Blush and Zebrackets: Two Schemes for Typographic Representation of Nested Associativity

Michael Cohen

Introduction

The notion of a hierarchical (de)composition of a structure or action, often conveyed by a graph tree, is universal. A related concept is a stack, a LIFO (last in, first out) protocol, which can be used to track tree traversal. (For example, a single-threaded computer program execution relies on a stack to reflect invocation and return from subroutines.) While trees are graphical and inherently two-dimensional, streams of characters are one-dimensional. Graph trees can be mapped to nested expressions (sets) or outlines, which use delimiters and indentation, respectively, to denote the tree structure. Delimiters (usually paired parentheses) and indentation (usually carriage returns, linefeeds, tabs, and spaces – "whitespace" in Unix[1] parlance) associate items that belong to nodes of the tree, while showing their context in the overall scheme. Left parentheses and deeper indentation represent a "push" onto a stack, and right parentheses and shallower indentation map to a "pop."

Bit-mapped terminals and high-resolution printers suggest the possibility of more elaborate presentations[2] which exploit underutilized human visual acuity. Figure 1 shows some simple axes of variation (under LATEX/TEX,[3] the document formatter with which the examples were prepared). Combining a computer filter (to analyze the text) with an extra coding dimension, orthogonal to standard techniques (to display it), can algorithmically (systematically) vary the way documents look.

Blush

Motivation

Procedural computer languages often have many blocks of code (program statements) nested logically inside enclosing envelopes of code (which in turn are nested...). In the programming language C, for example, these blocks might be alternative (*switch...case*), conditional (*if...else*), or iterative (*for, repeat,* or *while*). Pairwise delimiters scope the ranges of blocks of code, but with even properly indented code fragments, it is difficult to determine what exactly is enclosing which, resolving the extent of the various sections, especially if there are many lines between the delimiters.[4] *Blush* addresses this visual ambiguity by "reverse fielding" (a.k.a. "reverse videoing") nested blocks of

Figure 1 Some traditional and non-traditional ways of typographically indicating nested associativity.

Delimiter		Shape
		Explicit tag
Indention		
Frames		
Parenthesis		
Color/shading		
Dynamic effects		Momentary highlight (during editing)
		Synchronous flashing
Rules		Overlines and underlines
Braces		Superscripted overbraces and / or
		Superscripted underbraces
Type	Style	Serif, sans serif
		Roman, italicized
		Condensed, regular, extended
	Weight	Light, regular, medium, bold, extra-bold
	Size	Small, medium, large
	Spacing	Proportional or variable-spaced
		Fixed pitch or mono-spaced

5
NeXT
is a trademark of
NeXT Computer, Inc.

6
Reynolds, L. 1988.
Legibility of type.
Baseline,
(Cassandre):26-29.

code, alternating between black-on-white and black-on-gray. The patterns introduced by *Blush* are like variable-sized parentheses; successively highlighted blocks of code, enveloping inner blocks, suggest the structure of the underlying program or text.

Examples

Computer Program A particularly complicated and deeply nested section of an (Objective-C) computer program is shown in figure 2, after having been automatically indented. (For the purposes of illustration, the exact meaning of the segment is unimportant.) In figure 3, *Blush* has been run on this program (under Unix), turning the indentations into "gutters." Typically, *blush* ("standard") input is precomposed with *indent*, to align the code, and expand, to normalize tabs and spaces. An indentation quantum of two characters' width ("*-i2*") seems aesthetic, perhaps since it makes the gutters' width about the height of a line. For the example, a reasonably sized section is extracted with *head* and *tail* in order to display the invocation and the output on a single page.

Along with the command line invoking *Blush*, the frame around the program fragment shows the context as it would be presented on a computer monitor, including buttons, scrollers, titles and windowing tools. For hardcopy output, these would not normally be printed, as in the presentation of the following examples.

Outline

Although *Blush* was originally developed as a programming tool to macroscopically display nesting levels in computer programs, it works equally well on any indented outline. Figure 4 is an outline illustrating a concert's hierarchical decomposition; in figure 5, this outline has been "piped through" *Blush*.

Implementation

Blush was written as an active filter in C on a NeXT[5] workstation, but works on any Unix machine. It augments an input stream, using a program's indentation as an indication of block depth, inserting forward/reverse fielding display style escape sequences to distinguish successively nested blocks. This highlighting, in which blocks of code "blush," is the origin of the name. The examples are set in Letter Gothic, a monospaced sans serif typeface. Constant pitch allows display via normal computer terminals (or emulators); sans serif faces are thought to be more robust in reverse-fielded displays.[6]

One complication is that indentation levels in documents, unlike pure stacks, often don't exhaustively pop back (inside-)out, instead skipping some number of frames. Output of a non-interpolating blushing filter is shown in figure 6, exhibiting the (perhaps arguably) confusing bleeding of the *second violins* into the *brass*, the *kettledrums* into *other* and the elision of wrappers at the end. To allow clean display of the frames, therefore, *Blush* inserts blank, but appropriately shaded, lines into the output stream (*see figure 5*).

Visible Language
Volume 26, Numbers 3 and 4
Summer / Autumn 1992
Diagrams as Tools for Worldmaking

Figure 2 A section of a computer program.

```
if (!gotHit)
  do {
    if (!g) {
      i = 0;
      g = [glist objectAt:i++];
    }
    if (![g isSelected] && [g hit:&p]) {
      gotHit = YES;
      if (!shift) {
        [self deselectAll:self];
        [slist addObject:g];
        gvFlags.groupInSlist = [g isKindOf:[Group class]];
        gvFlags.clusterInSlist = [g isKindOf:[Cluster class]];
      }
      [g select];
      [conferenceSpeaker gbUser:conferenceID
       selectObjectAt:[glist indexOf:g]];   /* <jf> individ */
      DIRTY(YES);
      if (shift)
        [self getSelection];
      if (deepHit ||
          !(([slist count] == 1 &&
             [[slist lastObject] isKindOf:[SourceSinkIcon class]]) ?
            [self translateRotate : event] :[self translate:event])) {
        [self cache:[g getExtendedBounds:&eb]];
      }
    } else {
      g = [glist objectAt:i++];
    }
  } while (!gotHit && g != startg);
```

A similar work-around is required for a more subtle pathology: if the indentation levels of the source are inconsistent, such that the nesting pops out to a previously unpushed indentation, then lines of different hierarchical depth would be both adjacent and similarly shaded. This problem typically arises because a previous line, as a continuation of a still earlier line, was indented according to where the line break was, to align logically parallel elements.

This situation is observed towards the bottom of figure 2, around "[self translateRotate:event] : [self translate:event])) {". Again, the inappropriate juxtaposition is resolved with the interpolation of blank blushed lines (*see figure 3*).

Discussion

Blush offers a seamless way to extend "pretty printers" for ASCII documents. *Blush* has an option that skips an arbitrary number of columns, making it suitable for formatting FORTRAN, SPSS and other languages with label fields or sequence numbers. Further, *Blush* is compatible with outline processors, since the shading scheme is invariant under hierarchical expansion/contraction.

Properly presented, the shape of a document or program's text suggests its meaning or flow-of-control, and *Blush* helps highlight that shape. The gutters and interpolated lines introduced by *Blush* allow thinner indentation increments and deeper nesting than would otherwise be perspicuous, and make explicit the alignment at which whitespace indentation only hints.

Zebrackets

Motivation

While *Blush* provides an extension to indentation-based hierarchical representation, a large-scale approach to textual presentation, *Zebrackets* takes a small-scale approach, focusing on in-line representation of nested associativity.

To represent parenthetical expressions, traditionally typewritten documents use parentheses, "( )", for first-level subphrases, extended by (square) brackets, "[ ]", for doubly nested phrases (parentheses within parentheses), and alternating the two sets of delimiters for the rare more deeply nested phrases. Parentheses and brackets are overloaded; they are used in prose for delimiting subordinate expressions, appositives, citations and cross-references, and in mathematical formulae and computer programs for associative precedence, array subscripts, numeric ranges, function parameters and arguments, as well as special interpretations (like "((n))m" denoting "n mod[ulo] m"). Editors also use brackets to set off editorial substitution and interpolation ("[sic]"), ellipsis ("[...]"), elision ("[expletives deleted]"), etymology ("[MF *braguette* codpiece, fr. dim. of *brague* breeches, fr. OProv *braga*, fr. L *braca*, fr. Gaulish *brāca*, of Gmc origin; akin to OHG *bruoh* breeches – more at breech]"), etc. For

Visible Language
Volume 26, Numbers 3 and 4
Summer / Autumn 1992
Diagrams as Tools for Worldmaking

Figure 3 Output of *Blush* on computer program.

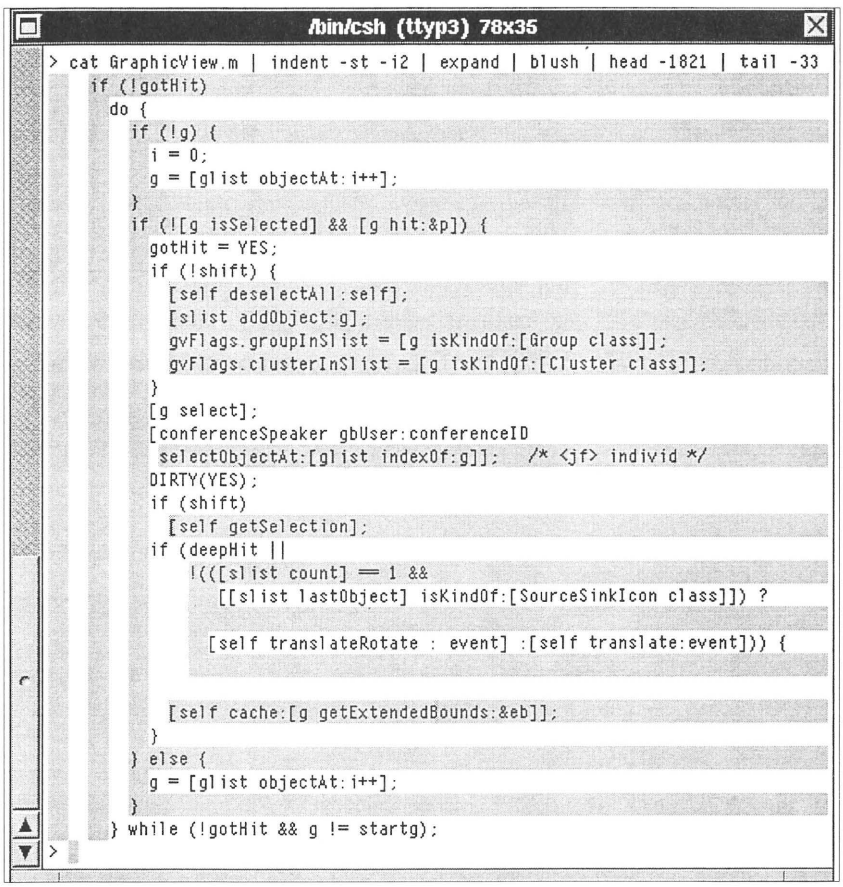

Figure 4 An outline (decomposition) of an instrumental concert.

```
concert
  chorus
    soprano
    alto
    tenor
    bass
  orchestra
    strings
      basses
      cellos
      violas
      violins
        G-string
        D-string
        A-string
        E-string
    brass
      horns
      trumpets
      trombones
      tuba
    woodwinds
      bassoons
      clarinets
      flutes
      oboes
    percussion
      bass drum
      cymbals
      snare drum
      triangle
      tubular bells
      wood block
      xylophone
      timpani
    other
      harp
```

Figure 5 Output of *Blush* on instrumental concert outline.

```
concert
  chorus
    soprano
    alto
    tenor
    bass
  orchestra
    strings
      basses
      cellos
      violas
      violins
        G-string
        D-string
        A-string
        E-string
    brass
      horns
      trumpets
      trombones
      tuba
    woodwinds
      bassoons
      clarinets
      flutes
      oboes
    percussion
      bass drum
      cymbals
      snare drum
      triangle
      tubular bells
      wood block
      xylophone
      timpani
    other
      harp
      piano
```

Figure 6 Output of non-interpolating *Blush*.

```
concert
  chorus
    soprano
    alto
    tenor
    bass
  orchestra
    strings
      basses
      cellos
      violas
      violins
        G-string
        D-string
        A-string
        E-string
    brass
      horns
      trumpets
      trombones
      tuba
    woodwinds
      bassoons
      clarinets
      flutes
      oboes
    percussion
      bass drum
      cymbals
      snare drum
      triangle
      tubular bells
      wood block
      xylophone
      timpani
    other
      harp
      piano
```

Figure 7 **Chemical Compound, before and after *Zebrackets*.**

ACTIVE INGREDIENT: Hydramethylnon [tetrahydro-5, 5-dimethyl-2(1*H*)-pyrimidinone(3-[4-(trifluoromethyl)phenyl]-1-(2-[4-(trifluoromethyl) phenyl]ethenyl)-2-propenylidene)hydrazone]

ACTIVE INGREDIENT: Hydramethylnon [tetrahydro-5, 5-dimethyl-2(1*H*)-pyrimidinone(3-{4-(trifluoromethyl)phenyl}-1-(2-{4-(trifluoromethyl} phenyl}ethenyl}-2-propenylidene)hydrazone]

Figure 8 **LISP code, before and after *Zebrackets*.**

```
(DEFUN ANY (LST)
 (COND ((NULL LST) NIL)
       ((CAR LST) T)
       (T (ANY (CDR LST))) ) )

(DEFUN ANY {LST}
 {COND {{NULL LST} NIL}
       {{CAR LST} T}
       {T {ANY {CDR LST}}} } )
```

Figure 9 **Objective-C code, before and after applying *Zebrackets*.**

```
[inspectorPanel setAccessoryView:[[[[accessory contentView] subviews]
                                   objectAt:0] removeFromSuperview]];

[inspectorPanel setAccessoryView:[{{{accessory contentView} subviews}
                                   objectAt:0} removeFromSuperview]];
```

Figure 10 **First order predicate calculus, before and after applying *Zebrackets*.**

$$\forall P\ \forall v\ \forall w\ \forall x\ \forall y\ \forall z\ ((P(v,w) \Rightarrow P(w,v)) \wedge ((P(x,y) \wedge P(y,z)) \Rightarrow P(x,z)) \Rightarrow$$

$$\forall P\ \forall v\ \forall w\ \forall x\ \forall y\ \forall z\ (\{P\{v,w\} \Rightarrow P\{w,v\}\} \wedge (\{P\{x,y\} \wedge P\{y,z\}\} \Rightarrow P\{x,z\}) \Rightarrow$$

7
Traditionally precise LISP punctuation puts a space before parentheses that close an association begun on another line. Many LISP interpreters also have a super parenthesis, "]", that closes any pendant associations.

literature and journalism, these conventions have been adequate, since the reader could usually parse the subphrases. Extended schemes have used (curly or set) braces (a.k.a. "bracelets"), "{}", and angle brackets (a.k.a. "inequality signs"), "<>", to indicate more deeply nested phrases. But especially for non-natural languages, in which stacks of association are not only comprehensible but necessary and encouraged, a more extensible scheme is needed.

Using size to denote nesting (or any other kind of) level doesn't work, since juxtaposed expressions, at the same parenthetical depth, might require different sized delimiters for aesthetic (lexicographical) reasons. "$\left(A*(B+C)\right)$" suffices, but expressions that span use or over- and under-lines or braces break down across line boundaries.

Zebrackets extends parentheses, square brackets (and other pairwise delimiters) by systematically striping them according to an index reflecting their order in the text. Each index of the respective delimiter pairs is cast into a binary pattern, which is superimposed on the characters as striations. Alternate encoding schemes are also possible, but informal experiments suggest that users tend to look only for matching delimiters. Some of these special-purpose modes are mentioned later in the discussion.

Examples

Chemical Compound Figure 7 shows the chemical formula for a popular roach control system, as shown on its box. The top version is without brackets and the bottom version is zebracketed. What is the "matching bookend" to the parenthesis before "3-" in the second line? A quick scan of the zebracketed version finds the parenthesis closing the association in question.

LISP The LISP programming language relies on parentheses for delimiting lists,[7] the language's basic data structure. The function shown in figure 8 performs a generalized "inclusive or." As in the previous example, the top version is without zebrackets, while the bottom version with zebrackets performs the same function and elucidates the associations.

Objective-C Figure 9 shows a line from an Objective-C program, with and without zebrackets as before.

Logic Figure 10 shows the first order predicate calculus notation indicating that symmetry and transitivity imply reflexivity. *Zebrackets* illuminates the precedence.

Association beyond single parenthetical pairs Just as pairs of identically valued parentheses point at each other, like matching bookends, so do multiple sets of same-valued delimiters associate beyond the scope of a single pair of parentheses. In figure 11, the zebrackets are not just reinforcing patterns already present, but are adding new information.

Implementation

The implementation of *Zebrackets* comprises two aspects: a filter to generate permuted invocations of the underlying parentheses

Figure 11 Association beyond single parenthetical pairs

$$(\ln|\{\ln|\{x^2-2\}|\}|)' = \{\ln|\{x^2-2\}|\}^{-1}\,\{x^2-2\}^{-1}\,\{x^2-2\}'$$

8
Knuth, D.E. 1986. *The Metafont Book.* Reading, Massachusetts: Addison-Wesley.

9
Tufte, E.R. 1983. *The Visual Display of Quantitative Information.* Cheshire, Connecticut: Graphics Press.

10
Tufte, E.R. 1990. *Envisioning Information.* Cheshire, Connecticut: Graphics Press.

and brackets, and the delimiter fonts themselves. A filter was written to intelligently parse expressions, substituting zebrackets for ordinary delimiters. *Zebrackets* glyphs are implemented with METAFONT,[8] an algorithmic typographic computer language, as extension fonts in the Computer Modern typeface. Unlike *Blush*, *Zebrackets* adopts a minimalist "less (ink) is more (data)" philosophy,[9] adding information to regular parentheses and brackets by resetting some pixels in the characters.

Because of the tendency of white features to bleed out into a black background, the striations need not be terribly thick to be perceivable. In the examples shown, the stripes are 1 pt. (1/72.27 inches) thick on 12 pt. type, subsuming about 2.6 arc-minutes (≈ 0.044 degrees ≈ 0.77 milliradians) of visual angle (assuming a reading distance of 18 inches). This is greater than the accepted (if heuristic) industrial minimum for visual acuity, 1 arc-minute.

Since there are usually more ascending than descending characters, readers (of English) tend to look slightly above the line of print, deriving meaning from the "top coastline"(upper half) of the text. Therefore, the index of the parenthetical pair is encoded with the LSB (least significant bit) at the top of the parenthesis or bracket. Further, for even curved parentheses, the stripes are drawn horizontally (rather than radially), so that the reader might imagine an invisible line drawn through the intermediate text. Since the delimiter pairs are horizontally symmetrical, they can be conceptually associated, and since the bands are aligning, they can also be visually associated, like toothpicks holding bread around a thick sandwich.

Discussion

For single levels of nesting, the extended parentheses and brackets are identical to the unenhanced, since an index of zero leaves the delimiters unbanded in *Zebrackets'* default positively-coded scheme, chosen to preserve the backwards compatibility of the curve substrate.

For deeper levels, *Zebrackets* tries to maintain this backwards compatibility by being non-distracting to users who don't know about it. It is, by design, "just noticeable," right over the limen, or edge of perceptibility. The added information is meant to be clear to the user actively seeking it and transparent to any user not actively seeking it. (Such an effect is like making something a little smaller to call attention to it.) By designing a scheme that is both noticeable and ignorable, one is obtained that is unambiguous but unobtrusive, unmistakable but unassuming. In practice, however, *Zebrackets* has two problems: For linear reading without searching, zebrackets can actually be distracting, introducing high-frequency noise (that looks like printer spotting). On the other hand, zebracket striations can be difficult to see, especially for readers with less sharp eyes and zebracketed documents are not robust under (perhaps repeated) copying by low-resolution devices (like most faxes).

Figure 12 Foreground stripes.

```
(DEFUN ANY (LST)
 (COND ((NULL LST) NIL)
       ((CAR LST) T)
       (T (ANY (CDR LST))) ) )
```

Figure 13 Different encoding modes

Indexing the streaks "inside-out," (so that the outer delimiters pairs have a higher index (than inner)), orders the evaluation of expression trees. Displaying breadth (or depth (or both)), instead of an incremental index, is useful for visualizing expression complexity. Striping the delimiters "upside-down," from the bottom, might be better for languages (like Hebrew) that carry more information in the "lower coastline."

Variations that overcome these limitations are possible. For instance, grayscale striations (not yet implemented) might disappear at normal reading speed, but be visible when doing a detailed search. Alternatively, using black stripes to tick the parentheses, instead of dropping out (white) segments, is more legible, if less inconspicuous (*see figure 12*).

For too-deeply parenthesized expressions, *Zebrackets* degrades, but gracefully. The filter maintains a stack, wrapping around (repeating) the indexing scheme if the delimiters exhaust the range of uniquely encodable depths. Most of the *Zebrackets* fonts in this paper use four bands (allowing $2^4 = 16$ different versions of each pair of delimiters), but denser representations are possible (perhaps adaptively chosen, so that the maximum depth of the expression determines the fineness of the striations, or tuned by the user, to match visual acuity). Further, invocation of different encoding modes, perhaps for special purposes, is straight-forward (*see figure 13*).

Conclusion

These utilities recall the counter-intuitive advice "To clarify, add detail."[10] *Blush* and *Zebrackets* are tools in a suite of prettyprinters (like Unix's *vgrind*) that start to treat words and documents as pictures, with attendant increases in information/ink value: denser data without loss of legibility.

Documents should look like what they mean: context after content, form after function, process after product and style after substance. Creative orthography frees words from traditional (technologically imposed) constraints, allowing textual re-presentation of multidimensional concepts by projecting multilayered structure into linear text. Extended electronic typography, as manifested by tools like *Blush* and *Zebrackets*, provides additional parsing cues and differentiates between heretofore duplicate symbols.

The handwritten "publishing" of pre-Gutenberg scribes was arbitrarily subtle, with its attendant human caprice (and mistakes). Printing can be thought of as having rigidified this information transmission. The research described here loosens some of that determinism, not by randomizing the presented information, but by softening the digitized boundaries, thereby expanding the range of expression. *Blush* and *Zebrackets* indicate evolving modes of written representation, algorithmic descriptions driving adaptive displays, as style catches up to technology.

Visible Language
Volume 26, Numbers 3 and 4
Summer / Autumn 1992
Diagrams as Tools for Worldmaking

Diagramming, in the age of fast computer graphics, has been revitalized as a tool for finding and communicating patterns. What is new is the ability for a diagram on a computer screen to be dimensionally manipulated in real time. Instead of the two dimensions of a piece of paper, dynamic diagrams have four: three spatial dimensions and an active time dimension.

Operations for working with dynamic diagrams deal with both structure and observation. Tools for describing and altering the model are transformations; tools for changing the way its behavior is viewed are transpositions. Transpositions, the subject of this paper, change the way the display of the model proceeds, letting the viewer look at the unfolding results in different ways. These may be spatial, allowing the viewer to move the point of view or the point viewed; procedural, allowing the viewer to change the flow of time; or organizational, allowing the viewer to arrange multiple simultaneous views and to interject auxiliary measuring tools.

Vijay K. Sivasankaran is a design and systems planner at Doblin Group, a Chicago-based strategic design planning firm. There, he works with design and information system teams with a focus on developing software. He received his degree in product design from the National Institute of Design in India. Following work in a systems design consultancy firm in India, he did his thesis on dynamic diagramming as a computer-supported tool for design processes at the Institute of Design, Illinois Institute of Technology. His current research is directed toward computer-supported dynamic diagramming, visual decision support systems and development of prototypes for design planning systems.

Charles L. Owen, professor of design at the Institute of Design, Illinois Institute of Technology, in Chicago, teaches in the design graduate and doctoral programs, directs the Design Processes Laboratory and publishes the *Design Processes Newsletter*. He has worked in the fields of product design, computer-supported design, design methodology and computer graphics – teaching, conducting research and consulting. He has written computer programs for business and institutional applications, has published widely and lectured extensively in the United States and abroad. His current research is directed toward structured planning techniques, computer-supported "dynamic" diagramming and the development of design support systems employing computer-assisted processes.

Design Processes Laboratory,
Institute of Design,
Illinois Institute of Technology,
Chicago, Illinois 60616.

Visible Language 26:3/4,
Vijay K. Sivasankaran and
Charles L. Owen, pp. 450-473,

Rhode Island School of Design,
Providence, Rhode Island 02903.