Aligning Text in Mathematica* Graphics: A Question About the PostScript Infrastructure

Michael P. Barnett†

June 2, 2010

1 Introduction

This report is a working document that seeks advice from POSTSCRIPT experts about certain details of font encoding. I need these details to align built-up multi-font text in MATHEMATICA graphics objects. These objects comprise diagrams that I include in LaTeX manuscripts. The diagrams pertain to topics in natural science, mathematics and the humanities. The text is aligned by the TMG (text in MATHEMATICA) package that I coded.

Fig. 1 and nearly 30 similar diagrams are in a recent paper on nuclear magnetic resonance (NMR) that I wrote with István Pelczer [1]. The construction of these diagrams prompted the work on TMG. The package contains an `encode`

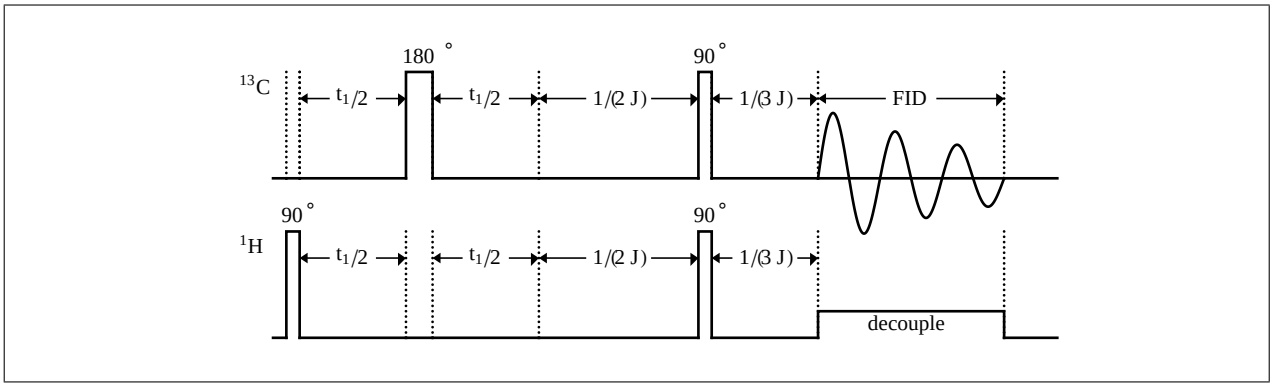

Figure 1: A pulse sequence diagram.

function that positions the contents of separate MATHEMATICA `Text` expressions precisely. This is a standard need, *e.g.*, when a set of related diagrams consists of varied selections of modules that contain text. The definitive description of `Text` in *The Mathematica Book* [2] states:

> "**Text[***expr, coords, offset***]** specifies an offset for the block of text relative to the coordinates given."

The description goes on to mention sample offsets that include "{-1,0} left-hand end at $\{x, y\}$" and "{0, -1} centered above $\{x, y\}$". The obvious extension is that {-1,-1} puts the lower left corner of the text at $\{x, y\}$. The description in [2] refers to the "bounding rectangle that surrounds the text".

The idea of bounding rectangles that surround text has been inherent in the use of moveable type for millenia [3] and, more recently, in phototypesetting [4]. It is associated with the idea of a baseline, defined as "the line upon which most letters 'sit' and below which descenders extend" [5]. Fig.2 shows a sequence of words and isolated characters in serif, sans-serif and Greek fonts, that were typeset by elementary LaTeX coding. The rectangles that surround the characters and the baseline were drawn by `\rule` commands. Typesetting software has customarily treated the vertical coordinate of a piece of text as the position of its baseline, since the inception of the field in the late 1950s [4].

*MATHEMATICA is a registered trademark of Wolfram Research Inc.

†Meadow Lakes, Hightstown, NJ 08520, michaelb@princeton.edu

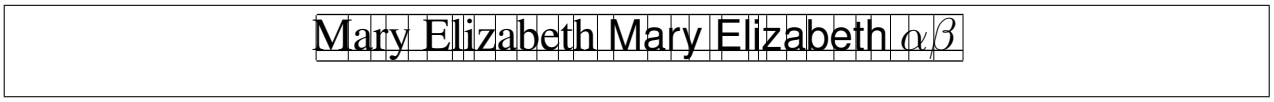

Figure 2: Bounding boxes aligned on baseline.

Digital fonts were developed that treated each character as if it were contained in a rectangle, that had the point size as its height, with baselines positioned for consistency between fonts. This paralleled the design of metal type slugs.

Fig. 3 shows some examples of unexpectedly bad alignment produced by `Text` commands. These all contain the offset pair `{-1,-1}`, and the same y value is used in the `Text` and `Line` expressions that produced each row. These

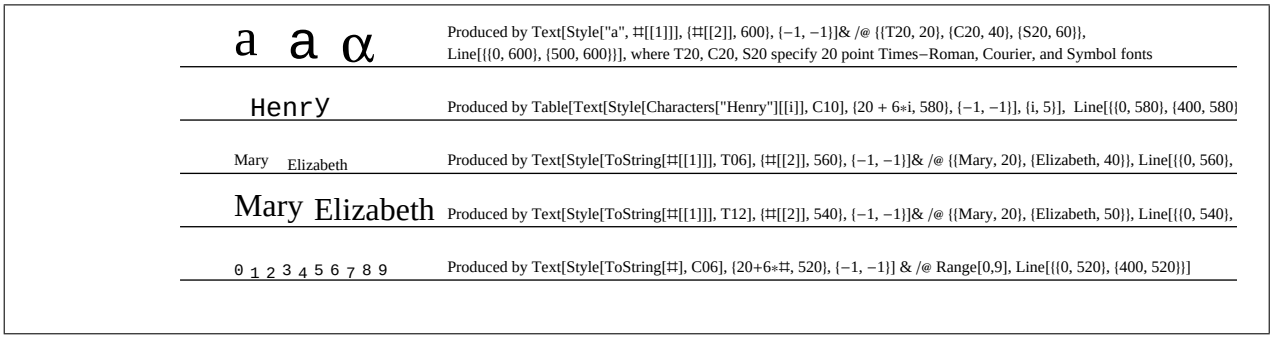

Figure 3: Some examples of unexpected misalignment.

are concise forms of

```
{Text[Style["a", FontFamily -> "Times-Roman", FontSize -> 20], {20, 600}, {-1, -1}],
 Text[Style["a", FontFamily -> "Courier", FontSize -> 20], {40, 600}, {-1, -1}],
 Text[Style["a", FontFamily->"Symbol", FontSize->20], {60, 600}, {-1, -1},
 Line[{{0, 600}, {500, 600}}]}

{Text[Style["H", FontFamily -> "Times-Roman", FontSize -> 20], {26, 580}, {-1, -1}],
 Text[Style["e", FontFamily -> "Times-Roman", FontSize - >20], {32, 580}, {-1, -1}],
 ...
 Text[Style["y", FontFamily -> "Times-Roman", FontSize -> 20], {50, 580}, {-1, -1}],
 Line[{{0, 580}, {400, 580}}]}

{Text[Style["Mary", FontFamily -> "Times-Roman", FontSize -> 12], {20, 560},{-1, -1}],
 Text[Style["Elizabeth", FontFamily -> "Times-Roman", FontSize -> 12], {40, 560}, {-1, -1}],
 Line[{{0, 540}, {400, 540}}]}

{Text[Style["Mary", FontFamily -> "Times-Roman", FontSize -> 20], {20, 540},{-1, -1}],
 Text[Style["Elizabeth", FontFamily -> "Times-Roman", FontSize -> 20], {50, 540}, {-1, -1}],
 Line[{{0, 540}, {400, 540}}]}

{Text[Style["0", FontFamily -> "Courier", FontSize -> 6], {20, 520}, {-1, -1}],
 Text[Style["1", FontFamily -> "Courier", FontSize -> 6], {20, 526}, {-1, -1}],
 ...
 Text[Style["9", FontFamily -> Courier", FontSize -> 6], {20, 554}, {-1, -1}],
 Line[{{0, 520}, {400, 520}}]}
```

There are several reasons for the alignment effects in Fig. 3.

1. In the 1^{st} row, the letter "a" in Times-Roman and Courier fonts, and the α do not line up because the different fonts are coded with different baselines in their repective bounding boxes.
2. In the 2^{nd} row, the **bases** of the rectangles around the individual letters in "Henry" are aligned. This makes the "y" high relative to the other letters.
3. In the 3^{rd} and 4^{th} rows, the string "Mary" has consistent baselines. So does "Elizabeth". But the "y" in "Mary" pushes it up, relative to "Elizabeth".

4. In the 5^{th} row, I think that the digits do not line up because 0, 3, 5, 6, 8 and 9 were coded using one set of conventions, and 1, 2, 4 and 7 using a different set.

A practical consequence of the alignment of the personal names is in the depiction of Tudor genealogy. I wrote a MATHEMATICA script in the 1990's to display genealogies, because I work with historians. (The algorithmic issues actually have a commonality with the construction of metabolic pathway diagrams). Fig. 4 is a minimalistic display of relationships that dominated British society for half a century. The misalignment would be unacceptable in a scholarly journal that dealt with the substantive issues that this presents.

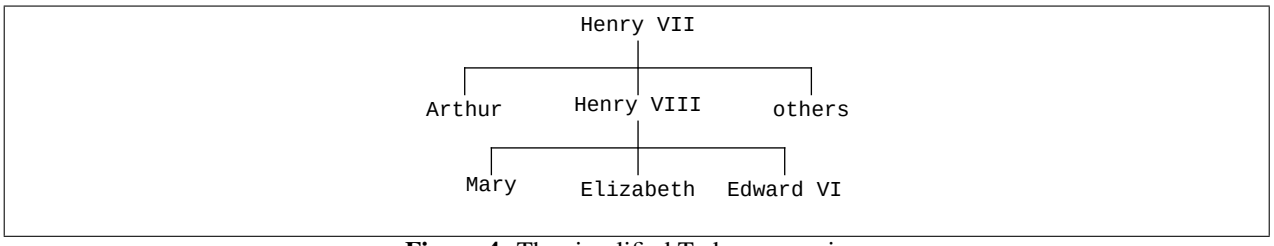

Figure 4: The simplified Tudor succession.

Some forms of undesirable alignment become more pronounced as font size is decreased. This may be related to an apparent drift in the snugness of the bounding rectangle. Although I have not found relevant data directly, some simple syntactic errors make the system display the rectangles that it seems to use. This is done in Fig. 5 by using `null` as the y offset. The system treats it as 0. In the 1^{st} row, the string "Ay" is set successively in 40, 20, 10 and 5 point Courier type. The y coordinate in the `Text` expression is 550, and a `Line` expression draws a line with $y = 550$ across the page. The serif of the "y" touches this line in 40 point type, but not in the smaller sizes. A line drawn with $y = 557.6$ shows the elevation of the serifs of the "A" relative to those of the "y" in 40 point type.

In the 2^{nd} row, the two letters "A" and "y" are set by separate `Text` statements, with $y = 500$. The serif of the 40 point "y" touches a line with this coordinate, but the relative elevation of the "A" has dropped to 6.1 points. As the size decreases, the "y" continues to move up relative to the "A".

In the 3^{rd} row the rectangles surrounding the "y" seem to have moved down slightly, relative to the serif with decreasing point size. Fig. 6 shows the 5 point example, magnified 8-fold by the `scale` parameter in the LaTeX

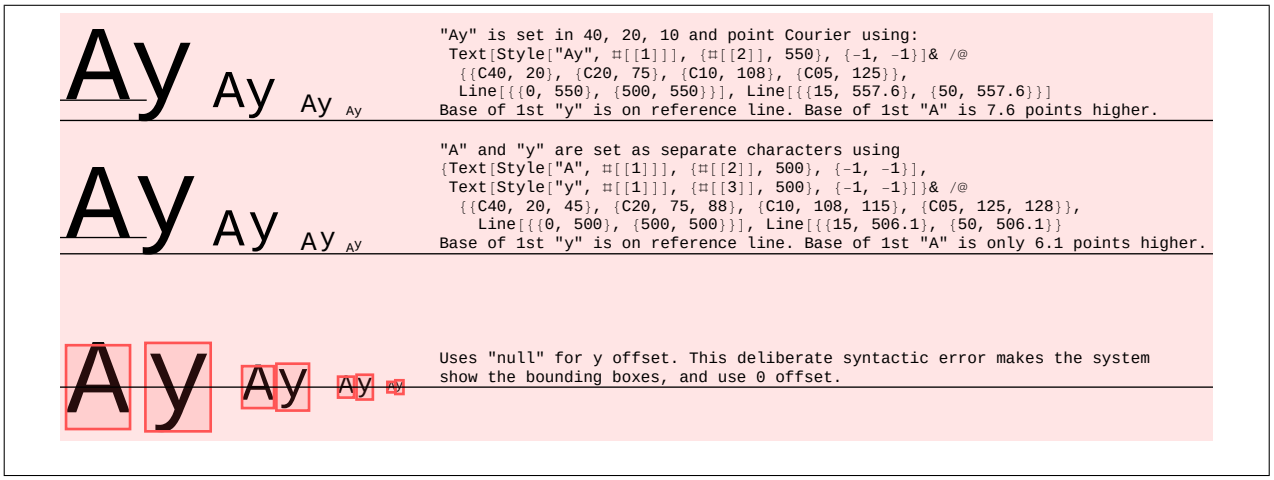

Figure 5: Forced behaviour that may show a bounding box problem.

`includegraphics` command. The position of the horizontal line at $y = 450$ emphasizes the change.

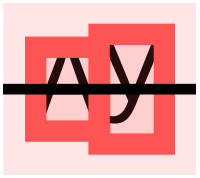

Figure 6: 8-fold magnification of 5 point example.

2 The tmg encode function

I align (horizontally) a body of text that is displayed on a single line, by putting the i-th character, denoted here by c_i, into a separate expression of the form

$$\texttt{Text[Style[}c_i\texttt{,FontFamily->}f_i\texttt{, FontSize->}s_i\texttt{],\{}x_0+h_i,\,\bar{y}\texttt{\},\{-1,}\sigma(f_i,s_i,c_i)\texttt{\}]}$$

where

$$h_i = \sum_{j=1}^{i-1} \frac{s_i}{10} w(f_i, c_i)$$

and

1. c_i, f_i and s_i are the i-th character and the font style and font size in which it is set,
2. $w(f_i, c_i)$ is the width of c_i in 10 point type (this gives $w(\text{"}\texttt{Courier}\text{"}, c)$ the value 6 for the entire character set),
3. $\sigma(f_i, s_i, c_i)$ is the offset that puts the baseline of the character onto the line $y = \bar{y}$, that is specified in the coordinates part of the `Text` statement,
4. x_0 is the starting x coordinate of the text.

I developed methods to find widths and offsets by trial and error. Using these, I found the widths for the Courier, Times-Roman and Symbol fonts very easily and accurately. I found the offsets for the Courier and Symbol fonts for point sizes 4 to 10, and Times-Roman for size 12, with considerable difficulty and some uncertainty.

I would like advice on finding the offsets algorithmically from the font tables.

The information may be in the chapter on fonts in the *Postscript Language Reference* manual [6]. The learning curve for this seems non-trivial, and I do not want to climb it unnecessarily.

The TMG expression

$$\texttt{encodeString[}font,\ size,\ x,\ y,\ string\texttt{]}$$

sets $string$ in the specified font face and font size, starting with the baseline of the first character at position (x, y). In the more general expression

$$\texttt{encodeSequence[}item_1, item_2, \ldots\texttt{]}$$

each item is either

1. a character string, that has the head `String`, to be set in uniform font and size on a common baseline,
2. a letter or a letter followed by one or more characters that is each a letter or digit, with the head `Symbol`, *i.e.* an identifier without a value — its `ToString` value is treated as an item of type 1,
3. `ps[`n`]`: changes the font to size n without altering the baseline,
4. `tf[`f`]`: changes the font to style f,
5. `tf[`f, n`]`: changes the font to style f and size n,
6. `sub[`s`]`: sets s in the decoration size, sunk to subscript level,
7. `sup[`s`]`: sets s in the decoration size, raised to superscript level,
8. `subSup[`s_1, s_2`]`: sets s_1 and s_2 as subscript and superscript, left aligned,
9. `lSubSup[`s_1, s_2`]`: does correspondingly, right aligned,

10. `tab[`$\bar{x}$`]` changes the x coordinate for the next displayed object to $\bar{x}$,
11. `vtab[`$\bar{y}$`]` changes the y coordinate for the next displayed object to $\bar{y}$,
12. `hs[`$\bar{n}$`]` increases the x coordinate by n,
13. `vs[`$\bar{n}$`]` increases the y coordinate by n.

This set of commands will be extended to provide powerful algorithmic formating capabilities.

The following statement produced Fig. 7 for comparison with Figs. 3 and 4. It uses `encode` expressions.

```
export[alignedByEncode =
{AbsoluteThickness[.1],
 encode[ps[20], vtab[620], tab[20], tf["Times-Roman"], "a",
  tab[40], tf[Courier], "a", tab[60], tf[Symbol], "a"][[1]],
 Line[{{20, 620}, {80, 620}}],
 encodeString[Courier, 10, 20, 600, "Henry"][[1]],
 Line[{{20, 600}, {50, 600}}],
 encodeString["Times-Roman", 6, 20, 580, "Mary"][[1]],
 encodeString["Times-Roman", 6, 40, 580, "Elizabeth"][[1]],
 Line[{{20, 580}, {70, 580}}],
 encodeString["Times-Roman", 12, 20, 560, "Mary"][[1]],
 encodeString["Times-Roman", 12, 50, 560, "Elizabeth"][[1]],
 Line[{{20, 560}, {100, 560}}],
 encode[tf[Courier], ps[6], tab[20], vtab[540],
    "0", "1", "2", "3", "4", "5", "6", "7", "8", "9"][[1]],
 Line[{{20, 540}, {60, 540}}] }]
```

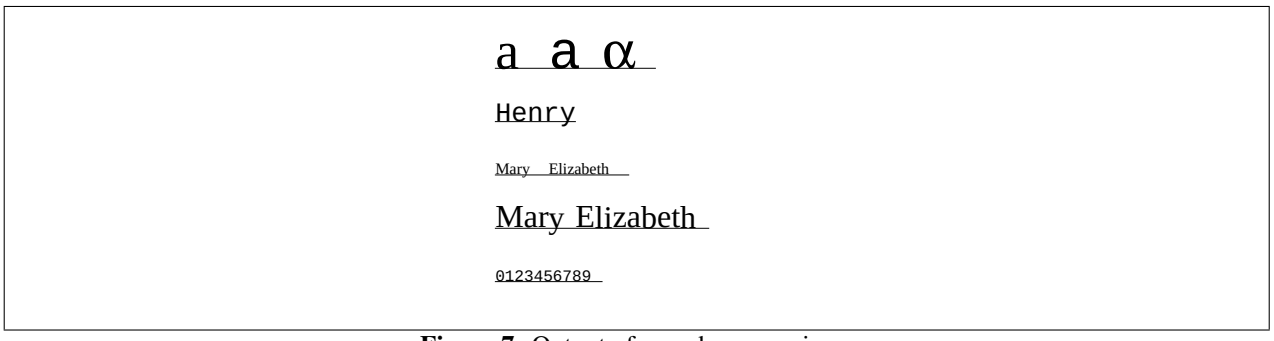

Figure 7: Output of encode expressions.

The following statement produced Fig. 8.

```
tudorTreeEdges =
 {Line[{{200, 600}, {200, 590}}], Line[{{135, 590}, {265, 590}}],
  Line[{{#, 590}, {#, 580}}]& /@ {135, 200, 265},
  Line[{{200, 570}, {200, 560}}], Line[{{145, 560}, {255, 560}}],
  Line[{{#, 560}, {#, 550}}]& /@ {145, 200, 255}}

encodedTudorNames =
{encodeString[Courier, 8, 178.4, 605, "Henry VII"][[1]],
 encodeString[Courier, 8, 120.6, 575, "Arthur"][[1]],
 encodeString[Courier, 8, 186.0, 575, "Henry VIII"][[1]],
 encodeString[Courier, 8, 250.6, 575, "others"][[1]],
 encodeString[Courier, 8, 135.4, 545, "Mary"][[1]],
 encodeString[Courier, 8, 178.4, 545, "Elizabeth"][[1]],
 encodeString[Courier, 8, 233.4, 545, "Edward VI"][[1]]}

export[refinedTudors = {tudorTreeEdges, encodedTudorNames}]
```

The alignment is imperfect but I believe it can be improved by fine tuning the offsets. I think that each letter has a range of offsets that are acceptable in one context, and a different range in another context, with very narrow overlap.

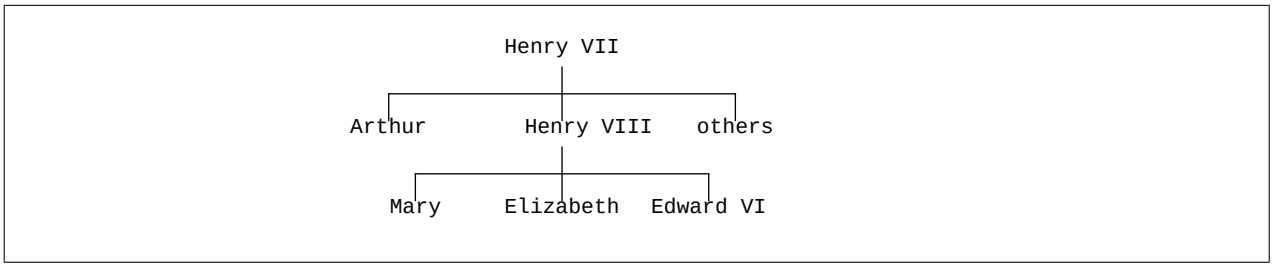

Figure 8: Tudor names produced by encode expressions.

I have been using just one or two contexts for each letter, and picking an offset within the range of acceptability in an arbitrary manner.

The present technique is adequate for the alignment of short expressions with each other, as needed in the pulse sequence diagram of Fig. 1. That diagram, and the other diagrams in [1], were produced by *ad hoc* coding before I started TMG. Using TMG, I will be able to extend the options for including explanatory text in the diagrams, even in its present crude form. An algorithmic basis for TMG would enable many other applications of MATHEMATICA graphics in the kernel mode.

References

1. M. P. Barnett and I. Pelczer, Pulse sequence editing by symbolic calculation, J. Magnetic Resonance, J. Magn. Reson. 204 (2010) 189–195 (doi:10.1016/j.jmr.2010.01.009).

2. S. Wolfram, The Mathematica Book, 2nd ed. Addison-Wesley, New York, 1991, or later editions

3. Movable type, see`http://en.wikipedia.org/wiki/Movable_type`.

4. M. P. Barnett, Computer Typesetting, Experiments and Prospects, MIT Press. 1965.

5. Baseline (typographY), see `http://en.wikipedia.org/wiki/Baseline_(typography)`

6. PostScript Language Reference, Adobe Systems Incorporated.`http://www.adobe.com/devnet/postscript/pdfs/PLRM.pdf`.