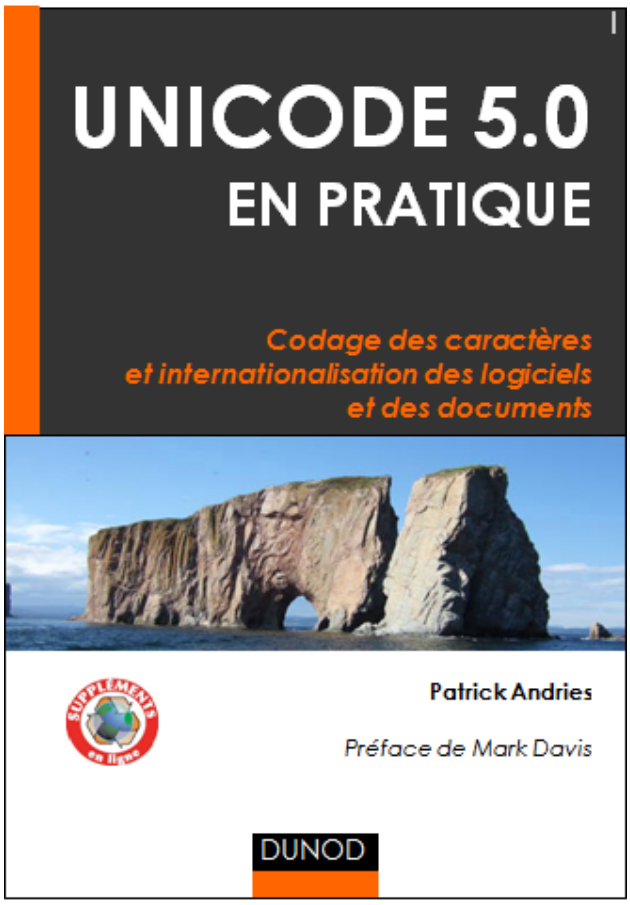
« Je ne connais pas d'autre ressource moderne, ni en anglais ni en français, regroupant une telle gamme de sujets utiles, et les expliquant de façon si claire et si accessible à un large public. J'espère que vous apprécierez cet ouvrage autant que moi. »

Mark Davis, Ph. D.
Président et cofondateur du Consortium Unicode

Commandez en ligne



Unicode et les polices : deux mondes

Patrick Andries
Conseils Hapax http://hapax.qc.ca
Membre du consortium Unicode

Comment passe-t-on du monde des caractères qui est celui d'Unicode à celui des glyphes qui représentent ceux-ci visuellement ? Comment sont stockés ces glyphes ? Qu'est-ce qu'une police de « caractères » comme on dit encore trop souvent ? Voilà les questions auxquelles cet article tente de répondre tout en introduisant une des dernières techniques en matière de polices : OpenType, le successeur des polices TrueType. L'article décrit également brièvement le moteur de rendu qui interprète les polices et équipe toutes les versions récentes du système d'exploitation Windows de Microsoft : Uniscribe.

1. Unicode

1.1. Des caractères et non des glyphes

Comme nous l'avons vu dans l'Introduction à Unicode et à l'ISO/CEI 10646 de ces actes, cette norme ne se préoccupe que des caractères et non de leurs aspects, leurs glyphes. Unicode préconise également de stocker les caractères en « ordre logique », c'est-à-dire dans l'ordre phonétique et le plus souvent dans l'ordre de saisie. Enfin, Unicode présume l'existence d'un moteur de rendu qui pourra, le cas échéant, réordonner les caractères en ordre visuel et pourra substituer aux caractères des glyphes adéquats tout en les plaçant de manière précise.

1.2. Pourquoi ce modèle ?

On pourrait se demander pourquoi Unicode a choisi ce modèle, appelé modèle caractère-glyphe, alors qu'*a priori* on pourrait penser qu'il aurait pu être tellement plus simple de ne coder et de ne stocker que des glyphes.

Au-delà du simple fait que la codification de variantes de caractères est une tâche herculéenne, car on crée chaque jour de nouvelles polices, il est nettement plus judicieux de se concentrer sur les caractères sans se soucier de leur apparence pour toute une série de traitements qui vont du correcteur grammatical et de la traduction automatique à l'indexation en passant par la recherche et remplacement de texte, le copier-coller, l'exportation de texte, le changement de police ou le changement de style de paragraphe ou de caractère. En codant des caractères et non des glyphes, on s'assure que ces opérations fonctionneront quelle que soit l'apparence finale de la chaîne de caractères à traiter.

2. Fonctionnement d'un moteur de rendu

Nous décrivons ci-dessous le fonctionnement quelque peu simplifié et idéalisé d'un moteur de rendu. Il ne s'agit pas ici de décrire un moteur de rendu particulier, mais de mieux comprendre les différentes étapes reliées à la composition et la part que jouent Unicode et les polices à chacune de ces étapes.

2.1. Découpe en passages de même style

Une des premières tâches d'un moteur de rendu est de découper le texte stocké en mémoire en *passages* (ou parle aussi parfois de *segments*) de même style, même police, même système

d'écriture et enfin de même directionalité[1]. La figure 1 ci-dessous illustre une telle découpe. Remarquez que les caractères arabes sont représentés en ordre logique et non contextualisés.

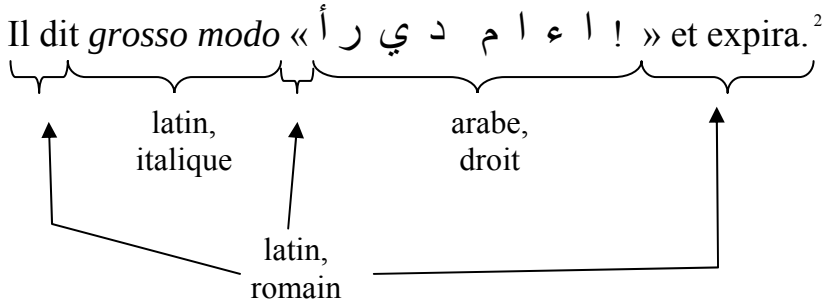


Figure 1. – Découpe de phrase bidirectionnelle

2.2. Coupure de lignes

Une coupure de lignes de base est relativement simple à mettre en œuvre pour l'écriture latine : il est permis de couper une ligne à l'espace, au trait d'union (réel ou virtuel) ou à la tabulation. Pour d'autres langues, telles que le thaï ou le lao, les mots se suivent sans qu'une espace ne les sépare. La coupure des lignes dans ces langues doit s'effectuer à la frontière de syllabes. Dans ces langues, une coupure de ligne de base acceptable nécessite un algorithme de syllabisation similaire à ce qu'on utilise en français pour effectuer la division de mots en fin de ligne. Notons qu'Unicode fournit des recommandations et propriétés de caractère qui permettent de mettre en œuvre un algorithme de coupure de lignes[3] de base.

Il est important de noter que la coupure de lignes est un processus itératif qui implique souvent, plusieurs passages aux glyphes (voir 2.3. Réordonnancement et 2.4. Sélection des glyphes, les deux étapes suivantes). En effet, pour déterminer une position de coupure permise, on utilise les propriétés des caractères Unicode ; mais pour savoir si la chaîne ainsi découpée est de bonne taille et peut rentrer dans l'espace réservée pour la ligne, il faut en connaître la taille en termes de glyphes, éventuellement crénés, ligaturés ou les deux. Si la chaîne ne rentre pas, on choisira un autre point de coupure plus proche, une nouvelle fois en se référant aux propriétés des caractères, pour s'enquérir ensuite de la taille des glyphes correspondants à cette chaîne de caractères raccourcie.

Le processus décrit ci-dessus est volontairement épuré. Ainsi, TeX n'effectue-t-il pas ce processus sur une base de lignes mais il détermine à chaque essai les meilleures positions de coupure de lignes au niveau du paragraphe tout entier.

---

[1] C'est-à-dire droite-à-gauche (l'arabe, l'hébreu, le syriaque, le n'ko, etc.) ou gauche-à-droite (le latin, le grec, le cyrillique, etc.).

[2] Ce que l'on peut représenter également sous la forme suivante, si on remplace les lettres arabes par des capitales : Il dit *grosso modo* « DE L'EAU ! » et expira. On voit donc que le mot arabe est stocké en mémoire en ordre logique, c'est-à-dire de gauche à droite alors que l'écriture arabe s'affiche de droite à gauche.

[3] Voir <http://hapax.qc.ca/pdf/Chapitre-14.pdf>, section 14.2, *Commandes de mise en page*.

2.3. Réordonnancement

Après avoir découpé le texte en mémoire, le moteur de rendu réordonne habituellement les passages pour les faire passer de l'ordre logique à l'ordre visuel et ainsi faciliter le passage à l'affichage des glyphes. Cette remise en ordre implique alors l'application de l'algorithme bidirectionnel[4].

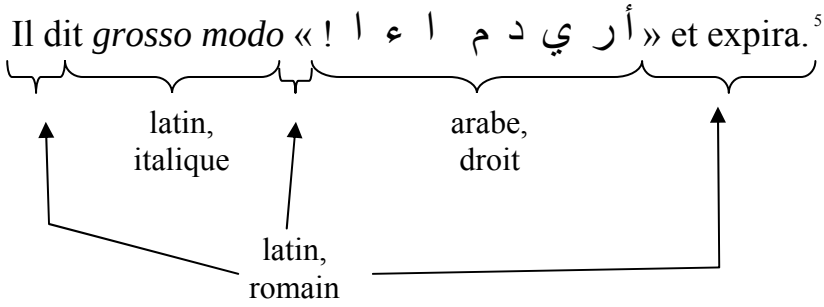


Figure 2. – Passage en ordre visuel

Il est important de noter que ce réordonnancement doit avoir lieu après la coupure ligne. En effet, si on coupe la ligne après le premier mot arabe il faut que celui-ci reste sur la première ligne après le passage en ordre visuel.

2.4. Sélection des glyphes

L'étape suivante consiste à passer de l'espace de caractère qu'est Unicode à la représentation de ces caractères en termes de glyphes, les composants des polices. Le passage des caractères aux glyphes est la première étape obligatoire et systématique. Par la suite, selon le type de

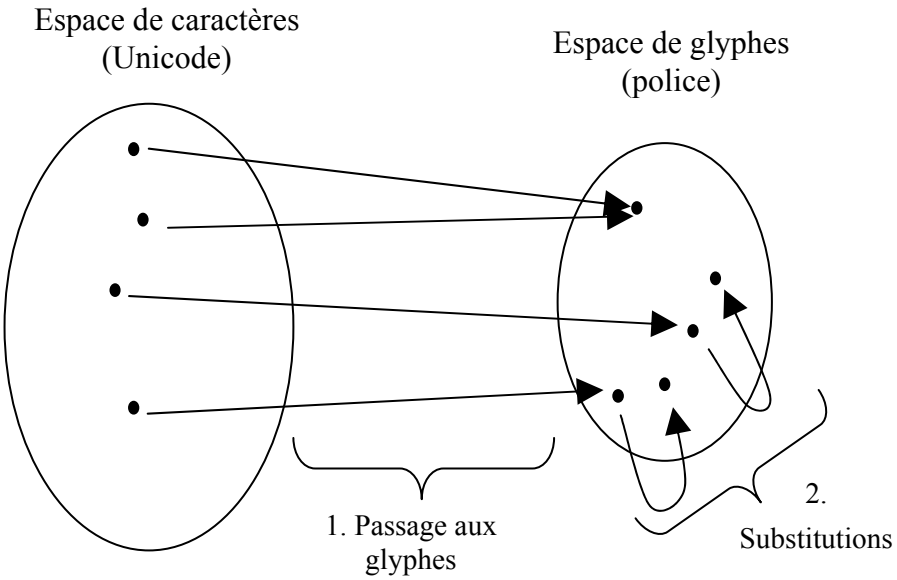


Figure 3. – Deux espaces

---

[4] Voir <http://hapax.qc.ca/pdf/Chapitre-3.pdf> section 3.12, *Comportement bidirectionnel*, et section 9.2, *Arabe* dans <http://hapax.qc.ca/pdf/Chapitre-9.pdf>. Unicode ne prescrit pas la mise en œuvre de cet algorithme bidirectionnel, mais bien le résultat que la mise en œuvre doit produire.

[5] En remplaçant les lettres arabes par des capitales : Il dit *grosso modo* « ! UAE'L ED » et expira.

police utilisée, la forme précise du glyphe à afficher s'effectuera par un nombre de substitutions optionnelles. Nous reviendrons en détail sur ce processus par la suite.

2.5. Justification

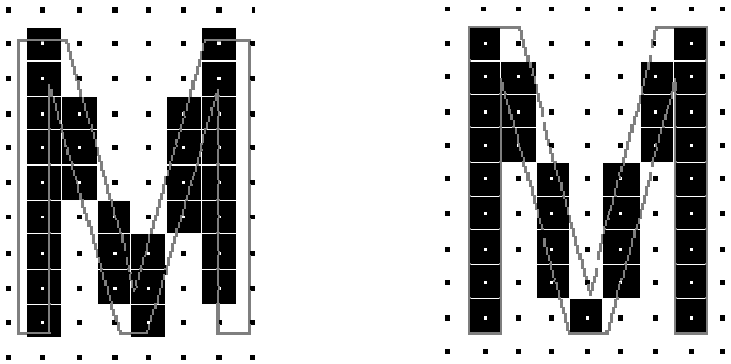
Les suites de glyphes sont ajustées pour respecter les règles de justification en vigueur. Pour ce faire, l'algorithme de justification changera l'espace intermot ou, si la police contient ces informations, utilisera une variante d'œil plus étroit ou plus large pour certains glyphes.

Exemple de texte dit en drapeau, aligné à gauche ou encore fer à gauche. Exemple de texte dit en drapeau, aligné à gauche ou encore fer à gauche. Exemple de texte dit en drapeau, aligné à gauche ou encore fer à gauche.

Exemple de texte dit justifié, en bloc ou en encore en paragraphe réparti, avec au besoin la coupure de mots. Exemple de texte dit justifié, en bloc ou en encore en paragraphe réparti, avec au besoin la coupure de mots.

2.6. Tramage et optimisation du rendu

Les glyphes dans la majorité des polices modernes sont définis en terme d'équations qui définissent les contours de ces glyphes[6]. Or, en fin de compte, ces contours devront apparaître sous la forme de pixels noircis ou non. Si on n'avait à rendre une police qu'à une seule force[7] et à une seule résolution d'écran ou d'imprimante, il n'y aurait nul besoin de définir les glyphes sous la forme de contours, une image en mode point suffirait. L'utilisation de contours permet à la police de s'adapter à différentes résolutions et forces de corps.



Tramage non optimisé Ajusté à la grille

Figure 4. – Optimisation du rendu

Un trameur se charge du passage des contours vectoriels à une trame de carrés équidistants disposés en une grille. La tâche du trameur est donc de noircir les bons carrés pour épouser au

[6] D'autres sont définis directement en termes de pixels, on appelle ces polices des polices « bitmap » ou en mode « point ».

[7] Taille du caractère en points : « un corps de force 6 », « du 6 » de façon elliptique.

mieux le contour de la lettre à rendre. Cette tâche n'est toutefois pas aisée, car il suffit parfois d'une petite différence de contour ou une troncature correcte, mais malencontreuse, pour qu'un mauvais carré soit noirci. Ainsi, même si le contour est parfait l'apparence tramée peut être illisible. Ce problème est plus fréquent, ou du moins plus visible, aux petites résolutions, car les carrés sont plus gros. Pour y pallier, on a prévu différentes méthodes qui permettent d'optimiser le rendu et d'ajuster le résultat du tramage. Dans le cas des polices TrueType et de leurs successeurs, les polices OpenType, cette optimisation se fait à l'aide d'instructions supplémentaires. Celles-ci permettent de modifier la forme des contours pour l'adapter à la grille de tramage donnée, c'est-à-dire pour une résolution et une force données. Ce nuancement des contours permet de guider l'algorithme de tramage et l'aider à noircir les bons carrés. L'optimisation précise d'une police TrueType pour en assurer un rendu de qualité à l'écran est un véritable travail de bénédictin.

3. Les polices de glyphes[8]

3.1. Historique

En 1985, Adobe révolutionne la production et la publication de documents électroniques sur microordinateurs en introduisant le langage de description de page PostScript. Adobe frappe un autre grand coup quand il signe un accord avec la société Linotype, chef de fil américain dans le domaine des polices, pour numériser et convertir son énorme bibliothèque de polices en format PostScript. Grâce à l'utilisation commune de Postscript par les tireuses d'épreuves (l'imprimante laser) et au système imageur (l'imprimante de qualité), on pouvait garantir la cohérence du résultat. Bien que la définition de l'imprimante laser n'était que de 300 points par pouce (ppp) et celle du système imageur était de 1270 ppp, la géométrie de la page était identique. On pouvait donc être assuré que ce qu'on imprimait sur son imprimante laser serait reproduit par l'imprimante à haute définition. C'était une énorme innovation.

Ce succès ne fut pas sans inquiéter les deux meneurs de file de la micro-informatique de l'époque – Apple et Microsoft – qui se mirent à la recherche d'un autre système de rendu de polices pour écran. De ces efforts naquit TrueType qu'Apple sortit en 1990 et Microsoft un an plus tard.

Pendant toutes les années 90, les systèmes d'exploitation de Microsoft ne proposaient que des polices TrueType. Cependant Adobe et Microsoft se réunirent pour mettre au point un format de police unifié. Le résultat, désormais connu sous le nom d'OpenType, est un format multi-plateforme qui peut contenir à la fois des contours TrueType et PostScript ainsi que de nouvelles fonctionnalités de typographie de pointe. Avec Windows 2000 et Windows XP, Microsoft offre un trameur PostScript intégré, ceci signifie que ces deux versions de Windows prennent en charge de manière native OpenType (que les contours des glyphes soient décrits sous forme PostScript ou TrueType). OpenType semble promis à un bel avenir et sa prise en charge par les applications (car comme on le verra une partie de la prise en charge est laissée aux applications) se répand de plus en plus, c'est pourquoi dans le reste de cet article nous nous concentrerons sur ce format de police.

3.2. Une police, un ensemble de tables

Mais, sous le capot, quel est le format que prend une police ? Il s'agit d'un ensemble de tables

[8] Nous avons délibérément évité le terme traditionnel de « police de caractères » car ces fontes ne contiennent justement pas des caractères, mais des glyphes – les œils en typographie traditionnelle française – associés à des caractères ou des suites de caractères.

où chaque table a une fonction différente. Certaines tables sont standard et parfois communes à TrueType (l'ancien format compris par toutes les applications Windows) d'autres introduits par OpenType. Notons enfin que certaines tables peuvent être définies par un fondeur sans accord préalable ou permission avec des tiers, on les dit privées.

Nom de la table[9]	Prise en charge
cmap	TrueType et OpenType
glyf	TrueType et OpenType
hmtx	TrueType et OpenType
CCF	PostScript et OpenType
GSUB	OpenType
GPOS	OpenType
…	…

Tableau 1. – Tables OpenType

Rappelons aussi que les polices sont passives : elles ne font par elles-mêmes rien. Elles définissent des informations. Un programme les interprète, il s'agit du moteur de rendu.

3.2.1 La table « cmap »

Cette table obligatoire permet de passer de l'espace des caractères (par exemple Unicode) à l'espace des glyphes. Notez qu'à un même glyphe peuvent correspondre plusieurs caractères. C'est typiquement le cas de ces trois caractères dans les polices qui les connaissent :

U+002D TRAIT D'UNION-SIGNE MOINS
U+00AD TRAIT D'UNION VIRTUEL[10]
U+2011 TRAIT D'UNION INSÉCABLE

Ces caractères ont, en règle générale, le même œil[11] dans l'écriture latine et se réfèreront donc habituellement au même glyphe dans une police; ils ne se distinguent que par leur propriété (de sécabilité) et c'est pourquoi ces trois caractères sont codés séparément dans Unicode. Il existe, comme on le verra plus en détails plus tard, également des cas où un glyphe ne correspond à aucun caractère Unicode.

3.2.2. La table « glyf »

La seconde table obligatoire, en l'absence d'une table CCF (cf. 3.2.3), héritée de TrueType est la table `glyf`. Il s'agit souvent de la table la plus volumineuse, elle contient les contours de glyphes. Chaque entrée de la table correspond à là définition d'un glyphe. Le tableau 2 ci-dessous illustre schématiquement le contenu de la table `glyf`.

[9] Notons que les noms des tables en TrueType sont en minuscules alors que les noms de tables OpenType sont en majuscules.

[10] Le trait d'union virtuel n'a nécessairement d'œil (de dessin), en effet depuis Unicode 4.0 ce caractère est de catégorie générale `Cf` (« Autre caractère de formatage »).

[11] L'œil est le dessin du glyphe, on dit au pluriel des œils.

N° de glyphe *x*	Rectangle englobant (coins opposés)
	Contour 1
	Contour 2
	Instructions de nuancement
N° de glyphe *x+1*	Rectangle englobant (coins opposés)
	Contour 1
	Contour 2
	Contour 3
	Contour 4
	Instructions de nuancement
N° de glyphe *x+2*	Rectangle englobant (coins opposés)
	Contour 1
	Contour 2
	Instructions de nuancement
	…

Tableau 2. – La table « glyf »

Chaque entrée commence par la définition du rectangle englobant le glyphe, il s'agit d'une paire de coordonnées qui correspondent aux coins opposés du plus petit rectangle qui englobe le glyphe. Vient ensuite une série de contours fermés sous la forme d'une série de points qui définissent la forme du glyphe. Parmi les autres données qu'on retrouve pour chaque glyphe, il faut signaler les instructions de nuancement[12] – il s'agit d'un véritable code assembleur – qui permettent d'optimiser le rendu à faible définition en modifiant légèrement les contours à une force et une définition données.

Une entrée simplifiée pour une barre oblique « / » peut donc se représenter en XML, comme ci-dessous. Ces données structurées se trouvent codées dans la police sous forme binaire[13].

```
<Glyphe xMin="2" yMin="-30" xMax="574" yMax="1423">
      <contour>
                  <pt x="85" y="-30" sur="1"/>
                  <pt x="2" y="-30" sur="1"/>
                  <pt x="495" y="1423" sur="1"/>
                  <pt x="574" y="1423" sur="1"/>
      </contour>
```

[12] On dit parfois « hinting » même en français bien que, *stricto sensu*, ce terme ne s'applique qu'aux polices PostScript.

[13] Cf. page 697 de Yannis Haralambous, ouvrage cité dans la bibliographie.

```
      <instructions>
            <!-- code compilé  se trouve ici -->
      </instructions>
</Glyphe>
```

La valeur de l'attribut `sur` vaut « 1 » quand le point définit un point de contour, « 0 » quand il définit un de commande (ou de contrôle)[14]. Ce contour correspond au dessin illustré à la figure 5 ci-dessous.

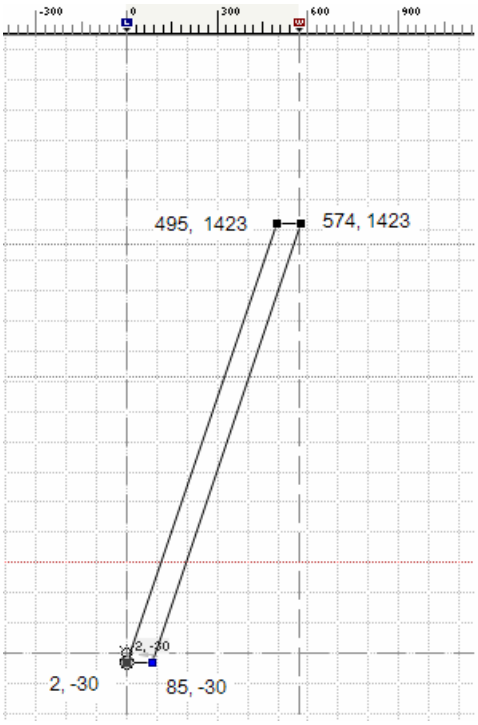


Figure 5. – Contour d'un glyphe

3.2.3. La table « CCF »

Si la table `glyf` contient des conteurs TrueTrype, La table CCF contient des contours PostScript. En effet, les contours peuvent être exprimés indifféremment en format TrueType ou en PostScript. Chaque mention de la table `glyf` dans ce texte désigne la table `glyf` ou la table CFF. Le seul endroit où la technologie importe est dans la section 2.6 : le nuancement est explicitement la responsabilité du créateur de la police pour TrueType, alors qu'en PostScript, le créateur fournit des contraintes que le trameur doit respecter (le trameur doit faire plus de travail).

3.2.3. La table « hmtx »

Cette table obligatoire est l'une des plus simples. Il s'agit de la table des « métriques horizontales » de la police. Elle contient pour chaque glyphe, sa chasse et son approche gauche. La distance horizontale entre la position actuelle (avant l'insertion du glyphe) et l'extrême gauche du glyphe est son approche gauche. La distance horizontale parcourue après l'insertion du glyphe est sa chasse. Ainsi, un glyphe en surplomb aura-t-il un chasse nulle et probablement une approche gauche négative s'il doit suivre le glyphe surimprimé de la sorte alors qu'une espace aura une grande chasse bien que ce glyphe n'ait pas d'œil.

[14] Voir l'article de Youssef Aït Ouguengay dans ces mêmes actes.

3.2.3. La table « GSUB »

Une des nouvelles tables introduites par OpenType est la table GSUB. Cette table définit la substitution d'un ou plusieurs glyphes par un ou plusieurs autres glyphes. Plusieurs types de substitutions sont possibles :

Simple	a → A
Décomposition	ffi → f f i
Sélection de variante	& → &, &, &, &, &, &
Ligature	
	f f i → ffi
	ل ا → لا
Contextuelle	*dans* → *dans*

Ainsi, on trouve des substitutions qui permettent de remplacer un glyphe par un autre, de décomposer une ligature en ses glyphes constituants, de sélectionner une variante, de former des ligatures à partir de deux glyphes (comme la ligature arabe *lam alif*) ou encore de modifier un glyphe selon son contexte. On peut se demander ce qui distingue la sélection de variante de la substitution simple. Après tout, on substitue un glyphe par un autre dans les deux cas. La différence réside dans le fait que pour la sélection de variante le logiciel permet à l'utilisateur de choisir la variante qu'il préfère alors que pour la substitution simple aucune interaction avec l'utilisateur n'est nécessaire. Dans le cas de *dans*, les lettres sont par défaut pourvues d'un queue et dépourvues d'une attaque (*d a n s*). Grâce à une substitution contextuelle, la terminaison du s final est supprimée et le d initial prend un petit trait appelé attaque.

On voit qu'à l'aide de ces substitutions il est tout à fait possible de définir des glyphes qui ne correspondent directement à aucun caractère Unicode particulier, mais bien à un suite de caractères Unicode.

3.2.4. La table « GPOS »

La table GPOS définit le déplacement de glyphes par rapport à leur position habituelle, à d'autres glyphes (par exemple une paire de glyphes pour mettre en œuvre le crénage) ou pour placer des diacritiques sur des lettres de base ou encore sur d'autres diacritiques.

Le déplacement d'un seul glyphe permet de mettre ce glyphe en indice, en exposant ou de mieux centrer des parenthèses quand on compose en petites capitales (cf. ci-dessous). Remarquez comme les parenthèses du dernier (BACH) en petites capitales, mieux centrées que son voisin, laissent une bien meilleure impression.

(BACH) (BACH) (BACH)

Grâce au déplacement de paires de glyphes, on peut non seulement mettre en œuvre un crénage traditionnel horizontal (cf. ci-dessous), mais également déplacer chacun des deux glyphes séparément dans toutes les directions.

AVION → AVION

La table GPOS permet de préciser pour chaque diacritique un point de fixation (d'une certaine classe) qui viendra s'attacher à un point de fixation de même nom défini sur le glyphe de base. Chaque glyphe de base possède autant de points de fixation qu'il n'y a de classe de diacritiques. Ainsi si l'on prend un O, comme dans l'illustration ci-dessous, on définit un ensemble de points de fixation dont un que l'on nommera haut_centre. Tous les diacritiques qui peuvent s'attacher à ce point définissent un point de fixation de classe haut_centre.

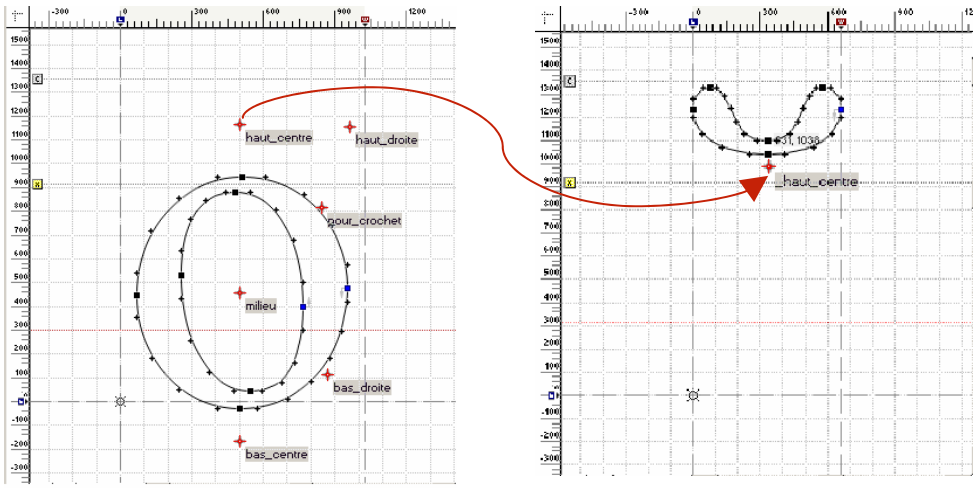


Figure 6. – Points de fixation

Dans l'exemple ci-dessus, nous avons illustré le cas d'un signe diacritique « brève » que l'on veut adjoindre à un O. Un moteur de rendu OpenType placera le brève correctement au-dessus du O en utilisant les points de fixation.

Plusieurs langues (le vietnamien) et notations (comme l'alphabet phonétique international) utilisent de multiples signes diacritiques. La table GPOS permet de définir ce genre de positionnement relatif.

ặ Ẵ Ế

Figure 7. – Exemple de lettres vietnamiennes à accents multiples

3.2.5. GSUB ou GPOS ?

En théorie, il est tout à fait possible de se passer de GPOS pour placer ou déplacer un diacritique. En effet, on peut obtenir le même résultat en utilisant des substitutions de plusieurs glyphes (la lettre de base et un ou plusieurs diacritiques) vers un nouveau glyphe précomposé.

Chaque solution comporte un avantage : les substitutions sont plus rapides car aucun calcul de

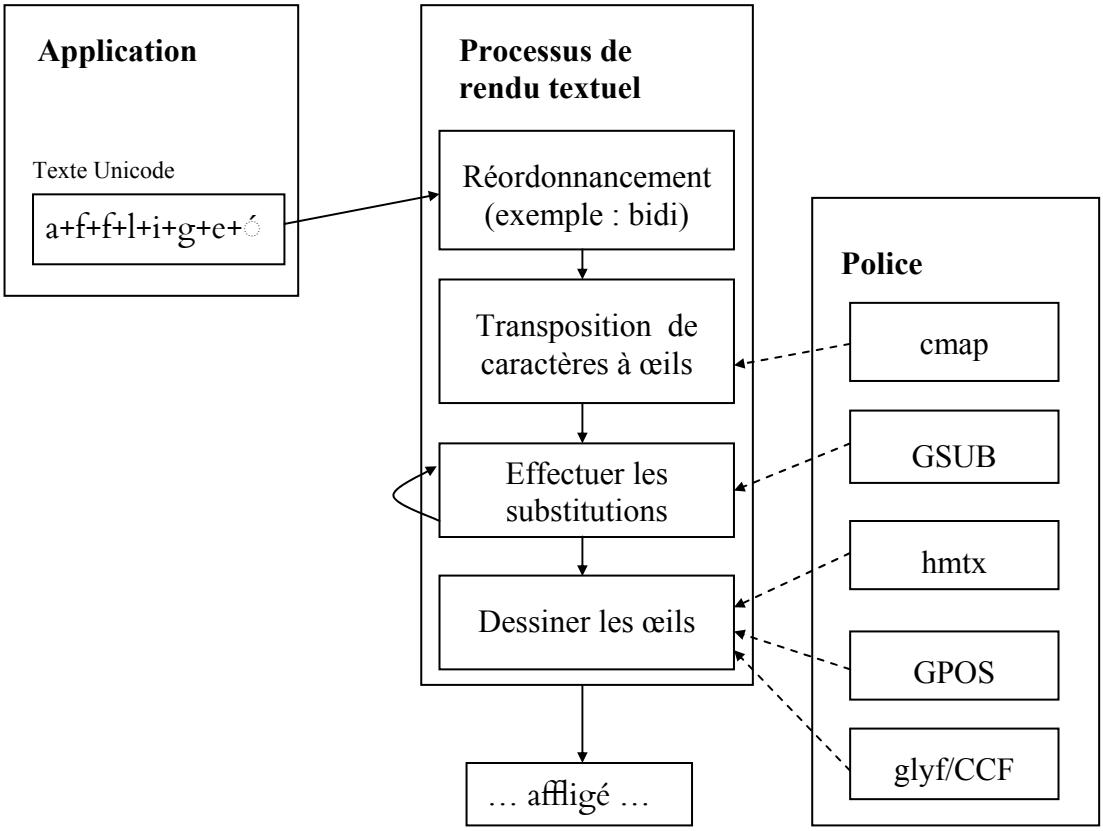


Figure 8. – Étapes internes du processus de rendu

positionnement n'est nécessaire. Toutefois, pour des écritures fortement diacritiquées comme l'arabe ou certaines variantes latines, l'utilisation de GPOS est nettement plus économique en terme de développement de la police et de taille de la police car on définit dans ce cas nettement moins de glyphes. En principe, toutes les lettres de base peuvent être ornées d'un ou plusieurs diacritiques génériques [U+0300..U+036F], on conçoit facilement que la création de glyphes précomposés pour toutes ces combinaisons est une tâche impossible à réaliser avec des substitutions.

4. Processus de rendu

Maintenant que nous avons introduit les principales tables OpenType, intégrons-les au processus de rendu. La figure 8 ci-après illustre un modèle de rendu textuel typique. Dans cet exemple, le texte-source Unicode est en forme normalisée KD[15], le résultat affiché pour sa part utilise la ligature « ffl » et un é précomposé. Le processus de rendu commence d'abord par exécuter l'algorithme bidirectionnel d'Unicode[16] dans l'espace des caractères. Il effectue ensuite le passage des caractères aux œils grâce à la `cmap`. Pour chaque passage directionnel[17], il remplace les glyphes selon les règles définies par la table GSUB et, le cas échéant, des paramètres passés par l'application[18]. Ces remplacements se font de manière itérative : une ligature issue d'une substitution peut participer à une substitution ultérieure.

[15] Voir l'article du prof. Zenkouar dans ces mêmes actes.
[16] Unicode ne prescrit pas la mise en œuvre de cet algorithme, mais le résultat de sa mise en œuvre.
[17] C'est-à-dire dans une suite de caractères de même directionalité (droite-à-gauche, gauche-à-droite).
[18] Dans le cas d'OpenType, les substitutions sont réalisées par le client et non le système d'exploitation. Le client a donc toute latitude pour sélectionner les substitutions désirées. En revanche, dans le cas du système ATSUI sur Macintosh, tout le processus est effectué de manière transparente par le système d'exploitation.

Ensuite, une fois l'ordre et l'identité des œils déterminés, on calcule leur position relative précise à l'aide de la table `hmtx` et on raffine celle-ci grâce à la table GPOS. Enfin, le système d'exploitation utilise les tables `glyf` ou CCF et certaines autres tables pour dessiner les œils.

La prise en charge des fonctionnalités originales d'OpenType, comme les substitutions, nécessite une modification des applications. C'est ce qui explique qu'il ne suffit pas de créer une police OpenType pour que MS Word affiche les ligatures qu'on aura pu ajouter à ces polices. En fait, aucun progiciel de Microsoft ne prend actuellement en charge les ligatures latines ! Microsoft n'utilise la formation de ligatures GSUB que dans les langues non latines où elles s'avèrent essentielles. Toutefois d'autres produits, comme InDesign d'Adobe, permettent la formation de ligatures latines OpenType pour les écritures latines.

5. Un moteur de rendu Uniscribe

Uniscribe est une DLL[19] qui fait partie du système d'exploitation Windows. Il s'agit d'un moteur de rendu conçu pour prendre en charge les fonctionnalités des polices OpenType sur Windows. Uniscribe est une partie intégrale de Windows 2000 et Windows XP. La sortie d'Uniscribe s'est accompagnée de la sortie de nouvelles versions d'Internet Explorer (la version 5) et d'applications faisant partie d'Office 2000 conçues pour tirer parti des fonctionnalités offertes par Uniscribe.

Uniscribe est en réalité un ensemble de DLL, chacune de celles-ci, appelée moteur de façonnage par Microsoft, est chargée d'une écriture[20]. Chaque moteur de façonnage applique un sous-ensemble de fonctionnalités OpenType dans un ordre lié à l'écriture qu'il prend en charge. La première version d'Uniscribe ne prenait en charge que trois écritures : la dévanâgarî, le tamoul et l'arabe. Depuis lors, plusieurs versions améliorées d'Uniscribe sont sorties avec Windows XP et Office XP, elles ont notamment introduit la prise en charge du thaï, du syriaque et de la composition des accents latins. Toutefois, de nombreuses écritures ne sont toujours pas prises en charge correctement, particulièrement les ligatures latines. Pour ce qui a trait aux ligatures touarègues, celles-ci ne seront certainement pas prises en charge avant la sortie de la prochaine version de Windows, au nom de code « Longhorn ».

La figure 9 illustre les principales opérations quand une application prévue pour travailler avec Uniscribe désire afficher une chaîne de caractères arabes (avec potentiellement du texte latin). L'application divise d'abord ① le texte en passages de même style (par exemple de même police, couleur et force). L'application appelle Uniscribe ② pour diviser ces passages en « items », c'est-à-dire des fragments de même écriture et directionalité. Cette découpe permet de découvrir le moteur de façonnage qui sera chargé de rendre chaque segment.

[19] DLL est l'abréviation anglaise de « bibliothèque à lien dynamique ». Une DLL est un ensemble de petits programmes qui peuvent être appelés par un programme plus important pour effectuer une tâche ou une série de tâches particulières. Le principal avantage des DLL tient au fait qu'elles ne sont chargées en mémoire que lorsqu'on les appelle, libérant ainsi la mémoire pour d'autres processus.

[20] En général, car il existe aussi des moteurs de façonnage responsables de l'affichage des chiffres.

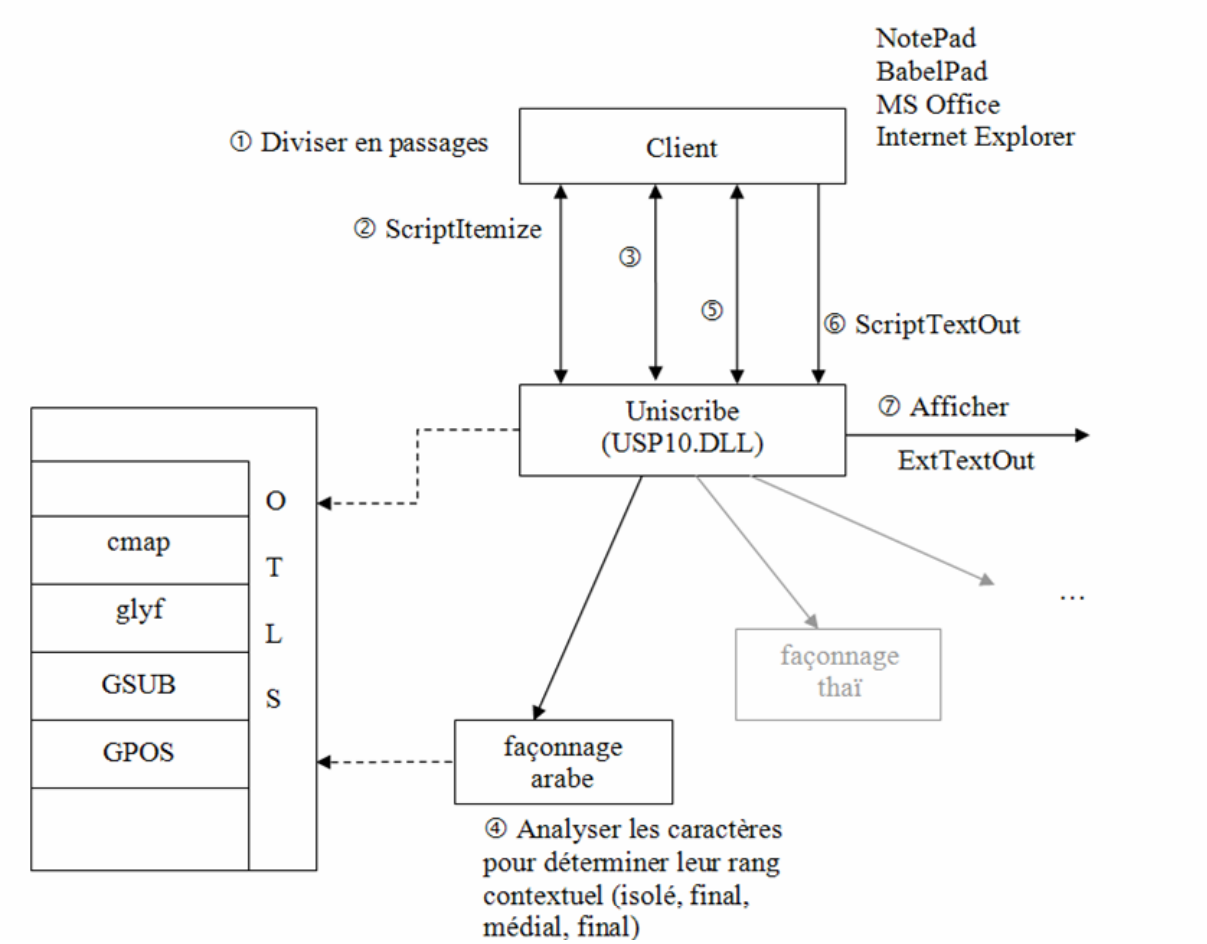


Figure 9. – Uniscribe

En ③, l'application appelle la fonction `ScriptShape` d'Uniscribe, pour chaque passage, en lui fournissant les caractères à afficher et le résultat de `ScriptItemize`. La fonction `ScriptShape` rend un vecteur de glyphes qui correspond aux caractères fournis en entrée. Pour déterminer ce vecteur de glyphes, le moteur de façonnage, arabe en l'occurrence, doit analyser les caractères fournis en entrée pour déterminer leur rang contextuel ④, c'est-à-dire si un caractère donné est un caractère initial, médial, terminal ou isolé. Ensuite, avec cette connaissance, le moteur arabe effectue les substitutions (GSUB) nécessaires. Le moteur de façonnage peut également effectuer d'autres substitutions qui n'impliquent pas le rang contextuel des caractères. Nous reviendrons plus en détails sur ce sujet. Notons que, pour accéder aux polices, Uniscribe et les applications peuvent profiter de la bibliothèque d'OTLS qui facilite l'accès aux différentes tables OpenType.

En ⑤, l'application appelle la méthode ScriptPlace avec le résultat de ④ pour calculer la chasse[21] et les coordonnées des glyphes. Enfin, l'application appelle `ScriptTextOut` ⑥ pour afficher les glyphes d'un passage en fournissant les glyphes, leurs approches et coordonnées. `ScriptTextOut` appelle la fonction système d'exploitation ⑦ `ExtTextOut` de manière appropriée et le passage arabe s'affiche. Notons enfin que l'application peut appeler d'autres fonctions Uniscribe pendant ce processus comme `ScriptGetCMap` qui rend les numéros de glyphes associés à une chaîne de caractères pour une police donnée. Grâce à cette fonction on peut, par exemple, vérifier si une police permet d'afficher une chaîne de caractères, en

[21] La largeur des glyphes.

d'autres mots qu'aucun caractère n'a pas de glyphe. Si certains caractères ne sont pas pris en charge l'application pourra par exemple décider de choisir une autre police et réessayer la même opération avec cette nouvelle police.

6. Fonctionnalités et règles OpenType

6.1. Hiérarchie d'accès

On appelle « règle » une substitution (définie dans la table GSUB) ou un positionnement (défini dans la table GPOS). Le moteur de rendu filtre les substitutions et positionnements définis dans une police pour n'appliquer que ceux qui correspondent à l'écriture et à la langue utilisées. Même les fonctionnalités qui ne sont pas par nature liées à une écriture ou une langue sont structurées selon la hiérarchie suivante :

Écriture > Langue > Fonctionnalité[22] > Groupe[23] > Règle

Une langue est toujours associée à une écriture, elle précise une convention graphique particulière à une écriture pour la langue en question. Beaucoup de langues écrites à l'aide d'une même écriture partagent les mêmes conventions graphiques, c'est pourquoi OpenType définit également une « langue » commune ou implicite représentée par la valeur `dflt`. En règle générale, les fonctionnalités rangées sous `dflt` suffisent à représenter la quasi-totalité des langues qui utilisent cette écriture. Le développeur de police pourra modifier ce comportement typographique implicite pour une écriture en précisant des langues pour lesquelles un traitement différent doit être effectué. C'est à l'application-client de fournir la langue en vigueur pour un passage donné.

Dans l'exemple ci-dessous, on a ajouté une hiérarchie « turque » à l'écriture latine pour distinguer le type de ligatures permises en turc et exclure ainsi certaines ligatures permises pour d'autres langues utilisant l'écriture latine. On se rappellera que le turc distingue le i et le ı (i sans point), on ne peut donc escamoter le point du i comme on le fait habituellement en latin pour les ligatures fi et ffi.

```
Latin <latn>
  Implicite <dflt>
    Ligatures habituelles <liga> (fonctionnalité)
      Toutes les ligatures-f (groupe GSUB)
         f f i -> ffi
         f f l -> ffl
         f f -> ff
         f i -> fi
         f l -> fl
  Turc <TRK>
    Ligatures habituelles <liga> (fonctionnalité)
      Les ligatures ff et fl (groupe GSUB)
         f f l -> ffl
         f f -> ff
         f l -> fl
```

Le même principe peut être utilisé pour distinguer les lettres serbes italiques des lettres cyrilliques italiques habituelles ou les formes précises des caractères CJC unifiés dans Unicode. Le composant linguistique permet en quelque sorte de désunifier les différentes formes linguistiques d'un même caractère qu'Unicode unifie à dessein par écriture. Il est important de noter que ce système ne permet pas de définir des exceptions qui seraient soustraites à la liste des substitutions implicites à l'écriture : il faut préciser toutes les

[22] C'est ce que les spécifications d'OpenType appellent en anglais des « features ».

[23] C'est ce que les spécifications d'OpenType appellent en anglais des « lookups ».

fonctionnalités à effectuer pour une langue particulière, aucune des fonctionnalités implicites (`dflt`) ne s'appliquera à cette langue.

Il nous reste à introduire deux autres éléments de la hiérarchie : les fonctionnalités et les groupes. Les fonctionnalités correspondent à un ensemble de règles typographiques qui définissent la manière d'utiliser des glyphes pour une langue donnée. Ainsi, une police arabe pourra-t-elle définir une fonctionnalité qui précise les règles de substitution des glyphes initiaux alors qu'une police kanji pourra définir une fonctionnalité qui permet de positionner verticalement des glyphes. Une fonctionnalité peut définir des substitutions de glyphes, des positionnements de glyphes ou les deux. Comme ces fonctionnalités doivent être interprétées par un moteur de rendu (ou de façonnage pour Microsoft), il est préférable de n'utiliser que des fonctionnalités prédéfinies[24] pour assurer une interopérabilité maximale. Enfin, un groupe rassemble une série de règles, c'est-à-dire une série de substitutions (GSUB) ou de positionnements (GPOS). Un groupe peut être utilisé par plusieurs fonctionnalités. La répartition des règles parmi les groupes est laissée au développeur.

L'exemple, tronqué ci-dessous, illustre quelques fonctionnalités et groupes associés à l'écriture arabe. Trois fonctionnalités sont brièvement illustrées : la composition et décomposition de glyphes (`ccmp`), les formes initiales (`init`) et les ligatures obligatoires (`rlig`), d'autres fonctionnalités sont bien sûr permises, les plus fréquentes sont celles associées aux autres aspects contextuels des lettres arabes (`isol`, `medi`, `fina`), elles sont en fait obligatoires pour Uniscribe.

Alors que la sémantique des fonctionnalités enregistrées est définie et publique, les règles précises qui constituent la mise en œuvre de ces fonctionnalités dépendent du fondeur. L'ordre des fonctionnalités est également défini pour une écriture donnée, la fonctionnalité `ccmp` est toujours la première exécutée quelle que soit l'écriture. En d'autres mots, on effectue les règles précisées par le fondeur dans la fonctionnalité `ccmp` avant de considérer toute autre fonctionnalité.

```
Arabe <arab>
  Implicite <dflt>
    Composition et décomposition <ccmp> (fonctionnalité)
      décomposer (groupe GSUB)
          إ -> ا + ٕ (décomposer alif hamza)
          (autres substitutions)

      composer (groupe GSUB)
           ٕ + ً -> hamza_fathatan
          (autres substitutions)

    Formes initiales <init> (fonctionnalité)
         lam -> lam.init
         çad-> çad.init
         sîn -> sîn.init
         (autres substitutions)

    Ligatures obligatoires <rlig> (fonctionnalité)
         lam alif -> lam_alif
```

[24] Yannis Haralambous énumère ces fonctionnalités dans l'annexe D.10 de son ouvrage cité dans la bibliographie qui conclut cet article.

Bien que chaque fonctionnalité OpenType puisse en théorie[25] être associée à des langues différentes et puisse permettre de traiter différemment les glyphes de ces langues, certaines fonctionnalités prédéfinies – et donc prises en compte par de nombreuses applications – ont été explicitement conçues pour prendre en compte le niveau linguistique introduit dans la hiérarchie d'accès aux fonctionnalités. Une de ces fonctionnalités se nomme « Formes locales » (`locl`), elle permet d'associer des variantes de glyphes à une langue particulière. Un exemple typique est celui du bulgare qui écrit quelques-unes de ses lettres cyrilliques différemment des lettres cyrilliques habituelles. Une police bulgare pourra décider que les formes bulgares sont implicites et déclarer des variantes de glyphes pour le russe. Une police russe fera sans doute l'inverse.

6.2. Intégrer le tout

Tout cela est bien beau. Mais comment tous ces différents concepts et fonctions s'intègrent-ils en pratique ? Tout commence par la détection de l'écriture choisie. Dans le cas d'Uniscribe, c'est le rôle de la méthode `ScriptItemize`. Ensuite, le moteur de rendu détermine la langue utilisée pour chaque segment analysé par `ScriptItemize`, celle-ci peut être précisée par du balisage (l'attribut `xml:lang` en XML par exemple) ou, en l'absence d'un tel balisage, correspondre à la langue du profil de l'utilisateur actuel.

À partir de maintenant le moteur de rendu peut commencer à effectuer les fonctionnalités définies dans la police choisie. Toutefois, certaines des règles peuvent nécessiter l'intervention de l'utilisateur (pour lui permettre par exemple de choisir la formation de ligatures optionnelles `dlig` comme les ﬅ et ﬆ en latin). Il est important de noter que l'ordre d'application des fonctionnalités est fixe pour une écriture donnée. Dans le cas de l'arabe, l'ordre standard[26] est précisé dans le tableau 3 ci-après. Certaines des fonctionnalités énumérées ne s'appliquent qu'à certaines langues qui utilisent l'écriture arabe.

Fonctionnalité	Description	Table	Obligatoire
Formes linguistiques			
ccmp	Composition et décomposition	GSUB	
isol	Forme isolée du glyphe	GSUB	
fina	Forme finale du glyphe	GSUB	X
medi	Forme médiale du glyphe	GSUB	X
init	Forme initiale du glyphe	GSUB	X
rlig	Ligature obligatoire	GSUB	X
calt	Alternatives de jonction	GSUB	
Formes typographiques			
liga	Ligatures habituelles	GSUB	
dlig	Ligatures optionnelles	GSUB	
cswh	Lettres à paraphes contextuels	GSUB	
mset	Positionnement de diacritiques à l'aide de substitution	GSUB	

[25] En pratique ce n'est pas toujours le cas, certaines fonctionnalités ne devraient pas être utilisées avec certaines écritures. C'est le cas, par exemple, de la fonctionnalité « cpsp » – espacement des textes écrits en capitales – qui ne devrait pas être utilisé avec les écritures cursives comme l'arabe.

[26] Précisé ici : <http://www.microsoft.com/typography/otfntdev/arabicot/features.htm>.

Positionnements			
curs	Positionnement cursif	GPOS	
kern	Crénage par paires	GPOS	
mark	Positionnement de diacritiques par rapport au glyphe de base	GPOS	
mkmk	Positionnement de diacritiques par rapport à un autre diacritique	GPOS	

Table 3. – Ordre d'exécution des fonctionnalités par le moteur de façonnage arabe

6.3. Rôle du moteur de rendu

Prenons un extrait assez typique de la fonctionnalité `init` qui fournit la forme initiale de glyphes arabes :

uni062C ج → uni062C.init[27] ﺟ
uni0628 ب → uni0628.init ﺑ

La première ligne de cet extrait se lit de la façon suivante : quand le moteur de façonnage trouve un djîm non contextualisé, le remplacer par sa forme initiale. Mais voilà, rien n'indique dans la police quand il faut appliquer cette fonctionnalité ! C'est le moteur de façonnage (cf. ④ de la figure 8) qui détermine quand il invoquera les substitutions liées à une fonctionnalité particulière. Pour ce faire, il doit pouvoir découper un texte en « mots » pour l'écriture arabe et identifier les caractères qui se trouvent en positions initiales.

6.4. Ordre d'exécution des fonctionnalités et des groupes

D'emblée, rappelons que le moteur de rendu effectuera d'abord toutes les substitutions (GSUB) avant de passer au traitement des positionnements (GPOS).

L'ordre dans lequel les fonctionnalités sont exécutées dépend du moteur de rendu. Dans le cas des moteurs de façonnage de Microsoft, il arrive que certaines fonctionnalités GSUB soient traitées séquentiellement alors que d'autres sont effectuées simultanément. En règle générale, les moteurs de façonnage distinguent entre les comportements typographiques essentiels des écritures et les raffinements typographiques. Les fonctionnalités essentielles s'effectuent avant les autres. Dans le cas de l'écriture latine, la fonctionnalité `ccmp` s'appliquera avant `liga` ou `dlig`, parce que `ccmp` (par exemple, la composition de caractères par l'adjonction de diacritiques) est considéré comme essentiel alors que la formation des ligatures est considérée comme du peaufinage typographique.

Pour les écritures dites complexes (par exemple la dévanâgarî, l'arabe, le khmer ou le thaï), les fonctionnalités sont habituellement exécutées séquentiellement alors que, pour les écritures prétendument simples (par exemple, latine ou cyrillique) elles sont habituellement exécutées simultanément.

L'exécution simultanée de plusieurs fonctionnalités signifie que les groupes de règles respectifs de ces fonctionnalités seront exécutés dans l'ordre de leur définition dans la liste

[27] Nous adoptons ici pour le nom des glyphes la convention PostScript et celle du langage de description de fonctionnalités OpenType d'Adobe. Voir l'article sur la police Hapax Berbère de ces mêmes actes pour plus de détails.

générale des groupes de règles. Cette liste énumère tous les groupes de règles de toutes les fonctionnalités.

Pour clarifier les choses, imaginons une police qui définit trois fonctionnalités (A, B et C) et leurs groupes de règles :

```
FonctA
     GroupeA1
     GroupeA2
     GroupeX
FonctB
     GroupeB1
     GroupeB2
     GroupeX
FonctC
     GroupeC1
     GroupeC2
     GroupeC3
     GroupeX
```

Le groupe X ci-dessus est un groupe partagé par plusieurs fonctionnalités. Comme nous le disions plus haut, il existe une liste de tous les groupes indépendante de la liste des fonctionnalités. L'ordre des règles dans cette liste est indépendant de l'ordre des fonctionnalités. Supposons que la liste des groupes est ordonnée de la façon suivante :

```
GroupeA1
GroupeB1
GroupeC1
GroupeA2
GroupeB2
GroupeC2
GroupeC3
GroupeX
```

Maintenant, si le moteur de façonnage exécute séquentiellement les fonctionnalités A et C, tous les groupes associés à FonctA seront d'abord traités avant que tous les groupes associés à FonctC le soient. Ce qui donne dans l'ordre :

```
GroupeA1
GroupeA2
GroupeX
GroupeC1
GroupeC2
GroupeC3
GroupeX
```

Si les fonctionnalités A et C s'effectuent simultanément, les groupes s'exécuteront alors dans l'ordre suivant :

```
GroupeA1
GroupeC1
GroupeA2
GroupeC2
GroupeC3
GroupeX
```

6.5. Ordre d'exécution des règles

Il est important de signaler que le moteur de rendu applique d'abord un groupe de règles à tous les glyphes d'un passage avant de passer au groupe de règles suivant. Pour chacun des

glyphes du passage, on passe en revue toutes les règles du groupe jusqu'à ce qu'une règle s'applique. Si c'est le cas, on exécute la transformation décrite par cette règle et on passe au glyphe qui suit le contexte d'application de la règle (représenté à la gauche des flèches), sinon on laisse le glyphe en question intact et le moteur de rendu passe au glyphe suivant.

Ce qui donne en pseudocode :

```
pour chaque groupe dans les fonctionnalités sélectionnées faire
    glyphe ← premier glyphe du passage;
    tant que il reste des glyphes faire
        règle_exécutée ← faux;
        règle ← première règle du groupe;
        tant que il reste des règles et pas règle_exécutée faire
            si condition d'exécution de règle est remplie alors
                exécuter règle;
                glyphe ← glyphe suivant le contexte d'exécution;
                règle_exécutée ← vrai;
            sinon
                règle ← prochaine règle du groupe;
        si pas règle_exécutée alors
            glyphe ← prochain glyphe du passage;
```

Ainsi, si on remplace « f f » par « f_f » le glyphe suivant sera celui qui suit « f f » et il n'est pas possible de remplacer « f_f » dans cette itération. Pour contourner ce comportement, il suffit de définir un autre groupe (car on itère sur tous les glyphes pour un groupe), le premier groupe produira le « f_f », le second pourra modifier le « f_f ».

Soit le passage « …raffiné… » pour lequel on désire remplacer le « ffi » par une ligature. On pourrait a priori penser que deux règles de substitution simples comme celles ci-dessous feraient l'affaire.

```
Latin <latn>
  Implicite <dflt>
    Ligature <liga> (fonctionnalité)
      ligatures_ff_ffi (groupe)
        f f    →  ff
        ff i →  ffi
```

Imaginons que le moteur de rendu considère le glyphe correspondant au premier « f » :

.	.	.	r	a	f	f	i	n	é	.	.	.
					^							

Il trouve une règle qui s'applique (« f f » → « ff »). Après la substitution, le glyphe courant devient le i, le glyphe qui suit le contexte d'exécution de la règle, à savoir « f f ». Schématiquement, on a donc :

.	.	.	r	a	ff	i	n	é	.	.	.
						^					

Mais voilà la seconde règle de ce groupe ne s'exécutera pas, car la première règle d'un groupe, dont la condition d'exécution est remplie, gagne et met fin à la boucle. Le résultat final ne contiendra donc que la ligature « ff ». Pour pallier ce problème, il suffit d'introduire un deuxième groupe dans la même fonctionnalité :

```
Latin <latn>
  Implicite <dflt>
    Ligature <liga> (fonctionnalité)
      ligature_ff (groupe)
         f f    →   ff
      ligature_ffi (groupe)
         ff i →   ffi
```

On aurait évidemment pu, dans notre cas, n'avoir qu'une seule substitution qui remplaçait « f f i » par ffi, si on ne désirait effectuer que ce type de ligature et ne pas traiter la formation des autres ligatures de cette famille (f f → ff, f i → fi). Quand on veut traiter cette famille de ligatures dans un même groupe, il est important de préciser en premier lieu dans un groupe les substitutions les plus longues. En effet, si on définit d'abord dans un même groupe une substitution plus courte comme dans les règles ci-dessous,

f f → ff
f f i → ffi

il est facile de vérifier que l'on n'obtiendra jamais la ligature ffi puisque les f f seront d'abord changés en ligature ff et que la condition d'exécution de la règle suivante ne sera jamais vraie, puisqu'il ne restera plus de f f dans le texte d'origine.

6.6. Exécution parallèle et simultanée des fonctionnalités et règles

Revenons un instant sur l'ordre d'exécution des fonctionnalités et des règles dans OpenType. Si on considère le pseudocode présenté ci-dessus comme une méthode `appliquerRègles()`, membre d'une classe `passage` qui contient le passage à rendre, méthode à laquelle on passera en arguments les fonctionnalités à appliquer, on peut alors représenter l'ordre d'exécution des fonctionnalités et de leurs règles respectives de la façon suivante pour les écritures dites simples :

```
passage.appliquerRègles({ccmp, rlig, …}) # application simultanée des
                                    # fonctionnalités GSUB ccmp, rlig, …
…
passage.appliquerRègles({kern, mark, mkmk, …}) # simultanée GPOS
```

Et pour les écritures complexes :

```
passage.appliquerRègles({ccmp}) # GSUB séquentiel
passage.appliquerRègles({rlig}) # GSUB séquentiel
…
passage.appliquerRègles({kern}) # GPOS séquentiel
passage.appliquerRègles({mark}) # GPOS séquentiel
```

Maintenant, cette distinction entre écritures simples et complexes n'est peut-être pas opportune car toutes les écritures – même la latine quand on considère une mise en œuvre perfectionnée – sont complexes, c'est la prise en charge par le moteur de façonnage correspondant qui peut être simple ou complexe. On peut se demander si cette différence n'est donc pas gratuite et ne complexifie pas la description et le fonctionnement des moteurs OpenType.

7. Conclusion

Nous avons vu dans cet article comment les caractères d'Unicode sont transformés en glyphes pour être finalement rendus à l'écran ou sur tout autre dispositif comme une imprimante. Dans cette brève introduction aux technologies de polices de glyphes nous avons voulu

fournir un aperçu de ces différents processus et bien faire comprendre les implications liées à des techniques aussi pointues qu'OpenType. En particulier, il ne suffit pas d'écrire quelques substitutions dans une police pour qu'elles soient prises en compte. En effet, les fonctionnalités sont définies par les fabricants de moteur de rendu (p.ex. Uniscribe sur Windows), le concepteur de police devra donc respecter les noms de ces fonctionnalités s'il veut utiliser ce moteur. Il devra également prendre en compte l'ordre défini dans lequel ces fonctionnalités sont exécutées pour une écriture donnée.

Enfin, on ne peut que vivement recommander la lecture de l'excellent livre de Yannis Haralambous mentionné ci-dessous à tous ceux qui désirent comprendre dans le détail les différentes techniques liées aux polices de glyphes, y compris celles du Macintosh (AAT) ou de TeX (Metafont).

8. Ressources

- Fontes et codages, par Y. Haralambous, O'Reilly France, 2004, ISBN 2-84177-273-X
- Unicode en français: <http://hapax.qc.ca>
- Listes de diffusion et forum de discussion
  - Typo : <https://www.irisa.fr/wws/info/typographie>
  - Unicode Afrique : <http://fr.groups.yahoo.com/group/Unicode-Afrique/>
  - Unicode fr : <nntp://fr.comp.normes.unicode>
- FontForge (logiciel libre de création de polices) : <http://fontforge.sourceforge.net/> (version française existe)
- Babelmap (outil de visualisation des caractères) :<http://hapax.qc.ca/BabelMap_fr.html>