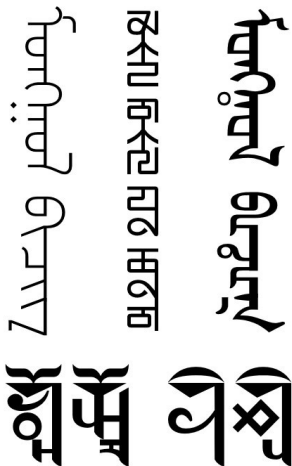
Vertical Text

- Some writing systems are vertical
- But “vertically” is said in many ways...
- Columns written left-to-right or right-to-left?
- Are glyphs rotated? (See Chinese vs. Manchu)?
- Some systems can be *optionally* vertical (like CJK), others mandatory (like Mongolian)
- How does vertical text interact with embedded horizontal text?



Mongolian

- The traditional Mongolian alphabet is written *only* vertically
- Mongolian government plans to reintroduce this script (currently uses Soviet-era Cyrillic orthography)
- Software is not even close to ready!
- Better and more general methods for handling vertical text are needed

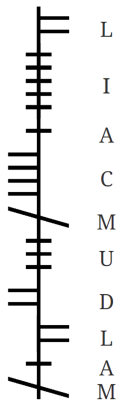


How Many Possible Writing Directions Are There?

- We've seen left-to-right, right-to-left, and top-to-bottom writing
- We seem to be missing one direction...
- Surely no one has written bottom-to-top?

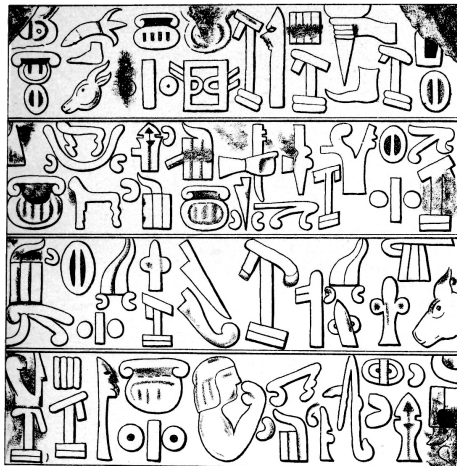
How Many Possible Writing Directions Are There?

- We've seen left-to-right, right-to-left, and top-to-bottom writing
- We seem to be missing one direction...
- Surely no one has written bottom-to-top?
- In fact, yes!
- The Ogham script was first used to write an early form of Irish, it is written/read bottom-to-top.
- This inscription records a person's name, "Mal Dumcail"



Anatolian Hieroglyphs

- Used to write Luwian, an ancient Anatolian language



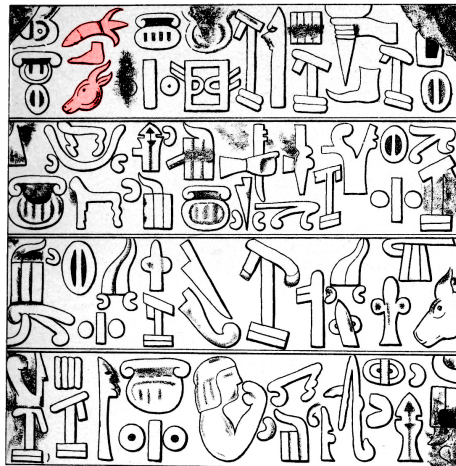
Anatolian Hieroglyphs

- Used to write Luwian, an ancient Anatolian language
- Typically written in *boustrophedon* order, from the Greek “as the ox plows”
- Alternating left-to-right, right-to-left lines
- Animal and human shaped glyphs face the start of the line



Anatolian Hieroglyphs

- Used to write Luwian, an ancient Anatolian language
- Typically written in *boustrophedon* order, from the Greek “as the ox plows”
- Alternating left-to-right, right-to-left lines
- Animal and human shaped glyphs face the start of the line
- However, *within* a line, glyphs may vertically stack



How is Maya text read?

Horizontally or Vertically?



Yes.



Maya is typically (though not always) read in zig-zag columns of two. If there are an odd number of columns, the last is vertical.

Now consider that each Maya quadrat *also* has an *internal* reading order which varies depending on the quadrat!



Egyptian

- Obviously, needs no introduction
- LTR or RTL, where human/animal glyphs face start of line



Egyptian

- Obviously, needs no introduction
- LTR or RTL, where human/animal glyphs face start of line
- Like, Maya organized into quadrats
- This is a fundamental part of the script, not higher layout
- Common Egyptological practice to transcribe Hieratic texts into “full” hieroglyphs, Egyptian typography is thus a *real* need!



- Technical notations for math, music, &c are at the boundaries of “text”
- Obviously, completely non-linear
- Are things like this even within the remit of “text” handling, encoding?
- Personally, I'm not sure

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

PRELUDE
Op. 28, No. 7
Frederic Chopin

Andantino
Piano *p* *dolce*
con Pedale

mp *mp* *riu. c. dim.* *pp*

In Summary

- Even writing direction is far more complex than we might imagine
- Seemingly *every* possibility has been instantiated somewhere
- When we consider multi-directional text, the space of options becomes *huge*!
- The most complex scripts (e.g. Nastaliq, Maya, Egyptian, Anatolian) have a *global* and *local* writing direction (almost fractal in character...)
- Physical order doesn't always neatly correspond to “logical” reading order

Table of Contents

- 1 Introduction
- 2 Three Myths Computer Scientists Believe About Text
- 3 Unicode: A Critique**
- 4 Prolegomena to Any Future Text Encoding
- 5 Fonts as Programs
- 6 Programmable Typography: The Dubsar Approach
- 7 Onward!

Unicode: A Critique

- *Unicode* really is *universal*
- This was hard won! Decades of work and advocacy
- A monumental achievement to be sure, but...
- ...No technology should be sacrosanct!
- I want to offer some critiques of Unicode

Unicode: A Critique

- *Unicode* really is *universal*
- This was hard won! Decades of work and advocacy
- A monumental achievement to be sure, but...
- ...No technology should be sacrosanct!
- I want to offer some critiques of Unicode
- Now, excuse me while I duck...



The Incoherence of Plain Text

- Unicode is about “plain text”
- “Formatting” is explicitly outside its purview
- But what exactly *is* “plain text?”
- Let’s ask the Unicode specification!

Plain text is the underlying content stream to which formatting can be applied...[It] must contain enough information to permit the text to be rendered legibly, and nothing more

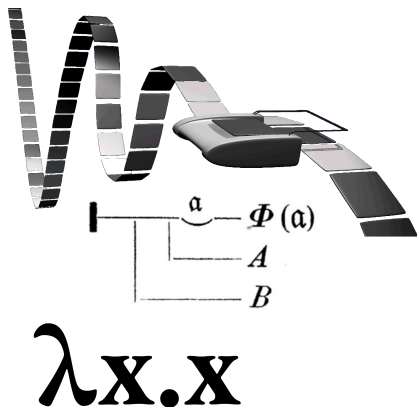
- I want to deconstruct this category through a genealogical critique

Whence Computers?

- Computers have a *dual* ancestry!

Whence Computers?

- Computers have a *dual* ancestry!
- They are the result of the “high-brow” work of Lovelace, Frege, Peirce, Gödel, Turing, Church, von Neumann, &c



Whence Computers?

- Computers have a *dual* ancestry!
- They are the result of the “high-brow” work of Lovelace, Frege, Peirce, Gödel, Turing, Church, von Neumann, &c
- *But*, they are *also* the descendants of the “low-brow” world of typewriters, teletypes, telegraphs, Baudot codes, and so forth *qua physical artifact*
- These machines/encodings placed *severe* restrictions on text
- But importantly, these restrictions are historical *contingencies*



A Fable

I'd like to share a sort of fable with you

This is a story from the history of English

A Taste of Middle English

Piers Plowman, written *circa* 1370–86 by William Langland.

Middle English

In a somer seson whan softe was þe sonne
I shoop me in-to shroudes as I a sheep weere
In habite as an heremite vnholy of werkes
Wente wide in þis world wondres to here

Modern English

In a summer season when the sun was soft
I clothed myself in shrouds as if I were a sheep [*or shepherd*]
In a habit like a hermit unholy of works
I went wide across this world wonders to hear

A Taste of Middle English

Piers Plowman, written *circa* 1370–86 by William Langland.
Notice anything?

Middle English

In a somer seson whan softe was þe sonne
I shoop me in-to shroudes as I a sheep weere
In habite as an heremite vnholy of werkes
Wente wide in þis world wondres to here

Modern English

In a summer season when the sun was soft
I clothed myself in shrouds as if I were a sheep [*or shepherd*]
In a habit like a hermit unholy of works
I went wide across this world wonders to hear

A Taste of Middle English

Piers Plowman, written *circa* 1370–86 by William Langland.

What's this strange letter!?

Middle English

In a somer seson whan softe was **þe** sonne
I shoop me in-to shroudes as I a sheep weere
In habite as an heremite vnholý of werkes
Wente wide in **þis** world wondres to here

Modern English

In a summer season when the sun was soft
I clothed myself in shrouds as if I were a sheep [*or shepherd*]
In a habit like a hermit unholy of works
I went wide across this world wonders to hear

The Letter Thorn — þ

- This is a letter called *thorn*
- Adapted from the Runic Alphabet (in its various forms)
- Denotes dental fricatives (the “th” sound) (we’ll ignore Ð/ð for now)
- Still used in Modern Icelandic!

An example of Anglo-Saxon Runes

The Franks Casket, currently at the British Museum, London
Old, not Middle, English!



An example of Anglo-Saxon Runes

The Franks Casket, currently at the British Museum, London
Old, not Middle, English! notice that the bottom is written right-to-left!



So, what happened?

- P clearly isn't used to write English anymore
- It seems very useful! (e.g. *hothead* isn't *hopead*!)
- Any guesses?

The Printing Press

- (Western) printing began in Continental Europe
- Continental European languages didn't/don't use Þ
- Thus, early type imported to England didn't have it either
- Þ fell out of use due to *technological limitations*.



And by the way...

The y in “ye” (as in “Ye Olde Shoppe”) was a 15/16th century workaround for printing þ. It’s not pronounced yee!



All Urdu software is broken because the underlying data is broken
—Zeerak Ahmed

- Nastaliq *really* needs fine grained layout control to be displayed correctly
- Unicode shunts this into “formatting”
- The result? Urdu speakers abuse spaces/control characters
- Or worse, give up and transliterate into Latin
- Obviously, monospaced/unformatted English doesn't look *great*
- But this is more than just aesthetics, it's about *correctness*

Han Unification

- The so-called “Han Unification” will be familiar to many of you, but I’d be remiss if I didn’t mention it
- Unicode encodes characters from the CJKV (Chinese, Japanese, Korean, Vietnamese) tradition with the same codepoint where they share a common origin
- Sometimes this works out fine
- Other times, originally related characters have diverged significantly
- Multi-lingual East Asian plain text is hard!
- `ιμαgine trying to read EngLish if LatIn, GrEEk, and CyriLLic wErE "unified"`
- Alternative encodings like TRON did not unify CJKV characters
- Unification was probably the right choice, but...
- Not if “formatting” is treated as a bonus!

We're computer scientists, not linguists, so let's take a look at a more familiar kind of language!

An example of Algol 60 (from the original report)

```
procedure euler (fct, sum, eps, tim) ; value eps, tim ;
integer tim ; real procedure fct ; real sum, eps ;
comment euler computes the sum of fct(i) for i from zero up to
infinity by means of a suitably refined euler transformation. The
summation is stopped as soon as tim times in succession the absolute
value of the terms of the transformed series are found to be
less than eps. Hence, one should provide a function fct with one
integer argument, an upper bound eps, and an integer tim. The
output is the sum sum. euler is particularly efficient in the case
of a slowly convergent or divergent alternating series. ;
begin integer i, k, n, t ; array m[0:15] ; real mn, mp, ds ;
i := n := t := 0 ; m[0] := fct(0) ; sum := m[0]/2 ;
nextterm: i := i+1 ; mn := fct(i) ;
    for k := 0 step 1 until n do
        begin mp := (mn+m[k])/2 ; m[k] := mn ;
            mn := mp end means ;
    if (abs(mn)<abs(m[n]))^(n<15) then
        begin ds := mn/2 ; n := n+1 ; m[n] :=
            mn end accept
    else ds := mn ;
    sum := sum + ds ;
    if abs(ds)<eps then t := t+1 else t := 0 ;
    if t<tim then go to nextterm
end euler
```

Algol 60 (continued)

- Notice the **bolding**?

Algol 60 (continued)

- Notice the bolding?
- This is *not* syntax highlighting
- It's a *part* of the syntax itself!

Algol 60 (continued)

- Notice the bolding?
- This is *not* syntax highlighting
- It's a *part* of the syntax itself!
- Algol 60's syntax relied on boldness to distinguish keywords from identifiers
- Implementations used workarounds to delimit keywords (such as quotes), called “stropping”
- Algol 60 was designed first as a *print* notation for algorithms
- Its creators weren't constrained by arbitrary typographic limitations!

APL's "Hieroglyphic" Character Set

- APL was designed around a novel character set
- Widely derided (unfairly!) as unreadable "hieroglyphics"
- Hard to support in the early 1960s (even now!)
- Its successor J replaced APL's beautiful symbols with a rather clunky ASCII-based syntax for this reason (notice the unbalanced brackets)

APL

```
life ← {↑1 ω∇.∧3 4=+/,~1 0 1◊.Θ~1 0 1◊.ϕ<ω}
```

J

```
life =: (3 3 (+/ e. 3+0,4&{)@,;._3 ])@(0,0,~0,.0,.~])
```

There's a lesson here

- For 15th century printers, P was not “plain text”
- After all, English can clearly be read without it

There's a lesson here

- For 15th century printers, P was not “plain text”
- After all, English can clearly be read without it
- By contrast, for the Algol 60 committee, bolding *was* “plain text”
- Algol 60 *doesn't* work without it

There's a lesson here

- For 15th century printers, P was not “plain text”
- After all, English can clearly be read without it
- By contrast, for the Algol 60 committee, bolding *was* “plain text”
- Algol 60 *doesn't* work without it
- Many things that Unicode deems “formatting” are needed for *correct* and not just “pretty” display
- Conclusion: “plain text” is *not* a natural, trans-historical, cross-linguistic category
- It is *always* indexed to a time and place
- History is replete with examples of technologies *limiting* and *flattening* the expressivity of writing

The Long Shadow of Legacy Encodings

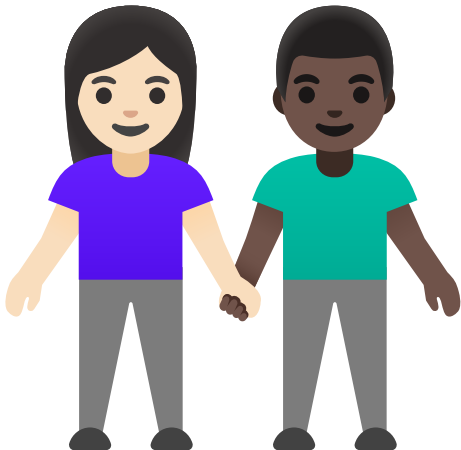
- An explicit goal of Unicode was round-trip compatibility with legacy character sets and encodings, essentially the “Cartesian product” of previous encodings
- A worthy goal perhaps, but problematic...
- The source of much of Unicode’s complexity
- In particular, the ultimate reason for its multiple normal forms
- Unicode is now older than many/most of its predecessors were when it (Unicode) was new
- Nobody remembers or cares about e.g. Classic Mac OS codepages, must we pay the price of compatibility forever?
- This isn’t free! Security issues (w.r.t. string equivalence), bugs, immense programmer effort

The Necessity of Structure

- Unicode is nominally “flat” and unstructured
- But as we shall see, it contains many forms of structure
- To return to our Platonic theme...
- is this Unicode’s original (ig)noble lie?
- The perceived simplicity of unstructured text is deceptive
- Merely rearranges where the complexity is located.
- Simpler yes, but simpler for whom?
- Can we cut this Gordian Knot and simply accept structure?

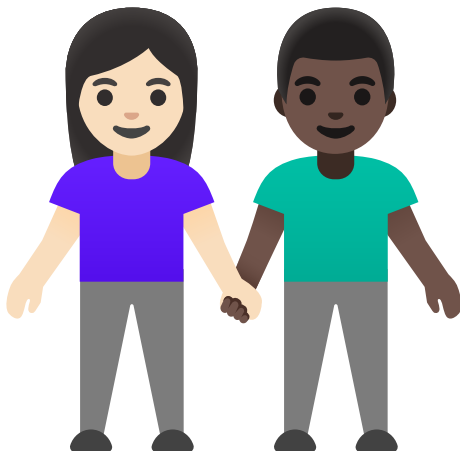
Emoji

Many emoji are built from other emoji with “ZWJ” sequences



Emoji

Many emoji are built from other emoji with “ZWJ” sequences



- 1  *Woman*
- 2  *Skin Color Modifier*
- 3  *Zero Width Joiner*
- 4  *Handshake*
- 5  *Zero Width Joiner*
- 6  *Man*
- 7  *Skin Color Modifier*

Egyptian Layout Control

- Unicode recently-ish added layout control characters for Egyptian—previously only had rather useless individual hieroglyphs
- Somewhat like a binary version of the ASCII-based *Manuel de Codage* (MdC) Egyptian encoding (but more expressive)

• E.g. <-i-mn:n-m-HAt:t->



- Fully-bracketed design
- Requires a context-free grammar to describe!

	1343	1344	1345
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			
A			
B			
C			
D			
E			
F			

The Necessity of Structure (continued)

- Emoji and Egyptian are both structured, but in very different ways
- Unicode implementations must parse/validate both
- Many other examples of the same problem
- Wouldn't a single mechanism which covers both be better?
- Why not XML? It's an option, but...
 - ① many XML-based markup languages overlap in functionality already present in/proposed for Unicode—very awkward semantic mismatch
 - ② Different markup standards can't easily coordinate or compose
 - ③ XML isn't *text* as such, can't use XML as an identifier
- In the case of BiDi text, structure is *implicit*—reconstructed (lossily!) and discarded at every application boundary
- Again, why not solve this *once*?

Table of Contents

- 1 Introduction
- 2 Three Myths Computer Scientists Believe About Text
- 3 Unicode: A Critique
- 4 Prolegomena to Any Future Text Encoding**
- 5 Fonts as Programs
- 6 Programmable Typography: The Dubsar Approach
- 7 Onward!

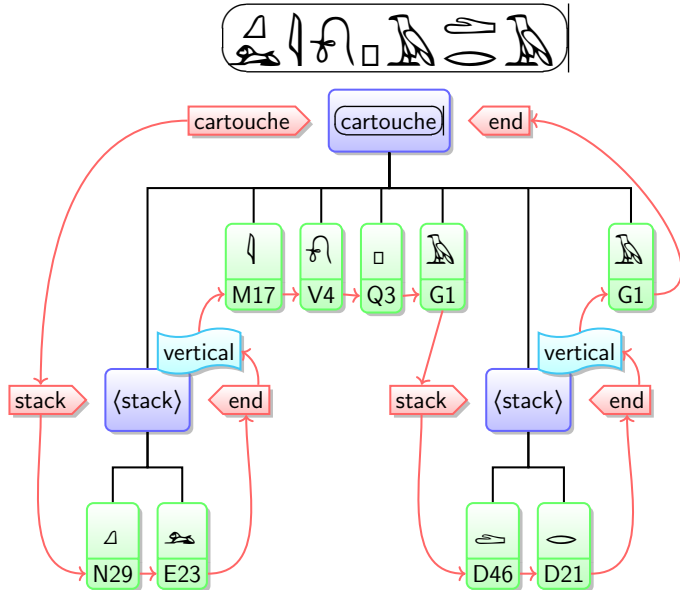
Prolegomena to Any Future Text Encoding

- I want to sketch a new kind of text encoding—a scheme I call Telecode
- If we were to do Unicode over again, how would we?
- Difficult to untangle different spheres of interest. These categories can't always be neatly separated
- Linguists, paleographers, programmers, everyday users all want different things
- Sometimes you want the text in itself, other times you need metadata to interpret the text (cf. Han unification)
- So don't choose! Have both!
- This is a *telescoping* encoding—a single mechanism which represents structure at all levels of organization

Structured Representation

- Telecode text is a sequence of ordered, labeled n -ary trees
- It has three kinds of entities
 - 1 Base Characters: letters of the Latin alphabet, numerals, punctuation, Egyptian Hieroglyphs, &c (like regular, non-combining characters in Unicode)
 - 2 Tags: introduce structures, e.g. Egyptian cartouches, Maya quadrats, multi-character diacritics (â€)
 - 3 Attributes: modify base characters and structures
- Branches introduce leaf nodes, tags introduce branch nodes/delimit children, attributes modify nodes—closely related to XML data model
- Base characters are stored in *reading order* (where possible)
- However, all kinds of entities are provisioned from separate encoding spaces, no need to tokenize
- Crucially, this means it is easy to *ignore* structure we don't care about!

Structured Representation (continued)



Full decomposition

- Telecode is a fully *decomposed* representation of text
- All glyph modification, e.g. diacritics, case, is accomplished through attributes
- A few benefits:
 - Diacritic agnostic/case-insensitive search much easier
 - No case-folding needed for case insensitive search!
 - Casing operations are performed uniformly for all writing systems with case (e.g. Latin, Greek, Cyrillic, Armenian, (kind of) Georgian Mkhedruli), though some difficult cases remain like Turkish i/ı and German ß

Easy Segmentable

- Segmenting text by grapheme is an important part of text processing
- “Grapheme” has several tendentious accounts/definitions—our concern is more theoretically modest
- I would say: a grapheme is close to a “unit of interactivity”
- Counting and indexing text by grapheme are simple operations in Telecode
- The number of graphemes in a Telecode string is exactly equal to the number of base characters
- A Telecode grapheme is a base character plus any attributes which modify it

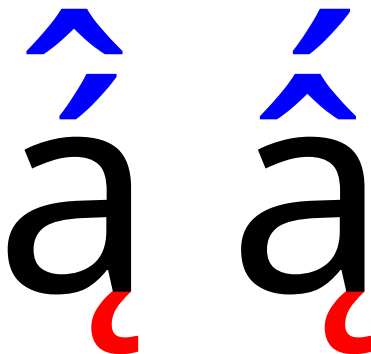
- The properties of a Unicode codepoint (e.g. combining class, case) can't generally be derived from encoding in *any* transmission format
- Unicode-aware string operations require the Unicode database
- This database is *huge* (megabytes)
- Onerous demand for embedded programs
- Bloats binaries even in regular programming
- Telecode is self-documenting by contrast
- Because Telecode is entirely decomposed and because all entities are encoded in different encoding spaces, no database is needed for most string operations

Self-normalizing

- Text encodings should be self-normalizing
- All equivalent (for some definition thereof) text should be *impossible* to represent in an unnormalized fashion
- Telecode is fully decomposed, but that's not enough
- Sometimes order of attributes matter, but other times not
- Basically, some properties are *orthogonal*

Self-normalizing

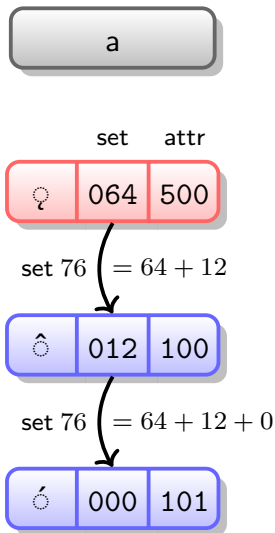
- Text encodings should be self-normalizing
- All equivalent (for some definition thereof) text should be *impossible* to represent in an unnormalized fashion
- Telecode is fully decomposed, but that's not enough
- Sometimes order of attributes matter, but other times not
- Basically, some properties are *orthogonal*
- Unicode separates combining characters into *classes*, the ogonek is in one, the acute and circumflex are in another



Self-normalizing (continued)

The solution is simple:

- 1 Separate attributes into sets of mutually orthogonal classes of properties (e.g. diacritics which attach above or below a letter)
- 2 Encode class and attribute as separate fields in a codeword
- 3 For all but the first attribute in a contiguous sequence, encode the set field as a relative offset
- 4 Unsorted attribute sequences are now impossible to represent!



The Data Structure and Its Properties

- We should be careful to not abandon the expected algorithmic and algebraic properties of common operations over text
- Valid Telecode is closed under concatenation and substrings if the substring contains only complete trees or is a subtree of some tree in the original text
- Unicode normal forms are not!
- Regular expressions are an important part of text processing infrastructure
- Thankfully regular tree languages have similar closure/decidability properties
- Already commonly used—theoretical basis for XML schema languages
- Only define operations over *well-formed* Telecode

Authoring Telecode

- Obvious objection:
writing/editing Telecode text
sounds *annoying*
- To be sure, I would not expect
users to manually juggle tags
- Telecode is similar to XML,
XML tooling could be adapted
- Smart editing tools developed
for Lisp in the Lisp/Emacs
world, such as Paredit, also
could be instructive
- I personally love the
WordPerfect model:
simultaneous views of display
and underlying formatting data

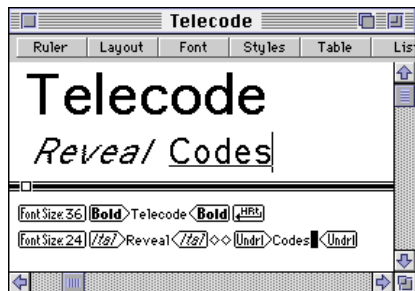
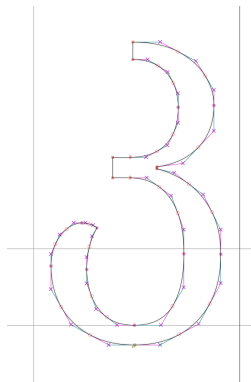


Table of Contents

- 1 Introduction
- 2 Three Myths Computer Scientists Believe About Text
- 3 Unicode: A Critique
- 4 Prolegomena to Any Future Text Encoding
- 5 Fonts as Programs**
- 6 Programmable Typography: The Dubsar Approach
- 7 Onward!

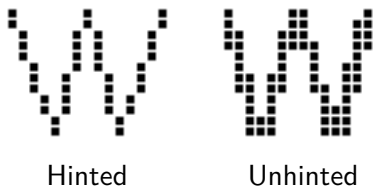
The Status Quo

- OpenType/TrueType are the standard font tech
- Glyph shapes are outlines made of Bézier curves
- Letterforms are augmented with metadata regarding how they compose to form text
- OpenType has a set of predefined shaping models, which describe kinds of such behavior
- These can be highly expressive, but even so still limited to arranging pre-defined glyphs—can't synthesize new letterforms
- Unfortunately, some models (e.g. Indic) are inadequate
- Recent versions generalize shaping considerably
- SIL Graphite has (to me) a far more elegant approach, just define arbitrary state machines



Some Difficult Parts of Text Rendering

- Hinting—naive rasterization breaks down at low resolutions, “fixup” outlines when low-res rendering
- TrueType embeds a VM, bytecode describes *how* to modify glyph paths
- Ligatures/Substitution $fi \rightarrow \text{fi}$
- Kerning, Shaping—like a 2D generalization of kerning
- Recall Maya/Egyptian quadrat layout—extreme kind of shaping
- More quotidian—making sure stacked diacritics don’t collide



Varied World Texts
Varied World Texts



Towards Programmable Typography

- My claim: fonts are *functions* from abstract text to concrete visual forms
- We're computer scientists! So why is computer typography stuck in a hot-metal paradigm of finite, set shapes?
- More general mechanisms can be paradoxically simpler
- All “advanced” typography features fall out as modes of use
- For systems like Nastaliq with *fundamentally unbounded* sets of letterforms, I see no other real option
- Programmable typography affords us *theoretical modesty*, a declarative format must be opinionated!

Declarative Languages Considered Harmful

...as interchange formats

- Would you rather implement WebAssembly or OpenType?
- Declarative formats are great for humans—bad as interchange formats
- OpenType shaping is *already* Turing-complete!
- Why does this keep happening? (see CSS)
- Declarative formats reify the entire domain complexity of a problem, its entire breadth is now everyone's problem
- There's a reason we want De Bruijn in Theorem Provers!

- The obvious question, what about METAFONT?
- Inspiring work, but has major flaws
- Still fundamentally designed around pre-generating finite, binary fonts
- The METAFONT language wouldn't work well as a font format
- Still worth learning from

Table of Contents

- 1 Introduction
- 2 Three Myths Computer Scientists Believe About Text
- 3 Unicode: A Critique
- 4 Prolegomena to Any Future Text Encoding
- 5 Fonts as Programs
- 6 Programmable Typography: The Dubsar Approach**
- 7 Onward!

The Dubsar Typography Engine

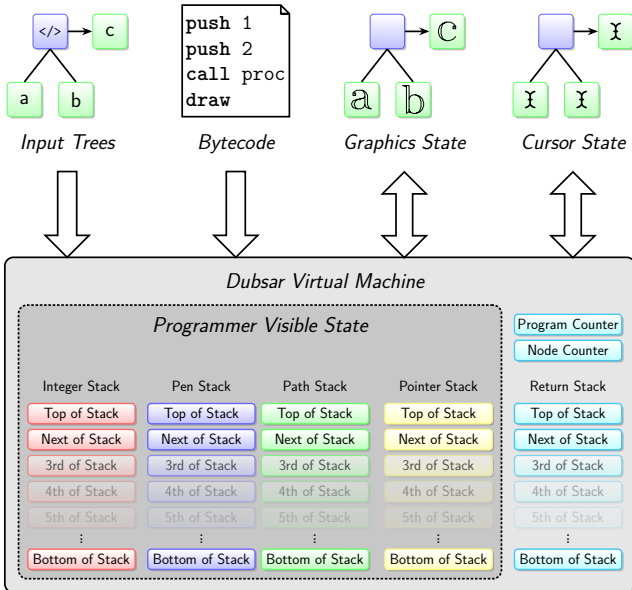


- After the Sumerian word for scribe (lit. “tablet-writer”)
- This overview will be intentionally vague—the VM is in constant flux
- A Dubsar font is a program in the language of the Dubsar VM bytecode
- This code is constructed such that it fully embodies the logic of a writing system
- The Dubsar VM thus represents a uniform interface to all writing systems

The Dubsar Virtual Machine

- Simple, typed stack machine—just data stacks and return stack (like Forth, unlike JVM/WASM)
- No recursion, statically known stack height thus no stack overflow or underflow, statically (unlike TrueType hinting VM!)
- Ensured by requiring both sides of conditionals have same net stack
- *Word* not byte machine, thus endian-agnostic
- Types are almost trivial, just number of input and output integers, paths, pens, and pointers (to nodes)
- Dubsar runtime traverses each input tree in depth first order
- When a scribeme is encountered, a textual entity which is “drawn together” (indicated through the encoding), a handler is dispatched with an opaque reference to the current scribeme node as an argument
- Bytecode can arbitrarily inspect the input trees
- Bytecode includes drawing and usual arithmetic/branching operators of course, subroutines for common operations (e.g. serifs)

Dubsar Virtual Machine Schematic



Structured Input

- Dubsar is mostly designed for Telecode
- However, takes as input basically arbitrary tree sequences
- It was originally designed for rendering TEI documents directly
- Fonts could be created that ingest Unicode, though obviously forgoing the benefits of actually structured text

Stroke-based glyphs

- Dubsar is stroke rather than outline based
- Glyph shapes are the envelope of a virtual nib stroked along a path
- Much more compact
- More natural for CJK
- Arguably a more “semantic” approach to describing glyphs

Hinting, Kerning, Shaping, Ligatures, Substitution

- Hinting is largely obviated by the stroke-based model
- Even so, rendering PPI is runtime-accessible, handlers can simply conditionally do something different if they desire
- Same mechanism allows for optical sizing
- When glyphs are drawn, they are tagged in the graphic state with the current scribeme node
- Handlers can inspect and modify the graphical render state any time, up to deletion, affine translation, &c of individual elements
- This is sufficient to implement kerning, shaping, substitution

- Each scribeme handler is responsible for returning the state of the next cursor
- Not just position but size and orientation
- Cursors can change angle and size to follow the local writing direction of text

Block-Level Rendering

- “Justification” means different things for different scripts
- Arabic uses a form of justification called *kashida*, (parts of) letters stretched, not whitespace
- If fonts are to fully capture the logic of a writing system, this must include block-level layout
- Dubsar requires input to be pre-annotated with break points
- Line-breaking handled by the font
- Includes feature to “freeze” parts of the graphical state—prevents expensive line-breaking code re-executing on every keystroke
- Isn't this wasteful?
- In practice not really, Knuth-Plass compiles to ~7KB of bytecode, ~3% of the smallest TrueType font family on my computer
- This is more than made up elsewhere

- Hershey Fonts—early stroke fonts—adapted to Dubsar
- No manual kerning! Each glyph's code places itself by measuring distance from neighbor
- Important for fonts with wide coverage, who wants to kern every pair?

Dubsar: A New Approach to Rendering Non-Western Text

Case Study: Maya, Typographic Rendering of Lintel 14

My Maya font is *very* much a work in progress, but it's getting there

Transcription

4-"IMIX" | 4-mo[lo] | TZAK-ja | K'AWIIL | KAB?-MUWAAN |
[CHAN] NAL-la | CHAK-ba-ya ka-KAAN | u-WAY-ya-ja-la |
T1001-IXIK-K'UH | IX-CHAK-JOL | IX-sa-ja |
ya-ANATZ'?-AJAW

Transliteration

chan imix chan mol tza[h]kaj k'awiil kab? muwaan chanal chak
bay kaan uwayjal? ...[l] ixik k'uh ix chak jol[oom] ix saj[al]
yanatz'? ajaw

Translation

(On the day) 4 Imix (on) the 4th of Mol, K'awiil Kab(?)
Muwaan Chanal Chak Bay Kaan, the wahy(?) of ...l Ixik K'uh Ix
Chak Joloom, Sajal and mother of the king, was conjured.

Source:

https://www.mesoweb.com/monuments/Yaxchilan/media/Lintel_14.pdf

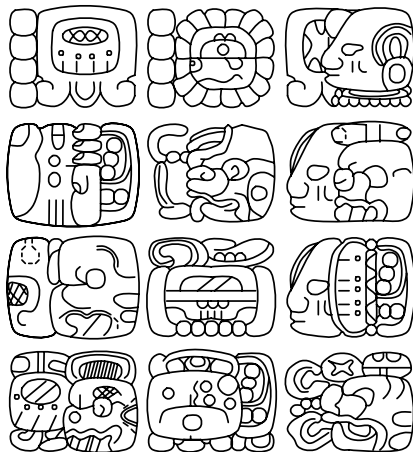


Table of Contents

- 1 Introduction
- 2 Three Myths Computer Scientists Believe About Text
- 3 Unicode: A Critique
- 4 Prolegomena to Any Future Text Encoding
- 5 Fonts as Programs
- 6 Programmable Typography: The Dubsar Approach
- 7 Onward!

- So, where does this leave us?
- Can Unicode really be displaced? Will font designers become programmers?
- Replacing Unicode would be difficult due to inertia...I'm not delusional (about this anyway)
- Even if Telecode can never truly replace Unicode, it could form the basis for better string data structures in PLs, &c
- There are applications where interoperability is not a major concern, such as embedded programming—Telecode may be a good fit
- I'm forging ahead with Telecode in my own work—OSes and PLs
- I'm much more sanguine on programmable typography, I'm preparing for a public release of Dubsar in the not-too-distant future

Writing...has this strange quality, and is very like painting; for the creatures of painting stand like living beings, but if one asks them a question, they preserve a solemn silence. —Phaedrus 275d

Questions?

email: `kevingraaf@kbg.sh`

website: `https://kbg.sh`

code: `https://fossil.kbg.sh`