La LETTRE GUTenberg

Numéro 57 – mars 2026

Numéro spécial

FONTES VARIABLES

ROBOTOFLEX, UNE FONTE VARIABLE

Roboto a été l'une des premières fontes principales des systèmes Android qui, en 2011, étaient encore en basse définition ; elle a été redessinée par Christian Robertson pour être compatible avec les nouveaux écrans Android de bien meilleure résolution vers 2018 ; Google Fonts en a commandé, sous le nom de Roboto Flex, la version variable qui a été conçue et dessinée par David Berlow et son équipe de Font Bureau (qui vient d'arrêter de fonctionner en 2024).

Roboto est une linéale, plus précisément une néo-grotesque, dans la lignée de Helvetica, Univers, Arial (voir la figure 1).

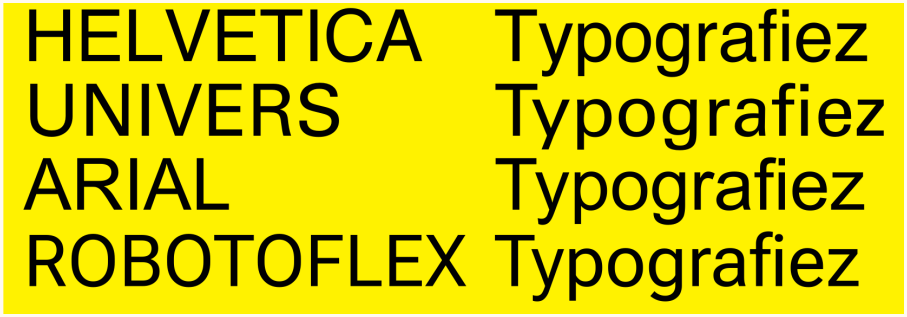


FIGURE 1 – Les linéales en question. Dans cet exemple, les lettres *fi* montrent mieux le style des fontes que les lettres *ph*.

En fait, ce qui nous intéresse dans cette fonte, c'est qu'il s'agit d'une « fonte variable » librement téléchargeable[1].

Nous allons présenter, dans une première partie, ce que sont les fontes variables et comment LualaTeX permet de s'en servir ; puis, dans une seconde partie, ce que sont les axes ou variants de RobotoFlex, puis ceux d'autres fontes variables, et à quoi ils peuvent servir. La conclusion nous permettra de proposer quelques opinions sur la créativité et les fontes variables.

Note de la rédaction

Le présent article fait suite à l'exposé de Jacques André, consacré aux fontes variables, lors de la Journée GUTenberg organisée à l'ÉNS le 16 novembre 2024[a].

Cette *Lettre* est composée avec la fonte variable Robotoflex.

[a] https://www.gutenberg-asso.fr/Journee-GUTenberg-2024

1. On peut la télécharger par exemple sur GitHub (https://github.com/googlefonts/roboto-flex/blob/main/fonts/), où elle est sous licence *SIL Open Font License v1.1*. Il s'agit du fichier RobotoFlex[GRAD,XOPQ,XTRA,YOPQ,YTAS,YTDE,YTFI,YTLC,YTUC,opsz,slnt,wdth,wght].ttf. De même, on y trouve la fonte variable associée RobotoMono à https://github.com/googlefonts/RobotoMono.

PREMIÈRE PARTIE : LES FONTES VARIABLES

Dès le XVI^e siècle les dessinateurs de caractères déclinent leurs fontes avec des variantes de forme. Ainsi Granjon propose-t-il pour ses *Civilités*[2] au moins deux variantes : bastarde et courante [1]. Au XVIII^e siècle, Fournier offre pour le seul romain de son caractère six variantes (en jouant sur l'étroitesse et la grosseur d'œil) comme le montre la figure ci-dessous.

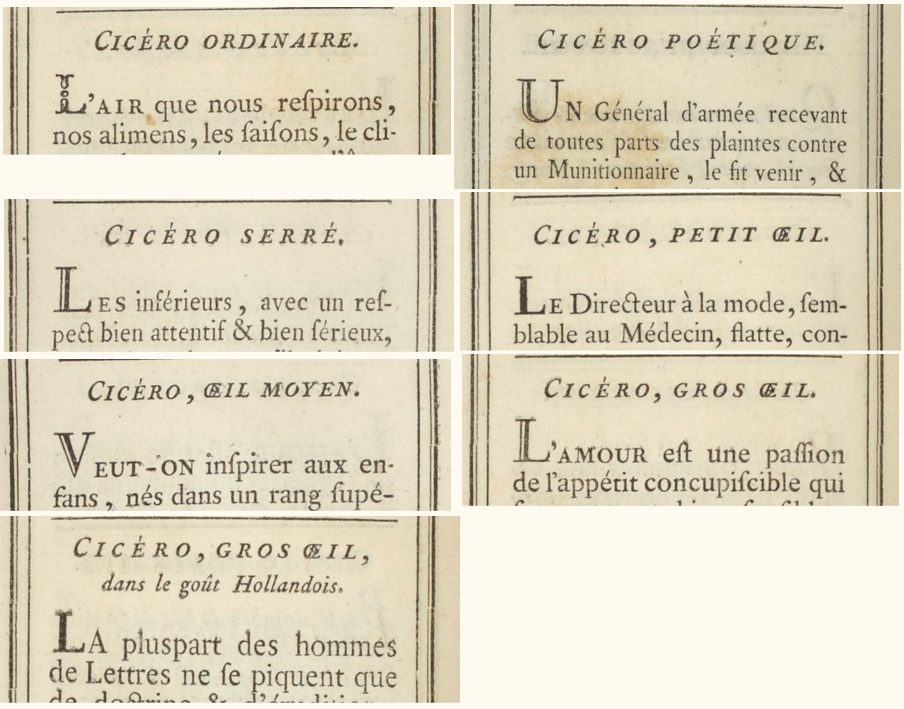


FIGURE 2 – Sept variantes du Cicéro romain de Fournier, *Manuel typographique*, tome II, 1765 [Gallica].

2. Granjon a dessiné un caractère de civilité, qui est devenu célèbre au point d'être considéré comme une œuvre. Pour cette raison, et à l'instar du *Romain du Roi*, on compose son nom en italiques avec majuscule.

Histoire des variantes

Du temps des caractères mobiles

Au XIXe siècle, les variantes s'étendent (par exemple, à partir d'une fonte « normale » l'Anglais Figgins (1766-1844) décline les variantes égyptienne, mécane, linéale [2]) tandis que la mécanisation (plus précisément l'usage d'un pantographe associé à une fraiseuse) permet la fabrication de caractères en bois « variables » (voir la figure ci-dessous).

FIGURE 3 – Caractères en bois de même corps, mais de chasse variable (extrait de [3]).

Trois séries (*Roman*, *Antique* et *Gothic*, avec des variations de nom *XXX Condensed*, *XX Condensed*… *Expanded*, *Extended*) de caractères en bois américains, découpés à l'aide d'un pantographe à partir d'un modèle unique pour chaque série. Années 1880.

Mais c'est au milieu du XXe siècle que Frutiger présenta son fameux tableau des variations d'`Univers` (voir la figure 4 page suivante) où il apporta un premier effort de quantification, et l'usage d'« axes » pour désigner les variantes.

En résumé, les caractères en plomb, ou en bois, faisaient partie de familles composées souvent de variantes de types plus ou moins gras, plus ou moins étroits, avec un œil (mesuré par sa *x height* comme on dit…) plus ou moins gros, etc. Chaque variante étant alors elle-même déclinée en italique ou romain, et ce pour chaque corps. Physiquement, une fonte était une collection de tiroirs de casses.

Aujourd'hui, avec les caractères « numériques »

Comme pour les fontes en plomb, qu'elles aient été dessinées à la main, à l'aide d'outils (tels que FontLab, FontForge, Glyphs, etc.) ou programmées (par exemple avec METAFONT[3]), les fontes numériques[4] sont constituées de plusieurs fichiers, chaque variante de fonte numérique correspondant à un fichier spécifique.

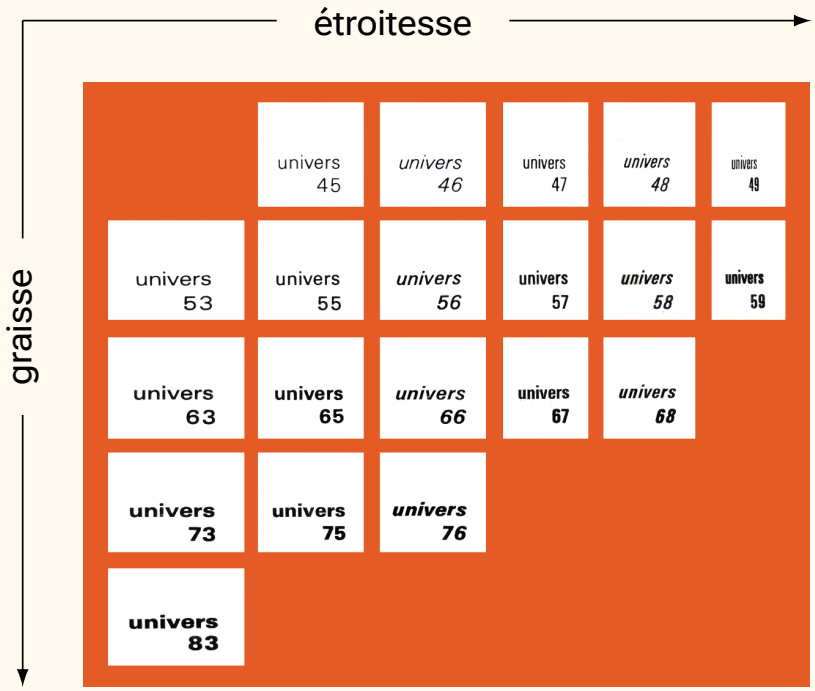
Mais, pour les graphistes « papier » et depuis peu pour ceux du web (notamment les utilisateurs de CSS), de nouveaux besoins[5] se sont fait sentir en matière de fontes (untel veut une graisse comme ci ou un œil comme ça, un autre tel un

3. Nous reviendrons sur le rapport entre METAFONT et les fontes variables dans la conclusion de cet article, en page 61.

4. On trouvera une histoire des fontes numériques dans mon article « Histoire technique des fontes numériques »[4]. Pour un aperçu des techniques actuelles des fontes numériques, l'ouvrage fondamental reste toujours, 20 ans plus tard, celui de Yannis Haralambous, *Fontes & codages* [5].

5. Parfois esthétiques, parfois techniques, mais aussi parfois non conformes aux habitudes

FIGURE 4 – Univers (Frutiger, 1968) : 21 variantes, 5 chasses et 5 graisses ; première « quantification » des « axes »



caractère plus ou moins condensé, etc.). On a alors commencé à voir des fontes offrant de plus en plus de graisses différentes (avec toute une terminologie, pas du tout normalisée, comme *black, bold, medium, regular, light, thin*, sans oublier leurs *extra-* et *semi-*), des variantes étroites ou élargies, des jambages plus ou moins restreints, etc. !

Ces fontes sont une réelle avancée mais, sans parler des problèmes de normalisation (notamment terminologiques), présentent des inconvénients :

1. La gestion de ces sous-fontes et l'adressage des glyphes (c'est-à-dire comment y accéder) ne sont pas toujours aisés. Comment, par exemple en LaTeX, étendre la commande `\textbf` pour spécifier les variantes *medium*, *semi-bold* ou *extra-black* d'un caractère ? Nous verrons en section « Sélection des fontes variables », page 22, comment résoudre ce problème, au demeurant indépendant des fontes variables.
2. Ces sous-fontes sont décidées par le dessinateur. Mais l'utilisateur peut avoir des besoins différents, par exemple un éditeur voudra un gras légèrement plus noir que le *semi-bold* proposé ; de même un maquettiste web (humain ou un programme) pourra avoir localement besoin d'agrandir la chasse d'un titre en prenant des caractères qui chassent légèrement plus… si le dessinateur l'a prévu.

C'est un peu cette liberté de choix que proposent aux utilisateurs les fontes variables. En fait, ce n'est pas entièrement nouveau : des tentatives allant dans ce sens se trouvaient déjà dans l'AAT [5] ou les *Multi-Masters* d'Adobe [6].

typographiques ! On attribue aux utilisateurs de CSS la demande de minimiser le nombre de fichiers de fontes ; ils ont été entendus puisqu'une fonte variable ne comprend plus qu'un seul fichier.

Attention !

Sauf mention contraire, les exemples qui vont suivre seront composés avec la fonte `Roboto Flex`, fournie par le fichier `RobotoFlex-VariableFont.ttf`. Elle sera chargée via la commande `\RobotoFlex` que nous avons définie par :

```
\newfontfamily\RobotoFlex{RobotoFlex-VariableFont.ttf}
```

Pour des questions de place :

- Le recours à ces commandes ne sera pas rappelé dans les exemples.
- Les modifications des caractéristiques par défaut de la fonte en cours le seront par le biais d'options passées en argument de la commande `\addfontfeatures`[a] qui sera souvent employée au moyen de son alias `\aff` (abréviation de *add font features*) que nous avons défini par :

  ```
  \NewCommandCopy\aff\addfontfeatures
  ```

- Une grande majorité de ces exemples vont utiliser la fonctionnalité `RawAxis` par le biais de :
  `\addfontfeatures{RawAxis={⟨fonctionnalité⟩}}`
  Aussi allons-nous fréquemment recourir à la commande `\axe` que nous avons définie par :

  ```
  \newcommand\axe[1]{\addfontfeatures{RawAxis={#1}}}
  ```

a. Les commandes `\newfontfamily` et `\addfontfeatures` sont gentiment fournies par le package fontspec (cf. section « Fontes variables et LaTeX » en page 15).

TABLEAU 1 – Choix de graisse (`wght`) et de largeur (`wdth`) de caractères

Code	Résultat
`uu`	uu
`\axe{wght=100}u\axe{wdth= 15}u`	uu
`\axe{wght=100}u\axe{wdth=150}u`	uu

Fontes variables : généralités

Le concept de fontes variables existe depuis 2016 comme extension aux fonctionnalités des fontes OpenType. Officiellement, OpenType[6] appelle « *Variable fonts [the] fonts that use OpenType Font Variations mechanisms* » (« [on appelle] fontes variables des fontes qui utilisent des mécanismes de variations de fonte de OpenType »). Tout est dans ce mot « variations »...

6. Le format OpenType a été créé par Adobe et Microsoft à partir du format TrueType (de Microsoft et Apple) et de PostScript d'Adobe. Depuis 2007 OpenType est la norme ISO/CEI 14496-22 sous le nom "*Open Font Format*" (`https://www.iso.org/standard/74461.html`). Microsoft est propriétaire du nom et joue encore le leader. Voir [7]. Les fontes variables sont plus précisément définies dans [8]. Outre les milliers d'articles sur le web, on lira avec profit l'introduction aux fontes variables de John Hudson [9].

Choix de variantes de graisse et de largeur des caractères

Comme l'avait fait Frutiger (voir la figure 4 page 5), les variantes (on dit les axes) les plus utilisées par les fontes variables concernent la graisse ou la chasse (ou largeur) des caractères. Grâce aux commandes du type `\addfontfeatures{RawFeature=sups}`, LualaTeX permet de faire des substitutions de glyphes ou de fixer des positionnements de glyphes avec des fontes OpenType ; il en est de même pour le choix de variantes grâce à des commandes du type `\addfontfeatures{RawFeature={+axis={wdth=75}}}` où on demande d'utiliser une `wdth` (*width* = chasse) de 75 (sous-entendu 75 % de la chasse normale — on aura souvent l'occasion de revenir sur la mesure des variations)[7].

Dans le tableau 1 page ci-contre, on imprime (deux fois) le « u normal » (avec les valeurs par défaut des axes) de la fonte RobotoFlex, puis avec les valeurs minimales de l'axe graisse (`wght`) et de l'axe largeur (`wdth`) et enfin les valeurs maximales de ces axes. Nous verrons en page 14, dans la section « Inspecter une fonte variable », comment trouver ces valeurs : minimales, maximales et par défaut.

L'exemple 1, ci-après, montre que l'on peut modifier deux axes (et même plus de façon générale) en même temps. Notons aussi que les paramètres passés aux fonctions `wght=` et `wdth=` peuvent être des variables (ici respectivement `\i` et `\j`) et non des constantes comme dans le tableau 1 page précédente. C'est tout l'intérêt des fontes variables : les calculs nécessaires sont effectués non pas au chargement, comme pour les fontes statiques, mais à l'exécution (au *run time*). Nous y reviendrons dans la section « Fontes variables et LualaTeX » en page 19.

Exemple 1 – Combinaison de variantes selon deux « axes »

code

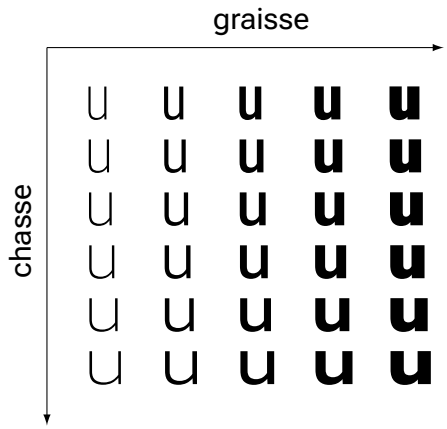
```
% Requiert le package `pgffor`
\centering
\setlength{\unitlength}{1cm}
\begin{picture}(8,6)
  \put(0,5.25){\vector(1,0){5.25}}
  \put(0,5.25){\vector(0,-1){5}}
  \put(2.2,5.5){graisse}
  \put(-.5,2.2){\rotatebox{90}{chasse}}
  \put(0,5){%
    \begin{picture}(8,6)
      \foreach \i in{100,300,...,900}{
        \foreach\j in{25,50,...,150}{
          \addfontfeatures{%
            RawAxis = {wght={\i}, wdth={\j}}%
          }%
          \put(.05mm*\i,-.28mm*\j){\Huge\makebox[1em][l
    ]{u}}%
        }%
      }
```

7. Il est à noter qu'il n'y a pas de vérification d'erreurs lors de l'utilisation de `RawFeature`. Lorsqu'une interface fontspec existe pour une fonctionnalité, il est généralement préférable de l'utiliser. Si la police ne dispose pas de la fonctionnalité ou si celle-ci entre en conflit avec une autre fonctionnalité, fontspec tentera d'avertir et/ou de résoudre les problèmes. C'est typiquement le cas ici où il serait préférable de recourir à `\addfontfeatures{RawAxis={wdth=75}}`.

code (suite)

```
    \end{picture}%
  }
\end{picture}
```

résultat



Principes des fontes variables

Fontes statiques

On appelle « fontes statiques » nos fontes habituelles, qui ne sont pas des fontes variables. Voici en gros leur principe de fonctionnement[8].

Une fonte est formée d'un ensemble de caractères qui est structuré (sous forme d'arbre, ou plutôt de forêt, dans le cas des fontes OpenType ; ou sous forme de tables et fichiers divers, comme pour les fontes TEX vues par NFSS) de façon à accéder aux caractères de diverses manières : par leur nom, leur code, leurs propriétés (supérieures, petites capitales, ligatures, terminales, etc.), etc. Chaque caractère y est lui-même décrit :

— par un programme décrivant son glyphe sous forme de contours ;
— et par la liste de ses propriétés (métriques, comme la chasse ; ou de position, par rapport à la ligne de base par exemple).

Exemple 2 – Description de la lettre « u » en MetaPost

code

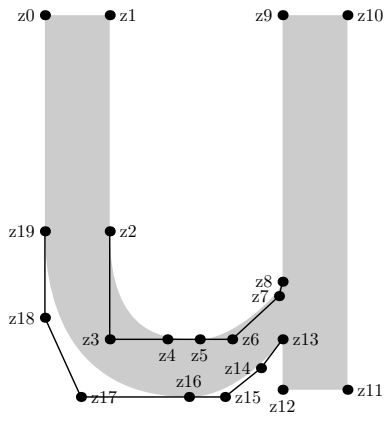
```
% Requiert les packages `luamplib` et `graphicx`
\scalebox{2.7}{\begin{mplibcode}
def u = begingroup

dotlabeldiam:=1.4bp;
labeloffset:=.50mm;
defaultscale:=.25;
numeric x; x:=10;

 z0=(0,67)   ;   z1=(9,67)     ;   z2=(9,37)     ;
```

8. On trouvera dans [5, 10, 11] des introductions utiles sur ce sujet, ou plus encore.

code (suite)

```
 z3=(9,22)  ;  z4=(17,22)   ;  z5=(21.5,22) ;
 z6=(26,22) ;  z7=(32.5,28) ;  z8=(33,30)   ;
 z9=(33,67) ; z10=(42,67)   ; z11=(42,15)   ;
z12=(33,15) ; z13=(33,22)   ; z14=(30,18)   ;
z15=(25,14) ; z16=(20,14)   ; z17=(5,14)    ;
z18=(0,25)  ; z19=(0,37)    ;

fill z0--z1--z2
.. controls z3 and z4 .. z5
.. controls z6 and z7 .. z8
--z9--z10--z11--z12--z13
..controls z14 and z15 .. z16
..controls z17 and z18 .. z19--cycle
withcolor .8white;

dotlabel.lft("z0",z0)   ; dotlabel.rt("z1",z1)    ;
dotlabel.rt("z2",z2)    ; dotlabel.lft("z3",z3)   ;
dotlabel.bot("z4",z4)   ; dotlabel.bot("z5",z5)   ;
dotlabel.rt("z6",z6)    ; dotlabel.lft("z7",z7)   ;
dotlabel.lft("z8",z8)   ; dotlabel.lft("z9",z9)   ;
dotlabel.rt("z10",z10)  ; dotlabel.rt("z11",z11)  ;
dotlabel.bot("z12",z12) ; dotlabel.rt("z13",z13)  ;
dotlabel.lft("z14",z14) ; dotlabel.rt("z15",z15)  ;
dotlabel.top("z16",z16) ; dotlabel.rt("z17",z17)  ;
dotlabel.lft("z18",z18) ; dotlabel.lft("z19",z19) ;

pickup pencircle scaled .2pt;
draw z2--z3--z4--z5--z6--z7--z8;
draw z13--z14--z15--z16--z17--z18--z19;

endgroup;enddef;

beginfig(1) u endfig;
\end{mplibcode}}
```

résultat



Le programme décrivant un glyphe par ses contours, à l'aide des courbes de Bézier, ressemble à celui donné (en MetaPost pour les habitués du monde TEX) en

page 8, ici pour la lettre « u ». Ce contour est alors mis à l'échelle en fonction du corps demandé et projeté sur la grille correspondant à l'imprimante ou à l'écran cible, et rempli pixel par pixel. Cette opération de remplissage nécessite un certain savoir-faire[9] pour obtenir un bon rendu. En effet, selon notamment la position du caractère par rapport à la grille, les deux barres verticales de notre « u » pourraient avoir des épaisseurs variables, ce qui choquerait l'œil !

Fontes paramétrées

On a dénommé « statique » la fonte dont on a montré la définition des contours pour le caractère « u » car, si on regarde bien la définition des points, on n'y utilise que des coordonnées statiques, des valeurs numériques constantes (0, 67, 9, 22, etc.). On peut imaginer de faire des « u » plus larges, qui chassent plus, avec des valeurs plus grandes pour les abscisses des points de la branche de droite, et même définir une fonction linéaire[10] qui décrivent ces variations, par exemple définir des points avec des coordonnées du type $z = (a + bx, c)$ où a, b, c sont des constantes numériques. L'exemple 3 montre ainsi deux « u » légèrement différents (le second chasse plus que le premier) selon la valeur donnée à x.

C'est la méthode employée par les programmes qui ont permis de produire des familles de fontes, dont bien sûr METAFONT pour les `Computer Modern`.

Mais… les imprimantes et autres RIP (qui produisent des images tramées) n'acceptent que des constantes dans les équations des courbes[11]. La solution a été alors de fournir autant de fontes « statiques » que possible ; mais c'était le choix du dessinateur et non celui de l'utilisateur.

Exemple 3 – Description paramétrée de la lettre « u » en MetaPost pour avoir deux chasses différentes

code

```
\setlength{\unitlength}{0.1\textwidth}
\scalebox{1.5}{\begin{mplibcode}
def u(expr x)=begingroup

dotlabeldiam:=1.4bp;
labeloffset:=.50mm;
defaultscale:=.25;

z0=(0,67); z1=(9,67); z2=(9,37); z3=(9,22);
z4=(17,22); z5=(21.5,22); z6=(26+x,22);
z7=(32.5+x,28); z8=(33+x,30); z9=(33+x,67);
z10=(42+x,67); z11=(42+x,15); z12=(33+x,15);
z13=(33+x,22); z14= (30+x,18); z15=(25+x,14);
```

9. Les premiers systèmes de gestion de fontes, que ce soit PostScript, TEX, Ikarus, etc. étaient discrets, voire secrets, sur ces « bricolages ». Aujourd'hui certaines firmes sont spécialisées dans ce travail, comme Harfbuzz dont on reverra le nom ci-après, justement à propos des paramètres `rendering=harfbuzz`!

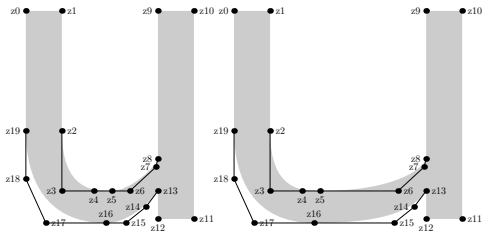
10. Voire des fonctions plus complexes ; on en trouve dans des fontes écrites en METAFONT, p. ex. dans [5].

11. Pour des raisons de temps de traitement mais aussi de passage de paramètres. En fait, la première LaserWriter de Canon, pilotée par PostScript, le permettait en désactivant un mécanisme de cache qui évitait de recalculer deux fois le même caractère. Nous avions montré, avec Victor Ostromoukhov, qu'alors on pouvait ainsi rendre complètement aléatoire l'impression de chaque occurrence des glyphes de la fonte *Punk* de Knuth [12].

code (suite)

```
14  z16=(20,14); z17=(5,14); z18=(0,25); z19=(0,37);
15
16  fill z0--z1--z2
17  .. controls z3 and z4 .. z5
18  .. controls z6 and z7.. z8
19  --z9 -- z10--z11--z12--z13
20  ..controls z14 and z15 .. z16
21  ..controls z17 and z18 .. z19--cycle withcolor .8white;
22
23  dotlabel.lft("z0",z0); dotlabel.rt("z1",z1);
24  dotlabel.rt("z2",z2); dotlabel.lft("z3",z3);
25  dotlabel.bot("z4",z4); dotlabel.bot("z5",z5);
26  dotlabel.rt("z6",z6); dotlabel.lft("z7",z7);
27  dotlabel.lft("z8",z8); dotlabel.lft("z9",z9);
28  dotlabel.rt("z10",z10); dotlabel.rt("z11",z11);
29  dotlabel.bot("z12",z12); dotlabel.rt("z13",z13);
30  dotlabel.lft("z14",z14); dotlabel.rt("z15",z15);
31  dotlabel.top("z16",z16); dotlabel.rt("z17",z17);
32  dotlabel.lft("z18",z18); dotlabel.lft("z19",z19);
33
34  pickup pencircle scaled .2pt;
35  draw z2--z3--z4--z5--z6--z7--z8;
36  draw z13--z14--z15--z16--z17--z18--z19;
37
38  endgroup;enddef;
39
40  beginfig(0) u(0); endfig;
41  beginfig(1) u(15); endfig;
42  \end{mplibcode}}
```

résultat



Le système Panose et Infinifont

Le système Panose [13] a été créé en 1985 par Benjamin Bauermeister. Au départ, c'était un système de classification des fontes (il est bien décrit dans *Fontes & Codages* [5, p. 418-432]) dont plusieurs paramètres sont réutilisés pour les valeurs d'axes des fontes variables [12]. Bauermeister a créé aussi une société, ElseWare Corporation, et le système Infinifont, qui était un système interactif de création de fontes très paramétrées. Comme avec METAFONT, les fontes produites étaient statiques. Infinifont a permis de produire des centaines de clones de fontes.

12. Le système Panose a été adopté par les CSS, notamment pour la graisse. Les CSS ont ainsi défini les valeurs 100 à 900 que l'on va voir et normalisé les noms comme *Light, Bold*, etc. que l'on trouve actuellement dans la majorité des fontes variables.

Multi-Masters d'Adobe

Vers 1990, Adobe a proposé, et implémenté notamment pour la fonte `Minion`[13], le concept de *Multi-Masters* (MM) qui est à la base des fontes variables. Le processus est le suivant (on suppose vouloir dessiner une fonte avec variation de la chasse et on montre donc ce qui se passe pour notre lettre « u ») et est illustré en figure 5 :

1. Puisque les dessinateurs de caractères n'aiment pas programmer, on utilise des outils basés sur l'emploi de la souris.
2. Le dessinateur, travaillant donc sur la chasse des caractères, dessine (dans un corps 1000 par exemple) le « u » le plus étroit et celui le plus large possibles. Il donne aussi une valeur numérique à ces extrêmes, *min* (par exemple 100) pour le minimum et *max* (p. ex. 800) pour le maximum.
3. Il indique, à la souris, quels sont les points de contrôle qui sont liés dans les deux extrêmes (il réunit les points zi_{min} à zi_{max}) et le pas de parcours de cet intervalle.
4. Pour chaque valeur du pas (que l'on appelle *act* pour cet exemple) et par interpolation linéaire, le système calcule facilement les abscisse x_{act} et ordonnée y_{act} de chaque point *zi*.
5. Le système calcule aussi la chasse de ce caractère et la met dans le fichier TFM, ou l'équivalent correspondant.
6. Ces calculs sont faits pour tous les points de tous les caractères de la fonte et les valeurs actualisées de la description de leurs contours sont donc enregistrées dans les tables comme TFM, et ce pour tous les pas. Ce qui revient à définir une collection de fontes statiques, pour chaque chasse *act* spécifique.
7. Ces fontes sont alors chargées et utilisées comme des fontes normales.
8. Pour employer une fonte spécifique, l'utilisateur doit donner une valeur *act* comprise entre *min* et *max* pour indiquer quelle variation (ici sur la chasse) il veut, c'est-à-dire quelle sous-fonte il veut.

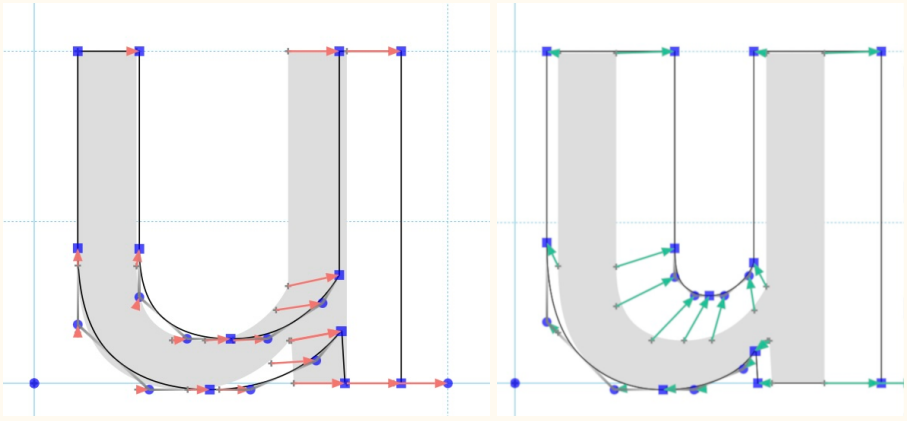


FIGURE 5 – *Masters* (ici le minimal est en gris, le maximal dessiné par contours) et vecteurs définissant les variations du « u » d'une fonte selon l'axe *width* (à gauche) ou *weight* (à droite).

Le procédé des *Multi-Masters* n'a pas eu un gros succès. Outre sa spécificité « Adobe » (en pleine guerre des fontes), le temps de calcul et la place prise en mémoire (du fait que toutes les occurrences possibles d'une fonte MM étaient calculées et chargées avant le déroulement du texte source) ont freiné les utilisateurs ; `Minion-MM` et les (rares) autres fontes *Multi-Masters* ont été oubliées...

13. Nous vous renvoyons à l'article de Thierry Bouche, dans les *Cahiers* GUTenberg, qui lui est consacré. [6]

Fontes variables

Le principe des *Multi-Masters* a été repris pour les fontes variables OpenType, accéléré et même étendu à d'autres variations tout en permettant leur cumul. La principale différence est que chaque sous-fonte statique (pour une valeur donnée d'un axe) n'est calculée qu'au moment de l'exécution (*run time*). Par ailleurs, les vecteurs définissant les champs de variations des points de contrôle peuvent avoir des orientations dépendant de l'espace où ils sont, divers quadrants étant prédéfinis. Nous n'en dirons pas plus ici (voir [14]). Précisons seulement que le travail des dessinateurs est désormais bien plus complexe, nécessitant notamment une très grande rigueur (par exemple pour l'ordre dans lequel les points extrêmes sont reliés entre eux !).

Mais contrairement à ce que nous avons appelé fontes dynamiques (où chaque glyphe est recalculé au moment de son utilisation), les fontes variables sont calculées au *run time* de façon globale (une variation donnée s'applique à tous les glyphes).

Quels axes ?

Nous avons vu, en section « Fontes variables : généralités », page 6, qu'on appelle « axe » un type de variation subie par une fonte à la demande d'un utilisateur. Ainsi, faire varier la graisse correspond à l'axe *weight*, faire varier la chasse à l'axe *width*, faire varier l'inclinaison à l'axe *slant*, etc. Il s'agit donc d'une figure de style, à notre avis un peu abstruse, mais à laquelle nous nous plierons ; nous parlerons souvent d'*axe* pour désigner la variation elle-même.

Les fontes variables d'OpenType définissent 5 axes (à chacun desquels on donne un nom de code, formé de 4 lettres en minuscules), appelés les *registered axes* (axes officiels) :

`wght` (*weight*) : variations de graisse ;
`wdth` (*width*) : variations de chasse ;
`slnt` (*slant*) : penché ;
`ital` (*italic*) : italique.
`opsz` (*optical size*) : ajustement optique ;

Nous revenons sur ces axes en section « Les cinq axes principaux et RobotoFlex », page 27. Remarquons dès à présent que les trois premiers axes couvrent des modifications de glyphes, difficiles à programmer en LaTeX, tandis que les deux derniers font partie des possibilités basiques de LaTeX, en ce sens qu'ils concernent globalement la fonte.

À ces cinq axes, chaque créateur de fonte peut en ajouter d'autres ; leur nom de code est alors donné en capitales. On les appelle *customs axes* (axes privés). Notre fonte RobotoFlex en propose neuf (en plus des officiels) : `XTRA`, `XOPQ`, `YOPQ`, `YTAS`, `YTDE`, `YTUC`, `YTLC`, `YTFI` et `GRAD`. Nous y revenons en détail en section « Les autres axes de RobotoFlex », page 40 et en annexe, notre exemple 30 page 69 indique comment nous avons obtenu la liste de ces axes.

Globalement, on trouve une centaine d'axes différents proposés par les fontes variables existant aujourd'hui, dont certains sont déjà devenus des standards *de facto* (voir la section « Autres axes » en page 48).

Valeurs des axes

À chaque axe est affecté un paramètre muni d'une « valeur » qui va justement permettre la variation de cet axe. Par exemple, à l'axe de graisse est affecté le paramètre `wght` (*weigth*). La valeur est choisie par l'utilisateur de la fonte, sinon c'est la valeur par défaut. Ces valeurs ne sont absolument pas définies par OpenType, mais pour un axe donné d'une fonte donnée, elles doivent être comprises entre une valeur minimale et une maximale ; ces deux valeurs constituent avec la valeur par défaut un triplet qui fait partie des caractéristiques de la fonte (et auxquelles l'utilisateur a accès, comme le montre la section « Inspecter une fonte variable » en de la présente page) :

> Triplet d'axe : *(valeur minimale, par défaut, maximale)*

Selon les fontes, ces valeurs peuvent représenter des concepts différents. Par exemple pour l'axe `ital`, certaines fontes proposent une valeur booléenne (0 ou 1) spécifiant simplement si la fonte est à mettre en italiques ou pas ; tandis que d'autres supposent que la valeur définit l'angle d'inclinaison des italiques ; certaines proposent même une combinaison de l'angle des italiques et de choix de glyphe, comme dans le tableau 4 page 17.

Notons bien que, mis à part les angles d'inclinaison (donnés pour `slnt` et `ital`, en degrés), ces valeurs ne sont pas des mesures et s'expriment donc sans unité.

Toutefois, pour certains axes, des valeurs sont « recommandées » par certains organismes. C'est ainsi que, par exemple, le W3C a défini diverses normes pour les CSS [15], dans lesquelles on donne une échelle pour noter la graisse avec les paires suivantes :

(100,*thin*)
…
(400,*regular*)
…
(700,*bold*)
…
(900,*black*)

Ces valeurs ont été adoptées (et même étendues) par de nombreuses fontes variables (dont RobotoFlex, voir la section « Axe `wght` : graisse (*weight*) » en page 27), sans que ce soit une norme ! En fait, Glyphs (le logiciel de dessin de fontes le plus utilisé aujourd'hui) donne par défaut les valeurs des CSS pour l'axe de graisse [16]. On note une tendance générale (tant chez les dessinateurs que chez les utilisateurs) à faire correspondre des noms à des valeurs numériques (par exemple *bold* à 700 ou, nous le verrons plus bas, *condensed* à 75 %, etc.). Ces noms vont notamment servir à caractériser des occurrences prédéfinies d'une fonte spécifique (voir la section « Instances » en page 17). Par exemple, la fonte variable `Inconsolata` propose une série de « sous-fontes » dont :

`Inconsolata-Condensed-Bold` avec `wdth=75`, `wght=700`
`Inconsolata-SemiCondensed-ExtraLight` avec `wdth=87.5`, `wght=200`

Inspecter une fonte variable

Si la grande majorité des fontes variables propose aujourd'hui au moins les deux axes `wdth` (*width*) et `wght` (*weight*), chaque fonte a sa propre liste d'axes. Un peu comme chaque fonte OpenType a sa propre liste de caractéristiques, liste que l'on peut connaître (voir [17]) par la consultation de logiciels de gestion de fontes (comme FontForge), par programme ou tout simplement par consultation de la notice fournie par la « fonderie ».

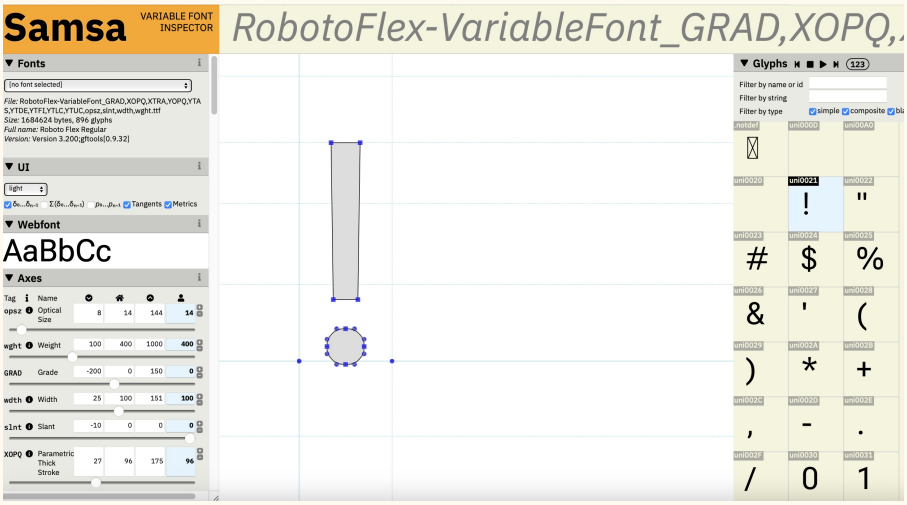


FIGURE 6 – Inspection d'une fonte variable, ici RobotoFlex, par Samsa.

Ces outils sont utilisables pour les fontes variables ; mais de nouveaux outils d'analyse de fontes – appelés *font inspector* ou *font viewer and tester*, voire *tool for proofing, generating and animating fonts* – sont disponibles, le plus souvent en libre, et permettent d'afficher les divers axes des fontes variables, de les faire varier et de montrer les contours correspondants des glyphes.

Par exemple, la figure 6 montre ce qu'affiche Samsa Font Inspector[14] pour la fonte RobotoFlex :

- la colonne de gauche liste les divers axes et donne les triplets (valeurs minimale, par défaut et maximale) de chacun d'entre eux, avec un curseur que l'on peut déplacer et qui fait varier à volonté les formes du glyphe affiché dans le carré central) ;
- dans le carré central est affiché le glyphe (sélectionné à droite) en présentant ses contours et points de contrôle (en bougeant un ou plusieurs curseurs de la colonne de gauche, on voit les modifications apportées à ces contours).
- enfin, la colonne de droite affiche les différents glyphes de la fonte.

Nous verrons plus bas (en section « Programme Lua d'extraction des triplets des axes d'une fonte variable », page 69) comment disposer de ces mêmes informations avec `luacode`.

Fontes variables et LaTeX

LaTeX permet l'utilisation des fontes variables grâce à l'extension fontspec de LuaLaTeX[15].

Syntaxe des commandes

La commande principale d'appel de fonte est [18] :

```
\fontspec{⟨font⟩}[⟨font features⟩]
```

14. https://www.axis-praxis.org/samsa/. Plusieurs de nos figures (p. ex. figure 5 page 12) utilisent des affichages de ce logiciel.
15. Les mécanismes de gestion des fontes variables de XeLaTeX sont très différents ; nous n'en parlerons pas ici.

avec (de façon simplifiée) :

```
⟨font features⟩ = [RawAxis={⟨axis⟩=⟨value⟩}]
```

Cette instruction appelle l'occurrence de la fonte variable ⟨font⟩ calculée avec la valeur ⟨value⟩ pour l'axe ⟨axis⟩.

TABLEAU 2 – Syntaxe des commandes fontspec d'utilisation d'un axe de fonte variable

Code	Résultat
`g`	g
`\addfontfeatures{RawAxis={wght=800}} g`	**g**
`\addfontfeatures{Weight=800} g`	**g**
`\addfontfeatures{Width=500} g`	g
`\addfontfeatures{Slant=-10} g`	*g*
`\addfontfeatures{OpticalSize=11} g`	g

Nous avons déjà illustré cette syntaxe (voir le tableau 1 page 6) et nous revenons ci-dessous sur celle du tableau 2 :

```
\addfontfeatures{RawAxis={wght=800}}
```

dont l'argument :

```
RawAxis={wght=800}
```

peut se raccourcir en `Weight=800` s'il s'agit de l'un des axes officiels ! Et de même pour les quatre autres, bien sûr.

Comme on l'a vu (cf. l'exemple 1 page 7), on peut combiner plusieurs axes dans une même commande. Le tableau 3 page suivante montre également l'emploi de `\addfontfeatures`[16] dans ce cas. Comme indiqué à la note 7 page 7, on y constate que recourir à `RawAxis=` est préférable à `RawFeature={+axis=` : en effet, avec l'option `RawFeature`, fontspec transmet simplement les propriétés telles quelles au chargeur de polices sous-jacent, ici luaotfload, sans les interpréter davantage donc sans tenir compte du fait que les deux paramètres s'appliquent à l'axe de la police ; or luaotfload n'accepte qu'un seul paramètre `axis` par police, de sorte que les deux paramètres se remplacent mutuellement et que le deuxième `g` est condensé (largeur 50), mais n'est plus extra-gras. Par conséquent, en utilisant `RawFeature`, les paramètres de largeur ont commencé à influencer ceux du poids.

Rappelons que, dans cet article :

- `\addfontfeatures` ;
- `\addfontfeatures{RawAxis={⟨fonctionnalité⟩}}` ;

seront souvent raccourcis en, respectivement :

- `\aff` ;
- `\axe{⟨fonctionnalité⟩}`.

16. Dont `\addfontfeature` est un alias.

Tableau 3 – Commandes de combinaison d'axes

Code	Résultat
`\aff{RawAxis={wght=800,wdth=50}} g`	**g**
`\aff{RawFeature={+axis={wght=800}}} g` `\aff{RawFeature={+axis={wdth= 50}}} g`	**g** g
`\aff{RawAxis={wght=800}} g` `\aff{RawAxis={wdth= 50}} g`	**g** **g**

Problèmes de rendu

Nous avons vu (cf. note 9 page 10) que le bon positionnement des pixels des fontes numériques nécessitait un « moteur de rendu », dont la société Harfbuzz s'est fait une spécialité. Elle a ainsi été parmi les premières a bien implémenter les fontes OpenType — notamment variables. Depuis 2019, LualaTeX utilise ce moteur par défaut. Mais, dans certains cas[17], il faut utiliser l'argument `Renderer=OpenType`. Le tableau 4 montre le cas d'une fonte, `Brito`, dont l'italique ne marche que si on emploie cette option (en dernière ligne, les glyphes sont bien penchés et cursifs). Précisons que ce n'est pas la fonte qui est en cause (elle fonctionne sans problèmes avec d'autres moteurs), mais bien LualaTeX. Dès que l'on voit une anomalie, il faut voir si elle persiste avec cette option…

Tableau 4 – Variation d'italique. L'inclinaison peut être choisie par l'utilisateur. À partir d'un certain angle, certains glyphes sont modifiés (ici « y » et « e »). Cet exemple ne fonctionne bien qu'avec l'option `Renderer=OpenType`. Fonte utilisée ici : `BritoVariableVF.ttf`.

Code	Résultat
`paye`	paye
`\axe{ital=.30} paye`	paye
`\axe{ital=.60} paye`	paye
`\axe{ital=.60}` `\aff{Renderer=OpenType} paye`	paye

Instances

C'est désormais l'utilisateur qui va choisir la graisse qu'il veut voir donner à une fonte. Mais le dessinateur propose des valeurs qui lui conviennent ou qui sont conventionnelles. Il peut s'agir des valeurs de graisse, mais aussi d'italique, voire de chasse (caractères étroits ou élargis), ou combinées, etc. Les sous-fontes créées avec ces valeurs des axes sont appelées *instances*.

Par exemple, RobotoFlex propose les instances données dans le tableau 5 page suivante. La liste des instances est donnée par un inspecteur de fontes, tel Samsa : avec cet outil, on l'obtient en déroulant la colonne de gauche (voir la figure 6 page 15).

17. Sans donner de vraies explications, ceci est noté dans le *fontspec manual* [18, p. 56] ; lui même cité par https://faq.gutenberg-asso.fr/3_composition/texte/symboles/polices/moteurs_rendu.html.

TABLEAU 5 – Les instances de RobotoFlex. Chaque nom a été composé par une commande comme : `{\fontspec{\RobotoFlex}[Instance=Light] Light}`

Thin	*Thin Italic*
Extra Light	*Extra Light Italic*
Light	*Light Italic*
Regular	*Regular Italic*
Medium	***Medium Italic***
Semi Bold	***Semi Bold Italic***
Bold	***Bold Italic***
Extra bold	***Extra bold Italic***
Black	***Black Italic***
ExtraBlack	***ExtraBlack Italic***

L'option `Instance=Bold` est équivalente à `Weight=700`, la valeur 700 étant définie par le dessinateur. Notons que cette valeur est accessible par otfinfo dans les tables de la fonte, comme le montre l'exemple suivant :

```
$ otfinfo --variable AROneSans-VariableFont_ARRR,wght.ttf

Axis 0:                 ARRR
Axis 0 name:            AR Retinal Resolution
Axis 0 range:           10 60
Axis 0 default:         10
Axis 1:                 wght
Axis 1 name:            Weight
Axis 1 range:           400 700
Axis 1 default:         400
Instance 0 name:        Regular
Instance 0 position:    10 400
Instance 1 name:        Medium
Instance 1 position:    10 500
Instance 2 name:        SemiBold
Instance 2 position:    10 600
Instance 3 name:        Bold
Instance 3 position:    10 700
```

Souvent (ce n'est pas le cas de RobotoFlex) les fontes variables sont distribuées dans un répertoire contenant notamment :

— le fichier unique de la fonte variable (et parfois un second lorsque l'italique est considérée comme une fonte à part, c'est par exemple le cas de `EBgaramond-variable`);
— un répertoire, généralement appelé `Static`, contenant un jeu de fichiers, chacun correspondant à une instance de la fonte variable.

Cette possibilité n'est bien sûr pas réservée à la graisse et on trouve des fontes statiques basées sur les valeurs d'un autre axe, notamment `wdth`, voire de plusieurs axes à la fois. Par exemple, la nouvelle fonte `Brito`[18] comprend la fonte variable

18. Voir https://www.skritur.eu/fonts/brito.

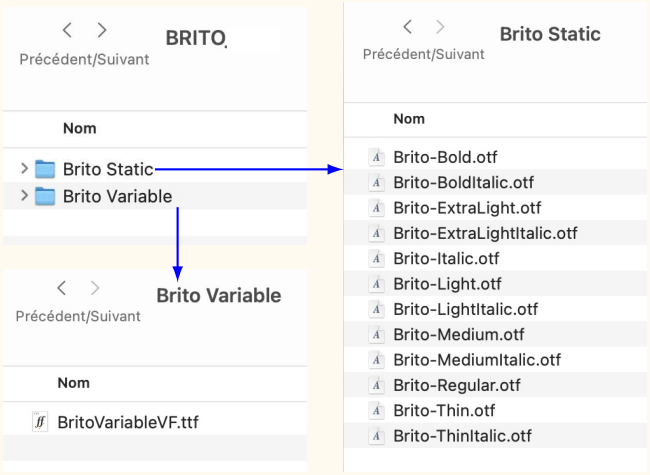


FIGURE 7 – La distribution d'une fonte variable comprend en général la fonte variable et ses instances sous forme de fontes statiques. Ici le cas de `BritoVariableVF`.

`BritoVariableVF.ttf` et la douzaine de fontes statiques qui sont donc des instances de la fonte variable pour des valeurs prédéfinies de `wght` et de `ital` (voir la figure 7).

Fontes variables et LuaLaTeX

Rappelons [17] que lorsqu'il traite un fichier source, LuaLaTeX (plus précisément le module luaotfload) traduit, dès la mention d'une fonte (dans une commande `\fontspec{RobotoFlex}`, `\setmainfont{RobotoFlex}`, etc.), l'arborescence de la fonte fournie sous forme OpenType vers une forme, également arborescente, écrite en Lua. Cette fonte est en fait mise sous forme de deux fichiers, l'un `RobotoFlex.luc` comprenant les programmes décrivant les contours de chaque glyphe, et l'autre `RobotoFlex.lua` reprenant les tables GSUB, GPOS, etc. (un peu comme les fichiers `.tfm` et `.ps`). Voir à ce sujet l'annexe de la page 63. Ces deux fichiers sont mis en mémoire cache de l'utilisateur[19] de façon à les y retrouver lors d'une prochaine session. Ces fichiers sont ensuite traduits en une série de fichiers compatibles avec NFSS, les appels à des caractères de cette fonte se faisant alors avec la machinerie des fontes LaTeX.

Exemple 4 – exécution d'un programme à fonte variable ; extrait du fichier `.log` en page suivante

code

```
\fontspec{SUSE-VariableFont_wght.ttf}
\ExplSyntaxOn
\newlength{\chA}
% \l_tmpa_int : numéro d'essai (entre 1 et 4)
% \l_tmpb_int : nombre aléatoire entre 0 et 800
\int_do_until:nNnn {\l_tmpa_int} = {4} {
  \int_incr:N \l_tmpa_int
  \int_set:NV \l_tmpb_int {\int_rand:nn {0}{800}}
```

19. Dans le cas de macOS, ces fichiers sont dans `/Library/texlive/2025/texmf-var/luatex-cache/generic/fonts/otl`. Si on a appelé la fonte `EBGaramond`, on va y trouver les fichiers `ebgaramond-bold.lua`, `ebgaramond-bold.luc`, `ebgaramond-bolditalic.lua`, `ebgaramond-bolditalic.luc`, `ebgaramond-initials.lua`, `ebgaramond-initials.luc`, etc.

code (suite)

```
  \typeout{
    i=\int_to_arabic:n {\l_tmpa_int},~
    random=\int_to_arabic:n {\l_tmpb_int}
  }
  {\aff{
      Weight={\int_to_arabic:n {\l_tmpb_int}}
    }
    wght=\int_to_arabic:n {\l_tmpb_int},~ A
    \settowidth{\chA}{A}~ chasse~ A=\the\chA
  }\\
}
\ExplSyntaxOff
```

résultat

wght=786, A chasse A=11.95776pt
wght=206, A chasse A=11.0592pt
wght=541, A chasse A=11.5776pt
wght=256, A chasse A=11.12833pt

Le fichier `.log` de la présente page, généré à l'exécution de code de l'exemple 4 page précédente, est explicite ; il indique bien la création des fontes en question.

```
i=1, random=786
...
Package fontspec Info:
(fontspec)              Font family 'SUSE.ttf(0)' created
(fontspec)              for font 'SUSE.ttf' with options
(fontspec)              [Weight={786}].
(fontspec)
(fontspec)              This font family consists of the
(fontspec)              following NFSS series/shapes:
(fontspec)
(fontspec)              - 'normal' (m/n) with NFSS spec.:
(fontspec)              <->"[SUSE.ttf]:mode=node;script=
(fontspec)              latn;language=dflt;Weight={786};"
...
i=2, random=206
...
Package fontspec Info:
(fontspec)              Font family 'SUSE.ttf(1)' created
(fontspec)              for font 'SUSE.ttf' with options
(fontspec)              [Weight={206}].
...
i=3, random=541
...
Package fontspec Info:
(fontspec)              Font family 'SUSE.ttf(2)' created
```

```
(fontspec)              for font 'SUSE.ttf' with options
(fontspec)              [Weight={541}].
...
i=4, random=256
...
Package fontspec Info:
(fontspec)             Font family 'SUSE.ttf(3)' created
(fontspec)             for font 'SUSE.ttf' with options
(fontspec)             [Weight={256}].
...
```

Il en est de même avec une fonte variable, mais cette fois il n'y a que deux fichiers résultants, puisque les fontes variables ne sont formées que d'une seule fonte OpenType. Ces deux fichiers, dans le cas de notre fonte RobotoFlex, sont `robotoflex-variablefont.lua` et `robotoflex-variablefont.luc`; on les trouve donc dans notre cache.

Mais c'est lors de l'utilisation d'une fonte variable, par exemple lors d'une instruction `\fontspec{RobotoFlex}[Rawfeature={axis={wdth=100}}]`, que se passent tous les calculs pour dessiner les caractères avec les valeurs demandées (ici `100`) pour les axes (ici seulement `wdth`). Le calcul nécessite d'évaluer les coordonnées de chaque point de contrôle du contour d'un caractère en fonction des points des « masters » (extrêmes minimal et maximal, voir la figure 5 page 12) et de la valeur demandée pour l'axe, ici 100. Le calcul revient à une règle de trois[20], mais il faut le faire pour chaque point d'un caractère, en fonction des quadrants et pour tous les caractères de la fonte. Le calcul est encore plus complexe si on fait en même temps des variations sur plusieurs axes.

Enfin, il faut aussi modifier certaines valeurs de la métrique des caractères (dans notre exemple de la figure 5 page 12, il faut modifier la chasse du caractère « u », et des autres). Les fichiers `.lua` et `.luc` sont modifiés avec les valeurs ainsi calculées, qui sont enregistrées en mémoire cache[21], ce qui permet de ne pas refaire les calculs en cas de réutilisation de la même fonte.

Insistons sur le fait que ces calculs se font pendant l'exécution du programme, et non à l'avance. L'exemple 4 page 19 montre, dans le fichier `.log`, que les fontes `susettf-1-wghtxxx.lua` (avec `xxx` tiré aléatoirement[22]) sont créées à chaque appel de `i` et la sortie confirme que la chasse du caractère A (dépendant de sa graisse) est bien évaluée au moment de l'exécution.

Après l'exécution de ce programme, on trouve dans le cache les dix fichiers de la liste ci-dessous, correspondant à la fonte « variable » `suse.lua` et aux quatre fontes « statiques » créées pour chaque valeur de l'axe `wght` :

```
susettf-1.luca          susettf-1.luc
susettf-1-wght786.lua   susettf-1-wght786.luc
susettf-1-wght206.lua   susettf-1-wght206.luc
```

20. Les calculs sont nettement plus complexes, car ils tiennent compte du quadrant de l'espace de sortie; voir par exemple https://learn.microsoft.com/en-us/typography/opentype/spec/otvaroverview.

21. ... avec un nom comme `robotoflexvariablefontttf-1-wdth100.lua`...

22. Ici, ce tirage aléatoire correspond à un choix d'ordonnées dans le rang des ordonnées prévues par les *masters*, et non un choix d'ordonnées complètement aléatoires au moment de l'exécution comme c'était dans le cas de la fonte dynamique `Punk`, mentionnée en note 11 page 10.

```
susettf-1-wght541.lua   susettf-1-wght541.luc
susettf-1-wght256.lua   susettf-1-wght256.luc
```

Tout ceci peut paraître compliqué[23] et très chronophage. Mais il faut se dire qu'à part les programmes de démonstration comme cet article, on ne commande pas des occurrences de fonte variable très souvent et que ces calculs sont sauvegardés grâce aux mécanismes de cache.

Exemple 5 – Une fonte variable n'est pas une famille de fontes

code

```
\setmainfont{EBGaramond-VariableFont_wght.ttf}%
L'appel du \textbf{gras} \textit{n'est pas} normé.

\setmainfont{RobotoFlex-VariableFont.ttf}%
L'appel du \textbf{gras} \textit{n'est pas} normé.
```

résultat

L'appel du gras n'est pas normé.
L'appel du gras n'est pas normé.

Sélection des fontes variables

Une famille de fontes est en général formée de sous-fontes. On sait que LuaLaTeX (et plus précisément fontspec, dont on consultera avec profit le manuel[24] [25] permet, après la commande `\setmainfont{⟨nom de fonte⟩}`, d'avoir directement accès aux sous-fontes correspondantes comme par exemple le gras et l'italique (si elles existent bien sûr). Mais dans le cas d'une fonte variable, ces sous-fontes ne sont pas définies, comme le montre l'exemple 5.

Mais, nous l'avons vu, les fontes variables définissent en général des « instances » qui peuvent ainsi permettre de définir nous-mêmes nos habituelles commandes comme `\textbf`. Voir l'exemple 6 où l'on reprend les choix standards pour `\textbf…` et `\textit…`, mais où l'on préfère que `\textit{\textbf…}` donne un caractère moins noir et plus penché.

Exemple 6 – Appels standards pour une fonte variable

code

```
\setmainfont{RobotoFlex-VariableFont.ttf}
[
  UprightFont        = *,
  BoldFont           = *,
  BoldFeatures       = {Instance=Bold},
  ItalicFont.        = *,
  ItalicFeatures.    = {Instance=Italic},
```

23. Et ça l'est effectivement ; ou plutôt comme le dit le manuel de luaotfload [19, p. 37] : « *It looks complicated because it is complicated!* » (en français « Ça a l'air compliqué parce que c'est compliqué ! »).
24. et plus précisément la deuxième section de sa deuxième partie : [18, §2 *Font selection*]
25. Le manuel du package fontspec n'est malheureusement disponible qu'en anglais ; n'hésitez pas à le traduire en français !

code (suite)

```
    BoldItalicFont     = *,
    BoldItalicFeatures = {RawAxis={wght=600,slnt=-7}}, % choix perso
    Renderer           =  OpenType
  ]
  L'appel \textbf{de gras} \textit{n'est pas} \textit{\textbf{normé}}.
```

résultat

L'appel **de gras** n'est pas ***normé***.

En utilisant les primitives de plus bas niveau de LaTeX, on peut étendre les commande du type `\textbf` à d'autre valeurs de gras (avec par exemple `th`=*thin*, `eb`= *extrabold*, etc. L'exemple 7, inspirée du manuel du package luaotfload (voir [19, p. 21]), montre ainsi une dizaine d'appels de gras, les valeurs de gras étant celles prédéfinies par les instances de RobotoFlex.

Exemple 7 – *n* nuances de gris

code

```
\DeclareFontFamily{TU}{RobotoFlex-VariableFont}{}
\newcommand\DeclareSourceVariable[2]{%
  \DeclareFontShape{TU}{RobotoFlex-VariableFont}{#1}{n}{%
    <-> \UnicodeFontFile{RobotoFlex-VariableFont.ttf}
    {\UnicodeFontTeXLigatures axis={weight=#2};}%
  }{}%
  \DeclareFontShape{TU}{RobotoFlex-VariableFont}{#1}{it}{%
    <-> \UnicodeFontFile{RobotoFlex-VariableFont.ttf}
    {\UnicodeFontTeXLigatures axis={weight=#2};}%
  }{}%
}
\DeclareSourceVariable{th}{100}  % thin
\DeclareSourceVariable{el}{200}  % extra light
\DeclareSourceVariable{ l}{300}  % light
\DeclareSourceVariable{ r}{400}  % regular
\DeclareSourceVariable{ m}{500}  % medium
\DeclareSourceVariable{sb}{600}  % semibold
\DeclareSourceVariable{ b}{700}  % bold
\DeclareSourceVariable{eb}{800}  % extrabold
\DeclareSourceVariable{bb}{800}  % black
\DeclareSourceVariable{ub}{1000} % extrablack

\fontfamily{RobotoFlex-VariableFont}\selectfont
\fontseries{th}\selectfont a\textit{b}
\fontseries{el}\selectfont c\textit{d}
\fontseries{ l}\selectfont e\textit{f}
\fontseries{ r}\selectfont g\textit{h}
\fontseries{ m}\selectfont i\textit{j}
```

code (suite)

```
\fontseries{sb}\selectfont k\textit{l}
\fontseries{ b}\selectfont m\textit{n}
\fontseries{eb}\selectfont o\textit{p}
\fontseries{bb}\selectfont q\textit{r}
\fontseries{ub}\selectfont s\textit{t}
```

résultat

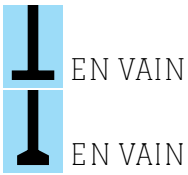
ab cd ef gh ij **kl mn op qr st**

Exemple 8 – Définition personnelle d'un style de fonte ; ici on utilise les axes SERF et SELE, qui jouent sur les formes des empattements.

code

```
% Requiert le package `trimclip`
\newcommand{\zoomserif}[1]{%
  \clipbox{10pt 0pt 10pt 50pt}{%
    \colorbox{cyan!35}{%
      \fontsize{100}{100}\selectfont #1%
    }%
  }%
}
\setmainfont{GRADUATE}[Renderer=OpenType]
%
                        \zoomserif{I} EN VAIN\par
\axe{SERF=30, SELE=-20}\zoomserif{I} EN VAIN
```

résultat



Rien n'interdit de se définir soi-même des fontes ayant des particularités stylistiques. L'exemple 8, qui utilise une fonte variable appelée GRADUATE, laquelle est munie d'axes jouant sur la forme des empattements (voir la section « Axes SERF et SELE : modification des patins » en page 49), emploie pour cela le concept de *font face* de NFSS [18, §4.3].

Définition de la fonte du jour

Pour rédiger cette *Lettre*, nous avons choisi la définition suivante :

Exemple 9 –

```
\setmainfont{RobotoFlex-VariableFont.ttf}[
  UprightFont         = *,
  UprightFeatures     = {
    SmallCapsFont     = {FiraSans-Regular.otf},
    SmallCapsFeatures = {Letters = SmallCaps}
```

```
    },
    BoldFont              = *,
    BoldFeatures          = {
      SmallCapsFont       = {FiraSans-SemiBold.otf},
      SmallCapsFeatures   = {Letters = SmallCaps},
      Instance            = Medium
    },
    ItalicFont            = *,
    ItalicFeatures        = {
          SmallCapsFont   = {FiraSans-Italic.otf},
      SmallCapsFeatures   = {Letters = SmallCaps},
      Instance            = Italic
    },
    BoldItalicFont        = *,
    BoldItalicFeatures    = {
          SmallCapsFont   = {FiraSans-BoldItalic.otf},
      SmallCapsFeatures   = {Letters = SmallCaps},
      Instance            = Medium Italic
    },
    Renderer              =  OpenType
  ]
  \setsansfont{RobotoFlex-VariableFont.ttf}
  \setmonofont{RobotoMono-VariableFont_wght.ttf}[
    UprightFont           = *,
    UprightFeatures       = {
      RawAxis             = {wght = 350}
    },
    Renderer              = OpenType
  ]
```

Les options `SmallCapsFont` et `SmallCapsFeatures` permettent d'utiliser les petites capitales de la fonte `Fira Sans`[26], car RobotoFlex n'en propose pas[27]. Nous avons tenté de créer ces petites capitales manquantes en utilisant les propriétés de variabilité de la fonte elle-même, ce qui a contribué au retard avec lequel paraît le présent article. Nous rendrons compte dans un prochain article de ce travail de création de petites capitales pour une fonte variable qui en serait dépourvue.

La dernière ligne permet de composer les commandes `\texttt` et le *verbatim* avec le caractère à chasse fixe de cette fonte RobotoFlex, légèrement plus gras que l'instance *Light*(avec `wdth=330` alors que pour *Light*, `wdth=300`), mais nettement plus fine que le *Regular* (pour qui `wdth=400`), ce qui permet de distinguer visuellement ces deux caractères sans pour autant attirer l'œil.

26. La fonte `Fira Sans` a été dessinée pour le système d'exploitation Firefox OS par Erik Spiekermann et Ralph du Carrois ; elle est basée sur `FF Meta`, du même Spiekermann.

27. Le liste des possibilités de RobotoFlex peut être obtenue en tapant `otfinfo -t RobotoFlex[GRAD,XOPQ,XTRA,YOPQ,YTAS,YTDE,YTFI,YTLC,YTUC,opsz,slnt,wdth,wght].ttf`, comme pour toute fonte OpenType [17, p. 59].

Animation des fontes variables

Les publicités de nombreuses fontes commerciales sont animées[28] au point que certaines personnes confondent les deux concepts (fonte variable et fonte animée). Ces animations sont également utilisées par certains *font inspectors* qui permettent, d'une part, de manipuler des curseurs pour voir les variations liées sur un texte donné, d'autre part, de faire varier ces axes dans des boucles temporelles[29].

Pour animer une fonte, il suffit de montrer diverses occurrences d'un même texte composé avec des variants différents (mais successifs) d'une fonte contrôlés par un outil de gestion du temps. Avec LualaTeX, on doit pouvoir utiliser Javascript et le package animate pour montrer une animation d'une fonte variable. Nous ne savons en dire plus, n'ayant pas regardé cette technique que nous considérons en amont du concept de fonte variable. Nous donnons toutefois, en page 56, un exemple de fonte variable munie d'un axe lié au temps, utilisable en animation.

Nous utilisons cette fonte pour numéroter les pages impaires de ce numéro : ceux qui auraient imprimé cette lettre[30] auront noté que les numéros des pages impaires correspondent à un cheval, qui s'animera en feuilletant les pages, à la manière d'un folioscope.

SECONDE PARTIE : LES AXES ET LEURS USAGES

Les variations faites sur une fonte le sont selon des « axes » tels que la graisse, la chasse, etc.

28. Pour n'en donner qu'un exemple (qui plus est, simpliste !) : https://www.monotype.com/sites/monotype/files/styles/width_910/public/2019-10/variablefonts_hero_v2.gif?itok=-mRDSimn.

29. C'est par exemple le cas de Dinamo Font Gauntlet (https://fontgauntlet.com) où, après avoir téléchargé une fonte variable, on peut l'animer en cliquant sur le curseur *play*.

30. Notons que le site de la *Lettre* en donne toujours deux versions. L'une présente un fond ivoire et est destinée à la lecture sur écran, qui permet de profiter des nombreux hyperliens que contient le texte, voire de leur prévisualisation si votre lecteur de PDF le permet. L'autre, sur fond blanc, est destinée à être imprimée… ce qui permet de profiter de l'effet folioscopique.

31. Les noms de ces axes nous fournissent l'occasion de nous souvenir de l'excellent magazine *U&lc* fondé par Herb Lubalin pour ITC. On en trouvera des reproductions ici : https://archive.org/details/volume-7-2_202511/Volume%201-1/

Les fontes variables OpenType distinguent :

— les cinq axes officiels (*registered axes*), avec leurs noms de quatre lettres en bas-de-casse :

`wght` – *weight*, graisse ;
`wdth` – *width*, chasse ;
`slnt` – *slant*, penché ;
`ital` – *italic*, italique ;
`opsz` – *optical size*, ajustement optique ;

— des axes définis par les dessinateurs (*customs axes*, axes privés), avec un nom de quatre lettres en capitales ; certains sont devenus des standards.

Nous allons décrire les cinq axes officiels et certains axes privés. Nous essaierons d'utiliser toujours la même présentation : un caractère vu avec la valeur par défaut, puis la valeur minimale et la valeur maximale pour cet axe (voir page 14). Puis un schéma montrant en général les *masters* (avec les principaux vecteurs qui les lient).

Tableau 6 – Deux échelles de graisse. À gauche celle des CSS basée sur le modèle de Panose (rapport épaisseur du fût / hauteur des caps) ; à droite celle des graphistes, basée sur l'épaisseur des fûts.

	Échelle CSS4	Épaisseur du fût
Thin	100	16
ExtraLight	200	30
Light	300	54
Regular	400	82
Medium	500	100
SemiBold	600	
Bold	700	146
ExtraBold	800	
Black	900	
Extrablack	1000	

Les cinq axes principaux et RobotoFlex

Rappelons que les cinq axes jouissent de commandes abrégées (`weight`, ...)

Axe `wght` : graisse (*weight*)

Les variations sur la graisse (axe `wght`) consistent en un élargissement du tracé des traits. La métrique des caractères est légèrement affectée (notamment la chasse). Comme beaucoup de fontes variables, RobotoFlex offre un large champ de variations de graisse, du très fin au très noir (voir le tableau 7 page suivante et le tableau 5 page 18).

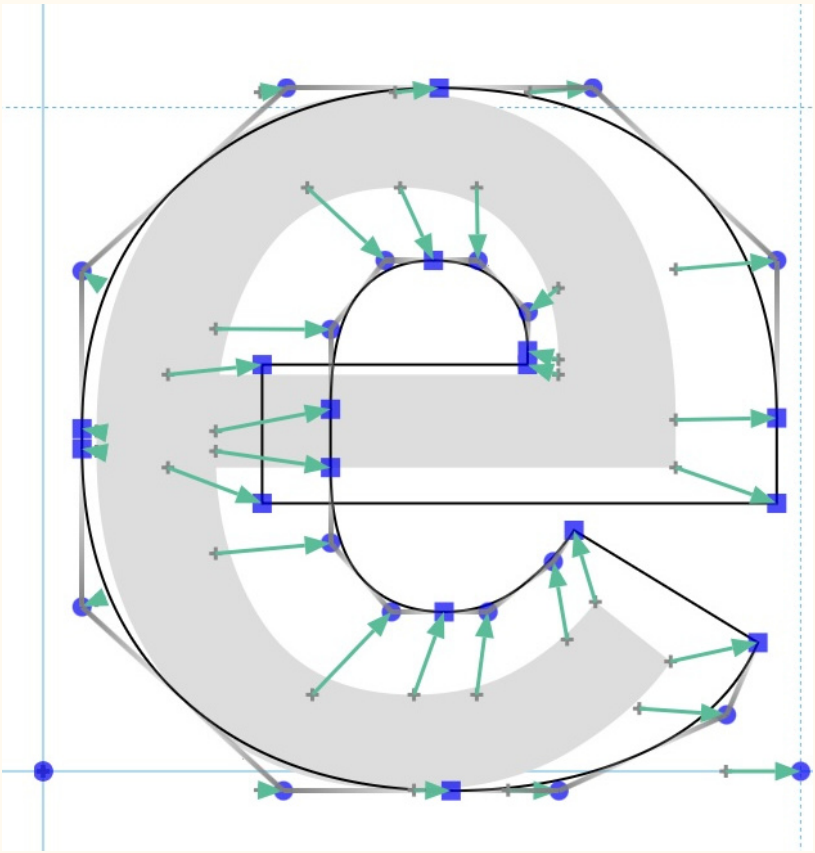
Valeurs du paramètre `wght`

Le concept de graisse ne date que du début du XIXe siècle [20] et sa mesure n'est pas rigoureusement définie, même si *a priori* ça semble intuitif... Deux modèles au moins font un peu référence, celui de Panose [5, p.418] et celui de Karow [21] ; leur principe commun est de mesurer le rapport entre l'épaisseur d'une lettre (I cap. au

TABLEAU 7 – Variations de l'axe `wght`

Code	Résultat
`e`	e
`\aff{Weight= 100} e`	e
`\aff{Weight=1000} e`	e

FIGURE 8 – Variation de graisse par élargissement du tracé



centre, ou H cap. juste au dessus de la transversale) et sa hauteur. Le W3C a adopté pour les CSS [15] le modèle de Panose et a quasiment normalisé cette échelle. Mais, répétons-le, cette échelle n'a rien de normatif pour les fontes variables et on trouve d'autres échelles, comme ces fontes qui donnent directement l'épaisseur du fût (en 1/1000e de cadratin), c'est la pratique usuelle des dessinateurs de fontes. Voir le tableau 6 page précédente.

Rappelons par ailleurs que ces échelles sont discontinues, mais les paramètres des axes peuvent prendre n'importe quelle valeur, par exemple `wght=350` qui donnera un gras intermédiaire entre un *Light* et un *Regular*, voire en deçà ou au-delà des valeurs minimales et maximales prévues… charge au système d'impression de faire l'extrapolation nécessaire.

RobotoFlex suit les recommandations du W3C et utilise les valeurs des CSS pour la graisse, c'est-à-dire `Thin=100`, `Regular=400`, `Bold=700`, etc. Bien que ce ne soit spécifié nulle part, même si c'est naturel, la graisse suit une fonction linéaire, comme le montre la figure 9 page ci-contre.

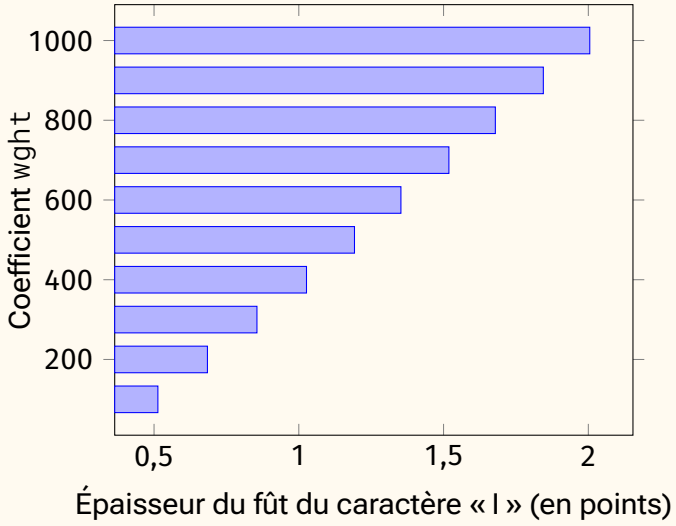


Figure 9 – La graisse de RobotoFlex suit une fonction linéaire.

Usages

Cet axe est utilisé surtout par question de goût : « j'aimerais un **bold** moins noir, alors je prends `Weight=470` (au lieu de `500`) qui me donne ce nouveau **bold**. »

Le gras a en général un rôle d'accroche (dans un page, l'œil est attiré par le gras) mais lorsqu'au contraire il est maigre, il a plutôt un rôle de distinction (comme pour l'italique, l'œil ne voit la différence qu'au moment de la lecture du « maigre »). Voir l'exemple 10 ci-après.

Exemple 10 – Variations de graisse ; rôles de distinction (à gauche) ou d'accroche (à droite).

code

```
\newcommand\montexte[1]{%
  Les \gras{#1}{alignements de Carnac}
  sont un ensemble de quatre sites mégalithiques
  (\gras{#1}{Kermario, le Ménec, Kerlescan, Petit Ménec
   })
  exceptionnels, situés sur les communes de Carnac
  et de La Trinité-sur-Mer dans le département
  du \gras{#1}{Morbihan} en Bretagne.%
}
\newcommand\gras[2]{{\addfontfeatures{Weight=#1}#2}}
%
\parbox[t]{.45\linewidth}{\montexte{250}}
\hfill\vrule\hfill
\parbox[t]{.45\linewidth}{\montexte{750}}
```

résultat (suite)

Les alignements de Carnac sont un ensemble de quatre sites mégalithiques (Kermario, le Ménec, Kerlescan, Petit Ménec) exceptionnels, situés sur les communes de Carnac et de La Trinité-sur-Mer dans le département du Morbihan en Bretagne.	Les **alignements de Carnac** sont un ensemble de quatre sites mégalithiques (**Kermario, le Ménec, Kerlescan, Petit Ménec**) exceptionnels, situés sur les communes de Carnac et de La Trinité-sur-Mer dans le département du **Morbihan** en Bretagne.

Axe `wdth` : chasse (*width*)

Les variations de chasse (axe `wdth`) consistent en un élargissement (extension) ou un rétrécissement (compression) horizontal du tracé des contours (mais plus subtil qu'un simple `\scalebox{x}[1]`). La métrique des caractères est nettement affectée (mais seulement la chasse donc) alors que la hauteur des œils n'est pas modifiée et que le gris reste quasiment le même. Voir le tableau 8 et la figure 10 page suivante.

TABLEAU 8 – Variations de l'axe `wdth`

Code	Résultat
`e`	e
`\aff{Width= 25} e`	e
`\aff{Width=150} e`	e

Valeurs du paramètre `wdth`

Les valeurs du paramètre `wdth` (*width*, chasse) sont également au gré du dessinateur de la fonte. Mais ici encore, l'usage est désormais de suivre le même concept que celui adopté par le W3C pour les CSS :

> Les valeurs sont spécifiées soit sous forme de pourcentages, soit sous forme de mots-clés correspondant à un pourcentage, tel que défini dans le tableau 9[32].

TABLEAU 9 – Échelle css4 de compression / extension

Nom	%
ultra-condensed	50
extra-condensed	62,5
condensed	75
semi-condensed	87,5
normal	100
semi-expanded	112,5
expanded	125
extra-expanded	150
ultra-expanded	200

32. en anglais : *Values are specified either as percentages or as keywords which map to a percentage as defined in the following table* [15]

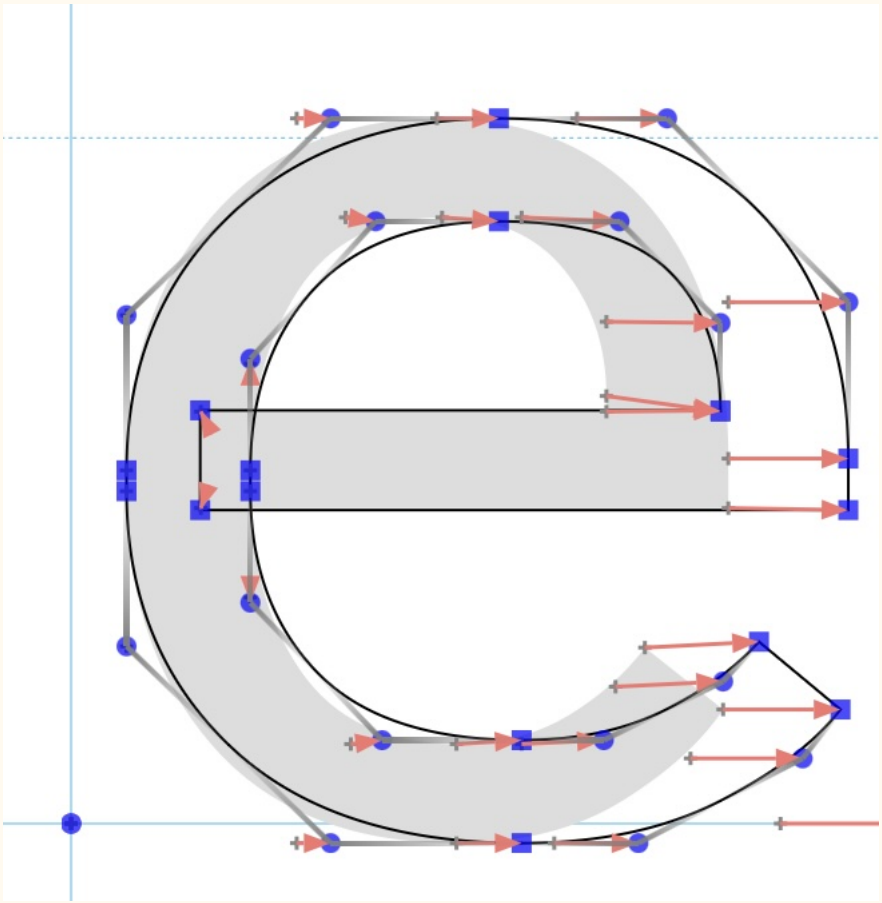


FIGURE 10 – Variation de chasse par modification horizontale du tracé

RobotoFlex emploie le principe des css (les valeurs de `wdth` peuvent aller de 25 (%) à 150 (%) du défaut (100%), mais multiplié par un pourcentage qui n'est pas le même pour la compression que pour l'extension, ce qui donne les deux droites de la figure 11 page suivante.

À noter que RobotoFlex ne définit pas d'instance pour l'axe `wdth`.

Usages

Les graphistes aiment à choisir la largeur de leurs caractères pour des raisons esthétiques. Tout d'abord pour les mots (par exemple pour faire des logos) ; le texte de l'exemple 11 est composé avec `Width=25` puis `Width=150`) pour illustrer les extrêmes possibles avec RobotoFlex.

Exemple 11 –

code

```
\Huge
{\addfontfeatures{Width= 25} amococadiz} ---
{\addfontfeatures{Width=150} amococadiz}
```

résultat

amococadiz — amococadiz

Pour de petites variations de chasse, le gris n'est pas modifié de façon sensible.

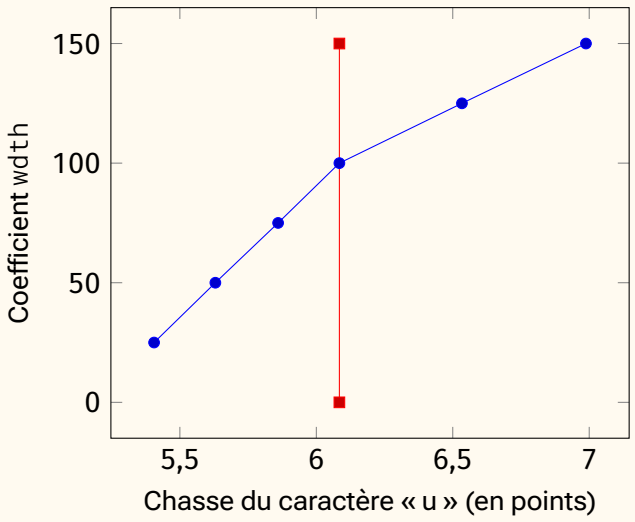


FIGURE 11 – Variations de la chasse du « u » de la fonte RobotoFlex en fonction du paramètre `wdth` (en rouge, la chasse par défaut).

Nous pouvons alors composer le même texte en jouant sur sa chasse de façon à avoir des compositions plus ou moins longues. L'exemple 12 montre que l'on peut, sans modification apparente à l'œil, gagner une ligne dans une petite composition.

Exemple 12 – Gain d'une ligne en changeant légèrement `wdth`. Le texte est de Jack Kerouac [22].

code

```
\parbox[t]{.4\linewidth}{\Kerouac}
\hfill\vrule\hfill
\parbox[t]{.4\linewidth}{\aff{Width=65}\Kerouac}
```

résultat

J'étais venu en France et en Bretagne, uniquement pour opérer des recherches sur ce vieux nom qui est le mien, qui a près de trois mille ans, et qui n'a jamais changé durant tout ce temps. Qui voudrait changer un nom qui signifie simplement maison (ker), dans le champ (ouac) —

J'étais venu en France et en Bretagne, uniquement pour opérer des recherches sur ce vieux nom qui est le mien, qui a près de trois mille ans, et qui n'a jamais changé durant tout ce temps. Qui voudrait changer un nom qui signifie simplement maison (ker), dans le champ (ouac) —

Cette diminution a été faite par tâtonnements. Mais on peut procéder aussi par calculs. C'est ce qui arrive quand un graphiste veut qu'un texte (en général un titre) tienne exactement dans la justification. On peut obtenir cela en agrandissant les espaces intermots, ou en agrandissant l'interlettrage (ceci est possible pour les fontes OpenType grâce à la propriété `\addfontfeatures{LetterSpace=}`).

Des Anglais utilisent une méthode, dite *de Leicester*, qui consiste à augmenter très légèrement le corps : ça ne se voit pas et évite les blancs superflus. Nous nous inspirons de cette méthode (qui revient à une règle de trois…) pour trouver la chasse qu'il faut donner au titre pour qu'il ait la bonne justification, connaissant celle qu'il a avec le `wdth` courant.

Le code de l'exemple 13 montre un exemple de calculs pour ces trois méthodes[33]. La première ligne présente le texte original, composé sans intervention particulière. En deuxième ligne de sortie, on voit que l'usage de `\makebox[s]`… peut donner des blancs trop grands ou échouer. Sur la troisième ligne, l'interlettrage peut donner une impression d'espaces-mots trop petites ou au contraire conduire à un chevauchement des lettres. En dernière ligne, notre solution joue donc sur la chasse des lettres, et nous semble meilleure.

Exemple 13 – Trois méthodes pour donner à un texte une justification précise

code

```
% Extensions requises : fontspec, xfp,
% calculator, xstring, ifthen, tcolorbox

\makeatletter

%%%
% Définition des macros Leicester et Interlettre
\newcommand*{\getlengthencm}[1]{%
  \strip@pt\dimexpr0.0351510\dimexpr#1\relax\relax
}
\newcommand{\@mawidth}{}
\newlength{\@extr}\newlength{\@courant}
\newcounter{monw}

\newcommand{\Leicester}[2]{% txt justif
  \settowidth{\@courant}{#1}%
  \ifthenelse{\@courant>#2}{
    \setcounter{monw}{25}}{
    \setcounter{monw}{151}
  }%
  \addfontfeatures{Width=\themonw}%
  \settowidth{\@extr}{#1}%
  \renewcommand{\@mawidth}{%
    \fpeval{%
      \themonw*\getlengthencm{#2}/\getlengthencm{\@extr}
    }%
  }%
  {\addfontfeatures{Width=\@mawidth}#1}%
}

\newlength{\TrackL}\newlength{\TrackLL}
\newcounter{cntr}
\newcommand{\JAtrack}[2]{% texte=#1 sur longueur #2
```

33. Contrairement à ce qu'on croit, la chasse d'un texte ne varie pas linéairement en modifiant le paramètre `wdth=`, du moins pour RobotoFlex. En dressant la courbe `chasse(wdth)` en fonction de `wdth` pour un texte donné, on voit (en figure 11 page précédente) qu'elle est en fait formée de deux segments, l'un légèrement oblique jusqu'à la valeur par défaut de la chasse, l'autre nettement plus penché, du défaut au maximum. Suivant que le texte est plus court ou plus long que la justification voulue, on utilise une règle de trois avec la valeur du minimum ou avec celle du maximum (d'où le `\ifthenelse` de la macro `Leicester`). Outre cette bilinéarité de RobotoFlex, notre macro manque de généralité puisqu'on utilise les valeurs minimales (25) et maximales (151) de l'axe `wdth`, faute de disposer d'une macro qui nous donnerait automatiquement ces valeurs.

code

```
    \settowidth{\TrackL}{#1}%
    \setlength{\TrackLL}{\dimexpr \the#2-\the\TrackL}%
    \expandarg\StrLen{#1}[\longr]%
    \setcounter{cntr}{\numexpr \longr-1}%
    \setlength{\TrackL}{\dimexpr (\the\TrackLL*12)/\thecntr}% 1em=12pt
    \LENGTHDIVIDE{\the\TrackL}{1pt}{\Ratio}%
  }
  \newcommand{\Interlettre}[2]{%
    \JAtrack{#1}{#2}%
    {\addfontfeatures{LetterSpace=\Ratio}#1}%
  }
  %%%

  \newlength{\majustif}
  \setlength{\majustif}{4.6cm}
  \newcommand{\ligne}[1]{%
    \makebox[\the\majustif][s]{#1}%
  }
  \newcommand{\cfglignes}{%
    \parindent=0pt
    \baselineskip=1.5\baselineskip
  }
  \newcommand{\sanscouleur}{%
    \let\peutetre@couleur\relax
  }
  \newcommand{\aveccouleur}{%
    \def\peutetre@couleur{\colorbox{cyan!35}}%
  }
  \newcommand{\quatrelignes}[4]{%
    \parbox{5.5cm}{%
    \scalebox{1}{%
      \addfontfeatures{Width=100}%
      \fontsize{12}{12}\selectfont
      \setlength{\fboxsep}{0em}%
      \peutetre@couleur{%
        \begin{minipage}{\the\majustif}\cfglignes
        \vspace{.7ex}\leavevmode
        #1\\
        #2\\
        #3\\
        #4\\
        \vspace{\dimexpr-\baselineskip+1ex}
        \end{minipage}%
      }%
    }}%
  }
  \newcommand{\exemples}[1]{%
    \quatrelignes
      {\hbox{#1}}
      {\ligne{#1}}
```

code (suite)

```
    {\ligne{\Interlettre{#1}{\majustif}}}
    {\ligne{\Leicester{#1}{\majustif}}}%
}

\makeatother

\scalebox{0.85}{%
  \noindent\sanscouleur
  \quatrelignes
  {Sans justification}
  {Espaces inter-mots}
  {Interlettrage}
  {Chasse}%
  \hspace{-4em}\aveccouleur
  \exemples{Un titre vraiment court}%
  \hspace{-1em}%
  \exemples{Un titre beaucoup trop long}%
}
```

résultat

Sans justification	Un titre vraiment court	Un titre beaucoup trop long
Espaces inter-mots	Un titre vraiment court	Un titre beaucoup trop long
Interlettrage	Un titre vraiment court	Un titre beaucoup trop long
Chasse	Un titre vraiment court	Un titre beaucoup trop long

Axe `slnt` : oblique (*slant*)

Les variations de l'axe `slnt` consistent à *pencher* un caractère d'un angle donné par la valeur de l'axe (supposée en degrés, dans le sens trigonométrique). Dans l'exemple 14 page suivante on voit (3e ligne) que les horizontales restent horizontales et que les verticales subissent une rotation de cet angle alors qu'avec une rotation classique (2e ligne) les horizontales ne le restent pas.

Ce même exemple 14 page suivante montre aussi que TEX offre la même possibilité avec son concept d'obliques (avec LATEX : `\textsl`, `\slshape`).

Nous verrons à la section suivante sur l'axe `ital`, page 37, que caractères penchés et caractères italiques se distinguent par la substitution de caractères cursifs pour certains des seconds.

La métrique des caractères est affectée[34], tandis que la hauteur des œils n'est pas modifiée et que le gris reste quasiment inchangé.

L'angle de rotation est en général compris entre 0 °et – 10 °, parfois – 15 °(voir le tableau 10 page suivante). Certaines fontes, disons expérimentales, comme GT PLANAR de la fonderie suisse Grilli, offrent une variation de l'angle de – 90 ° à + 90 ° !

34. Les caractères penchés chassent plus que les droits. À noter qu'en général les caractères italiques chassent moins que les droits, car leur dessin est plus compact !

TABLEAU 10 – Variations de l'axe `slnt`

Code	Résultat
`Te`	Te
`\aff{Slant=-10} Te`	*Te*
`\aff{Slant=0} Te`	Te

Exemple 14 – Comparaison rotation / oblique

code

```
% Nécessite le package `booktabs`
\newcommand\grilleR[1]{
  \begin{picture}(1em,1em)
    \linethickness{.5pt}{
      \color{blue}{%
        \multiput(0,0)(0,.2em){6}
        {\line(1,0){1em}}%
      }%
      \multiput(0,0)(.2em,0){6}
      {\line(0,1){1em}}%
    }%
    \put(0,0.2em){#1}
  \end{picture}
}
%
\fontsize{29}{29}\selectfont
\lstset{basicstyle=\normalsize\ttfamily}
\centering
\begin{tabular}{ll}
 \lstinline|\grilleR{R}|
   & \grilleR{R}                    \\\midrule
 \lstinline|\grilleR{\rotatebox{-10}{R}}|
   & \grilleR{\rotatebox{-10}{R}} \\\midrule
 \lstinline|\aff{Slant=-10}\grilleR{R}|
   & \aff{Slant=-10}\grilleR{R}   \\\midrule
 \lstinline|\grilleR{\textsl{R}}|
   & \grilleR{\textsl{R}}
\end{tabular}
```

résultat (suite)

```
\grilleR{R}
```


```
\grilleR{\rotatebox{-10}{R}}
```


```
\aff{Slant=-10}\grilleR{R}
```


```
\grilleR{\textsl{R}}
```


Axe `ital`(ique)

Pour beaucoup de typographes, l'italique est une fonte cursive (penchée comme l'écriture manuelle) dont le dessin des lettres peut être très différent de celui du romain de la même famille. Exemple :

Times romain : a d e f g p q v w y
Times *italique* : *a d e f g p q v w y*

C'est pourquoi de nombreuses familles de fontes variables distinguent les deux formes au point de ne pas proposer l'italique comme une variante. C'est le cas de `EBGaramond`, par exemple — cette fonte présente deux fichiers distincts :

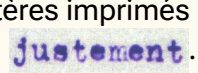
— `EBGaramond-VariableFont_wght.ttf`;
— `EBGaramond-Italic-VariableFont_wght.ttf`.

Il en va de même pour les fontes `JunicodeVF`.

Une simulation classique de l'italique consiste à pencher les caractères, tout simplement ; avec les fontes variables, on obtient ce résultat avec l'axe `slnt` que nous venons de voir, mais les caractères restent penchés sans avoir la cursivité de l'italique ! Nous l'avons vu sur RobotoFlex.

OpenType a donc prévu une meilleure simulation de l'italique en penchant les caractères (avec un angle choisi par l'utilisateur, donné dans l'appel `Rawaxis={slnt=alpha}`) mais aussi en remplaçant certains caractères par des caractères plus cursifs (voir l'exemple 15 page suivante). Certaines fontes, comme `Brito`[35], choisissent un angle au-dessus duquel ont lieu simultanément les deux opérations (inclinaison et substitution), comme le montre la figure 12 page suivante.

Axe `opsz` : correction optique (*optical size*)

Ceux d'entre nous qui ont connu les machines à écrire se souviennent qu'avec du matériel usé (ruban encreur ou même barres de frappe), les caractères imprimés avaient tendance à se boucher, l'encre s'étalant de trop, comme ici :  . Les graveurs de caractères en plomb connaissaient bien ce phénomène (qui était surtout visible dans les petits corps) et le prévenaient alors en étroitisant les tracés et en agrandissant les œils des lettres d'autant plus que le corps était petit : le

35. Voir note 18 page 18, figure 7 page 19 et tableau 4 page 17.

Exemple 15 – L'axe `ital` simule les italiques (cf. page 37)

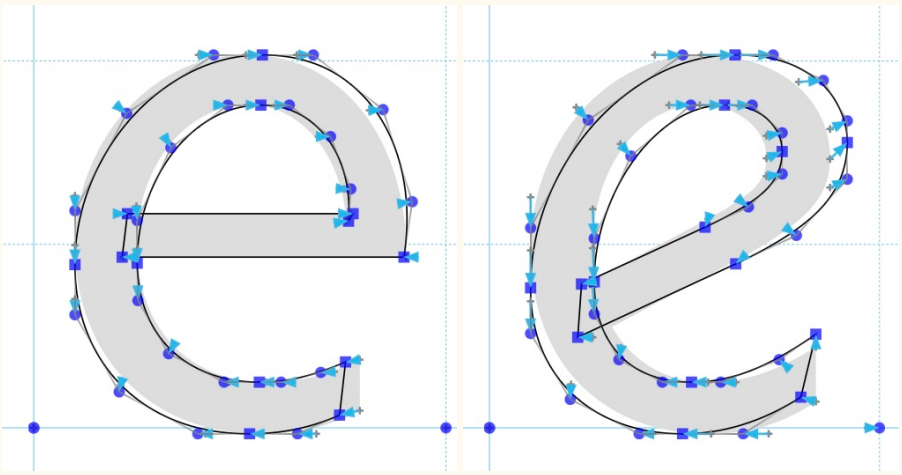
code

```
% Requiert le package `pgffor` et
% \newfontfamily\BVF{BritovariableVF.ttf}
% (définie plus haut)
\BVF
\foreach\i in{0,.20,.49,.50,.51,.80,1}{%
  \makebox[3em][l]{\small i=\i}%
  \addfontfeatures{
    Renderer = OpenType,
    RawAxis={ital={\i}}
  }lep\\
}
```

résultat

i=0 lep
i=.20 lep
i=.49 *lep*
i=.50 *lep*
i=.51 *lep*
i=.80 *lep*
i=1 *lep*

Figure 12 - L'axe `ital` illustré avec la fonte `Brito`



Brito romain : a d e f g p q v w y
Brito *italique* : *a d e f g p q v w y*

dessin d'un corps 6 n'était donc pas homothétique à celui d'un corps 16. Les bonnes fontes numériques font de même depuis longtemps. Knuth a été l'un des premiers à n'utiliser des fonctions de *scaling* que sur de petites gammes de corps (p. ex. entre 8.5 et 9.5) ce qui nécessitait de nombreuses sous-fontes ; c'était le cas donc de toutes les `Computer Modern`, que l'on retrouve aujourd'hui avec les `lmr` ce qu'on peut visualiser comme suit :

```
$ cd /usr/local/texlive/2024/texmf-dist
cd fonts/opentype/public/lm
otfinfo -z lmroman5-regular.otf
otfinfo -z lmroman7-regular.otf
otfinfo -z lmroman12-regular.otf

design size 5pt, size range (3pt, 5.5pt], subfamily ID
1, subfamily name Regular
```

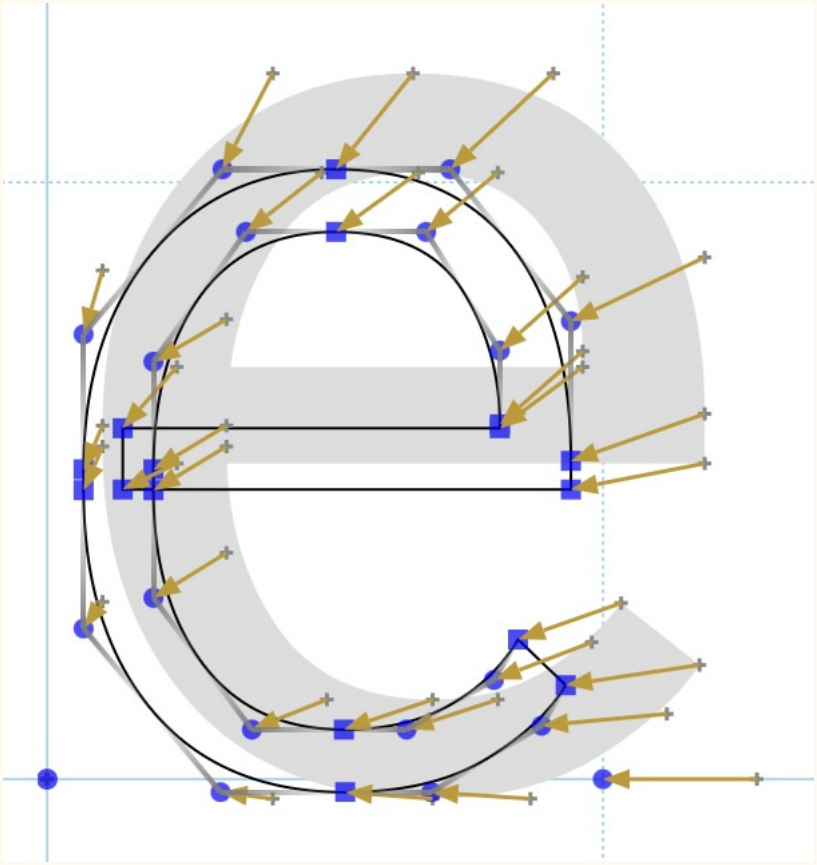
```
design size 7pt, size range (6.5pt, 7.5pt], subfamily ID
1, subfamily name Regular
design size 12pt, size range (11pt, 14pt], subfamily ID
1, subfamily name Regular
```

L'axe `opsz` des fontes variables permet à l'utilisateur de forcer le système de traitement de texte à appliquer cette correction optique quand il le désire (voir le tableau 11 et la figure 13).

Tableau 11 – Axe `opsz` (correction optique) : le tracé du caractère est étroitisé à mesure que son corps diminue

Code	Résultat
`e`	e
`\axe{opsz=  8} e`	e
`\axe{opsz=144} e`	e
`deb`	deb
`\axe{opsz=100} deb`	deb
`\axe{opsz=144} deb`	deb

Figure 13 – Axe `opsz` – correction optique par étroitisation du tracé et agrandissement de l'œil



Les autres axes de RobotoFlex

Comme on l'a vu en section « Quels axes ? », page 13, outre ces cinq axes standards, RobotoFlex propose des axes privés que nous listons dans les sections qui suivent.

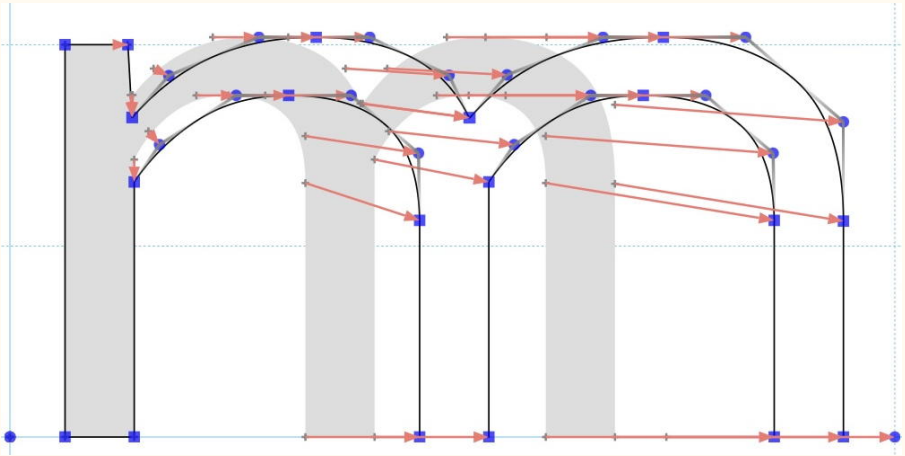
Axe XTRA : *XTRAnsparent*

Modification des contre-poinçons, par extension horizontale, sans changer l'épaisseur des fûts — d'où le nom [36] : cet axe n'intervient que sur les éléments *transparents* du caractère. Évidemment, la chasse est augmentée pour les signes concernés. Voir le tableau 12 et la figure 14.

Tableau 12 – Contrôle de l'extension horizontale par l'axe XTRA (nécessite d'être sous l'effet de `\aff{Renderer=OpenType}`).

Code	Résultat
`\axe{XTRA=323} e`	e
`\axe{XTRA=603} e`	e

Figure 14 – Axe XTRA : extension horizontale



La principale différence avec l'axe `wdth` est qu'ici seules les lettres avec contrepoinçon (creux plus ou moins ouvert) sont concernées : les signes qui n'en possèdent pas, tels `i`, `l`, + et d'autres, ne sont pas modifiés. De surcroît, l'extension en *x* est plus importante pour XTRA. Nous comparons ces deux axes dans le tableau 13.

Tableau 13 – Comparaison de l'utilisation des axes `wght` (`Weight`) et XTRA (nécessite d'être sous l'effet de `\aff{Renderer=OpenType}`)

Code	Résultat
`mame+lili`	mame+lili
`\axe{Width=151} mame+lili`	mame+lili
`\axe{ XTRA=603} mame+lili`	mame+lili

36. Ce nom est évidemment discutable. Pourquoi ne pas l'avoir appelé XCTP, pour eXtension du ConTrePoinçon (ou du CounTerPunch) ?

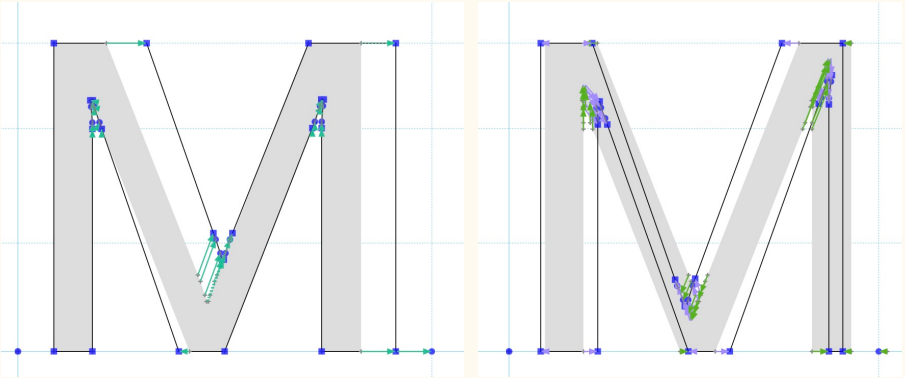
Axes XOPQ et YOPQ, épaisseur des fûts

Ces deux axes permettent de modifier le tracé des fûts, soit en *x*, c'est-à-dire horizontalement (donc en épaisseur), soit en *y*, c'est-à-dire verticalement (donc en étroitesse). Les horizontales ou diagonales sont plus ou moins affectées… voir le tableau 14 et la figure 15.

Tableau 14 – Axes XOPQ et YOPQ – modification de l'épaisseur des fûts (nécessite d'être sous l'effet de `\aff{Renderer=OpenType}`)

Code	Résultat
`MaNimuX`	MaNimuX
`\axe{XOPQ= 27} MaNimuX`	MaNimuX
`\axe{XOPQ=175} MaNimuX`	**MaNimuX**
`MaNimuX`	MaNimuX
`\axe{YOPQ= 25} MaNimuX`	MaNimuX
`\axe{YOPQ=135} MaNimuX`	**MaNimuX**
`minimorum`	minimorum
`\axe{YOPQ=135} minimorum`	**minimorum**
`\axe{XOPQ=175} minimorum`	**minimorum**

Figure 15 – Modification des fûts du caractère M selon les axes *x* et *y*



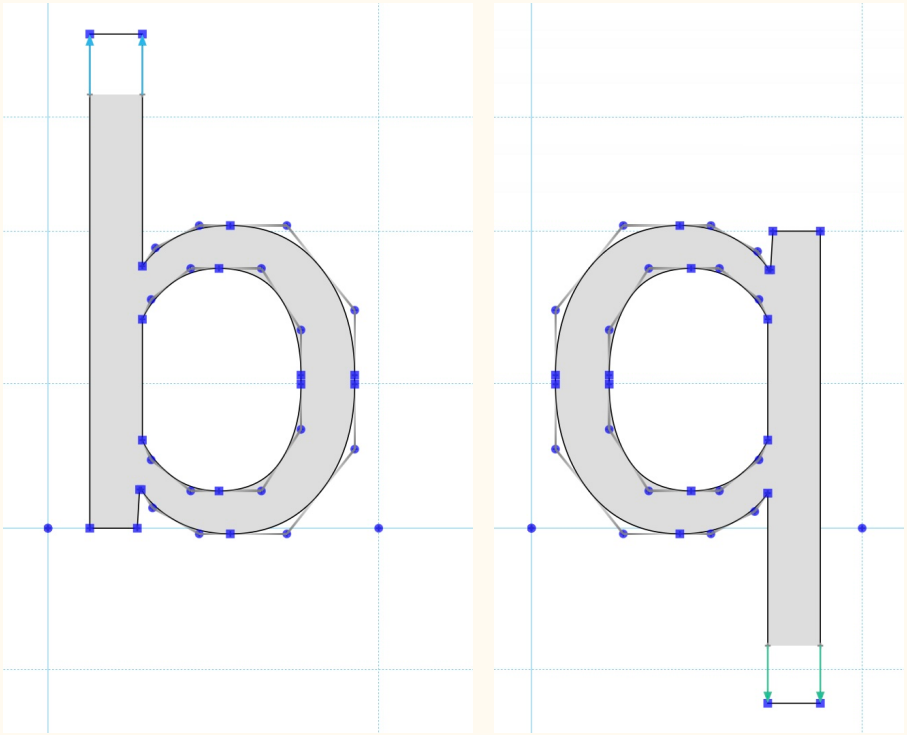
Axes YTAS et YTDE (AScendantes et DEscendantes)

Modification de la hauteur des ascendantes ou de la profondeur des descendantes. Comme le montrent la figure 16 et le tableau 15, seuls les hampes ou jambages sont modifiés — en *y*, d'où le nom — et la hauteur d'œil reste inchangée.

TABLEAU 15 – Utilisation des axes YTAS et YTDE (nécessite d'être sous l'effet de `\aff{Renderer=OpenType}`)

Code	Résultat
`b\axe{YTAS=649}b\axe{YTAS=854}b`	bbb
`p\axe{YTDE=-305}p\axe{YTDE=-98}p`	ppp

FIGURE 16 – Axes YTAS et YTDE – contrôle des dimensions des seuls fûts



Le beau baladin file dans le bidon lent !
Le beau baladin file dans le bidon lent !

On trouvera un exemple d'emploi de l'axe YTAS (combiné avec d'autres axes) en section « Harmonisation des hauteurs de caractères[37] », page 45.

37. Cet exemple nous a été fourni par Daniel Flipo que nous tenons à remercier ici !

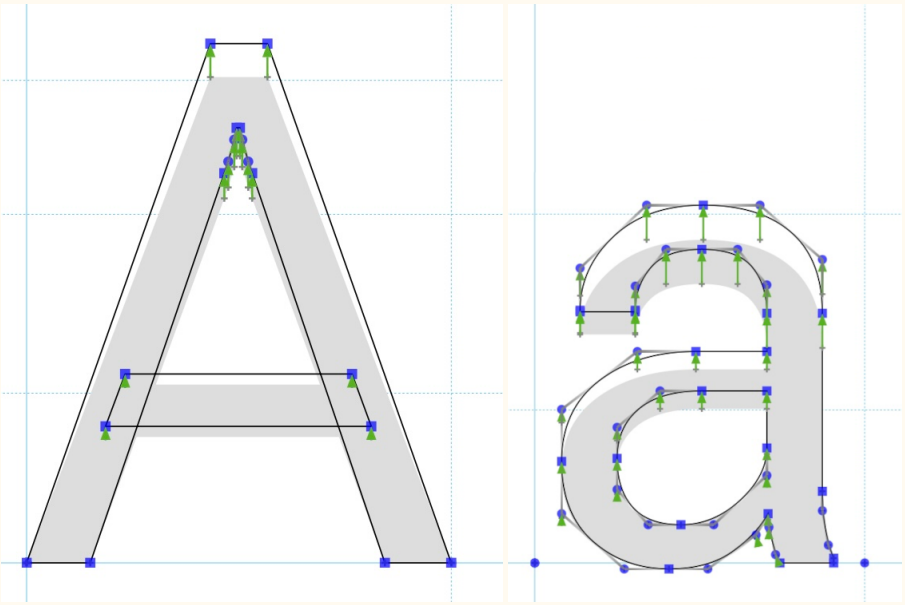
Axes YTUC et YTLC, hauteur des capitales (*Upper Case*) ou des bas-de-casse (*Lower Case*)[38]

Légères variations en y des capitales ou des bas-de-casse, sans modification de la chasse. En fait, tout se passe à peu près comme si on faisait `\scalebox{1}[1.05]{X}` sur la lettre X (la valeur 1.05 est approximative et dépend de la valeur donnée à YTUC ou à YTLC). Voir le tableau 16 et la figure 17.

TABLEAU 16 – Utilisation des axes YTUC et YTLC (nécessite d'être sous l'effet de `\aff{Renderer=OpenType}`)

Code	Résultat
`A\axe{YTUC=528}A\axe{YTUC=760}A`	AAA
`a\axe{YTLC=416}a\axe{YTLC=570}a`	aaa

FIGURE 17 – Axes YTUC et YTLC – contrôle de la hauteur des lettres



On trouvera dans parmi les exemples d'usage, à partir de la page suivante, et notamment en section « Harmonisation des hauteurs de caractères[39] », page 45, des exemples concrets d'utilisation de ces variations.

38. Les noms de ces axes nous fournissent l'occasion de nous souvenir de l'excellent magazine *U&lc* fondé par Herb Lubalin pour ITC. On en trouvera des reproductions ici : https://archive.org/details/volume-7-2_202511/Volume%201-1/

39. Cet exemple nous a été fourni par Daniel Flipo que nous tenons à remercier ici !

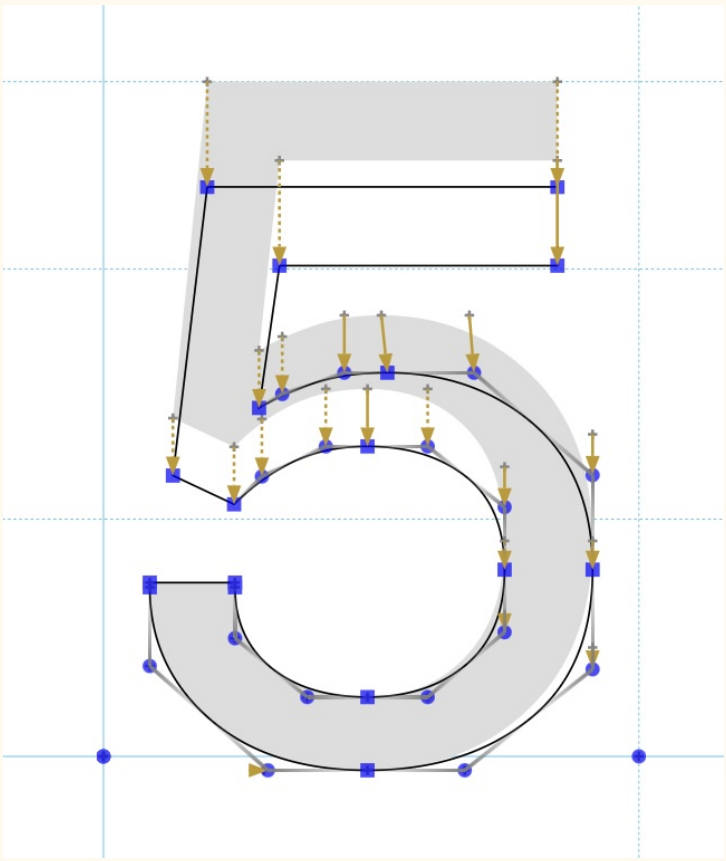
Axe YTFI (*FIgure* = chiffre)

Comme le montrent le tableau 17 et la figure 18, cet axe fonctionne selon le même principe que l'axe YTUC, mais appliqué aux chiffres.

TABLEAU 17 – Utilisation de l'axe YTFI (nécessite d'être sous l'effet de \aff{Renderer=OpenType})

Code	Résultat
5\axe{YTFI=560}5\axe{YTFI=788}5	555

FIGURE 18 – Axe YTFI – contrôle de la hauteur des chiffres



Exemples d'usage

À titre de démonstration, voici deux exemples d'emploi des axes de variation liés à la métrique des caractères que nous venons de voir (YTUC, YTLC, YTFI).

Gros œil au goût hollandais

Nous avons vu (en figure 2 page 3) qu'au XVIII^e siècle Fournier appelait ainsi des caractères dont l'œil, la hauteur d'x, était assez importante, mais dont les hampes étaient également grandes. L'exemple 16 page suivante[40] montre comment, en donnant aux paramètres correspondants YTUC et YTLC leurs valeurs maximales, on peut avoir (à droite) un caractère paraissant plus gros (donc plus lisible) alors qu'il a la même hauteur de corps et la même chasse que celui de gauche (composé avec les valeurs par défaut de YTUC et YTLC). Ce « goût hollandais » est peu apprécié des Français en général qui du coup n'aiment pas les caractères comme Le Monde ou Lucida [23].

40. NDLR Nous regrettons de ne pas connaître la référence de ce texte cité par Fournier en 1767.

Exemple 16 – Le même caractère, modifié en « gros œil »

code

```
\parbox[t]{.45\linewidth}{\grosoeil}
\hfill\vrule\hfill
\aff{Renderer=OpenType,RawAxis={YTLC=570,YTUC=760}}
\parbox[t]{.45\linewidth}{\grosoeil}
```

résultat

Souvent, les hommes corrompent tout, parce qu'ils sont eux-mêmes corrompus. Il n'est point de crime à qui ils n'aient donné le nom de vertu, ni de vertu qu'ils n'aient accusée de foiblesse ou de folie ; de sorte qu'ils sont capables de louer les plus grands vices, & de condamner les plus grandes vertus. [Texte utilisé par Fournier, 1767]

Souvent, les hommes corrompent tout, parce qu'ils sont eux-mêmes corrompus. Il n'est point de crime à qui ils n'aient donné le nom de vertu, ni de vertu qu'ils n'aient accusée de foiblesse ou de folie ; de sorte qu'ils sont capables de louer les plus grands vices, & de condamner les plus grandes vertus. [Texte utilisé par Fournier, 1767]

Harmonisation des hauteurs de caractères[41]

Il arrive souvent que l'on ait à mélanger dans une même phrase des caractères de fontes différentes ; c'est en particulier le cas lorsque l'on utilise la commande `\texttt{…}` avec une fonte sans empattement différente de la fonte courante, notamment par ses hauteurs de capitales ou de bas-de-casse. C'est ce qui se passe dans l'exemple suivant avec `EBGaramond` et RobotoFlex, où l'on voit que les caractères `HT` et `ux` sont légèrement plus hauts en RobotoFlex, ce qui peut choquer l'œil.

Exemple 17 – Harmonisation de hauteurs, 1 : le problème

code

```
% Requiert le package `xcolor`
\newlength{\UCheight} \newlength{\oeil}
\newlength{\DESdepth} \newlength{\lthickness}
\newlength{\samplewidth}
\newcommand*{\redline}[1]{\hbox to 0pt{%
  % Si vous vous demandez pourquoi je n'ai pas utilisé
  % \makebox au lieu de la primitive \hbox, essayez voir
  \color{red}\rule[#1]{\samplewidth}{\lthickness}\hss
}}
\newcommand*{\exemple}{{\fontsize{30}{48}\selectfont
   \setlengths
  \leavevmode
  \redline{-\DESdepth}\redline{-\lthickness}%
  \redline\oeil \redline\UCheight
  \sample}}
\newcommand*{\leg}[1]{{\footnotesize\fontspec{
   EBGaramond-Italic.otf}
```

41. Cet exemple nous a été fourni par Daniel Flipo que nous tenons à remercier ici !

code (suite)

```
    \hspace*{0.5cm}#1}}
\newcommand*{\sample}{HT\textsf{HT}, bqux\textsf{bqux}}

\newcommand*{\setlengths}{
  \settodepth{\DESdepth}{p} \setlength{\lthickness}{0.
   2pt}
  \settoheight{\oeil}{x} \settoheight{\UCheight}{H}
  \settowidth{\samplewidth}{\sample}
}

\setmainfont{EBGaramond-Regular.otf}
\setsansfont{RobotoFlex-VariableFont}
\exemple\leg{Pas d'ajustements}
```

résultat

HTHT, bquxbqux *Pas d'ajustements*

On peut penser corriger ceci en utilisant les propriétés de mise à l'échelle *(scaling features)* de fontspec [18, § 6.2], mais, comme le montre l'exemple suivant, les deux options ne fonctionnent pas ensemble (dans un cas les bas-de-casse sont bien alignées, mais pas les capitales ; dans l'autre c'est le contraire).

Exemple 18 – Harmonisation de hauteurs, 2 : avec `Scale=Match...` de fontspec

code

```
% On reprend les définitions de l'exemple précédent.
\setmainfont{EBGaramond-Regular.otf}

\setsansfont{RobotoFlex-VariableFont}[Scale=
   MatchLowercase]
\exemple\leg{MatchLowercase mais pas les capitales}

\setsansfont{RobotoFlex-VariableFont}[Scale=
   MatchUppercase]
\exemple\leg{MatchUppercase mais pas les bas de casse}
```

résultat

HTHT, bquxbqux *MatchLowercase mais pas les capitales*

HTHT, bquxbqux *MatchUppercase mais pas les bas de casse*

Les axes YTLC, YTAS et YTDE des fontes variables permettent d'ajuster les deux hauteurs en même temps :

Exemple 19 – Harmonisation de hauteurs, 3 : avec les axes YTLC, YTAS et YTDE des fontes variables

code

```
\setmainfont{EBGaramond-Regular.otf}
\setsansfont{RobotoFlex-VariableFont}[
  Scale = MatchUppercase,
  Renderer = OpenType,
  UprightFeatures = {RawAxis={YTLC=480,YTAS=770,YTDE=
   -310}}
]

\exemple\leg{Cette fois ça fonctionne}
```

résultat

HTHT, bquxbqux *Cette fois ça fonctionne*

Notons toutefois que si cette dernière solution résout bien notre problème, elle manque de généralité dans la mesure où il faut donner « à la main » (donc par tâtonnements ou par calculs compliqués comme dans l'exemple 13 page 33) les valeurs des axes.

Axe GRAD : niveau (*Grade*)

Altération des traits et formes (et donc du niveau de gris), sans changer la chasse. Cet axe agit globalement sur tous les caractères de la fonte, comme le montre le tableau 18.

Tableau 18 – Axe GRAD — changement du niveau de gris (nécessite d'être sous l'effet de `\aff{Renderer=OpenType}`)

Code	Résultat
`e\axe{GRAD=-200}e\axe{GRAD=150}e`	eee

Usage de GRAD

Une application évidente de cet axe GRAD est la modification locale du gris du texte. En particulier, ceci peut répondre au besoin classique de graphistes qui trouvent que les notes de bas de page n'ont pas le bon gris. La commande de l'exemple 20 page suivante[42] montre les effets d'éclaircissement ou de noircissement dûs à GRAD.

42. Il faudrait en fait modifier la définition de `\footnote` dans la classe actuelle en donnant à GRAD une valeur choisie par tâtonnement pour tout le document.

Exemple 20 – Axe GRAD — choix du niveau de gris pour les notes en bas de page

code

```
\addfontfeatures{Renderer=OpenType}

\newcommand{\manote}[2]{%
  \footnote{{\addfontfeatures{
        RawAxis={GRAD=#1}
      } GRAD=#1 : #2}}%
}

Une fois n'est pas coutume, nous illustrons notre propos
   \ldots{} par ces trois notes%
\manote{-200}{\Perec}%
\manote{0}{\Perec}%
\manote{150}{\Perec}!
```

résultat

Une fois n'est pas coutume, nous illustrons notre propos... par ces trois notes[a][b][c] !

a. GRAD=-200 : L'auteur étude les fois que le lancement de la tomate il provoquit la *réaction yellante* chez la Chantatrice et demonstre que divers plusieures aires de la cervelle elles était implicatées dans le response, en particular, le trajet légumier, les nuclei thalameux et le fiçure musicien de l'hémisphère nord. [24]

b. GRAD=0 : L'auteur étude les fois que le lancement de la tomate il provoquit la *réaction yellante* chez la Chantatrice et demonstre que divers plusieures aires de la cervelle elles était implicatées dans le response, en particular, le trajet légumier, les nuclei thalameux et le fiçure musicien de l'hémisphère nord. [24]

c. **GRAD=150 : L'auteur étude les fois que le lancement de la tomate il provoquit la *réaction yellante* chez la Chantatrice et demonstre que divers plusieures aires de la cervelle elles était implicatées dans le response, en particular, le trajet légumier, les nuclei thalameux et le fiçure musicien de l'hémisphère nord. [24]**

Autres axes

Nous venons d'examiner tous les axes de RobotoFlex que nous venons de tous citer ne sont que quelques-uns parmi la centaine d'axes « privés » (*custom*) acceptés par OpenType[43].

Nous allons montrer ici certains de ces axes, en utilisant d'autres fontes variables[44] que RobotoFlex (puisque bien sûr ces axes n'y sont pas implémentés) et en cherchant surtout à montrer la diversité des propriétés modifiées mais sans vouloir être complet ! Nous essayons de les classer selon la nature de la modification apportée.

Modifications sur la métrique des caractères

Outre les axes déjà cités, YTUC, YTLC, YTFI,... qui affectent des longueurs dans le dessin des glyphes, mentionnons :

43. Il est assez difficile d'en trouver une liste exhaustive. On consultera par exemple https://fonts.google.com/metadata/fonts.

44. À nouveau, ces fontes sont librement téléchargeables, par exemple depuis des sites comme GitHub, CTAN, Google Fonts, etc.

Axes SERF et SELE : modification des patins

SERF (*SERIF*) permet de donner une forme plus ou moins triangulaire aux empattements. Voir l'exemple 21.

Exemple 21 – L'axe SERF modifie la forme des empattements

code

```
% Requiert le package `trimclip` et la
% commande \zoomserif définie exemple 8 page 24
\setmainfont{GRADUATE}[Renderer=OpenType]
%
\axe{SERF= 0}\zoomserif{I}\par
\axe{SERF=15}\zoomserif{I}\par
\axe{SERF=30}\zoomserif{I}
```

résultat

SELE (*Largo Serif*) permet le rétrécissement d'empattements. Voir l'exemple 22.

Exemple 22 – L'axe SELE amincit les empattements

code

```
% Requiert le package `trimclip` et la
% commande \zoomserif définie exemple 8 page 24
\setmainfont{GRADUATE.ttf}[Renderer=OpenType]
%
\axe{SELE=  0}\zoomserif{I}\par
\axe{SELE=-10}\zoomserif{I}\par
\axe{SELE=-20}\zoomserif{I}
```

résultat

Axe ENLA (*ENLArge*)

Agrandir l'œil de certaines minuscules, comme cela se faisait dans des écritures anciennes. Le tableau 19 page suivante traite notamment le caractère Unicode U+01A3 (*latin small letter OI*) qui a été utilisé dans diverses orthographes latines de langues turques, comme l'ouïghour ou le tatar, et dans celle du tadjik.

TABLEAU 19 – L'axe ENLA agrandit les œils. Fonte utilisée ici : Recursive-VariableFont_CASL,CRSV,MONO,slnt,wght.ttf, avec la fonctionnalité Renderer=OpenType.

Code	Résultat
Hab\char"01A3	Habɋ
\axe{ENLA=050} Hab\char"01A3	Habɋ
\axe{ENLA=100} Hab\char"01A3	Habɋ

Modifications de forme et de positionnement des glyphes

Des variations sont appliquées (glyphe par glyphe) de façon à modifier leur positionnement dans la ligne (en *x*, *y*, avec rotation…) et parfois leur forme. Le but est souvent de donner une impression d'aléatoire comme dans l'écriture manuscrite. La fonte utilisée dans les exemples de cette sous-section est ShantellSans-VariableFont_BNCE,INFM,SPAC,wght.ttf.

Axe INFM (*INForMality*) : allure d'écriture manuscrite

Déformation de la taille et de la forme des lettres pour donner une allure d'écriture manuscrite. Comme on le constate avec le tableau 20, les déformations restent les mêmes pour chaque occurrence d'une lettre pour la même valeur d'axe (il n'y a rien d'aléatoire).

TABLEAU 20 – L'axe INFM déforme les caractères. Il nécessite d'être sous l'effet de \aff{Renderer=OpenType}.

Code	Résultat
manuscrit	manuscrit
\axe{INFM= 50} manuscrit	manuscrit
\axe{INFM=100} manuscrit	manuscrit

Axe BNCE (*BouNCy*) : dynamique, gonflable

Les caractères sont plus ou moins soulevés ou baissés (voir le tableau 21) ; cet axe est souvent utilisé pour composer des animations.

TABLEAU 21 – L'axe BNCE décale verticalement chaque caractère. Il nécessite d'être sous l'effet de \aff{Renderer=OpenType}.

Code	Résultat
ondulé	ondulé
\axe{BNCE=-100} ondulé	ondulé
\axe{BNCE= 100} ondulé	ondulé

Axe SPAC (*space*) : agrandissement de l'approche gauche

L'approche gauche des caractères est agrandie, donnant alors l'équivalent d'un interlettrage. Voir le tableau 22 page ci-contre.

TABLEAU 22 – L'axe SPAC modifie l'approche gauche de chaque caractère. Il nécessite d'être sous l'effet de `\aff{Renderer=OpenType}`.

Code	Résultat
`Élargi`	Élargi
`\axe{SPAC= 50} Élargi`	É l a r g i
`\axe{SPAC=100} Élargi`	É l a r g i

Modification du gris, de la lisibilité…

Outre GRAD et opsz déjà vus, les axes ARRR et SOFT entrent dans cette catégorie.

Axe SOFT (*SOFTness*) : adoucissement

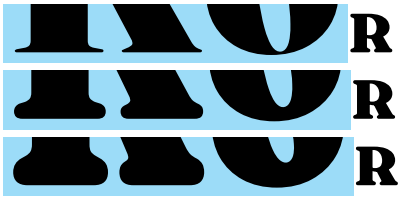
Complément de variations *Optical Sizing* ; selon l'intuition du dessinateur, il peut s'agir d'un arrondissement des extrémités, d'un agrandissement des contre-poinçons ou de l'épaisseur de fûts…Voir l'exemple 23.

Exemple 23 – L'axe SOFT « adoucit » l'allure des caractères

code

```
\fontspec{Fraunces-VariableFont_SOFT,WONK,opsz,wght.ttf}
\addfontfeatures{Renderer=OpenType}
\newcommand{\glyphwithfocus}[2]{%
  \clipbox{2pt 0pt 1pt 55pt}
  {%
    \colorbox{cyan!35}{%
      \fontsize{100}{100}\selectfont%
      \axe{#1}#2%
    }%
  }%
}
\huge
\glyphwithfocus{SOFT=  0}{Ro}R\par
\glyphwithfocus{SOFT= 50}{Ro}R\par
\glyphwithfocus{SOFT=100}{Ro}R
```

résultat



Axe CNTR (*CoNTRast*) : constraste et gris du texte

Avec la didone expérimentale Amazonia Beta-VF, le contraste est modifié par variation de l'épaisseur des déliés. Dans le tableau 23, on constate que l'effet est visible sur gros corps mais que le résultat n'est pas flagrant pour des corps courants.

TABLEAU 23 – L'axe CNTR modifie l'épaisseur des déliés. Il nécessite d'être sous l'effet de \aff{Renderer=OpenType}. Fonte utilisée ici : Amazonia Beta-VF.ttf.

Code	Résultat
\axe{CNTR= 0} AbdE	AbdE
\axe{CNTR= 50} AbdE	AbdE
\axe{CNTR=100} AbdE	AbdE

Axe ARRR (*Augmented Reality Retinal Resolution*)

Optimisation du dessin des lettres pour en augmenter la lisibilité en se basant sur la résolution rétinienne de l'engin d'impression[45]. Sont notablement affectés : les « trous d'encre », les pointes et extrémités, etc. Ceci est surtout utile pour les très petites définitions — et visible sur les très gros corps, comme on le constate avec l'exemple 24.

Exemple 24 – L'axe ARRR augmente la résolution et, ce faisant, la lisibilité

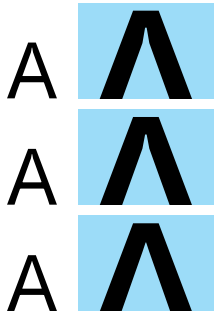
code

```
% Requiert les packages `pgffor` et `trimclip`
\fontspec{AROneSans-VariableFont_ARRR,wght.ttf}
\addfontfeatures{Renderer=OpenType}
\fontsize{30}{40}\selectfont
\foreach \i in {10, 20, 30, 60}{
  \axe{ARRR=\i}A
  \fontsize{100}{40}\selectfont
  \clipbox{10pt 40pt 10pt 0pt}{%
    \colorbox{cyan!35}{%
      \axe{ARRR=\i}A%
    }%
  }\par
}
```

résultat

45. Pour en savoir plus, voir https://github.com/niteeshy/ar-one-sans. Cette fonte est censée être adaptée aux univers virtuels en 3D tels que les offre la réalité augmentée. Elle a été développée à l'université de Reading, dont la réputation en matière d'arts graphiques n'est plus à faire. S'il semble évident que cette fonte répond au problème des pièges à encre, fréquent dans les très petits corps, nous avons un peu plus de mal à faire le lien avec la réalité augmentée — notre expérience professionnelle en matière de réalité augmentée, tout comme virtuelle, ne nous ayant jamais fait rencontrer de problème de composition lié à la résolution rétinienne ; la dénomination de cet axe n'aurait-elle pas été influencée par l'air du temps ?

résultat (suite)



Substitution de style

Tout comme les axes `ital` et `slnt` permettent de changer de style (de romain à italique ou penché), d'autres axes permettent de passer à un autre style, au sens large : par exemple de sans-sérif à mono avec ou sans patins (axe `MONO`), ou de romain cursif (`CRSV`). Ces changements peuvent aller jusqu'à des déformations très à la mode (`WONK`).

Certains de ces axes sont en fait binaires : il n'y a pas de variation mais seulement (selon que la valeur de l'axe est inférieur ou supérieur à une limite) passage d'un état à l'autre. C'est le cas de l'italique de la figure 7 page 19 et ce sera par exemple le cas, ci-après, de l'axe `MONO`. On peut alors se demander pourquoi les dessinateurs n'ont pas simplement utilisé une propriété *font-variant* d'OpenType.

Certains de ces axes de substitution sont illustrés dans les exemples de cette sous-section qui utilisent la fonte `Recursive-VariableFont_CASL,CRSV,MONO,slnt,wght.ttf`.

Axe `MONO` (*monospace*)

Passage de sans-sérif (avec chasse) à caractère à chasse fixe ; le tableau 24 montre que des patins sont ajoutés à certaines lettres à partir d'une certaine valeur.

Tableau 24 – L'axe `MONO` rend la chasse fixe… et n'oubliez pas les patins ! Il nécessite d'être sous l'effet de `\aff{Renderer=OpenType}`.

Code	Résultat
`Trial Segment`	Trial Segment
`\axe{MONO=0.50} Trial Segment`	Trial Segment
`\axe{MONO=0.51} Trial Segment`	Trial Segment
`\axe{MONO=1 } Trial Segment`	Trial Segment

Axe `CRSV` (*Cursive*)

Passage d'un caractère droit à cursif. L'axe ne prend que trois valeurs : 0 (droit), 1/2 et 1 (cursif). Voir le tableau 25 page suivante.

Axe `CASL` (*Casual*) : désinvolte, flippant

Les lettres perdent leur rigidité au profit d'une allure plus décontractée… on l'observe avec le tableau 26 page suivante.

TABLEAU 25 – L'axe CRSV influe sur la cursivité des caractères. Il nécessite d'être sous l'effet de `\aff{Renderer=OpenType}`.

Code	Résultat
`Quel fayot!`	Quel fayot !
`\axe{CRSV=0.5} Quel fayot!`	Quel fayot !
`\axe{CRSV=1} Quel fayot!`	Quel fayot !
`\axe{CRSV=1,MONO=1} Quel fayot!`	Quel fayot !

TABLEAU 26 – L'axe CASL n'est utilisé, Outre-Atlantique, que le vendredi, pour le *casual Friday*. Il nécessite d'être sous l'effet de `\aff{Renderer=OpenType}`.

Code	Résultat
`Skieur`	Skieur
`\axe{CASL=0.6} Skieur`	Skieur
`\axe{CASL=0.9} Skieur`	Skieur

Axe WONK (*Wonky*) : bancal, détraqué, fêlé…

Substitution de caractères. Puisqu'il n'y a aucune « variation », on se demande pourquoi l'auteur de la fonte n'a pas choisi d'utiliser un *stylistic set*. Voir l'exemple 25.

Exemple 25 – L'axe WONK rend gentiment aléatoire la fonte, par substitution de caractères

code

```
\Huge
% Normal
\fontspec{       Fraunces-VariableFont_SOFT,WONK,opsz,
   wght.ttf}
\addfontfeatures{Renderer=OpenType}
\axe{WONK=0}voyou \& humain\par
\axe{WONK=1}voyou \& humain\par
% Italique
\fontspec{Fraunces-Italic-VariableFont_SOFT,WONK,opsz,
   wght.ttf}
\addfontfeatures{Renderer=OpenType}
\axe{WONK=0}voyou \& humain\par
\axe{WONK=1}voyou \& humain
```

résultat

voyou & humain
voyou & humain
voyou & humain

résultat (suite)

voyou & humain

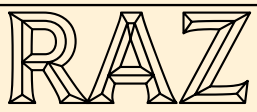
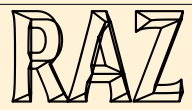
Caractères en 3D

Certains caractères imitent des caractères en 3D. Divers axes permettent de donner l'impression de les faire bouger, par exemple de les tourner (voir ci-dessous l'axe XROT) ; d'autres (comme EDPT) modifient leurs contours. Ceci est évidemment à des fins d'animation spatiale…

Axe XROT (*X ROTation*) : rotation autour d'un axe vertical

Comme on le constate avec le tableau 27, cet axe modifie une lettre représentant ce caractère en 3D de façon qu'on ait l'impression qu'il y a eu une rotation selon un axe vertical (vers la gauche ou la droite). Un autre axe existe, YROT, qui fait pencher ces mêmes images 3D en avant ou en arrière ; nous ne l'illustrons pas ici, mais nous vous invitons à le tester.

Tableau 27 – L'axe XROT crée, en deux dimensions, un effet de vue en perspective sur un caractère qui en comporte trois. Il nécessite d'être sous l'effet de `\aff{Renderer=OpenType}`. Fonte utilisée ici : `TiltPrism-Regular-VariableFont_XROT,YROT.ttf`.

Code	Résultat
`RAZ`	
`\axe{XROT=-45} RAZ`	
`\axe{XROT=+45} RAZ`	

Couleur

Il est possible colorier des caractères, non seulement d'une couleur unique mais de façon très structurée pour donner des effets spéciaux[46]. OpenType propose des axes, dont CPAL et COLR. Nous nous contentons de les citer ici.

Animation

Dès le début de cet article (voir la section « Animation des fontes variables », page 26), nous avons signalé que les fontes variables sont très utilisées pour en faire des animations et que nous n'en parlerions pas ici. Disons cependant que certains axes sont définis en vue de ces animations, dont TIME.

Axe TIME

L'idée est d'appeler plusieurs formes d'un même caractère comme s'il s'agissait des diverses étapes de ce caractère en mouvement. Le caractère a toujours le même code, mais en faisant varier la valeur de l'axe TIME, on obtient des images différentes.

46. Pour ne donner qu'un exemple, voir https://www.fontself.com/colorfontweek/2018#abelone !

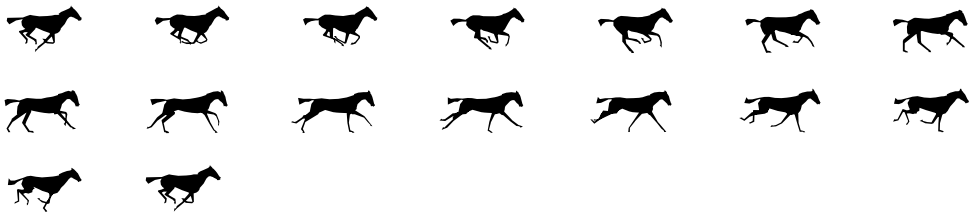
Un très bon exemple est celui de la fonte Muybridge[47]. Elle n'a qu'un caractère (Unicode U+1F40E, HORSE), mais il lui correspond 16 images d'un cheval au galop comme on le voit avec l'exemple 26... et sur chaque page impaire de ce numéro, où des occurences consécutives de ce cheval remplacent le numéro de page : feuilleter le numéro crée bien l'illusion du galop.

Exemple 26 – L'axe TIME donne un glyphe différent pour chaque valeur qui lui est attribuée

code

```
% Requiert le package `pgffor`
\fontsize{30}{30}\selectfont
\fontspec{vertopal.com_MuybridgeGX.ttf}
\addfontfeatures{Renderer=OpenType}
\newcommand{\imageno}[1]{\axe{TIME=#1}\char"1F40E\ }
%
\foreach \i in {0,1,...,15}{\imageno{\i}}
```

résultat



L'usage de Javascript et de l'extension animate permettrait d'avoir l'animation montrée ici : https://www.axis-praxis.org/playground/horse/.

CONCLUSION

En guise de conclusion, non pas sur la fonte RobotoFlex elle-même mais sur les fontes variables en général, nous aimerions prendre un peu de recul et faire quelques remarques.

Vraie nécessité ou expérimentation ?

Tout d'abord, il faut noter que le concept de fonte variable est relativement nouveau et qu'il a été adopté par un grand nombre de dessinateurs, notamment chez les jeunes dont les nouvelles créations sont toujours « variables ». En revanche il nous semble que l'ajout de variations dans les grandes fontes classiques se fait attendre (sans doute est-ce dû à la difficulté de la chose ?).

47. Du nom de l'Anglais qui a étudié le mouvement d'un cheval au galop grâce à un stroboscope. La fonte se trouve sur le site https://db.onlinewebfonts.com mais dans le format .woff. On l'a convertie en .ttf grâce au convertisseur www.vertopal.com/en/convert/woff-to-ttf.

Si la majorité des fontes variables disponibles aujourd'hui implémentent bien les quatre axes standards (et notamment ceux de la graisse `wght` et de la chasse `wdth`), on en trouve beaucoup moins qui proposent des axes privés. En fait, on a alors l'impression que les fontes qui présentent des axes nouveaux sont là plus pour expérimentation ou démonstration que pour réellement satisfaire les besoins des utilisateurs.

Composition ou graphisme ?

L'informatisation de la typographie a conduit à une désacralisation du métier de typographe. Nous tous, utilisateurs des logiciels TEX, faisons le travail longtemps réservé aux ouvriers du livre. Seule la partie en amont, la fabrication des caractères, était il y a encore peu réservée à une « élite », celle des graveurs de poinçons, puis des dessinateurs de caractères. Certes, on sait créer (ou plutôt simuler) des glyphes qui n'existent pas dans une fonte (par exemple des petites capitales ou des penchés), voire jouer avec des positionnements non standards (hauteur des exposants, espacements entre lettres, etc.), mais en principe on ne modifiait pas le dessin des caractères. Désormais, avec les fontes variables, et c'est justement leur objet, l'utilisateur peut modifier des glyphes en faisant varier certains paramètres.

Mais tous les utilisateurs n'ont pas le même rapport aux fontes[48] et ici nous distinguons ceux qui font de l'édition de textes (autrefois on disait du « travail de labeur ») de ceux qui sont plus graphistes (et qui dessinent des affiches, des couvertures, voire des pages de sites web, etc.). Nous appelons ici *compositeurs* les premiers et *graphistes* les seconds.

Frutiger a défini une vingtaine de fontes numérotées de 45 à 83 (voir page 5). En jouant sur les axes `wdth` et `wght`, on pourrait[49] aujourd'hui en définir une infinité, dont celles de la figure 19 page suivante. Frutiger avait limité son nombre de variantes pour des raisons de coût (il fallait graver tous les poinçons correspondants, ce qui coûtait cher !) mais aussi sans doute pour n'offrir que ce qui est raisonnable : a-t-on besoin de caractères hyper-gras et hyper-condensés ? OpenType tranche : c'est-à l'utilisateur de décider… Si c'est l'auteur de la fonte qui prévoit les variations, on n'a rien à dire[50] ; sinon ?

Nous avons vu (en figure 2 page 3) que Fournier proposait diverses fontes en modifiant la taille des jambages et celle des hampes (donc du rapport œil/hauteur des capitales), mais pas seulement l'une ou l'autre comme dans l'exemple 27 page suivante.

48. Les lecteurs des *Cahiers GUTenberg* se souviennent sûrement de l'approche philosophique d'Emmanuel Souchier [25] basée sur des concepts de communication et de linguistique ; finalement cette approche revient un peu à notre vision matérialiste.

49. Nous ne connaissons pas de fonte variable basée sur `Univers`. Mais comme RobotoFlex lui ressemble, c'est cette fonte qu'on utilise en figure 19 page suivante, où les numéros 19 et 91 sont des inventions personnelles. Notons que Frutiger permettait une contraction plus grande que celle proposée par RobotoFlex !

50. On lit souvent dans les licences des fontes classiques des mentions telles que « *Your licence is permission to use these fonts ; you do not own them and may not alter them in any way whatsoever.* » (en français « La licence de ces fontes ne vous permet que leur utilisation ; vous n'en êtes pas propriétaire et ne pouvez en aucun cas les modifier. ») Il faut espérer que ces mentions sont désormais corrigées dans le cas des fontes variables qui laissent justement aux utilisateurs la possibilité d'altérer les fontes !

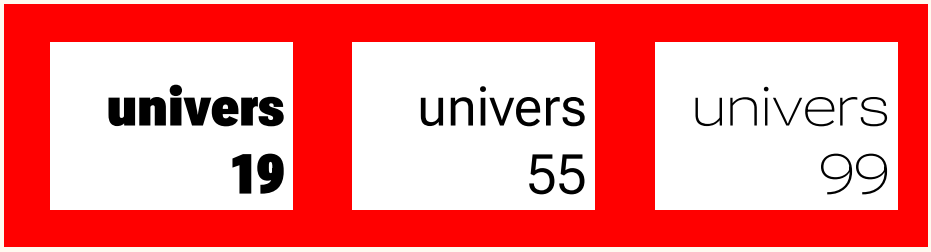


FIGURE 19 – Frutiger accepterait-il ces variations parmi ses propres fontes? Voir page 5.

Exemple 27 – Utilisation différenciée des axes YTAS et YTDE

code

```
\aff{Renderer=OpenType}
\axe{YTAS=854}
\Large
                mes poules pondent bien peu\par
\axe{YTDE=-305} mes poules pondent bien peu\par
\axe{YTDE= -98} mes poules pondent bien peu
```

résultat

mes poules pondent bien peu
mes poules pondent bien peu
mes poules pondent bien peu

... où la troisième ligne ne sera pas acceptée par un compositeur (à cause du déséquilibre entre les hampes et les jambages, la lisibilité n'est pas bonne). Les fontes variables permettent ainsi beaucoup de variations, mais sans contrôle...

Le succès des fontes variables sur le web (succès également dû à l'influence du W3C et ses CSS), à notre avis plus important qu'en édition traditionnelle, vient du fait que ce sont des *graphistes* qui écrivent pour le web et non des *compositeurs* (en donnant à ces métiers le sens du début de cet item).

Variations limitées

Si l'on regarde le catalogue des variations proposées par les fontes variables (voir page 48 et suiv.), on voit que toutes[51] ces variations modifient le dessin de tout ou d'une partie des glyphes d'une fonte et concernent :

- des variations de propriétés géométriques de tous les glyphes d'une fonte (chasse, graisse, ...), ou seulement de certains glyphes (hauteur des hampes et des jambages, longueur ou pente des sérifs, ...),
- des variations de propriétés non géométriques de tous les glyphes d'une fonte (couleur, allure 3D, ...). On peut ranger ici les variations dont l'objet est le gris du texte, ou la lisibilité (GRAD, ARRR, SOFT, etc.).
- des variations de style pour tous les glyphes d'une fonte (mais parfois ça ne concerne que quelques caractères). C'est le cas des variations MONOspaces, CuRSiVe, CASuaL, WONKy, etc.

51. Ou à peu près toutes; penser aux fontes comme Muybridge (en section « Axe TIME », page 55).

Mais, à part quelques tentatives comme INFM (déformation d'allure manuscrite), voire à part certains (vieux !) ensembles stylistiques des fontes OpenType (par exemple les formes fleuries ou lettres ornées de EBGaramond), voire à part l'italique, les fontes variables n'offrent pas de variations stylistiques. Or l'un des buts des fontes variables est de remplacer une collection de fontes OTF par une fonte *unique*. L'impossibilité de réaliser ce rêve [52] vient du fait qu'on ne sait pas aujourd'hui, au vu de deux *masters* représentant deux des M de la collection du TRW (figure 20), passer de l'un à l'autre comme on a pu le faire par exemple pour les deux « u » (plus ou moins gras) de la figure 5 page 12.

FIGURE 20 – Début des « M » de la collection du TRW (*Typenrepertorium der Wiegendrucke*), Berlin ; extrait de [26] permettant de classifier les styles des caractères imprimés des incunables.

En principe les variations programmées concernent tous les glyphes d'une fonte et non un seul. À fortiori, elles ne sont pas prévues pour faire varier une partie (voire plusieurs différemment) d'une lettre. Les graphistes amateurs d'hybridation et autres déformations de caractères (que dès 1925 combattait déjà Thibaudeau, voir figure 21, du moins pour la typographie de labeur) doivent alors utiliser les outils classiques de dessins de caractères…

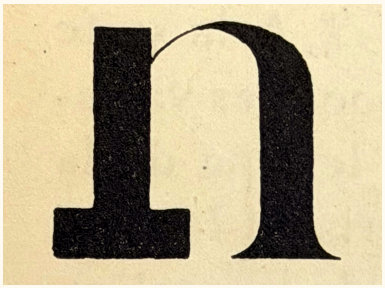


FIGURE 21 – Ce que ne permettent pas (a priori) les fontes variables.

« Il devient impossible […] de n'être pas *choqué* […] du mélange dans la même lettre de plusieurs formes d'empattements, [par exemple comme ci-contre] d'une tête de jambage sans empattement (type de l'Antique) avec base quadrangulaire personnelle à l'Égyptienne et le double jambage à terminaison triangulaire de l'Elzévir. »
Francis Thibaudeau, *Manuel français de typographie*, 1924 [27].

Quelques limitations

Les variations des glyphes d'une fonte variable sont conduites par la variation d'une valeur numérique affectée à un paramètre (comme wdth ou MONO). Ces valeurs numériques sont soit continues dans un intervalle (dont les limites sont celles appelées *minimum* et *maximum* du triplet lié à un axe), soit en nombre limité ; elles sont choisies par le dessinateur de la fonte.

Pertinence des valeurs utilisées

Les variables continues sont les plus fréquentes et correspondent à des grandeurs géométriques des caractères. Mais à part quelques cas (comme l'axe slnt qui suppose en général indiquer un angle exprimé en radian ou en degré), elles ne correspondent pas à une unité physique (et notamment pas à une longueur)

52. Tant qu'à rêver, on pourrait rêver d'une fonte unique qui par le simple jeu des variations permettrait de tracer tous les caractères typographiques du monde entier, passé et à venir !

directement liée à un axe. Ce ne sont pas des mesures[53] mais plutôt des échelles. En principe (voir la section « Valeurs des axes » en page 14), celles-ci ne sont pas obligatoirement linéaires ni même uniformes (voir la variation de chasse en figure 11 page 32).

Avec un peu d'habitude (à condition que les dessinateurs utilisent les même triplets), un utilisateur connaîtra la différence entre une graisse de 150 et une de 800. Il en est sans doute de même pour les axes de hauteur des capitales et bas-de-casses, ou de la hauteur/profondeur des ascendantes/descendantes. Mais on peut regretter qu'on ne nous offre pas directement quelque chose comme le rapport « hauteur des hampes/hauteur d'œil » ; et tant qu'à jouer avec ces morceaux de lettres, pourquoi ne pas jouer sur d'autres paramètres comme le faisait Donald Knuth en proposant les 62 paramètres des fontes `Computer Modern` (voir la figure 22 page ci-contre et l'ouvrage de Yannis Haralambous [5, p. 920-924]). On y revient plus bas.

D'autres variations « continues » sont beaucoup moins faciles à comprendre ou assimiler… Typiquement, il est recommandé[54] que l'axe de correction optique (`opsz`) prenne des valeurs liées à un rapport entre le corps et la distance de lecture et si la variation consiste à choisir des spécificités prédéfinies (par exemple diminution de l'épaisseur des fûts ou agrandissement des œils) la façon d'y arriver est laissée au soin du dessinateur. Là encore, l'utilisateur devra expérimenter pour avoir l'effet désiré. Il en est de même pour des axes comme `XTRA` (gris du texte), `GRAD`, etc.

Variante stylistique ou véritable variation ?

Certaines variations sont discontinues, c'est-à-dire que la valeur de leur axe ne prend que quelques valeurs discrètes. Par exemple, la fonte `Recursive` a un axe (`CRSV`, cursive) qui donne une forme cursive aux caractères selon qu'il est en deçà ou au delà d'une certaine valeur (0.8) ; il se comporte donc comme un booléen. C'est aussi le cas de l'italique (à partir d'une certaine valeur de `ital`, certains glyphes sont modifiés) mais ce côté booléen est caché par le fait que les valeurs modifient aussi une propriété géométrique (l'angle). Certains utilisateurs pourront se demander s'ils en sont pas ici en présence d'une variante stylistique et s'il ne s'agit pas alors d'une autre fonte. C'est ce que font de nombreuses fontes, dont RobotoFlex, `Junicode`, etc. pour lesquelles on dispose de deux fichiers, par exemple `JunicodeVF-Roman.ttf` et `JunicodeVF-Italic.ttf`.

53. Ceci est tout à fait dans l'esprit du monde de la création de caractères. Déjà, du temps de la gravure des poinçons, le typographe n'utilisait aucune règle graduée mais des outils de comparaison (calibres, compas à pointe sèche, etc.), tout le travail se faisant donc proportionnellement (plus grand que, identique, moitié…). Aujourd'hui un dessinateur travaillant avec FontForge ou Glyphs utilise en fait une grille où il place des objets par rapport à telle ou telle ligne) et où, si des mesures peuvent quand même être effectuées, elle le sont de manière relative, par exemple en 1/1000e de… on ne sait pas quoi finalement.

54. https://learn.microsoft.com/en-us/typography/opentype/spec/dvaraxistag_opsz

FIGURE 22 – Quelques-uns des 62 paramètres utilisés par METAFONT pour générer la famille Computer Modern (Extrait de [5]).

Renaissance d'un débat ancien

Le processus de variation d'une fonte se fait en deux temps. D'abord, lors de la création d'une fonte variable, le dessinateur trace les *masters* extrêmes d'un axe (voir figure 5 page 12) et donne des indications sur les vecteurs passant de l'un à l'autre. Ensuite, lors de l'utilisation d'une occurrence particulière d'une fonte, à l'exécution, grâce aux indications du dessinateur, le système calcule les courbes des glyphes de la fonte pour la valeur choisie d'un axe. Le dessinateur n'a rien « programmé » : son travail revient à donner le tracé d'une courbe de départ et celui de la courbe quand elle sera modifiée ; sauf que les « indications » qu'il doit donner sont de plus en plus complexes et qu'il faut remplir de nombreux formulaires, tables, etc. (que les graphistes ne connaissent pas sous le nom de « paramètres »).

On se souvient que le dessinateur Gerard Unger[55] ne voulait pas se servir de METAFONT car il ne se considérait pas comme programmeur et encore moins comme « paramétreur ». Certes, il faut programmer pour utiliser METAFONT et les outils équivalents. Mais il n'avait pas compris que ce que proposait justement METAFONT, c'était des fontes variables[56] avant la lettre. En fait, le différend entre les dessinateurs utilisant Glyphs et les informaticiens avec METAFONT n'est pas (directement) une question de programmer ou pas. Les premiers disent ce qu'ils veulent obtenir (en fournissant donc les masters extrêmes d'une variation liée à un axe) tandis que les seconds disent comment obtenir ces extrêmes. Ce n'est pas le lieu ici de comparer les méthodes inductives et celles déductives, mais l'émergence des fontes variables pourrait être l'occasion de relancer le débat lancé il y a plus de 40 ans par Donald Knuth [28].

Jacques André & Patrick Bideault

Remerciements

Les auteurs remercient Daniel Flipo pour son aide en matière de fontes, ainsi que Bernard Peyréga, Bastien Dumont et Denis Bitouzé, qui ont aidé à mettre cet article en page.

Note de la rédaction

Le présent article comporte de nombreuses illustrations et exemples de code, ce qui rend complexe son intégration au sein de la *Lettre*.

Divers messages adressés aux adhérents de l'association témoignent même des difficultés rencontrées.

Vous constaterez que certains bas de page sont un peu vides. C'est la solution que nous avons trouvée ; elle est imparfaite, mais privilégie la lisibilité du texte et nous semble permettre une consultation agréable des différents documents et exemples de code illustrant l'article.

55. En réponse à l'article de Donald Knuth sur son concept de Meta-font [28], Gerard Unger avait écrit : « *Besides being a designer, I have no objection to acting as a system operator ; but I don't want to become a programmer…let alone a parameterizer.* » (en français « En plus de mon rôle de designer, je n'ai aucune objection à agir en tant qu'opérateur système ; mais je ne veux pas devenir programmeur… et encore moins paramétreur. ») [29, p. 354]

56. METAFONT ne produit aujourd'hui que des fontes statiques. Il ne doit pas être compliqué de générer les occurrences dynamiques lors de l'exécution du source, la difficulté étant toutefois le passage de la valeur demandée pour un axe, et de faire les transformations correspondantes.

ANNEXES

OTF et Lua

Nous donnons ici quelques pistes pour analyser une fonte OpenType telle qu'elle est traduite en fichiers `.lua` lors de son traitement par fontspec. LualaTeX n'utilise pas directement le mécanisme des fontes de TeX, mais celui mis au point par Hans Hagen pour ConTeXt, et en particulier l'extension `luaotfload` [19].

Les fontes OpenType (y compris les fontes variables) sont décrites par des arborescences dans leur fichier binaire `RobotoFlex.ttf` ou `RobotoFlex.otf`, etc. On peut en avoir une version lisible (en ASCII) en en demandant une copie par un outil de gestion de fontes tel que Glyphs (le fichier est alors transcrit en `RobotoFlex.glyphs`) ou FontForge (fichier de sortie : `RobotoFlex.sfd`). De son côté, LualaTeX[57] transcrit chaque fonte OpenType en deux fichiers ; ce sont ici `RobotoFlex.lua` et `RobotoFlex.luc`. De type `octet-stream/mime`, ce dernier est hélas peu lisible ; mais il contient les descriptions des glyphes par contours.

Le fichier `RobotoFlex.lua` correspond aux arborescences du fichier OpenType (voir notre article [17, p. 52]) et est dans un format lisible.

Le listing 1 montre le fichier `.lua` de la fonte RobotoFlex. Le sous-arbre le plus important en terme de OpenType est celui `resources`. Deux nœuds sont spécifiques aux fontes variables :

instancenames, qui donne la liste des instances (voir la section « Instances », page 17) ;

et variabledata, où sont collectées toutes les informations liées aux axes.

C'est l'accès à ces données que nous allons montrer.

Accès au fichier d'une fonte OTF par Lua

En fait, le fichier d'une fonte OTF utilisé par LualaTeX n'est pas directement celui mis dans le cache sous forme `RobotoFlex.lua` et `RobotoFlex.luc`, mais, afin d'être compatible avec le traitement par NFSS, une copie dont la structure interne est légèrement modifiée.

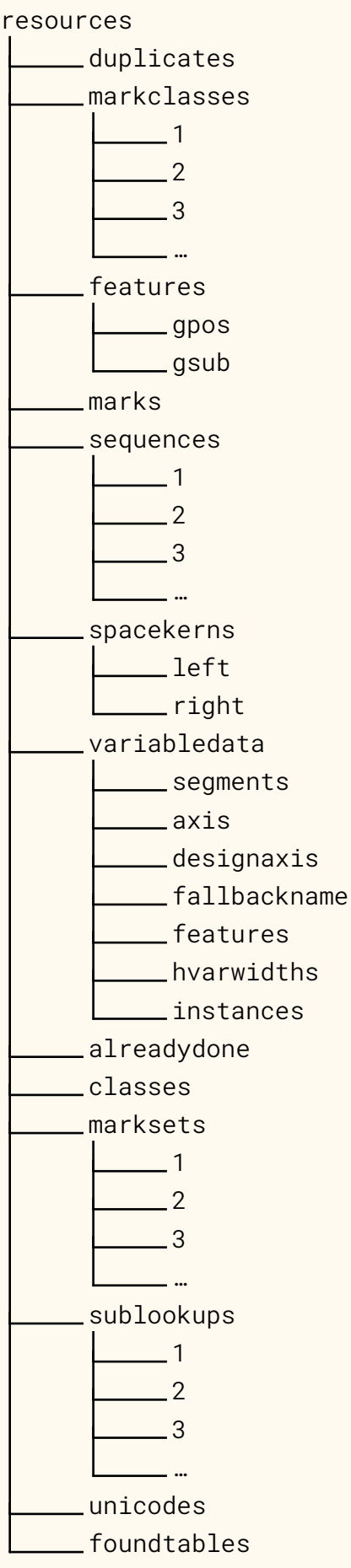
Listing 1 – Arbre de `RobotoFlex.lua` tel qu'il est en mémoire cache. Les coupes (marquées par ...) sont pour les besoins de cet article !

```
return {
  ["cache_version"] = 3.134,
  ["compacted"] = true,
  ["creator"] = "context mkiv",
```

57. Ceci est fait par les opérations de l'extension fontloader-luaotfload de Ulrike Fischer [30]. Tous ces fichiers `.lua` et `.luc` sont mis dans `…/texlive/2025/texmf-var/luatex-cache/generic/fonts/otl` ; voir la section « Fontes variables et LualaTeX » en page 19.

```
  ["descriptions"] = {
    [0] = {…}, -- NULL
    …
    [33] = { -- 33 = code décimal de point d'exclamation ( hexa = 21)
      ["boundingbox"] = { 186, -22, 434, 1456 },
      ["index"] = 4,
      ["name"] = "uni0021",
      ["unicode"] = 33,
      ["width"] = 620,
    },
  }
  ["format"] = "truetype",
  ["goodies"] = {},
  ["metadata"] = {
    ["ascender"] = 1900,
    ["averagewidth"] = 1145,
    ["boundingbox"] = { -345, -600, 2580, 2461 },
    ["fontname"] = "RobotoFlex-Regular",
    …
    ["version"] = 65536,
  },
  ["instancenames"] = { "thin", "extralight", "light",… }, -- instances des VF
  ["properties"] = {…},
  ["resources"] = {…     -- principales tables Open Type
    ["features"] = {
      ["gpos"] = {…},
      ["gsub"] = {…}
    }
  }
  ["unicodes"] = {-- tous les car  par glyphname
    ["acutecombstack.case"] = 983065,
    …
  },
  ["variabledata"] = {         -- Fontes variables
    ["axis"] = {               -- axes
      {
        ["default"] = 14.0,    -- triplet : valeur défaut
        ["flags"] = 0,
        ["maximum"] = 144.0,   -- max
        ["minimum"] = 8.0,     -- min
        ["name"] = "optical size",
        ["tag"] = "opsz",      -- code
      },
    },
    …
  }
  ["tableversion"] = 0.004,
  ["time"] = 1709187274,
}
```

Figure 23 – Sous-arbre LuaLaTeX de la table `resources` de la fonte `RobotoFlex-VariableFont.ttf`. Comparer avec la structure OpenType de la même fonte du listing 1 page 63.



L'accès aux fontes se fait directement (sans charger d'extensions) par LuaTEX [31, § 2.7], avec des commandes simples, comme le montre l'exemple 28, qui donne le numéro interne de la fonte choisie.

Exemple 28 – Accès au numéro de la fonte via la commande LuaTEX `\fontid`

code

```
\font\RobotoFlex="RobotoFlex-VariableFont.ttf"
\no\ de ma fonte = \fontid\RobotoFlex
```

résultat

n° de ma fonte = 793

Ce numéro sera éventuellement à utiliser plus tard pour nommer la dite fonte.

Les fontes (variables ou pas) utilisées par LualaTEX ont une structure en arbre, proche donc de celles OpenType, composée de tables et, intérieurement, de structures C [31, § 6]. On peut comparer l'arbre OpenType de RobotoFlex (voir le listing 1 page 63) avec celui interne à LualaTEX de la même fonte RobotoFlex (en figure 23 page précédente).

Outre celles utilisées de façon interne, diverses extensions ont été écrites (en Lua) pour la manipulation des fontes ; on en trouvera une liste dans l'article de Manuel Pégourié-Gonnard [32]. La majorité de ces macros est écrite pour les implémenteurs/développeurs et bien peu d'utilisateurs « ordinaires » de LualaTEX ont besoin de s'en servir. Certaines macros toutefois permettent simplement d'analyser une fonte (variable ou pas) et d'afficher certaines propriétés. Par exemple la liste des glyphes présents dans une fonte OpenType, ou les dimensions de la boîte englobante d'un caractère (*bbox*), ou, en ce qui concerne les fontes variables, les triplets de tel ou tel axe.

Communication entre LualaTEX et Lua

Ce que l'on veut faire, c'est manipuler une fonte par des routines écrites en Lua et appelées depuis un programme LualaTEX. Nous montrons ceci par le biais de l'exemple 29, qui est assez simple. Il imprime le sommet de l'arbre d'une fonte.

Exemple 29 – Document LualaTEX appelant une procédure écrite en Lua travaillant sur une fonte.

code

```
% Requiert le package `luacode`
\begin{luacode*}
function Lister(fontid)
  local lafonte = luaotfload.fontloader.fonts.hashes.identifiers[fontid]
  vardata = lafonte          -- contenu du fichier .resources
  tex.sprint("Premier niveau de l'arbre de la fonte \\#" ..fontid.."\\par")
  for k, v in pairs(vardata) do
    tex.sprint("\\verb|" .. k .. "|" .. ' = ')
    if (type(v) == "string" or type(v) == "number") then
      tex.sprint('\\verb|' .. v ..'|\\par')
    else
      tex.sprint(type(v) ..'\\par')
    end
  end
  tex.sprint('-- -- -- --')
end
\end{luacode*}

\setlength{\parindent}{0pt}
\font\RobotoFlex="RobotoFlex-VariableFont.ttf"
```

code (suite)

```
21
22  \directlua{Lister(\fontid\RobotoFlex)}
```

résultat

```
Premier niveau de l'arbre de la fonte #793
type = real
goodies = table
cache = yes
fullname = Roboto Flex Regular
name = RobotoFlex-VariableFont.ttf
specification = table
subfont = 1
properties = table
direction = 0
size = 655360.0
writingmode = horizontal
psname = RobotoFlex-Regular
encodingbytes = 2
unscaled = table
extend = 1000
parameters = table
descriptions = table
nomath = boolean
streamprovider = 2
embedding = subset
format = truetype
designsize = 0
squeeze = 1000
filename = /home/patrick/.local/share/fonts/RobotoFlex-VariableFont.ttf
identity = horizontal
tounicode = 1
slant = 0
units = 2048
fontname = RobotoFlex-Regular
characters = table
resources = table
shared = table
units_per_em = 2048
----
```

Le programme maître est en LuaLaTeX ; l'appel à une procédure Lua se fait par la commande `\directlua` (en ligne 22). La procédure appelée est décrite dans l'environment `\begin{luacode*}`…`\end{luacode*}` (lignes 2-17) défini dans l'extension luacode appelée dans le prologue. On pourrait mettre son contenu dans un fichier `mesprocs.lua` qui est à appeler par la commande :

```
\directlua{require 'mesprocs.lua'}.
```

Notre programme appelle donc la routine `Lister` (toujours en ligne 22) en lui passant la fonte que l'on veut examiner, notre habituelle RobotoFlex. Celle-ci est connue sous son nom de fichier (`RobotoFlex-VariableFont.ttf`), mais le système de gestion des fontes ne connaît les fontes que par leur numéro d'entrée dans un tableau interne, numéro accessible par la commande `\fontid` (voir la section « Accès au fichier d'une fonte OTF par Lua » en page 63). C'est donc ce numéro qu'on fournit (lignes 20-22).

Du côté Lua, notre fonction `Lister` accède à la fonte, laquelle est dans le tableau `identifiers[fontid]`… qui est lui-même en mémoire à la ligne 4 :

```
luaotfload.fontloader.fonts.hashes
```

… adresse interne du *fontloader* sur laquelle nous n'en dirons pas plus, sauf qu'on la retrouve dans tous les programmes Lua travaillant sur les fontes.

Les fonctions Lua (ici il n'y en a qu'une) sont écrites en Lua[58].

Le programme commence à sauver dans `lafonte` la fonte étudiée (ligne 4), et dans `vardata` la table qu'on veut analyser. Ici, c'est la fonte elle-même, mais on pourrait choisir une sous-table (dont la liste va apparaître en sortie, au bas de l'exemple 29 page 66), par exemple `lafonte.resources`.

`tex.sprint("…")` (ligne 6) est une commande de communication de Lua vers le site hôte TEX (`sprint` est l'abréviation de `string print`). À noter que la contre-oblique a un rôle spécial dans les chaînes de caractères Lua et doit être doublée pour être transmise à TEX. Ici (ligne 6) notre paramètre est en fait `"…" ..fontid.. "…"` où `..` indique la concaténation de chaînes, et où `fontid` est écrit sous sa forme alphanumérique (ici « 793 »). Notre programme LualaTEX reçoit donc le texte :

```
Premier niveau de l'arbre de la fonte \#793\par
```

… qui sera donc la première ligne imprimée en sortie.

La table `vardata` (ici donc la fonte) est une succession d'éléments de la forme `(nom, valeur)`. C'est elle que parcourt notre programme. Ceci se fait avec la boucle `for` (lignes 7-14). Une particularité de Lua est que les variables peuvent être multiples. Ici en écrivant `for k,v in pairs(vardata)`,on prend un à un (mais dans n'importe quel ordre) tous les éléments de la table `vardata` (considérée comme une paire) et on affecte le champ `nom` à `k` et la valeur `valeur` à `v`.

Pour chaque paire `k,v`, on imprime le nom `k` et sa valeur `v`. Pour le nom, LualaTEX reçoit des commandes (produites ligne 8) comme `\verb|units_per_em| =` où l'on voit l'utilité de la commande `\verb` puisque le nom actuel contient des « _ ».

Pour les valeurs, nous distinguons en gros deux cas selon leur nature (donnée par la commande `type`) : si c'est un scalaire directement imprimable (chaîne de caractères ou nombre) on passe à LualaTEX la valeur ; sinon — ce peut-être un booléen , une table, etc. — on passe le type (ligne 12). La boucle finie, la fonction Lua `Lister` envoie à LualaTEX le fragment LaTEX `-- -- -- --` (ligne 15) qui va être imprimé tel quel et se termine ; le contrôle passe alors à LualaTEX qui imprime `+ + + +`, à la suite des tirets puisque le dernier appel de `tex.sprint` (ligne 15) ne demandait pas une fin `\par`.

Ces tables étant en fait des arbres, on pense de suite à une procédure récursive en Lua qui permettrait de balayer un arbre en entier. On en trouve effectivement sur le web[59]. Plusieurs points sont alors à programmer pour avoir une sortie propre.

— Les noms des nœuds peuvent être quelconques et la solution d'utiliser `\verb` n'est pas toujours appropriée. Lua permet la substitution de caractères dans une chaîne (fonction `string.gsub`), mais pas facilement le traitement d'expressions régulières.

— Les sous-arbres étant souvent très emboîtés, la gestion des retraits (indentations) peut s'avérer délicate.

— Par ailleurs, certains sous-arbres peuvent avoir un très grand nombre de feuilles (voire de sous-arbres). La moindre fonte aujourd'hui propose plusieurs centaines de glyphes, chacun représenté par au moins un sous-arbre (à titre d'exemple, `EBGaramond-italic` offre plus de 2 000 glyphes). Il faut alors

58. On renvoie le lecteur aux nombreuses introduction à Lua et aux manuels complets cités dans par exemple [33, 32, 34]. On citera quelques spécificités du langage à l'occasion de leur emploi ici.

59. Par exemple à `https://stackoverflow.com/q/9168058` qui sort l'arbre dans le `.log`.

choisir entre être complet (et sortir des centaines de pages) ou se limiter (au moyen encore de paramètres). Ou écrire des procédures (de préférence interactives) qui n'extraient que certaines informations.

— Cette profondeur des sous-arbres amène en général les auteurs de ces procédures récursives à limiter le parcours (en général par un paramètre `depth`). Ici encore, on trouve de telles procédures sur le web [60].

Plutôt que d'utiliser de telles procédures généralistes, on montre ici comment accéder à des propriétés précises, en l'occurrence les axes et leurs triplets d'une fonte variable.

Programme Lua d'extraction des triplets des axes d'une fonte variable

Ce programme (donné ci-après dans l'exemple 30) est simple [61], la seule difficulté étant de connaître la structure de l'arbre de la fonte. Nous avons vu celle-ci en listing 1 page 63 et en figure 23 page 65. Nous vérifions tout d'abord qu'il s'agit d'une fonte variable (ce qui n'est pas le cas si le champ `variabledata` est vide – voir le listing 1 page 63) et sinon nous accédons à la sous-table `axis` dont on sait qu'elle contient les champs `minimum`, `default`, `maximum`, `name`, `tag` (et `flags` dont on ne se sert pas ici).

Exemple 30 – Lister les axes proposés par une fonte variable

code

```
% Nécessite le package `booktabs`
\begin{luacode*}
local function vide(s)
  return s == nil or s == ''
end
function Axesde(fontid)
  local lafonte=luaotfload.fontloader.fonts.hashes.
   identifiers[fontid]
  if not vide(lafonte.resources.variabledata)
  then
    local vardata = lafonte.resources.variabledata["axis
   "]
    for k, v in pairs(vardata) do
      tex.sprint('\\texttt{'..v["tag"]..'}&')
      tex.sprint(v["minimum"]..'&')
      tex.sprint(v["default"]..'&')
      tex.sprint(v["maximum"]..'&')
      tex.sprint(v["name"]..'\\\\')
    end
  else
    tex.sprint("Pas d'axes\\\\")
  end
end
\end{luacode*}

\newcommand{\MontrerAxes}[1]{
```

60. Notamment à <urlstackexchangeàretrouver> qui permet d'imprimer le sous-arbre `axis` d'une fonte variable.

61. Il devrait être amélioré notamment par plus de contrôle sur les valeurs testées.

code (suite)

```
    \font\mafonte="#1"
    Axes de #1\smallskip\par
    \begin{tabular}{lrrrl}
      \toprule
      Axe & min. & défaut & max. & Objet \\\midrule
      \directlua{Axesde(\fontid\mafonte)}
      \bottomrule
    \end{tabular}
}

\MontrerAxes{RobotoFlex-VariableFont.ttf}
\bigskip

\MontrerAxes{cmr12}
```

résultat

Axes de RobotoFlex-VariableFont.ttf

Axe	min.	défaut	max.	Objet
opsz	8.0	14.0	144.0	optical size
wght	100.0	400.0	1000.0	weight
grad	-200.0	0.0	150.0	grade
wdth	25.0	100.0	151.0	width
slnt	-10.0	0.0	0.0	slant
xopq	27.0	96.0	175.0	parametric thick stroke
yopq	25.0	79.0	135.0	parametric thin stroke
xtra	323.0	468.0	603.0	parametric counter width
ytuc	528.0	712.0	760.0	parametric uppercase height
ytlc	416.0	514.0	570.0	parametric lowercase height
ytas	649.0	750.0	854.0	parametric ascender height
ytde	-305.0	-203.0	-98.0	parametric descender depth
ytfi	560.0	738.0	788.0	parametric figure height

Axes de cmr12

Axe	min.	défaut	max.	Objet
Pas d'axes				

La sortie se fait dans un tableau de LualaTeX, ligne par ligne depuis Lua (lignes 6-10). Si ce n'est pas une fonte variable, ce qui est le cas de la fonte `cmr12` de TeX, le résultat est simplement « pas d'axes » (ligne 17).

Accès en ligne aux valeurs des axes d'une fonte variable

Le programme que nous venons de donner (dans l'exemple 30 page précédente) donne les valeurs du triplet d'axes d'une fonte variable, mais celles-ci ne sont pas directement utilisables dans des instructions. Nous proposons, dans l'exemple 31 page ci-contre ci-après, des fonctions Lua dont les résultats sont utilisables dans un programme source LualaTeX.

Ces fonctions rendent les valeurs minimale, maximale ou par défaut d'un axe `monaxe`. Pour ce faire, on déclare une fonction Lua `ValAxes(fontid,monaxe,monchamp)` (ligne 6) qui retourne la valeur choisie `monchamp` du triplet de l'axe `monaxe` choisi pour une fonte spécifiée par son numéro `fontid`. Cette fonction est appelée par l'une des trois fonctions `MinAxe`, `MaxAxe` et `DefAxe` pour les champs spécifiques du triplet. Elle retournent, sous forme décimale (fonction `tonumber`), les valeurs correspondantes au programme hôte par la commande `tex.sprint(tonumber(sortie))`.

Nous utilisons ces valeurs, minimale et maximale, pour l'axe `wght` en spécifiant les options du type `[Weight=\MinAxe{\RobotoFlex}{"wght"}]`.

La procédure `\montest` permet de faire des appels condensés avec plusieurs fontes.

Exemple 31 – Programme de test de fonte, présentant les valeurs minimale, maximale et par défaut de l'axe `wght`

code

```
1  \begin{luacode*}
2  local function vide(s)
3    return s == nil or s == ''
4  end
5
6  function ValAxes(fontid,monaxe,monchamp)
7    local resultat
8    local lafonte = luaotfload.fontloader.fonts.hashes.
      identifiers[fontid]
9    if vide(lafonte.resources.variabledata)
10   then else
11     local vardata = lafonte.resources.variabledata["axis"]
12     for k, v in pairs(vardata) do
13       if v["tag"] == monaxe then resultat = v[monchamp] end
14     end -- for
15   end--pas vide
16   return resultat
17 end
18
19 function MinAxe(fontid, axe)
20   local sortie = ValAxes(fontid, axe, "minimum")
21   tex.sprint(tonumber(sortie))
22 end
23
24 function DefAxe(fontid, axe)
25   local sortie = ValAxes(fontid, axe, "default")
26   tex.sprint(tonumber(sortie))
27 end
28
29 function MaxAxe(fontid, axe)
30   local sortie = ValAxes(fontid, axe, "maximum")
31   tex.sprint(tonumber(sortie))
32 end
33 \end{luacode*}
34
35 \newcommand\MinAxe[2]{%
36   \directlua{MinAxe(\fontid#1,#2)}%
37 }
38 \newcommand\DefAxe[2]{%
```

code (suite)

```
    \directlua{DefAxe(\fontid#1,#2)}%
  }
  \newcommand\MaxAxe[2]{%
    \directlua{MaxAxe(\fontid#1,#2)}%
  }

  \newcommand{\montest}[1]{\font\mafonte{#1}
    \fontspec{#1}[Renderer = OpenType]
    La graisse peut aller du
    {\aff{Weight=\MinAxe{\mafonte}{"wght"}}très maigre}
    au
    {\aff{Weight=\MaxAxe{\mafonte}{"wght"}}très gras}
    avec #1.
  }

  \montest{BritoVariableVF.ttf}\par
  \montest{RobotoFlex-VariableFont.ttf}\par
  \montest{JunicodeVF-Roman.ttf}
```

résultat

La graisse peut aller du très maigre au **très gras** avec BritoVariableVF.ttf.
La graisse peut aller du très maigre au **très gras** avec RobotoFlex-VariableFont.ttf.
La graisse peut aller du très maigre au **très gras** avec JunicodeVF-Roman.ttf.

Glossaire

Références

[1] Hendrik D.L. Vervliet. *Robert Granjon, letter-cutter ; 1513-1590 : an oeuvre catalogue*. New Castle, Delaware, États-Unis d'Amérique : Oak Knoll Press, 2018.

[2] Malou Verlomme. *Fontes variables, la matrice typographique*. 2017. URL : https://www.youtube.com/watch?v=YOwsQN5PKSU.

[3] *American wood type 1828-1900*. Saratoga, Californie, États-Unis d'Amérique : Liber Apertus Press, 1969.

[4] Jacques André. « Histoire technique des fontes numériques ». In : *Histoire de l'écriture typographique – le* XX*e siècle*. T. II. traduit en anglais (par P. Bideault) "The prehistory of digital fonts", *TUGboat*, Vol. 44 (2023), Nº 1, p. 21-57, https://tug.org/TUGboat/tb44-1/tb136andre-prehistory.pdf. Adverbum, 2016, p. 114-143.

[5] Yannis Haralambous. *Fontes et codages*. Paris : O'Reilly, 2004. URL : https://hal.archives-ouvertes.fr/hal-02112931/document.

[6] Thierry Bouche. « Minion MM : installer une famille de fontes ». In : *Cahiers GUTenberg* 26 (1997), p. 45-70. URL : http://www.numdam.org/item/CG_1997___26_45_0/.

[7] Microsoft. *OpenType® Specification Version 1.9.1*. 31 mai 2024. URL : https://learn.microsoft.com/fr-fr/typography/opentype/spec/.

[8] Microsoft. *OpenType Font Variations Common Table Formats*. 9 mai 2024.

[9] John Hudson. *Introducing OpenType Variable Fonts*. 24 sept. 2016. URL : https://medium.com/variable-fonts/https-medium-com-tiro-introducing-opentype-variable-fonts-12ba6cd2369.

[10] Jacques André. « Caractères numériques : introduction ». In : *Cahiers GUTenberg* 26 (1997), p. 5-44. URL : http://cahiers.gutenberg.eu.org/fitem?id=CG_1997___26_5_0.

[11] Roger Hersch. *Visual and Technical Aspects of Type*. Cambridge, GB : Cambridge University Press, 1993.

[12] Jacques André et Victor Ostromoukhov. « Punk : de METAFONT à PostScript ». In : *Cahiers GUTenberg* 4 (1989), p. 23-28. URL : http://www.numdam.org/item/CG_1989___4_23_0.pdf.

[13] PANOSE *Classification Metrics Guide*. (v. aussi PANOSE 2 : https://www.w3.org/Fonts/Panose/pan2.html). 14 fév. 1997. URL : https://monotype.github.io/panose/pan1.htm.

[14] Microsoft. *OpenType Font Variations Overview*. 30 mai 2024. URL : https://learn.microsoft.com/en-us/typography/opentype/spec/otvaroverview.

[15] Chris Lilley. *CSS Fonts Module Level 4*. 14 mai 2025. URL : https://drafts.csswg.org/css-fonts-4/.
[16] Rainer Erich Scheichelbauer. *Creating a variable font*. 21 avr. 2024. URL : https://glyphsapp.com/learn/creating-a-variable-font.
[17] Jacques André et Patrick Bideault. « La fonte du jour : Infini ». In : *La Lettre GUTenberg* 45 (2023). URL : https://www.gutenberg-asso.fr/IMG/pdf/lettre45-ecran.pdf.
[18] Will Robertson. *The fontspec package – Font selection for XeLaTeX and LuaLaTeX*. 11 mai 2024. URL : http://mirrors.ctan.org/macros/unicodetex/latex/fontspec/fontspec.pdf.
[19] Elie Roux *et al. The luaotfload package – LaTeX3 Project*. 3 déc. 2024. URL : https://mirrors.mit.edu/CTAN/macros/luatex/generic/luaotfload/luaotfload-latex.pdf.
[20] Jacques André et Christian Laucou. « Le gras et le livre ». In : *Histoire de l'écriture typographique, le XIXe siècle*. Adverbum/Atelier Perrouseaux, 2013, p. 22, 140-153.
[21] Peter Karow. *Typeface Statistics*. Hamburg : URW, 1993.
[22] Jack Kerouac. *Satori à Paris*. Trad. par Jean Autret. Paris : Gallimard, 1971.
[23] Jacques André. « Lucida a-t-elle un gros œil ? » In : *Lettre GUTenberg* 5 (1995), p. 24-26.
[24] Georges Perec. « Experimental demonstration of the tomatotopic organization in the Soprano (Cantatrix sopranica L.) » In : *Banana Split* 2 (1980). Ce texte est recueilli dans [35].
[25] Emmanuel Souchier. « Quelques remarques sur le sens et la servitude de la typographie ». In : *Cahiers GUTenberg* 46-47 (2006), p. 69-98. URL : http://www.numdam.org/item/CG_2006___46-47_69_0/.
[26] *Tabelle der M- und Qu-Formen, in TW - Typenrepertorium der Wiegendrucke*. URL : https://tw.staatsbibliothek-berlin.de/html/mshapes.xql.
[27] Francis Thibaudeau. *Manuel français de typographie moderne, faisant suite à « La Lettre d'imprimerie »... Cours d'initiation...* Bureau de l'Édition, 1924. URL : https://gallica.bnf.fr/ark:/12148/bpt6k65315750/f81.
[28] Donald E. Knuth. « The concept of a meta-font ». In : *Visible language* (1982). en français : « Le concept de Metafonte », *Communication et langage*, no 55, p. 40-53., p. 3-27.
[29] Gerard Unger *et al.* « Other Replies to Donald E. Knuth' article "The Concept of a Meta-Font" ». In : *Visible Language* 16.4 (1982), p. 353-356. URL : https://journals.uc.edu/index.php/vl/issue/view/360.
[30] Ulrike Fischer. *The fontloader-luaotfload package*. 20 août 2017. URL : https://ctan.org/tex-archive/obsolete/macros/luatex/generic/fontloader-luaotfload.
[31] *LuaTeX Reference Manual*. 10 fév. 2025. URL : https://mirrors.ibiblio.org/CTAN/systems/doc/luatex/luatex.pdf.
[32] Manuel Pégourié-Gonnard. « Un guide pour LuaLaTeX ». In : *Cahiers GUTenberg* 54-55 (2010), p. 13-35. URL : http://cahiers.gutenberg.eu.org/fitem?id=CG_2010___54-55_13_0.
[33] Paul Isambert. « LuaTeX : vue d'ensemble ». In : *Cahiers GUTenberg* 54-55 (2010), p. 3-12. URL : http://cahiers.gutenberg.eu.org/fitem?id=CG_2010___54-55_3_0.
[34] *Lua Doc*. URL : https://www.lua.org/docs.html.
[35] Georges Perec. *Cantatrix sopranica L. et autres écrits scientifiques*. La Librairie du XXe siècle. Paris : Éditions du Seuil, 1991. ISBN : 9782020136501.

ACRONYMES

AAT *Advanced Apple Typography* (typographie avancée d'Apple)

ASCII *American Standard Code for Information Interchange* (code américain normalisé pour l'échange d'information)

CSS *Cascading Style Sheets* (feuilles de style en cascade)

CTAN *Comprehensive TEX Archive Network* (réseau complet d'archives TEX)

ÉNS École Normale Supérieure

NDLR Note de la Rédaction

NFSS *New Font Selection Scheme* (nouveau schéma de sélection de fonte)

OTF *Open Type Format* (format Open Type)

PDF *Portable Document Format* (format de document portable)

RIP *Raster Image Processors* (processeur d'image matricielle)

TFM *TEX font metric* (métrique de fonte TEX)

W3C *World Wide Web Consortium* (un organisme de standardisation chargé de promouvoir la compatibilité des technologies du World Wide Web)

Si vous remarquez des erreurs, merci de nous les signaler par courriel à `secretariat@gutenberg-asso.fr`.

La rédaction rappelle au lecteur que le code source de la présente revue est inclus *dans* le présent PDF. On y accède en fin d'article en cliquant sur les trombones.

Enfin, nous vous conseillons de télécharger la présente *Lettre* pour la lire grâce à un logiciel dont la seule fonction est la consultation de fichiers PDF : cela évitera d'éventuelles erreurs d'affichage.

Avec opiniâtreté, ténacité et entêtement, et par ordre croissant du nombre de lettres de leurs patronymes, celui de leur prénom étant utilisé en cas d'égalité, ont contribué à cette *Lettre* : Daniel Flipo, Jacques André, Bastien Dumont, Denis Bitouzé, Bernard Peyréga et Patrick Bideault.

GUTenberg

Association GUTenberg
15 rue des Halles
BP 74
75001 Paris
France
secretariat@gutenberg-asso.fr

Site Internet : https://gutenberg-asso.fr/
Cahiers : https://cahiers.gutenberg-asso.fr/ et https://www.numdam.org/journals/CG/
Lettre : https://lettre.gutenberg-asso.fr/
Problèmes TEXniques :
liste d'entraide : https://gutenberg-asso.fr/-Listes-de-diffusion-
site de questions et réponses : https://texnique.fr/
foire aux questions : https://faq.gutenberg-asso.fr/

Cette association est la vôtre : faites-nous part de vos idées, de vos envies, de vos préoccupations à l'adresse secretariat@gutenberg-asso.fr.
Adhérents, vous pouvez aussi échanger sur la vie de l'association sur la liste de diffusion adherents@gutenberg-asso.fr.

ADHÉSION À L'ASSOCIATION

GUTenberg étant reconnue d'intérêt général, vous recevrez en temps voulu un justificatif vous permettant de bénéficier d'une réduction fiscale de 66 % du montant de votre cotisation ou de votre don.

- Les adhésions sont à renouveler en début d'année pour l'année civile.
- Les administrations peuvent joindre un bon de commande revêtu de la signature de la personne responsable ; les étudiants doivent joindre un justificatif.

Tarifs[62] 2026

Les membres de GUTenberg peuvent adhérer à l'association internationale, le groupe international d'utilisateurs de TEX (TUG), et recevoir son bulletin *TUGboat* à un tarif préférentiel[63] :

Type d'adhésion	Prix
Personne physique	30 €
Personne physique + adhésion TUG	100 € (= 30 € + 70 €)
Personne physique à tarif réduit	10 €
Personne physique à tarif réduit + adhésion TUG	55 € (= 10 € + 45 €)
Association d'étudiants	65 €
Organisme ou association à but non lucratif	130 €
Personne morale à but lucratif	195 €

Règlements

Les règlements peuvent s'effectuer par :
- paiement en ligne sécurisé[64] : https://gutenberg-asso.fr/Adherer-en-ligne
- bulletin et chèque : https://gutenberg-asso.fr/Adherer-a-l-association

La Lettre GUTenberg
Bulletin irrégulomestriel & apériodique de l'association GUTenberg
Directeur de la publication : P. Bideault
Comité de rédaction : P. Bideault, D. Bitouzé, C. Chevalier & B. Dumont
Adresse de la rédaction : Association GUTenberg
15 rue des Halles – BP 74 – 75001 Paris
ISSN : 2742-6149 (version numérique)

62. Dans ce tableau, une personne physique à tarif réduit est étudiant, demandeur d'emploi ou plus largement toute personne non redevable de l'impôt sur le revenu (sur présentation d'un justificatif) ; un organisme peut être doté ou non de la personnalité morale : laboratoire de recherche public, etc.
63. En tarif normal, 70 € (au lieu de 90 $) ; en tarif réduit, 45 € (au lieu de 65 $).
64. En carte bancaire.