THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Université Toulouse III - Paul Sabatier
Discipline Informatique

Présentée et soutenue le 1e avril 2008 par Guylhem Aznar

Informatisation d'une forme écrite de la Langue des Signes Française

---

JURY

Directeur de thèse :
M. Patrice DALLE — *Professeur, Université de Toulouse III*

Rapporteurs :
M. Bernard LANG — *Directeur de Recherche, INRIA Rocquencourt*
M. Jaime LOPEZ KRAHE — *Professeur, Université de Paris VII*
M. Antônio da ROCHA COSTA — *Professeur, Universidade Católica de Pelotas*

Examinateurs :
Mme Brigitte GARCIA — *Maître de Conférence, Université de Paris VII*
M. Jean Pierre JESSSEL — *Professeur, Université de Toulouse III*

École Doctorale Mathématiques, Informatique et Télécommunications de Toulouse
Unité de recherche UMR 5505

Résumé

Cette thèse étudie les moyens pour informatiser une forme écrite de la langue des signes.

L'état de l'art étudie respectivement l'écriture, l'encodage, la localisation des systèmes d'exploitation à travers l'exemple de linux, les langues signées, et l'informatisation des langues signées.

Il est suivi par une observation au Brésil d'une classe d'enfants utilisant les logiciels existants.

Suite à ces éléments, une nouvelle approche, basée sur une segmentation du problème en couches fonctionelles intégrables au système d'exploitation, est proposée.

Le fonctionnement des couches est étudié, ce qui permet d'identifier des problèmes de variabilités. Un algorithme pour gérer ces variabilités est proposé.

La problématique de l'encodage est alors étudiée plus en détail, en comparant les méthodes qu'il est possible de mettre en oeuvre, et en proposant une approche intermédiaire entre les objectifs de vitesse et de place, vu les impératifs d'Unicode.

Liste des publications associées :
http://www.irit.fr/publications.php3?code=2628&nom=Aznar%20Guylhem

« Langue naturelle mais sans écriture, la langue des signes n'a pu être introduite jusqu'ici dans l'enseignement que comme un palliatif, utile mais difficile à maîtriser, destiné principalement à la transmission d'informations et à l'amélioration de la communication.

Rendre possible son écriture, c'est donner à ses locuteurs et à ses utilisateurs la possibilité de prendre du recul sur la pratique qu'ils en font, d'en maîtriser les performances et les effets secondaires. C'est une possibilité pour la langue des signes d'accéder au statut de véritable langue d'enseignement. »

P. Jouison in *Écrits sur la langue des Signes Française*. B.Garcia (Éd.), L'Harmattan, 1987.

Remerciements

Cette thèse d'informatique, qui s'est étendue sur une durée de 4 ans, a été effectuée dans des conditions matérielles pas toujours les meilleures, du fait qu'elle a été réalisée en parallèle à mon internat de médecine, et surtout du hasard de la vie, qui m'a fait tester les conséquences physiologiques de l'application d'une grande énergie cinétique sur le passager d'un véhicule à moteur.

Je ne regrette toutefois rien, et serais immédiatement prêt à recommencer une telle aventure scientifique.

En effet, grâce à certaines personnes, j'ai eu l'extraordinaire opportunité d'approfondir un sujet à la fois intéressant, multidisciplinaire et extrêmement prometteur ainsi que, je le crois, d'apporter des contributions originales et pratiques à la problématique rencontrée.

Pour cela, il me faut tout d'abord remercier les professeurs Patrice Dalle et René Caubet de l'UFR Mathématique Informatique et Gestion, ainsi que les Professeurs Hugues Chap et Jacques Simon de l'UFR de Médecine Purpan, pour m'avoir permis de réaliser mon DEA. Si je n'avais pas reçu leur soutien, je n'aurais certainement pas pu terminer mon DEA, et donc commencer cette thèse...

Mais parmi eux, je tiens à remercier plus particulièrement le professeur Patrice Dalle, qui m'a offert chaque année, du début de mon DEA à la fin de ma thèse, sa confiance et son amitié, et ce même dans les moments les plus difficiles que les aléas de la vie vinrent s'ajouter à ma troisième année de thèse.

Ensuite, je dois aussi remercier très chaleureusement le professeur Antônio Da Rocha Costa, pour m'avoir accueilli dans son laboratoire au Brésil, pour réaliser sur place des études complémentaires sur l'utilisation qui était faire de l'écriture de la Langue des Signes en m'introduisant auprès de Marianne Stumpf, enseignant l'écriture de la Langue des Signes à l'Escola Frei Pacífico.

Si le sujet ne fait que naître en France, il est déjà fort développé au Brésil : il nous faut donc savoir tirer parti de leurs apports et de leur expérience.

Mais cette liste ne serait pas complète sans remercier aussi Valérie Sutton, qui m'a fait voir ses travaux, et m'a encouragé dans ma démarche. Bien que dans des conditions difficiles, elle continue de s'occuper de ceux qui ont eu dans la vie moins de chance qu'elle, en offrant ses moyens pratiques et ses travaux intellectuels à la communauté.

Avant propos

La Langue des Signes Française (LSF) est une langue utilisant la communication visuo-gestuelle. D'aucuns pourraient se demander l'intérêt qu'il peut y avoir à l'écrire, ce qui pourrait d'ailleurs être vu comme un facteur de dégradation de l'information véhiculée, encore plus si l'on fait appel à l'informatique, qui souvent n'a pas la souplesse de l'écriture manuscrite.

Toutefois, le stockage et le transfert des connaissances dans notre société actuelle nécessitent de recourir à des moyens traditionnels, qui ne sont pas dans ce cas parfaitement adaptés à la richesse de l'information. En effet, tout le monde n'a pas encore à sa disposition, pour enregistrer des informations en LSF, un caméscope et un magnétoscope, alors que les stylos et le papier sont très largement accessibles, et que l'informatique domestique connaît un développement inimaginable il y a quelques décennies.

Écrire avec un stylo et du papier - ce n'est certes pas un sujet d'informatique. Ce n'est d'ailleurs pas non plus le sujet de cette thèse, puisqu'il est question de l'adaptation de l'informatique à l'écriture, ici, des langues des signes.

Du stylo à l'informatique existe une grande distance, mais une continuité logique: le transfert d'information. En effet, l'utilisation principale actuelle de l'informatique, que ce soit dans le domaine de la bureautique ou de l'internet, est de véhiculer une information, majoritairement écrite.

Dans le cas de la langue des signes, le transfert de cette information se voit entravé de plusieurs manières : tout d'abord parce que le choix d'un formalisme d'écriture, aussi complet soit-il, ne saurait remplacer l'aspect 3D+T de la communication visuo-gestuelle, et ensuite parce que les outils informatiques traditionnellement employés ne sont pas adaptés à des formes d'écriture justement si complètes.

Comme il est présenté dans cette thèse, certains ont préféré adapter le formalisme d'écriture de la langue des signes aux contraintes de l'informatique telle que prévue pour l'écriture des langues latines que nous employons tous les jours.

Je crois toutefois que c'est faire fausse route, et je postule que les outils informatiques peuvent être adaptés à une langue aussi riche que la langue des signes. Il y a pour cela plusieurs raisons : tout d'abord, ne serait-ce que parce que l'outil doit s'adapter à l'usage qui en est fait, et non le contraire, et ensuite parce que mes travaux ont pour cadre une thèse d'informatique où la complexité du travail réalisé n'est pas un problème.

L'objet de cette thèse est de supprimer les freins qu'il peut y avoir à l'utilisation de l'informatique comme support de l'écriture de la langue des signes, afin de ne pas induire des dégradations supplémentaires de l'information véhiculée ; il sera laissé aux linguistes le soin d'améliorer le passage de l'information visuo-gestuelle à l'information écrite, qui constitue un parallèle indissociable à ce travail.

Le sujet de mon DEA, en rapport avec le groupement perceptuel, pourrait sembler en continuité ténue avec ce sujet de thèse.

Toutefois, le sujet était précisément « Conditions et critères pour l'émergence de l'iconicité dans la Langue des Signes Française » - et non simplement le groupement perceptuel.

Passer à l'informatisation de l'écriture de la langue des signes ne me semble donc pas un pari osé, mais plutôt une extension directe des travaux précédents : en effet, faire émerger l'iconicité, ou toute autre dimension de la LSF n'a de sens que si les informations extraites peuvent être stockées, comparées à des transcriptions ou entre elles, puis évaluées.

Pour faire une parallèle sportif, avant de vouloir organiser des épreuves d'athlétisme, il faut un moyen d'enregistrer les performances réalisées, par exemple en chronométrant, et un moyen de comparer ces performances, dans ce cas en comparant les temps mesurés. Une évaluation pourra alors avoir lieu, et un gagnant pourra être identifié, sans devoir faire courir à nouveau tout le monde.

Dans le cas des langues des signes, on parle actuellement d'avatars signants, c'est à dire de personnages virtuels recréant cette communication visuo-gestuelle, alors même qu'il n'y a pas de forme intermédiaire commune et acceptée pour stocker l'information transmise.

Ceci me semble être un pari beaucoup plus osé que le sujet de cette thèse...

Mes travaux ont donc été guidés par les besoins du traitement d'image, mais ont intégré des apports dans le domaine de l'encodage et de l'algorithmique.

Cette thèse aborde donc les problèmes que l'informatisation d'un formalisme graphique de la Langue des Signes devra prendre en compte.

De tels problèmes ne se rencontrent que rarement quand on est dans une langue et un formalisme d'écriture homogène - encore moins de nos jours, où la plupart des langues disposent désormais d'une informatisation robuste.

Toutefois, ils sont présents dès lors que plusieurs langues sont en présence et qu'un formalisme graphique présente des caractéristiques très spécifiques.

Par exemple, comment un texte écrit sous forme de glyphes arrangés verticalement peut-il intégrer une citation d'un texte horizontal ?

La question peut sembler saugrenue et correspondre à un cas très rare - mais comme on le verra un peu plus bas, l'écriture de la langue des signes dans le formalisme SignWriting pose un tel problème pour son intégration avec du français écrit.

De même, à quel niveau peut se situer le copier-coller dans un formalisme d'écriture qui utilise un assemblage bidimensionnel d'éléments unitaires ?

Au niveau d'un caractère, d'un groupe de caractères, ou du glyphe constitué par tous ces caractères ?

Que faire alors si la forme de ce glyphe dépend du contexte, tel que déterminé par les autres glyphes, comportant d'autres caractères ?

Et dans le cas d'un caractère, quels sont les paramètres ou variations significatives de celui-ci pour le différentier d'un autre, par exemple pour les regrouper ?

On voit à travers de telles questions que des points aussi basiques que le choix de l'encodage, de l'informatisation de cet encodage, du rendu et de la saisie doivent être examinés avant de passer à des sujets de plus haut niveau.

Table des matières

Table des figures

1. Introduction

Avant d'entrer plus en détail dans la problématique, il semble intéressant de se pencher sur les contraintes historiques qui ont conduit au développement de ces problèmes.

Pour cela, il faut tout d'abord s'intéresser à la langue des signes, et plus spécifiquement à la Langue des Signes Française (LSF).

De manière simple, celle-ci peut se définir comme une réponse au problème du transfert de l'information pour les sourds, aussi bien entre eux qu'avec les entendants.

Elle ne serait être réduite à un simple « transfert d'information », puisqu'une langue ne sert qu'entre mille autres choses à informer : elle sert à plaisanter, à mentir, à expliquer, à raconter...

Toutefois, au sens informatique, une langue est un moyen de transférer une information, un transcodage de données mentales en un support physique, quelle que soit la nature des données ainsi transportées : humour, mensonges, explications, histoires...

La LSF est un tel transcodage, efficace car adaptée aux contraintes du support.

Par contre, elle a été beacoup critiquée : il faut donc la replacer dans son contexte historique avant d'en étudier les caractéristiques pour mieux comprendre les critiques qui lui ont été adressées, et les conséquences actuelles de cette situation.

Pour comprendre ces éléments, un autre cadre doit alors être posé, plus éducatif et linguistique qu'informatique.

Après seulement pourront être envisagées les contraintes informatiques existantes, ainsi que la direction donnée à ce travail.

1-1. La Langue des Signes Française

1-1-1. Présentation

La langue des signes est une langue de communication visuo-gestuelle, utilisée par les communautés de sourds pour communiquer entre eux.

À partir de gestes, de mouvements du corps et de mimiques faciales, des messages complexes sont transmis d'un locuteur à l'autre.

De nombreuses langues des signes existent dans le monde, avec des différences plus ou moins marquées.

La LSF est la forme utilisée en France. Les chiffres disponibles sont rares et toujours incertains. On estime généralement à environ 150 à 200 000 le nombre de locteurs de la LSF, pour moitié sours et pour moitié entendants. Il s'agit de la langue naturelle des sourds, dont le statut de langue à part entière n'est plus en question.

Son histoire est assez complexe en France, et notamment marquée par une longue période d'interdiction, du congrès de Milan en 1880, jusqu'à 1991 où un amendement, abusivement appelé « Loi Fabius », lève cette interdiction.

Si la critique de la LSF a été surtout faite dans des buts politiques, en suivant un idéal d'égalité et d'intégration des sourds dans la population, elle a aussi été guidée par une mauvaise compréhension de la nature de la LSF, qui est une véritable langue, et non un ersatz de langue.

1-1-2. Brève chronologie

On peut retracer la brève chronologie suivante *[1853-Berthier-Éducation], [1996-Baynton-Forbidden_Signs], [2003-Garcia-Notation_Transcription_LS]:*

18e siècle: à la suite de quelques précurseurs, quelques sourds de riche famille ont pu être rééduqués avec succès, dans une perspective exclusivement oraliste et médicale. On a découvert qu'ils étaient intelligents et qu'ils pouvaient apprendre un langage pour exprimer leur pensée.

1760 : l'Abbé de l'Épée fonde sa première école, rue des Moulins à Paris pour enseigner à des enfants sourds d'origine modeste. Il invente une méthode pédagogique à partir des signes des enfants, les « signes méthodiques », calqués sur le français. Sa méthode dite « méthode française » connait un vaste rayonnement français et européen.

1789 : Mort de l'Abbé de l'Épée.

1794 : Transfert de l'école des sourds-muets de Paris du couvent des Célestins vers l'emplacement actuel de l'INJS (Institut national des jeunes sourds), rue Saint-Jacques.

19e siècle : A. Bébian notamment senseur à l'Institut, donne un statut de langue d'enseignement à la langue des signes qui s'élabore dans l'école. Bébian tente de lui donner une forme écrite mais choisit de ne pas mener ce travail à terme.

1816 : Laurent Clerc, répétiteur sourd à l'Institut, accompagne T.H Gallaudet aux États-Unis pour fonder la première école pour enfants sourds et exporte la LSF sur le continent américain où elle se développe rapidement et se mèle à la LS locale.

1830-1880 : La France subit un contrôle croissant de l'État dans les écoles où l'oralisme fait des émules. Les enseignants sourds sont progressivement dépossédés des postes stratégiques et évincés des Conseils d'administration où se prennent les décisions importantes. Dans ces structures, l'idée s'impose qu'un enfant sourd ne peut accéder à l'intelligence et donc à la connaissance que par la langue vocale.

1880 : Congrès de Milan : interdiction de la langue des signes comme langue d'enseignement et de communication dans toutes les écoles pour enfants sourds en Europe. À Milan ne sont données que des « résolutions », sans caractère juridique, mais celles-ci seront suivies par tous, États-Unis exceptés.

1880-Années 1970 : Les enfants sourds sont pour la plupart illettrés ou en grande difficulté face à l'écrit et les adultes en marge de la société.

1975 : Congrès mondial des sourds de Washington : la délégation française découvre une communauté sourde états-unienne puissante, intégrée, instruite, revendicative où les ravages de l'oralisme sont moindres.

1976 : Création du centre IVT (International Visual Theatre) à Vincennes par Alfredo Corrado, acteur états-unien sourd, Jean Grémion, metteur en scène entendant, et Bill Moody, interprète états-unien ; IVT devient un centre de recherches pour une expression théâtrale sourde.

Années 1980 : Émergence au sein de la communauté sourde d'un fort mouvement en faveur de l'éducation bilingue en France.

1991 : « Loi Fabius » : les parents d'enfants sourds peuvent choisir entre deux options pour l'éducation de leur enfant : français oral et écrit (oralisme) ou LSF et français oral et écrit (bilinguisme) : *Loi 91-73 (titre III) article 33 du 18 janvier 1991 portant dispositions relatives à la santé publique et aux assurances sociales.*

1993 : Emmanuelle Laborit, actrice sourde, gagne le prix Molière pour son interprétation dans « les Enfants du Silence ». Elle publie peu après un ouvrage autobiographique, « Le cri de la mouette ».

1994 : L' émission hebdomadaire « L'œil et la main » sur Arte permet une meilleure connaissance de la culture sourde.

2002 - 13 février : Conférence de presse de Jack Lang annonçant la LSF comme pouvant être langue d'enseignement à l'école

2005 - 11 février : Loi en faveur de la personne handicapée, comprenant notamment la reconnaissance officielle de la LSF, le droit pour les parents de choisir le bilinguisme à l'école et la possibilité de recevoir un enseignement en LSF : *loi 2005-102 du 11 février 2005 pour l'égalité des droits et des chances, la participation et la citoyenneté des personnes handicapées.*

À travers cette chronologie, on perçoit que la LSF n'est pas une langue nouvelle, mais que les évolutions favorables à sa diffusion officielle sont relativement récentes, ce qui a une influence sur la qualité du support informatique dont elle dispose.

1-1-3. Structuration

La Langue des Signes est en effet une langue dite naturelle, c'est à dire apparue spontanément au sein de communautés de sourds à travers le monde et ayant évolué seule, selon l'utilisation qui en a été faite par ses locuteurs dans l'histoire.

Si les gestes sont souvent bijectifs avec des mots, la LSF est dotée d'un vocabulaire et d'une grammaire qui lui sont propres, et ne saurait en aucun cas être réduite à une simple code de transcription du français oral ou écrit.

La richesse des LS les place dans une position unique au sein des langues : il s'agit de langues originales : de la forme la plus évoluée et la plus complexe de communication visuo-gestuelle.

L'expression de phrases en LS ne se réduit pas aux gestes produits par les deux mains *[1995-Jouison-LSF]*.

En effet, c'est le corps tout entier qui peut être mis à contribution pour produire un énoncé, bien que l'on puisse distinguer quatres grands types de paramètres corporels qui interviennent principalement : les mains, l'expression du visage, le regard et les mouvements du corps.

Chaque énoncé est constitué d'une suite de gestes des mains que l'on appelle signes et qui sont agencés suivant une syntaxe régie par une logique spatiale. Un néophyte est aussi immédiatement frappé par la rapidité de communication et la simultanéité des gestes et des mimiques de la langue des signes. La simultanéïté est un paramètre important en LSF : plusieurs gestes par des segments corporels distincts aboutissent à la communication d'un seul ou de plusieurs concepts.

Un autre paramètre important est l'utilisation qui est faite de l'espace *[1997-Cuxac-Spatial]* : ainsi, notamment en ce qui concerne l'expression du temps, un signe, selon sa position en avant ou en arrière de l'épaule droite, sera ainsi situé dans le futur ou dans le passé.

Le nombre d'itérations du mouvement de positionnement indiquera le degré d'antériorité (exemple : « Lundi »), ainsi répéter deux fois en arrière le mouvement signifie « il y a deux lundis de ça »).

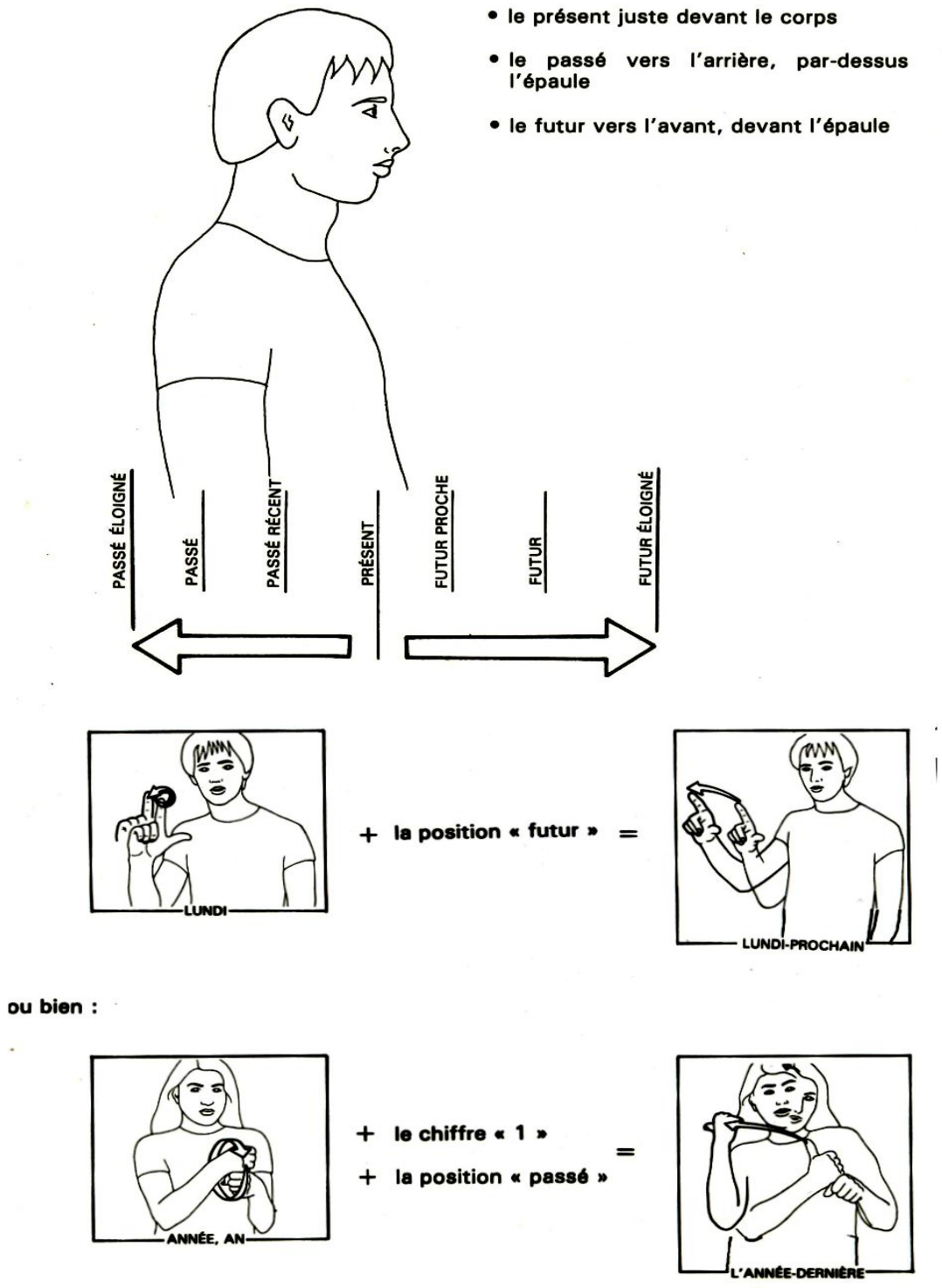


Figure 1: Ligne des temps, lundi prochain, l'an dernier, selon *[1987-Moody-LSF]*

Enfin, tout au moins dans le cadre du modèle théorique élaboré par Christian Cuxac *[1996-Cuxac-Iconicité]*, les LS sont structurées par et selon une logique qui relie l'unité gestuelle du signe avec l'expérience réelle, ce qui se matérialise dans la langue par diverses structures, appelées « structures de transfert » - de taille, de forme, de situation, ou de personnes *[2003-Sallandre-Unites_Discours]*

1-1-4. Recherches en LSF

Les LS sont des langues à part, originales et complexes.

Peu de chercheurs ont donc été amenés à s'y intéresser, et nombre de recherches ont été consacrées aux possibilités d'appareillage auditif des sourds (médecine), plutôt qu'à leur intégration sociale et à leurs spécificité culturelles et linguistiques

Pourtant, il y a près de deux siècles, *[1853-Berthier-Éducation]* on remarquait déjà que *« si l'éducation des sourds-muets devait se résumer dans l'articulation, la lecture sur les lèvres ou même la dactylologie, on ne pourrait commencer à leur enseigner une science (...) que lorsqu'ils seraient assez avancés dans l'étude de la langue pour comprendre les explications qu'on aurait à leur donner par cette voie »*.

Dans le cadre d'une présentation générale de la LSF, il faut en outre isoler deux caractéristiques essentielles à la compréhension de sa complexité.

1-1-5. Rôle du visage

L'informatisation des mimiques est un problème complexe à lui seul.

Des modifications fines sont à l'origine de différences sémantiques importantes : ainsi, par exemple, selon que les joues sont plus ou moins gonflées et les yeux plus ou moins ouverts, on exprimera telle ou telle nuance qualifiante.

Un paramètre extrêmement important dans le visage pour la grammaire est la direction du regard, qui sert notamment à distribuer les rôles ou encore à signaler (s'il est décroché de l'interloculeur) l'entrée dans une structure de transfert.

Le visage porte beaucoup d'informations linguistiques, mais pourtant est relativement négligé par les formalismes graphiques.

Il s'agit donc d'un des aspects les plus difficiles à rendre avec un formalisme graphique.

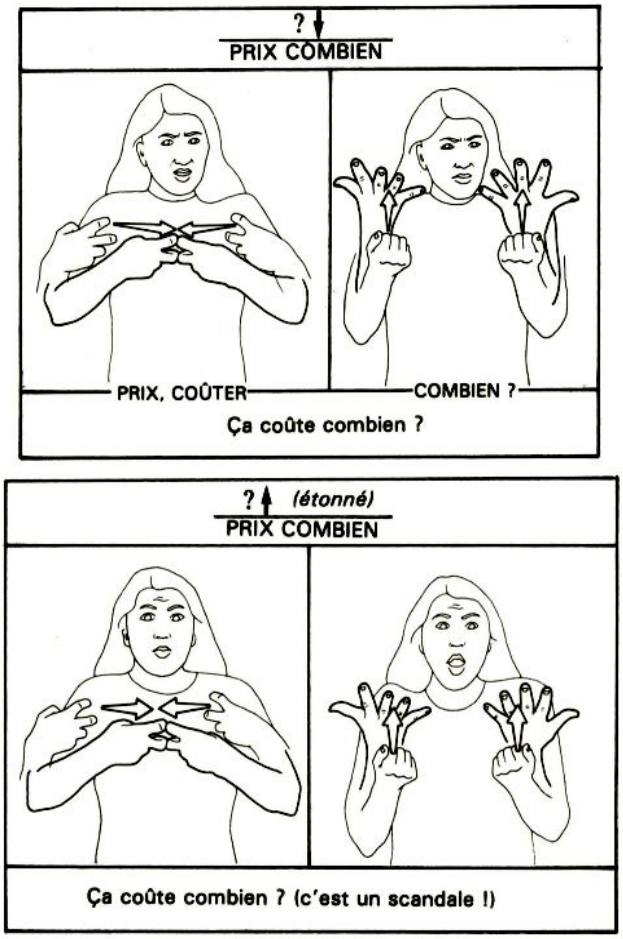


Figure 2: Différence de sémantique et de modalité indiquée par le visage

(i) Construction de références et suivi référentiel

La richesse et la complexité de la LSF sont bien illustrées par l'exemple de l'histoire du « petit chaperon rouge », mettant en jeu à la fois le regard et l'iconicité pour réaliser le positionnement itératif d'éléments.

Ainsi, en regardant un certain point du plan lui faisant face et en y réalisant simultanément un geste, le locuteur y positionne un concept *[1999-Sallandre-Transferts].*

Plusieurs concepts peuvent ainsi être successivement positionnés dans l'espace dit « de signation ». Ultérieurement, en regardant à nouveau l'endroit de l'espace où le concept a été posé, le locuteur peut rappeler ce concept, comme par le biais d'un pronom en français *[1997-Vergé-Regard]*.

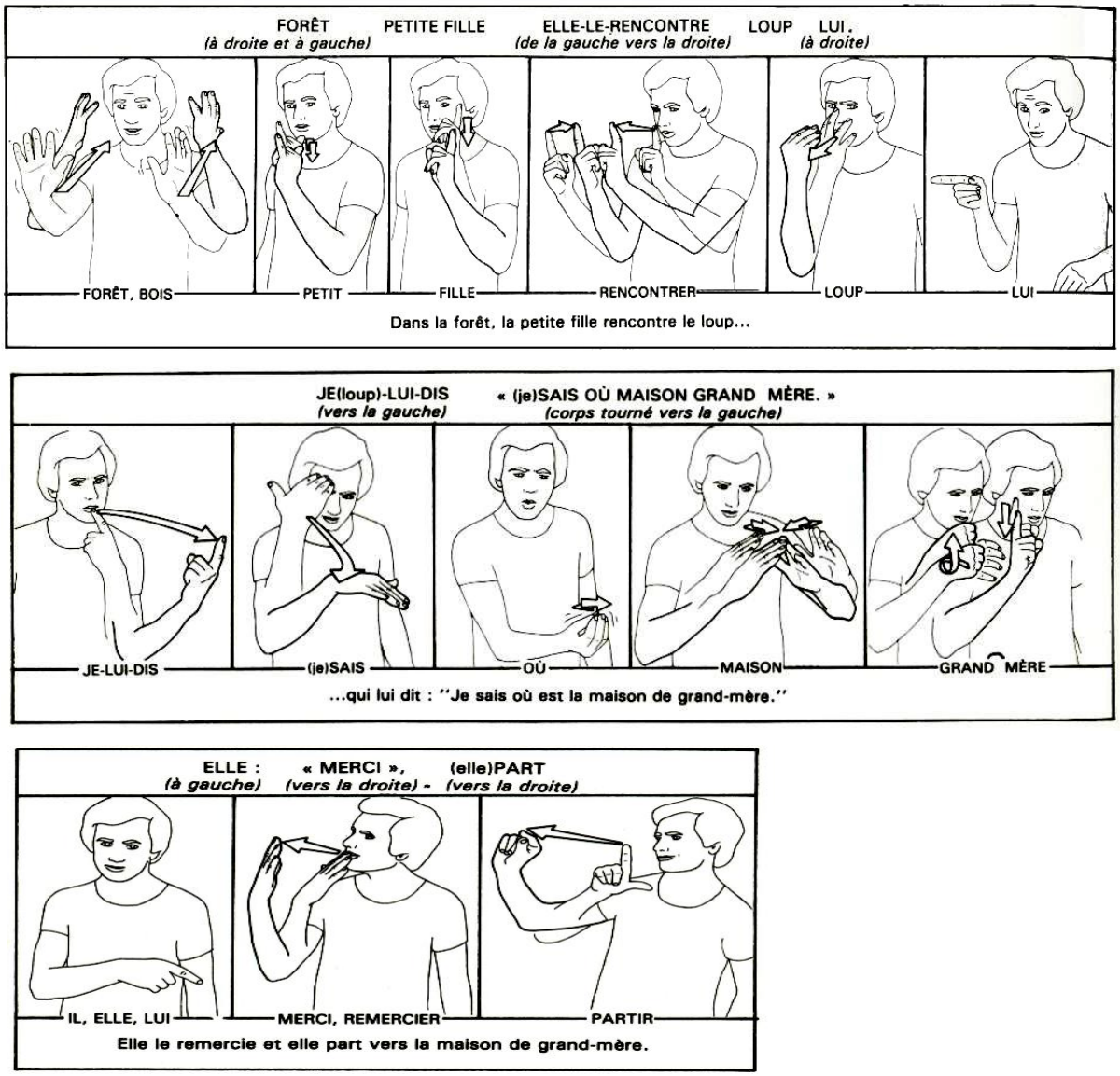


Figure 3: Le petit chaperon rouge en LSF

1-1-6. Critiques

Les détracteurs des LS, dont l'un des plus connus fut Alexandre Graham Bell (inventeur du téléphone), la considèrent comme un langage artificiel construit par des pédagogues *[1996-Baynton-Forbidden_Signs]*, en raison notamment des efforts initiaux de l'abbé de l'Épée .

Toutefois, il a été constaté *[2004-Fusellier-Apparition]* l'apparition d'une langue des signes même chez des sourds isolés.

On lui a aussi reproché d'éloigner les sourds du monde des entendants, de les confiner dans une communauté linguistique réduite, et de ne pas pouvoir utiliser la gestuelle comme mode de communication pour transmettre des pensées abstraites.

Les différentes appellations historiques des LS attestent du faible degré de fiabilité qu'on a peu leur accorder : « langage mimique » , « langage mimo-gestuel », « langage des gestes » , « langage des sourds-muets ».

L'absence d'une forme écrite a aussi été un argument utilisé pour refuser à la LSF l'appellation même de « langue ».

Mais de telles critiques sont sans vrai fondement, et aujourd'hui très datées : d'un point de vue philosophique, il s'est surtout agi de l'acceptation « d'une autre normalité » par les entendants, et de la désobjectisation du sourd, devenant une personne *[2002-Séguillon-Échec]*.

1-1-7. Efficacité

En effet, de nombreux travaux ont prouvé l'intérêt majeur et l'intégrité des potentiels linguistiques de la LSF comme vecteur de communication, qui devrait donc être privilégié par rapport aux tentatives d'apprentissage des langues orales avec « démutisation » et lecture labiale (oralisme), qui ont été marquées par un faible taux de succès.

En 1997, le professeur en Sciences de l'Éducation à l'université de Lyon II , Charles Gardou estime ainsi que *« 80% des jeunes sourds sont illettrés et moins de 10% d'entre eux accèdent à l'enseignement supérieur » [1998-Gillot-Rapport-Sourds].*

Si ces manques semblent imputables à la presque totale absence d'enseignement de la LSF en milieu scolaire et au choix majoritaire de l'oralisme, il semble bien que le problème de l'éducation des sourds soit également celui de leur accès à l'écrit *[1991-Cuxac-Problème-Écrit].*

Comment font les sourds pour accéder à l'information, puisque celle-ci est majoritairement transmise par l'écrit ? Pour commencer, comment font-ils pour apprendre, et pour stocker leur connaissances ? Peuvent-ils prendre des notes, ou s'échanger de brefs messages?

Pour répondre à ces questions, étudions le traitement de l'information par les sourds, afin de voir comment ils procèdent.

1-2. Traitement de l'information par les sourds

1-2-1. Transmission directe du savoir

La LSF permet une transmission directe du savoir en face-à-face. Cette transmission peut éventuellement être différée grâce à des dispositifs vidéo (VHS, DVD, fichiers multimédias), mais ceux-ci sont peu adaptés à une communication interpersonnelle, à la consultation rapide d'un index ou au stockage du savoir.

Les formes écrites ont beaucoup d'avantages, comme la transmission à faible bande passante. Les exemples modernes d'utilisation de l'écrit abondent : messagerie instantanée, SMS, bavardage sur internet...

1-2-2. Difficultés d'apprentissage pour les sourds

Actuellement comme rappelé plus haut, seule une minorité de sourds a un accès satisfaisant à l'écrit de la langue vocale, et la plupart ne maîtrisent pas bien l'écrit.

Il y a pour cela plusieurs explications potentielles :

- la LSF est le mode de communication naturel de la majorité des sourds en France.

Si la plupart apprennent aussi le français, il s'agit pour eux d'une deuxième langue, « apprise »

- pour accéder à l'écrit d'une langue, *quelle qu'elle soit,* il faut d'abord avoir acquis des compétences langagières et linguistiques permettant de comprendre ce que sont une langue et la communication linguistique *[2004-Hepsos-Conceptual_Precursors]*. Or, dépossédés le plus souvent de la LS et n'ayant qu'un accès limité à la forme orale de la langue vocale, les sourds sont privés de ces compétences.

Seule la LS leur perttant d'acquiérir les compétences linguistiques, et les savoirs sur le monde qui sont les deux conditions clés à une entrée pleine dans l'écrit.

Malgré toutes les technologies modernes, dont les implants cochléaires, les résultats sont ce que l'on connaît, avec une majorité de sourds illettrée, alors que des zones comme le sud Brésil ayant mis en jeu moins de moyens technologiques, mais plutôt favorisés l'apprentissage bilingue, obtiennent de bien meilleurs résultats *[2005-Skliar-Bilingual_Brazil]* !

Un autre exemple nous vient d'europe du nord, où la LS a une place comme langue reconnue dès le début de l'éducation et où les résultats sont bien meilleurs qu'en France.

Comme expliqué par Christian Cuxac, *« la méthode oraliste subordonne toutes les acquisitions à la connaissance préalable de la langue orale, seule admise à véhiculer les informations. On imagine l'effort intense auquel doit être constamment soumis un enfant qui n'entend pas , placé dans des conditions où l'essentiel de ce qu'il doit appréhender passe par l'audition. Ce choix pédagogique institutionnalise le retard scolaire voire l'échec scolaire de l'enfant sourd ». [1991-Cuxac-Problème-Écrit]*

Et dans le cas du bilinguisme, le problème est l'usage exclusif du français comme support de l'écrit. Les aspects pédagogiques sur ce sujet ont fait l'objet de nombreux travaux *[2005-Leroy-Pedagogie]*.

S'il n'est pas possible pour l'instant pour un sourd d'écrire en LSF, ce devrait être un objectif du système éducatif français, ne serait-ce que dans l'optique de l'égalité des chances.

Depuis 2005, de telles justifications ne sont plus à chercher : la LSF peut être utilisée comme première langue parce que c'est un droit : droit de communiquer à distance en LSF, droit d'avoir des traces, des documents en LSF, etc. (L*oi n° 2005-102 du 11 février 2005 pour l'égalité des droits et des chances, la participation et la citoyenneté des personnes handicapées*)

Pour permettre l'application de tels droits, la LSF doit être enseignée. L'écriture de la LSF facilite son enseignement par la mémorisation, l'évaluation *[1999-Rosenberg-Writing_ASL] ...*

Regardons plus précisément les avantages que peut présenter une forme écrite.

1-2-3. Intérêt d'une forme écrite de la LSF

Une forme écrite répondrait aux reproches qui sont faits à la LSF :

- de manière indirecte, par son usage dans l'enseignement (pour prendre des notes, utiliser des documents en LSF) elle pourrait aider à réduire l'illettrisme chez les sourds grâce à une forme écrite correspondant à leur première langue

- elle rapprocherait les sourds des entendants en permettant aux sourds d'utiliser des dispositifs usuels de transfert différé (ex : livres, ordinateurs, ...) au lieu de dépendre exclusivement de vidéos

- elle agrandirait les communautés linguistiques grâce à ces dispositifs de transfert différé permettant un plus grand accès aux connaissances disponibles, et facilitant des contacts multimédias à faible bande passante (messagerie instantanée, courriel...)

L'écrit est entretenu par des outils qui nécessitent eux-même un minimum de maîtrise de l'écrit (ex : livres, ordinateur, journal) : ne pas pouvoir les utiliser pour enrichir ses connaissances fait que le processus d'illétrisme se perpétue, et a pour résultats des difficultés d'accès à la connaissance, c'est à dire un isolement.

Les sourds ont compris la complexité de la situation, et utilisent pour l'instant la vidéo pour transmettre leurs informations sans devoir imposer une étape de traduction dans une autre langue !

Mais l'accès à la connaissance se fait par l'acquisition de données préalablement stockées et dans le cas des sociétés humaines, la forme stockée des données est la forme écrite.

La problématique d'accès à la connaissance pour un sourd est donc double : au delà de l'apprentissage, il doit aussi déchiffrer la forme écrite de cette langue, différente de celle qu'il acquiert en premier.

On pourrait croire que les avancées de ces dernières décennies dans le domaine de l'informatique ont facilité le stockage de la connaissance sous forme vidéo, avec notamment des encodages numérique donnant des fichiers plus compacts, et de vastes espaces de stockage disponible à prix réduits.

1-2-4. Place de la vidéo

Mais ce n'est que partiellement vrai : jamais auparavant l'information vidéo n'a été aussi accessible que maintenant (ex : Divx en Peer-to-Peer, Youtube...), et malgré tous ses avantages, la forme vidéo dans son état actuel est mal adaptée au transfert d'information par les sourds, avec en particuliers des problèmes d'indexation du contenu *[2005-Snoek-Video_Indexing]*.

Ainsi, à l'inverse d'un livre, il n'est pas possible de feuilleter facilement une vidéo, de se référer à une table des matières, ou comme par le biais de l'hypertexte actuellement, d'être redirigé vers un autre document.

Si des recherches se font dans ces domaines, leurs résultats ne seront accessibles qu'à moyen ou long terme et rien ne laisse supposer que la vidéo détrônera l'écriture comme forme privilégiée d'accès à la connaissance - on peut plutôt imaginer des images complémentaires.

La plupart des travaux s'intéressent d'ailleurs à l'écrit, en cherchant à en améliorer l'organisation notamment avec des hyperliens vers d'autres contenus *[1997-Wilcox-Dynomite]*, le groupement temporel *[1998-Chiu-Dynamic-Grouping]*, pour faciliter le rangement et l'annotation *[1998-Schilit-Active_Reading]* pour faciliter l'interaction avec les livres.

L'intégration multimodale *[2002-Norrie-Web-Integration]* qui en résulte est très prometteuse pour une utilisation pédagogique *[2003-Ward-Taking_Class_Notes],* avec notamment la création d'outils spécifiques *[2004-LaViola-Mathpad2].*

Mais de tels travaux ne sont pas pour l'instant applicables à la LSF, vu le manque d'une forme écrite.

Il est donc légitime de s'y intéresser, pour faciliter le stockage et l'acquisition de la connaissance par les sourds : l'écriture de leur langue naturelle, la LSF.

1-2-5. Écriture de la LSF

Les justifications d'une écriture de la LSF qui ont été présentées dans ce chapitre ne cherchent pas à remplacer des travaux de meilleure qualité menés par des linguistes ou des spécialistes de l'éducation.

Il ne s'agit que d'une présentation des difficultés rencontrées par la simple évocation d'un tel sujet. Mais d'autres difficultés, moins sociolinguistiques et plus techniques, existent.

En effet, si des linguistes se sont déjà intéressé à ce problème, et si des solutions modernes existent, soit elles apportent des contraintes majeures à un support informatique correct, soit elles ne tirent pas parti des possibilités offertes par l'informatique.

Cette thèse d'informatique vise donc à étudier ce problème globalement pour en isoler les sous-problèmes, pour lesquels des solutions sont proposées puis étudiées.

1-3. Contraintes informatiques

1-3-1. Cas de la vidéo

Des contraintes informatiques existent en effet, qui empêchent par exemple dans le cas de la vidéo de conserver pleinement la richesse de la LSF.

La forme qui semble intuitivement la plus adaptée pour garder le message dans son intégralité semble la forme vidéo. Toutefois, une vision monoscopique ne permet pas de positionner finement les objets dans l'espace de signation : une vision stéréoscopique est nécessaire à l'être humain pour apprécier correctement les distances.

Des systèmes de traitement d'image complexes, mettant notamment en jeu des modèles articulaires [2003-Gianni-Pose_reconstruction] permettent juste d'obtenir une approximation de la position spatiale à partir d'une source monoscopique.

De plus, les résolutions employées par les vidéos de dialogues en LSF ne permettent pas d'appréhender la direction précise du regard, la pupille étant le plus souvent réduite à quelques pixels. L'augmentation de la résolution pourrait être une solution, mais la gestion informatique de résolutions de plus en plus grandes est problématique *[2005-Watson-Pixels]*.

Le problème majeur de l'utilisation de la vidéo est celui des difficultés d'utilisation de cette dernière : les annotations, index et autres défilements rapides, quoique devenant techniquement possibles, restent peu accessibles et complexes *[2005-Snoek-Video_Indexing]*.

Les fichiers générés sont aussi d'une taille malgré tout extrêmement grande : traditionnellement, selon l'encodage et la résolution, on admet qu'un giga-octet permet de contenir environ deux heures de vidéo.

Même si « une image vaut mille mots », c'est l'équivalent de milliers d'images, et d'encore plus de textes, qui sont donc des formes beaucoup plus compactées d'information : ce même giga-octet permet de stocker des encyclopédies...

1-3-2. Cas des images

Les images ne sont toutefois guère mieux gérées que la vidéo par les systèmes informatiques actuels. Leur premier problème est celui de la résolution : la plupart des images, et la totalité des images acquises sont de résolution finie car sous forme non vectorielle.

Les possibilités d'agrandissement sont donc limitées, ce qui en contraint la manipulation.

De plus, les images sont stockées sous forme informatique de fichiers : la description est au mieux en méta-information (ex : entêtes EXIF, base de données..) ou sinon réduite aux quelques mots qui forment le nom du fichier.

La recherche d'image, par les caractéristiques des objets représentés (ex : rechercher les couchers de soleil, les visages) est à peine sortie du domaine de la recherche, et présente encore des difficultés importantes pour les utilisateurs.

Il semble donc que la forme la plus adaptée au stockage et à la manipulation de la connaissance reste actuellement la forme textuelle - certainement du fait que l'informatique a été créée pour manipuler l'information numérique, puis adaptée à l'information textuelle.

1-3-3. Cas des textes en français

Bien que l'informatique moderne soit relativement bien adaptée à la gestion de l'information textuelle, il suffit de vouloir ouvrir un fichier texte (.txt) encodé sous la norme ISO-8859-15 par Solaris ou Linux sous MacOS-X qui s'attend à de l'UTF-8 ou sous Windows qui utilise encore un autre encodage, le « 1252 », héritier du CP-850 de DOS, pour redécouvrir l'importance des contraintes informatiques sur la gestion de la langue.

De même, ne cherchez pas de ligature « oe », aussi appelée « e dans l'o », sur votre clavier AZERTY.

Si vous avez de la chance et si la chaîne de reproduction et d'impression de ce document a correctement fonctionné, vous verrez peut être cette ligature ici : œ.

Cette ligature n'étant pas initialement dans la norme ayant abouti au clavier moderne, elle s'est vu tout simplement supprimée par l'informatique, même si elle était présente sur les claviers des vieilles machines à écrire et encore enseignée à l'école.

Pas d'existence sur le clavier, pas d'utilisation, donc disparition programmée une fois que les logiciels de correction orthographique des traitements de texte cesseront de la substituer aux deux lettres séparées.

Ce n'est d'ailleurs que grâce à eux que les guillemets français « », différents des guillemets anglais " ", persistent...

Qui a donc dit que les contraintes de l'informatique n'influençaient pas les langues ?

1-3-4. Méthodologie choisie : encodage de caractères

L'étude du problème, qui va de la vidéo au domaine textuel, a été réalisée sous le prisme de l'encodage de caractères.

Ce choix pourrait sembler osé.

Toutefois, il est à mon avis totalement logique de traiter l'informatisation d'un nouveau formalisme d'écriture, c'est à dire de petits dessins manuscrits, comme un problème d'encodage.

Cet avis personnel s'objectivera par la lecture de l'état de l'art, qui constitue la majeure partie de cette thèse et dont la connaissance influence grandement la compréhension du problème et les solutions envisageables.

Cette thèse cherche avant tout à rassembler le plus possible d'éléments permettant de comprendre correctement le problème pour faire le point sur le problème de l'écriture de la langue des signes, puisque le plus complexe est beaucoup plus souvent la compréhension que la réalisation.

Toutefois, au delà de la compréhension, des mises en œuvre pratiques ont été réalisées, afin de tester la validité des hypothèses qui ont pu émerger de cet état de l'art, dans le but de soutenir le postulat qu'il est possible d'adapter les outils informatiques à une langue aussi riche que la LSF.

Pour cela, l'écriture a été étudiée, y compris dans les langues vocales.

1-3-5. Projets d'écriture

La naissance de l'écriture dans les sociétés humaines a été longue et laborieuse, et sera détaillée pour faire le point entre les différentes stratégies d'écriture, et mieux comprendre comment elles sont utilisées pour les LS.

Pour ces dernières, l'écriture a souvent été limitée à des dessins des gestes, avec des flèches pour indiquer la direction des mouvements. En cela, il ne s'agissait que d'images et non d'écriture.

L'historique de l'écriture des langues des signes est détaillé peu après dans l'état de l'art, mais sans entrer dans trop de détail, on peut mentionner que pour la plupart de ces systèmes, il ne s'agit pas d'une écriture stricto-sensu mais d'une description de certaines des propriétés des signes réalisés.

De plus, seuls les signes lexicaux ont été étudiés, ainsi dans la notation créée par William Stokoe *[1960-Stokoe-Structure]*, l'aspect facial n'est pas pris en compte.

Il ne s'agit pas d'une écriture, mais d'une transcription incomplète.

Des systèmes ultérieurs comme HamNoSys *[1987-Prillwitz-HamNoSys]* ont été créés afin de parfaire cette fonctionnalité de transcription.

Le seul système moderne abouti qui se soit donné comme vocation d'être un système de communication par écrit (une écriture) est le système SignWriting, élaboré à partir de 1974 par Valérie Sutton.

À titre d'exemple, opposons à SignWriting, qui utilise des symboles regroupés en signes pour résumer en 2D des gestes 3D+T le système SEA *[2004-Herrero-SEA]*, qui utilise l'alphabet romain en changeant la signification des lettres et est en cela peu innovant ou complexe à supporter d'un point de vue informatique.

1-3-6. Adapter l'outil à l'usage, ou l'usage à l'outil ?

Comme rappelé dans le cas du français avec la ligature « e dans l'o » et avec les guillemets français, l'usage doit parfois s'adapter à l'outil.

Devrait-on faire de même avec la LSF, et suivre l'exemple de SEA qui cherche à suivre les contraintes technologiques plutôt que de les dépasser ?

Il me semble dommageable de suivre une telle approche, ne serait-ce que d'un point de vue philosophique *[2003-Tayon-Technosceptique]*, ou dans un souci de conception esthétique *[2000-Hummels-Design].*

L'informatisation d'une forme écrite de la LSF a donc un objectif tout particulier, puisqu'il s'agit de donner aux sourds une forme qu'ils peuvent utiliser, sans pour autant brider la richesse de leur langue par les contraintes informatiques habituelles.

Une fois ces bases conceptuelles posées, il est temps de s'intéresser à l'état de l'art des écritures, de leur encodage, et de la gestion informatique de ces encodages.

2. État de l'art

L'état de l'art se divise en 3 parties : formalismes graphiques, écriture de la langue des signes, informatisation de l'écriture.

D'emblée, certaines limitations doivent être posées sur l'importance relative de chaque partie par rapport à l'objet de cette thèse.

Cet état de l'art a pour but de préciser les différences d'un point de vue informatique, et non linguistique.

Si certains concepts linguistiques doivent être introduits et définis pour être ensuite étudiés d'un point de vue informatique, cette thèse ne saurait prétendre donner un état de l'art exhaustif d'un point de vue linguistique - ce n'est pas l'objet.

La présentation linguistique des formalismes d'écriture et de l'écriture de la langue des signes se bornera donc à la présentation de concepts de base, nécessaire pour suivre les travaux ensuite proposés. Le lecteur sera redirigé vers des travaux de linguistique pour approfondir au besoin certains des concepts présentés.

Ensuite, la plus grande partie de cet état de l'art sera consacrée à l'aspect informatique des systèmes d'écriture, en réalisant une étude historique des systèmes d'encodage pour mieux comprendre le fonctionnement des encodages actuels, puis en effectuant une analyse du fonctionnement des systèmes d'exploitation modernes pour mieux situer les places respectives des différents processus.

2-1. Formalismes d'écriture existants

2-1-1. Domaines concernés

Il faut premièrement définir les domaines concernés par cet état de l'art des formalismes d'écriture existants.

Les langues parlées naturelles vont être brièvement étudiées à part, étant donné qu'il s'agit d'un sujet déjà relativement bien connu et particulièrement bien étudié par les linguistes.

2-1-2. Chronologie de l'écriture

Il est difficile de dater l'apparition des langues humaines - on considère généralement que les premières langues sont nées il y a environ 100 000 ans.

La datation de l'apparition de l'écriture est rendue plus aisée par l'existence de traces précises (mais des traces peuvent également avoir disparu), et actuellement il est accepté que la première écriture est née il y a 5 000 ans en Mésopotamie, à Sumer (Irak actuel).

Brigitte Garcia *[2003-Garcia-Notation_Transcription_LS]* rappelle la chronologie suivante relative aux premières écritures:

3300 avant JC : écriture sumérienne (Sumer, en Mésopotamie)

3200 avant JC : hiéroglyphes égyptiens. Système qui associe des procédés logo-idéographiques (c'est-à-dire en rapport avec les signifiés de la langue) et phonétiques

2000 avant JC : traces d'écriture en Amérique centrale

1400 avant JC : premières traces de l'écriture chinoise (idéographique surtout)

1400 avant JC : premier alphabet, en Syrie du Nord

1300 avant JC : alphabet phénicien, l'ancêtre de nos alphabets (ne note que les consonnes), au sud-ouest du Sinaï, c'est-à-dire au nord-est de l'Égypte et à l'ouest d'Israël

800 avant JC : alphabet grec, caractérisé par l'invention de la notation des voyelles

400 avant JC : alphabet latin (celui qui sert aujourd'hui pour le français écrit et la plupart des écritures alphabétiques du monde), en Italie

500 après JC : premières inscriptions arabes

600 après JC : codification de l'écriture arabe (écriture alphabétique).

Grâce à cette chronologie, on peut comparer l'évolution de l'écriture des autres langues à celle de la langue des signes précédemment donnée.

Une conclusion peut en être tirée : l'évolution des formes écrites des langues vocales a été un processus très long, sur plusieurs millénaires, et différents formalismes d'écriture ont été explorés selon les divers types de langues vocales.

2-1-3. Définition de l'écriture

Écrit et oral correspondent à des fonctions, à des situations de communication différentes et ont de ce fait de fortes spécificités. La transcription est la représentation la plus fidèle possible d'un énoncé oral préalable n'est pas une écriture *[2003-Garcia-Notation_Transcription_LS]*.

L'écriture, elle, permet de transmettre directement du sens, sans passer par une production orale.

2-1-4. Écritures fondées sur le principe phonographique dominant (notation de signifiant)

Il s'agit des écritures les plus répandues. Elles utilisent un ensemble de lettres relativement réduit, appelé alphabet.

Le principe est qu'à un symbole graphique (lettre) correspond un son (écritures alphabétiques et écritures consonantiques) ou une syllabe (écritures syllabiques), mais du fait de l'histoire et de l'évolution, le principe s'est largement complexifié (lettres dites « muettes », multi-fonctionnalité des lettres).

De manière générale, la combinaison de plusieurs lettres donne un son - comme l'exemple B.A « BA » appris aux enfants : les lettres se combinent en mots, qui eux même se combinent en phrases, mais chaque élément peut être redécoupé en sons.

2-1-5. Écritures fondées sur un principe logographique dominant (notation de signifié)

Les items de base ne sont plus des lettres mais des logogrammes, qui pris isolément conservent un sens : chaque logogramme correspond à une idée.

Ainsi, prenons l'exemple des logogrammes lune et soleil en japonais : pour l'écriture des dates, il est possible d'utiliser soit le calendrier grégorien, soit un calendrier basé sur des ères.

Pour écrire des dates selon le « calendrier occidental », on indique le type de calendrier (ici SeiReki), puis le nombre d'année en chiffres arabes, qui sera suivi du logogramme lunaire, lui même suivi d'un chiffre pour signifier le nombre de mois puis du logogramme solaire et d'un autre chiffre pour signifier le nombre de jours.

Le lien est le nombre de rotations autour ou par rapport à l'astre céleste considéré.

Figure 4: Lune

Figure 5: Soleil

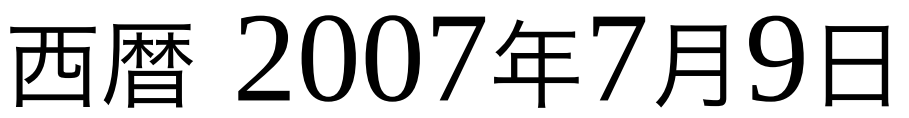


Figure 6: Le 9 juillet 2007

Les logogrammes peuvent être découpés *[2000-Babcock-CHA]* en « squelettes » et l'on peut alors identifier un nombre limité d'idéogrammes qui ne peuvent être davantage découpés. Ces éléments « premiers » des idéogrammes en sont appelés les « racines ». On peut ainsi définir des regroupements.

2-1-6. Écritures hiéroglyphiques anciennes

Une différence supplémentaire est à noter dans le cas des écritures dites hiéroglyphiques, qui contiennent trois types de symboles.

Outre des idéogrammes et les phonogrammes, elles contiennent des déterminatifs, qui fournissent des informations supplémentaires nécessaires pour interpréter correctement les signes alentours.

Les déterminatifs déterminent la sémantique.

La particularité calligraphique principale est l'utilisation de quadrats : le sens ne vient donc pas de la combinaison de squelettes de sous dessins racines dans un nouveau dessin, mais de l'arrangement spatial de ces dessins entr'eux et des déterminatifs.

Cet arrangement est plus complexe qu'un simple flux unidimensionnel, comme utilisé dans les langues idéographiques et alphabétiques.

Pour mieux illustrer l'organisation de cette écriture, prenons l'exemple de « Isis » et de « trône »: seul le déterminatif du quadrat inférieur droit diffère. Dans le premier cas, il s'agit d' « œuf » 𓐍, et dans le second, de « maison » 𓉐.

Figure 7: Isis et Trône

Ce sont les mêmes symboles graphiques qui pouvaient, selon les contextes, renvoyer tantôt à un son, tantôt à un signifié.

Les hiéroglyphes égyptiens sont les plus connus, mais d'autres exemples existent : hiéroglyphes hittites, mayas, et de la vallée de l'Indus.

2-1-7. Lignes d'analyse

Au final, on peut retirer deux grands principes dans les écritures historiques, qui le plus souvent ont été combinés

(i) Relation au signifié

Premièrement, selon le rapport qui existe entre le symbole et ce qu'il représente, on sépare :

- les écritures où le symbole graphique représente le signifié,
- les écritures où le symbole graphique représente le signifiant,

(ii) Organisation spatiale

Enfin, selon l'utilisation qui est faite de l'espace, on sépare les écritures monodimensionnelles (la plupart) des bidimensionnelles (hiéroglyphiques).

Utilisons ces grandes lignes d'analyse pour étudier une langue écrite, par exemple le japonais.

2-1-8. Cas du japonais

Le japonais est un système qui mixe trois sous-systèmes aux fonctionnalités complémentaires :

- kanjis : emprunt aux caractères chinois mais totalement appropriés par le japonais et donc susceptibles de deux lectures différentes,
- kana : écriture syllabique (phonographique),
- romaji : latinisation complète (essentiellement réservé aux emprunts).

Selon les deux lignes d'analyse, on peut voir :

(i) Le symbole représente le signifié

Les hiraganas, utilisés pour la grammaire, et les katakanas, utilisés pour les mots étrangers, ont été mis en correspondance avec un alphabet adapté de l'alphabet latin, appelé « romaji », et utilisé exclusivement pour transcrire les noms empruntés aux langues étrangères

Il n'est pas strictement phonétique, car certaines oppositions sont neutralisées (consonne R = L), mais ce système peut se voir remplacé sans difficulté par un autre alphabet, comme par exemple l'alphabet latin.

Cette pratique du romaji peut être illustrée par la présentation des tables de correspondance utilisées à cet effet.

Figure 8: Alphabet : hiragana en haut à droite, katakanas en bas, romaji à gauche

Sur une même touche d'un clavier standard coexistent le caractère de l'alphabet latin pour taper une séquence romaji, et les caractères les plus fréquents en japonais.

Le passage d'un mode à l'autre se fait par des touches dédiées, comme ici entre les touches Alt et espace.

La saisie des mots japonais se fait d'abord dans un système alphabétique choisi par l'utilisateur ; une fois le mot entier entré, il peut être converti en kanji en appuyant sur la touche espace, à plusieurs reprises, jusqu'à ce que la proposition soit satisfaisante pour l'utilisateur *[2003-Aznar-C700]*.

Figure 9: Un clavier japonais

(ii) Le symbole représente le signifiant

Les kanjis viennent compléter les systèmes précédemment présentés : ils ont une représentation sémantique identique au chinois mais avec un sens et une prononciation propre au japonais.

Ils correspondent à la partie logographique, comme vu précédemment avec l'exemple de la lune et du soleil dans la date.

Les kanjis ont évolué, et évoluent encore : *[2001-Herledan-Kanjis_Histoire]* nous apprend :

Avant la seconde Guerre mondiale, les kanji utilisés en japonais n'avaient jamais été standardisés. Il fallait connaître un minimum de 4 000 de ces caractères pour comprendre un quelconque journal ou magazine.

Durant l'occupation du Japon, le ministre japonais de l'éducation s'attela à la lourde tâche de simplifier la langue écrite. Le but était de réduire l'usage des caractères chinois à un strict minimum, mais en nombre suffisant pour pouvoir lire et écrire des textes normaux dans la vie pratique japonaise.

Le principe retenu fut de miser sur la fréquence d'utilisation des kanji pour ne choisir que les plus fréquents. Les résultats de ces travaux ont abouti en 1946 à une liste de 1850 caractères que l'on appela les tooyoo kanji.

Cette liste fut complétée par une autre liste de 284 kanji qui servaient à l'écriture des noms et des prénoms. Les 881 premiers caractères (parmi la liste totale) ont été appelés les kyooiku kanji. Ce sont les kanji destinés à l'usage éducatif et utilisés dans les écoles ; leur apprentissage est obligatoire après six ans d'études primaires.

Des études ont montré que la connaissance des kyooiku kanji permettait de comprendre 90% des documents les plus courants, alors que la connaissance des tooyoo kanji permettait d'en comprendre près de 99%.

En 1977, les kyooiku kanji passèrent de 881 à 996 caractères.

De même, en 1981, la liste des tooyo kanji, a, elle aussi, été étendue à 1945 caractères. Cette nouvelle liste s'appelle les jooyoo kanji ou kanji d'usage courant que l'on utilise aujourd'hui.

Enfin, en 1992, les autorités japonaises officialisèrent et publièrent une nouvelle liste appelée gakushuu kanji.

Celle-ci demeure toujours en vigueur dans les écoles et compte 1006 caractères, lesquels incluent les 966 issus des kyooiku kanji, sans oublier les chiffres courants au Japon: Ichi 1 Ni 2 San 3 Shi 4 Go 5 Roku 6 Shichi 7 Hachi 8 Kyu 9 Ju 10.

一 二 三 四 五
六 七 八 九 十

Figure 10: Chiffres japonais (réservés aux documents calligraphiés)

Au-delà de l'aspect historique et culturel, cet exemple nous permet surtout de constater que les langues idéographiques évoluent, tout comme les langues alphabétiques, et que même si elles ont un répertoire plus vaste que le répertoire de lettres des langues alphabétique, elles n'utilisent pas à une fréquence identique tous les idéogrammes.

Ces deux conclusions sont à garder à l'esprit sur ce sujet d'informatisation d'une forme écrite de la langue des signes, qui comme il sera vu ci-après, se rapproche de ces caractéristiques d'évolutivité, de large répertoire, et de fréquences variables.

Une fois le vocabulaire de base de l'écriture introduit, et les concepts mis en évidence sur l'exemple du japonais, on peut étudier les formalismes d'écriture utilisés pour les LS comme la LSF.

2-2. Formalismes d'écriture existant pour les langues des signes

2-2-1. Introduction

Les langues sans écriture étaient considérées jusqu'au 20e siècle par les linguistes comme incomplètes, considération qui servait aussi à justifier l'idéologie de dominance (minorités régionales, sourds) et de colonisation (langues africaines).

Au delà de ces débats sur la place qu'il faille attribuer ou non à l'écrit, étudions les systèmes principaux ayant existé pour l'écriture de la langue des signes et la motivation de leurs créateurs.

La motivation est étudiée car il est important de remarquer qu'aucun système de notation n'a été institutionnalisé ou ne s'est imposé par l'utilisation qui aurait pu en être faite par les sourds.

Vu que la plupart sont des systèmes créés par des linguistes, le plus souvent entendants et non sourds, on peut aussi s'interroger sur l'adaptation de l'outil créé aux besoins réels et pratiques des utilisateurs *[2004-PrélazGirod-Accès_Écrit], [2001-Flood-Learning_to_write_in_SW],*

Cette étude ne se veut donc pas exhaustive, étant donné qu'il existe par ailleurs d'autres états de l'art réalisés dans des buts plutôt linguistiques *[2003-Boutet-Formalisation_Graphique_LSF],[2003-Garcia-Notation_Transcription].*

Ainsi, certains systèmes comme le Berkeley Transcription System *[2001-Hoiting-BTS_For_Understanding]* ou les formalismes plurilinéaires destinés à la recherche linguistique *[2002-Cuxac-LSCOLIN]* et non à l'utilisation quotidienne par les sourds, ne seront pas étudiés.

Afin de présenter la problématique, seuls seront retenus des exemples illustrant au mieux les différences précédemment évoquées, en se focalisant sur leur dimension informatique et la possibilité d'une vaste diffusion.

D'emblée, il convient de remarquer que la quasi totalité de ces systèmes est monolinéaire, c'est à dire que les symboles se succèdent sur une ligne, de gauche à droite. Ainsi, ils sont très proches des formalismes d'écriture des langues vocales, et non originaux.

2-2-2. Dactylologie : signation de l'écrit

La dactylologie ne saurait être considérée comme une écriture de la LSF : elle n'est que la réalisation signée de lettres de l'alphabet latin, à même seulement de permettre la citation de mots écrits des langues vocales

Il existe d'ailleurs plusieurs alphabets dactylologiques, selon les langues des signes, et selon l'alphabet d'origine de la langue vocale du pays.

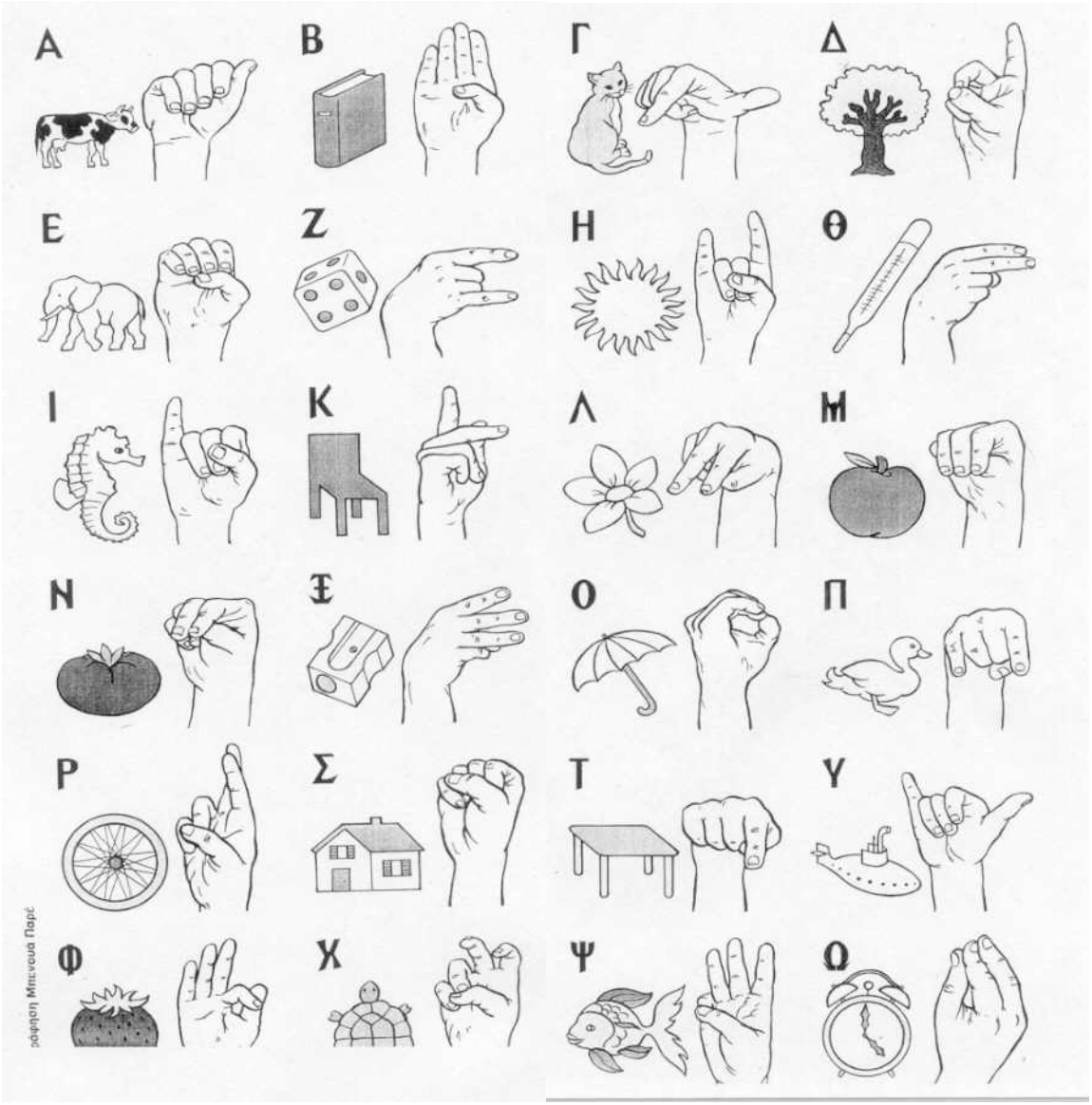


Figure 11: Dactylologie grecque

On voit bien dans cet exemple que seules les lettres de l'alphabet source ont une correspondance dactylogique, les autres (ex : W, Z, J...) bien qu'elles puissent être utilisées par une autre langue utilisant un alphabet différent, n'ont pas de dactylologie.

Dans le cas du français, il s'agit de même d'une signation du français écrit, c'est à dire de l'inverse d'une forme écrite de la LSF, puisque le but n'est pas de produire ainsi des messages en LSF, mais plutôt de pouvoir citer ou nommer des mots du français.

2-2-3. Mimographie de Bébian : mémorisation

Bébian ne prétendait pas noter autre chose que des signes isolés, c'est à dire hors contexte : *« il s'agit d'abord bien moins d'écrire un long discours mimique, que d'établir une sorte de vocabulaire, et de fixer le sens de chaque mot isolément ».*

Le titre, *« Mimographie ou essai d'écriture mimique, propre à régulariser le langage des sourds-muets »,* résume bien l'objectif de ce formalisme : la mémorisation du vocabulaire de la langue des signes (« régularisation », pour Bébian, signifiant « enregistrement »).

4 jeux de symboles sont utilisés, pour indiquer respectivement le mouvement, la conformation de la main, les différentes parties du corps, et des points physionomiques, c'est à dire principalement les expressions du visage.

2-2-4. Stokoe : démonstration du caractère linguistique

W. Stokoe, professeur de littérature médiévale, travaille à l'Université Gallaudet vers la fin de la décennie 1950. À l'époque, les LS n'étaient pas considérées comme des langues, mais comme une pantomime agençant des touts indécomposables.

Le principal critère de définition d'alors pour une langue était la double articulation : Stokoe cherche donc à distinguer deux unités, équivalentes aux monèmes et aux phonèmes.

Il extrait trois paramètres *[1960-Stokoe-Structure]*: le designator (configuration des mains), le tabula (l'emplacement de signation), le signatum (mouvement effectué par les mains). Pour chaque paramètre, il définit un nombre limité d'unités possibles qu'il nomme chérèmes, et établit la double articulation par le lien qui existe entre un sens et une combinaison de chérèmes de ces trois sous ensembles.

La notation s'inspire fortement des travaux de Bébian, tout en cherchant une certaine abstraction : le but n'est plus d'être fidèle à la signation, mais plutôt d'offrir un nombre minimal d'éléments permettant de distinguer un signe des autres signes ce qui réalise ainsi une *abstraction des caractères physiques perçus pour ne retenir que les traits fonctionnellement distinctifs [2003-Garcia-Notation_Transcription_LS]*.

La notation utilise l'ordre tabula/designator/signatum le plus fréquemment, bien que quelques variations soient possibles (cas des gestes à deux mains).

Ultérieurement, Battison rajoutera en 1973 un autre paramètre : l'orientation, mais le problème principal du système de notation Stokoe est l'impossibilité d'utiliser son système pour des discours, c'est à dire d'aller au-delà de la notation des signes isolés.

De nombreuses variantes du système de Stokoe ont été créées par les linguistes de différentes LS du monde, très différentes les unes des autres, ce qui gêne l'échange de données. *[2001-Crasborn-Signphon]*

De plus, le système est peu facile à utiliser pour l'échange d'information : il est principalement utile aux linguistes, et non aux sourds.

Transcription : $\varnothing$ $F^{x}F$ $\substack{\div \\ \omega}$

Chérème	*Symbole*	*Signification*
Tabulateur (TAB)	$\varnothing$	Espace neutre devant le torse
Désignateur (DEZ)	$F^{x}F$	Configuration manuelle F pour les deux mains initialement en contact
Mouvement (SIG)	$\substack{\div \\ \omega}$	Séparation (÷) des mains et rotation simultanée du poignet (ω)

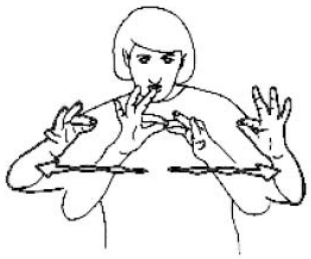


Figure 12: Exemple de notation avec le système de Stokoe du signe « histoire », issu de *[2000-Losson-Signeur-Virtuel]*

2-2-5. HamNoSys : transcription

HamNoSys *[1987-Prillwitz-HamNoSys]* ne prétend pas non plus être une écriture stricto-sensus, mais un système de transcription. Créé à la fin de la décennie 1980 à Hambourg, en Allemagne, ce système vise comme celui de Stokoe à noter des signes, et non à écrire ou communiquer.

Toutefois, il diffère par deux points important : tout d'abord par sa prétention à l'universalité, se donnant objectif d'être utilisable pour toutes les LS là où Stokoe ne visait qu'à noter l'ASL, et ensuite par le choix d'utiliser des symboles iconiques de la forme signifiante.

En conséquence, les symboles graphiques sont donc très iconiques, pour faciliter l'apprentissage du système et sa notation. Il compte actuellement 200 symboles permettant de représenter les configurations, les emplacements, le mouvement et l'orientation des mains, mais reste monolinéaire, et utilise les 4 paramètres traditionnels de Stokoe.

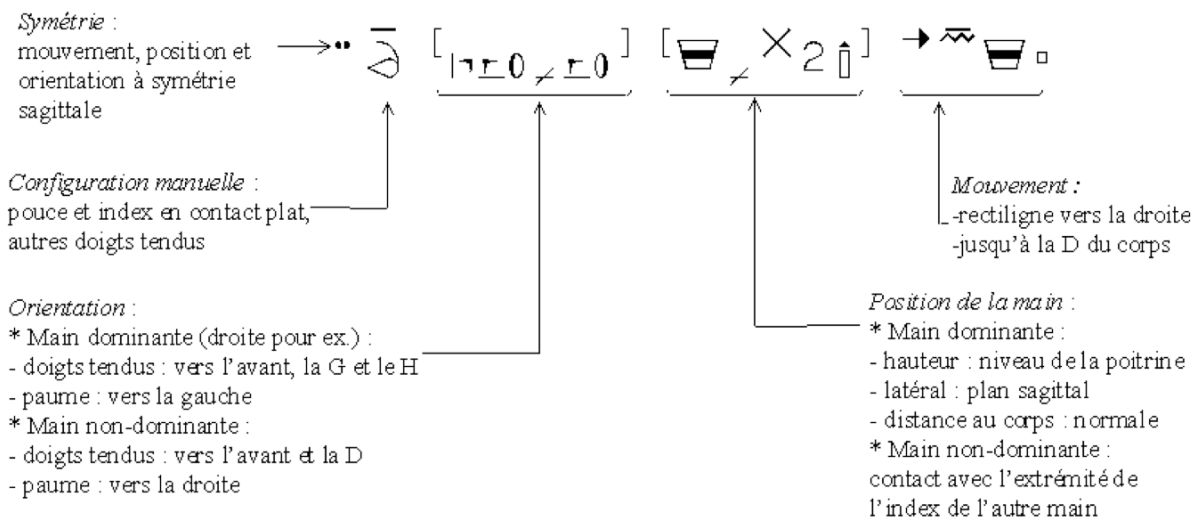


Figure 13: Exemple de transcription HamNoSys du signe « Histoire », issu de *[2000-Losson-Signeur-Virtuel]*

2-2-6. SignWriting : enseignement

Conçu par Valérie Sutton à partir de 1974 pour noter les pas des ballets, SignWriting (abrégé SW) a progressivement évolué vers un système destiné principalement à noter les LS, notamment dans le cadre de l'enseignement, pour l'échange d'information ou encore la prise de note.

Il s'agit là de la grande particularité du système, qui n'a pas été inventé uniquement pour la recherche (Stokoe, HamNoSys), mais pour la diffusion du savoir *[1995-Sutton-SignWriter]*.

C'est peut être une des raisons du succès de ce système, qui est enseigné et utilisé dans plusieurs écoles de divers pays à travers le monde, et que les sourds se sont approprié, notamment à travers des journaux en SW.

Pour les linguistes, SW est alphabétique, mais du fait de l'icônicité des LS et de l'aspect visuel de SW, il s'agit d'un système dont les exemples sont plus parlants : la ressemblance n'est pas un fait objectif, mais le résultat d'un processus cognitif et l'iconicité dépend du contexte culturel *[2001-Taub-Iconicity]*.

Ceci est particulièrement bien illustré par l'exemple de « joyeuse hanukkah » en ASL, où les mains prennent la forme des branches de bougies.

Figure 14: Exemple : Joyeuse hanukkah en SignWriting

Cet exemple très parlant ne doit toutefois pas faire croire que SW n'est pas alphabétique. Il s'agit surtout d'une excellente du processus cognitif sous-jacent.

(i) Utilisation isolée

SW peut aussi être utilisé de manière isolée, par exemple pour la dactylologie : il permet donc d'inclure dans un document en langue des signes des passages d'une langue alphabétique.

Le rendu visuel des lettres n'est par contre pas conservé dans cette opération, à l'inverse du japonais qui avec le romaji peut conserver les lettres latines.

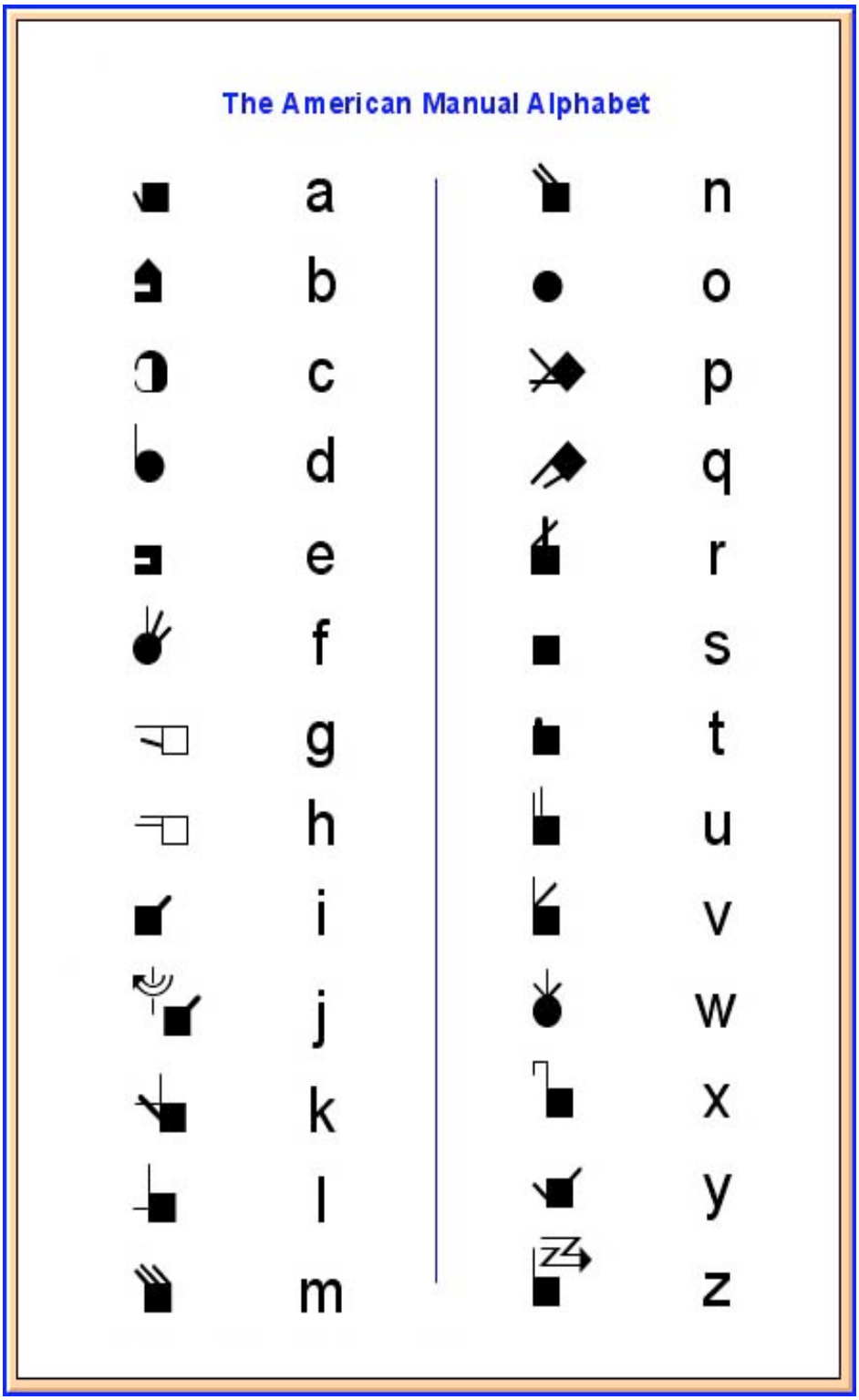


Figure 15: Dactylologie de l'ASL en SignWriting

Il convient alors de signaler la richesse de ces symboles, et la très grande combinatoire possible, puis de fixer immédiatement une définition claire du vocabulaire employé .

On définit ainsi les termes suivants :

- signe : production gestuelle en LS

- signe SW : écriture en SW, pouvant être incorrectement appelé « signe »

- symbole : plus petit élément de base pour composer ce signe, constitué par une association de plusieurs paramètres

- paramètre : propriété isolée d'un symbole de base, pouvant être commune à plusieurs symboles de base très différents, comme par exemple le remplissage

- symbole de base : nom de l'un de des paramètres de classification

- caractère : dénomination informatique d'un élément graphique

- glyphe : représentation de la forme graphique précise de l'élément graphique : un symbole de base est le glyphe d'un symbole selon des paramètres et des attributs de police de caractère (taille, gras...)

La différence caractère/glyphe sera détaillée ultérieurement.

(ii) Vaste choix de symboles

Les symboles composant les signes SW sont définis par la norme SSS (Sign Symbol Sequence) de l'IMWA (International Movement Writing Alphabet) *[2004-Sutton-SSS]*.

Ils sont particulièrement nombreux : une analyse de l'archive SymbolBank 2004 relève 25 973 symboles, subdivisés par 6 paramètres : 8 catégories, 10 groupes de 50 symboles de base, 5 variations, 6 remplissages et 16 rotations.

On différencie ainsi 425 symboles de base, tous les groupes n'ayant pas 50 symboles de base.

Les catégories correspondent aux différents types de gestes (mouvement, segment du corps, positionnement) et les groupes aux segments précis du corps, les symboles aux attitudes de ces segments.

Les variations, remplissages et rotations permettent de représenter finement les positionnements tridimensionnels des attitudes de ces segments.

Ainsi, lles remplissages différents (selon des motifs de noir et de blanc) servent à indiquer la facette exposée, et les 16 orientations spatiales servent à spécifier la chiralité (droite ou gauche) ainsi que l'orientation dans un plan.

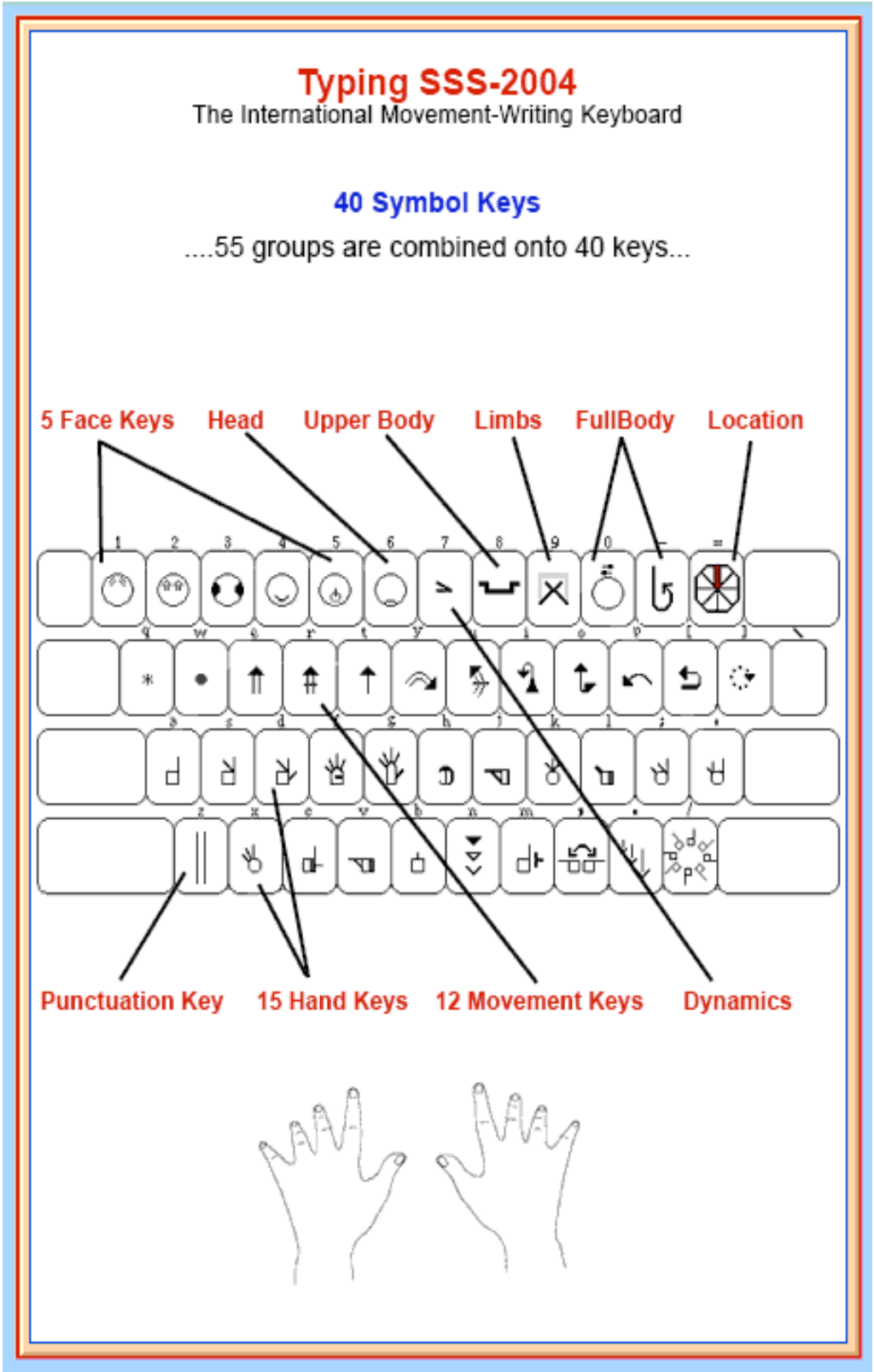


Figure 16: Norme SSS rangée par groupes sur un clavier

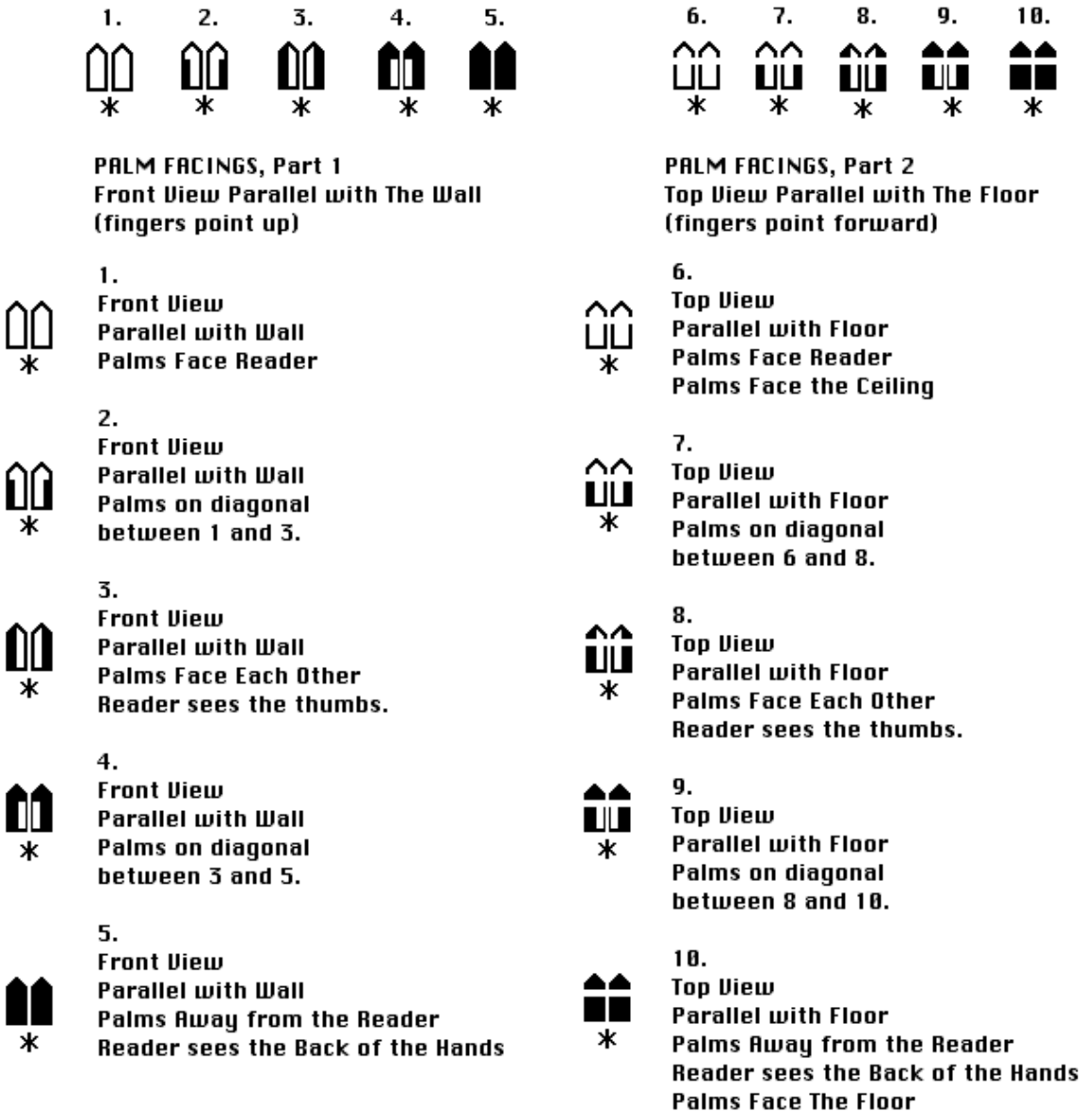


Figure 17: Remplissage pour indiquer les points de vue sur les mains

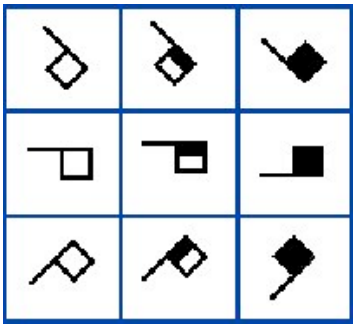


Figure 18: 6 symboles : différentes rotations et remplissages du symbole de base index tendu

Les éléments de base, appelés symboles de base, servent à définir des symboles, qui sont associés pour recréer des gestes, par le biais des signes SW.

Lors de la création de signes SW, une autre particularité importante est à noter d'un point de vue informatique : l'utilisation de la bidimensionnalité.

(iii) Spatialisation analogique

En effet, les symboles sont positionnés au sein d'un espace bidimensionnel nommé cellule qui contient le signe SW. Les cellules se succèdent, de gauche à droite ou de haut en bas selon les habitudes.

Ce positionnement bidimensionnel des symboles sert à représenter l'aspect visuel de la scène, notamment par le positionnement respectif des segments corporels les uns par rapport aux autres, ainsi que leurs mouvements et leurs interactions.

Prenons un signe SW et décomposons-le en ses symboles.

Figure 19: Signe SW

L'ordre de décomposition en symboles ici adopté est arbitraire, et uniquement choisi à titre d'exemple.

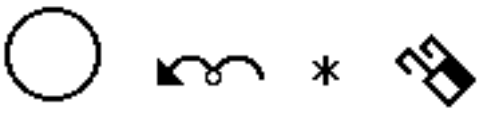


Figure 20: Décomposition de ce signe SW en symboles

On voit dans cette décomposition les symboles « figure », « double translation circulaire », « réaliser un contact », et « main sur la tranche, deux doigts crochus ». Le symbole de base « translation » a par exemple été modifé par des paramètres pour réaliser un symbole plus précis : « double translation circulaire ».

La disposition spatiale des symboles est importante : cette spatialisation est dite analogique, dans la mesure où le positionnement n'est pas relatif, c'est à dire uniquement d'un symbole par rapport à un seul autre, mais absolu, c'est à dire d'un symbole par rapport à un ou plusieurs autres, sur un repère fixe et commun à tous.

(iv) Aspect combinatoire

Un texte en SW est donc un ensemble complexe de symboles regroupés et disposés spatialement en signes SW, eux mêmes disposés spatialement suivant une directionalité et une progression de bloc donné, que l'on peut comparer au « sens de lecture ».

Il y a donc un grand choix de signes SW qu'il est possible d'utiliser.

La conséquence est la complexité de l'informatisation : plusieurs dizaines de milliers de symboles uniques sont utilisables à partir des centaines de symboles de base, et ce dans un infinité de combinaisons liées au choix de leur groupement et de leur positionnement pour constituer des signes SW, eux mêmes agencés spatialement d'une certaine manière.

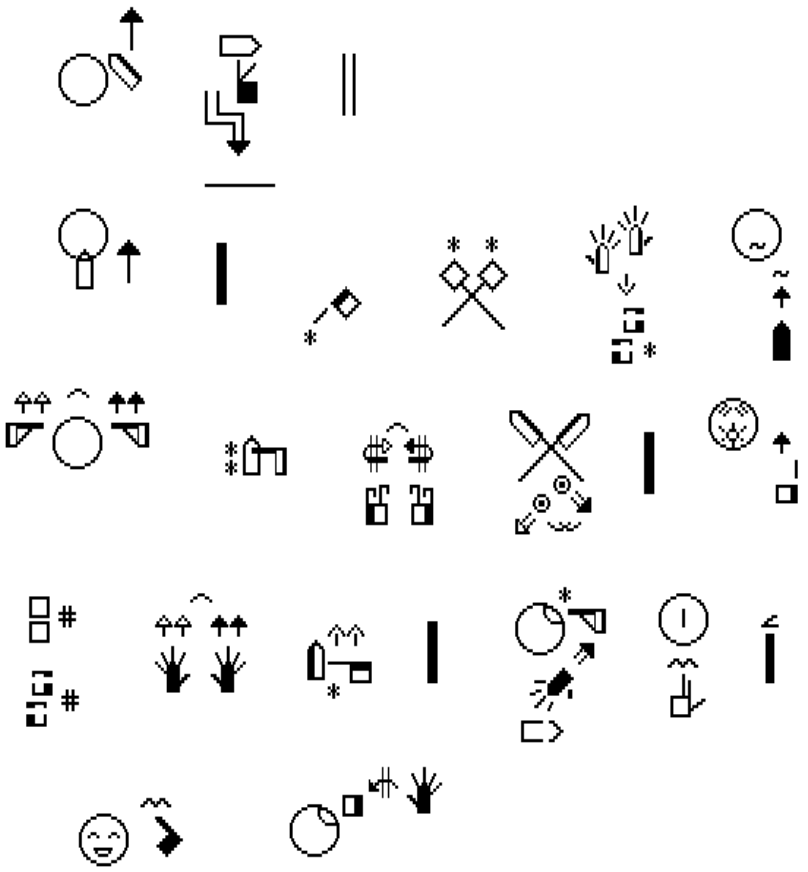


Figure 21: Remerciements

Comment donc lire cet exemple plus complexe ?

(v) Limitations du système SW

On voit donc plusieurs limitations au système SW.

Tout d'abord, depuis 1994, certains sourds ont utilisé SW de haut vers le bas, au lieu de gauche vers la droite, ce qui leur semble plus facile à utiliser : le choix de cette directionalité verticale est devenue prédominante dans les texte longs.

Le mode d'écriture n'est donc pas figé, et varie encore.

De plus, chaque année, le système SSS connaît aussi de nombreux ajouts, ce qui tendrait à montrer qu'il n'est pas fini, mais encore en évolution.

Le nombre de symboles de base et donc de symboles est donc variable selon la version considérée *[2004-Sutton-SSS]*.

De même, il n'a jamais été démontré que certains aspects spécifiques de la langue des signes américaine, relativement proche de la LSF, soient gérables par SW, même si son utilisation dans les écoles a été étudiée *[2001-Flood-Learning_to_write_in_SW]* et même recommandée par certains *[2004-PrélazGirod-Accès_Écrit]* dans le cas de la LS suisse.

2-2-7. Projet LS-Script

Le projet LS-SCRIPT a été lancé pour tout d'abord connaître les pratiques graphiques et les attentes de la communauté des sourds français par rapport à une éventuelle écriture de la LSF *[2007-Garcia-LSSCRIPT]*.

Ce projet, financé par l'ANR dans le cadre du programme RIAM, et dirigé par Brigitte Garcia, de l'Université Paris 8, a comporté de nombreuses études, qui ont permis d'interroger les limites et l'applicabilité du formalisme SW aux LS.

Dans la suite de ce projet, il est possible d'imaginer un futur formalisme améliorant voire dépassant SW, pour notamment gérer l'espace de signation et une grammaire basique, ou d'autres concepts plus fins.

Toutefois, un tel formalisme n'est pas encore finalisé, vu les enjeux importants *[2006-Boutet-Enjeux]*.

Seuls certains aspects ont été décrits, par exemple sur un plan informatique *[2004-Lenseigne-Gesture_representation]*.

Cette thèse a donc retenu SW, qui est présenté en détail et dont l'informatisation sera ensuite étudiée tout au long du document, tout en cherchant aussi à ne pas en dépendre totalement.

En effet, SW est pour l'instant le système de notation des LS le plus répandu et le plus pratiqué.

D'un point de vue informatique, il s'agit aussi du formalisme le plus complexe car il présente des innovations par rapport aux écritures existantes.

En particulier, il faut noter le mode original d'utilisation de l'espace graphique : bidimensionnalité, représentation analogique de l'espace de signation, verticalité...

SW va donc servir à présenter les problèmes et les solutions devant être généralisables à un nouveau formalisme.

Mais les systèmes graphiques à venir supprimeront éventuellement certaines contraintes, et en amèneront possiblement d'autres.

L'étude de ces contraintes informatiques doit donc paradoxalement rester aussi générique et extensible que possible, po

Il est maintenant temps d'étudier les encodages existants, pour comprendre deux éléments importants :

-comment les écritures des langues vocales ont pu être informatisée ?

- quelle est l'influence des contraintes informatiques ?

À travers les réponses à ces questions, il sera possible de tisser des parallères avec les LS, et donc d'éviter d'étudier un encodage trop spécifique à SW.

2-3. Encodages existants

2-3-1. Historique

Certains font commencer l'encodage à l'époque grecque, où des systèmes de torches étaient utilisés pour transmettre des messages *[2007-Haralambous-Encodings],* mais leur véritable essort est liée à la découverte de l'électricité.

(i) Encodages télégraphiques

Les premiers encodages furent rendus nécessaires par la télégraphie, pour transporter le texte de façon la plus compacte et la plus rapide possible.

Le précurseur des communications numériques a été le code inventé par Samuel Morse en 1835, qui assigne à chaque lettre, chiffre et signe de ponctuation une combinaison unique de signaux intermittents produit par un manipulateur, au début monotouche (« pioche »), laissant au soin de l'opérateur de moduler la durée des traits et des points.

On peut remarquer visuellement cette première optimisation : les lettres les plus fréquentes ont les codes les plus courts (e != z)

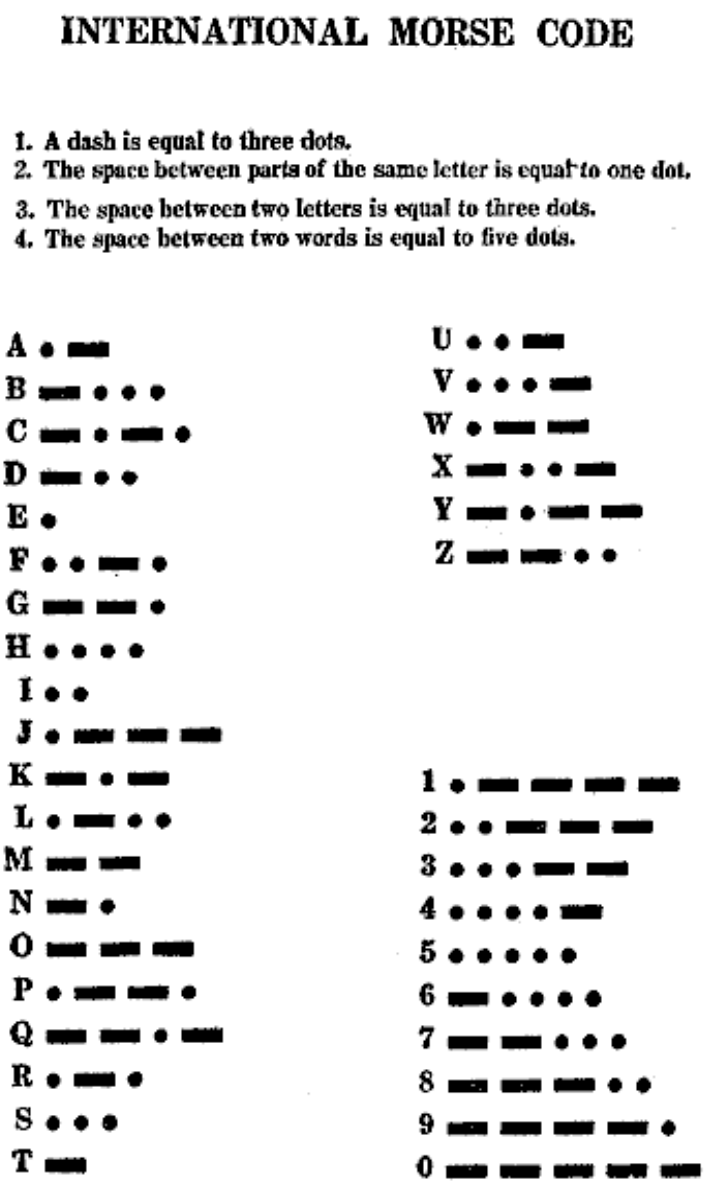


Figure 22: Le Morse

Le Comité Consultatif International Télégraphique et Téléphonique standardisa ensuite plusieurs encodages successifs, dont le premier, le CCITT 1 fut créé par Jean Maurice Émile Baudot en 1870 : les caractères étaient alors composés à l'aide d'un clavier à cinq touches, une par bit de caractère. Baudot laissa son nom dans l'histoire comme unité de mesure de transmission, le baud.

(ii) Jeux de caractères

Le CCITT 1 correspond à une étape importante : le début des jeux de caractères, c'est à dire l'utilisation de modélisations mathématiques pour obtenir le plus de caractères possibles avec le moins de complexité possible.

En effet, seuls 2^5=32 caractères différents pouvaient être obtenus, ce qui n'était pas suffisant pour les 26 lettres + 10 chiffres. 2 caractères spéciaux ont donc été utilisés pour passer d'un « jeu de caractère », c'est à dire d'une signification, à l'autre : lettre ou chiffre. 12 codes restant disponibles dans le jeu de caractère des chiffres, ils furent attribués à d'autres usages : ce fut donc aussi le début de la segmentation des jeux de caractères.

Letters								
	0	1	2	3	4	5	6	7
00	NUL	Y	E	I	A	U	/	O
10	FS	B	G	F	J	C	H	D
20	LS	S	X	W	-	T	Z	V
30	ER	R	M	N	K	Q	L	P

Figures								
	0	1	2	3	4	5	6	7
00	NUL	3	2	℀	1	4	℣	5
10	FS	8	7	℁	6	9	℔	0
20	LS	℣	℔	?	.	℁	:	'
30	ER	-	)	£	(	/	=	+

Figure 23: Le CCITT 1

Le CCITT 1 fut modifié en 1901 par Donald Murray, qui voulait ajouter de nouveaux symboles et surtout utiliser un clavier habituel de machine à écrire . Il en profita pour optimiser le code pour que les caractères les plus fréquents soient codés par des 5-uplets ayant le moins de transitions possibles entre le 1 et le 0, afin de minimiser l'usure du matériel.

Letters								
	0	1	2	3	4	5	6	7
00	NUL	T	CR	O		H	N	M
10	LF	L	R	G	I	P	C	V
20	E	Z	D	B	S	Y	F	X
30	A	W	J	FS	U	Q	K	LS

Figures								
	0	1	2	3	4	5	6	7
00	NUL	5	CR	9		✕	,	.
10	LF	)	4	✕	8	0	:	;
20	3	''	WRU	?	'	6	✕	/
30	-	2	BEL	FS	7	1	(	LS

Figure 24: Le code Murray

(iii) Perte du lien caractère/touche

Dès lors, il n'y eut plus de de lien direct entre le code lui même et la disposition des touches du clavier : il s'agit d'une autre étape historique importante, qui marque la dissociation entre code et dispositif d'entrée.

Il est à noter que le code CCITT 2 datant de 1901 est encore utilisé dans le réseau Télex, qui compte encore de nos jours 400 000 lignes rien qu'en Allemagne.

(iv) Encodage des formalismes d'écriture usuels

Les codes précédemment présentés avaient une utilisation téléphonique. Une modification majeure eut lieu avec les débuts de l'informatique, pour faciliter la gestion numérique des textes sur bandes magnétiques.

Après les systèmes CCITT sur 5 bits, des travaux commencèrent sur des systèmes 6 bits, avec notamment le projet Fieldata. IBM pour sa part créa le codage Binary-Coded-Decimal (BCD) sur 6 bits, offrant donc 2^6=64 caractères, abondamment utilisé par les premiers ordinateurs IBM et servant de base à son évolution EBCDIC.

Mais la plus grande évolution fut l'encodage sur 7 bits proposé par Ivan Idelson (Cluff-Foster-Idelson) en 1956, qui fut ensuite normalisé en "Standard Code for Information Interchange" (SCII) par l'organisme de standardisation américain - d'où le nom ASCII.

	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17
000	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
020	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SS	ESC	FS	GS	RS	US
040		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
060	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
100	‘	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
120	P	Q	R	S	T	U	V	W	X	Y	Z	[	~	]	^	_
140	@	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
160	p	q	r	s	t	u	v	w	x	y	z	{	¬	}	|	DEL

Figure 25: Table ASCII décimale

Ce code ASCII, standardisé en 1963, connut une révision en 1967, mais il n'a plus évolué depuis 1986. Il intégra aux travaux initiaux de Idelson ceux de Bob Berner et de Hugh McGregor Ross, qui ont notamment apportés les séquences d'échappement et la gestion de caractères non affichables.

(v) Parité et caractères non affichables

7 bits étaient utilisés, car les systèmes informatiques de l'époque utilisaient dans la plupart des cas des octets, c'est à dire 8 bits, ce qui permettait le 8e bit pour le contrôle d'erreur (parité).

L'ASCII, en utilisant 7 bits, définit 2^7=128 caractères, dont seulement les 95 caractères situés entre 32 et 126 sont affichables : les autres correspondent à des caractères de contrôle, définissant la manière dont le texte est manipulé.

Ainsi, le 7e caractère, "*bell*", provoquait une sonnerie, le 8e-"*backspace*", un retour en arrière, le 9e-"*tab*", une tabulation horizontale, le 10e-"linefeed", l'avancée du papier dans une imprimante...

L'utilisation de codes pour manipuler non seulement le texte, mais la manière dont il était gérée fut aussi une grande innovation : les normes CCITT précédentes ne permettaient pas une telle richesse de manipulation.

Il s'agit donc d'une étape importante, puisque l'encodage gère désormais non seulement l'encodage du flux de caractères d'un formalisme d'écriture usuel, mais aussi les informations supplémentaires nécessaires pour le reproduire de manière la plus fidèle possible (majuscules, minuscules, retour à la ligne, tabulations...)

(vi) Optimisation mathématique

Un autre point à retenir de la norme ASCII est les optimisations mathématiques particulières pour accélérer les opérations algorithmiques de l'époque.

Signalons par exemple que les caractères en majuscule ne différent des caractère en minuscule que par un seul bit, ce qui simplifiait les conversions logicielles, les vérifications de groupe (pour exclure les caractères n'étant pas des lettres) et de manière générale, les algorithmes devant ignorer la différence entre majuscules et minuscule.

La répartition des caractères dans l'encodage n'a donc pas été choisie au hasard, mais optimisée dans un but mathématique de rapidité algorithmique, là où les optimisations précédentes visaient une réduction de l'usure du matériel.

(vii) Standardisation

Dès 1968, Lyndon B. Johnson, 36e président des États-Unis d'Amérique, requis l'utilisation de ce code pour tous les équipements informatiques, dans un souci de standardisation : "*All computers and related equiment configurations brought into the Federal Government inventory on and after July 1, 1969, must have the capability to use the Standard Code for Information Interchange and the formats prescribed by the magnetic tape and paper tape standards when these media are used*".

Ce fut certainement une raison importante du succès de l'encodage ASCII.

(viii) Début des difficultés

Toutefois, les difficultés allaient naître de cette standardisation, où les comportement précis de tous les caractères n'étaient pas définis. Par exemple, le code 127 "delete", était destiné à l'origine à reperforer à tous les endroits possibles une carte perforée (mettre les 7 bits à 1).

Il fut toutefois interprété par certains constructeurs comme « effacer » au sens propre, et DEC l'utilisa ainsi pour supprimer le caractère juste avant le curseur sur les systèmes vidéos.

Le caractère "backspace" était pourtant destiné à cet usage : il fut donc réattribué à "delete" la fonction d'effacer le caractère suivant le curseur sur les systèmes vidéos Unix, ce qui correspond dans cet exemple particulier à une triple évolution, non normalisée bien sûr, du standard !

Les exemples similaires abondent, par exemple sur l'utilisation des caractère "return" et "linefeed" pour marquer la fin des lignes de texte, et furent à l'origine des premières incompatibilités entre systèmes d'exploitations. Ainsi, les premiers Macintosh utilisaient simplement "return", alors le systèmes DEC et MS-DOS utilisaient à fois "return" et "linefeed".

(ix) Aggravation des difficultés

Les difficultés furent aggravées par le fait que le standard ASCII était un standard destiné à la langue anglaise. Chaque pays fit donc sa norme incompatible, comme le Z62010 en France... La norme ISO 646 vit donc le jour en 1972, mais n'améliora guère la situation, vu qu'aucun code ne fut rajouté. Elle ne consista qu'à la réattribution de certains codes à des caractères spécifique à une langue, et ce pour chaque langue.

Ceci vit renforcer les problèmes de compatibilité, puisqu'il devenait impossible de transmettre les 95 caractères affichables d'un pays à l'autre, de manière inchangée !

(x) Retour des jeux de caractères

Des jeux de caractères, utilisant le 8e bit, furent donc développés. Citons par exemple le CP-437, développé par Microsoft, rajoutant des caractères semi-graphiques pour dessiner à l'écran, le CP-850 contenant moins de ces caractères pour faire de la place aux caractères les plus utilisés en Europe occidentale, dont les lettres accentuées françaises, le CP-863 sorte de mélange du CP-437 et du CP-850, destiné au Canada...

Cette période de retour des jeux de caractère est liée au manque de normalisation d'une utilisation du 8e bit. Face aux exigences du marché, en l'absence d'une norme officielle, chaque constructeur dût développer ses propres encodages pour répondre aux besoins de ses clients.

2-3-2. Les normes ISO-8859

Une grande étape fut franchie avec la normalisation de ces utilisations du 8e bit pour ajouter 128 autres caractères tout en conservant la compatibilité avec les 128 premiers caractères normalisés par l'ASCII *[1998-ISO-8859-1]*.

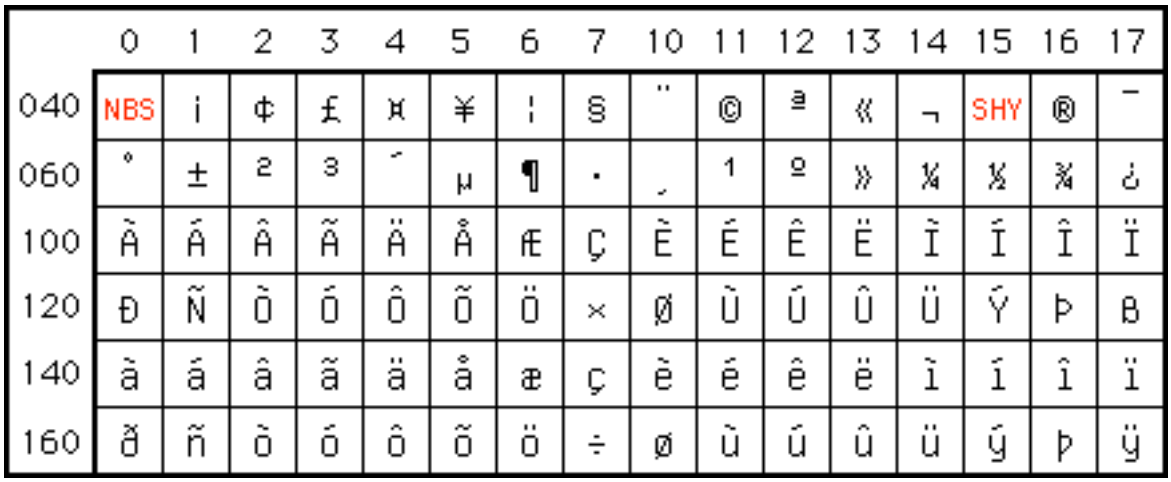
Ainsi, 2^8=256 caractères pouvaient dès lors êtres gérés.

Chaque groupe de langue utilisa les 128 nouveaux caractères pour gérer ses propres langues.

La norme ISO-8859-1, dite Latin 1, fut destiné à l'utilisation en Europe, mais dût coexister avec le CP-850 de MS-DOS, suivi du Windows-1252, globalement compatible avec le ISO-8859-1, correspondant à une préversion du 8859-1 à laquelle furent rajoutés de nouveaux caractères.

[2007-Haralambous-Encodings] apporte plus de précisions sur les parentés entre les encodages Microsoft, Apple et ISO, mais pour les besoins de cette thèse, une telle précision n'est pas utile : il suffit d'étudier l'historique des encodages dans ses grandes lignes pour en extraire les points clés, comme précédemment mentionnés (ex : optimisation mathématique, apport des normalisations)

L'ISO-8859-1 gère donc les langues d'Europe de l'ouest, mais aussi l'Afrikaans et le Swahili : le Swahili, n'utilise aucun caractère accentué (comme l'Anglais, le Malais et l'Indonésien), donc n'importe quelle version d'ISO-8859 fait l'affaire - par commodité, le premier. Pour l'Afrikaans, dans la pratique, l'ISO-8859-1 est utilisé pour les mêmes raisons.



	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17
040	NBS	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	SHY	®	¯
060	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
100	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
120	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
140	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
160	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Figure 26: Complément de l'ASCII pour la norme ISO-8859-1

D'autres normes 8859 moins connues existent, citons :

- la 2 pour les langues d'Europe centrale : Polonais, Tchèque, Slovaque, Slovène, Croate, Hongrois, Roumain, probablement aussi le Sorbe ou Sorabe (Wende en allemand). Il y a toutefois un problème pour le Roumain, car le Latin 2 contient formellement les caractères avec une cédille, quand en roumain on utilise une virgule souscrite [1986-Ecma-latin1_4].

- la 3 initialement destinée au turc et à l'espéranto, initialement prévue pour les langues d'Europe du Sud, y compris le Turc, les langues d'Espagne (donc Catalan, Basque), et le Français, est en pratique d'usage limité à ces deux langues

- la 4 *[1998-ISO-8850-4]* pour les langues nordiques, n'est plus utilisée, car elle a été remplacée par le 13, et pour le cas de l'Estonien par le 9.

Pour ces 4 premières, l'appellation « latin n » est équivalente:ISO-8859-1/Latin 1.

Pour les suivantes, l'appellation varie, étant donné qu'il ne s'agit plus de langues immédiatement latines ou utilisant l'alphabet latin :

- la 5 pour le Cyrillique *[1999-Ecma-8859-5]* ,

- la 6 pour l'Arabe *[2000-Ecma-8859-6]*,

- la 7 pour le Grec *[1999-ISO-8859-7]*,

- la 8 pour l'Hébreu *[2000-ECMA-8859-8]*,

Viennent à nouveau des normes ISO, mais dont l'appellation latin est décalée:

- la 9 reprenant la 1, mais remplaçant les caractères islandais par les caractères turcs à des fins de compatibilité élargie *[1999-ECMA-8859-9]*.

Dite latin 5, elle est utilisée pour le Turc et l'Azéri même s'il manque le schwa, un « e » renversé. À noter, elle est utilisée aussi comme norme officielle aux Pays Bas, vu qu'elle répond à des besoins d'une population plus large que le latin 1.

- la 10 réarrangeant la 4 pour être plus utile aux langues nordiques qu'aux langues baltes qui conservèrent l'utilisation de la 4. Elle ne gère pas toutefois le Lapon Skolt *[1998-ISO-8859-10]*.

Dite latin 6, elle est peu utilisée en pratique. Elle pourrait être intéressante pour les langues sâmes (ou lapones), mais elle est même dans ce cas incomplète.

Suivent alors d'autres encodages non latin :

- la 11 pour le Thaïlandais *[1999-ISO-8859-11]*,

- la 12 (abandonnée officiellement en 1997) pour le devânagarî (Indien),

Puis viennent :

- la 13 correspondant comme la 10 à un réarrangement de la 4 pour les langues baltes *[1999-ISO-8859-13]*,

Dite latin 7, elle couvre les besoins des langues du pourtour de la mer Baltique, donc aussi le Polonais, l'Allemand et les langues de Scandinavie. Mais dans la pratique il est réduit en utilisation aux langues baltes et à l'Este (ou Estonien).

- la 14 pour les langues celtiques,

Dite latin 8, elle est utile pour le Breton, le Gaélique et les dialectes parlées au Pays de Galles et en Irlande.

- la 15 pour l'Europe occidentale en remplacement de la 1, qui inclut les caractères euro, la ligature œ et des lettres finlandaises et estoniennes, est une évolution majeure *[1998-ISO-8859-15]*.

Dite latin 9, elle est actuellement la norme recommandée en Europe vu le passage à l'Euro. Elle est aussi mieux adaptée pour le Français, le Finnois (en concurrence avec Latin 1) et l'Este ou Estonien (en concurrence avec Latin 7) *[2002-André-Codage]*

- la 16 pour l'Europe du sud, prévue pour l'Albanais, le Croate, le Hongrois, l'Italien, le Polonais, le Roumain, le Slovène, mais sachant aussi gérer le Finnois, le Français, l'Allemand et la nouvelle orthographe Irlandaise *[2000-ISO-8859-16].*

Dite latin 10, il s'agit d'une police proche à la fois du latin-9 et du latin-2, qui cherche à gérer plus les lettres que les symboles

Certaines langues ont été mieux gérées que d'autres : ainsi l'Allemand a ses sept caractères spéciaux aux mêmes positions dans toutes les variantes latines de l'ISO-8859 (1-4, 9-10, 13-16).

La situation commençait à devenir difficile à gérer, particulièrement pour les langues indiennes et asiatiques ne trouvant pas dans les 128 nouveaux codes assez d'espace pour gérer tous leurs caractères.

Certaines normes furent développés, dont les KOI8 pour le russe, le Big5, le Shift JIS pour le japonais, le EUC-KR pour le coréen...

En ce qui concerne les normes utilisées en Europe du fait de leur vaste diffusion, citons aussi:

- le cp1252 : équivalent du cp850 sous Windows
- le cp1254 : équivalent de l'ISO-8859-16 sous Windows

Des détails sur ces encodages moins importants peuvent être trouvés dans [2007-Haralambous-Encodings]. Il est toutefois important de bien étudier l'historique des encodages les plus importants dont l'évolution nous permet de tirer des conclusions sur leur évolution progressive dans le domaine de l'informatisation des formalismes d'écriture des langues parlées naturelles, pour les extrapoler ultérieurement aux langues signées : recherche de la concision, puis de la maximisation des combinaisons avec un code le plus réduit possible, entraînant ainsi des incompatibilités.

(i) Accéder aux symboles

Un autre problème est l'accès à tous ces symboles. Le cas du japonais a été précédemment étudié, mais la complexité est tout aussi importante pour les langues latines si l'on considère l'ensemble du répertoire ISO: aucun clavier ne dispose d'autant de touches.

Les systèmes d'exploitation modernes permettent donc d'accéder à une vaste proportion du répertoire par un système de combinaison de touche. La conséquences est toutefois une complication de l'étape d'entrée. Le clavier dit US-International, utilisé par exemple au Brésil, permet de mieux percevoir cette complexité.

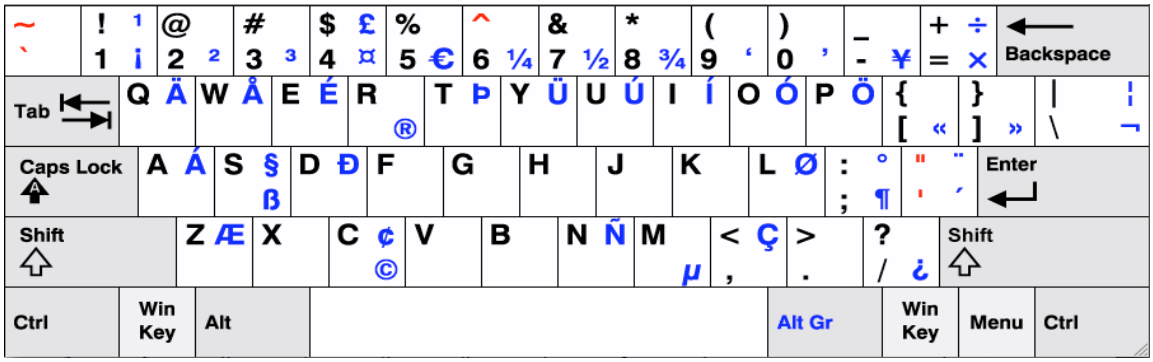


Figure 27: Clavier US-International

Sur la touche à droite du L par exemple, le point-virgule (;) est obtenu directement, les deux points (:) sont obtenus à l'aide de la touche Shift, le symbole numéro (°) à l'aide de la touche AltGr, et le symbole Pi (¶) à l'aide des touches Shift+AltGr.

D'autres claviers existent avec une complexité encore plus élevée : voyons par exemple le clavier Canadien normalisé, qui rajoute à la touche AltGr une autre touche de modification, la touche CtrlGr.

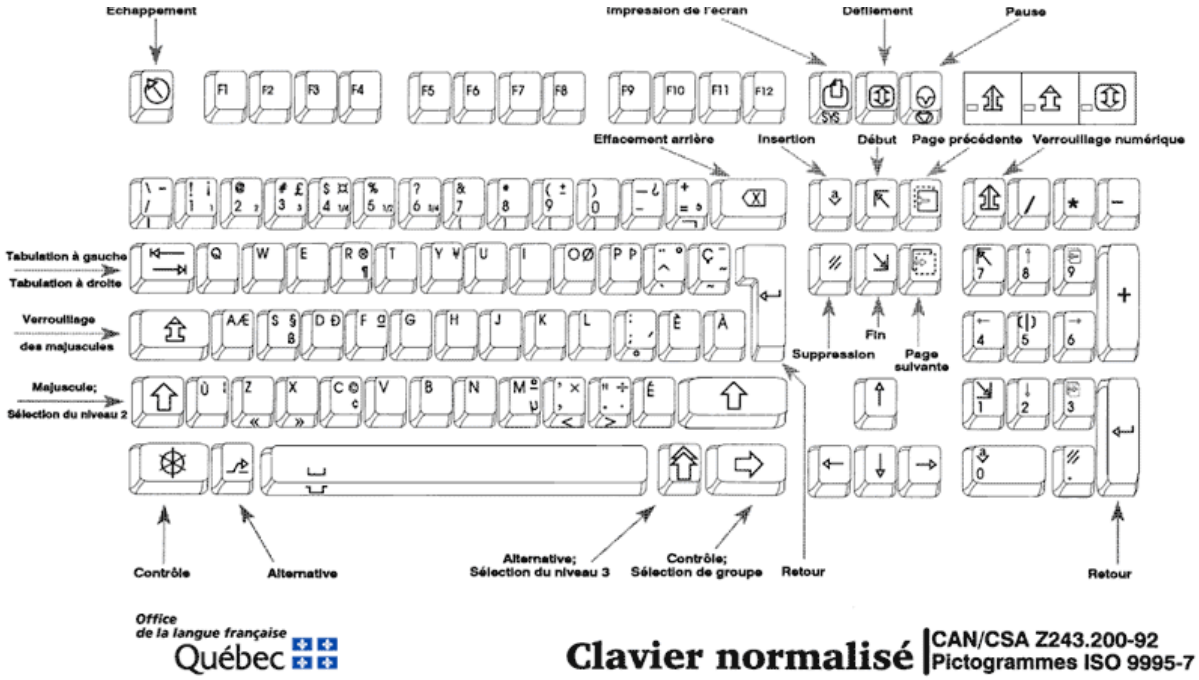


Figure 28: Clavier canadien normalisé CAN/CSA Z243.200-92

On peut ainsi positionner sur une simple touche de clavier 6 éléments au maximum, et 5 en pratique:

- un directement : niveau 1, ici le point «.»

- un autre avec la touche shift : niveau 2, ici les guillemets anglais «"»

- deux avec respectivement les touches AltGr et AltGr+Shift : niveau 3, ici le symbole supérieur «>», car il n'est pas fait usage de la combinaison AltGr+Shift

- deux autres avec de la même manière les touches CtrlGr et CtrlGr+Shift : sélection de groupe, ici le point médian «·» et le symbole diviser «÷»

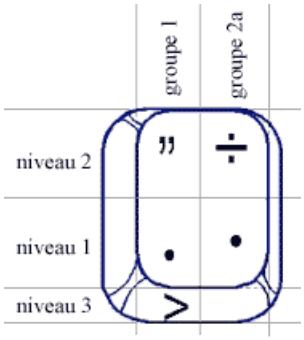


Figure 29: Détail d'une touche du clavier normalisé

Se pose alors le problème de la conservation des équivalences entre les claviers des différentes langues et des différents pays : certains systèmes de combinaison sont donc normalisés par l'ISO, pour garantir un résultat identique des combinaisons de touche à travers les claviers.

Mais de la même manière, plusieurs normes coexistent, et les plus grandes normes comme l'ISO-9995 et l'ISO-88884 autorisent un trop grand nombre de variabilités nationales *[2002-Andre-Unicode]*.

Ajouter un caractère sur un clavier aussi encombré, comme dans le cas de l'Euro, dans une position identique à travers plusieurs pays, est donc un problème à part entière *[2002-Andre-Euro]*, au delà de la difficulté technique de l'implémentation *[1999-Aznar-Euro]*.

2-3-3. Unicode

La réponse définitive aux multiples normes de jeu de symboles fut apportée par l'introduction en 1991 du format d'encodage le plus complet, mais aussi le plus complexe : Unicode *[1991-Unicode-Standard]* .

Les différents ISO-8859 ont été favorisés dans les années 1990 car ils avaient l'avantage de la simplicité, un octet correspondant à un caractère. Toutefois, ils n'avaient pas l'avantage de l'unicité, le caractère de correspondance variant selon l'ISO-8859...

Unicode a donc été créé, pour supporter toutes les écritures du monde. Les premières versions utilisaient 16 bits (65 536 caractères), ce qui était jugé suffisant pour supporter le tout.

Mais si la version 2.0 d'Unicode (1996) contenait 38 885 caractères, la 3.0 en 2000 en contenait 49 194, la 3.2 en contenait 95 156 et la 4.0 en contenait 96 382 : dès 1996 avec la version 2.0, les 16 bits furent jugés insuffisants.

Il est maintenant estimé que de 21 à 22 bits seront nécessaires.

Actuellement, Unicode inclut grâce à la norme ISO 10646 *[2005-Unicode-4.1]* la plupart des caractères nécessaires pour écrire les langues humaines.

De la même manière que les normes ISO-8859 ont conservé la compatibilité avec l'ASCII sur les 127 premiers caractères, Unicode conserve la compatibilité avec la norme 8859 la plus utilisée, l'ISO-8859-1, sur les 256 premiers caractères.

Cette notion de compatibilité dans les normes est importante à noter : bien que l'ISO-8859-15 ait remplacé la norme ISO-8859-1 en Europe occidentale, et par contamination dans les autres pays du monde, le consortium Unicode a choisi la solution de compatibilité historique.

(i) Fonctionnement d'Unicode

La notation d'un code a été standardisée sous la forme U suivi d'un tiret ou d'un plus, et de la notation hexadécimale, comme par exemple U+FEFF.

Unicode réserve actuellement 2^20+2^16=1 114 112 codes, répartis en 17 plans de 2^16=65 536 caractères, de U+0 à U+10FFFF.

Sur ce million de codes, seuls 100 000 caractères ont été assignés en 2005:

- Le premier plan, dit « Plan Basique Multilingue » ou « Plan 0 » contient la plupart des caractères des langues actuelles. La majorité des codes assignés servent à représenter les caractères Chinois, Japonais et Coréens.

Les 256 premiers caractères sont identiques à la norme ISO 8859-1 qui a été jusqu'à la sortie norme ISO-8859-15 dite « Latin 9 » prévue pour supporter l'Euro, la norme de référence en Europe, en Amérique du Nord et du Sud et en Australie et Nouvelle Zélande, et qui reste la norme la plus utilisée.

Il contient aussi de U-E000 à U-F8FF une zone réservée pour une utilisation privée : ce concept d'utilisation privée est hérité des encodages asiatiques, pour les systèmes ou programmes devant coder des caractères non standards.

- Le deuxième plan, dit « Plan supplémentaire Multilingue » ou « Plan 1 », sert à encoder les langues anciennes, les symboles mathématiques et numériques.

- Le troisième plan, dit « Plan supplémentaire Idéographique » ou « Plan 2 », est utilisé pour 40 000 idéogrammes Chinois, soit anciens, soit rarement utilisés dans la langue moderne.

- Les plans en cours d'attribution, où se font actuellement les propositions de normalisation

- Le quinzième plan, dit « Plan supplémentaire pour objectifs spécifiques » ou « Plan 14 », contient quelques caractères non recommandés et des caractères servant à préciser des variations.

- Le seizième plan ou « Plan 15 » et le dix-septième plan, ou « Plan 16 » sont totalement réservés pour une utilisation privée

Le fonctionnement détaillé d'Unicode est expliqué dans *[2002-André-Codage]* et *[2002-Andries-Unicode]* dont les illustrations suivantes sont tirées. Il est aussi possible de se référer à [2007-Haralambous-Encodings], excellent ouvrage de référence, très détaillé.

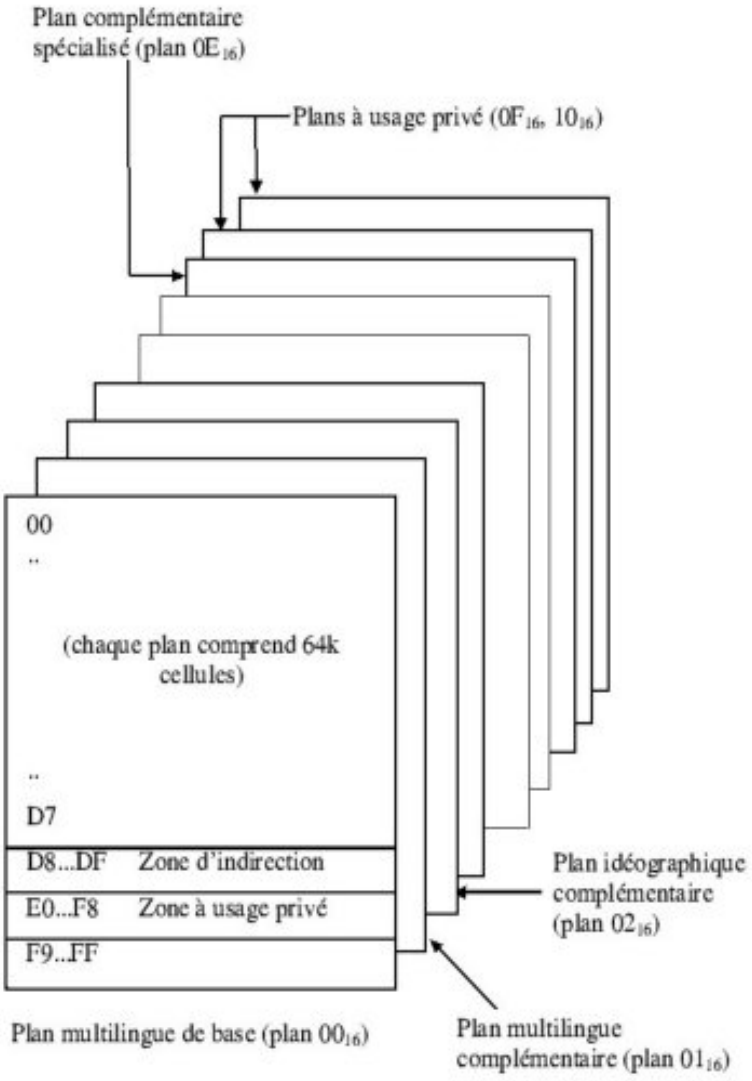


Figure 30: Organisation planaire d'Unicode

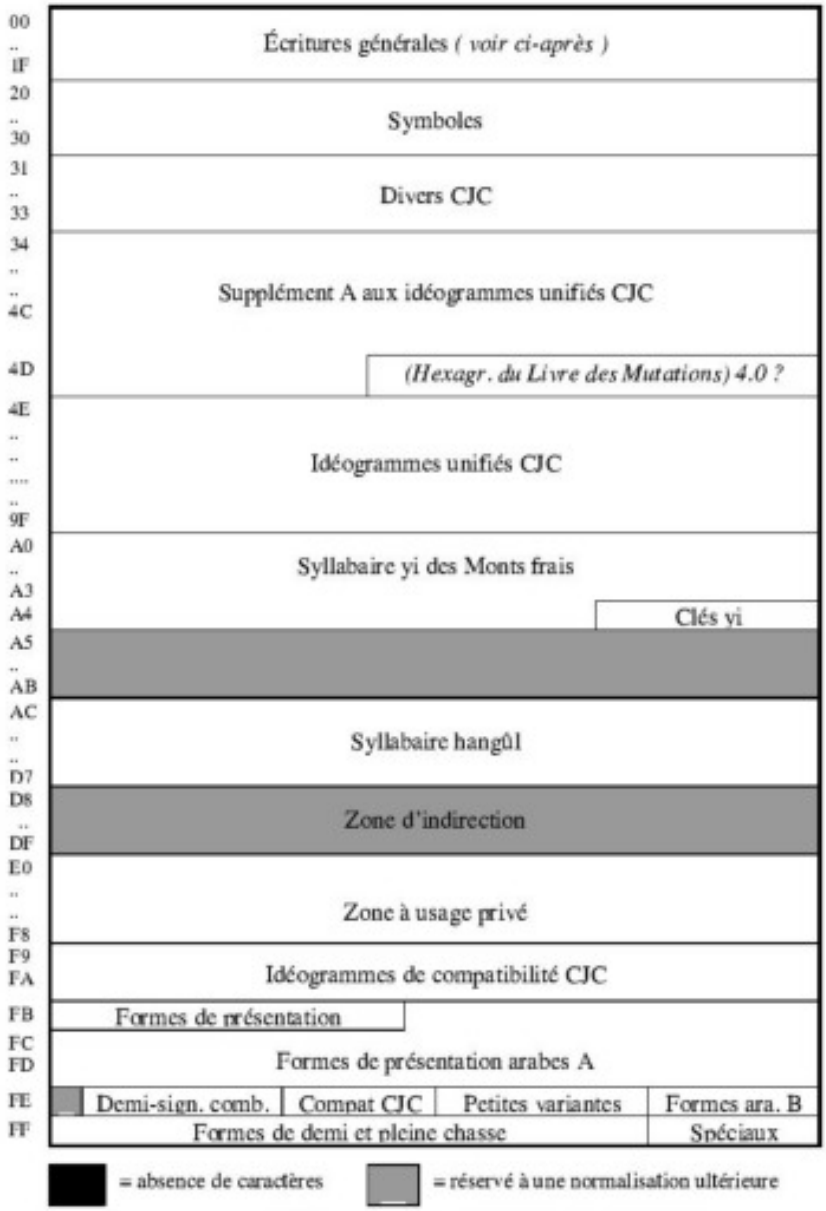


Figure 31: Contenu du premier plan

(ii) Encodage

Dans un but de compression, Unicode accepte plusieurs formes de présentation *[1993-ISO-UCS].*

Les plus couramment utilisées sont:

- UTF-8, spécifié dans le RFC 229, codage de taille variable de 1 à 4 octets, permettant d'être en moyenne moins coûteux en occupation mémoire, mais ralentissant les opérations où interviennent les extractions de sous-chaînes. Il est compatible avec l'ASCII, ancien standard 7 bit. Il est incompatible avec l'ISO 8859-1 « Latin 1 », dans la mesure ou le 8e bit est dupliqué. Ainsi, le « é » est codé « Ã© », et « GaÃ¯a » est une chaîne de quatre caractères correspondant à « Gaïa » lorsque l'ISO-8859-1 essaye de représenter l'unicode. Dans son cas, à la différence de codes de tailles fixes, comme un caractère peut être représenté sur plusieurs octets, il est impossible d'extraire une sous-chaîne par accès direct : il faut la parcourir depuis le début pour savoir où commence la lettre à extraire, ce qui peut être très pénalisant même sur une machine rapide.

- UCS-2, qui ne permet pas d'accéder à l'intégralité des caractères Unicode, uniquement aux 2^16 caractères du plan 0 - mais qui a l'avantage d'utiliser en permanence 2 octets pour tous les caractères.

- UTF-16, souvent confondu avec UCS-2. encodage de taille variable entre 2 octets (alors identique à UCS-2) et 4 octets (pour accéder à tout Unicode), très largement répandu. Il s'agit par exemple de l'encodage utilisé par Java

- UTF-32, qui comme UCS-4 permet d'accéder à l'intégralité des caractères Unicode, à l'aide d'un seul code 32 bits en bijection avec le caractère Unicode correspondant, dans les cas où les contraintes de rapidité et de taille de stockage ne sont pas les plus importantes.

Un point important est à noter : l'utilisation du caractère U+FEFF (ordre de bit, pour différencier entre petit-boutien et gros-boutien ("little-endian", "big-endian") est nécessaire pour UCS-2 et UTF-16 : le caractère est présent au début du fichier.

UTF-8, bien que ce ne soit pas nécessaire, utilise aussi ce caractère, qui par glissement sémantique sert maintenant à indiquer qu'un fichier est encodé selon un des mécanismes d'Unicode.

UTF-7 (ancien), UTF-9 et UTF-18 (expérimentaux, définis dans la RFC 4042) sont moins populaires et donc non détaillés ici.

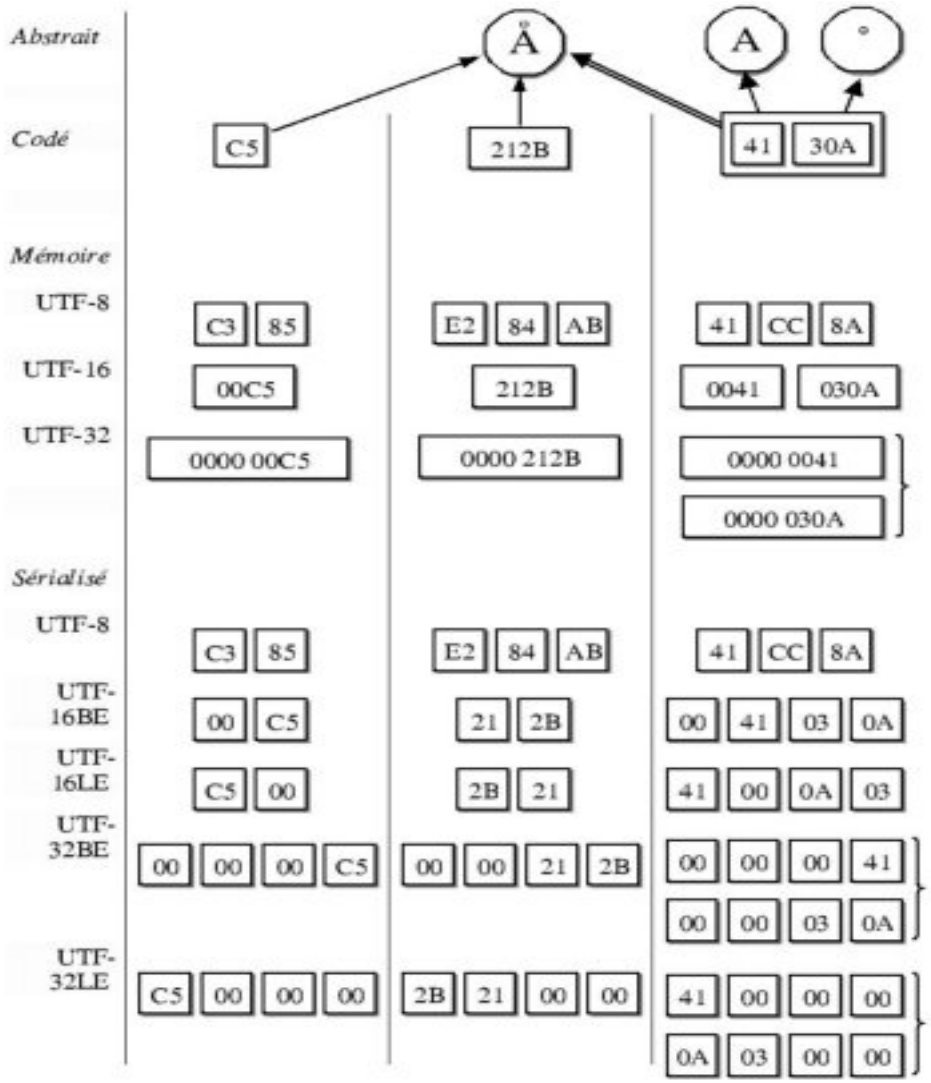


Figure 32: Exemple de sérialisation selon différents encodages

(iii) Vocabulaire unicode

Unicode ne fait qu'attribuer un numéro à des caractères, qui sont décrits textuellement, comme « lettre latine e minuscule avec un accent aigu » pour « é ».

Unicode ne code en revanche pas les descriptions graphique des caractères, les glyphes, Il n'y a donc pas une bijection entre la représentation du caractère et son numéro de code, comme c'est le cas dans une police ASCII ou latin-1 classique.

Toutefois, la plupart des descriptions de la norme associent un glyphe représentatif, à des fins d'illustration.

Glyphe	Numéro	Nom officiel français
D	U+0044	LETTRE MAJUSCULE LATINE D
≦	U+2270	NI PLUS PETIT NI ÉGAL À

Figure 33: Glyphe représentatif à côté d'un extrait de la norme Unicode

Mais le glyphe représentatif n'est qu'un exemple : il n'y a pas bijection, vu que le rendu du glyphe diffère selon la police choisie voire le contexte dans certaines langues comme l'Arabe, dont la détermination précise peut nécessiter des automates à états *[1998-Fanton-FSA_Arabic]*.

Glyphes (œils)	Caractères Unicode/ ISO 10646
A 𝒜 𝐀 𝘈 𝔸 A	U+0041 lettre majuscule latine a[13]
ffi ffi ffi ffi ffi	U+0066 lettre minuscule latine f + U+0066 lettre minuscule latine f + U+0069 lettre minuscule latine i
ﻓ ف	U+0647 lettre arabe fa'

Figure 34: Plusieurs possibilités de glyphes représentatifs

Ainsi, le caractère français « é » peut-il être décrit de deux manières : soit en utilisant directement le numéro correspondant au é, soit en faisant suivre le numéro du « e » par celui de l'accent aigu sans chasse.

Quelle que soit l'option choisie le même glyphe sera affiché : on dira du premier caractère é qu'il est précomposé, du second que c'est une composition ou une décomposition, puisque deux caractères forment un seul glyphe composé des deux.

De nombreux glyphes sont dans ce cas et peuvent être codés de ces deux manières.

Le plus souvent, le glyphe précomposé est préférable : c'est le cas pour le grec polytonique, par exemple, lequel, codé en décomposition, peut ne pas être satisfaisant graphiquement, les différents constituants du glyphe étant parfois mal disposés et peu lisibles selon les polices de caractères.

Pour toutes les nouvelles langues, le mode décomposé est par contre préféré, afin de limiter la duplication de caractères.

(iv) Redondance

De très nombreux glyphes se ressemblent, car des caractères très similaires ont été intégrés dans l'Unicode à différentes positions pour préserver des distinctions utilisées par des encodages précédents, et donc permettre une conversion vers l'Unicode sans perte d'information.

Tout l'alphabet latin est par exemple dupliqué en « pleine taille », pour les applications utilisant des polices de caractères à largeur fixée, puis l'alphabet est réencodé avec les logogramme japonais pour l'utilisation en romaji...

De même, à la fois les combinaisons de caractères et les caractères permettant d'aboutir à cette combinatoire sont encodés.

Pour reprendre l'exemple du « é », les caractères Unicode U+0065 (e minuscule latin), U+0301 (accent aigu pour combinaison) et U+00E9 (e minuscule latin avec accent aigu) sont tous encodés.

(v) Moteur unicode

De cette nécessité de création de glyphes à partir d'un nombre supérieur de caractères séparés, parfois très différents entre eux, vient la notion de moteur unicode : à partir de la forme décomposée, il crée la forme recomposée.

De même, certains systèmes d'écriture, comme la devânagarî ou les caractères arabes, nécessitent un traitement complexe des ligatures : les graphèmes changent en effet de forme en fonction de leur position et/ou par rapport à leurs voisines.

Ce travail de gestion de passage d'une séquence à un tout est le travail de ce moteur unicode, présenté en détail plus bas. Dans les deux cas, il s'agit d'un élément essentiel.

(vi) Premier cas (composition) : variabilité des caractères, unicité du glyphe

Dans le premier cas, lors de l'échange de documents entre deux systèmes d'exploitation, ou même deux applications, l'identité entre glyphes issus de plusieurs caractères recomposés et de glyphes issus de caractères précomposés doit être établie, au risque, sinon, d'incompatibilité.

Parfois, certains systèmes ne suivent pas cette recommandation : ainsi MacOSX utilise le NFD (forme de normalisation décomposée) au lieu du NFC dans la gestion de fichiers Unix, ce qui est sans conséquences dans la plupart des cas... sauf par exemple, lorsque des archives (zip, tar...) contenant des fichiers dont le nom est en NFC (composé) et doit être décompressé par un logiciel n'étant pas au courant de cette spécificité de MacOSX: le système de fichier exigeant la différence, il refuse la création du fichier en NFC, au lieu de l'accepter et de créer par exemple le fichier en NFD par équivalence.

Il y a plusieurs raisons techniques, dont la possibilité que le programme ne sache plus retrouver l'un de ses fichiers en raison de l'incompatibilité ’e/é, mais le résultat est gênant pour l'utilisateur puisque même avec un système Unicode, des incompatibilités liées à l'encodage en Unicode peuvent persister ; par exemple une archive tar crée sous GNU/Linux ne peut être décompressée sous MacOSX avec tar en ligne de commande.

(vii) Second cas (variance contextuelle) : unicité du caractère, variabilité du glyphe

Dans l'alphabet arabe, chaque lettre possède quatre allographes, à l'exception d'un petit nombre de lettres dont le tracé reste invariable.

Chaque variante s'utilise dans un contexte précis dépendant de sa place:

- en position indépendante, lorsque la lettre est seule dans le mot ;
- en position initiale d'un mot ;
- en position médiane ;
- en position finale.

À l'origine, l'alphabet arabe n'avait pas de telles variantes, qui sont nées des déformations impliquées par la graphie cursive, laquelle procède par des adaptations liées à la nécessité de ne pas lever le calame pour ne pas interrompre le trait. L'informatisation se doit de respecter cette graphie *[1992-Lagally-ArabTex]*.

Position	*finale*	*médiane*	*initiale*	*isolée*
Graphie	ه	ه	ه	ه
Lettres liées	ههه ه			

Figure 35: Variantes de hâ

De simples variantes non pertinentes, les allographes ont ensuite acquis le statut de formes normées et obligatoires, comme illustré par ce tracé d'une lettre, hā (rappelons que l'arabe se lit de droite à gauche).

Le processus ayant mené à la différenciation des allographes est ici très clair et dépend entièrement de la nécessité de ne pas lever le calame :

- la forme fondamentale est celle de la graphie isolée : le calame trace une simple boucle fermée ;
- lorsque la lettre apparaît en début de mot, le tracé commence comme pour une boucle mais ne peut se terminer une fois la boucle fermée : il doit repartir vers la gauche pour permettre la jonction avec la lettre suivante, d'où la boucle à l'intérieur de la première boucle, restée plus ou moins inachevée (certains scripteurs la ferment moins que dans l'image présentée ici) ;
- en milieu de mot, il n'est pas possible de tracer une boucle simple sans interrompre le trait. De la même manière que l'on trace un f cursif dans l'alphabet latin, on procède à une double boucle. On aurait pu se contenter d'une boucle ne partant pas vers le bas, mais une telle graphie représentait déjà une autre lettre ;
- en fin de mot, la boucle est simplement reliée à la lettre précédente.

Pour gérer une telle complexité en Unicode, la solution est un moteur unicode, qui gère la différence entre un symbole et un glyphe.

À partir du contexte, et des règles de compositions graphiques précédemment énoncées, le glyphe adéquat est généré.

Le même procédé est utile pour les langues européennes, bien que les variantes aient à la fois moins d'importance sur la lecture du texte, et ne soient pas liées à un contexte de manière aussi forte. Citons par exemple les ligatures f.

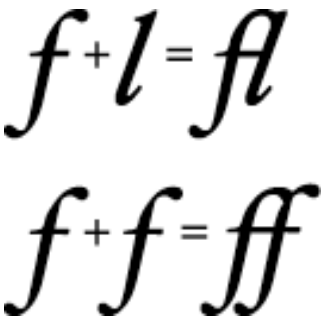


Figure 36: Ligature de f

D'autres correspondances entre des caractères et des glyphes existent - elles seront étudiées ci-bas avec les fonctionnalités informatiques nécessaires, vu leur relation avec l'informatisation et notamment les polices de caractères.

(viii) Rareté des polices de caractères

Mois d'une dizaine de polices de caractère sont « Pan-Unicode », c'est à dire capable de supporter la majorité des glyphes d'Unicode.

La plupart du temps, les polices ne contiennent qu'un sous ensemble des glyphes d'Unicode, dans la mesure ou la plupart des applications n'ont besoin que de quelques combinaisons de registres, comme par exemple les caractères Latin + caractères les plus fréquents + langues parlées localement.

En outre, les applications savent de plus en plus combiner des polices différentes pour obtenir des glyphes correspondant à différents registres, comme par exemple une police Japonaise + une police Russe.

Certains de ces sous-ensembles sont d'ailleurs normalisés : WGL-4 de 652 caractères pour toutes les langues utilisants les alphabets Latins, Grecs et Cyrilliques, MES-1 (335 caractères), MES-2 (1062 caractères), etc.

Les dernières normes ISO-8859 définissent d'ailleurs les caractères par leurs noms et leur position Unicode : elles sont ainsi devenues des schémas de codage de caractères Unicode, permettant de coder plus simplement une partie du registre Unicode par un octet.

Dans les cas où un glyphe n'est pas disponible, les logiciels de rendu affichent en général un rectangle vide, ou le glyphe du caractère unicode U+FFFD spécialement prévu pour cet usage.

Certains systèmes d'exploitation utilisent des polices affichant la valeur hexadécimale du caractère ne pouvant être affiché.

On comprend donc que le terme de police Unicode doit être utilisé très prudemment : avoir une police qui représente un certain nombre voire théoriquement toutes les représentations graphiques que l'on peut obtenir avec Unicode n'est pas suffisant, il faut en plus que le système d'affichage possède les mécanismes de représentation idoines capables de gérer les ligatures, variantes contextuelles et autres formes conjointes de certaines écritures.

Au contraire, une police qui ne représente que certains glyphes mais qui sait comment les afficher mérite mieux le terme de police Unicode.

(ix) Conclusion

À partir des exemples donnés, la complexité du support Unicode peut être mieux comprise.

Un système complet nécessite à la fois un moteur complet, et des polices complètes.

On constate toutefois que les systèmes d'encodage, du Morse à l'Unicode, correspondent à une évolution progressive dans laquelle on observe principalement :

- la rupture du lien entre caractère et touche du clavier

- l'introduction de caractères non affichables pour gérer les flux de texte

- l'optimisation de la disposition de l'encodage à des fins mécaniques (usure) puis algorithmiques (rapidité/simplicité)

- le rajout de caractères avec préservation de la comptabilité

- la standardisation lorsque plusieurs normes coexistent

- la rupture du lien entre caractère et apparence graphique du glyphe

Ainsi, l'encodage informatique des langues parlées fait appel à une complexification progressive, qui peut paraître inutile de par la complexité résultante, mais qui est nécessaire pour répondre aux besoins progressivement complexes.

Il semble légitime de comparer cette évolution à celle des langues signées

2-3-4. Encodage des formalismes d'écriture des langues signées

(i) Introduction

Une évolution à peu près similaire se retrouve pour les formalismes d'écriture des langues signées, avec toutefois, une différence majeure liée à l'historique de l'évolution des technologies.

Boris Hessen, scientifique soviétique, postulait dès 1931 *[1931-Hessen-SocioEconomic_Science]* que les besoins socio-économiques dirigeaient l'évolution de la science, et que les technologies et théories ne naissaient que s'il existait préalablement un besoin économique.

Comme démontré plus haut lors de l'étude de l'état de l'art des formalismes d'encodages, tous ont été conçus pour manipuler une forme bien précise d'une langue écrite, l'anglais, et dans un but initial de télécommunication.

L'informatisation ne fut qu'un produit dérivé imprévu, et de multiples difficultés sont survenues lors des tentatives d'adaptation de ces encodages à des langues différentes de celles pour laquelle elles ont été conçues.

Ces difficultés, comme par exemple des problèmes d'incompatibilité, ont retardé une utilisation effective et partagée des formalismes.

L'utilisation informatique des langues signées souffre actuellement de cette situation.

(ii) L'outil détermine le besoin

À travers l'état de l'art, il apparaît que tant qu'un encodage n'a pas permis de proprement informatiser une langue donnée, non seulement son utilisation pour cette langue a été limitée, mais l'informatisation même de cette langue a été aussi ralentie. Pour le français, le cas du Z62010 et des problèmes de compatibilité avec les autres encodages est par exemple à rappeler.

Ainsi, aussi paradoxal que cela puisse paraître, l'outil informatique détermine les besoins et les usages. Le postulat de Hessen sur l'apparition des technologies semble validé même dans la diffusion des technologies.

Autre exemple, puisqu'il était difficile de bien informatiser des textes en français avec les normes de l'époque, des simplification arbitraires ont été décidées - comme expliqué précédemment avec la ligature oe (œ) et les guillemets français («») qui n'ont pas été encodés.

Pourtant, des précurseurs des encodages (Morse, Baudot) aux derniers nés (Iso 8859, Unicode), l'évolution technologique a rendu possible cette pratique, mais il semble que le besoin, non entretenu, ait disparu.

Même maintenant, la ligature oe et les guillemets français sont laissés au bon vouloir des logiciels, et non accessibles à l'utilisateur.

Il pourrait être rétorqué que ces caractères («œ») ne sont pas nécessaires, mais puisqu'il est impossible encore actuellement d'avoir des adresse électroniques accentuées, les accents sont-ils eux-aussi inutiles ? Où poser la limite entre le nécessaire et l'inutile ? Et surtout, est-ce que de telles limites sont objectives, ou proviennent-elles d'une limitation des outils informatiques, elles même conditionnées par le besoin socio-économique ?

De telles considérations font de l'encodage une étape importante : en fixant une norme, elle définit ce qui persistera et ce qui disparaîtra.

L'absence d'une forme écrite de la LSF a par exemple été utilisée comme un argument contre les langues signées. Avec la numérisation progressive de la société, l'existence et la qualité de l'informatisation de la LSF deviennent des questions de premier plans : si la LSF est traitée comme l'ont été la ligature oe et les guillemets français, son usage risquerait peut être d'en pâtir.

Il est donc nécessaire de s'intéresser aux encodages des langues signées utilisés actuellement, pour les restituer de manière comparative, sans longueur inutile, avec les encodages des autres langues.

Dans un souci de simplicité de l'étude et de comparabilité, seuls seront étudiés les principaux encodages, définis ainsi par la popularité et l'utilisabilité qu'ils auront eus.

(iii) Encodages binaires

Les premiers encodages apparus pour SW sont à peu près similaires aux encodages 6-bits des langues écrites : le logiciel SignWriter, développé en 1987, utilise ainsi un encodage à longueur variable *[1995-Sutton-SignWriter]*.

Il était donc incompatible avec tout autre encodage, comme pour la norme Z62010.

(iv) Encodages pseudo ISO

Les encodages tels qu'utilisés par HamNoSys *[1987-Prillwitz-HamNoSys]*, utilisent volontairement un nombre limité de caractères, afin de pouvoir être informatisés par une simple police de caractère, comme pour les normes ISO-8859.

(v) Encodages en attributs

Les nouveaux éditeurs en SW *[2003-DaRocha-SignWriting]*, comme SWEdit *[2002-Torchelsen-SWEdit]*, utilisent l'encodage SWML décrit ci-dessous *[2001-DaRocha-SWML]*, qui est très différent des encodages précédemment étudiés.

Il s'agit en effet d'un encodage par attributs, popularisé par l'utilisation d'internet *[2001-DaRocha-SW_Similarity]*.

Pour un signe, l'ensemble des caractères le composant est indiqué.

À l'instar des formes décomposées d'Unicode (NFD), pour chaque caractère correspondant à un symbole de base, les variations comme la rotation et le remplissage sont indiquées par rapport au symbole.

D'autres attributs positionnels viennent compléter ces informations, pour situer dans un plan bidimensionnel chaque caractère.

Le tout est réalisé dans une notation XML *[1996-Bray-XML]*, c'est à dire de manière textuelle, en utilisant un encodage comme le ISO-8859-1 pour le texte.

Des traitements plus complexes peuvent alors être réalisés dans une opération de traitement automatique des langues *[2001-DaRocha-SignWriting].*

Regardons tout d'abord un exemple d'encodage SWML, à partir des glyphes correspondant au début du compte « la belle au bois dormant ».

Dans cet extrait, la directionnalité est de haut en bas, et la progression de bloc de gauche à droite. 7 signes SW sont affichés, composés chacun de multiples symboles. Deux séparateurs et un terminateur sont présents.

En regardant l'encodage SWML qui en est fait, on retrouve les symboles de base et les symboles correspondant à chaque signe SW.

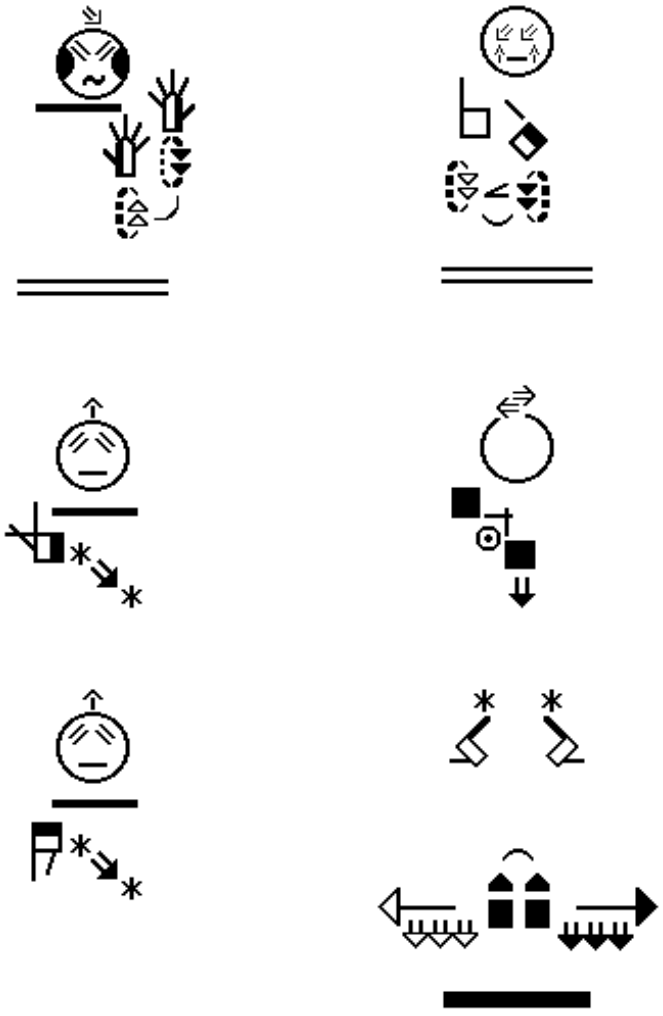


Figure 37: Extrait de « la belle au bois dormant » en langue des signes américaine

```
<?xml version="1.0"?>
<!DOCTYPE swml SYSTEM "http://www.signpuddle.com/swml/
swml-s.dtd">
<swml dialect="S" version="1.1" lang="sgn" glosslang="">
    <sign lane="0">
      <gloss></gloss>
      <symbol x="140" y="162">08-02-001-01-01-06</symbol>
      <symbol x="92" y="80">03-06-001-01-01-08</symbol>
      <symbol x="92" y="80">03-04-001-01-01-01</symbol>
      <symbol x="82" y="117">05-01-001-01-01-01</symbol>
      <symbol x="137" y="100">01-05-001-01-02-01</symbol>
      <symbol x="115" y="122">01-05-001-01-02-09</symbol>
      <symbol x="143" y="133">02-10-007-01-01-05</symbol>
      <symbol x="121" y="158">02-10-007-01-05-09</symbol>
      <symbol x="92" y="80">03-01-001-01-01-03</symbol>
      <symbol x="92" y="69">04-03-001-01-01-06</symbol>
    </sign>
    <sign lane="0">
      <gloss></gloss>
      <symbol x="82" y="110">08-04-002-01-01-01</symbol>
    </sign>
    <sign lane="0">
      <gloss></gloss>
      <symbol x="87" y="73">04-04-001-01-01-01</symbol>
      <symbol x="87" y="84">03-01-001-01-01-01</symbol>
      <symbol x="87" y="84">03-06-001-01-01-01</symbol>
      <symbol x="85" y="127">05-01-001-01-01-01</symbol>
      <symbol x="94" y="144">02-01-001-01-01-01</symbol>
      <symbol x="104" y="152">02-03-001-01-01-06</symbol>
      <symbol x="119" y="165">02-01-001-01-01-01</symbol>
      <symbol x="62" y="124">01-03-018-01-02-01</symbol>
    </sign>
    <sign lane="0">
      <gloss></gloss>
      <symbol x="86" y="72">04-04-001-01-01-01</symbol>
      <symbol x="86" y="83">03-01-001-01-01-01</symbol>
      <symbol x="86" y="83">03-06-001-01-01-01</symbol>
      <symbol x="84" y="126">05-01-001-01-01-01</symbol>
      <symbol x="93" y="143">02-01-001-01-01-01</symbol>
      <symbol x="103" y="151">02-03-001-01-01-06</symbol>
      <symbol x="118" y="164">02-01-001-01-01-01</symbol>
      <symbol x="74" y="137">01-09-029-01-02-03</symbol>
    </sign>
```

(passage à la colonne de droite)

```
    <sign lane="0">
      <gloss></gloss>
      <symbol x="77" y="96">01-01-001-01-01-09</symbol>
      <symbol x="99" y="107">01-01-001-01-05-02</symbol>
      <symbol x="105" y="142">02-10-007-01-04-05</symbol>
      <symbol x="70" y="137">02-10-007-01-05-13</symbol>
      <symbol x="88" y="163">08-02-001-01-01-05</symbol>
      <symbol x="89" y="148">08-01-001-01-01-01</symbol>
      <symbol x="87" y="61">03-03-001-01-01-04</symbol>
      <symbol x="87" y="61">03-06-001-01-02-01</symbol>
    </sign>
    <sign lane="0">
      <gloss></gloss>
      <symbol x="71" y="110">08-04-002-01-01-01</symbol>
    </sign>
    <sign lane="0">
      <gloss></gloss>
      <symbol x="73" y="118">01-01-001-01-06-11</symbol>
      <symbol x="99" y="128">01-01-001-01-06-01</symbol>
      <symbol x="85" y="137">02-01-001-01-01-07</symbol>
      <symbol x="101" y="162">02-03-001-01-01-05</symbol>
      <symbol x="87" y="68">04-03-001-01-03-03</symbol>
    </sign>
    <sign lane="0">
      <gloss></gloss>
      <symbol x="142" y="82">01-05-047-01-01-08</symbol>
      <symbol x="96" y="82">01-05-047-01-01-16</symbol>
      <symbol x="141" y="69">02-01-001-01-01-01</symbol>
      <symbol x="108" y="69">02-01-001-01-01-01</symbol>
    </sign>
    <sign lane="0">
      <gloss></gloss>
      <symbol x="98" y="121">01-05-011-01-06-01</symbol>
      <symbol x="116" y="121">01-05-011-01-06-01</symbol>
      <symbol x="105" y="109">08-02-001-01-01-01</symbol>
      <symbol x="141" y="128">02-05-001-03-01-07</symbol>
      <symbol x="44" y="128">02-05-001-03-02-15</symbol>
      <symbol x="132" y="145">02-03-005-01-01-05</symbol>
      <symbol x="56" y="144">02-03-005-01-02-05</symbol>
    </sign>
    <sign lane="0">
      <gloss></gloss>
      <symbol x="81" y="121">08-04-001-01-01-01</symbol>
    </sign>
    <sign lane="0">
      <gloss></gloss>
    </sign>
</swml>
```

Figure 38: Encodage SWML correspondant (coordonnées absolues)

(vi) Discussions sur le fonctionnement de SWML

Rappellons que chaque élément d'un signe SW est nommé symbole, et que ces symboles sont codés par une séquence de 6 entiers permettant de retrouver l'équivalent du caractère correspondant dans la norme SSS.

Les attributs x et y servent à composer le véritable glyphe final, pour un signe SW. Il y a dans cet exemple 6 signes SW, regroupés par des séparateurs.

Bien que les langues signées soient d'une densité sémantique supérieure aux langues écrites, si l'on faisait une comparaison entre les lettres et les symboles, puis entre les signes SW et les mots, il est ici nécessaire d'utiliser une ligne pour chaque lettre, et de en moyenne 8 lignes pour chaque mot.

Cet encodage est donc très complexe d'un point de vue informatique, à la fois d'un point de vue temps de calcul, que de l'espace de stockage nécessaire.

Il s'agit d'un encodage n'ayant pas de comparaison dans les langues écrites. Le choix d'un support XML/textuel pour l'encodage est source de difficultés supplémentaires puisque les systèmes d'exploitations ont développé une systématisation du problème très différente, comme précédemment présenté.

(vii) Outils, besoins, et retard d'encodage

Les langues signées comme expliqué dans l'état de l'art ont longtemps été pratiquées en France dans un contexte marqué par l'interdiction, et à l'international marqué par une population de locuteurs peu nombreuse.

En conséquence, il semble qu'il n'y ait jamais eu de besoin d'encodage des langues signées, par exemple pour les télécommunications (ex : télégraphe en langue des signes...)

Il s'agit de la différence majeure des encodages des langues écrites dites naturelles et des langues signées : les premiers encodages informatiques des langues écrites naturelles, comme vu précédemment, sont nés des besoins de télécommunications ; ils ont ensuite évolué, pour être utilisés dans l'informatique, en étant au besoin réadaptés.

Mais dans le cas des langues signées, en l'abscence de besoin de télécommunication pour des causes diverses, les encodages sont nés en même temps que la diffusion de l'informatique auprès des sourds.

Et de la même manière que pour le français, les usages qui n'ont pas été rendus possibles ne sont pas devenus des besoins.

Ainsi, tous les encodages présentés présentent des limites importantes, mais qui ne sont que peu critiquées.

La situation est-elle destinée à rester figée ?

(viii) Conclusion

Les encodages des langues signées et des langues écrites naturelles partagent de nombreux parallèles, mais connaissent des différences liés à l'historique d'apparition de ces encodages.

Les encodage existants des langues signées restent incompatibles avec les autres langues écrites, dans la mesure où ils nécessitent des logiciels spécifiques puisque les systèmes d'exploitation modernes, commençant à bien gérer l'Unicode, n'ont pas été conçus pour gérer les encodages XML. Ces derniers sont certes très utiles, mais au niveau applicatif, et non en tant qu'encodages des caractères pour lesquels ils n'ont pas été conçus. De tels usages sont donc laissés à la discrétion des logiciels utilisés, qui dans la quasi-totalité des cas ne supportent pas les encodages spécifiques des langues des signes.

Étant donné le taux élevé d'illettrisme et la diffusion continue des nouvelles technologies, il semble toutefois toujours possible de proposer un nouvel encodage mieux adapté.

Le but serait d'inverser la tendance du mauvais support, en proposant un encodage capable de supporter toute la richesse de la langue des signes, pour créer puis entretenir le besoin d'une écriture de la langue des signes.

La question est alors pourquoi entretenir ce besoin ? À quoi bon faire naître et entretenir un besoin ? La question semble plus philosophique qu'informatique ; pour ma part il me semble nécessaire de ne pas créer de limite artificielle entre inutile et utile, pour les sourds, dans leur langue native, en ne mettant à leur disposition que des encodages imparfaits.

La place de l'informatique n'est pas d'imposer des limites ou des contraintes, mais de supporter les usages qui peuvent être faits, ici par exemple par des sourds.

À eux de déterminer ce qui doit être fait, et à l'informatique de supporter toutes les complexités nécessaires pour cela.

Mettons-nous donc dans l'optique d'un support informatique le plus complet possible de la langue des signes, en examinant les fonctionnalités nécessaires aux utilisateurs d'un encodage de la LSF.

2-4. Fonctionnalités informatiques nécessaires

2-4-1. Introduction

L'intérêt du support informatique d'un encodage est la gestion automatisée.

Ainsi, bien qu'il soit possible de définir isolément un encodage, puis de le normer, cette dite norme n'en devient pas pour autant un standard : les utilisateurs d'autres normes ne peuvent pas en bénéficier.

Et même dans le cas d'un seul utilisateur, les différents logiciels qu'il peut être amené à utiliser ne peuvent pas tous en bénéficier.

On voit donc que les fonctionnalités nécessaires pour un support informatique peuvent s'entendre de manières différentes.

2-4-2. Objectifs logiciels

Actuellement, deux grandes classes de logiciels sont utilisés par les langues écrites :

- Bureautique : Traitement de texte, tableur, présentation
- Communication : Courrier électronique, messagerie, navigation internet

Une troisième classe de logiciel est plus rarement utilisée en pratique, et plutôt réservée aux professionnels :

- Linguistique : analyse de texte, de style, statistiques, aide à la génération, traitement automatique des langues

Toutefois, l'objectif n'est pas de se limiter à une liste restreinte d'applications, mais de supporter tous les logiciels existant pour les langues écrites, afin d'en faire bénéficier les langues signées sans travail complémentaire pour chaque logiciel, en permettant le partage de données (échange de fichiers, impression...) et la compatibilité avec les langues écrites.

2-4-3. Niveaux de support

La conséquence est la définition de plusieurs niveaux de support d'une informatisation de la LSF.

On peut ainsi séparer un support :

- limité à une application logicielle, comme dans le cas des formats de fichiers ; bien que certains de ces formats deviennent parfois des standards (ex : PDF) la plupart des formats de fichiers sont spécifiques à un logiciel donné.

Dans le cas de SWEdit et de SignWriter, l'encodage SWML utilisé par l'un n'est pas utilisé par l'autre.

Si d'autres logiciels supportant SWML existent, ils demeurent toutefois des exceptions. Un logiciel choisi par un utilisateur ne connaissant pas cet encodage ne le supporte donc pas.

- élargi au système d'exploitation, comme dans le cas des encodages d'écriture des langues écrites ; bien que certains systèmes d'exploitation aient des difficultés dans ces conversions (MacOSX : NFD/NFC)

Dans le cas des systèmes d'exploitation modernes, il est possible de voir les kanjis d'un texte japonais sans travail particulier.

Il est aussi possible de les copier et de les coller dans tout autre logiciel, pour pouvoir ensuite les envoyer par courrier électronique, ou les imprimer.

Au niveau du système d'exploitation, il faut aussi séparer différent niveaux.

(i) Support complet, par le système d'exploitation

Ce support complet, au niveau du système d'exploitation, vient de la gestion de plusieurs sous-problèmes, qui ne sont alors pas laissés au bon vouloir des logiciels.

Le système d'exploitation offre en effet dans ces cas un support à trois niveaux:

- tout d'abord un support interne, pour que les encodages ne soient pas perdus en passant d'une application à l'autre, comme pour l'exemple « Gaïa » en Unicode UTF-8 qui n'est pas conservé lors du passage en ISO-8859-1

- ensuite un support au niveau de l'affichage, pour que les encodages puissent être rendus dans une forme immédiatement utilisable et reconnaissable par l'utilisateur

- enfin un support au niveau de l'entrée, pour que le texte ainsi encodé puisse être modifié ou complété

(ii) Support complet, comprenant le système d'exploitation

Un quatrième niveau de support mérite d'être mentionné à part : la localisation.

Il s'agit de la possibilité pour un système d'exploitation d'être intégralement traduit : les menus, messages, fichiers d'aide et autres dialogues sont dans une langue donnée en totalité. Il découle du support des niveaux précédents.

Ce niveau de support permettrait que les sourds puissent bénéficier d'un système intégralement dans leur langue, comme il est offert pour les langues régionales en France (suivant l'exemple de GNU/Linux, Microsoft propose ainsi Windows en breton, alsacien, basque...)

(iii) Supports accessoires, pouvant être dévolus aux logiciels

D'autres niveaux de support plus accessoires sont:

- la capacité de vérification orthographique et grammaticale d'une langue, comme pour les textes sous Word

- la capacité de resynthèse de la forme originale, comme avec VoiceOver sous MacOSX qui lit à haute voix le texte sélectionné

Dans les deux cas, ils nécessitent des travaux complexes supplémentaires, et ne bénéficient que rarement aux utilisateurs, excepté dans des contextes particuliers.

Étant donné que tous les niveaux de supports sont offerts par Unicode ou par un de ses sous ensembles, comme le ISO 8859-1, étudions maintenant les fonctionnalités nécessaires à Unicode pour des cas plus complexes, en regardant notamment ce qu'il faut pour gérer correctement l'affiche d'un texte dans une fenêtre graphique.

2-4-4. Fonctionnalités nécessaires à Unicode

Même si Unicode a été créé pour gérer les scripts de toutes les langues humaines, il serait faux de croire que les deux aspects précédemment présentés, correspondance entre des caractères et un glyphe, et variance contextuelle, sont les deux seuls points importants.

D'autres attributs sont nécessaires pour répondre aux besoins informatiques, notamment d'affichage dans une surface de dimension fixée.

(i) Directionalité

Le premier de ces attributs est la directionalité.

En effet, si dans une langue donnée la directionalité est fixée, comme par exemple de gauche à droite en Français, gérer plusieurs langues peut être problématique.

Par exemple, comment intégrer des citations s'écrivant de gauche à droite comme le Français, et des citations s'écrivant de droite à gauche comme l'Arabe ?

Il y a donc besoin d'une différence entre la représentation en mémoire, et la représentation à l'écran.

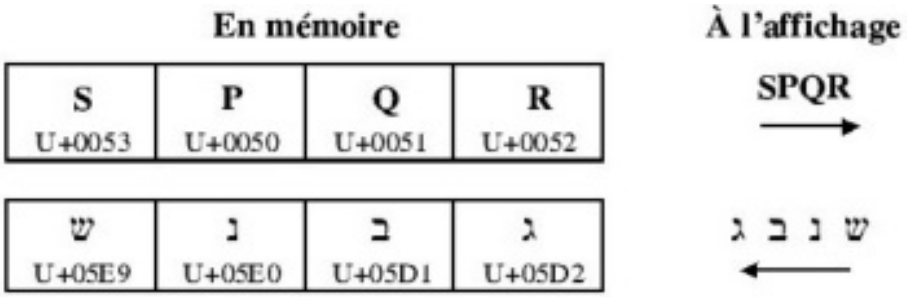


Figure 39: Illustration de la directionalité : alphabet latin et hébreu

Pour répondre à ce besoin, Unicode définit la directionalité du flux écrit :

- de gauche à droite, pour les langues latines
- de droite à gauche, pour les langues sémites
- de haut en bas, pour les langues asiatiques
- de bas en haut, pour le runique

(ii) Progression de bloc

La directionalité n'est qu'un des deux paramètres nécessaires à l'affichage du texte. En effet, le flux textuel ne peut la plupart du temps se contenir dans la surface d'affichage voulue (colonne, widget, boite de texte, label...)

Pour cela, une autre dimension est utilisée : celle de la progression de bloc. En français, elle correspond au sens de passage d'une ligne à l'autre : de haut en bas.

(iii) Séparation des mots

À l'issue des deux étapes précédente, un problème peut se présenter si les mots ne sont pas individualisés par la présence de séparateurs.

Dans le cas de certaines langues, comme pour le Pâli (dans lequel le Canon Bouddhique Théravada est écrit), le Thaï, le Lao, le Khmer Cambodgien et le Birman, les lettres ne sont pas individualisées en mots.

Dans d'autres cas, comme la césure dans les langues latines (passage à la ligne avec coupure du mot en son milieu par un trait d'union), des séparateurs doivent pouvoir être introduits au besoin.

Si le problème de segmentation existe aussi pour la langue des signes, il s'agit plus d'un problème de segmentation temporelle que de segmentation de l'écrit. En effet, les formalismes d'écritures comme SW séparent les signes, composés de symboles.

(iv) Mode d'écриture

Une fois proprement segmenté, le flux textuel est donc découpé en blocs, selon la présence de séparateurs entre les « mots », dits blocs.

Ces blocs sont à leur tour insérés dans la première dimension, suivant la directionalité.

Enfin, ces blocs sont insérés dans l'espace d'affichage voulu mais dans la plupart des cas, il est insuffisant (comme par exemple ce paragraphe qui ne peut tenir sur une seule ligne) : on passe alors à la progression par bloc, avec dans le cas des langues latines un retour à la ligne. On définit ainsi un mode d'écriture.

(v) Algorithme Bidi

L'algorithme Bidi permet de gérer les problèmes de mode d'écriture opposés, comme présenté dans cet exemple de dictionnaire Uyghur/Chinois/Russe, se lisant de droite à gauche pour le Uyghur et le Chinois, mais de gauche à droite pour le russe. Dans les trois cas, la progression de bloc se fait de haut en bas.

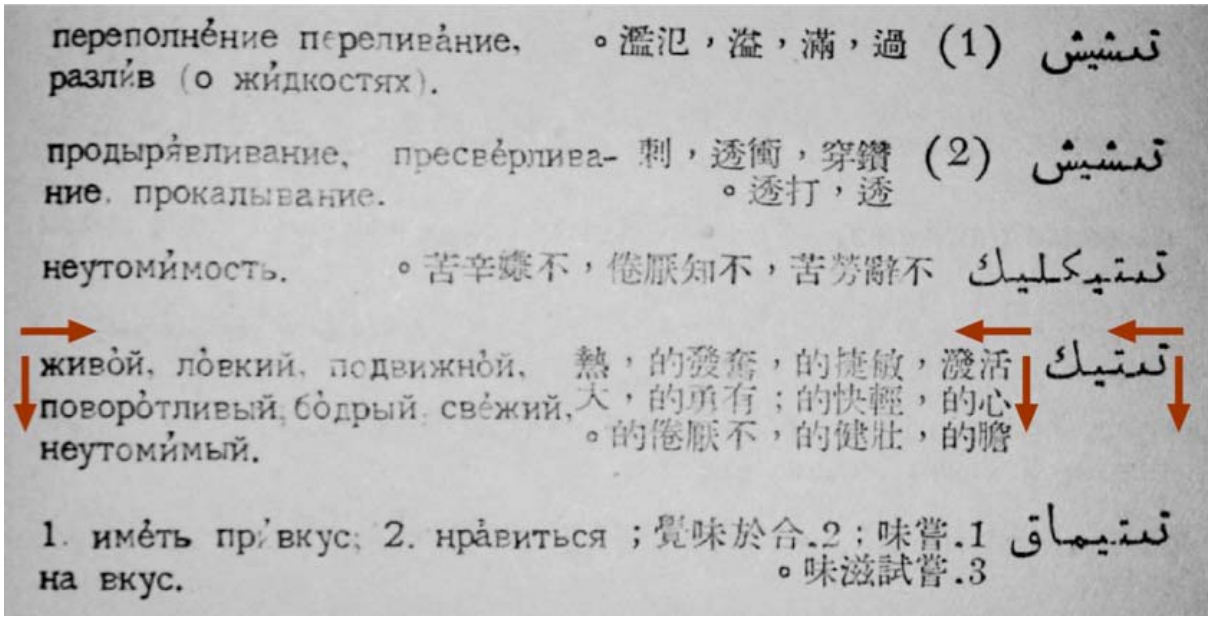


Figure 40: Dictionnaire Uyghur/Chinois/Russe

Même si le mode d'écriture est souple, comme illustré par les titres imprimés en directionalité de haut en bas sur la tranche de certains livres, la plupart des langues associent en général une directionalité et une progression de bloc pour un mode d'écriture qui leur est spécifique.

Figure 41: Exemple d'application de l'algorithme BiDi avec Pango

On parle donc de texte bidirectionnel dès que plusieurs modes d'écritures sont associés. Par extension, on peut aussi en parler pour le boustrophédon, qui alterne la directionalité pour chaque progression de bloc.

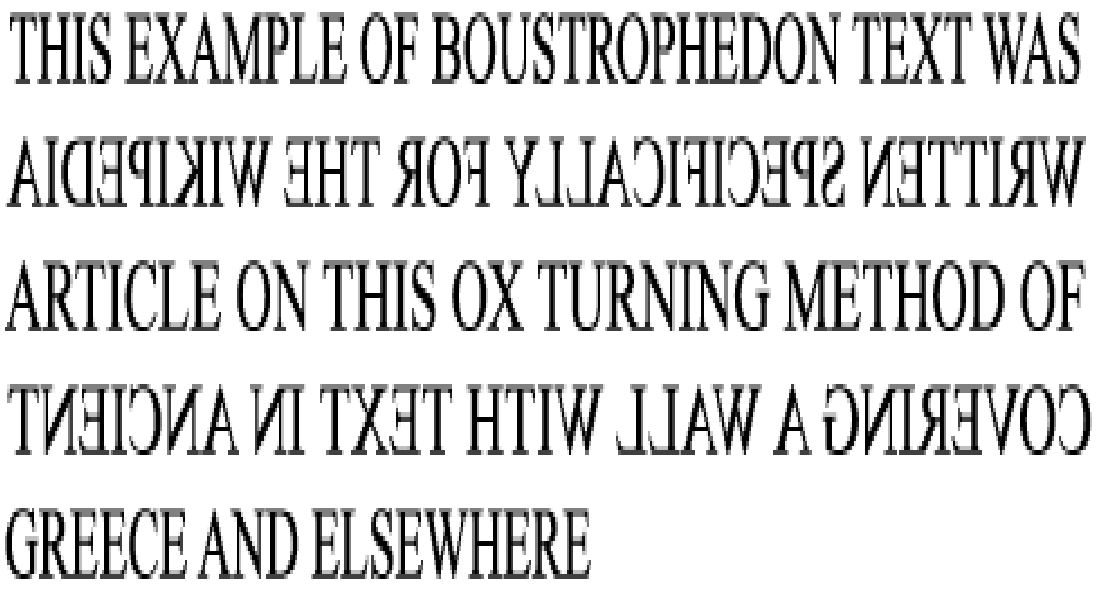


Figure 42: Exemple de boustrophédon

Mais comment l'algorithme BiDi fait-il pour gérer des cas aussi complexes ?

Comment peut-il gérer le mélange de textes se lisant dans des sens différents, même sur une seule ligne ?

(vi) Indicateurs pour BiDi

L'algorithme BiDi fonctionne en recherchant des séquences concaténées de caractères dits « forts », c'est à dire dont le mode d'écriture peut être deviné par le caractère (exemple : caractère hébreu : écriture de droite à gauche).

Ces séquences sont dites des courses. Les caractères dits faibles (exemple : les signes de ponctuations) qui sont situés entre ces courses héritent du mode d'écriture.

Au besoin, des indicateurs de directionalité (U+200E : gauche à droite, U+200F : droite à gauche) peuvent être insérés pour forcer une directionalité différente des caractères faibles.

Ces indicateurs peuvent être aussi temporaires (U+202A, U+202B), avec gestion d'une pile des directionalité (U+202C), et d'exceptions (U+202D, U+202E).

Il ne s'agit ici que d'une présentation très synthétique de l'algorithme BiDi, destinée à introduire la notion d'indicateurs, nécessaires pour toute fonctionnalité de citation de langues à mode d'écriture non identique.

BiDi est par ailleurs abondamment documenté *[2006-Davis-BiDi]* dans la littérature, et des implémentations existent dans de nombreux langages, du C à Haskell *[2001-Atkin-BiDi-Implementations].*

(vii) Séparateurs et autres caractères spéciaux

Il faut aussi noter la présence d'autres caractères, qui influent sur la mise en page et le rendu d'un bloc de texte..

Citons ainsi :

- U+2028 : nouvelle ligne et U+2029 nouveau paragraphe, destinés à ne plus laisser la gestion de la mise en page à un format spécifique d'un logiciel (retour chariot,
 ou <p> en HTML, etc) et à la considérer comme partie intégrale du texte

- U+00AD : séparateur de césure, qui peut être introduit à chaque endroit voulu dans un mot, mais n'être utilisé et affiché qu'en cas de passage effectif à la ligne, ou inversement être introduit lorsque le logiciel crée une césure automatique et supprimé tout aussi automatiquement lors d'opération de mise en page, pour bien le différencier du trait d'union ou du tiret

- U+200C : désunion sans largeur, pour interdire la ligature entre deux lettres qui seraient sinon liées (ff, fl...)

- U+200D : union sans largeur, pour forcer au contraire une ligature

- U+2060 : union de mots, pour éviter la césure entre deux mots, similaire en cela à l'espace non sécant de l'ISO 8859-1

Beaucoup d'autres caractères spéciaux existent pour gérer les cas particuliers de typographie mathématique qui ne seront pas détaillés ici.

D'un point de vue linguistique, il est plus intéressant de noter U+FFF9, pour l'insertion d'une annotation, U+FFFA, pour séparer les mots de cette annotation et U+FFFB pour indiquer la fin de l'annotation.

Ainsi, il est possible d'insérer des commentaires sémantiques dans un texte, ou de préserver des formes originales sans les afficher, même dans des cas particuliers comme l'annotation de vidéos *[2004-Brafford-Annotation].*

Signalors que des travaux actuels cherchent à améliorer les algorithmes de segmentation textuelle d'Unicode *[2008-Davis-Segmentation]*, à côté de la segmentation en ligne, déjà bien établie *[2006-Freytag-Breaking]*.

(viii) Limites liées au lien glyphe/caractères

La variation contextuelle, et l'obtention d'un glyphe à partir de plusieurs caractères ne peuvent logiquement s'appliquer qu'après application des modes d'écritures pour l'affichage.

Se pose alors un problème : la sélection à la souris, pour par exemple les fonctions de copier coller : si le rendu n'a pas conservé de lien entre les glyphes et les caractères, il ne sera pas possible de sélectionner indépendamment certaines parties des glyphes.

Pour illustrer le problème en français, sélectionner les trois derniers caractères du mot œuf, comportant la ligature oe, devrait être possible pour obtenir « euf », ou pour appliquer des fonctions de recherche de chaînes de caractères.

(ix) Conclusion

Il ne s'agit là aussi que d'une présentation très limitée du problème, destinée à introduire la notion de préservation des liens entre glyphe et caractère, nécessaire pour les opérations avancées de sélection.

Toutefois, il est possible de percevoir les apports d'Unicode dans le rendu de textes, afin que ce dernier ne soit plus dépendant du format d'un logiciel spécifique donné, mais conservé quelque soit le logiciel qui décode le texte.

Un problème apparaît alors : quelle est la juste séparation des compétences de chaque élément ? Où mettre la séparation entre fonctions et format du logiciel d'un côté (ex : gras, passage à la ligne), et format d'encodage textuel de l'autre ?

Il semble possible de considérer un texte comme une agrégation progressive de lettres : d'abord sous forme de mots, séparés par des espaces, puis sous forme de lignes, séparées par des retours à la ligne, puis enfin sous forme de paragraphes, séparés par des lignes vides. En cela, les codes Unicode de nouvelle ligne et de nouveau paragraphe semblent aussi justifiables que le code « espace », qui n'a pas à figurer dans le format du logiciel mais plutôt dans l'encodage textuel, car exprimant la volonté d'agrégation ou de séparation de l'utilisateur.

Toutefois, cette première question mérite un travail séparé, et est hors du propos de cette thèse. Elle introduit toutefois la notion d'indépendance la plus poussée possible du format sous-jacent afin de conserver l'aspect graphique.

Mais surtout, elle entraîne une autre question : comment est gérée toute cette complexité d'affichage Unicode ? N'y a-t-il pas besoin d'une coopération entre le moteur Unicode, qui applique la norme, et la police de caractère, qui supporte la norme plus ou moins bien comme précédemment présenté ?

2-4-5. Gestion de police

Vue cette dualité, et surtout le support incomplet par les polices, le moteur de rendu Unicode ne gère pas seul tous les problèmes précédemment présentés. Il se base sur des tables de typographie contenues dans les polices, précisant de quelle manière ces dernières supportent ces problèmes.

(i) Ancrage

En effet, pour des problèmes aussi simples que le positionnement d'accents, un endroit d'ancrage doit être signalé.

Dans le cas de langues comme le Vietnamien, supportant la présence de plusieurs accents sur chaque lettre, il est nécessaire de définir à quelles coordonnées spatiales, et de quelle manière, les accents sont positionnés « à la queue-le-leu », puisque chaque accent supplémentaire occupe de l'espace.

ệ	lettre minuscule e accent circonflexe lettre minuscule e + accent circonflexe
ẹ̀	lettre minuscule e + diacritique point en chef
ệ	lettre minuscule e accent circonflexe + diacritique point souscrit lettre minuscule e + accent circonflexe + diacritique point souscrit lettre minuscule e + diacritique point souscrit + accent circonflexe
ẹ̀	lettre minuscule e + diacritique point en chef + diacritique point souscrit lettre minuscule e + diacritique point souscrit + diacritique point en chef
ḛ́	lettre minuscule e tréma + diacritique accent aigu lettre minuscule e + diacritique tréma + diacritique accent aigu
ḛ́	lettre minuscule e accent aigu + diacritique tréma lettre minuscule e + diacritique accent aigu + diacritique tréma

Figure 43: Accentuation multiple de lettres

De même, le « i » en Lithuanien peut conserver son point lorsqu'un accent est rajouté : la police de caractère doit donc disposer de ces variations si le moteur Unicode veut les gérer.

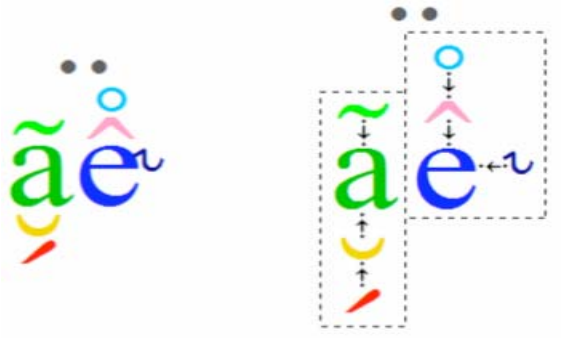


Figure 44: Positionnement relatifs des accents les uns par rapport aux autres

Dans le cas contraire, une stratégie de fonctionnement *a minima* doit être indiquée, pour que les différents accents puissent au moins déjà être affichables.

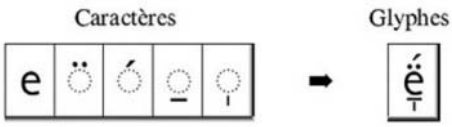


Figure 45: Séquence de caractères à traduite en glyphe

(ii) Interlignage

La conséquence d'un possible empilement horizontal est la gestion de l'interlignage : il ne faut pas que la ligne précédente empiète sur la ligne suivante. Il peut même parfois être nécessaire de réduire la taille d'un caractère, pour ne pas rendre le paragraphe visuellement difficile à lire.

Là encore, une police de caractère peut renseigner sur l'interlignage nécessaire selon les conditions rencontrées.

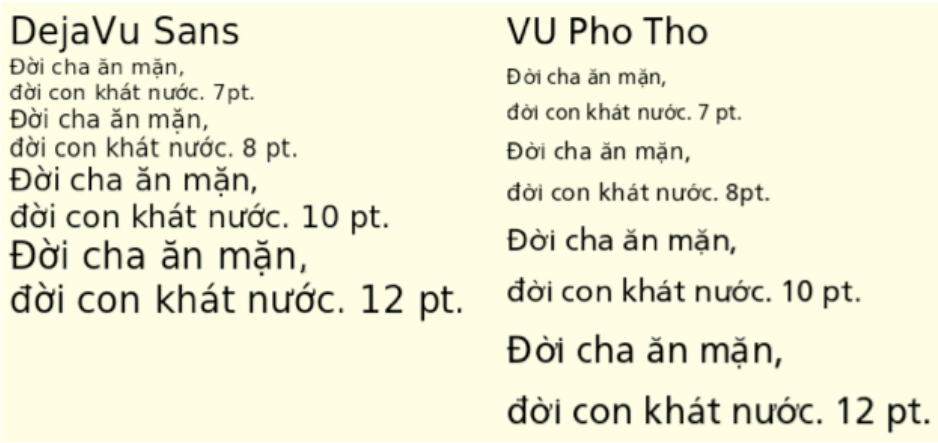


Figure 46: Interlignage recommandé par une police Vietnamienne

(iii) Conséquences sur les polices

Pour les problèmes précédemment présentés et leurs conséquences sur l'ancrage et l'interlignage, des renseignements supplémentaires, dépendant de la police, doivent être liés à cette dernière.

Dans ce domaine, les premiers apports ont été réalisés par Adobe pour ses polices PostScript Type 1, puis MultipleMasters : ces dernières ont introduit le concept d'une infinité de glyphes pour un caractère donné, en tenant compte de paramètres numériques.

Au delà des deux cas présentés, beaucoup d'autres problèmes existent, et vont être introduits ci-dessous. Ces approches d'Adobe ont donc été complétées par des approches modernes, gérant de manière plus extensive ces cas particuliers.

(iv) Approches actuelles : OpenType, AAT, Graphite

Basé sur TrueType, OpenType de Microsoft introduit des tables avancées pour gérer chaque problème. Apple a pour sa part fait évoluer le format TrueType GX en Apple Advanced Typography, ou AAT.

Graphite est une approche indépendante, implémentant les avancées technologiques de l'un comme de l'autre, avec des caractéristiques propres.

Parmi les tables avancées présentes dans les polices des approches actuelles, on peut notamment citer :

- pour OpenType : les tables de position des glyphes (GPOS) et de justification (GSUB)

- pour AAT : les tables d'attachement des accents (acnt), de variation des glyphes (gvar), de ligature (lcar) et de métamorphose (morx)

- pour Graphite : les tables Silf, Gloc, Glat

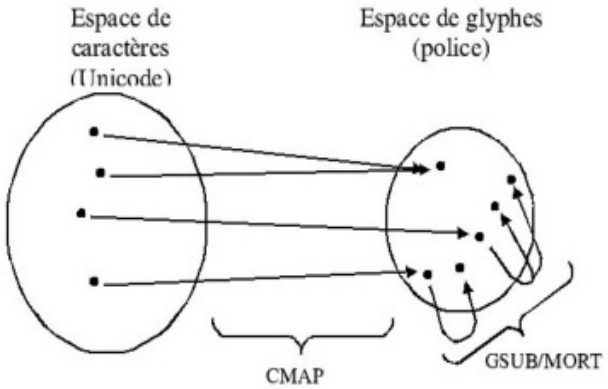


Figure 47: Application des tables pour l'association caractère/glyphe

Ces trois grands systèmes sont toutefois très différents.

(v) OpenType contre AAT

Il s'agit des systèmes les plus répandus, étant donné que Graphite n'est pas installé par défaut avec un système d'exploitation. La situation pourrait toutefois être amenée à changer, vu l'intérêt que de nombreux développeurs de logiciels libres portent à Graphite.

Actuellement, OpenType tend à s'imposer sur AAT, Apple supportant de plus en plus le format ouvert OpenType défendu par Microsoft et Adobe.

Une raison est la popularité de ce format, qui se traduit par la mise à disposition de nombreuses polices OpenType - plus que de polices AAT, alors que cette dernière technologie est pourtant plus ancienne.

La cause est la plus grande simplicité de création des polices OpenType.

Pour résumer, OpenType utilise des tables de manière associative, alors qu'AAT utilise ses tables pour manipuler une automate à état finis : OpenType ne peut être que déclaratif alors que AAT peut en plus être algorithmique.

En conséquence, la ligne de séparation entre compétences de la police et compétences du moteur Unicode varie : une police OpenType requiert moins d'information sur les écritures spécifiques, étant donné que ces informations sont gérées par le moteur Unicode (Uniscribe pour Windows).

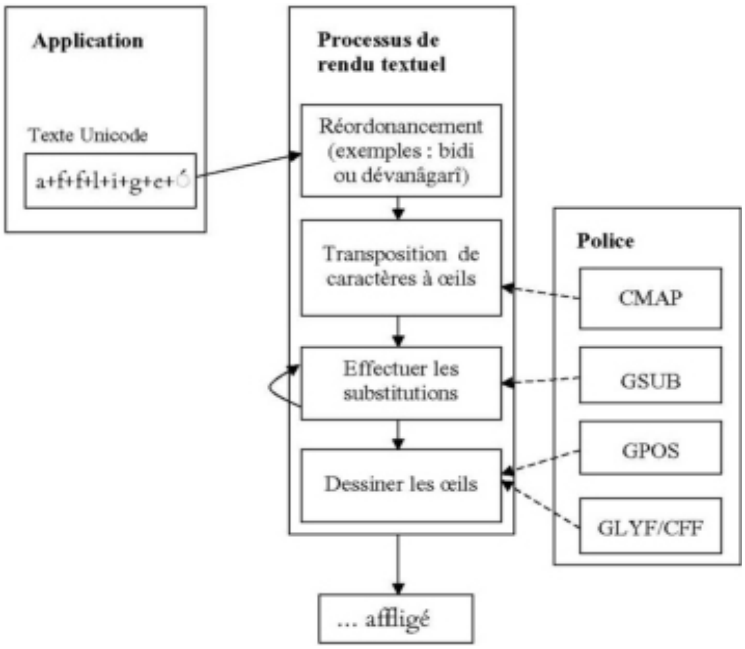


Figure 48: Exemple d'application des tables d'OpenType pour le rendu

Table	Description
CMAP	Correspondances entre caractère et glyphe
GLYF	Dessin TrueType des glyphes (sous forme vectorisée)
CFF	Programme de police PostScript (définit l'œil des glyphes)
GSUB	Substitution de glyphes : 1-1, 1-*n* ou *n*-1
GPOS	Déplacement des glyphes et positionnement des glyphes les uns par rapport aux autres sur les deux axes (X,Y).

Figure 49: Rôle des différentes tables d'OpenType intervenant pour le rendu

Elle est donc plus facile à créer, mais ne peut être utilisée pour des formalismes d'écriture complexe que si le moteur lui même sait gérer de tels formalismes.

Certaines limites dépendantes des approches sont à signaler, comme la limitation d'OpenType à 65 536 glyphes.

(vi) Nécessité d'un post-processeur

Dans tous les cas, les tables présentées pour ces trois approches sont dites de « typographie avancée », car elles permettent une gestion extrêmement fine de l'affichage du glyphe.

Pour cela, elles contiennent des instructions de conversion, à appliquer dans un certain ordre, ce qui concerne notamment les ligatures.

ct	practical	exact	objection	ſections	directions	ſubtract
ff	offended	offset	different	ſtaff	effect	affabrous
ffi	ſufficient	difficult	officers	affiance	chaffing	muffin
ffl	afflict	offlet	ruffle	affation	ſafflower	ſnaffle
fi	finding	beneficial	field	deficient	ſuperficies	confine
fl	chiefly	reflect	flower	flat	defly	rifling
ſ	eaſy	ſurvey	preſent	inſpects	alſo	uſes
ſh	ſhewing	ſhilling	publiſh	cruſh	lordſhip	waſh
ſi	curiouſity	ſince	beſides	buſineſs	deſign	conſider
ſl	aſleep	ſlope	ſluice	tranſlate	ſlight	iſle
ſſ	neceſſary	groſſly	aſſign	paſſing	poſſeſs	leſſer
ſt	firſt	ſtretch	inſtrument	moſt	waſte	diſtance

Figure 50: Exemples de ligatures : ff, ffl, etc.

L'ordre est important : les ligatures f d'ordre le plus élevé doivent être appliquées en premier : « f + f + l » doit être remplacé par la ligature « ffl » ; sinon si le remplacement « f + l » en ligature « fl » est appliqué d'abord.

Il faut alors une autre règle pour « f + fl », au risque de voir sinon les deux f non ligaturés. De même, les ligatures forcées par des caractères Unicode spécifiques, ou au contraire l'interdiction de ligature doit être gérée - mais il ne s'agit là que d'un cas simple.

Des cas beaucoup plus complexes existent, comme vu en arabe pour la variabilité contextuelle, qui peut aussi s'associer aux mécanismes de ligature, dans le cas par exemple du devanagari, où la suite de caractères codés présente un rendu textuel très différent et nécessite donc une parfaite définition du comportement des caractères inclus.

Des clarifications de leur fonction sont donc ainsi parfois proposées dans une temptative de simplification *[2004-Constable-ZWI]*.

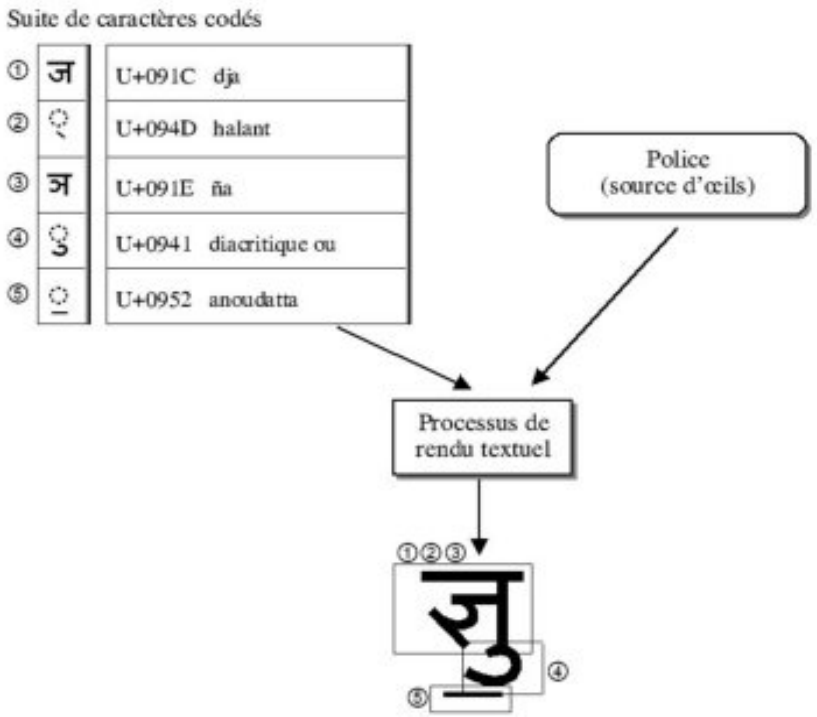


Figure 51: Rendu textuel complexe avec 5 éléments

Toutes ces instructions, que ce soit de ligature, de variation contextuelle, etc. sont appliquées par le moteur Unicode, qui se base pour cela sur les informations contenues dans ces tables de la police.

Comme déjà indiqué, AAT gère d'ailleurs les cas les plus complexes par un automate à état finis.

Ce dernier commande les tables morx et just, responsables respectivement de la substitution des glyphes et de la justification.

À titre d'exemple, la table morx renseigne des opérations dites de type :

- 0 pour le changement d'ordre des glyphes, nécessaire pour les suites complexes de consonnes de certaines langues du sud-est asiatique, où jusqu'à sept consonnes peuvent se succéder : si la dernière est un R, elle doit apparaître en premier

- 1 pour la substitution contextuelle, comme par exemple lors de l'entrée de chiffres séparés par des symboles mathématiques : l'astérisque * est remplacée par un symbole multiplier x

- 2 pour les ligatures, comme présenté avec l'exemple du ffl

- 4 pour les substitutions de glyphes, de manière automatique, et non contextuelle

- 5 pour les insertions de glyphes, avant ou après le glyphe courant, nécessaires parfois en arabe

La table « just » sert à disposer les glyphes de manière optimale sur un espace donné, en gérant plusieurs cas de justification non détaillés ici car hors propos.

La fonctionnalité nécessaire à Unicode, comme présentée dans les pages précédentes, nécessite donc ce partage des compétences entre le moteur d'affichage, qui exécute des requêtes selon la grammaire prédéfinie des tables, et les tables qui contiennent ces instructions, différentes pour chaque police.

2-4-6. Conclusion

Cette étude du support informatique des encodages est très générale et correspond plus à un cahier des charges qu'à un état de l'art, vu la variabilité qui existe dans l'implémentation de la norme Unicode, et surtout les limites des systèmes informatiques actuels.

Il est nécessaire d'étudier un exemple particulier de système d'exploitation, pour voir plus précisément comment sont supportés les encodages précédemment mentionnés et les propriétés précédemment présentées - prenons GNU/Linux.

2-5. Support informatique avec GNU/Linux

2-5-1. Introduction

Le choix du système GNU/Linux pour illustrer le fonctionnement des niveaux de supports par les systèmes d'exploitation est arbitraire. Néanmoins, vu le peu de différences dans les implémentations de systèmes modernes, il est facilement reproductible, et surtout propice à l'expérimentation [2004-Aznar-PDA_Linux].

La gestion du clavier, qui n'a d'ailleurs volontairement pas fait l'objet d'un chapitre précédemment, va d'ailleurs être étudiée plus en détail, vu la généralisation du système à multiples articulations.

Des découpages plus fins seraient possibles, mais par facilité, on peut opposer un mode dit « console », et un mode dit « graphique ».

2-5-2. Mode console

Malgré les affirmations contraires des auteurs de logiciels, plusieurs modes persistent dans les systèmes d'exploitation, dont un mode console « textuel ».

Le support de langues dans ce mode a fait l'objet de travaux spécifiques, proches de cette étude, dans le cas par exemple des langues indiennes *[2001-Ratheesh-Linux_Console_Indian].*

(i) Séquence de chargement

Le mode console correspond à toutes les étapes qui précèdent en général le mode graphique. On peut y retrouver successivement : le bios, le bootloader (Grub ou Lilo dans le cas de Linux), le chargement du noyau, puis le lancement des divers scripts et démons de sous-contrôle de tâches.

(ii) Travail du noyau

Le travail du noyau en ce qui concerne le support des langues écrites est de :

- recevoir un code clavier, dit scancode, par exemple 0x1e et 0x9e

- transformer ce scancode en keycode, par exemple 30 et 158

- faire correspondre ce keycode à un caractère, par exemple le caractère 113 en ASCII ou ISO-8859-15, par le biais d'une keymap

- utiliser une police de caractère pour afficher le glyphe correspondant, dans ce cas la lettre q, ou sinon déclencher une action, par exemple changer de console virtuelle pour la séquence ALT+F1.

(iii) Correspondance scancode/keycode

Les scancodes correspondent à l'aspect matériel du clavier : ils sont globalement normalisés, bien que des touches exotiques puissent correspondre à des scancodes inhabituels (ex : touches dites « multimédia »), qui devront donc être déclarés par le biais de setkeycodes, comme par exemple "setkeycode e033 132".

Ici, le keycode 132 est ainsi déclaré.

(iv) Les keycodes

Les keycodes correspondent à une déclaration arbitraire dans le noyau linux, réalisée dans le fichier include/linux/input.h, comme par exemple pour les touches de fonction, de F1 à F12.

```
#define KEY_F1     59
#define KEY_F2     60
#define KEY_F3     61
#define KEY_F4     62
#define KEY_F5     63
#define KEY_F6     64
#define KEY_F7     65
#define KEY_F8     66
#define KEY_F9     67
#define KEY_F10    68
#define KEY_F11    87
#define KEY_F12    88
```

Figure 52: Keycodes des touches de fonctions

On constate que la séquence n'est pas continue entre F10 et F11. Cela est dû aux contraintes historiques de l'architecture PC, qui ont redéfini ultérieurement de nouvelles touches de fonction à partir d'usages différents.

Le caractère arbitraire de la table de correspondance entre un keycode et un rôle donné est lié à une architecture. Ainsi par exemple sur un ordinateur Sun, les codes précédents seront différents. Ceci est dû à l'existence de touches spécifiques suivant les architectures (exemple Sun: Copier, Coller...), mais les claviers à disposition inhabituelle, destinés à une même architecture, respectent toutefois les conventions de keycodes.

Certains claviers utilisent exclusivement des combinaisons de touches pour générer les keycodes standard *[1990-Buxton-Chord_Keyboards]* afin de réduire le nombre de boutons physiques ou d'accélérer la saisie.

Figure 53: Clavier de type frogpad

(v) Définition de l'effet des keycodes

La keymap lie les keycodes à des keysyms. Ces derniers, normalisés dans le noyau Linux, correspondent soit à des caractères, soit à des actions.

Il est à remarquer la double articulation arbitraire entre un keycode et un keysym: ainsi, pour que le keycode 88 corresponde à F12, d'abord il est défini sous le nom de KEY_F12 la valeur 88, puis cette valeur se retrouve affectée au keysym F12.

L'intérêt de cette double articulation est évident vue la différence entre claviers selon les langues : la touche à gauche du 1, au dessus de la touche tab, sert à obtenir un tilde (~) sur un clavier états-unien, et un carré (²) sur un clavier français.

D'autres configuration existent de la même manière, en général une par langue.

Parfois, l'architecture peut donner des contraintes supplémentaires.

Ainsi, le projet OLPC d'ordinateur à 100$ pour l'éducation des enfants ne définit pas de touches de fonction, ni de pavé numérique ou de pavé d'insertion, tout en suivant les modifications « linguistiques » de disposition des caractères sur les touches.

Il convient donc de déclarer la position de cette touche sur le clavier, de lui affecter un keycode, et des keysyms possibles. La position de cette touche sur le clavier ne doit pas être reliée au keysym, étant donné qu'elle peut changer.

Figure 54: Clavier OLPC états-unien

On peut ainsi constater entre les claviers brésiliens et états-uniens de l'OLPC que les position physiques sont identiques, mais que les arrangements diffèrent.

Figure 55: Clavier OLPC brésilien

(vi) Actions, table des strings

Les actions sont directement déclarées et exécutées, comme par exemple :

keycode 127 = Last_Console

Les keysyms correspondant à des actions sont normalisés dans le noyau linux: le keycode 127, qui correspond en général à la touche « menu windows », située contre la touche contrôle de droite, servira à retourner à la console virtuelle précédente.

Mais les séquences de caractères peuvent aussi être interprétées par les logiciels comme des actions. Ainsi dans l'exemple précédent, F10 correspond à une séquence de caractères normalisée par l'ANSI : échap [2 1

Cette correspondance est déclarée dans la table des chaînes ("*strings*" en anglais*)*, où \033 correspond au code ASCII de ESC qui marque le début de la séquence, et la tilde indique la fin de la séquence.

```
string Home = "\033[1~"
string Insert = "\033[2~"
string Remove = "\033[3~"
string End = "\033[4~"
string PageUp = "\033[5~"
string PageDown = "\033[6~"
string Macro = "\033[M"
string Pause = "\033[P"
string F1 = "\033[[A"
(...)
string F10 = "\033[21~"
(...)
string F35 = "\033[49~"
```
Figure 56: Chaînes des touches de fonction et d'action

Certains logiciels sous GNU/Linux, comme MC, reconnaîtront cette séquence de F10 et exécuteront la fonction quitter.

(vii) Correspondance entre keycodes et caractères

Le plus fréquemment, les caractères sont des symboles isolés. Dans ce cas, la keymap, permet aussi de déclarer des modifieurs.

Par exemple pour le clavier français [1997-Aznar-Francophones]:

```
keycode  27 = +dollar           +sterling       +currency       cent

keycode  28 = Return
        alt     keycode  28 = 0x080d
        altgr   keycode  28 = Linefeed
(...)
keycode  42 = Shift
(...)
keycode  58 =  Caps_Lock
```

Figure 57: Extrait de la keymap fr-latin9.map

Le keycode 27 donnera le caractère $ (dollar) en mode normal, £ (livre sterling) en second mode, ¤ (monnaie) dans le troisième, et ¢ (centime) dans le quatrième.

Le second mode est le mode majuscule dit Shift, le troisième mode le mode AltGr, et le quatrième mode le mode CtrlGr (rarement utilisé).

La déclaration de keycodes correspondant à ces modes sert à basculer temporairement vers ces modes. Un verrouillage peut aussi être obtenu, comme dans le cas de la touche Verr Maj, sous la touche Tab, aussi dite Caps Lock.

Le préfixe + peut servir à indiquer que le passage d'un mode à l'autre doit être géré même en cas de verrouillage dans un mode. Ainsi, la touche Verr Maj pressée, sous la touche Tab, l'appui sur la touche $ permettra d'obtenir directement un £, alors qu'en l'absence du préfixe plus, le caractère $ serait obtenu.

Il s'agit d'une syntaxe standardisée, mais il est aussi possible d'effectuer autrement les attributions. Le keycode 28 permettra d'obtenir l'action « entrée » en mode normal et en second mode (déclaration implicite de l'équivalence), le caractère 0x080d si alt est pressé en même temps, et l'action « retour chariot » en troisième mode. Par la déclaration de alt, on se rend compte que la séparation en quatre modes est partielle : d'autres modes implicites existent. Les touches servant à les obtenir sont dites des modifieurs. On peut séparer par exemple Alt, Control et Verr num. La séquence ctrl-d, qui correspond à l'action « fin de saisie », pourra ainsi être redéclarée au besoin. Elle est implicite avec la touche D.

(viii) Tables de composition

Il est à noter qu'un seul caractère donne un seul glyphe, à l'exception près des tables de composition qui associent à plusieurs keycodes un seul caractère, comme ê pour ^ suivi de e.

Ces tables de composition sont déclarées à part.

```
compose '`' 'O' to 'Ò'
compose '`' 'o' to 'ò'
compose '\'' 'O' to 'Ó'
compose '\'' 'o' to 'ó'
compose '^' 'O' to 'Ô'
compose '^' 'o' to 'ô'
compose '~' 'O' to 'Õ'
compose '~' 'o' to 'õ'
compose '"' 'O' to 'Ö'
compose '"' 'o' to 'ö'
compose '/' 'O' to 'Ø'
compose '/' 'o' to 'ø'
compose '-' 'o' to 'º'
compose '-' 'O' to 'º'
```

Figure 58: Table de composition de la lettre o

Il est particulièrement intéressant de remarquer que la keymap ne déclare pas à quelle norme elle obéit en général. Il est possible de le faire en rajoutant en toute première ligne du fichier : charset "iso-8859-15"

En l'absence de déclaration de correspondance, il est tout à fait possible d'utiliser une table de caractère avec une mauvaise police.

Reprenons l'exemple de l'ISO-8859-1 et de l'ISO-8859-15. Parmi les différences figure la gestion des fractions : les codes correspondants par exemple aux fractions en ISO-8859-1 ont été réattribués en ISO-8859-15 aux caractères manquants pour les langues européennes.

Afficher la keymap ISO-8859-15 en ISO-8859-1 donne une mauvaise interprétation des codes.

```
# Symbole euro
#compose '-' 'e' to '¤'
#compose '-' 'E' to '¤'
#compose '=' 'e' to '¤'
#compose '=' 'c' to '¤'
#compose '=' 'E' to '¤'
#compose '=' 'C' to '¤'
#compose 'e' '=' to '¤'
#compose 'c' '=' to '¤'
#compose 'E' '=' to '¤'
#compose 'C' '=' to '¤'
# S macron
#compose '^' 'S' to '¦'
#compose '^' 's' to '¨'
#compose '^' 'Z' to '´'
#compose '^' 'z' to '¸'
#compose 'v' 'S' to '¦'
#compose 'v' 's' to '¨'
# Z macron
#compose 'v' 'Z' to '´'
#compose 'v' 'z' to '¸'
# Ligature OE majuscule
#compose 'O' 'E' to '¼'
#compose 'O' 'e' to '¼'
# Ligature oe minuscule
#compose 'o' 'e' to '½'
# Y tréma majuscule
#compose '"' 'Y' to '¾'
#compose 'I' 'J' to '¾'
#compose '"' 'Y' to '¾'
```

Figure 59: Table de composition spécifique à l'ISO-8859-15 affichée en ISO-8859-1

On comprend facilement que la séquence "Y devrait afficher Ÿ, mais comme l'ISO-8859-1 utilise le code correspondant pour la fraction 3/4, cette dernière est affichée.

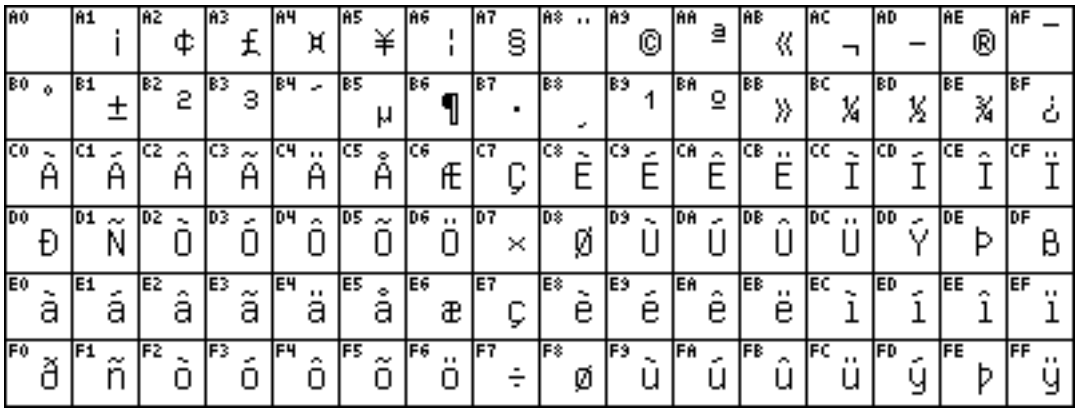


Figure 60: Table ISO-8859-1

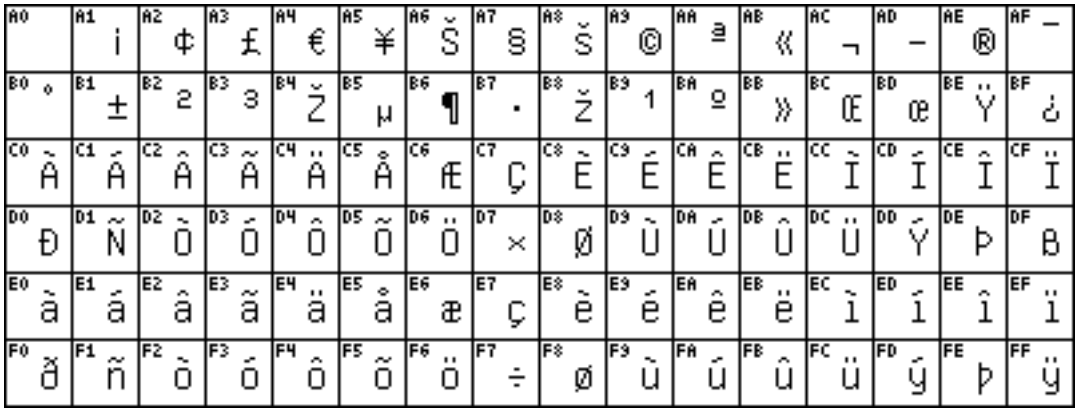


Figure 61: Table ISO-8859-15

Il s'agit d'une illustration des problèmes de compatibilités entre les codes, que nous allons maintenant étudier plus en détail avec les tables de correspondance.

(ix) Table de correspondance Unicode

Cette dernière a un rôle extrêmement important pour les polices non ISO-8859-1, étant donné la compatibilité prévue pour les 255 premiers caractères entre Unicode et ISO-8859-1.

En effet, si une police correspondant à une norme autre que ISO-8859-1 ou Unicode doit être utilisée, comment gérer les 255 premiers caractères ?

Dans le cas des ISO-8859, les 127 premiers caractères seront identiques, mais les suivants peuvent différer - ce qui est d'ailleurs tout l'intérêt des ISO-8859 : 127 caractères pour les besoins spécifiques de chaque groupe de langues.

Le rôle de la table de correspondance Unicode est de gérer ce problème : à chaque glyphe de la police, elle associe les codes Unicodes correspondants : ainsi, le 250e glyphe de la police peut correspondre au 250e code Unicode, ou ne pas y correspondre et correspondre plutôt au 320e, le 321e glyphe peut lui correspondre au 250e code Unicode, etc.

Étudions chacun de ces cas. La première possibilité est de ne pas tenir compte du problème, et de ne pas ajouter de table de correspondance Unicode. Dans ce cas, le système Linux ne saura pas afficher les fichiers textes comprenant des caractères Unicode au delà des 127 premiers caractères. Pour les fichiers encodés en ISO-8859, il affichera un glyphes dépendant de la position de ce dit glyphe dans la police. Ainsi, le code 250 fera apparaître le 250e glyphe. Et donc, selon que l'on soit dans un encodage ISO-8859-1 ou 2, (...) ou 15, le 250e glyphe sera différent.

Regardons maintenant le cas où une table de correspondance Unicode est présente. Cette fois, les fichiers contenant des caractères Unicode peuvent être affichés, vu la correspondance de codes qui indiquent quel glyphe afficher au delà des 127 premiers caractères. Par contre, l'ordre des 127 caractères suivants de la police détermine le résultat de l'affichage d'un fichier encodé en ISO-8859 : de même que dans l'exemple précédent, le code 250 fera apparaître le 250e glyphe, différent selon l'encodage ISO-8859 retenu.

Pour mieux comprendre ces notions, prenons l'exemple du symbole monétaire international (164e caractère, soit en hexadécimal A4), dont le code ISO-8859-1 correspond en ISO-8859-15 au symbole Euro, et regardons l'exemple des polices dites « lat9 », pour l'ISO-8859-15 :

- les lat9 sans suffixe ne contiennent pas de table de correspondance Unicode, ce qui fait que tous les symboles monétaires internationaux apparaîtront comme des € par effet de substitution.

- les lat9u incluent une table de correspondance Unicode, qui contient pour le 164e caractère:

0xA4 U+20AC

Ainsi, il est dit que le 164e caractère correspond au code Unicode U+20AC, défini comme le symbole Euro (€).

- les lat9v incluent une table de correspondance Unicode, et les caractères sont encodés dans l'ordre officiel, comme pour les lat9 sans suffixe :

0xA4 U+00A4

Ainsi, il est dit que le 164e caractère correspond au code Unicode U+00A4, défini comme le symbole monétaire international,

- les lat9w incluent une table de correspondance pour forcer dans tous les cas la substitution :

0xA4 U+00A4 U+20AC

Il est alors dit que le 164e caractère peut être indifféremment utilisé pour représenter le code Unicode U+00A4 et U+20AC, ce qui est certes incorrect, mais a l'avantage d'afficher le symbole Euro aussi bien dans les textes encodés en ISO-8859-15 qu'en Unicode, au détriment bien sur de l' ISO-8859-1

Ceci peut poser des problèmes pour les fichiers issus d'un ordinateur sous Windows ; le cp1252 ou le latin1 sont « incompatibles » avec le latin9, mais ce dernier a l'avantage de fonctionner tout de suite pour gérer l'Euro sous linux.

On comprend donc mieux le rôle de la table de correspondance Unicode pour une police de caractère : elle sert à déterminer quel glyphe sera affiché dans quel cas : c'est l'équivalent (en plus simple) d'un moteur Unicode.

Un autre point est à noter, qui justifie les tables de correspondance Unicode : le mode texte dépend des capacités de mémoire de la carte graphique, qui ont été standardisées à l'issue des normes précédemment présentées.

Ainsi, la mémoire disponible pour le mode texte est inférieure à ce qui est nécessaire pour stocker l'intégralité de l'espace Unicode.

(x) Conclusion

Le niveau de détail des paragraphes précédents est nécessaire pour en tirer des conclusions pratiques dans le cadre d'une informatisation de la langue des signes.

En effet, on perçoit déjà que la double articulation est un élément de base pour contenir une grande variété de caractères (ce qui est le cas de SW) sur un clavier.

Le rôle de la table de composition Unicode apparaît aussi de manière beaucoup plus claire : elle sert à faire, ou à défaire, les compatibilités.

À travers l'illustration des erreurs en cas d'utilisation du Latin 1 au lieu du Latin 9, et des problèmes existant dans les tables Unicode des polices lat9, on comprend que le codage de la langue des signes ne peut se concevoir comme un problème isolé, mais doit être mis en relation avec les autres encodages existants.

On peut s'apercevoir que des encodages non planifiés posent des problèmes similaires d'implémentation, comme c'est déjà le cas pour les « polices » de HamNoSys v2 et v4, qui entrent en conflit avec elles mêmes et avec les caractères de base de l'ISO-8859-1 *[2005-Bray-Dictionnary].*

Pour aller plus loin dans l'exemple, on va maintenant illustrer les concepts précédemment présentés de manière très pratique mais sommaire - plus de détails peuvent être obtenus dans *[1995-Brouwer-Keyboard].*

2-5-3. Mise en œuvre pour le mode console

(i) Chargement d'une table de clavier (keymap)

Le chargement d'une table de clavier se fait avec la commande loadkeys. Ainsi, pour charger la table d'un clavier français de France, en mode ISO-8859-1 :

loadkeys fr-latin1

Pour charger la table d'un clavier français de France, en mode ISO-8859-15 aussi abrégé en « latin 9 », afin de disposer du support de l'Euro :

loadkeys fr-latin9

Pour changer la table d'un clavier danois en mode ISO-8859-1 :

loadkeys dk-latin1

En général, la keymap comprend aussi la table de composition associée précédemment présentée.

Il est important de noter de la présence de touches dites « mortes ».

Il s'agit de touches qui n'ont pour fonction que de permettre l'utilisation de la table de composition : citons par exemple le tréma et l'accent circonflexe.

(ii) Utilisation des touches mortes

On recense :

- sur les claviers Belges et Français : l'accent aigu en AltGr de 1 pour les Français et en AltGr de ù pour les Belges, l'accent grave en AltGr de 7 pour les français & en AltGr de carré/cube pour les Belges, le tréma et l'accent circonflexe tous deux à côté du P

- sur les claviers Suisses et États Uniens : les accent aigus, graves, circonflexes, les trémas et la tilde

Dans tous les cas, les touches mortes fonctionnent de la même manières

(iii) Utilisation du clavier

Plusieurs dispositions de claviers sont utilisées sous Linux.

Les illustrations suivantes [1997-Aznar-Francophones] montrent les configurations les plus standards.

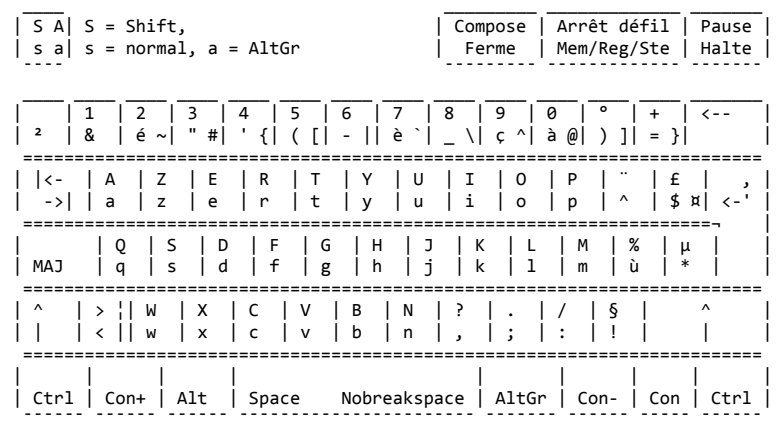


Figure 62: Clavier français ISO-8859-1 habituel

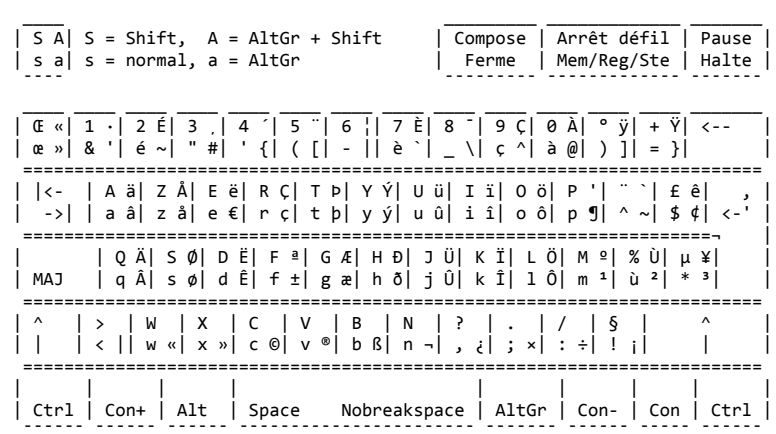


Figure 63: Clavier français étendu en ISO-8859-15

```
Shift  = « Majuscule, sélection de niveau 2 »
Caps   = « Verrouillage des majuscules »
AltGr  = « Alternative, sélection du niveau 3 »
CtrlGr = « Contrôle, sélection de groupe »
```

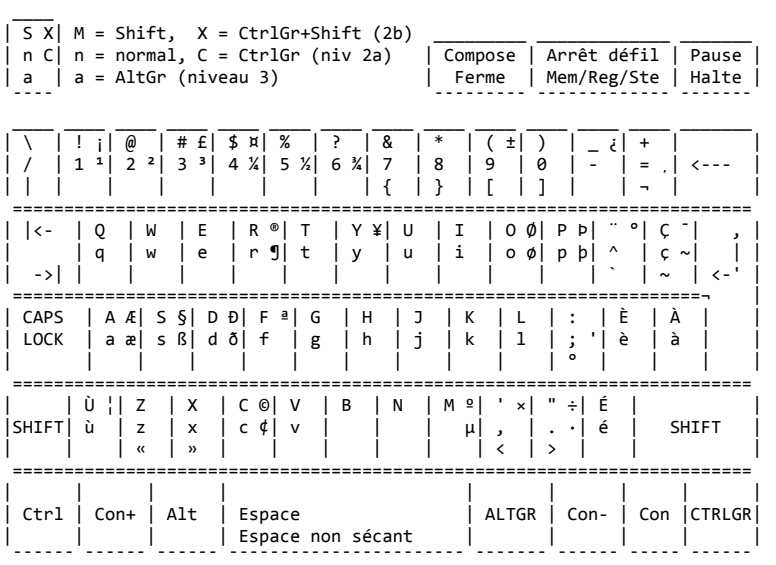


Figure 64: Clavier canadien normalisé CAN/CSA Z243.200-92

(iv) Chargement d'une police de caractères

Pour charger une police de caractère ISO-8859-1 de taille 16 points :

setfont lat1-16.psf

Pour charger une police ISO-8859-15 comprenant la table de correspondance unicode dite « v », présentée plus haut:

setfont lat9v-16.psf

Le chargement d'une police de caractère dans les versions précédentes du système GNU/Linux ne changeait que les glyphes.

Il est possible actuellement d'associer les glyphes et les tables de correspondances Unicode, qui peuvent aussi encore exister sous forme séparée en fichier .uni à charger avec loadunimap :

setfont lat9-16.psf

loadunimap lat9v.uni

(v) Locale

Une fois l'entrée et la sortie possible avec l'encodage choisi et la disposition du clavier voulue, il est possible de demander au système GNU/Linux de ne s'exprimer que dans la langue correspondante.

Mais en fait, outre la langue d'affichage des messages, tout un ensemble de paramètres doit être modifié. Ainsi l'ordre de classement de cette langue devra être respecté lors des opérations de tri : par exemple en français les lettres accentuées sont situées au même niveau que les lettres non accentuées, alors que strcmp rangeant par code met les lettres accentuées après le z.

Cet ensemble de paramètres modifiés définit une locale. De nombreuses fonctions standard sont impactées par les locales : citons par exemple int strcoll (const char *s1, const char *s2) qui est un équivalent de strcmp tenant compte des locales.

Pour régler la locale, il existe différente variables à exporter, avec chacune une fonction spécifique. Le format standard est langue[_PAYS[.ENCODAGE]][@variante]

Les crochets dénotent l'optionalité, par exemple on reconnaît: 'fr', 'fr_BE', 'fr_CH.ISO-8859-15', no@bokmaal, no@nynorsk,...

Il est ainsi possible de gérer plusieurs langues par pays, voire plusieurs variantes de la même langue d'un pays, et ce indépendamment pour chaque variable.

Citons comme variables reconnues :

- LC_COLLATE définit les équivalences de caractères pour les comparaisons (æ peut être équivalent à ae), pour les ligatures et pour les césures.

- LC_CTYPE définit les caractères affichables

- LC_MONETARY définit le format et le symbole de la monnaie utilisée

- LC_NUMERIC définit le format numérique : regroupement, séparateur des nombres justement dits « à virgule »...

Par exemple un million peut s'écrire 1'000'000 en notation anglaise, 1E6 en notation scientifique, 1 000 000 en notation française, un demi soit 0.5, 5E-1 ou 0,5

- LC_MESSAGES ou LANG définit la langue des messages

- LC_TIME définit le format de la date, les noms des jours et des mois

- LC_ALL donne la valeur par défaut des variables précédentes : si une LC_ n'est pas définie, LC_ALL est prise en compte

Plusieurs choix sont possibles. LANGUAGE liste des locales par ordre de préférence séparées par deux points (fr:es:en) : ainsi s'il n'y a ni version française, ni version espagnole d'un message d'erreur, la version anglaise sera affichée.

Illustrons ces notions par des exemples, où l'on essaye de voir la taille d'un fichier n'existant pas, en demandant à chaque fois dans une langue différence.

```
bash# export LANGUAGE=es_ES
bash# ls fichier_n_existant_pas
ls: fichier_n_existant_pas: No existe el fichero o el directorio
bash# export LANGUAGE=de_DE
bash# ls fichier_n_existant_pas
ls: fichier_n_existant_pas: Datei oder Verzeichnis nicht gefunden
bash# export LANGUAGE=en_US
bash# ls fichier_n_existant_pas
ls: fichier_n_existant_pas: No such file or directory
bash# export LANGUAGE=fr_FR
bash# ls fichier_n_existant_pas
ls: fichier_n_existant_pas: Aucun fichier ou répertoire de ce type
```

Figure 65: Réponse du programme dans la langue demandée

Toutefois, le mode console est optionnel : à des fins de légèreté ou pour une utilisation spécifique (linux embarqué), il peut être supprimé.

Dans ce cas, seuls les scancodes sont gérés par le noyau, et passés au système X-Windows.

2-5-4. Mode graphique

Différents X-Windows existent : citons principalement XFree et X.org. Mais le mode graphique fonctionne sur des principes identiques au mode console : il sépare de manière similaire les options de configuration.

(i) Travail de X-Windows

En s'exécutant soit en parallèle, soit en remplacement du mode console, iX exécute les mêmes fonctions pour gérer les langues écrites :

- réception d'un scancode passé par le noyau
- transformation de ce scancode en keycode selon l'architecture
- mise en correspondance de ce keycode avec un emplacement physique sur un modèle de clavier
- exécution de la fonction correspondante à cet emplacement selon le type de clavier, lui même dépendant du pays.

Ces différentes étapes sont configurées dans le fichiers XF86Config de la manière suivante :

```
XkbKeycodes    "xfree86"
XkbSymbols     "fr-latin9(pc105)"
XkbModel       "pc105"
```

Il est à noter que le paramètre optionnel XkbCompat sert à définir les sous variations de clavier : ainsi, préciser "swapcaps" active l'option inversant la fonction de la touche Contrôle et de la touche Verr. Maj.

Pour toutes ces options, les paramètres peuvent être augmentés ("*heritence*") en séparant les options avec une barre | ou remplacées avec un plus +.

On note donc par rapport au mode console de légères différences, dont le rajout d'une articulation sur l'emplacement physique du clavier, et la meilleure gestion des différences d'architectures.

(ii) Réception des scancodes

En effet, dès la première ligne de la configuration précédemment présentée figure l'option XkbKeycode "xfree86" : La valeur xfree86 est utilisée sur PC. Un Apple iBook utilisera pour sa part XkbKeycodes "macintosh". Une station IRIS de Silicon Graphics utilisera sgi/iris, etc.

Les fichiers se trouvent dans /usr/X11R6/lib/X11/xkb/keycodes : selon l'architecture, ils associent les keycodes aux scancodes.

Dans le noyau linux, cette option est directement compilée et ne peut être modifiée que par une table de clavier. Il faut donc une keymap par architecture et par pays, ce qui revient à dupliquer les configurations.

(iii) Association des emplacements et des keycodes

La disposition du clavier est précisée par le paramètre XkbModel, et peut être passée entre parenthèse avec l'option XkbSymbols. Ici, la disposition est pc105 : 105 touches d'un clavier PC standard sont donc définies les unes par rapport aux autres.

Typiquement sur un PC sont définis :

- une ligne de touche de fonction (ESC, F1 ...),
- la ligne de touches de chiffres,
- trois lignes de touches de lettres,
- une ligne de touches de contrôle contenant la barre d'espace

Des pavés supplémentaires sont aussi définis :

- pavé d'édition (Insert, Suppr, Début, Fin, Pg Haut, Pg Bas),
- pavé numérique (Verr Num, 7 8 9 ...)
- flèches,
- touches de contrôle (Impr écran/Syst, Arrêt défil, Pause/Attn)

Pour chaque élément, un numéro d'abcisse et d'ordonnée, correspondant éventuellement à un nom d'alias, établissent la correspondance entre un emplacement physique et un keycode.

Ainsi, AE00 alias TLDE correspond à la première touche en partant de la gauche de la cinquième ligne en partant du bas, touche comportant le caractère tilde ~ sur un clavier états-unien, et le caractère carré sur un clavier français. Le keycode associé est 49.

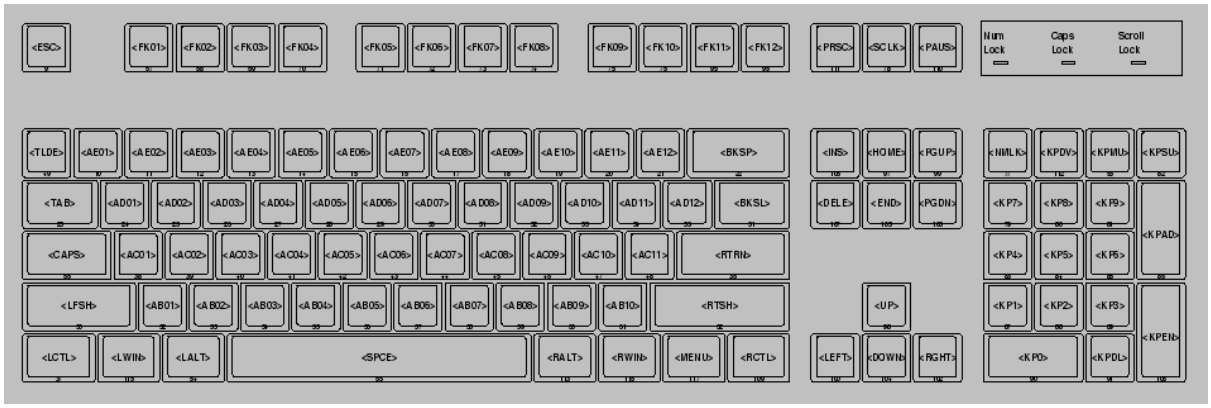


Figure 66: Libellés des touches sur un clavier de type PC sous X-Windows

Une définition plus précise de la géométrie est aussi possible par le paramètre XkbGeometry : à chaques emplacements physiques sont alors associées leurs coordonnées spatiales sur une échelle cartésienne millimétrique, afin de pouvoir afficher à l'utilisateur une image correspondante à la disposition physique de son clavier. Cette option est donc très peu utilisée, car uniquement cosmétique.

(iv) Description du comportement d'un emplacement

Une fois l'emplacement physique décrit, son comportement peut être défini de manière identique, quelque soit la langue et donc la disposition du clavier.

Cette fois, les fichiers symbols servent à définir ce comportement, sous forme de liste de rôles possibles.

Un clavier français habituel en ISO-8859-1 aura pour la touche AE00 :

key <TLDE> { [twosuperior] }

Un clavier états-unien aura :

key <TLDE> { [quoteleft, asciitilde] }

Un clavier français complet en ISO-8859-15 aura :

```
key <TLDE>  {    [       onehalf,      onequarter      ],
                 [  guillemotleft,  guillemotright    ]     };
```

Dans le premier cas, quelque soit le modifieur, le carré sera affiché ; dans le second cas, la quotation (« ` ») sera utilisée en mode normal et la tilde (« ~ ») en mode majuscule ; alors que dans le dernier cas, en mode normal la ligature oe minuscule sera utilisée (« œ »), ainsi qu'en mode majuscule (« Œ »), le AltGr servant à faire les guillemets français ouvrants ("«") et le AltGr majuscule les guillemets français fermants ("»").

On remarque justement que pour le clavier en ISO-8859-15, les entrées ISO-8859-1 correspondantes sont volontairement utilisées. En effet, au moment de l'écriture de cette disposition de clavier, la définition de l'ensemble des symboles ne comportait pas encore la norme Latin 9 association. Comme précédemment expliqué avec les tables de correspondance Unicode, il donc a été réutilisé la valeur correspondante en ISO-8859-1.

Les noms des keysyms ici utilisés, comme « guillemotleft », correspondent à des abréviations faciles à utiliser. Il est aussi possible d'utiliser le code Unicode hexadécimal correspondant, comme U+20AC pour l'euro - toutefois, ce fichier de configuration de la disposition du clavier devient alors moins lisible :

```
key <TLDE>  {    [       U+00BD,      U+00BC       ],
                 [       U+00AB,      U+00BB       ]     };
```

Dans les exemples précédemment donnés, il manque le cas très particulier du clavier canadien normalisé qui utilise aussi le CtrlGr pour définir d'autres associations que normal/majuscule/AltGr/AltGr+majuscule.

Voyons comment est définie cette même touche sur ce clavier canadien:

```
key <TLDE> {
        type= "THREE_LEVEL",
        symbols[Group1]= [ slash, backslash, bar ],
        symbols[Group2]= [ VoidSymbol, hyphen ]
};
```

On voit que deux groupes de trois niveaux sont déclarés de manière implicites, avec dans le cas du deuxième groupe un symbole vide en mode non majuscule pour bien signaler que seul CtrlGr+majuscule a une fonction.

Les groupes peuvent être aussi définis de manière séparée, lorsque comme dans le cas du clavier japonais une même touche correspond à des usages très différents (alphabet romain/kanjis) : dans ce cas la définition du premier groupe recense toutes les touches correspondante à ce premier cas (alphabet romain), et la définition du deuxième groupe la complète. Il est alors plus facile de faire évoluer indépendamment chaque définition de groupe.

Les types de niveaux sont définis à part, de même que les modifieurs permettant d'atteindre ces niveaux, les caractères spéciaux et de composition. Vu le peu d'intérêt que leur étude présente dans le cadre de cette thèse, ils ne seront pas abordés.

(v) Polices de caractères

À l'instar du mode console, X-Windows utilise des polices de caractères. Toutefois, le mode graphique permet beaucoup plus de variété de taille et de calligraphie (gras, italique ...)

Les polices sont gérées par X-Windows d'une manière originale : divers formats sont supportés (ex : true type, pcf) par un plugin adapté. Pour chaque format, les paramètres de base de X-Windows sont gérés, comme la famille, le sous-type, la taille, la résolution (en cas de police non vectorielle), etc.

La gestion des polices dévolue aux plugins ne mérite pas d'étude particulière. Il convient juste de signaler que le rendu des glyphes dans les cas de combinaison, de ligature ou de variance (ex : ha arabe) n'est n'est pas encore bien géré par X-Windows. Ce problème est laissé à la discrétion des bibliothèques comme Qt ou Gtk : il est à noter que ce dernier utilise Pango.

(vi) Rendu Unicode avec le moteur Pango

Pango est chargé du rendu de la police de caractères. Pour cela, il découpe le texte en paragraphe dépendant de la langue *[2006-Trager-International_Layout]* (ex : un paragraphe en japonais, suivi d'un en arabe), chacun subissant alors une série de traitements :

- passage de la série de codes Unicode à une séquence visuelle tenant compte du sens de lecture

- remplacement des glyphes selon le contexte (ex : ha arabe) en application de la police de caractères correspondante,

- découpage en segments insécables avec calcul du nombre de lignes à afficher, puis positionnement des segments sur chaque ligne en tenant compte du sens de lecture et de l'indentation.

Ainsi, il applique les diverses étapes précédemment présentées, dont l'algorithme BiDi, les tables de typographie avancées, puis l'adaptation à l'espace d'affichage. Le texte rendu sous forme de glyphes peut ainsi être utilisé dans les applications, sans nécessité de développement spécifique.

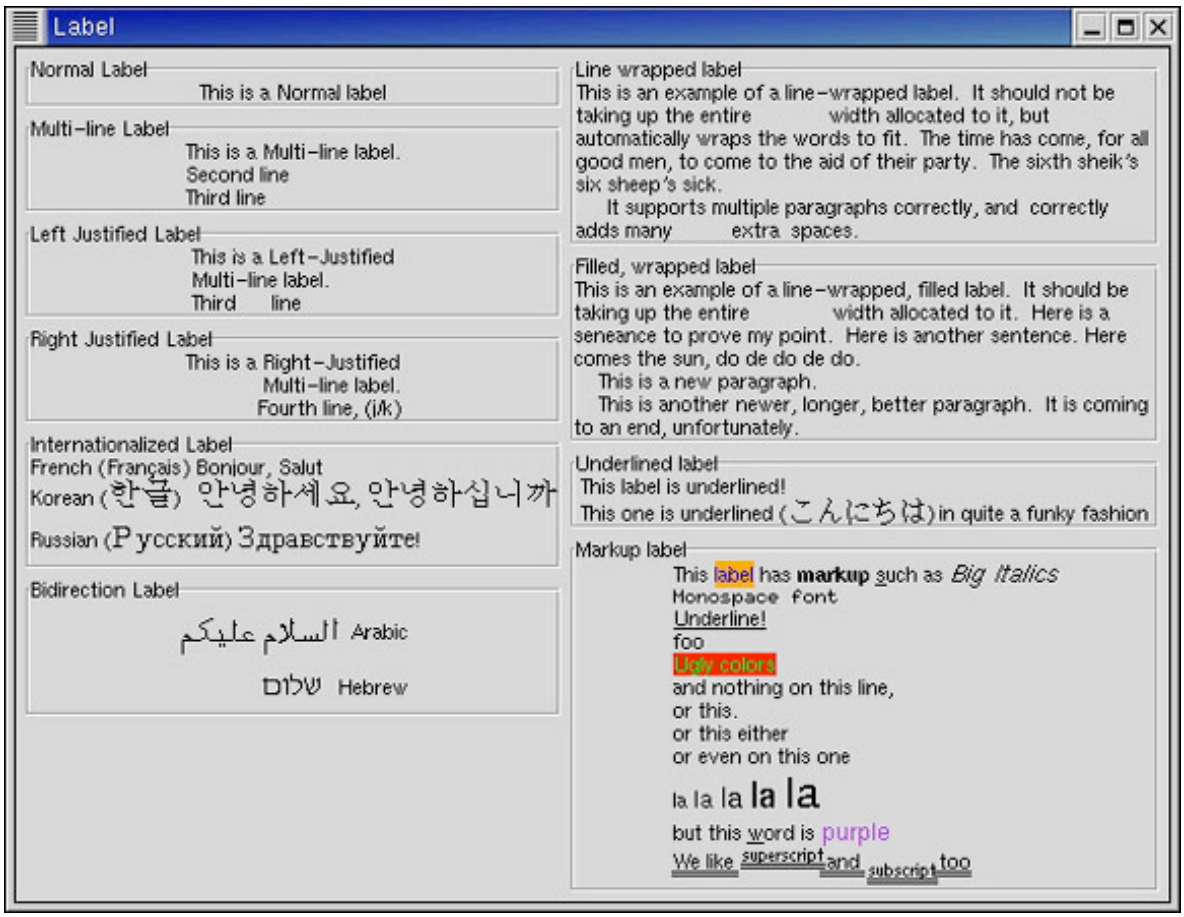


Figure 67: Labels en Unicode dans une application, avec des cas complexes/BiDi

L'équivalent de Pango est Core-Text sous MacOS-X, et Uniscribe sous Windows.

2-5-5. Conclusion

Le support réalisé par le système GNU/Linux, très complet, permet de passer totalement d'une langue à l'autre. Les opérations de gestion de l'entrée sont particulièrement bien développées, aussi bien dans le mode console que dans le mode graphique.

Toutefois, un bon support des fonctionnalités spécifiques d'Unicode est l'apanage de ce dernier mode, le mode console ne gérant que quelques combinaisons et substitutions de base, comme celles donnant les glyphes des différents modes ISO latin, car il est limité par l'implémentation du mode texte dans la ROM de la carte graphique (norme VGA), vu l'absence de moteur de rendu Unicode au niveau noyau dans l'émulateur de console (tty) en sa version actuelle.

Et même sous X-Windows, les applications ne faisant pas appel à un moteur Unicode pour gérer leurs Widgets ne pourront bénéficier des tables de typographies avancée précédemment présentées.

2-6. Discussion tenant compte des besoins identifiés et des fonctionnalités nécessaires

En opposant les différentes parties de l'état de l'art, il est possible de discuter des possibilités de réalisation pratique, en tenant compte des possibilités d'Unicode, des limites pratiques dans les implémentations existantes, et des besoins de SW pour écrire les langues signées.

2-6-1. Rappels

L'écriture de la langue des signes française n'est pas encore normalisée, ou définie précisément. Il existe juste des travaux de recherche comme LS-SCRIPT, et des formalismes d'écritures plus ou moins adaptés aux besoins des sourds. Parmi ces formalismes, SW a été retenu comme le formalisme de choix pour cette thèse étant donné sa très grande complexité, et surtout son aspect graphique 2d.

Mais suite à une étude d'Unicode, et surtout du détail de son implémentation sur un système d'exploitation, il apparaît surtout que les particulartiés de SW ne se prêtent pas à un support par Unicode. Détaillons pourquoi.

2-6-2. Combinatoire

Un premier problème avec SW est la vaste combinatoire. En effet, les symboles de base peuvent être associés librement, aucune contrainte n'ayant été formalisée - pas même des contraintes physiques ou anatomique *[1979-Mandel-Constraints]*.

En conséquence, il est impossible de se ramener à des cas simples comme ceux des accents, qui pour les langues écrites se combinent de manière formalisable, même dans les cas complexes comme pour le Vietnamien.

Il n'est donc pas possible de réduire la combinatoire par une formalisation mathématique simplificatrice, comme c'est le cas pour les points d'ancrage des accents.

2-6-3. Étendue du répertoire

Le second problème est le vaste choix des symboles pour composer les signes SW : il est à rapprocher de la largeur des systèmes d'écritures orientaux, notamment des milliers de logogrammes Chinois, qui requièrent une gestion précise par Unicode.

Toutefois, comme pour le problème des accents, diverses méthodes d'accès à ces logogrammes existent.

Par exemple, dans le cas des kanjis japonais, si le choix d'une orthographe lors de l'entrée alphabétique au clavier a été précédemment présenté, d'autres systèmes dits à squelette, sont à noter.

L'utilisateur trace sur un écran tactile les traits dans un ordre et une direction bien précise, puis affine ses choix ultérieurement, indiquant aussi a priori la classe du caractère dessiné *[2003-Aznar-Zaurus_C700].*

On peut aussi considérer que la saisie en romaji des alphabets hiraganas ou katakanas, tout comme le système à squelette où les traits prédominants sont saisis d'abord ne sont que des moyens de contournement du problème.

Toutefois, ils sont très fonctionnels, et des systèmes théoriques ou fonctionnels d'analyse de l'écriture manuscrite *[2002-Knoblich-Predicting_Strokes]* adoptent même des approches similaires *[2004-Spagnolo-Superposed_Strokes].*

En ce qui concerne la reconnaissance graphique, dans des cas plus simples, pour des lexiques plus limités, il est possible de simplifier encore plus la saisie.

Ainsi, il a été démontré et mis en œuvre que la reconnaissance d'un alphabet latin simplifié pouvait se faire sur une simple matrice 3x3, avec le système Graffiti.

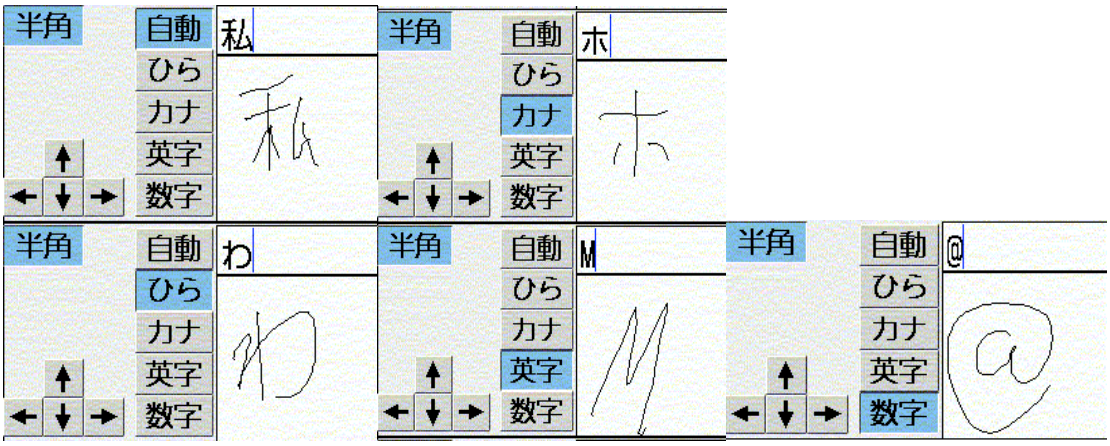


Figure 68: Saisie à main levée des alphabets japonais et des caractères par squelette sur assistant numérique personnel de type Zaurus sous Linux

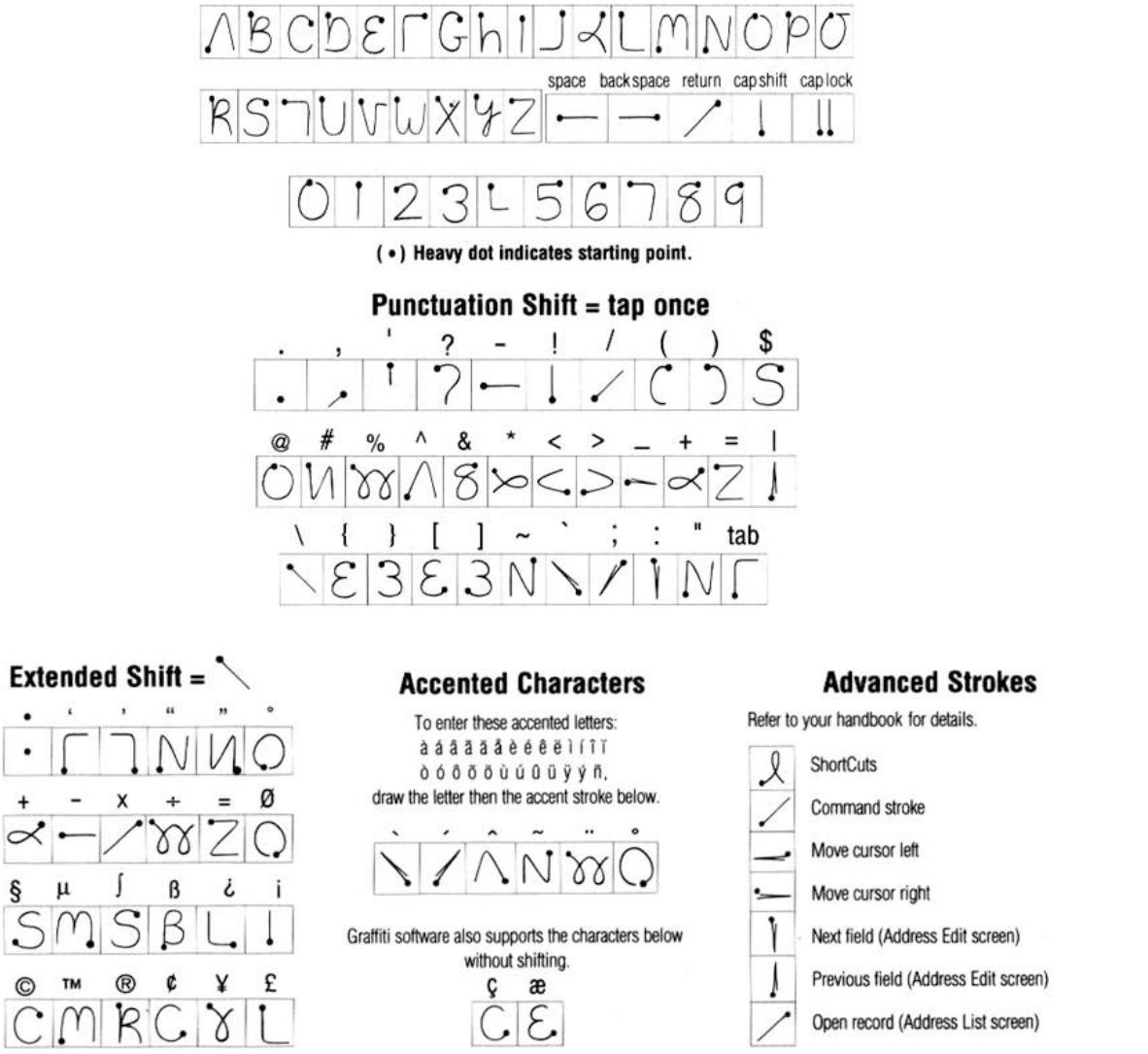


Figure 69: Système Graffiti, utilisé sur Palm

Mais la saisie de SW en reconnaissance de caractère, vu l'étendue du répertoire de signes SW, et la constante variabilité des symboles, est difficilement envisageable.

Certains ont toutefois essayé d'adapter un système équivalent à Graffiti, ce qui n'a pas permis une mise en œuvre fonctionnelle pratique *[2003-Rocha-Graffiti-SW]* en raison du trop vaste répertoire. Des mécanismes similaires aux racines japonaises auraient pu faciliter la tâche, mais aucun ordre n'est encore formalisé avec SW qui est d'ailleurs encore extrêmement variable et qui ne se prête donc pas à une saisie guidée.

Une utilisation de la classification SSS, appliquée ailleurs dans la saisie au clavier par un système de menus *[1995-Sutton-SignWriter]* pourrait être envisagée.

Elle requiert toutefois de naviguer dans des menus longs et complexes : en celà, elle ne simplifie pas l'accès au répertoire. Les mécanismes existants se limitent sinon aux symboles de base utilisés en bijection de l'alphabet latin - pratique dite de dactylologie, ne pouvant en aucun cas servir à la saisie en SW.

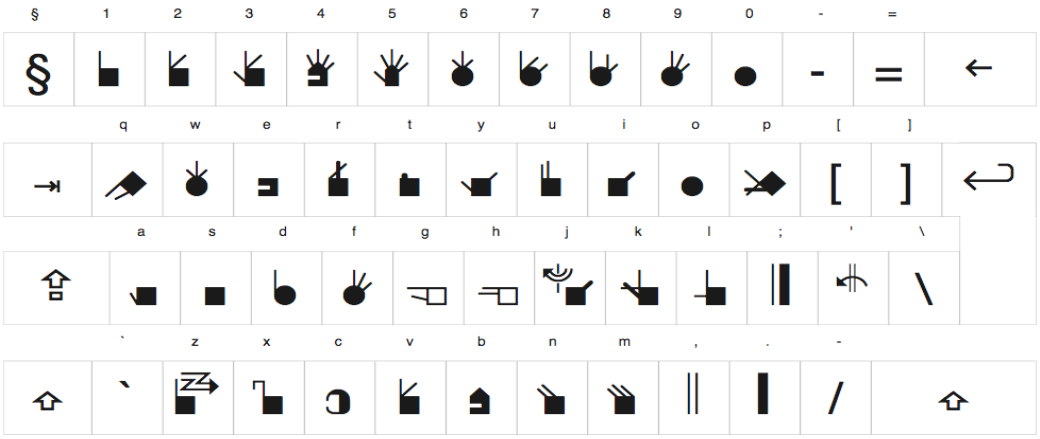


Figure 70: Menu de clavier SW pour la saisie d'un symbole de base

2-6-4. Spatialisation

Le troisième verrou scientifique à retirer de cet état de l'art est qu'Unicode, malgré tous ses caractères spécifiques visant à rendre l'aspect graphique indépendant du format logiciel, n'est pas adapté à un formalisme d'écriture bidimensionnel.

En effet, comment préciser qu'un symbole de base se situe à des coordonnées relatives par rapport à un autre, surtout lorsque de multiples symboles sont ainsi à positionner ?

Le cas des accentuations multiples des tuples de lettre est figé par des règles : il n'est pas possible de déployer des mécanismes similaires vu la spatialisation analogique de SW, qui fait que de telles coordonnées sont très variables.

De même, comment gérer la spatialisation de ces combinaisons, qui elles mêmes ne sont pas supportées ?

Une telle gestion de l'espace ne se fait qu'avec difficulté pour les langues comme l'indien ou l'arabe, qui requièrent de multiples opérations sur les glyphes avant d'arriver à la forme affichable finale. En plus, cette gestion nécessite des caractères Unicode spéciaux, pour indiquer au besoin à quel endroits les ligatures doivent ou ne doivent pas se faire.

2-6-5. Bidirectionnalité

La bidirectionnalité, c'est à dire l'utilisation de formalismes d'écriture différents dans un même paragraphe est aussi un point très particulier. Unicode utilise pour celà des marqueurs, ce qui permet de gérer les mélanges de formalisme, grâce à l'algorithme BiDi.

Il semble dans ce dernier cas, vu la grande précision et complétude de ce dernier, que l'intégration de SW ne devrait pas poser de problème particulier si les points précédents de support Unicode pouvaient être réglés, par contre, en l'abscence de support Unicode, le problème de la bidirectionnalité reviendrait se manifester.

2-6-6. Nécessité d'Unicode

Toutefois, malgré toutes ces difficultés, pour les raisons étudiées notamment pour la localisation, l'affichage et la propagation du format, un support d'Unicode est nécessaire.

En effet, faute d'un support Unicode, tout travail est condamné à rester l'apanage d'un logiciel spécifique, et ne pas pouvoir être étendu à l'ensemble du système d'exploitation.

2-6-7. Conclusion

Les apports de l'état de l'art sur la problématique présentée sont nombreux. Ils permettent notamment de mieux définir les enjeux et les directions à suivre. Les verrous scientifiques ont été précisés. À l'issue de la revue de l'état de l'art, il semble donc nécessaire d'étudier plus précisément la problématique du support en Unicode d'une forme écrite de la langue des signes, qui est un problème de recherche original puisque aucune possibilité n'a été prévue ou n'existe en Unicode pour gérer un formalisme d'écriture aussi complexe que SW.

2-7. Reformulation du sujet de thèse

2-7-1. Introduction

Suite à la discussion sur l'état de l'art, il convient donc de reformuler le sujet de thèse, puisque le problème n'est pas l'informatisation quelconque d'une forme écrite de la langue des signes.

En effet, des logiciels comme SignWriter ou SWEdit font déjà depuis de nombreuses années un support d'une forme écrite de la langue des signes.

Le problème est plutôt celui d'une informatisation de manière extensible et cohérente par rapport aux autres langues écrites.

Il faut pour celà tenir compte des contraintes d'Unicode, mais aussi de la langue des signes, et plus précisément de la langue des signes française (LSF), qui présente des caractéristiques propres, qui sont souvent différentes de celles des langues écrites, et qui ne sont donc pas gérées par leurs informatisations.

2-7-2. Reformulation

« Informatisation d'une forme écrite de la langue des signes » ...

(i) Réutilisabilité

.. « de manière à être utilisable en remplacement d'une autre forme écrite, et non comme une image ou un encodage XML »

Ces méthodes ne sont pas gérables par les systèmes d'exploitation moderne, ou l'algorithme BiDi.

Comme vu dans l'état de l'art, s'affranchir d'une telle limite, quelles que soient les possibilités de réalisations, condamnerait le résultat à des implémentations très difficiles dans les systèmes d'exploitation

(ii) Évolutivité

.. « tout en sachant s'adapter et évoluer pour répondre au besoin d'un autre formalisme d'écriture »

En effet, on retient le formalisme d'écriture le plus complexe existant actuellement, tout en gardant à l'esprit les possibilités d'évolution ultérieures (cas de SW) voire de remplacement par un autre formalisme (issu de LS-SCRIPT)

(iii) Encodage de caractères

« et gérant les contraintes et besoins spécifiques des normes d'encodage »

Comme étudié, un des manques majeurs des recherches actuelles sur la langue des signes et l'absence d'un formalisme commun de sauvegarde et d'annotation des données, pour s'interfacer avec des logiciels de traitement d'image.

Mais ce dernier point de la reformulation peut sembler moins évident que les deux précédents. Il est pourtant à la base de ces travaux. Étudions donc séparément les raisons sous tendant cette approche.

2-8. Justification d'un approche par encodage

2-8-1. Introduction

Il ne s'agit pas simplement de répondre à des besoins spécifiques et isolés du traitement d'image, ce que des modèles articulaires ou géométriques du corps du signeur pourraient théoriquement faire.

Comme présenté dans la reformulation, la difficulté est la coordination des trois approches : traitement d'image, linguistique et informatique, pour les différents besoins *[2003-DeLanghe-Traitement_sémantique_LSF]*.

À la base doit se trouver une compatibilité, en entrée et en sortie.

2-8-2. Objectifs

En effet, l'objectif de l'informatisation de la langue des signes est de pouvoir s'intégrer avec des logiciels traitement d'image:

- en entrée à des systèmes de saisie de langue des signes (ex : annotation de vidéo *[2002-Cuxac-LSCOLIN], [2004-Brafford-Annotation]*)

- en sortie à des systèmes de synthèse de langue des signes (ex : signeur virtuel aussi dit avatar signant *[2000-Losson-Signeur-Virtuel], [2002-Huenerfauth-ASL_Generation]*).

Pour l'instant, les systèmes doivent utiliser des approches imparfaites, comme la vidéo ou des images fixes *[2004-Vandamme-Websourd]* en l'absence d'une forme intermédiaire écrite informatisable.

Des problèmes similaires se sont posés pour les formats d'échange des graphiques *[2001-Taentzer-Graphs_Exchange_Format]*

De même, la forme intermédiaire utilisée par des systèmes de représentation de gestes *[2004-Lenseigne-Gesture_representation]* ne doit pas présenter d'incompatibilités ou d'impossibilités d'informatisation en traitement d'image.

Toutefois, de nombreux paramètres comme les occultations *[2004-McDermott-Occluding_contour]* ou la vision monoscopique *[2003-Gianni-Pose_reconstruction]* sont responsables d'incertitudes.

2-8-3. Linguistique

Les approches précédemment menées, d'un point de vue informatique, ont négligé l'aspect linguistique. Une approche linguistique dépend d'une grammaire de composition, basée sur un certain nombre de règles précises.

Toutefois, de telles règles de compositions n'ont pas encore été formulées, vu la jeunesse de ces langues et la complexité du sujet.

Pour l'instant, la linguistique isolée ne permet donc pas de bien prendre en compte le problème de l'écriture des langues signées.

2-8-4. Informatique

Les contraintes informatiques, comme présentées dans l'état de l'art des encodages et le support Unicode au niveau du système d'exploitation viennent surcontraindre la problématique.

Une conséquence est un grand besoin en espace de travail et en temps de calcul.

Mais une approche informatique pure ne saurait suffire : graphique par excellence, la LSF a des propriétés spécifiques qui sont rarement retrouvées dans les autres langues, et donc qui n'ont pas été prévues dans les normes informatiques de gestion des langues.

2-8-5. Conclusion

Une approche traitement d'image semble donc justifiée, pour ne pas chercher à appliquer des connaissances linguistiques encore absentes, tout en cherchant à respecter le plus possible les contraintes informatiques de la linguistique, ne serait-ce que pour qu'il soit un jour possible de gérer la langue des signes comme les autres langues écrites, en respectant et en étendant au besoin pour celà les normes existantes.

Suite à cette reformulation du sujet et à la précision des objectifs, il convient maintenant d'envisager la méthodologie de recherche.

3. Étude de la problématique et mise en œuvre

Il a été retenu que le support Unicode était nécessaire au niveau du système d'exploitation. Toutefois, le support Unicode d'un formalisme aussi complexe que SW est pour l'instant impossible. De même, les propriétés de l'écriture en SW par les sourds sont peu documentées

Il convient donc de réaliser des recherches pour préciser :

- la nature des problèmes rencontrés
- la segmentation des tâches à apporter
- les aspects spécifiques à SW

Ensuite seulement pourront être étudiés et comparés les possibilités éventuelles qui pourraient se présenter et répondre à ces besoins.

Ce chapitre se sépare donc en six parties, respectivement :

- étude sur site, auprès d'utilisateurs
- étude des problèmes de SWML
- resegmentation des tâches
- problème des variabilités
- encodage
- applicabilité

Pour débuter les recherches, commençons par préciser l'usage qui est fait par des utilisateurs habituels de SW.

3-1. Étude sur le terrain au Brésil

SW est de plus en plus utilisé par les sourds. Les foyers principaux d'éducation en écriture de la langue des signes sont situés en Amérique du nord (Californie) et en Amérique du Sud (Venezuela, Brésil).

Grâce à une bourse de mobilité internationale de l'Université Paul Sabatier, des recherches sur le terrain ont pu être entreprises auprès d'enfants apprenant SW, à l'Escola Frei Pacífico située à Pelotas, dans l'état de Rio Grande do Sul, où se situe l'une des communautés de sourds écrivant en SW des plus dynamiques *[2005-Skliar-Bilingual_Brazil].*

La professeur responsable de cet enseignement a par ailleurs soutenue un thèse d'informatique pédagogique *[2003-Stumpf-SW]* où plus de détails peuvent être trouvés si nécessaire, complétés si besoin par d'autres études plus pédagogiques *[2003-Slobin-Acquisition], [2001-Flood-Learning_to_write_in_SW]* et surtout par les résultats issus du projet LS-SCRIPT *[2007-Garcia-LSSCRIPT].*

3-1-1. Introduction

Outre la langue des signes Brésilienne, ces enfants scolarisés en cours moyen dans un cursus spécial pour les sourds, connaissent déjà un peu le Portuguais Brésilien et son alphabet.

Il s'agit d'une expérience nouvelle : les classe de cours moyens des années précédentes ne comportaient pas de cursus en SW.

L'apprentissage de SW se fait d'ailleurs sous l'excuse de « cours d'informatique », vu la crainte des administrateurs d'une opposition des parents.

3-1-2. Méthodologie

Une classe de jeunes sourds apprenant SW en langue des signes brésilienne a été observée en juin 2004 pendant une journée d'école, en la présence de leur professeur, pour mieux déterminer comment était utilisé SW.

En tant qu'observateur, je me suis positionné en fond de classe, sans interagir avec les élèves ou leur professeur, pour minimiser les biais comportementaux liés à cette présence externe.

3-1-3. Description de la classe

La classe comporte 6 élèves, 3 filles et 3 garçons. L'âge des élèves rend la classe comparable à celle d'un cours moyen. 4 élèves sont organisés en paire autour de 2 ordinateurs. Les 2 élèves restant sont seuls face à leur ordinateur. 2 ordinateurs ne sont pas utilisés. Cette disposition résulte d'un choix personnel des élèves : une paire s'est d'ailleurs réorganisée au cours de cette observation.

Figure 71: Élèves de la classe

3-1-4. Processus pédagogique

Plusieurs étapes successives sont réalisées, le professeur se déplaçant entre chaque étape d'un groupe d'étudiant à l'autre.:

- un nouveau signe est présenté
- la signification de ce signe est donnée en langue des signes Brésilienne
- le mot correspondant en brésilien est écrit au tableau, avec dactylographie du ou des mots correspondants

- le signe est écrit au tableau en SW
- le signe est dessiné dans le cahier des élèves à la main
- le signe est saisi sur ordinateur avec le logiciel SignWriter DOS

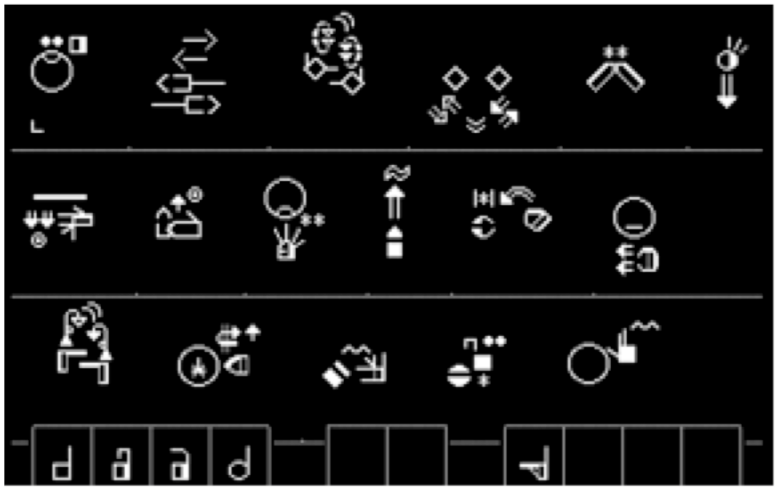


Figure 72: SignWriter DOS

Ce processus est parfois complété par une dernière étape, l'impression des signes saisis sur ordinateur. Toutefois, vu les coûts de l'impression, cette étape est peu fréquemment réalisée. De plus, le haut degré de pixellisation des signes imprimés rend difficile leur lecture, et n'incite pas à faire appel à l'impression informatique.

Figure 73: Résultat de l'impression : les signes sont très pixellisés

3-1-5. Outils de travail

(i) Le tableau

Il s'agit de l'outil central de la classe, fréquemment utilisé. Les signes qui sont en train d'être appris y sont écrits en SW, ainsi que d'autres signes fruits de discussions avec les élèves.

Le tableau sert donc de support de discussion, pour décider comment transcrire des signes en SW lorsque plusieurs possibilités existent, pour un même signe, à l'instar des variantes orthographiques en français (clef/clé), par exemple en changeant de point de vue, ou l'ensemble de symboles retenus pour être plus ou moins précis.

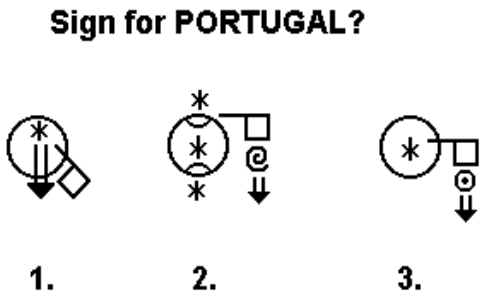


Figure 74: Illustration du problème de choix d'une transcription

Il sert aussi de guide pour donner des objectifs aux élèves, comme la transcription SW à reproduire avec le logiciel SignWriter.

(ii) L'ordinateur

Allumé par les élèves au début de la classe, il reste en fonction tout au long du cours. Il est utilisé dans le but pédagogique donné, les enfants n'essayant même pas de démarrer d'autres logiciels (ex : jeux) pourtant installés dessus. Ils utilisent par contre fréquemment des raccourcis claviers pour se distraire mutuellement par exemple en quittant ou en cachant l'application (ex : alt-tab) à l'insu de l'utilisateur.

(iii) Le cahier

Chaque élève a son propre cahier. En première année, les enfants apprennent à écrire des symboles isolés, en commençant par étudier les modalités de représentation planaire de conformations spatiales de segments du corps donnés.

Figure 75: Cahier personnel d'un élève, avec son signe et son nom.

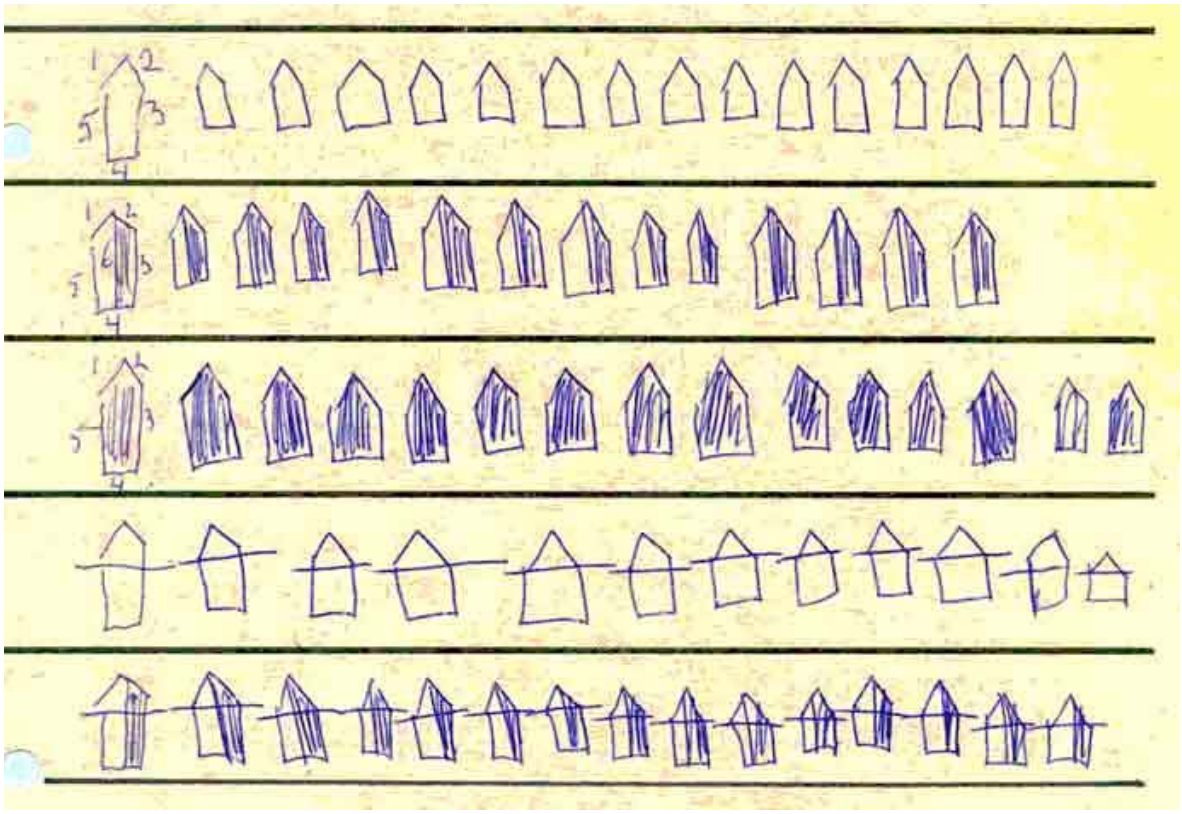


Figure 76: Premier essais de reproduction de symboles isolés

Ensuite, ils sont reproduits à la main, en apprenant les correspondances anatomiques et spatiales offertes SW.

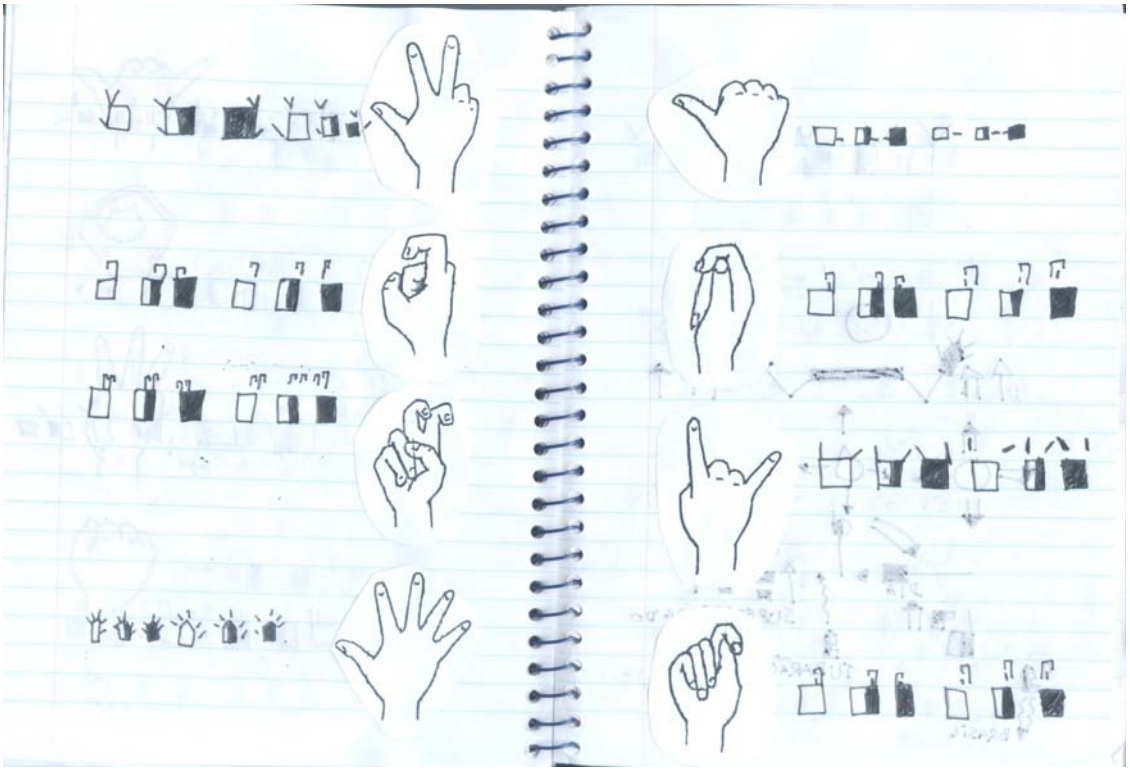


Figure 77: Apprentissage des correspondances anatomiques

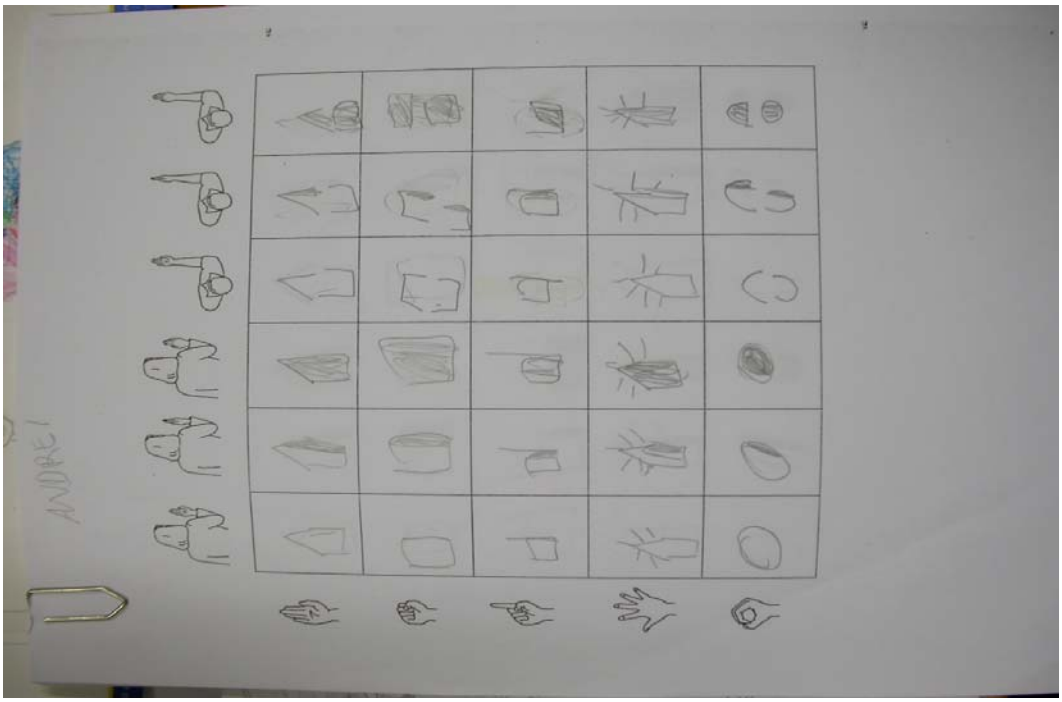


Figure 78: Apprentissage des correspondances spatiales

Ces symboles sont ensuite incorporés par l'élève, qui les maîtrise et les utilise pour ses propres besoins, par exemple pour stocker sur un support papier la manière de transcrire les lettres qui composent son nom en dactylographie.

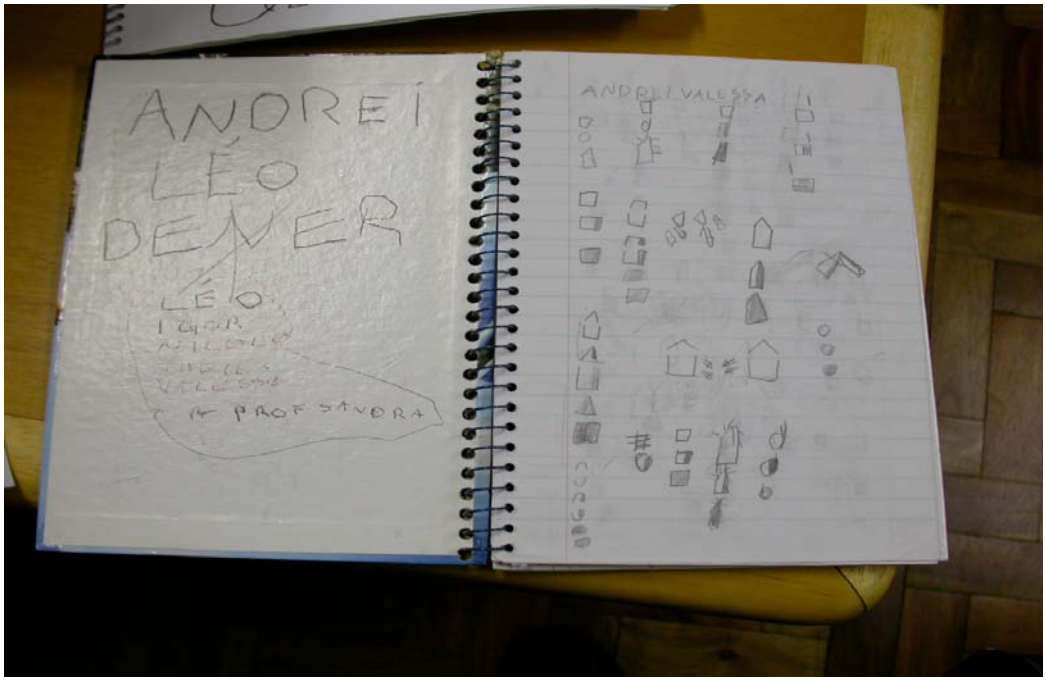


Figure 79: Cahier d'un élève en première année : symboles isolés

Petit à petit, les cours de SW permettent aux étudiants de former des associations de symboles pour exprimer des signes, représentant des concepts.

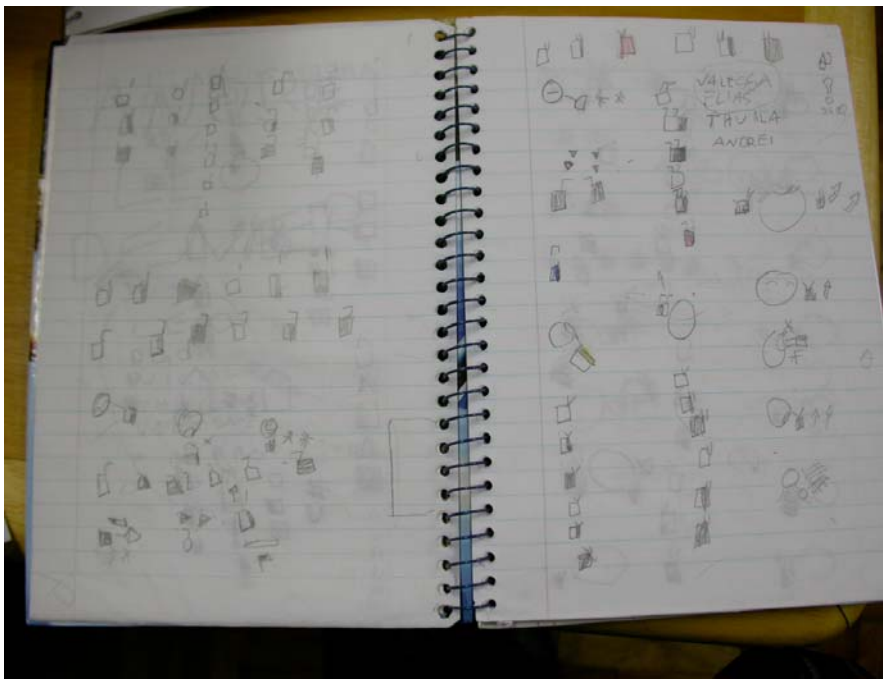


Figure 80: Premières associations de symboles

En deuxième année, les enfants apprennent ensuite à écrire :

- leur nom
- les variations de SW, comme les remplissages, les rotations
- la dactylographie et les chiffres
- les symboles
- les groupes de classification SSS
- les nouveaux signes qu'ils apprennent

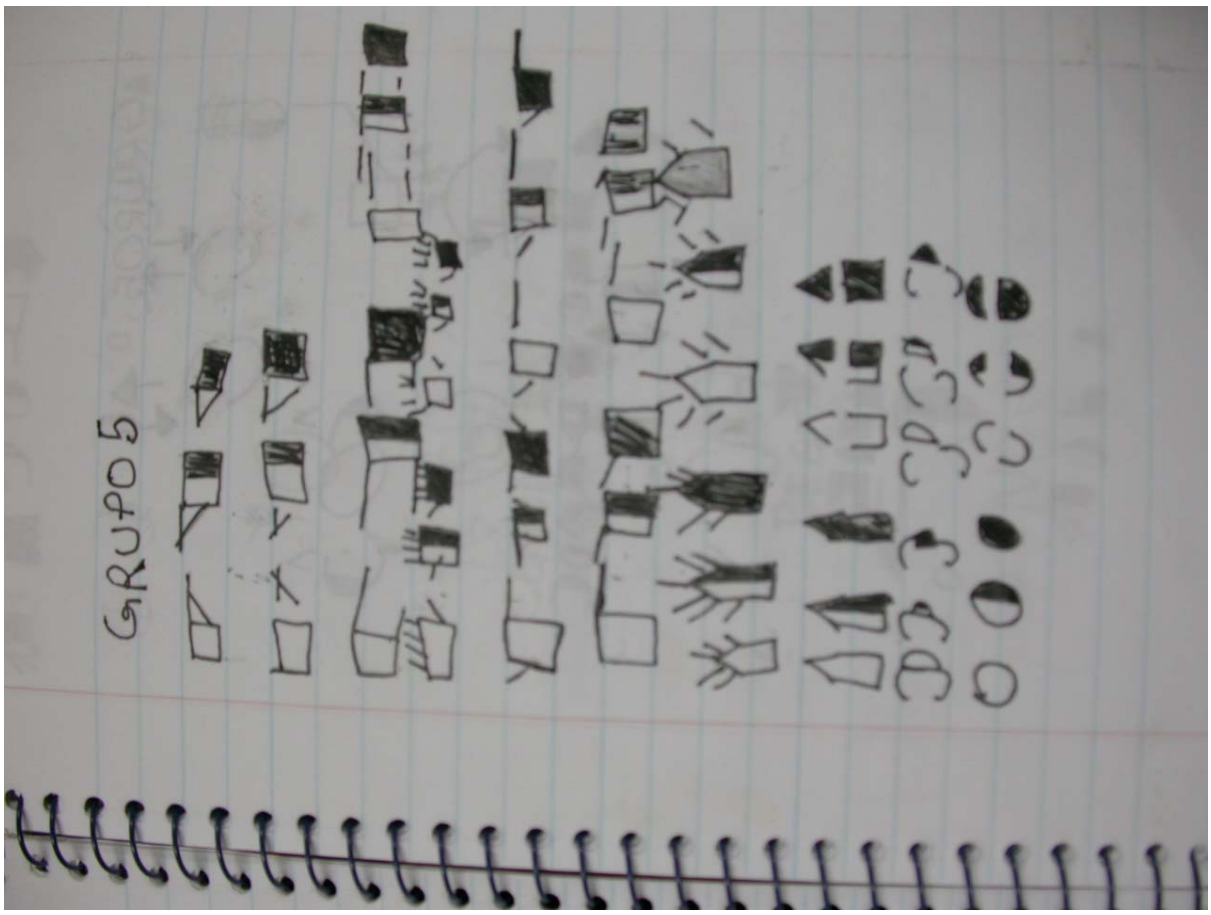


Figure 81: Exemple de groupes SSS de SignWriting

Ici, seul le groupe 5 de SW est présenté, mais chaque groupe est appris successivement, car l'organisation SSS facilite l'usage de SignWriter, qui fonctionne selon un système de menus.

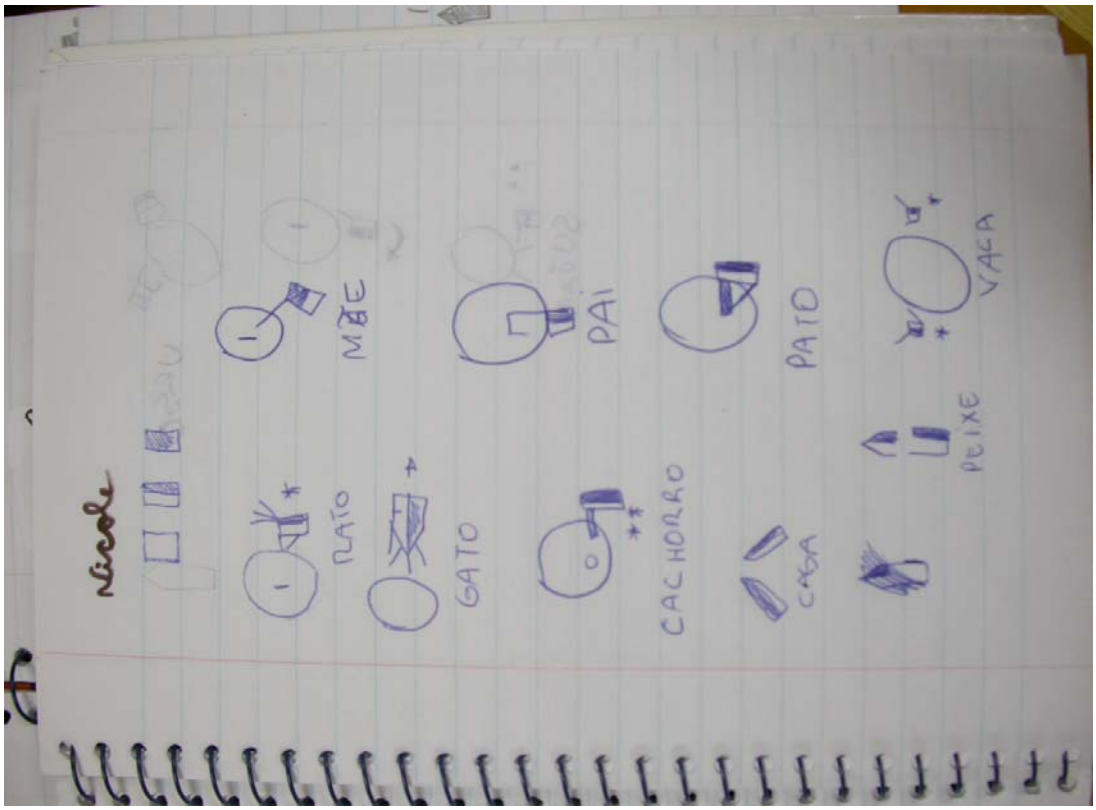


Figure 82: Approfondissement du vocabulaire

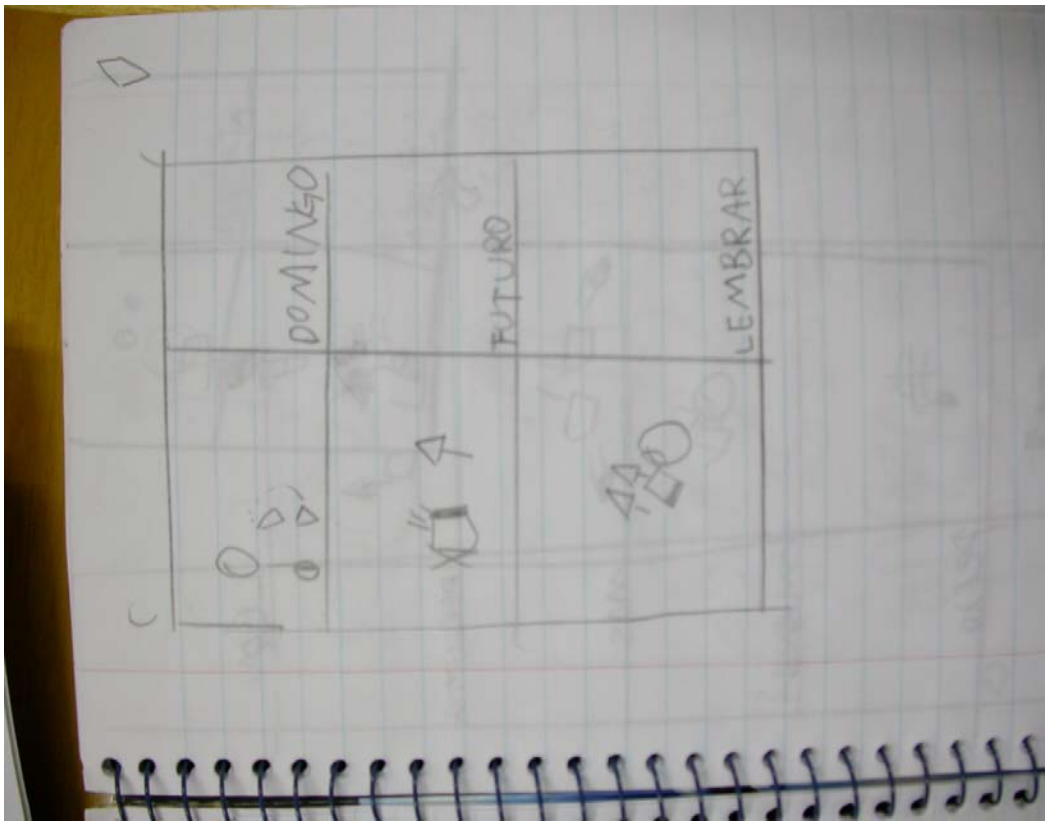


Figure 83: Association des signes SignWriting avec les mots en Portugais Brésilien

(iv) Les livres

Pour l'instant, seules deux classes ont suivies le cursus SW.

Ultérieurement, il est prévu d'apprendre en troisième année la transcription en SW des règles de grammaire de la langue des signes brésilienne, et en quatrième année la lecture sur des livres en SW.

Actuellement, vu le peu de livres disponibles et la volonté de l'administration de l'école de ne pas trop mettre en avant l'écriture de la langue des signes, de telles possibilités semblent très restreintes.

En attendant, les élèves écrivent des histoires pour commenter des dessins.

Figure 84: Livres en SignWriting

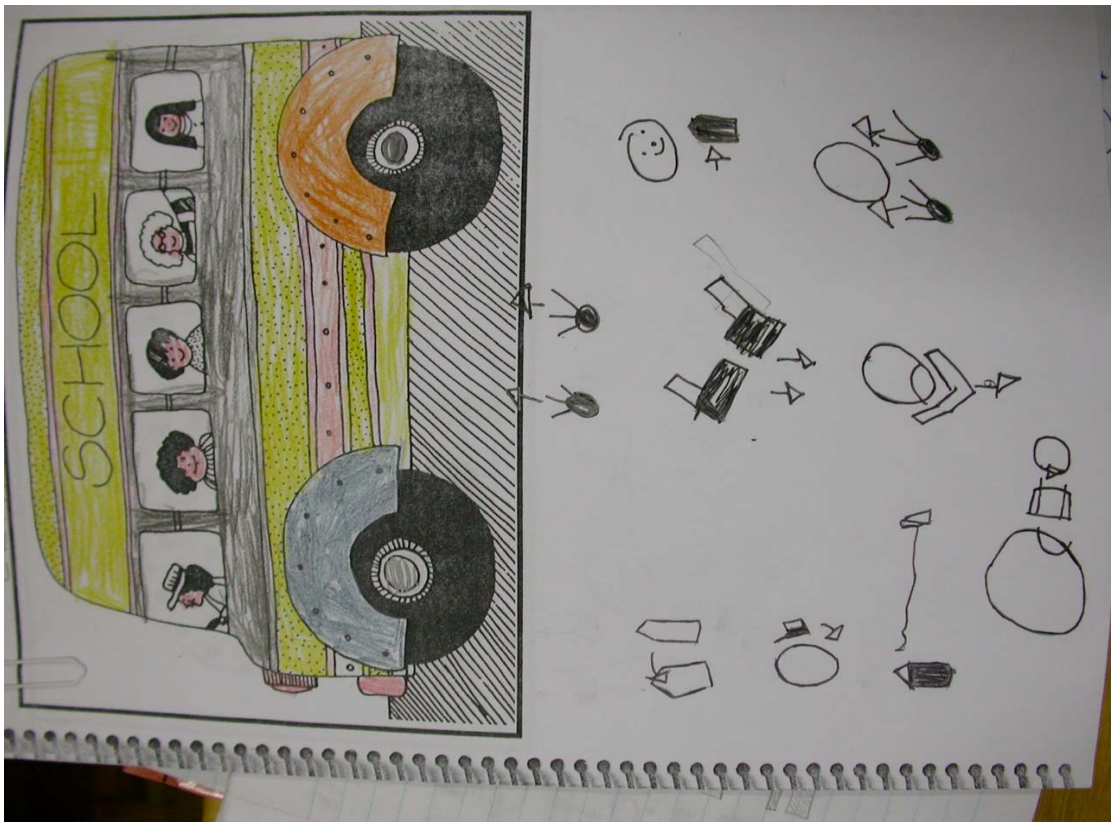


Figure 85: Commentaire des dessins

3-1-6. Interactions avec le professeur

Durant la phase de saisie informatique, chaque élève demande tour à tour la présence du professeur pour l'aider.

4 situations principales de demande d'aide ont été identifiées:

- savoir comment taper avec le clavier un symbole, rangé selon le système SSS (il est à noter toutefois la présence d'une feuille de papier sur chaque table présentant ce système de classification)

- choisir une combinaison de symboles de base pour transcrire un signe

- choisir la présentation visuelle en symboles du signe SW final par rapport au signe en langue des signes Brésilienne

- demander confirmation de la bonne réalisation du signe final

3-1-7. Délais

Chaque itération du processus pédagogique prend 5 minutes pour les enfants les plus rapides.

Le temps n'est pas réparti de manière identique entre chaque étape, car la dernière qui fait appel à l'ordinateur est aussi celle qui prend le plus de temps.

La moitié du temps sur l'ordinateur est consacré à des distractions, et l'autre moitié à une utilisation. Les distractions, sous forme de taquineries aux autres élèves, se réalisent principalement lors du positionnement des symboles pour créer un signe.

3-1-8. Collaboration

Même s'il ne s'agit que d'une petite série ne pouvant pas prétendre avoir un grand pouvoir de démonstration statistique, la collaboration entre les élèves est assez importante pour être notée.

En effet, lorsque le professeur n'est pas disponible pour répondre aux attentes d'un élève, un élève vient spontanément à l'aide de son camarade.

Il ne semble pas y avoir d'effet « meneur », puisque différents élèves sont venus tour à tour aider une de leur camarade rencontrant des difficultés dans l'utilisation de SignWriter DOS

3-1-9. Perturbations induites

Après une phase initiale de curiosité où les élèves ont demandé comment se signait le nom de l'observateur, d'où il venait et ce qu'il faisait, il y eut très peu d'interactions.

Les perturbations induites par l'observation semblent donc minimales, et il est possible de supposer que les enfants se sont comportés comme ils le font de manière habituelle dans la classe.

3-1-10. Interprétation

Le ratio de 6 élèves pour 1 professeur semble maximal : les problèmes de discipline sont en effet fréquents.

Parmi ces problèmes de discipline, il est à noter l'utilisation de combinaisons de touches, comme Alt+Tab, pour passer de SignWriter DOS à Windows.

L'ordinateur semble bien accepté, puisqu'il est utilisé pour la tâche prévue, et non pour jouer.

Le positionnement des symboles pour créer un signe semble l'étape la plus longue et la plus frustrante pour les enfants, puisque la majorité des distractions et taquineries se déroule lors de cette phase.

Beaucoup de temps est aussi perdu pour la navigation dans SSS, à la recherche d'un symbole de base particulier pour composer le signe, et ce malgré la présence d'une feuille imprimée sur chaque table rappelant l'organisation de SSS.

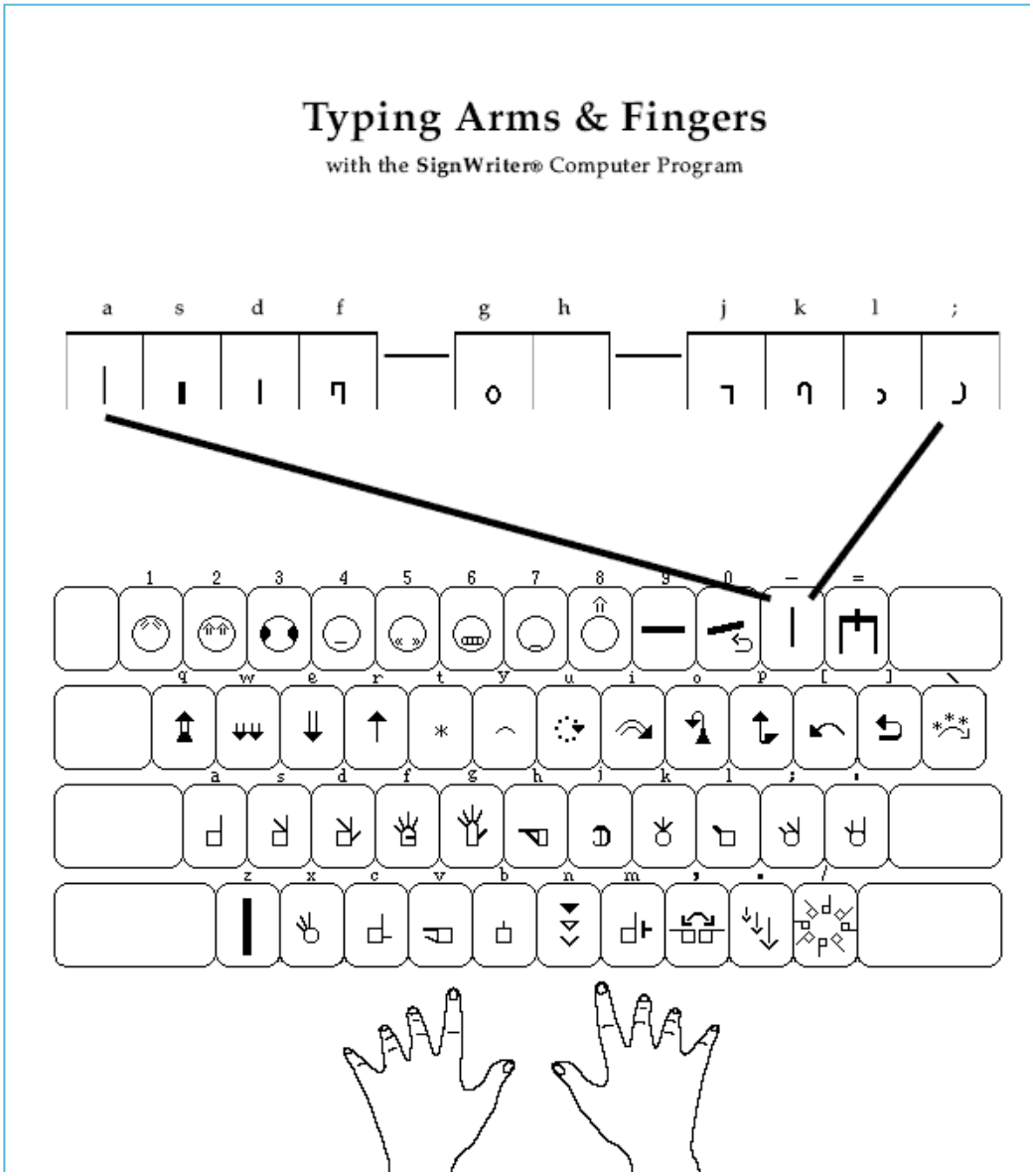


Figure 86: Navigation SSS dans SignWriter DOS

De plus, une telle étape de navigation est obligatoire, même pour les enfants qui sont encore entrain d'apprendre les bases de SW, en première année.

L'ordinateur semble donc imposer de lourdes contraintes au processus éducatif, puisque les étapes de collaboration au tableau ont à subir les exigences temporelles des interactions informatiques, qui ne suivent pas les objectifs pédagogiques d'apprentissage de l'écriture en SW.

S'il doit s'agir d'une classe d'écriture de langue des signes, le logiciel devrait être moins contraignant, paramètre pouvant se mesurer par le temps passé par les élèves à utiliser leur ordinateur pour un objectif autre qu'éducatif, et surtout par le moment de cette utilisation détournée de l'ordinateur.

3-1-11. Conclusion

Cette étude sur le terrain, auprès d'utilisateurs apprenants et novices informatique, permet de dégager de grands axes de travail.

Tout d'abord, peut être que l'utilisation de l'ordinateur n'est pas pertinente pour ce type d'élèves ou ce stade d'apprentissage ?

Mais si l'on accepte le principe de l'apprentissage par le biais d'un outil informatique, les limites d'un support informatique trop indépendant du système d'exploitation sont bien notées, avec la coexistence de SignWriter DOS et de l'environnement Windows.

Les conséquence de ce mécanisme de gestion de l'écrit sont multiples, avec la gêne représentée par certains raccourcis claviers, et les difficultés d'impression.

Surtout, une seule méthode de saisie, imposée par le logiciel, est possible. Le recours à de multiples sous menus imbriqués accessibles au clavier semble très inadapté aux enfants, vu leur distraction à cette phase, et ce malgré la présence de documentation destinée à les aider.

Vient alors une phase complexe de positionnement fin, où le problème du choix de symboles et de leur positionnement se pose même après la réalisation : une fois le signe final à l'écran, les enfants demandent à leur professeur qu'elle vienne valider ce résultat : il n'y a pas de possibilité de vérification automatique du degré de correction lexical.

Un autre point important est que les impressions très fortement pixellisées sont peu demandées. Il pourrait être donné comme cause des difficultés rencontrées lors de cette étude sur le terrain le choix fait par l'Escola Frei Pacifico d'utiliser un logiciel DOS, SignWriter *[1995-Sutton-SignWriter]*, qui fut d'ailleurs le premier logiciel supportant SW.

3-1-12. Critique du choix du format d'encodage

Ce dernier utilise un format binaire pour sauvegarder ses fichiers SW, avec lequel le type de chaque symbole de base ainsi que ses coordonnées sont stockés pour chaque symbole sur plusieurs bits.

Les inconvénients sont multiples, dont une interopérabilité difficile puisqu'il est impossible d'associer SW avec d'autres formalismes d'écritures, et surtout de nombreux problèmes pour mixer SW à d'autres formats électroniques.

Il semble donc nécessaire de retenir un autre format d'encodage. Continuons donc maintenant en choisissant le format le plus évolué retenu dans l'état de l'art, SWML.

3-1-13. Critique du choix d'un logiciel DOS

Le choix de SignWriter, logiciel DOS, a aussi posé un problème lors de cette étude sur le terrain - par exemple la découverte de l'environnement DOS, peu familier aux enfants, et l'utilisation d'Alt+Tab pour se distraire et passer à Windows, où seul le Portuguais Brésilien est utilisé.

Fort heureusement, un logiciel Java nommé SWEdit *[2001-Torchelsen-SWEdit]* permet d'éditer des documents au format SWML.

Continuons donc cette étude avec SWEdit, qui utilise le format SWML

3-2. Étude de SWEdit et SWML

L'encodage dénommé SWML, présenté plus haut *[2001-DaRocha-SWML]*, a résolu les problèmes les plus importants du format binaire de SignWriter en se basant sur XML *[1996-Bray-XML]* pour réaliser ces opérations de stockage du type et des coordonnées des symboles.

Ainsi les documents SignWriting sont devenus plus faciles à partager et à intégrer au sein d'autres contenus : il est par exemple possible d'associer du texte dans un alphabet donné, des images et du SW, dont l'usage n'est donc plus exclusif.

3-2-1. Logiciel SWEdit

SWML est d'ailleurs devenu l'encodage le plus répandu, car grâce au format XML qu'il utilise, il est facile à manipuler et à croiser avec d'autres formats.

Toutefois, des difficultés persistent : le logiciel SWEdit, ne permet pas de faire significativement mieux que SignWriter.

Il impose même des contraintes supplémentaires liées à son format XML.

Mais les problèmes les plus évidents sont liés à la conception même de SWEdit.

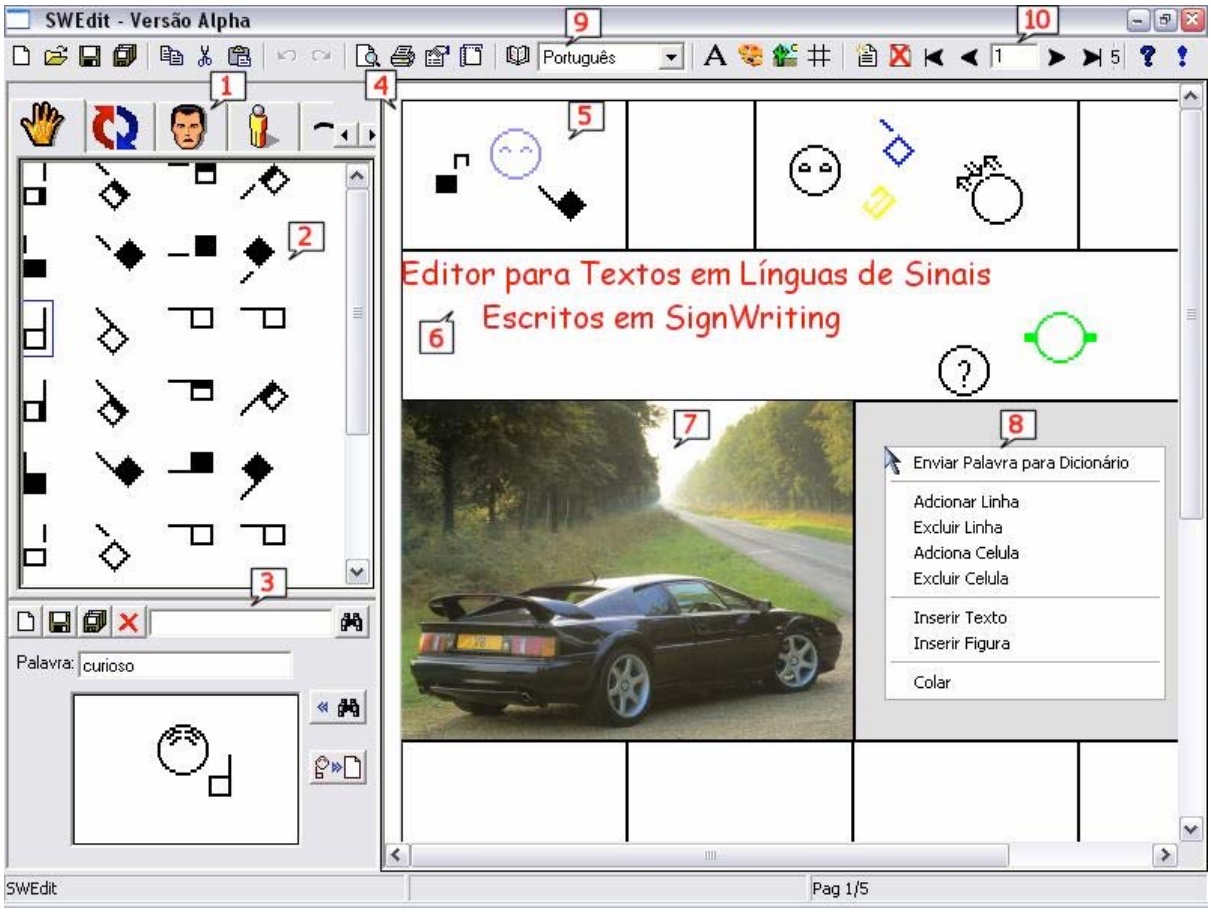


Figure 87: Logiciel SWEdit, où SignWriting n'est plus exclusif

(i) Non redimensionnable

Comme présenté précédemment, les fichiers SWML positionnent des images au format bitmap (GIF) pour chaque symbole. Ainsi, la sortie produite n'est pas redimensionnable.

Ce problème se pose tout particulièrement pour l'impression ou les traitements de texte. Il pourrait être contourné par l'utilisation d'images vectorielles, par exemple au format SVG *[1999-Ferraiolo-SVG]*, facilement intégrable à des documents XML, pour représenter les symboles de base. Une police SVG a d'ailleurs été développée avec SSS-2004, pour répondre à ce besoin.

(ii) Une seule police

Mais même avec une police SVG, le format SWML actuel ne gère qu'une seule police de caractère, qui plus est sous forme non standard.

En effet, la police n'est qu'une collection de glyphes au format de dessin vectoriel SVG, et non dans un format de police de caractère comme TrueType : il pose donc un problème pour l'utilisation de SignWriting en tant que forme écrite.

Les attributs des polices de caractères servent pourtant à véhiculer des informations supplémentaires (ex : gras, italique).

Ce problème pourrait être réglé par des modifications du format SWML, en rajoutant des paramètres de police de caractère aux descriptions des signes.

Mais le format SWML [2001-DaRocha-SWML] apporte aussi ses contraintes.

3-2-2. Format SWML

Actuellement, il est nécessaire de développer une nouvelle application supportant SignWriting pour chaque besoin.

Citons ainsi SignMail pour le courrier électronique, SignPuddle pour les dictionnaires, SignWebMessage pour les discussions...

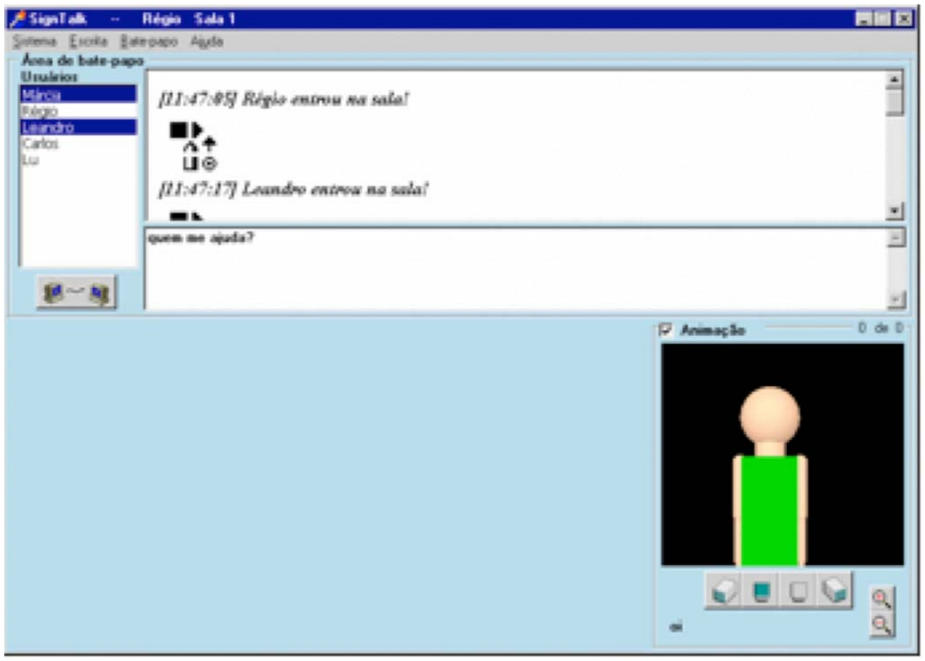


Figure 88: Logiciel de discussion en ligne SignTalk

Les applications doivent en effet offrir une méthode d'entrée et d'affichage SignWriting, en plus de devoir faire leur fonction dédiée, avec une interface souvent peu adaptée *[1954-Fitts-Law].*

(i) Conséquences pour le partage des données

Même avec des applications spécifiques, un problème de compatibilité revient lorsque les données doivent être partagées (mise en ligne, courriel, impression...).

En effet, soit l'application réceptrice comprend elle aussi le SWML, auquel cas il s'agit d'encore un autre application dédiée, soit un format de compatibilité doit être utilisé.

Étant donné l'aspect graphique de SignWriting, le format de compatibilité est la plupart du temps une image bitmap (PNG, GIF...) représentant les glyphes agrégés et positionnés.

La difficulté de stockage (taille), de recherche dans des documents, de mise à échelle pour l'impression revient alors.

(ii) Conséquences pour la localisation

De même, il est impossible de localiser simplement des systèmes d'exploitation. En effet, les mécanismes usuels reposent sur la mise en correspondance de chaînes à traduire avec les chaînes traduites (ex : "file" "fichier"), dans un encodage supporté.

XML n'est pas supporté par les systèmes d'exploitation, pas plus que des images pour remplacer ces chaînes traduites. Vu les problèmes supplémentaires de redimensionnabilité, il n'est pas possible d'envisager cette dernière méthode.

Il est donc impossible de localiser des systèmes d'exploitation en SignWriting.

(iii) Taille des fichiers

Dans tous les cas, le format XML est responsable de l'inconvénient majeur de SWML : la taille des fichiers produits. En effet, au minimum une ligne de XML est nécessaire pour positionner chaque symbole.

En conséquence, plusieurs dizaines de lignes de XML sont nécessaires pour positionner tous les symboles composant un signe.

Voici par exemple le signe « Brésil » :

Figure 89: « Brésil » en langue des signes brésilienne

Le fichier SWML correspondant à un signe aussi simple (deux symboles), est extrêmement long :

```
<?xml version="1.0" ?>
<swml version="1.0-d2" symbolset="SSS-1995">
     <generator>
          <name>SignWriter</name>
          <version>4.3</version>
     </generator>
     <sw_text>
          <sw_text_defaults>
               <sign_boxes>
                    <unit> pt </unit>
                    <height> 60 </height>
               </sign_boxes>
               <text_boxes>
                    <box_type> graphic_box </box_type>
                    <unit> pt </unit>
                    <height> 60 </height>
               </text_boxes>
          </sw_text_defaults>
     <new_line/>
     <sign_box>
          <!-- the B hand -->
          <symbol x="8" y="13">
               <shape number="21" fill="1" variation="1" />
               <transform flop="0" rotation="0" />
          </symbol>
          <symbol x="7" y="25">
               <!-- the movement -->
               <shape number="108" fill="0" variation="1" />
               <transform flop="1" rotation="4" />
          </symbol>
     </sign_box>
</sw_text>
</swml>
```

Figure 90: SWML correspondant à « Brésil »

Il s'agit là de la première version de SWML, basée sur la DTD95.

Bien que SWML ait été depuis amélioré dans un effort de concision, il reste nécessaire de faire un minimum de une ligne par symbole.

```
<?xml version="1.0"?>
<!DOCTYPE swml SYSTEM "http://www.signpuddle.com/swml/swml-s.dtd">
  <swml dialect="S" version="1.1" lang="sgn" glosslang="">
   <sign lane="0">
      <symbol x="96" y="98">03-01-001-01-01-01</symbol>
      <symbol x="131" y="116">01-05-032-01-02-01</symbol>
    </sign>
  </swml>
```

Figure 91: Équivalent de cet encodage avec la dernière version de SWML

Comme une phrase en langue des signes transcrite en SignWriting se compose de plusieurs symboles, chaque phrase nécessite de nombreuses pages de code XML.

Cette taille élevée des fichiers produits pose un problème de stockage, de retentissement certes limité vu la taille des dispositifs de stockage actuellement disponibles.

Elle pose surtout un problème pour les routines de manipulation des fichiers SWML, nécessitant beaucoup de ressources système pour manipuler de si gros fichiers.

La taille pose aussi un problème pour le partage de ces fichiers (courrier électronique, sites internet, téléchargement).

Ce problème n'est pas spécifique au SWML – il est commun à tous les formats utilisants XML. Plusieurs solutions ont été proposées *[2004-Bayardo-Binary_XML]*, fondées sur l'utilisation de formats binaires *[1999-Martin-Wap_XML]* ou de compression *[1999-Liefke-XMill]*, ce qui est toutefois contraire aux principes de XML *[1996-Bray-XML]*.

En conséquence, SignWriter a évolué lui aussi vers une version Java, SignWriter Tiger, qui reprend le format binaire SignWriter *[1995-Sutton-SignWriter]*.

3-2-3. Système d'exploitation

Mais même en modifiant le format des données, certains problèmes ne peuvent pas être résolus, puisqu'ils sont liés à une non intégration au système d'exploitation.

(i) Problème d'encodage

Les donnés « textuelles » sont représentées par des chaînes de caractères selon un encodage donné (ex : cp850, ISO 8859-15 latin 9, UTF-8).

Si la plupart des systèmes d'exploitation actuels gèrent ces multiples formats d'encodage, aucun ne gère l'XML en tant que format d'encodage. De mêmes, les formats binaires dits « propriétaires » comme SignWriter ne sont pas supportés au cœur du système d'exploitation.

Excepté de lourdes modifications dans le code source de chaque programme pour lui donner la capacité de gérer du SignWriting, il est impossible d'adapter tout un système d'exploitation à l'utilisation de SignWriting.

Il est donc actuellement impossible de généraliser l'utilisation de SignWriting aux différents logiciels d'un système d'exploitation : ces derniers doivent être conçus pour gérer le format voulu, ce qui restreint grandement le choix des programmes utilisables pour écrire en SignWriting.

Les problèmes de localisation du système d'exploitation et du partage de donnés persistent donc.

(ii) Duplication des fonctions

Les logiciels utilisant SignWriting gèrent ce problème en dupliquant des fonctions d'entrée/sortie appartenant au système d'exploitation.

Ils proposent par exemple un mécanisme propre d'entrée de donnés par des systèmes de menus, de glisser-poser, de claviers virtuels, etc.

Chaque programme réimplémente ainsi sa propre couche d'entrée, et duplique la fonction d'un logiciel natif (ex : courrier électronique, traitement de texte) afin d'y apporter le support de SignWriting.

Ce problème pourrait être contourné en utilisant des langages de programmation et des architectures orientées objet (ex : Corba).

(iii) Mode d'entrée

Mais même dans ce cas, il est quasiment impossible de s'interfacer avec le système d'exploitation.

En effet, toute les modifications des mécanismes gérant les périphériques d'entrée précédemment étudié (ex : au niveau de la double articulation), si elles sont faisables, sont particulièrement hasardeuses dans la mesure où elles interagissent avec des processus très complexes.

Donc si SWEdit impose un mode d'entrée, par un système de menus flottants manipulés à la souris, SignWriter DOS et SignWriter Tiger imposent aussi le leur.

Et il ne s'agit que de transpositions des menus claviers SSS en mode graphique ou en mode texte, sans possibilité de développer un mode de saisie indépendant de ces logiciels.

3-2-4. Conclusion

Que le logiciel soit graphique (Java) ou utilise DOS, que le format d'encodage soit basé sur XML ou non, il persiste des difficultés liées à la difficulté d'intégration dans un système d'exploitation.

En conséquence, il semble nécessaire d'envisager une resegmentation des tâches, faisant appel aux modalités de support étudiées dans l'état de l'art.

Les conclusions issues des aspects Unicode et Linux précédemment étudiés se rapprochent d'ailleurs méthodes employées pour supporter les langues écrites éloignées des langues latines *[2001-Ratheesh-Linux_Console_Indian]*, ou très différentes *[1999-Everson-Hieroglyphs]*, et même de celles extrapolées du support d'autres langues *[1996-Hasan-Sanskrit_Chinese].*

3-3. Resegmentation des tâches

Cette section à fait l'objet d'une communication à la conférence LREC 2004 *[2004-Aznar-FREU]*.

3-3-1. Introduction

En croisant les apports de l'état de l'art, les travaux sur le terrain et l'étude critique des systèmes d'informatisation de SW existants, il apparaît que les difficultés persistantes sont dues à une différence entre la méthode d'informatisation de SW et celles habituellement utilisées.

L'étude de l'implémentation des encodages et d'Unicode du système Linux nous apporte des pistes de travail.

En effet, on peut noter des l'existence séparée :

- d'encodage interne
- d'articulation avec les événements clavier
- de police de caractères

En conséquence, en adoptant une organisation globalement similaire, il devrait être possible de proposer une approche non monolithique, mais par couche.

3-3-2. Individualisation de couches

Toutefois, au cœur de cette approche est un encodage Unicode.

Avant d'individualiser des couches fonctionnelles, il faudrait donc disposer d'un encodage Unicode de SW.

Il a été noté les multiples verrous pour un support d'Unicode - il est donc nécessaire de postuler pour l'instant la possibilité d'un encodage Unicode de SignWriting, malgré ces difficultés précédemment notées.

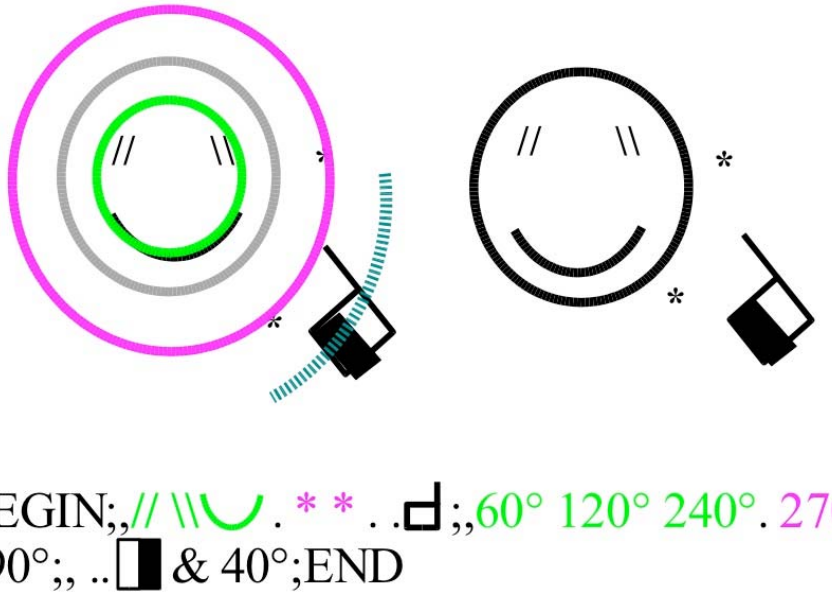
Afin d'avancer dans l'individualisation de couches fonctionnelles, proposons donc une approche arbitraire qui permettrait théoriquement de supporter Unicode ; la mise en œuvre d'un tel encodage sera ultérieurement étudiée plus en détail.

(i) Proposition d'un encodage Unicode

Pour résoudre les problèmes de positionnement relatif des symboles les uns par rapport aux autres, il est possible de considérer chaque signe sur un repère utilisant non pas les coordonnées cartésiennes, mais les coordonnées polaires.

Ainsi, chaque symbole du signe se voit attribuer des coordonnées relatives par rapport aux autres, de manière indirecte.

On peut alors proposer une grammaire de décomposition et de recomposition, complétant chaque couche par des informations angulaires.



BEGIN;,// \\◡ . * * . .◘;,60° 120° 240°. 270° 10° . . 290°;, ..◨ & 40°;END

Figure 92: Décomposition du signe « sourd » en LSF

(ii) Rappel sur le caractère arbitraire

Cette proposition d'encodage a pour seul but de donner une méthode pour positionner les symboles du signe SW, en séparant chaque élément utilisé de ses propriétés, et en produisant une séquence de codes ordonnées, à l'inverse de SWML.

De manière évidente, chaque signe SW pourrait utiliser une telle décomposition, puisqu'un signe est produit par le positionnement individuel de symboles sur un canevas à l'aide d'un logiciel comme SignWriter ou SWEdit : la décomposition peut se réduire au problème du passage d'un mode de coordonnées à un autre.

Chaque symbole nécessiterait alors un code Unicode donné, en considérant toutefois des possibilités de combinaisons pour appliquer des modifications géométriques sur ces symboles (orientation spatiale, rotation...).

Par contre, si la faisabilité est évidente, la simplicité l'est moins : il n'est pas garanti qu'une telle décomposition soit plus économe en code Unicodes que SWML, ou même que l'opération soit algorithmiquement plus simple.

De telles questions seront étudiées en détail. Pour l'instant, les conséquences d'un tel encodage doivent d'abord être analysées, et le fonctionnement expliqué.

(iii) Attribution de codes Unicodes

Chaque symbole se voit dans cette approche attribuer un ou plusieurs codes : dans l'exemple ci-dessus, l'index tendu a un code, la présentation perpendiculaire par rapport au visage un autre code.

On peut donc considérer que l'on attribue alors un code par «symbole de base».

Des « éléments de positionnement » servent pour leur part à manipuler les symboles sur les cercles concentriques :

- un pour séparer les couches successives, c'est à dire le début de chaque cercle concentrique (.)
- un pour indiquer quel type d'élément est manipulé (;) :
  - soit un symbole de base
  - soit ses coordonnées en degrés
  - soit une description des symbole de base par des « éléments d'information additionnels »

Les coordonnées sont indiquées de manière numérique, en utilisant au besoin des codes Unicodes pour correspondre aux valeurs angulaires les plus fréquentes, en découpant par exemple tous les 10°.

Le début et la fin d'un signe sont marqués par des « délimiteurs », ce qui permet d'envisager la gestion d'un mode d'écriture différent pour l'algorithme BiDi :

- un pour le début du signe (code BEGIN)
- un pour la fin (code END)

On peut alors proposer de manière totalement arbitraire une grammaire, en se basant sur quatre familles de codes:

- symboles de base
- éléments de positionnements
- éléments d'information additionnels
- délimiteurs

En comparant avec les codes Unicodes, les quatres familles proposées correspondent :

- aux caractères
- aux codes de ligature
- aux codes diacritiques (accents)
- aux codes de directionnalité

L'intérêt principal est d'utiliser un fonctionnement suffisamment similaire à Unicode pour pouvoir en envisager le support *[1999-Mudur-Shaping_Indian]*.

De nouveaux codes peuvent ensuite être proposés dans chaque famille suivant l'évolution de SignWriting, étant donné qu'il n'est rien postulé sur la composition précise des signes.

(iv) Critique

Cette approche est bien sur totalement arbitraire, comme précédemment rappelé, et de plus, elle pourrait même se montrer sous-optimale dans un certain nombre de cas.

Par exemple, si la plupart des signes sont organisés de manière concentriques, certains sont organisés de manière linéaire.

Il est toujours possible d'utiliser l'algorithme proposé pour les représenter au prix d'une légère lourdeur, une suite linéaire de ***n*** symboles nécessitant de définir ***n*** cercles concentriques où seule une position angulaire est utilisée.

Toutefois, même dans un tel cas limite, un nombre de codes inférieur par rapport à l'approche XML pourrait permettre d'encoder SW, puisqu'il est réalisé une décomposition dans un ordre donné et en séparant les symboles de base de leurs attributs.

En effet, là où XML note des valeurs numériques, des attributs et leurs valeurs numériques en codes ASCII, cette approche se contente d'une succession de valeurs numériques dans un plan Unicode.

Bien entendu, les avantages précis d'une telle méthode doivent être mesurés et comparés aux autres méthodes possibles avant de la valider. Une telle étude est réalisée à la fin de ce chapitre.

En attendant, ce postulat d'encodage permet d'envisager une nouvelle composition du problème, conforme aux bases d'Unicode : il s'agit d'une approche d'encodage dite « jetable », qui pourra être révisée ultérieurement.

3-3-3. Couche d'entrée

En suivant ce postulat d'encodage Unicode, on peut d'abord définir une couche d'entrée, dont le rôle est de placer les symboles les uns par rapports aux autres pour obtenir à partir d'un canevas graphique une décomposition selon cet algorithme.

L'intérêt majeur est de gérer cette couche d'entrée directement au niveau du système d'exploitation, pour éviter les lourdeurs d'entrée précédemment mentionnées.

Pour celà, un code, rentré par un dispositif quelconque (menu ou clavier), lorsqu'il est reçu par la gestion Unicode du système d'exploitation, lance la saisie spécialisée des autres codes, que ce soit par un dispositif visuel (type clavier virtuel, assistant de saisie) ou par combinaison de touches (type SSS clavier).

(i) Proposition de fonctionnement

Ce code peut provoquer soit un événement système, comme précédemment présenté, soit une composition par des mécanismes de ligatures, ayant un sens différent des lettres habituelles, grâce à des tables de compositions: il s'agit de mécanismes déjà existant avec les systèmes d'entrée des autres langues écrites.

Dans l'exemple d'une saisie au clavier, le code de délimiteur marque le début d'un premier cercle concentrique : d'autres peuvent être rajoutés. La sursaisie des symboles de base est effectuée avec leur positionnement, suivie au besoin des informations supplémentaires pour la sélection fine des symboles.

Le problème de l'entrée de SignWriting est donc déplacé au niveau du système d'exploitation, ce qui permet d'envisager une gestion plus souple et offrant plus d'alternatives.

Rappelons aussi la possibilité d'obtenir des caractères accentués soit de manière directe (è), soit de manière combinatoire (`+e=è) : à l'instar de cette méthode, on peut assigner au niveau de la table de clavier un ou plusieurs codes, au besoin sous forme de combinaisons de touches (comme présenté dans les tables de claviers) pour obtenir les symboles de base.

Cette couche d'entrée est ainsi séparée du reste du système, auquel elle n'est reliée que par sa dépendance en la couche d'encodage Unicode.

Plusieurs dispositions sont aussi envisageables selon les besoins spécifiques des langues des signes par pays.

(ii) Critique

La couche d'entrée correspond à un ensemble de règles de combinaison, voire à une interface d'entrée, déclenchée par la saisie d'un code particulier.

Par contre, il ne s'agit ici que d'une proposition de fonctionnement, qui a certes l'avantage d'être cohérente avec l'état de l'art et les possibilités logicielles offertes par les systèmes d'exploitation, mais pour laquelle il n'y a pas eu d'étude poussée.

Toutefois, des interfaces de saisie plus évoluées comme il existe pour les langues écrites usuelles *[2000-Ward-Dasher], [2005-Zhai-Sokgraph]*, ne peuvent être envisagées qu'une fois des mécanismes de composition unitaire définis ; il est aussi nécessaire d'avoir un mode d'ordonnancement défini ne posant pas de problème lors de l'utilisation *[2002-Poser-Athabaskan]*. Dans le cas de SW, plusieurs sont déjà proposés *[2001-Butler-SW_Ordering], [2004-Sutton-SSS]*.

L'approche proposée présente déjà des avantages par rapport aux approches existantes *[2005-Bray-Dictionnary]* qui connaissent des problèmes de compatibilité ou qui n'offrent pas à l'utilisateur de simplicité *[1954-Fitts-Law]* ou d'alternatives dans la saisie (menus SWEdit/SignWriter), ce qui semble pourtant après l'étude sur le terrain un problème prioritaire.

Un système d'aide à la saisie en SW mériterait d'être l'objet de travaux supplémentaires - contentons nous donc de proposer ce fonctionnement de base, qui permettra d'étudier plus bas les conséquences de problèmes spécifiques à SW (variabilité).

3-3-4. Couche de rendu

La manifestation physique de cette couche d'entrée est la couche de rendu : en effet, une fois une séquence de codes obtenue par quelque mécanisme de composition, l'algorithme postulé ne s'exprime que par la présentation à l'utilisateur du fruit de la composition graphique. La couche de rendu fonctionne à l'instar des autres modes de rendu des formes écrites par les moteurs unicode :

- les emplacement de positionnement des glyphes sont calculés en utilisant les couches concentriques et les valeurs en degrés

- les glyphes des symboles sont choisis en appliquant les règles de composition entre symboles de base et éléments d'informations supplémentaires

- les signes sont ainsi recréés

Cette couche consiste en un ensemble de calculs de positionnement et de règles de composition pour donner une représentation cartographique du signe.

Le fruit de cette couche est cette représentation inachevée, une reconstruction géométrique, où les glyphes doivent être effectivement insérés.

De cette couche de rendu, découle donc l'existence d'une autre couche : la couche de gestion des polices, ne serait-ce que pour contenir, comme présenté pour AAT, Opentype et Graphite, les informations utilisées par la couche de rendu sous forme de tables.

3-3-5. Couche de gestion des polices

À partir de la représentation inachevée, une police de caractère SignWriting doit être utilisée : prenons l'exemple de la collection de symboles au format SVG.

Les éléments graphique sont dimensionnés et insérés dans cette représentation inachevée, qui peut être passée au système d'exploitation et effectivement affichée.

Le fruit de cette couche est donc la reconstruction graphique finale, affichable.

3-3-6. Intérêt

L'intérêt et les conséquences sur le support logiciel sont la généralisation des mécanisme de support de SW : un élargissement du support aux autres logiciels, et une localisation du système d'exploitation, nécessite un support Unicode.

En effet, comme étudié dans l'état de l'art, dès qu'un formalisme d'écriture est supporté par Unicode, le moteur d'unicode du système d'exploitation et la police choisie, toute application peut afficher des textes dans ce formalisme d'écriture.

3-3-7. Discussion

Cette partie sur la resegmentation des tâches tranche profondément par rapport aux parties précédentes et même à l'état de l'art, qui étaient beaucoup plus détaillés. On note aussi beaucoup de propositions, mais pour l'instant peu d'évaluations et de mises en œuvre.

Il y a deux raisons pour cette approche très générale et plus orientées vers les propositions que la réalisation :

- tout d'abord, il est nécessaire de ne pas dépendre trop fortement des spécificités de SW, alors que d'autres formalismes graphiques peuvent ultérieurement être amenés le remplacer - comme par exemple un formalisme issu de LS-SCRIPT *[2007-Garcia-LSSCRIPT].*

La difficulté de cette partie est donc de faire l'équilibre entre la nécessité d'une approche détaillée des problèmes d'écriture de la langue des signes, et une approche très générale, extensible à d'autres formalismes d'écriture.

Ainsi, la resegmentation se contente d'isoler les propriétés les plus fondamentales de SW, qui ont le plus de probabilité d'être conservées dans un nouveau formalisme graphique, sans se focaliser sur des détails spécifiques à SW.

- ensuite, la présentation des travaux suit l'ordre de réalisation de ces derniers, la resegmentation du problème général en sous problèmes particuliers ayant été l'une des premières étapes pratiques.

Cette resegmentation, orientée par l'état de l'art, a permis d'isoler un nombre important de sous-problèmes - ici par exemple, chaque couche fonctionnelle peut être considérée comme un sous-problème séparé.

Se pose donc la question de leur mode de présentation. Il a été fait le choix de présenter tout d'abord leur articulation générale, pour faciliter la compréhension par rapport aux besoins notés dans l'état de l'art et sur le terrain.

Cette présentation est toutefois complétée plus bas par des approfondissement sur plusieurs points particuliers, qui sont repris et détaillés, comme par exemple les possibilités d'encodage qui sont étudiées comparativement.

Par contre, il n'a pas été possible d'étudier de manière complète la liste exhaustive de chaque aspect de chaque problème.

Certains ont ainsi été laissés à part, pour des travaux ultérieurs, en se concentrant sur les aspects les plus bloquants pour une informatisation d'une forme écrite de la langue des signes (verrous scientifiques).

3-3-8. Conclusion

En suivant le postulat d'un encodage Unicode, qui est détaillé ensuite, on peut proposer une resegmentation du problème conforme au fonctionnement des systèmes d'exploitation modernes, tel que détaillé dans l'état de l'art.

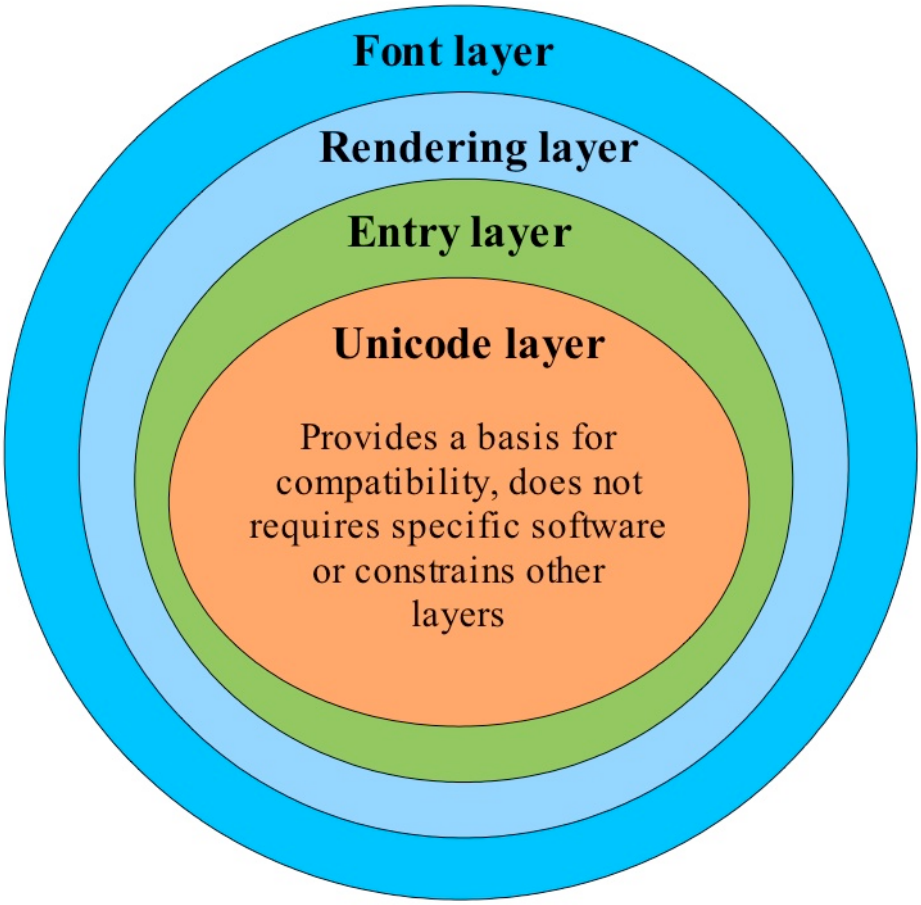


Figure 93: Hiérarchie des couches

L'existence de chaque couche peut être déduite de la couche précédente, d'où l'intérêt de postuler l'existence de la couche Unicode initiale avec l'algorithme proposé.

(i) Implémentation

Mais l'implémentation de certaines couches va requérir plus de travail que d'autres :

- la partie la plus complexe est la définition des règles de rendus, qui lient composition et décomposition,

- le plus long est la fabrication de police de caractères compatible OpenType,

- le plus important est la couche d'entrée qui ralentit le plus les utilisateurs lors de l'étude,

(ii) Espace Unicode

La réservation d'espace Unicode pour attribuer des codes pour ces besoins internes est un problème restant ouvert, car l'encodage Unicode de SignWriting peut tirer partie des propriétés précédemment énoncées dans l'état de l'art:

- moteur de rendu unicode : pour limiter le nombre de caractères nécessitant un code unicode spécifique

- plan spéciaux réservés à l'usage privé : pour préparer un système expérimental, et présenter un prototype avant

- encodages UTF-8, UTF-16, UTF-32 : pour aider à choisir un mécanisme facilement généralisable

- caractère de remplacement unicode : pour indiquer que du SignWriting ne peut être correctement affiché

- sous-ensembles Unicode et combinaison de polices : pour faciliter le déploiement rapide d'une solution Unicode

- redondance : pour garder n approches si elles sont intéressantes dans des cas particuliers

Mais quelle est la taille de l'espace Unicode nécessaire ?

Est-ce qu'il est possible d'obtenir des gains de vitesse ou de place au niveau des algorithmes par une assignation intelligente de ces codes ?

Ces questions sur l'espace Unicode sont traitées plus loin, avec une étude comparative des possibilités d'encodage.

(iii) Standardisation

L'important est surtout une standardisation au niveau d'Unicode, pour que le résultat soit affichable sur tout système d'exploitation et toute application supportant Unicode, sans risque de conflit ou d'incompatibilité.

Il s'agit même du plus grand intérêt de cette approche. Comme expliqué dans l'état de l'art en reprenant le cas de Pango sous Linux, tous les logiciels des suites KDE et GNOME pourraient après cette implémentation bénéficier de ces apports, sans devoir réécrire un nouveau logiciel SignWriting pour chaque besoin spécifique, puisque par exemple ces deux suites font appel à Pango.

Pour l'instant, les logiciels gérant les langues signées utilisent des encodages incompatibles avec les autres langues écrites - la standardisation serait aussi source de compatibilité.

Cette phase de définition d'un nouveau standard permet donc d'étendre le support, et de régler les problèmes de partage de données.

(iv) Bénéfice secondaire des couches fonctionnelles

La resegmentation du problème permet aussi la portabilité de l'approche, notion très intéressante.

Au delà du bénéfice pour un système d'exploitation donné, comme vu dans l'état de l'art, il faut rappeler que la gestion d'Unicode est globalement similaire au niveau des trois systèmes d'exploitation les plus populaires (Windows, Linux, MacOSX).

Il est donc possible d'envisager un support au niveau des différents systèmes d'exploitation, y compris mobiles, sans supplément notable de travail.

Les seules véritables différences sont dans l'implémentation des concepts d'articulation multiple de la couche d'entrée, et dans le détail du rendu par les moteurs (ex : tables ou automate).

(v) Réutilisabilité

La segmentation du problème permet aussi d'envisager des approches itératives jetables pour chaque sous système, en remplaçant au besoin chaque couche au fur et à mesure des évolutions sans devoir redéfinir l'intégralité du système, comme c'est pourtant le cas avec les approches monolithiques comme SWEdit.

Des changements lourds de conséquence au niveau applicatif, comme une autre directionnalité d'écriture, ou de nouvelles polices de caractères, deviennent simples à gérer et déployables à l'ensemble des logiciels utilisant un système d'exploitation compatible.

(vi) Influence sur le niveau de support

Il est ainsi possible de prévoir un niveau de support élevé, pouvant aller jusqu'à la location du système d'exploitation.

La localisation passe en effet par la fourniture à l'application de « traductions » des textes devant être affichés, comme vu dans l'état de l'art avec l'exemple de « ls » sous « bash ».

Toutefois, des contraintes spécifiques de SignWriting doivent alors être envisagées : en effet, la plupart des langues utilisent un encodage unidimensionnel.

(vii) Limites : présentation et mode d'écriture

Les langues latines s'écrivent de gauche à droite ; d'autres dans un sens différents (droite/gauche, ou haut/bas).

Ces cas relativement simples, restant unidimensionnels, sont toutefois relativement complexes au niveau applicatif, et peuvent nécessiter une modifications des logiciels.

Mais dans le cadre de la localisation par exemple, les menus d'une langue qui ne s'écrirait que de haut en bas ne peuvent plus être situés en haut de l'écran, et doivent être positionnés sur les côtés. Il faut sinon changer la directionnalité d'écriture de cette langue.

L'ordre d'affichage de la plupart des langues supporte toutefois des contraintes légères, et le texte même dans un alignement non habituel reste lisible : ainsi, les langues asiatiques conservent les menus en haut de l'écran.

Dans le cas retenu d'un encodage le plus riche possible des langues signées, la possible bidimensionnalité de l'écriture peut par contre avoir pour conséquence une inadéquation entre l'espace nécessaire pour afficher par exemple les menus, et l'espace requis.

Une illustration de ce problème est l'interlignage plus élevé nécessaire aux polices de caractère Vietnamiennes, comme illustré dans l'état de l'art

Les conséquences d'une informatisation devraient donc être mesurées, surtout au niveau de la lisibilité, en cas de localisation.

Un tel travail, hors du cadre de cette thèse, devrait être mené par des ergonomistes et des typographes, pour étudier les adaptations éventuelles à appliquer aux polices de caractères pour répondre aux contraintes informatiques spécifiques. Dans le domaine des langues écrites usuelles, de telles adaptations se font dans des circonstances expérimentales *[2006-Chaparro-Legibility]*, suivant des théories psycholinguistiques *[2004-Larson-Recognition].*

3-4. Problème des variabilités

Cette section à fait l'objet d'une communication à la conférence TALN 2005 *[2005-Aznar-Variabilités].*

3-4-1. Identification du problème des variabilités

L'encodage proposé reste arbitraire, et d'autres pourraient potentiellement exister. Mais quelque soit cette approche, elle permet d'étudier et de comprendre des problèmes qui n'apparaissent qu'en considérant un encodage Unicode.

Ainsi, si l'on regarde la production de signes SW à partir d'une langue signée, comme par exemple auprès des élèves de l'Escola Frei Pacifico, on relève une phase de décision sur l'aspect graphique final.

En effet, le professeur dessine au tableau une transcription, qui est discutée avec les enfants pour choisir les symboles faisant partie de ce signe, avant d'être acceptée.

Le choix de la manière de réalisation spatiale du signe (mouvement précis 3d+t) n'entre pas dans le propos de la discussion : considérons juste qu'un seul signeur réalise un signe lors d'une itération unique, et que plusieurs personnes ayant à écrire ce signe en SW comparent ensuite leurs résultats.

Une variabilité entre les sorties réalisées (des écrits), pour une même entrée (un signeur) peut alors se voir. Ce problème apparaît d'ailleurs aussi dans les manuels d'écriture en SW, qui donnent les possibilités existantes pour transcrire un même signe, tout en retraçant l'historique de leur évolution.

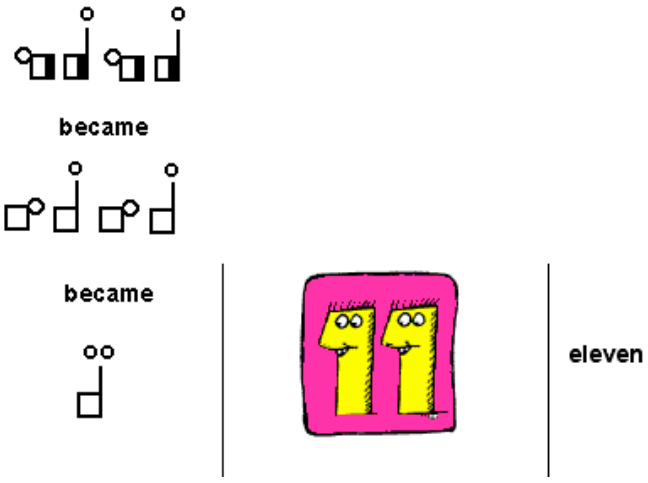


Figure 94: Chiffre 11 en langue des signes américaines

Ainsi, le manuel d'écriture révèle 3 possibilités pour écrire le chiffre 11, même si la dernière est préférée ; on peut d'ailleurs comparer l'écriture de 1, 11, 2 et 12 pour déduire que de telles variations doivent exister assez fréquemment pour d'autres signes.

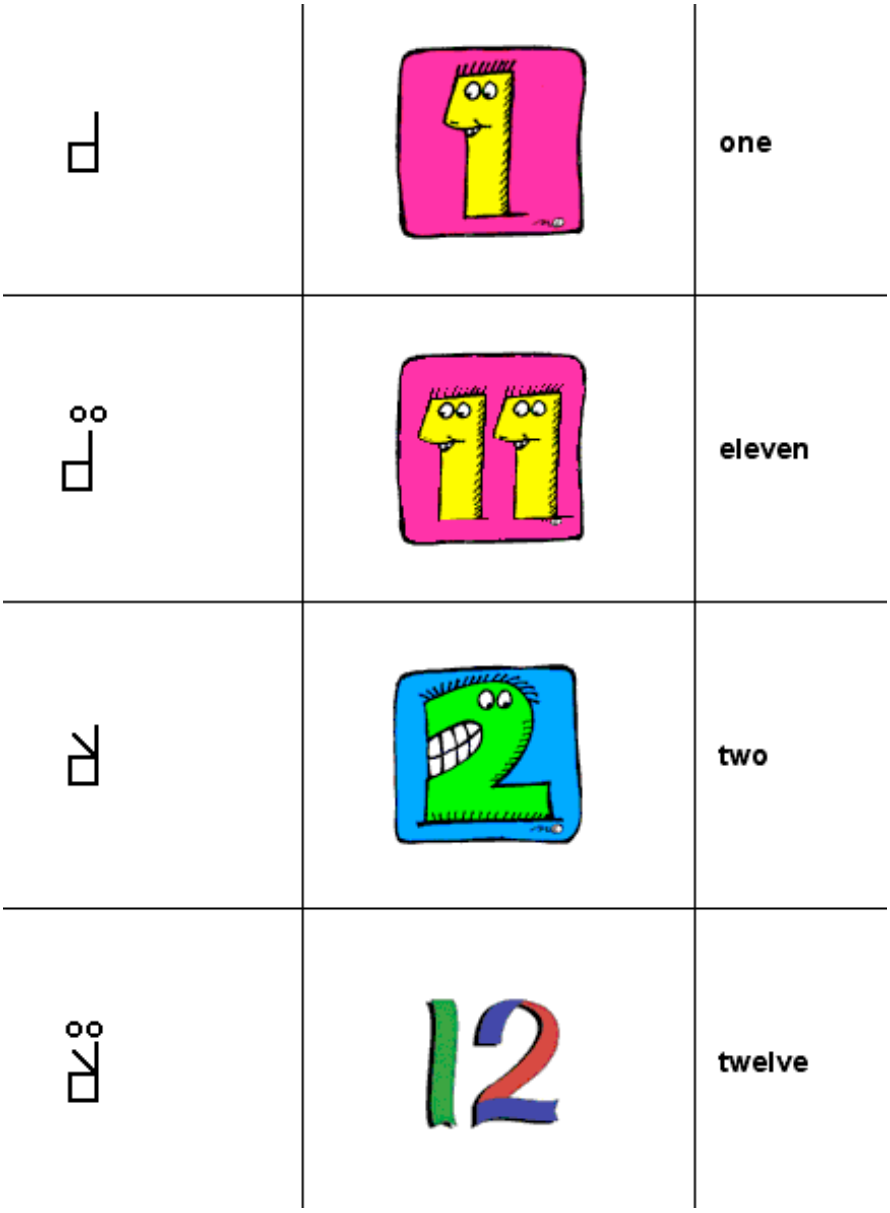


Figure 95: 1, 11, 2 et 12 en langue des signes américaine

Mais y a-t-il des cas où une forme est préférée, et des cas où toutes les formes devraient être considérées ?

Étudions pour répondre à cette question la manière pour différents locuteurs de transcrire un même signe.

3-4-2. Mise en évidence de la variabilité inter-personelle

Prenons par exemple le signe « sourd » en langue des signes française. La description de ce signe est assez simple : l'index tendu de la main dominante vient toucher l'oreille puis la bouche, éventuellement accompagné d'un mouvement simultané des sourcils vers le haut et des lèvres vers l'avant.

En SignWriting, vu la grande permissivité du formalisme d'écriture, « sourd » peut être représenté de plusieurs manières possibles:

Figure 96: « Sourd » en langue des signes française

Dans le premier cas, le mouvement des sourcils a été considéré comme un plissement des yeux, et avec deux contacts au niveau du visage est l'un des attributs principaux du signe.

Dans le second cas, le premier contact est remplacé par le mouvement des yeux.

Il serait toutefois possible d'associer les deux contacts et le mouvement des yeux, ou de faire tout autre combinaison de ces éléments, et de même, d'autres variations mineures aboutiraient à d'autres modifications ; mais dans tous les cas, ces modifications toucheraient au choix des symboles utilisés dans ce signe.

Au final, l'usage fait que les formes de « sourd » représentées s'imposent parmi toutes les variations envisageables : elles sont donc préférées aux autres variations.

(i) Discussion

Dans les langues écrites, comme le français, de tels équivalents orthographiques existent : citons « clef » et « clé ». Le choix des symboles peut être comparé aux choix des lettres pour le mot d'une langue écrite unidimensionnelle.

On peut aussi considérer que cette variabilité vient plutôt de l'ambiguïté de SW, qui est à la fois une écriture et un formalisme de notation : c'est alors le caractère partiellement phonétique de SW qui fait que deux personnes vont percevoir de manière légèrement différente la production d'un même signe, ou signer différent ce signe, et donc transcrire cette différence dans l'écriture de SW.

Le cas particulier du choix de repère pour représenter un signe en SignWriting (XY, YZ, XZ) n'est volontairement pas abordé ici, puisque un tel choix de repère aboutit sur une variation de certains symboles utilisés pour transcrire le signe.

Dans tous les cas, on se ramène à une variabilité inter-personelle.

3-4-3. Mise en évidence de la variabilité intra-personelle

Une autre variabilité existe aussi. Considérons le premier cas, et postulons qu'il s'agit de la seule forme admissible pour écrite « sourd » en SignWriting.

Figure 97: Entrée de dictionnaire de langue des signes américaine

Demandons alors à une personne maîtrisant bien SignWriting de reproduire à l'identique ce signe, un certain nombre de fois, sans en changer aucunement la composition en symboles.

Des variations mineures vont toutefois apparaître entre chaque itération :

Figure 98: Deux itérations du signe sourd

Elles sont dues à la méthode de positionnement des symboles, utilisée par la méthode d'entrée : ici un déplacement à la souris.

Les coordonnées précises de ce déplacement sont variables, et donc la position relative des symboles les uns par rapport aux autres l'est aussi

De telles variations, quasiment invisibles, se traduisent toutefois par des modifications importantes au niveau de l'encodage, puisque les coordonnées précises des symboles varient. Les algorithmes de reconnaissance doivent donc faire preuve d'une certaine « tolérance » pour identifier que ces deux itérations correspondent à un seul et même signe *[2004-DaRocha-Sign_Matching].*

Même en cas de l'utilisation de fonctions « copier-coller », où la position relative des symboles les uns par rapport aux autres ne change pas, la position absolue est variable.

En effet, elle dépend de l'endroit où le groupe de symboles se retrouve mis dans la cellule SignWriting.

Cette variabilité intra-personelle n'a pas d'équivalence dans les autres langues écrites, si ce n'est dans le cas où l'écriture se fait à la main : cette variabilité n'a pas d'autre explication que le caractère similairement analogique de l'écriture manuelle.

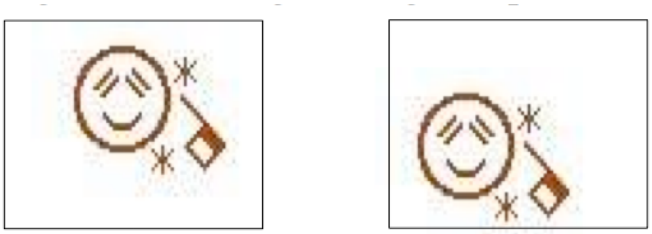


Figure 99: Variabilité de groupe

Pour établir une comparaison avec l'écrit, le glyphe de chaque lettre réalisée a une forme unique car analogique, alors que sur un ordinateur le glyphe de chaque lettre réalisée est identique, car numérique - en supposant juste que la police ne varie pas et qu'aucun autre mécanisme comme les ligatures ou les accents ne viennent s'ajouter.

3-4-4. Bilan des variabilités

Nous avons donc plusieurs types de variabilité :

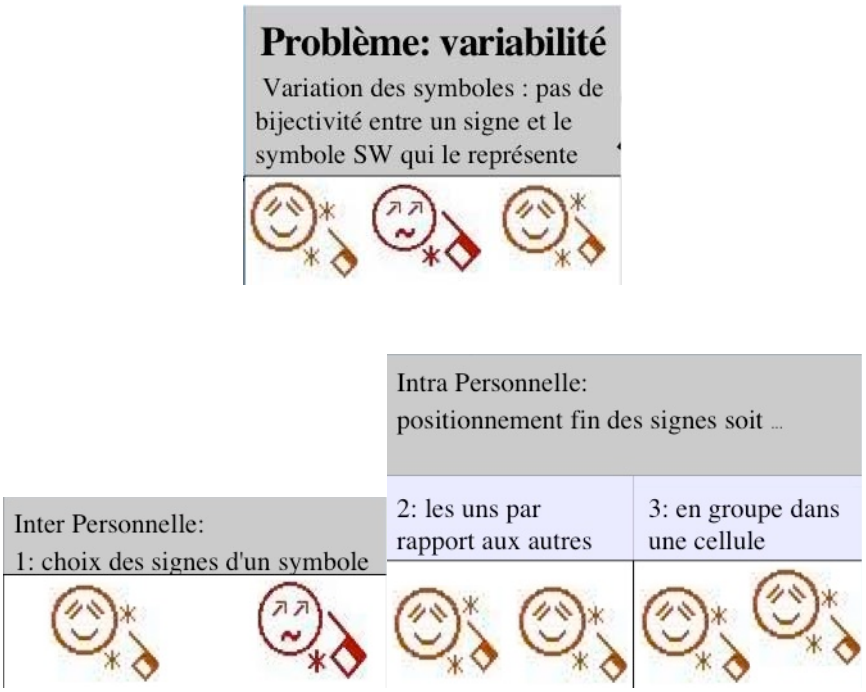


Figure 100: Illustration résumant les variabilités

L'existence de ces variabilités est liée à fois à la souplesse du formalisme d'écriture SignWriting, permettant de choisir des symboles et de les positionner librement qu'aux méthodes d'entrée et de positionnement des symboles en SignWriting.

Mais considérons maintenant les conséquences de ce problème.

3-4-5. Conséquences

La conséquence de cette variabilité est une non-bijectivité entre la combinaison finale de glyphes et le signe original, sans que cette différence ait une raison sémantique.

Ainsi, un système de traitement d'image utilisé comme mécanisme d'entrée devrait permettre d'obtenir pour les variabilités 2 et 3 de l'exemple ci-dessus un ensemble de codes identiques, puisque l'intention de l'utilisateur était d'entrer une variante donnée de « sourd ». En pratique, à moins qu'il ne soit prévu de gérer cette variabilité, ce ne sera pas le cas.

Dans le cas de la variabilité 1, un système de traitement d'image devrait par contre considérer la probabilité qu'il y ait soit une variante de « sourd », soit une fausse reconnaissance de « sourd ».

En pratique, si la différence est traitée, cela représente un coût temporel non négligeable *[2004-DaRocha-Sign_Matching]*, car la conséquence de cette dualité est une complexité algorithmique pour les problèmes de recherche textuelle (regexp) qui, dans ce cas, restent donc du domaine de la recherche *[2004-Aerts-Searching-SignWriting]*.

Ceci pose un problème principalement pour les applications bureautiques, mais aussi pour le partage de données *[2004-Crasbord-Sharing_data]* et pour la création de dictionnaires *[2004-Roald-Making_dictionnaries]* complets.

La méthode la plus efficace *[2004-DaRocha-Sign_Matching]* considère d'abord l'ensemble de symboles correspondant aux variantes du signe recherché, puis l'ensemble de positions possibles de chaque ensemble de symboles. Ensuite, les fichiers au format SWML sont parcourus, et il est calculé un score de vraisemblance pour chaque signe.

Cet algorithme n'est pas améliorable dans l'état actuel de la problématique. Faudrait-il donc neutraliser, où conserver ces variabilités ?

3-4-6. Neutraliser ou conserver les variabilités

Les variabilités peuvent être considérées soit comme un aspect intéressant et positif, soit comme un problème d'encodage à supprimer.

Regardons de plus près ce dilemme.

(i) Neutraliser

Pour neutraliser ces variabilités, plusieurs arguments existent :

- faciliter et accélérer les recherches textuelles

- éviter les problèmes de positionnement manuel des symboles, liés à la nature sous-contrainte des éditeurs actuels en SignWriting

- accélérer et simplifier la saisie des signes

- unifier les variabilités sémantique en LSF, dans une phase équivalente à celle de « consolidation orthographique » qu'ont connues les langues écrites

- réduire la combinatoire SWML, dans le but d'une simplification de la LSF

- faciliter la compréhension, les lecteurs d'un document pouvant ne pas connaître toutes les variantes d'un signe, ou même ne pas les comprendre si l'amplitude des variations est trop élevée

En effet, les théories généralement acceptées sur la reconnaissance de l'écrit font appel à une reconnaissance des formes des mots *[2004-Larson-Recognition]*.

La variabilité inter-personelle dans un cas extrême pourrait opposer deux variantes des plus différentes imaginables possibles, ce qui nuirait à la compréhension du signe.

- maintenir la cohérence interne d'un document, au cas où plusieurs variantes y coexistent

(ii) Conserver

Toutefois, plusieurs arguments s'y opposent :

- tout d'abord, le respect de la nature de la langue des signes, qui admet des variabilités dans la réalisation des signes, et qui même les utilise pour transporter des informations supplémentaires (iconicité: objet plus ou moins gros selon le gonflement des joues), voudrait que l'on ne neutralise pas ces différences.

- de plus, les variabilités des formes standards pourraient faire l'objet d'études pour les linguistes *[2007-Boutet-Structuration_LSF]*, afin de mesurer les évolutions des langues signées au cours du temps. De telles études ne seraient pas possible sans conservation de cette variabilité.

- ensuite, (option à la demande de l'utilisateur) la possibilité de correction pour les reconnaissances erronées, qui nécessite de garder la forme rentrée pour offrir les autres reconnaissances alternatives, ou surtout de compléter la forme rentrée, nécessite aussi de garder la variabilité.

- enfin, l'évolutivité de SignWriting, qui fait que des formes « standard » de signes, si elles existent, ne sont que temporaires, fait considérer la neutralisation comme l'équivalent d'un solution temporaire, dont les conséquences futures seront négatives.

Étudions toutefois les solutions envisageables pour chaque argument.

(iii) Solutions possibles pour neutraliser ces variabilités

Pour neutraliser les variabilités, plusieurs solutions existent selon le niveau d'application de cette neutralisation :

- si elle s'effectue dès la phase d'entrée, il est possible de discrétiser les coordonnées de positionnement des symboles pour créer des points d'ancrage « magnétiques » visuellement très distinguables.

Cette solution est appliquée dans les logiciels de dessins ainsi que par les autres langues écrites : par exemple, un accent circonflexe ne peut adopter qu'une certaine position sur le e.

Il s'agit alors d'une neutralisation liée aux possibilités retenues d'encodage.

La proposition d'encodage Unicode, qui utilise des couches concentriques et définit des valeurs angulaires, permet dans une certaine mesure de neutraliser ces variations.

Toutefois, dans le cas de la neutralisation des variabilités intra-personelles, il ne s'agit que d'une neutralisation partielle, liée au facteur de discrétisation d'un positionnement angulaire.

De même, l'ordre des couches peut toujours changer, et la variabilité inter-personelle n'est même pas traitée par cette approche.

La proposition d'encodage Unicode précédemment réalisée ne doit donc être considérée que comme une étape nécessaire, un postulat, et non une résolution du problème des variabilités.

- si elle s'effectue immédiatement après l'entrée, un agent logiciel peut venir remplacer une variante par sa forme standard, pour interagir avec l'utilisateur

Cette solution d'assistant d'entrée est appliquée par les systèmes de reconnaissance d'écriture des assistants personnels, par les systèmes où l'utilisateur se voit proposer la liste des choix possibles prédits à partir du segment afin d'accélérer l'entrée *[1996-Horstmann-Word_predicting]*, y compris lorsqu'elle s'effectue à la volée *[2000-Ward-Dasher]*

Il s'agit alors d'une neutralisation lors de l'étape d'entrée.

- si elle s'effectue ultérieurement, le remplacement n'est effectué qu'après que la forme variante ait été rentrée, affichée, vue et manipulée

Dans le cas des traitements de textes, la correction orthographique peut aussi être automatiquement effectuée après la saisie ou sur demande.

Il s'agit d'une neutralisation contextuelle, lors de l'étape de rendu, qu'elle soit automatique ou non (table morx type 1 ou 4).

Une approche algorithmique ou une approche par dictionnaire peuvent permettre de neutraliser les deux variabilités. Toutefois, elles vont souffrir de coûts de traitement élevés, à l'instar des solutions de rechercher/remplacer proposées dans la littérature, puisqu'il s'agit alors d'effectuer le même algorithme, mais sur chaque signe de l'ensemble du document avant de l'afficher.

De plus, aussi bien par algorithme que par dictionnaire, une telle neutralisation ne pourrait faire intervenir des décisions supplémentaires de l'utilisateur, comme par exemple de ne pas procéder à la neutralisation dans un cas précis voulu, un exemple qui se retrouve avec Microsoft Word, remplaçant tout le temps les guillemets anglais par des guillemets français, y compris dans du code source informatique...

Ces différents moments d'applications de la neutralisation pourraient aussi se voir considérés sur le même plan, en séparant juste la possibilité d'interaction de l'utilisateur de la possibilité de correction automatique.

Toutefois, la différence de moment d'application a un effet direct sur la rapidité de saisie et la reconnaissance du document par l'utilisateur, qui peut s'étonner que le système ne lui permette pas d'entrer ce qu'il désire, ce qui peut aussi le ralentir.

Ce sentiment d'absence de contrôle du logiciel peut provoquer une frustration de l'utilisateur, et il peut être donc souhaitable, pour éviter cette frustration et les problèmes précédemment mentionnés de conserver les variabilités.

Étudions donc les solutions correspondantes.

(iv) Solutions possibles pour conserver ces variabilités

Il est aussi envisageable de séparer plusieurs approches selon les possibilités d'automatisation, et donc au choix :

- de compléter la forme rentrée par une forme standard, automatiquement reconnue

- de laisser à l'utilisateur le soin de définir le lien avec une forme standard, voire même cette forme standard.

De la même manière, selon le moment où ces approches sont réalisées, on peut séparer les mêmes trois grands types de solution :

- immédiatement lors de l'entrée

Cette approche est adoptée par les systèmes de décomposition et de recomposition Unicode (NFC/NFD), qui selon le système d'exploitation (ex : MacOSX-NFD) substituent immédiatement à un code composé l'équivalent des codes décomposés, sauf dans le cas où cette décomposition n'est pas connue ou au contraire interdite.

- juste après

Le système Active Ink de Microsoft *[1998-Schilit-Active_Reading]*, immédiatement après une saisie d'écriture, remplace les caractères saisis par les caractères reconnus. Toute interaction ultérieure (ex : correction, précision) ramène la forme saisie

- ultérieurement

Les différents systèmes de reconnaissance de caractère (OCR) existants permettent, par exemple dans le cas de documents PDF, de superposer à l'image affichée un texte reconnu et mis par transparence sur cette image.

3-4-7. Approche proposée

(i) Conserver les variabilités...

Une évaluation linguistique de SignWriting semble nécessaire pour déterminer la valeur de ces variabilités, et vérifier qu'elles n'aient pas une fonction sémantique particulière à l'instar des variabilités orthographique dans les langues écrites, comme "color" et "colour" sont respectivement la forme états-unienne et la forme anglaise du même mot, et permettent donc de tirer des conclusions sur l'origine du texte.

Dans le domaine des langues écrites dites naturelles, des études sur le sujet ont été réalisées par des psychologues, avec notamment des modélisation de l'ambiguïté sémantique *[2004-Rodd-Semantic_ambiguity_modelling]*.

Cette thèse est toutefois une thèse d'informatique - une telle évaluation linguistique n'a donc pas été réalisée.

(ii) ... Mais neutraliser leur influence

D'un point de vue informatique par contre, l'évaluation de l'influence de ces variabilités peut se faire en étudiant la littérature des problèmes de « rechercher/ remplacer » en SignWriting dont l'algorithme a précédemment été commenté.

Exceptés les cas où de telles fonctions sont nécessaires au but même du logiciel et n'impactent pas fortement sur les performances, comme dans le cas de dictionnaires *[2004-Aerts-Searching-SignWriting] [2005-Aerts-Semantic-Searching]*, on peut constater que le coût temporel de gestion de ces variabilités est très élevé *[2004-DaRocha-Sign_Matching]*, puisque les fonctions de recherche dans un document sont l'équivalent de programmes de traitement et de compréhension d'image.

Les solutions proposées reposent sur des formules demandant soit un nombre de paramètres liés à la dynamique du signe comme la forme de la main, la direction du mouvement, le type de contact éventuel *[2004-Aerts-Searching-SignWriting] [2005-Aerts-Semantic-Searching]* soit la décomposition en symboles de ce signe *[2004-DaRocha-Sign_Matching]*.

Le problème est encore plus important pour une analyse lexico-sémantique *[2002-Huenerfauth-ASL_Generation]* de documents en SignWriting: quelque soit l'encodage, dans la mesure où un même signe peut être représenté par plusieurs signes SignWriting, toutes les équivalences seraient à alors considérer pour neutraliser l'effet des variations amenées par le formalisme d'écriture, et ne s'intéresser qu'au signe initialement transcrit.

(iii) Motivations

L'outil informatique ne devant pas imposer ses contraintes à l'utilisateur au delà du raisonnable, il semble donc souhaitable de conserver les variabilités pour ne pas réduire le problème.

Mais l'utilisateur des différents logiciels permettant de saisir des signes SW procède toujours de la même manière: il choisit le groupe des symboles de base, puis celui qu'il veut entrer, pour ensuite le positionner le symbole sur le plan bidimensionnel constitué par la cellule SignWriting.

Il ne lui est pas d'ailleurs offert le choix de procéder autrement, en l'absence de mécanisme alternatif efficace d'entrée .

Si certains logiciels proposent des dictionnaires permettant de saisir directement un signe, l'aide à la saisie n'est pas encore gérée: l'utilisateur doit encore une fois choisir et positionner son symbole.

La recherche d'un signe SW dans un document ou un dictionnaire constitue donc un problème de recherche à part entière.

Il faut donc étudier les possibilités de neutraliser l'influence de ces variabilité sur cet aspect précis de la problématique.

(iv) Choix

Il est donc proposé de neutraliser les variabilités lors de la phase d'entrée, ce qui est la solution qui présente le moins d'inconvénients et qui a d'ailleurs été déjà adoptée par les systèmes d'aide à la saisie des ordinateurs mobiles type tablette ou assistant numérique personnel avec par exemple le système Active Ink de Microsoft *[1998-Schilit-Active_Reading]*.

Les coûts de traitement ultérieur (rechercher/remplacer) peuvent être réduits en cachant les résultats des calculs dans le format sauvegardé, par exemple sous forme de méta-donnée.

En se situant dans l'étape d'entrée, cette approche permet à l'utilisateur de préciser ses intentions, par exemple en cas d'erreur de reconnaissance.

Un intérêt supplémentaire serait de compléter l'approche algorithmique et par dictionnaire d'une approche statistique, pour raccourcir la durée de saisie *[1996-Horstmann-Word_predicting].*

Celà peut se faire en présentant les symboles les plus fréquemment mis en premier à l'utilisateur, ou par le biais d'un système d'aide et de positionnement globalement similaire *[2005-Zhai-Sokgraph]*.

Étudions donc maintenant les deux parties de l'approche proposée : neutralisation et conservation.

3-4-8. Algorithme de neutralisation

Cette section à fait l'objet d'une communication à la conférence ÉDIT 2005 [2005-Aznar-Approche].

(i) Choix de l'utilisateur

Au lieu de ne laisser à l'utilisateur qu'un choix des symboles de base par des menus, qu'ils soient accessibles à la souris ou au clavier, les symboles les plus fréquemment mis sur le canevas de saisie sont co-affichés dans un autre menu.

Ce dernier est mis à jour au fur et à mesure de la saisie, de manière similaire aux algorithmes T9 des téléphones mobiles qui complètent les choix possibles au fur et à mesure que des caractères saisis permettent d'isoler des branches sur un arbre de possibilités.

Ainsi, ce menu est interactif : les symboles qui y figurent sont contexte-dépendants, et varient selon les choix de l'utilisateur.

En tirant parti des choix qui sont réalisés, il est donc possible de faire un système qui « apprend », pour répondre au mieux aux besoins de l'utilisateur.

Une telle approche, bien que moins dynamique que Dasher *[2000-Ward-Dasher]*, offre des raccourcis en plus des menus traditionnels.

(ii) Positionnement automatique des symboles

Aussitôt qu'un symbole est choisi dans l'un des menus, il est automatiquement positionné sur le canevas bidimensionnel à la position finale la plus probable.

Cette position finale est calculée statistiquement comme le résultat d'une fonction prenant en entrée la liste ordonnée des symboles déjà rentrés et positionnés, et le symbole courant.

Les logiciels existants, comme SWEdit, mettent par défaut les symboles au centre ou dans un angle du canevas.

En calculant une fonction statistique, il est possible pour le système d'apprendre au fur et à mesure des entrées qui sont réalisées par l'utilisateur : chaque signe étant composé d'un ensemble de symboles, le choix répétitif d'une liste ordonnée de symboles aboutira ainsi au positionnement automatique de ces symboles, accélérant la saisie.

L'utilisateur garde toujours la possibilité de déplacer le symbole de sa position par défaut statistiquement la plus probable vers une autre position. Même dans ce cas, le système apprend alors des choix de l'utilisateur, la différence entre la position prédite et la position choisie entrant dans le prochain calcul de cette fonction de probabilité.

(iii) Proposition itératives

Alors que de plus en plus de symboles sont positionnés sur le canevas, une représentation du signe final le plus probable (vu cet ensemble de symboles) est proposée à l'utilisateur à côté des menus de saisie des symboles de base.

Plusieurs signes peuvent aussi être proposés à l'utilisateur, classés par probabilité de correspondance. L'utilisateur peut naviguer dans cette liste, ou choisir directement un signe fini par un raccourci.

Dès qu'un signe est choisi, le processus d'entrée est terminé pour le signe courant, et l'interface de saisie passe au signe suivant. Le signe saisi vient s'ajouter aux propositions possibles de signes finis, permettant aussi un apprentissage au fur et à mesure des entrées de l'utilisateur : un dictionnaire des possibilités est créé à la volée.

(iv) Extensions

Le résultat est une cohérence progressive des documents saisis par un même utilisateur, la standardisation permettant de supprimer les formes variantes en SignWriting d'un même signe.

Il est possible d'imaginer tirer parti de cet algorithme pour construire, à partir d'un réseau d'utilisateurs, un dictionnaire des formes standards en SignWriting.

(v) Travaux nécessaires

Plusieurs travaux supplémentaires doivent être menés pour disposer des éléments nécessaires à cet algorithme :

- un algorithme de comparaison des signes et des symboles en liste ordonnée, pour établir la fonction de proposition statistique,

- des études statistiques sur les documents SignWriting existants, pour créer à la fois un dictionnaire de probabilité des signes et des symboles,

- un arbre de probabilité des symboles, pour faire correspondre à une séquence ordonnée de ***n*** symboles les probabilités du symbole ***n+1*** et sa position initiale,

- un arbre de probabilité des signes, pour faire correspondre un signe avec un sous-ensemble incomplet de symboles dans l'ordre de saisie, puis de positionner l'ensemble des symboles sur le canevas.

3-4-9. Évaluation et critique de l'approche proposée

Le principal inconvénient est la difficulté de reconnaissance d'un signe – aussi bien en cours de saisie pour proposer les symboles le composant, qu'en fin de saisie pour proposer un signe standard si nécessaire. Une méthode logicielle robuste sera nécessaire.

De plus, cette solution nécessite un dictionnaire recommandant des formes standards pour les signes SW.

Les dictionnaires existants ne se prêtent pas tous à cette manipulation, dans la mesure où les symboles correspondant à des signes sont parfois stockés sous forme de simples images bitmap.

Enfin, la quantité de travail nécessaire pour chaque élément, avant de pouvoir mettre en œuvre l'algorithme, est très importante.

Essayons donc d'évaluer les conséquences de cet algorithme, dont la mise en œuvre sera complexe.

(i) Sur la taille des données

Aucune variation spéciale de la taille des données n'est à prévoir, dans la mesure où les codes d'une variante sont remplacés par ceux d'une autre.

Toutefois, si les formes standard choisies comportent un nombre différent de symboles que leurs variantes, et que cette différence soit significative même sur de grandes séries, la taille des données pourrait aussi varier.

(ii) Sur la rapidité de saisie

La rapidité de saisie de l'algorithme proposé n'a pas été évaluée : elle n'est pas évaluable à moins de réaliser l'intégralité des travaux nécessaires, puis de proposer aussi de nouveaux critères de mesure.

En effet, les critères de comparaison existants, comme KSPC pour les textes saisis au clavier *[2002-MacKenzie-KPC]*, ne sont pas transposables aux langues signées, et la distance de Levenshtein *[1998-Delsarte-Levenshtein]* l'est encore moins aux formalismes d'écriture bidimensionnels.

(iii) Sur les méthodes de saisie

Les approches de reconnaissances de formes ne sont pas mentionnées dans cet algorithme (ex : saisie sur tablette graphique).

Elles requièrent au minimum un algorithme de traitement de formes, et ainsi pourraient bénéficier par rapport à l'algorithme présenté d'une reconnaissance automatique du symbole en cours de saisie.

Ainsi, elles dispenseraient l'utilisateur du parcours des menus pour choisir un symbole.

Les fonctions statistiques pourraient aussi être modifiées pour mettre à jour le symbole affiché au cours de la reconnaissance de saisie, en rajoutant aux calculs de probabilités basés sur la liste ordonnée de symboles des calculs sur la liste ordonnées des traits dessinés.

De même, le positionnement automatique pourrait être complété par des mécanismes de glisser/déposer ou de pointage de la position initiale.

Tous ces problèmes complexes, liés à l'interaction homme-machine, doivent faire l'objet d'études complémentaires par des équipes spécialisées.

Ils sont hors du sujet de cette thèse.

(iv) Conclusion sur la neutralisation

Le problème de neutralisation des variations est déplacé vers la couche d'entrée. Il est toutefois persistant - aucun encodage ne peut en lui même le régler.

Si le positionnement de l'algorithme de neutralisation a été évalué, l'algorithme lui même et ses performances sur l'accélération de la saisie n'ont pas été évalués, vu la quantité de travaux supplémentaires nécessaires en préalable : le scénario d'interaction avec l'utilisateur n'est donc qu'une proposition.

Toutefois, cette proposition a l'avantage d'être cohérente avec des approches similaires, notamment pour des ordinateurs dénués de claviers mais pourvus d'un écran tactile.

Une phase préalable d'évaluation et de calculs utilisant des documents au format SWML pourrait être envisagée, pour tirer parti de l'ensemble des documents existants pour effectuer une pré-évaluation des gains de temps escomptés, avant de lancer la quantité des travaux de recherche et d'implémentation nécessaires.

Dans tous les cas, l'avantage pour les logiciels est de disposer d'une forme standard, ce qui accélère les problèmes de recherche textuelle en supprimant leur aspect « traitement d'image », et en les transformant en simples recherches de correspondances d'un élément à une liste.

Mais vu que les motivations de cette approche sont liées à la mise à disposition des logiciels d'une forme standard, n'est-il pas possible de conserver à côté de ces formes standard leurs variations, pour les raisons précédemment évoquées ?

Pourquoi ne pas sauver à la fois la forme standard et la forme variable, toutes les deux étant disponibles et utilisables à l'issu de cet algorithme ?

3-4-10. Encodage de la variante pour préservation

Cette section à fait l'objet d'une communication à la conférence IB 2005 *[2005-Aznar-Préserver]*.

(i) Introduction

Si chacun peut transcrire les signes à sa manière avec SignWriting, les avantages liés à la numérisation pourraient être limités par les possibles incompréhensions amenées par l'utilisation de variantes peu fréquentes ou exagérées.

On pourrait même concevoir que le lecteur ne puisse pas déchiffrer correctement les signes, en cas de fortes variations inter-personelles ou d'un usage de plans 3d (ZY/XZ) ou des signes SW peu fréquemment utilisés.

Toutefois, de nombreux travaux ont démontré le rôle des aires motrices du cerveau dans la reconnaissance de mouvements.

SignWriting transcrivant la dynamique du mouvement, il est très probable que la reconnaissance des signes fasse aussi appel aux aires motrices du cerveau, comme il a été déjà démontré pour la reconnaissance de l'écriture des langues dites naturelles *[2002-Knoblich-Predicting_Strokes]*.

Comprendre un signe même si ses symboles varient ne serait donc que ralenti, et non rendu impossible, puisqu'en reconstituant mentalement le mouvement responsable du signe, l'utilisateur pourrait en déduire le signe équivalent.

Toutefois, aussi bien pour l'utilisateur que pour les logiciels s'intéressant aux variantes, il faudrait envisager de préserver les données saisies pour éviter de telles mises en correspondance.

(ii) Type de préservation

Étant donné la nécessité d'offrir un forme standardisée, il ne peut être envisagé de préserver la forme variable qu'à côté de la forme standard.

L'algorithme discuté précédemment, proposant et positionnant de manière automatique les symboles dès que le signe est reconnu, doit donc être complété.

À l'issue de cet algorithme, la co-préservation de la forme standard ne peut se faire qu'en sauvegardant à la fois le signe SW tel qu'édité et le signe SW standard, mis alors en méta-donnée.

(iii) Complétion de l'algorithme

Le signe édité est affiché pour faciliter sa modification éventuelle, le signe standard étant utilisé pour toute autre opération.

Une fois l'opération de saisie terminé, il est présenté à l'utilisateur la forme standard reconnue du signe.

Une interaction de l'utilisateur permet de permuter la forme reconnue et la forme saisie si nécessaire.

(iv) Travaux nécessaires

Le format d'encodage doit supporter des méta-données, pour encoder et gérer à la fois la version réalisée et la version standardisée.

Mais comme précédemment noté dans l'état de l'art, Unicode dispose de codes pour compléter spécifiquement une séquence affichable par une séquence elle non affichable, décrivant par exemple la séquence affichable.

Ainsi, il est possible d'envisager la mise en méta-donnée directement au niveau de l'encodage, sans dépendre d'un format logiciel quelconque.

Il est toutefois à prévoir un travail d'adaptation de la couche d'entrée et d'affichage, pour supporter cette dualité entre forme variable et forme standard d'un même signe, et surtout pour pouvoir utiliser soit l'une, soit l'autre.

(v) Extension

Dans l'algorithme, la saisie est suivie d'une éventuelle interaction avec l'utilisateur, ce qui lui offre l'opportunité de saisir des méta-données complétant ainsi ce signe.

Ces dernières peuvent amener diverses précisions sur le signe, approche souhaitée *[2004-Crasbord-Sharing_data]* mais encore non normalisée.

Ces méta-données pourraient aussi être utilisées pour simplifier le problème de recherche dans un texte, en permettant de compléter la recherche sur les formes standards par une recherche sur les formes variables, pour maximiser les chances de l'utilisateur d'atteindre le signe voulu au sein d'un document.

Pour l'étude linguistique de documents SW, un autre avantage se présente: les versions et les positionnements des symboles seraient conservés au sein des documents. Un corpus SW ainsi constitué pourrait être utilisé dans la réalisation du système de prédiction à l'entrée, ou pour toute autre étude statistique sur SW.

De plus, le support des méta-données dans les documents SW pourrait être étendu afin de supporter leur annotation *[1998-Schilit-Active_Reading]*, ou dans un but expérimental *[2002-Cuxac-LSCOLIN], [2007-Boutet-Structuration_LSF]*

Enfin, comme le problème semble se poser pour toute écriture bidimensionnelle, l'approche duale serait réutilisable dans un autre formalisme ou pour l'annotation de vidéos selon plusieurs langues et formalismes.

(vi) Récapitulatif

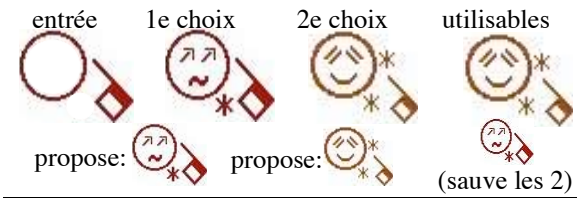


Figure 101: Illustration de l'effet de l'algorithme d'aide à la saisie

Lors de l'entrée d'un choix incomplet de symboles, le système propose un choix complet correspondant à un signe possible, en calculant un degré de possibilité selon le choix de symboles et leur proposition respective (1e choix).

Si ce signe ne correspond pas à une forme standard, elle se voit proposée (2e choix) - il s'agit là de l'algorithme d'aide à la saisie présenté dans la section précédente.

Par contre, les deux formes, la forme standard et la forme entrée, sont conservées et sauvegardées en même temps - les deux étant utilisables suivant le choix de l'utilisateur.

3-4-11. Évaluation et critique de l'approche proposée

(i) Sur la taille

Outre les codes indiquant le début et la fin des méta-données, l'encodage de la forme variable se rajoute à l'encodage de la forme standard.

Il est donc à prévoir une augmentation de la taille des données d'un facteur au moins égal à 2.

(ii) Sur la rapidité

Par contre, un surcoût temporel significatif n'est pas à craindre, le temps nécessaire pour écrire à côté de la forme standard la forme variable en méta-donnée étant négligeable devant l'algorithme de mise en correspondance d'une forme standard.

Un temps supplémentaire lors de la saisie pourrait être craint, si l'utilisateur doit valider chaque forme standard.

Toutefois, de tels systèmes utilisés dans d'autres langues, comme pour la saisie alphabétique au clavier en japonais, présentent un surcoût très modéré par la réutilisation d'éléments de l'interface : ainsi, après avoir saisi alphabétiquement en japonais, un utilisateur fait défiler les variantes possibles en kanji avec la touche espace jusqu'à ce qu'il trouve celle de son choix. Ici toutefois, une complexité supplémentaire serait de choisir, à sémantique identique, une forme différente.

Des approches statistiques permettent de réduire les délais d'interactions, en plaçant en haut de la pile la forme la plus fréquente, ou en optimisant la disposition des touches du clavier *[2005-Brewbaker-Genetic_Keyboard]* si plusieurs sont nécessaires pour les besoins d'une saisie bidimensionnelle.

(iii) Conclusion sur la préservation

L'approche ici proposée permet d'enrichir de méta-données le signe édité.

Un avantage immédiat pour l'utilisateur est de voir, s'il le désire, les signes comme il les écrit, et non comme le système les reconnaît.

Par défaut, l'utilisateur peut aussi se contenter de la forme reconnue par le système : même dans ce cas, la correction par l'utilisateur d'un signe mal reconnu est facilitée, et la saisie potentiellement accélérée par l'environnement logiciel *[1996-Horstmann-Word_predicting]*.

De plus, la possibilité d'utilisation a posteriori de ces méta-données est très intéressante pour les études linguistiques sur les formalismes d'écriture de la langue des signes *[2007-Boutet-Structuration_LSF]*.

3-4-12. Conclusion

L'approche proposée permet à la fois de neutraliser les variabilités, pour accélérer les travaux logiciels, et de les préserver, pour favoriser l'utilisation par des humains.

Des travaux très importants sont toutefois nécessaires avant de passer à une implémentation de cet algorithme.

Mais la gestion des variabilités est importante pour SignWriting, pour deux raisons :

- tout d'abord, car SignWriting est extrêmement permissif, et ne dispose d'aucune grammaire ou vérification de la possibilité anatomique de réalisation des mouvements,

- ensuite, car l'aspect bidimensionnel du formalisme d'écriture n'a pas son égal dans les autres langues écrites.

Le cas particuliers des formalismes de schémas pourrait être opposé à cette remarque *[2001-Taentzer-Graphs_Exchange_Format]*, mais il ne correspond ni à des problèmes d'encodage de caractères ni à une combinatoire à la saisie aussi élevée qu'en SW.

En conséquence, il faut passer d'une approche linguistique, où les variabilités sont traitées par des règles de composition, à une approche d'encodage de caractères, où les variabilités sont reconnues, identifiées, et laissées à disposition pour tout logiciel retraitant ultérieurement le fichier.

Toutefois, tout comme la segmentation du problème en sous-couches fonctionnelles, cette approche duale du mécanisme des variabilités se base sur le postulat d'un encodage Unicode.

Il est donc maintenant nécessaire de revenir à l'encodage des données, pour étudier plus précisément cette problématique, les conditions et les conséquences du support Unicode de SignWriting ayant bien été envisagée.

3-5. Couche d'encodage des données

Cette section a fait l'objet d'une communication à la conférence LREC 2006 *[2006-Aznar-Algorithm].*

3-5-1. Introduction

Si le postulat d'encodage Unicode précédemment réalisé est peu précis, il a toutefois permis de faire avancer la problématique en traitant notamment le problème de la segmentation, des variabilités, et des manières de les gérer.

Mais revenons sur cet encodage Unicode : nous avons vu qu'il fallait traiter deux problèmes séparés : l'encodage des symboles eux mêmes, et la manière de les organiser géométriquement pour aboutir au glyphe composé représentant le signe final.

Il est maintenant nécessaire d'envisager cet encodage sur un aspect plus pratique - notamment d'un point de vue vitesse et taille des fichiers, après avoir considéré les possibilités d'implémentation.

Intéressons-nous donc séparément à ces deux problèmes, une fois étudiées de manière exhaustive les approches possibles, en gardant à l'esprit les objectifs de l'approche Unicode:

- intégration au système d'exploitation
- minimisation de l'espace mémoire et disque occupé par rapport à l'approche XML
- minimisation du coût traitement mathématique et algorithmique

Pour ces objectifs, considérons plus précisément la manière dont est encodée un code Unicode en séquence de bits, sans considérer la méthode de transport de cette séquence de bits : comme expliqué précédemment, plusieurs méthodes (UTF-8, UTF-16...) permettent de passer du code Unicode à une chaîne de bits, mais malgré leurs différences mineures, elles sont globalement similaires.

Toutefois, dans l'objectif d'un véritable support Unicode de SignWriting, impliquant une normalisation, Unicode ne doit pas être considéré comme un simple vecteur d'une séquence de bits : l'esprit de la norme Unicode doit être respectée.

Ce critère supplémentaire sera considéré dans un deuxième temps, après avoir étudié les paramètres vitesse et taille.

3-5-2. Encodage des symboles en Unicode

Le premier problème est d'associer à chaque symbole existant un code Unicode.

Pour donner un code à chaque symbole de SignWriting, deux solutions opposées sont possibles, pour répondre préférentiellement aux objectifs antinomiques énoncés :

- minimisation de l'espace utilisé : approche séquentielle dite « mono-champ », les symboles n'étant pas rangés par groupes à signification particulière, mais en séquence sans ordre donné à l'intérieur d'un seul champ de bits, en étant dans une queue-leu-leu, tout au mieux structurée comme une suite de groupes

- minimisation du coût de traitement : approche par masque de bits dite « multi-champs », ou « bitwise » : les symboles étant groupés, chaque groupe correspondant à un paramètre donné et à un champ de bits à l'intérieur de l'entier représentant la valeur Unicode du symbole

Étudions plus précisément chacune de ces approches permettant de donner une code à chaque symbole SignWriting.

(i) Approche mono-champs

De manière évidente, la manière la plus simple de faire correspondre une code Unicode à chacun des 25 973 symboles est de procéder de manière séquentielle.

Il y a alors besoin de :

$$b_s = \ln_2(z) bits$$

avec z = nombre de caractères.

SignWriting comporte z=25 973 symboles, il faut donc utiliser 15 bits car log 2(25 973)=14.66, qui doit être arrondi à l'entier supérieur.

Un plan Unicode offre 16 bits - donc bien plus que nécessaire pour cette approche, qui en n'utilisant que moins de 50% du plan pourrait cohabiter avec des caractères déjà présents (ex : Le troisième plan, dit « Plan supplémentaire Idéographique » ou « Plan 2 », contenant principalement des caractères Chinois)

En cas d'occupation d'un plan dédié, la place restante pourrait aussi être affectée à l'organisation géométrique de ces symboles qui est traitée dans un second temps.

Afin de faciliter la démonstration, prenons à titre d'exemple un arbre de 2 niveaux, comportant trois branches (A, B, C) pour le premier niveau, comprenant chacune respectivement 4, 5 et 6 branches au second niveau, pour un total de 12 symboles. Classifions les symboles sur une représentation graphique de cet arbre.

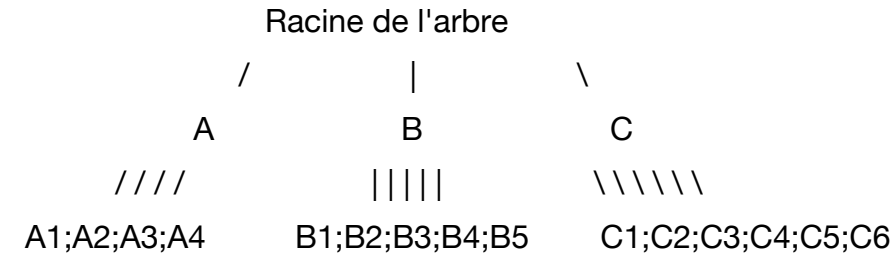


Figure 102: Représentation de l'arbre d'exemple

Pour suivre une approche mono-champs, il faut disposer les symboles en séquence. On ne peut pas créer de groupes particuliers, mais on peut essayer de structurer la séquences par suite de groupes.

A1, A2, A3, A4, B1, B2, B3, B4, B5, C1, C2, C3, C4, C5, C6

Figure 103: Représentation séquentielle de l'arbre précédent

Cette approche présente toutefois deux inconvénients majeurs :

- l'insertion de nouveaux symboles ne pourrait logiquement se faire qu'à la fin du bloc, et donc même en essayant de donner une structuration initiale à cette séquence, elle ne pourrait pas être conservée très longtemps.

Or les symboles dans SignWriting étant organisés selon une structure établie (catégorie, groupe, symbole de base, variation, remplissage, rotation), cet inconvénient serait beaucoup plus important que pour les autres langues, notamment pour le traitement logiciel de tels textes qui impliquerait une gestion cas par cas au delà de l'espace ordonné.

Ceci est particulièrement gênant pour SignWriting, qui évolue fréquemment. Un espace non attribué pourrait être laissé entre chacun des 6 paramètres de la structure, mais ce ne serait alors qu'un décalage temporel du problème, devant de toute manière se produire vu l'évolutivité de l'écriture de la langue des signes.

A1, A2, A3, A4, B1, B2, B3, B4, B5, C1, C2, C3, C4, C5, C6, A5, A6, A7

Figure 104: Représentation séquentielle de l'arbre précédent en rajoutant 3 A

L'insertion à l'intérieur de la séquence n'est pas non plus possible, puisque sinon des problèmes de compatibilité se présenteraient, comme dans le cas d'HamNoSys *[2005-Bray-Dictionnary]*.

De plus le consortium Unicode n'admet pas la réutilisation de codes précédemment affectés, dans un but de maintient de compatibilité. Cet exemple s'illustre facilement dans le cas de la monnaie européenne, qui avant de se nommer Euro s'appelait Écu : le code de l'Euro 0x20AC n'a pas pu se substituer au code de l'Écu 0x20A0, même si ce dernier n'existe plus. Il a donc été rajouté après.

- le traitement logiciel, qui même dans le cas où aucun nouveau symbole ne serait rajouté, et la séquence initiale serait alors conservée, impliquerait toutefois comme démontré ci-dessous la réalisation de divisions entières ou de modulos pour retrouver si le symbole appartient à tel groupe/catégorie/etc.

Une telle opération d'extraction de paramètre est nécessaire pour toute détermination des propriétés précises d'un symbole, par exemple pour l'afficher ou pour des opérations de rechercher/remplacer.

Bien que cette opération soit instinctivement plus rapide qu'une recherche de ces mêmes paramètres sur une pleine page de XML (dans le cas de SWML), il serait possible de l'améliorer encore plus avec une approche plus intelligente, par bits, qui outre d'être plus rapide, aurait aussi l'avantage d'être plus simple.

Étudions donc la possibilité de réaliser une telle approche par masque de bits.

(ii) Approche multi-champs

Dans l'optique d'une minimisation du coût de traitement logiciel, une approche par masque de bits permettrait de retrouver le type d'un symbole par simple décalage de bit ou masquage.

Toutefois, bien qu'elle optimise le traitement, elle augmenterait aussi la place nécessaire pour encoder les symboles.

Pour représenter cette place, prenons une hypothèse de simplification : considérons l'exemple d'un arbre où pour tout niveau ***n***, chaque nœud sur ce niveau présente le même nombre de branches, le nombre de branches pouvant varier à chaque niveau.

Selon cette simplification, on peut schématiser tous les ***n*** paramètres sous forme d'un ensemble ***E(n-1)*** de champs, vu que l'on commence à E0.

En suivant l'hypothèse simplificatrice, l'arbre utilisé à titre d'exemple, à 12 caractères peut ainsi en contenir 15 (3 sont non affectés), et se représenter par:

E0={A,B,C}

E1={1,2,3,4,5}

Dans le cas de SignWriting, chaque niveau représente un paramètre.

Selon la norme 2004, il y a 6 paramètres au total, constitués de 8 catégories, 10 groupes, 50 symboles de base, 5 variations, 6 remplissages et 16 rotations, pour un total 25 973 symboles.

On peut représenter l'arbre de SW par :

E0={1,2,3,4,5,6,7,8}

E1={1,2,3,4,5,6,7,8,9,10}

...

E5={1,...,16}

Le nombre de feuilles de cet arbre est :

$$\prod_{i=0}^{n-1}(card\,(Ei))$$

Pour encoder cet arbre selon l'approche séquentielle, il faudrait donc :

$$b_s = \ln_2(\prod_{i=0}^{n-1}(card\,(Ei)))\,bits$$

le logarithme binaire étant arrondi à l'entier supérieur.

Pour l'encoder selon l'approche multi-champs, dite bitwise, il faut par contre:

$$b_b = \sum_{i=0}^{n-1}(\ln_2(card\,(Ei)))\,bits$$

chaque logarithme binaire étant arrondi à l'entier supérieur.

Par exemple, le 6e niveau de l'arbre, correspondant au paramètre « rotation », a 16 valeurs possibles. Il prend donc log 2 (16)=4 bits.

Comparée à l'approche séquentielle, cette approche bitwise présente un coût moyen de ***n-1*** bit, ici au total ln 2 (8*10*50*5*6*16)=log 2(1.920E6)=20.87 soit 21 bits.

Toutefois, comme rappelé plus haut, les 6 paramètres de SignWriting ne respectent pas l'hypothèse de simplification : des positions de l'arbre ne sont donc pas attribuées, mais perdues

Le coût réel est alors plus grand, même s'il n'y a que 25 973 symboles et non 8*10*50*5*6*16=1,92 millions, car les pires scénarios doivent être considérés, vu que l'on encode dans un nombre de bits fixés.

La différence entre la fréquence d'un paramètre et la rareté d'un autre est responsable de ce coût, et au final, on observe un important gâchis d'espace.

On pourrait représenter graphiquement cette perte de place par un ensemble de verres posées les uns à côté des autres, et remplis différemment.

Le verre le plus rempli détermine la taille nécessaire pour tous les autres verres, et plus la somme de vide dans tous verres est importante, plus la perte est grande : l'espace qui n'est pas rempli est perdu !

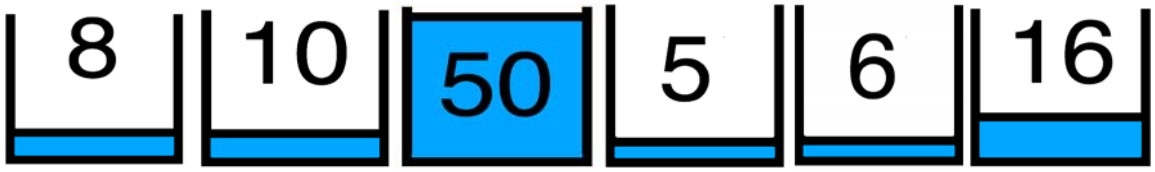


Figure 105: Représentation visuelle de l'espace gaspillé dans l'approche bitwise

(iii) Comparaison de l'espace nécessaire des approches

Pour la première approche, rappelons qu'il faut :

$$b_s = \ln_2(z)\, bits$$

Pour cette seconde approche, le coût peut être donc représenté de la manière suivante :

$$b_b = \ln_2(M_0) + \ln_2(M_1) + \ldots + \ln_2(M_n)$$

avec ***Mi*** étant pour tout les nœud d'un niveau ***i*** donné le plus grand nombre de branches filles rattaché à l'un de ces nœuds.

On remarque bien que par rapport à la solution séquentielle, la solution bitwise perd d'autant plus de place qu'il y a un gros écart entre les valeurs maximales que peuvent prendre les différents champs en fonction du champ précédent.

Mais procédons à une application numérique pour calculer le coût réel de cette seconde approche, en dehors de toute hypothèse simplificatrice sur l'arbre.

(iv) Application numérique : calcul du nombre de bits

Pour le besoin de cette application numérique, il faut dénombrer les symboles de SignWriting. Il est très simple d'analyser SymbolBank 2004 précédemment décrit, où figurent toutes les combinaisons de paramètres pour chaque symbole.

On suppose simplement que |En| == max(En), c'est à dire que le nombre d'éléments est égal à l'élément maximal. Les éléments absent (trous) sont pris en compte comme s'il étaient réservés.

Champs E*n*	**max (E*n*)**	**Nombre de bits**
E0 : Catégories	8	3
E1 : Groupes	10	4
E2 : Symboles de base	50	6
E3 : Variations	5	3
E4 : Remplissages	6	3
E5 : Rotations	16	4

Figure 106: Tableau synthétique de l'analyse de SSS-2004

On relève bien les 25 973 symboles précédemment mentionnés, subdivisés en 8 catégories, 10 groupes, 50 symboles de base, 5 variations, 6 remplissages et 16 rotations, et le calcul mathématique effectué nous confirme que 23 bits sont nécessaires pour le codage par masque de bits.

(v) Comparaison de l'espace nécessaire pour les approches Unicodes à SWML

Rappelons que la première approche, mono-champs, requiert 15 bits, soit la moitié d'un plan plan Unicode, et que la seconde approche, multi-champs, requiert 23 bits soit plus d'un plan Unicode.

Dans ces deux cas, il s'agit de la taille des tables de codage. La taille de la table codage de SWML est bien moindre, puisqu'elle se contente des 7 bits de l'ASCII.

Par contre, il faut raisonner en espace total nécessaire, qui fait intervenir non seulement la table de codage, mais aussi le nombre de codes nécessaires pour décrire un signe.

Là, l'approche SWML emploie beaucoup plus d'espace: si elle requiert beaucoup moins de bits pour son encodage, elle utilise par contre de très nombreux codes ASCII, puisqu'en moyenne un ligne entière de texte est nécessaire pour un seul symbole.

À partir de l'exemple précédent du signe « Brésil », reprenons une de ces lignes au format XML pour faire des calculs plus précis :

<symbol x="141" y="131">06-01-002-01-02-05</symbol>

Cette ligne présente 48 caractères, dont 19 sont utilisés pour indiquer le choix du symbole, et chaque caractère est encodé sur 7 bits, pour 128 possibilités ; c'est à dire qu'un symbole requiert 19 fois 7 bits, soit log 2(19x2^7)=11.25 soit 12 bits.

Pour un symbole isolé, du simple fait de la taille de la table d'encodage, même l'approche Unicode la plus économe, l'approche mono-champs, va donc nécessiter plus de place, ce qui semble contre-intuitif.

Toutefois, si l'on prend le total des symboles de SSS-2004, il faut 25 973x19=498 487 caractères de 7 bits chacun pour représenter tout SignWriting, donc il faut au total log 2(498 487*2^7)=25.93 soit 26 bits au total.

Dans ce cas, même l'approche Unicode la plus dispendieuse, l'approche multi-champs, va nécessiter moins d'espace de stockage.

Bien sur, il ne s'agit pas d'une situation réaliste, puisqu'il ne faudra pas nécessairement afficher tous les symboles contenus dans SSS-2004 pour un texte donné.

Par contre, ces exemples simples permettent de constater que dès qu'un nombre important de symboles va devoir être encodé, SWML qui est plus économe pour chaque symbole pris séparément perd rapidement son avantage, étant donné qu'il n'utilise que 10 possibilités de chaque code ASCII qui en comporte 128.

Le passage de l'ASCII vers l'Unicode va bien sûr dépendre de l'encodage Unicode sous-jacent (UTF-8, UTF-16...) mais il va influencer les approches Unicode et l'approche SWML de la même manière, et alors que SWML va nécessiter de plus en plus de place pour encoder les valeurs numériques décimales de chaque paramètre, les approches proposées vont requérir peu d'espace supplémentaire.

Les calculs pour chaque cas d'encodage ne sont donc pas nécessaires, et l'on comprend que les approches Unicode proposées ont toutes les deux un avantage au niveau espace de stockage - un avantage dont le chiffrage précis dépend de l'encodage Unicode retenu, et au maximum d'un facteur 19, si 1 code unicode remplace les 19 caractères du SWML.

Il va falloir procéder à des calculs supplémentaires pour tenir compte de ce nombre de codes - ce sera fait dans un deuxième temps, puisque un certain nombre de codes supplémentaires est nécessaire pour ajouter une grammaire de composition, qui n'a pas encore été étudiée.

Mais déjà, vu que la plupart des langues écrites ont été écrites dans un seul plan Unicode, il semble difficile d'envisager d'utiliser même un seul plan Unicode (selon l'approche mono-champs) pour n'y encoder qu'un seul formalisme d'écriture, si l'on ne prends pas le temps de documenter plus précisément les bénéfices liés à cette approche.

Or une opération de recherche de similarité de texte, ou de recherche textuelle, étant donné les variations inter et intra-personelles, fera appel à de nombreuses extractions de paramètres, et ce pour chaque symbole - il pourrait y avoir des bénéfices temporels.

Étudions donc maintenant les temps de calculs respectifs à chaque approche pour des opérations de base, de type extraction de paramètres, qui devront être réalisées soit par le moteur Unicode, soit par l'algorithme de neutralisation des variabilités, soit par les fonctions de rechercher/remplacer.

(vi) Temps de calcul de l'approche simple

Prenons l'exemple de l'extraction du paramètre variation.

Nous nous plaçons alors dans un cas simple, l'extraction de la variation (E3), puisqu'il y a toujours le même nombre de paramètres suivants, ici remplissage (E4) et rotation (E5).

Dans un autre cas, le nombre de paramètres suivants pourra être variable étant donné que toutes les possibilités ne sont pas exploitées en SignWriting, comme il a été rappelé lorsqu'il a été noté que l'hypothèse simplificatrice de l'arbre n'était pas respectée.

En suivant l'approche mono-champs, l'extraction du paramètre variation d'un code x s'exprime par la division entre le code x et le produit du cardinal des deux autres ensembles modulo le cardinal des variations

Si nous implémentons cette expression en C par le programme suivant, on se rend compte qu'il est nécessaire d'effectuer deux divisions entières.

```
/* approche mono-champ */
#define NUM_REMPL 6
#define NUM_ROT 16
#define NUM_VAR 5
int extraitvariation(int c) {
  return((c / (NUM_REMPL*NUM_ROT)) % NUM_VAR);
}
```

Figure 107: Utilisation du modulo dans l'approche mono-champs

Nous rappelons qu'en langage machine une opération de modulo est réalisée par une division, et qu'ayant moins de 65536 symboles, nous n'utilisons qu'un espace 16 bits.

Ici, nous avons donc comme opérations significatives deux divisions entières 16 bits, et nous pouvons estimer le temps nécessaire aux deux opérations à l'aide de données de la littérature *[1992-Hummel-Guide_programmeur]*, à titre d'exemple sur un microprocesseur Intel 486.

Une division 16 bits: 24 cycles d'horloge

Donc, deux divisions: 48 cycles d'horloges.

Les valeurs pour des processeurs différents devraient être comparables.

Étudions donc le temps nécessaire pour la deuxième approche.

(vii) Temps de calcul pour l'approche bitwise

Dans ce second cas, l'implémentation se ferait en C de la manière suivante:

```
/* approche multi-champ */
/* BITS_N=log2(NUM_N) arrondi à l'entier supérieur */

#define BITS_ROT 4
#define BITS_REMPL 3
#define BITS_VAR 3

int extraitvariation(int c) {
  /* On veut masquer "variation", de 3 bits : 111 en binaire == 7 en décimal */
  return((c >> (BITS_ROT+BITS_REMPL)) & 7);
}
```

Figure 108: Utilisation du masque de bits dans l'approche multi-champs

Ici, nous avons comme opérations significatives un décalage de bits, et un « et » bit-à-bit, aussi dénommé « masquage ».

De la même manière, il est possible d'estimer les coûts de ces opérations dans la littérature, sur un processeur identique (i486) pour pouvoir comparer aux calculs précédents.

Un décalage sur 32 bits: 4 cycles d'horloge

Un ET bit-à-bit sur 32 bits: 2 cycles d'horloge

Donc, pour toute l'opération: 6 cycles d'horloge

Nous pouvons ainsi nous attendre à un avantage de vitesse de l'ordre d'un facteur 48/6=8 dans le pire des cas, à l'avantage donc de l'approche bitwise, même dans le cas le moins favorable à cette méthode, puisque l'hypothèse simplificatrice de l'arbre s'applique vu que le nombre de paramètres suivants est toujours identique.

Toutefois, la méthode de comparaison théorique des temps de calculs ne présente pas la certitude et la qualité d'une comparaison pratique avec réalisation de la mesure sur des microprocesseurs plus modernes que le i486.

Testons donc chaque approche sur un microprocesseur contemporain.

(viii) Application numérique : comparaison des approches

Réalisons pour celà deux logiciels, l'un suivant la première approche, l'autre suivant la seconde, pour comparer leurs performances.

```
/* approche mono-champ */
#define NUM_REMPL 6
#define NUM_ROT 16
#define NUM_VAR 5
inline int extraitvariation(int c) {
  return((c / (NUM_REMPL*NUM_ROT)) % NUM_VAR);
}


int main(void) {
  int x;
  for (x = 0; x < 50000000; x++) {
    extraitvariation(19758);
  }
}  printf("Somme bits: %u\n", somme);
}
```

Figure 109: Logiciel de test pour l'approche mono-champs

```
/* approche multi-champ */
#define BITS_ROT 4 /* log2(NUM_ROT) arrondi à l'entier supérieur */
#define BITS_REMPL 3 /* idem */
#define BITS_VAR 3 /* idem */

inline int extraitvariation(int c) {
  /* On veut masquer "variation", de 3 bits : 111 en binaire == 7 en décimal
*/
  return((c >> (BITS_ROT+BITS_REMPL)) & 7);
}

int main(void) {
  int x;
  for (x = 0; x < 50000000; x++) {
    extraitvariation(19758);
  }
}
```

Figure 110: Logiciel de test de l'approche multi-champs

L'expérience est réalisée sur un processeur AMD Athlon XP 2400, sur 50 millions d'opérations pour réduire les variations observables sur de trop petites séries. Le temps nécessaire à chaque approche est mesuré à l'aide du logiciel *time*.

L'approche mono-champs prend 1.744 secondes, contre 0.292 secondes pour l'approche multi-champs.

Sur un processeur moderne, dans le cas le moins avantageux pour l'approche bitwise, cette dernière bénéficie quand même d'une exécution environ 6 fois plus rapide.

Sur un assistant personnel donc doté d'un microprocesseur moins puissant et donc moins optimisé pour les opérations de modulo, ou sur un processeur moins rapide, les données réelles devraient se rapprocher de l'estimation théorique d'un gain de temps de facteur 8.

De plus, une utilisation réelle fera appel à de multiples extractions de paramètres sur de multiples symboles, constituant les signes d'un texte donné.

L'avantage de vitesse est donc très significatif et en faveur de l'approche par groupes multi-champs, qui se traduira même dans le pire des cas par des délais 6 fois moins longs lors d'opérations de comparaison ou de recherche de texte pour l'utilisateur final.

À titre d'exemple, complétons ces mesures par une comparaison avec SWML, et réalisons un logiciel extrayant la même information du format SWML, puisqu'il s'agit de l'encodage de référence actuel.

(ix) Comparaison du temps des approches Unicode à SWML

Une estimation du temps nécessaire pour l'approche SWML, comme précédemment réalisée pour les approches Unicode, n'est pas possible, étant donné que son implémentation variera selon le cas d'application.

Faisons donc directement une application pratique du cas précédemment énoncé, appliqué à une recherche sur la source XML du document SWML : y extraire la variation d'un symbole est l'opération équivalente au cas précédent.

Une limitation est toutefois à noter: il pourrait n'y avoir qu'une lecture initiale du fichier SWML, avec copie des valeurs entières des paramètres en mémoire et réalisation des opérations dessus. Une telle optimisation pourrait néanmoins être réalisée de la même manière avec un encodage Unicode : elle n'apporte donc pas d'avantage à l'une ou l'autre des méthodes, et doit donc être négligée.

Nous nous intéressons donc ici au cas où le fichier est lu après ouverture, ou lorsqu'un programme recherche une chaîne dans un ou plusieurs fichiers.

La requête est réalisée de la manière la plus rapide possible, en faisant appel à des *regexps*, écrites selon la structure du format XML employé : balises « symbol » au début et à la fin, séparateur « - » entre les valeurs numériques des paramètres, parenthèses autour du paramètre que l'on cherche à récupérer - ici la variation, E3.

Pour lire ce paramètre, on déclare donc, compile puis exécute cette regexp:

```
regcomp(&preg,"<symbol[^>]*>[0-9]+-[0-9]+-[0-9]+-([0-9]+)-[0-9]+-[0-9]+</symbol>", REG_EXTENDED);
regexec (&preg, SWML, 2, tab, 0);
```

Le logiciel de test est alors réécrit :

```
/* approche précédente pour du SWML /*

#include <sys/types.h>
#include <regex.h>
#include <stdio.h>
#include <stdlib.h>

#define SWML "<symbol x=\"132\" y=\"87\">03-02-002-04-01-02</symbol>"

int
main (void)
{
  int x;
  regex_t preg;
  regmatch_t tab[2];
  regmatch_t *m;

  if (regcomp
      (&preg,
       "<symbol[^>]*>[0-9]+-[0-9]+-[0-9]+-([0-9]+)-[0-9]+-[0-9]+</symbol>",
       REG_EXTENDED) != 0)
    {
      printf ("regexp compile error\n");
      return (-1);
    }

  m = tab + 1;

  for (x = 0; x < 50000; x++)
    {

      regexec (&preg, SWML, 2, tab, 0);
    }
  regfree (&preg);
  return(0);
}
```

Figure 111: Logiciel de test de l'approche SWML

Les conditions de ce tests sont très favorables à cette méthode travaillant sur du SWML, puisque la requête de regexp est précompilée en dehors de la boucle for.

En pratique, le coût de l'opération de compilation de la regexp devra être rajouté à chaque itération, chaque extraction de paramètre différent faisant appel à une regexp différente.

Mais plaçons nous volontairement dans ces conditions défavorables pour pratiquer l'évaluation du temps d'exécution de cette requête de manière similaire aux approches précédentes. L'opération n'est par contre réalisée que 50 000 fois en raison de l'excès de lenteur de la méthode SWML, même sur un processeur AMD Athlon XP 2400.

	Approche simple	Approche bitwise	Approche SWML (1E-3 moins d'itérations)
Temps	1 744 ms	292 ms	1 788 (x 1000) ms

Figure 112: Synthèse des mesure de temps suivant les approches

La différence est frappante : il faut 1 000x plus de temps à l'approche SWML (1788 ms x 1 000) qu'à l'approche mono-champs (1 744 ms).

Rappelons toutefois que les approches Unicode ont été calculées dans le cas le plus désavantageux pour elles, alors que l'approche SWML a été placée dans le cas le plus avantageux.

Nous nous rendons ainsi compte que la meilleure méthode pour l'approche SWML est environ 1 000 fois plus lente que la plus lente des approches proposées (champ de bits unique), et est 6 000 fois plus lente que l'approche favorisée (champs de bits multiples), et ce même dans la configuration de cas la plus favorable pour SWML.

(x) Conclusion

Nous pouvons donc recommander l'approche par champs de bits multiples, qui se traduira lors de l'implémentation par une rapidité bénéficiable à l'utilisateur, au minimum d'un facteur 6 000 lors des opérations de recherche de texte.

L'attribution pratique de codes Unicode suivant cette approche nécessitera plus de place dans la table d'encodage que l'approche SWML qui se contente des caractères ASCII.

Mais cet inconvénient ne se manifeste que si l'on considère un nombre réduit de symboles isolés : dès que plusieurs symboles sont concernés, les approches Unicode permettront en plus des gains de place, comme démontré plus haut, vu le grand nombre de codes ASCII nécessaires à SWML, qui n'utilise que 10 possibilités sur chacune des 128 offertes par les 7 bits.

(xi) Approche intermédiaire : modifieurs

Beaucoup d'espace Unicode va être nécessaire pour suivre ces approches, à moins d'avoir recours à des modifieurs de manière similaire aux accents.

Un tel cas n'a pas été considéré précédemment, car il s'apparente à une simplification du problème, et que cette thèse s'intéresse aux difficultés que peut rencontrer le support informatique d'une forme écrite de la langue des signes.

Étudions toutefois les conséquences d'une telle simplification, qui sera toujours ultérieurement possible, du moins pour les paramètres se comportant de manière similaire aux accents, et ayant donc un intérêt modéré (ex : variations).

(xii) Utilisation d'un modifieur, donc de deux codes

Dans un tel cas, si un modifieur est utilisé, 2 codes seront nécessaires pour encoder un symbole : son propre code et le code de son modifieur.

Ce serait de toute manière le cas avec l'approche multi-champs, car elle nécessite sinon 8 fois plus d'espace que le total d'espace actuellement disponible dans Unicode - ce n'est donc pas juste une simplification du problème.

Mesurons donc l'espace nécessaire pour la table d'encodage des paramètres restants : dans le cas où un modifieur sert à représenter un paramètre, l'espace nécessaire est la somme des nombres de bits nécessaires pour chaque paramètre.

En considérant dans l'exemple choisi le paramètre variations (de cardinal 5, occupant 3 bits), il faut donc 2^17 codes au lieu de 2^23.

En conséquence, supprimer ***n*** paramètres rajoute un coût de stockage approximatif de ***n+1*** codes Unicode.

Cette approximation est due au cas particulier des « petits » paramètres, qui pourraient à plusieurs tenir dans un seul code comme modifieurs cumulatifs : par exemple dans le cas des paramètres ayant comme conséquence de modifier l'aspect 2d (vu la configuration 3d de l'objet considéré, comme la chiralité pour la main), on pourrait remplacer les modifieurs « rotation à gauche » et « face palmaire exposée » par le modifieur unique « rotation à gauche, face palmaire exposée ».

Rotation et variations qui requièrent respectivement 6 et 4 bits pourraient être un seul modifieur n'occupant que 10 bits, et réutilisable pour d'autres formalismes d'écriture. La partie principale de SignWriting n'occuperait plus alors que 13 bits, même suivant une approche bitwise, c'est à dire qu'elle tiendrait dans un seul plan.

De tels choix nécessiteront de faire appel à des linguistes.

Mais revenons au problème actuel : attribuer des codes Unicodes aux symboles, si leur positionnement n'est pas considéré, ne résout que la moitié du problème.

Il a d'ailleurs été précédemment signalé que les calculs sur l'espace total nécessaire devraient être complétés par le nombre de codes nécessaires à une grammaire d'articulation des symboles.

Étudions donc la seconde partie de ce problème : l'encodage des informations spatiales.

3-5-3. Encodage des informations spatiales

(i) Approches actuelles

SWML actuellement n'enregistre pas l'ordre de saisie des signes, et encode la position des signes en tant qu'attributs numériques de positionnement selon un repère orthonormal de taille 128x128 *[2001-DaRocha-SWML]*.

Dans Unicode n'existent que des méthode de composition 1d plus ou moins évoluées, que ce soit par ligature, substitution, variance contextuelle, comme vu dans l'état de l'art : Unicode reste donc monodimensionnel. Il n'existe donc aucune solution générique d'encodage des positions 2d des symboles composant un signe

(ii) Intérêt pour les autres formalismes d'écriture

Pourtant, il semble que de tels mécanismes de positionnement soient nécessaires pour supporter certaines écritures comme les hiéroglyphes, qui font partie des 8% de formalismes d'écriture non encore encodés et pour lesquels des mécanismes de rendu novateurs vont être nécessaire *[2002-Everson-Unicode]*, les dernières propositions *[2007-Everson-Hieroglyphs]* faisant toujours abstraction des problèmes de positionnement *[1999-Everson-Hieroglyphs][2005-Cook-Hieroglyphs].*

Une telle complétion d'Unicode semble très importante, puisque dans le cas des écritures indiennes, la généralisation de la localisation n'a été possible que lorsqu'a été offert un système Unicode non application-spécifique, après 20 ans de recherches *[1999-Mudur-Shaping_Indian]* !

(iii) Intérêt de l'encodage pour les mécanismes d'entrée

Surtout, l'intérêt d'un système adapté d'encodage de SignWriting est de pouvoir supporter les reconnaissances partielles ou incomplètes par un logiciel de traitement d'image, pour pouvoir au moins indiquer la position du segment corporel non reconnu, la sauvegarder, et laisser alors à d'autres outils (ex : analyse contextuelle) la possibilité de proposer des correspondances. Une reconnaissance incomplète de certains critères de groupement dans le domaine 3d+t [2003-Aznar-Émergence] pourrait de la même manière être gérée.

Étudions donc toutes les solutions potentielles pour encoder les informations de positionnement spatial des signes, en prenant en compte leurs avantages, leurs inconvénients et leur complexité.

(iv) Le problème de positionnement spatial

Le problème de positionnement spatial des symboles composant les signes se subdivise en deux sous-problèmes :

- positionnement individuel des symboles dans un espace 2d
- description de leur relation (ex : contact, fusion, ...)

Une syntaxe pour articuler ces deux éléments est aussi nécessaire, et le tout devra enfin être encodé dans une représentation binaire intégrable à Unicode, mais si le premier sous-problème est évident, le deuxième l'est moins.

Il est pourtant proposé d'encoder la relation établie, ne serait-ce que pour réduire le nombre de combinaisons de relations nécessaire pour obtenir un glyphe résultant, particulièrement dans SignWriting.

(v) Intérêt de la description de la relation des symboles

Le postulat d'encodage Unicode précédemment donné considère en effet les contacts comme des symboles identiques aux autres : ils sont donc positionnés spatialement comme tout autre symbole, bien qu'ils correspondent à l'établissement d'une relation entre deux symboles.

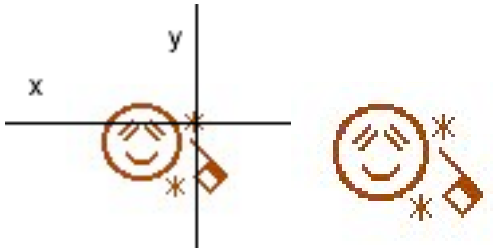


Figure 113: Positionnement individuel d'un symbole (ici contact) dans l'espace 2d

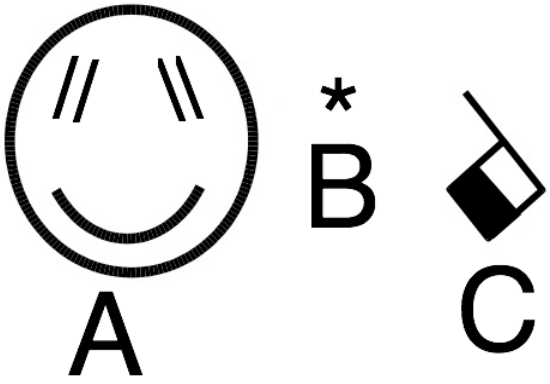
Décrire des relations permet de ne plus être limité à une échelle de discrétisation des coordonnées 2d pour séparer les cas de contact des cas de non-contact : ainsi lors de zooms rapprochés ou au contraire éloignés, le contact ou l'absence de contact pourrait être maintenue, peu importe l'échelle utilisée.

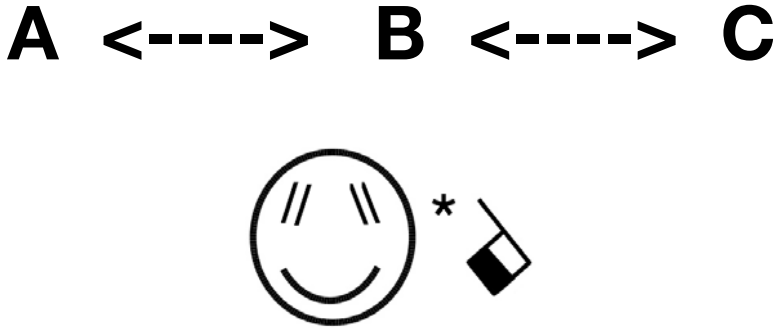
Dans le cas particulier des polices de petite taille, il serait alors possible de tirer parti du groupement perceptuel pour augmenter la lisibilité du rendu, sans conserver constants le rapport d'espacement ou celui entre taille de chaque signe.

Prenons pour illustrer ceci un exemple basé sur le signe sourd, en cherchant d'abord à éviter le contact entre le symbole de la main et le symbole du visage lors de réductions de taille, puis ensuite à le conserver lors d'agrandissements.

Figure 114: Sourd, sans contact, en zoom

Il est possible de décrire dans ce signe 3 groupes - nommons les A, B, C - dont la variation de police peut nécessiter de faire évoluer de manière indépendante leurs tailles et leurs éloignements respectifs.





A <-> B <-> C

A <> B <> C

Figure 115: Maintien de l'absence de contact B(A,C) en modifiant l'éloignement

Inversement, il pourrait être nécessaire d'avoir un contact qui se conserve.

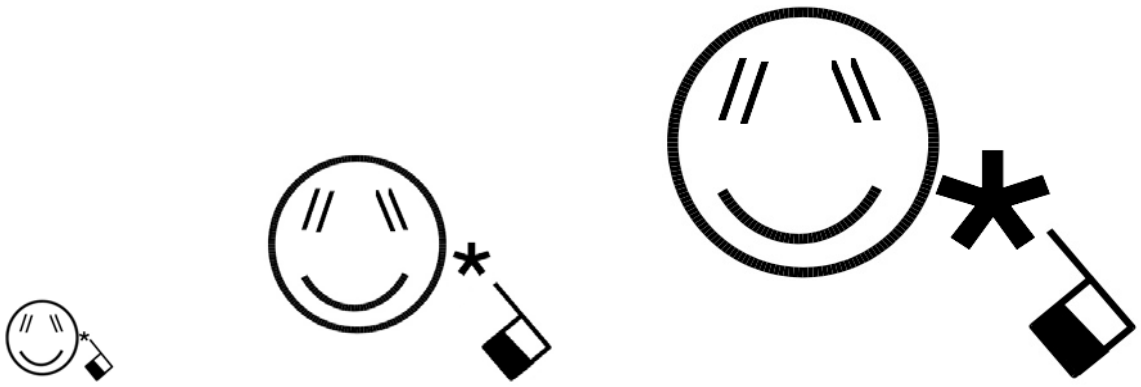


Figure 116: Maintien du contact B(A,C) en modifiant la taille d'un symbole

L'exemple proposé est certes discutable, car l'étoile signifie un contact en SW (et non une absence de contact), mais l'illustration visuelle sert surtout à établir la différence qu'il pourrait falloir faire lors de l'étape de rendu dans un cas hypothétique. Par ailleurs SW ne tire pour l'instant pas partie de telles propriétés, mais un nouveau formalisme graphique, ou même un meilleur rendu de SW, pourrait bénéficier d'une telle description de la relation graphique voire sémantique devant exister entre des symboles : un véritable grammaire de relation pourrait alors émerger.

De plus, la norme Unicode demande la description des méthodes de « composition » entre codes Unicodes - dans le cas particulier des relations spatiales, ce serait une manière indirecte de satisfaire ces contraintes, mais il serait beaucoup plus linguistique et correct, dans l'esprit de la norme Unicode, de définir de telles relations que de recourir au positionnement 2d(x,y).

Deux autres avantages indirects résultent de la description des relations :

- tout d'abord, les règles de composition dans le moteur de rendu ainsi que les règles d'encodage se trouvent simplifiées.

- de plus, les symboles indiquant des mouvements pourront être remplacés par un opérateur visuel de relation, qui pourrait alors être utilisé par l'interface en cas de manipulation d'un des symboles afin de conserver la cohérence de la relation tout en minimisant le nombre d'interactions.

Pour revenir à l'exemple précédent B(A,C), le déplacement du symbole main C reliée par l'opérateur de relation « contact » entraînerait le déplacement du point visuel de contact B le long du contour du visage si la relation B(A,C) exprimait que le contact doive se maintenir (et inversement l'en éloignerait le plus possible si la relation exprimait un absence de contact devant être conservée).

Dans le cas de l'acquisition par traitement d'image, un groupage supplémentaire pourrait aussi être rajouté pour rassembler les symboles issus de la détection d'un même mouvement, pour en faciliter la répartition par signe en cas de problèmes de segmentation, et en synthèse d'image pour définir les relations utiles aux avatars signants.

(vi) Positionnement individuel des symboles

Examinons donc maintenant les solutions potentielles pour tout d'abord positionner les symboles sur un plan.

On peut en décrire trois :

- coordonnées relatives
- coordonnées selon un repère orthonormal ou paramétrique
- coordonnées décrites par une fonction mathématique : groupement perceptuel, reconnaissance de formes, etc.

Par exemple, dans ce dernier cas, les symboles pour le flot de la mer peuvent se positionner sur une fonction mathématique sinusoïdale. Étudions pour chacune de ces possibilités les avantages et les inconvénients

(vii) Positionnement par coordonnées relatives

Avantages : Cette solution n'impose pas de limite à la taille du canevas, elle garde le point de départ et l'ordre de la séquence entrée par l'utilisateur, elle économise le nombre de bits nécessaires pour encoder un déplacement relatif (par rapport au nombre de bits nécessaires pour couvrir la totalité de l'espace des coordonnées)

Inconvénients : Elle fait appel à un algorithme complexe qui nécessite une reconstruction par étape avec prise en compte du point précédent pour reconstruire le glyphe, mais qui ne permet pas forcément d'atteindre toute les coordonnées si le nombre de bits utilisé pour le déplacement est limité.

(viii) Positionnement par coordonnées absolues

Avantages : Elle correspond à l'algorithme le plus simple, c'est approche actuellement retenue par SWML et proposée dans le postulat d'encodage Unicode, pouvant éventuellement s'ordonner selon la séquence saisie par l'utilisateur.

Inconvénients : La taille du canevas influe sur le nombre de bits nécessaire pour l'encoder, et elle ne prend pas en compte les positions relatives des objets.

(ix) Positionnement par une fonction

Avantage : L'algorithme est aussi complexe que la fonction considérée, la taille du canevas n'est pas limitée, et les symboles peuvent s'ordonner selon la séquence réalisée. Cette solution permet aussi d'économiser des bits si la fonction ne prend pas beaucoup de paramètres.

Inconvénients: Cette approche est complexe, puisqu'on doit trouver une fonction mathématique qui décrit la forme considérée.

(x) Discussion sur la méthode de positionnement

De manière évidente, tout ensemble de signes peut être positionné spatialement par n'importe quelle méthode : il ne s'agit que du positionnement d'éléments selon des coordonnées. De même, toutes les méthodes nécessitent un centre géométrique pour positionner un symbole - il est possible de choisir l'angle supérieur gauche (comme en True Type) ou le barycentre.

Si ce dernier semble plus logique, dans la mesure où il sera au mieux situé pour chaque symbole, il n'est pas mathématiquement le plus simple car il devra être calculé. De plus, il peut gêner l'application des algorithmes des polices, utilisant des description vectorielles par rapport au point de départ supérieur gauche.

Une méthode simple de positionnement selon un repère polaire ou orthonormal n'a qu'un inconvénient : elle lie la taille du canevas à la taille nécessaire pour le représenter, même dans le cas d'une méthode dichotomique, qui limite juste cette complexité à la finesse de représentation choisie. La méthode de positionnement selon un repère polaire, initialement choisie, semble limitée au sous problème des symboles comportant une tête, et ne permettre une simplification que par discrétisation suivant des cercles concentriques, c'est à dire avec les inconvénients des systèmes à coordonnées absolues sans avoir l'avantage de la simplicité, en se limitant à un « symbole dominant » dans le signe, qui en gouvernerait la syntaxe.

La méthode de positionnement relative, qui semblerait intéressante pour limiter la complexité de l'encodage, nécessite en fait une reconstruction complexe : son intérêt est limité de manière évidente au cas où plusieurs symboles sont en relation.

De plus, aucune méthode n'implique le stockage obligatoire de l'ordre de symboles saisis, qui est juste optionnel vu la possibilité de complexification de la séquence en cas d'allers et retours multiples.

(xi) Choix d'une méthode

Quelle que soit la méthode choisie, le coût temporel de l'encodage sera négligeable au final devant le coût temporel du passage d'une méthode à l'autre si le système devait passer d'une méthode à l'autre, vu la nécessité de reconstruire alors aussi une forme intermédiaire pour ce passage.

Toutefois, le système de positionnement risque d'évoluer avec le temps, ne serait-ce que pour suivre les évolutions de SignWriting ou des algorithmes, particulièrement pour la représentation mathématique. De plus, il n'est pas encore connu la disposition des symboles les plus fréquents à l'intérieur des signes, qui pourraient être encodées mathématiquement de manière à optimiser les formes les plus fréquentes, comme dans le cas de WMLc qui encode les balises HTML les plus fréquentes sur le plus petit nombre de bits *[1999-Martin-Wap_XML]*.

Vu l'absence d'avantages décisifs d'une méthode par rapport à l'autre, il est donc proposé de recourir de préférence au positionnement sur un repère de taille limitée, pour tout simplement simplifier l'algorithme d'encodage et les coûts nécessaires en cas de passage d'une méthode à l'autre. La méthode retenue est donc la plus simple : positionnement par coordonnées cartésienne avec une taille de canevas suffisante. On réserve toutefois la possibilité de supporter ultérieurement les autres méthodes, dans des problèmes spécifiques comme le tracé de trajectoire dans les images de sources vidéo (3d+t) et l'implémentation d'opérateurs de relations, si une telle possibilité d'évolution se faisait.

(xii) Évolutions possibles

Les avantages limités du positionnement relatif et du positionnement par fonctions entraînent donc que l'on les restreigne à une simple possibilité supplémentaire offerte pour les opérateurs de relation, au risque de compliquer la grammaire proposée.

Ces options doivent toutefois proposées car elles pourraient s'avérer utiles dans leurs contextes respectifs, suivant les évolutions du formalisme d'écriture - par exemple pour le support de l'icônicité et de l'établissement de relations.

Dans un tel cas, le choix du contexte se fera en fonction de l'existence ou non d'une relation entre symboles, tel que spécifiée par l'utilisateur. Cela signifie que les méthodes de positionnement relative et par fonction ne seront employées que lorsque les interfaces le permettront.

Proposons donc une méthode de représentation unifiée pour illustrer le problème de positionnement.

(xiii) Proposition de réalisation pratique

Nous proposons une grammaire faisant intervenir 3 types de codes, pouvant servir à définir les relations entre symboles pour produire le glyphe résultant, comme demandé par la norme Unicode:

- Le symbole (représenté tel que décrit dans la première partie)
- L'opérateur (qu'il soit de relation/positionnement)
- Des paramètres de ces opérateurs (ex : coordonnées)

L'ordre de cette séquence n'a pas d'importance vu qu'il s'agit d'une grammaire arbitraire, c'est pourquoi nous choisirons arbitrairement une syntaxe :

- Le nom-terminal "signe" représente ensemble de symboles positionnés.
- Un opérateur spécial "DÉLIMITEUR" est utilisé pour signaler la fin du signe.
- Par convention, les terminaux seront représentés en majuscules.

(xiv) Grammaire

signe -> signepartiel DÉLIMITEUR

signepartiel -> signelpartiel élément | élément

élément -> SYMBOLE | OPÉRATEUR

OPÉRATEUR -> DÉLIM

| <nombre> <nombre>

| CONTACT_RELATION

| PROXIMITÉ_RELATION <nombre> <nombre>

SYMBOLE -> {les 25 973 symboles de SSS-2004}

Les symboles possibles sont ceux supportés par SignWriting et recensés par la norme SSS-2004. Dans cet exemple, seuls deux opérateurs sont proposés - d'autres, qu'il faudrait bien sur définir, sont toutefois possibles.

(xv) Illustration

Considérons l'exemple du signe « sourd », faisant appel à un visage, un index tendu, et un contact entre l'index tendu et le visage.

Pour le représenter dans cette grammaire, deux possibilités existent selon que le contact est considéré comme une opérateur de mise en relation entre deux symboles, ou comme un symbole parmi d'autres :

VISAGE 10 20 INDEX 10 25 CONTACT_SYMBOLE 10 22 DÉLIMITEUR

VISAGE 10 20 INDEX 10 25 CONTACT_RELATION DÉLIMITEUR

Dans le premier cas, le contact est considéré comme un symbole : ses coordonnées sont donc nécessaires.

Dans le second cas, le contact est considéré comme un opérateur, qui s'applique donc aux deux éléments précédents et dont les coordonnées sont déterminées implicitement par ces derniers.

On voit donc que cette représentation permet d'exprimer de manière unifiée les différentes possibilités de positionnement ainsi que les relations entre symboles.

Elle se rapproche de la notation polonaise inversée, qui permet de simplifier grandement l'analyse syntaxique par les logiciels de rendu tournant sur des ordinateurs peu puissants en s'adaptant bien à un fonctionnement par pile.

(xvi) Évolution des opérateurs : opérateurs sémantiques

Deux opérateurs ont été donnés à titre d'exemple : CONTACT_RELATION et PROXIMITÉ_RELATION, signifiant respectivement l'établissement d'un contact entre deux symboles, et un déplacement relatif pour positionner un autre symbole.

Il est possible d'envisager de décrire des symboles sans opérateur de relation, les opérateurs n'étant alors tout simplement pas définis mais déduits de la liste des symboles.

À l'opposé, tous les symboles correspondant à des relations pourraient être définis comme des opérateurs.

Une liste des relations supportées par SignWriting est par ailleurs déjà définie *[2004-Sutton-SSS]*, mais d'autres opérateurs peuvent évidemment être définis pour toute évolution future de SSS : citons par exemple rapprochement, frottement...

Toutefois, comme discuté dans les justifications de cette approche, cette liste serait complétable par des opérateurs plus utile au traitement d'image, voire suivant certains schémas proposés pour la représentation des gestes sur l'espace de signation *[2004-Lenseigne-Gesture_representation]*:

- fusion
- trajectoire
- cotemporalité

Une étude détaillée des opérateurs et des syntaxes avec leurs avantages et leur inconvénients respectifs nécessiterait un travail avec des linguistes, pour définir des nouveaux opérateurs sémantiques (ex : action, propriété ...) tirant partie de la grammaire de la LSF.

Par exemple, le signe « lundi il y a deux semaines » se réalise par une double itération du « l », index et pouce tendu, sur une ligne imaginaire allant de l'arrière de l'épaule à l'avant, dans la direction de l'avant vers l'arrière pour signifier qu'il s'agit du passé.

Pour encoder « le lundi il y a deux semaines », on peut donc imaginer un nouvel opérateur sémantique « axe des temps » prenant comme paramètre le nombre d'itération du symbole:

SYMBOLE_LUNDI AXE_TEMPS -2

Toutefois, de tels opérateurs sémantiques pourraient être reliés à une interprétation de la langue des signes, et leur représentation graphique sous forme de glyphes SignWriting risquerait alors de varier selon la langue des signes considérée.

Dans un tel cas, il faudrait plutôt envisager des opérateurs non-sémantiques, reliés à des propriétés spatiales récurrentes dans les langues des signes - par exemple dans le cas considéré, l'axe « vers l'avant de l'épaule ».

Encore une fois, de tels travaux sont à réaliser par des linguistes.

(xvii) Critique

Unicode n'a pas pour rôle de se substituer à un traitement de texte.

Mais le but de cet encodage des informations spatiales n'est pas de reproduire un texte complet, juste un suite de signes composés de symboles.

Par ailleurs, comme précédemment relevé dans l'état de l'art, Unicode prévoit des codes pour séparer les lignes et les paragraphes dans le cas des autres langues écrites.

Dans le cas de SW, il n'y a pas de « syntaxe » normée, c'est à dire pas encore de limitation des combinaisons possibles. Ce n'est pas forcément un inconvénient, car ceci rend le système plus générique et plus adapté aux formalismes et aux problèmes futurs, en ne le rendant pas dépendant d'un formalisme particulier.

La restriction de cas pourra être traitée de manière simple immédiatement au niveau du logiciel de saisie pour les erreurs les plus évidentes (ex : positionnement de 3 mains se déplaçant de manière simultanée) et plus finement dans le futur, pour détecter les erreurs de saisie.

Ceci nécessite une étude de la grammaire d'encodage par des linguistes, pour proposer une liste finie d'opérateurs ainsi que des restrictions au niveau des possibilités. Ils devront aussi envisager la possibilité d'opérateurs syntaxiques ou sémantiques.

La méthode de description des positions géométriques est donc à considérer comme une opportunité d'encodage en attendant la définition d'une telle grammaire. Décrire la position géométrique correspond par ailleurs beaucoup plus aux attentes d'un système de traitement d'image (qui pourrait être utilisé en entrée) qu'une grammaire.

Il convient, maintenant qu'un mécanisme de positionnement a été sélectionné, d'évaluer l'espace qui lui est nécessaire.

(xviii) Évaluation de l'espace nécessaire

Comme base de calcul, définissons un taille de canevas de positionnement des symboles pour constituer un signe identique à celle proposée par SWML: 128x128.

Positionner un élément sur ce canevas nécessite alors 14 bits, car 128x128=16 384=2^14 ; mais vu la taille d'un plan Unicode il peut être envisagé d'utiliser un plan entier pour disposer d'un canevas √2^16=256 pixels de côté si un seul code représente les deux coordonnées.

(xix) Comparaison à l'approche SWML

Rappelons que pour l'approche SWML, 48 caractères sont nécessaires pour un symbole, dont 19 pour le choix du symbole.

Il reste donc 29 caractères encodés sur 7 bits pour définir la position, soit un total de log 2(29x2^7)=11.86 c'est à dire 12 bits pour définir la position géométrique.

Sur un seul symbole, l'approche Unicode n'a donc pas l'avantage.

Mais en prenant l'exemple d'un signe comportant 5 symboles, il faut log(5x29x2^7)=14.18 soit 15 bits : dès lors, l'approche Unicode présente un avantage.

De la même manière que pour l'attribution d'un code aux symboles, leur positionnement géométrique tire parti de l'approche Unicode proposée dès qu'il y a plusieurs symboles à positionner, pour devenir rapidement plus avantageuse.

De plus, comme précédemment rappelé avec l'encodage des symboles en Unicode, cette utilisation absolue de l'espace global ne représente que partiellement le problème, et le passage de ASCII vers Unicode aura des effet moindre sur cette dernière approche.

Dans ce cas, si 29 caractères servant à positionner dans un repère de 128x128 sont remplacés par 1 code pour positionner sur un repère 256x256, le gain est au maximum d'un facteur 29 - sans parler des avantages de précision obtenus puisque la résolution est elle multipliée par 4.

Toutefois, un supplément d'espace sera nécessaire pour les opérateurs voulus.

Il faut donc calculer l'espace nécessaire en considérant cela.

3-5-4. Évaluation de l'espace global nécessaire

L'espace nécessaire pour un symbole positionné va maintenant être étudié de manière globale, c'est à dire en considérant la table d'encodage des symboles, leur positionnement et les opérateurs nécessaires pour cela.

Avant d'évaluer cet espace global nécessaire, il faut toutefois décrire l'articulation qui va exister entre ces deux éléments.

En effet, deux méthodes d'encodage ont été présentées dans le cas des symboles : mono-champs, et multi-champs ou « bitwise ».

Les deux mêmes méthodes d'encodage sont possibles pour les informations géométriques de symboles, et il y a donc 4 possibilités d'articulation :

- encodage multi-champs du symbole, mono-champs de la géométrie
- encodage multi-champs du symbole, multi-champs de la géométrie
- encodage mono-champs du symbole, multi-champs de la géométrie
- encodage mono-champs du symbole, mono-champs de la géométrie

Considérons donc respectivement chaque cas selon le type d'encodage du symbole.

(i) Encodage multi-champs du symbole, mono-champs de la géométrie

Si le signe suit une approche mono-champs, il n'y a qu'une simple sommation des espaces.

Ce croisement d'approches ne présente aucun intérêt particulier.

(ii) Encodage multi champs du symbole, multi-champs de la géométrie

Si le symbole utilise une approche bitwise, le cas est plus complexe.

Il a été démontré précédemment que cette dernière est la méthode la plus rapide pour le traitement.

Par contre, elle présente l'inconvénient d'augmenter le nombre de bits nécessaires, selon l'information que l'on veut encoder, de manière très importante.

Choisir la méthode multi-champs pour les symboles implique donc de choisir pour les opérateurs et le positionnement cette même méthode, sauf à perdre les avantages de rapidité précédemment décrits puisqu'ils sont liés à la possibilité de calculer des masques de bits.

Mais dans ce cas, beaucoup d'espace est aussi nécessaire, et beaucoup est donc perdu.

Un autre inconvénient de ce choix est qu'il ne soit plus applicable aux autres formalismes d'écriture bidimensionnels, qui auraient pourtant intérêt à utiliser les mécanismes ici décrits : en effet, pour économiser de l'espace Unicode, symboles et positionnement doivent mélangés à l'intérieur d'une séquence de bits, encodée par plusieurs codes Unicode.

Un autre problème est qu'une telle approche est incompatible avec la norme Unicode.

Cette approche dite « liée » est donc à exclure.

(iii) Encodage mono champs du symbole, multi-champs de la géométrie

Dans ce cas, c'est l'approche bitwise de l'encodage de la position qui n'a aucun intérêt.

Cette approche est donc aussi à exclure.

(iv) Encodage mono champs du symbole, mono-champs de la géométrie

Cette approche, aussi dite « séquentielle » représente tout simplement les coordonnées du signe dans le système choisi par des codes séparés.

L'avantage de cette approche est qu'elle est utilisable pour les autres formalismes d'écriture, et respecte l'esprit de la norme Unicode.

Une fois des deux types d'articulations évoqués, considérons donc l'attribution de codes Unicode comme un problème d'encodage séquentiel de bits, et étudions les possibilités pour encoder ce flux en Unicode.

(v) Encodage des bits en Unicode

Une solution triviale pour encoder une séquence de ABCDEF bits de taille supérieure à l'espace Unicode disponible est de l'encoder sur deux codes Unicode, les bits de poids faible allant sur l'un, et les bits de poids fort sur l'autre, ce qui donne la séquence de codes U+ABC U+DEF (ou l'inverse, suivant que l'approche est petit-boutien ou gros-boutien).

Utiliser 1 plan ou 2 plans, ne permet que d'avoir 16 ou 17 bits : autrement dit, ceci n'offre pas de pas de différence significative.

Le problème de cette approche est la définition de la frontière dans la séquence de bit : si la frontière sépare symbole et géométrie, elle offre l'avantage de respecter au mieux la norme Unicode, mais en gaspillant plus de place.

Si elle les sépare de manière différente, elle offre la possibilité d'optimiser la répartition de place.

Étudions donc respectivement chaque cas.

(vi) Séparation propre symbole/géométrie

Avec l'approche mono-champs, un plan peut stocker le symbole (15 bits) : il est alors possible d'utiliser un autre code Unicode pour stocker la spatialisation qui se fait sur 14 ou 16 bits.

Prenons par exemple un code sur 16 bits.

(C == 16 bits)

C -> 0 symbole | 1 attribut

symbole -> symbole sur 15 bits

attribut -> 0000 spatialisation | 0001 contrôle (par exemple)

spatialisation -> xcoord ycoord

xcoord -> 6bits

contrôle = start | autre type de contrôle éventuel

Comme mentionné précédemment, une telle approche a l'avantage de respecter l'approche Unicode, en attribuant un code par symbole, et un autre (qui peut alors être un code privé) pour les informations supplémentaires comme les attributs et la spatialisation.

Une difficulté par contre précédemment notée est le rajout de nouveaux codes, qui ne pourra se faire que dans l'espace non attribué restant dans ce plan, nécessitant donc d'en optimiser l'aménagement pour en maximiser l'évolutivité.

De plus, une telle solution requérant un code pour le choix du symbole impose une approche mono-champs pour l'encodage de ce dernier, l'autre solution nécessitant sinon plus de bits que disponibles dans un champs : avec l'approche multi-champs, 23 bits sont nécessaires, soit plus d'un code.

Nous somme donc forcés d'utiliser plusieurs codes pour un seul symbole.

De prime abord, une telle possibilité semble peu conforme avec l'esprit de la norme Unicode, ou un code correspond à un seul caractère.

Étudions-en toutefois les modalités pratiques.

(vii) Séparation propre en plus de codes

Il est possible d'envisager cette même approche de répartition, en utilisant cette fois 2 codes pour le symbole et un pour la spatialisation.

Par exemple (C == 16 bis)

C -> 00 sym_lo | 01 sym_hi | 10 attribut

sym_lo -> 2 bits à 0 + 12 bits de poid faible du symbole (sur les 23),

sym_hi -> 3 bits à 0 + 11 bits de poid fort du symbole

attribut -> 00 spatialisation | 01 contrôle (par exemple)

spatialisation -> pareil que exemple précédent

contrôle -> pareil que exemple précédent

Une séquence de 3 codes est alors nécessaire au lieu d'une séquence de 2 codes, ce qui offre en échange une facilité de traitement, et une capacité d'évolutivité suivant les modifications de la norme SSS-2004.

Toutefois, on a alors besoin d'une séquence de 2 codes pour encoder un symbole, ce qui est contraire aux spécifications Unicodes.

En l'attente d'un élargissement d'Unicode ou d'une simplification de SignWriting, il s'agit toutefois de la seule approche possible pour l'approche multi-champs, à moins de recourir à une approche plus intelligente de séparation optimisée

(viii) Séparation optimisée

Imaginons donc une approche hybride mono/multi-champs, consistant à fusionner uniquement certain champs, c'est à dire ne stocker dans le premier code qu'un arbre permettant d'obtenir certains des 6 paramètres nécessaires (E0...E5).

Il a d'ailleurs été précédemment donné l'exemple de 13 bits pour 4 paramètres, les 2 paramètres restant étant encodés sur 10 bits.

Dans ce cas, une séquence de 3 codes est toujours nécessaires (2 pour le choix du symbole, 1 pour la géométrie) mais contrairement à l'approche mentionnée ci-dessus, une telle séparation optimisée devient pleinement envisageable dans l'esprit de la norme Unicode.

(ix) Discussion

Parmi les solutions possibles, plusieurs s'opposent, avec schématiquement:

- simplicité conceptuelle (utilisation de peu d'espace Unicode) en échange d'une difficulté algorithmique (pas de masque de bits)

- perversion d'Unicode (utilisation de plusieurs codes pour coder des bits) avec des gains algorithmiques (masques de bits)

L'approche la plus adaptée au problème semble être l'utilisation de deux codes Unicode par symbole, afin de permettre une évolution de la norme SSS et de faciliter l'encodage de symbole partiellement reconnus, dans le cas de systèmes de traitement d'image.

Mais cette solution n'est pas envisageable dans le respect de la norme Unicode si les 23 bits nécessaire sont répartis uniformément sur 2 codes.

Donc puisque chaque solution présentent des inconvénients propres, il faut envisager une approche hybride de séparation optimisée des paramètres, suivant la norme Unicode, en définissant ainsi le symbole dans le premier code et un modifieur du symbole dans le deuxième code.

Le positionnement géométrique, dans un souci d'extension au delà de SW gagnerait lui aussi à utiliser un encodage Unicode propre.

Ainsi, chaque opérateur de relation et chaque attribut de position devrait être encodé par un code Unicode séparé.

La lourdeur résultante et l'occupation importante de plans Unicode serait théoriquement compensée par l'apport conceptuel de ce nouveau mécanisme de rendu pour les autres formalismes d'écritures non encodés en raison des limitations (*positionning*) *[2002-Everson-Unicode] [2003-Anderson-Unicode_Historic_Scripts], [2004-Anderson-SEI-Unicode][2005-Cook-Hieroglyphs][2007-Everson-Hieroglyphs].*

De plus, vu la sous-utilisation des plans annexes d'Unicode, la plupart des formalismes d'écriture voulant être au premier plan *[2003-Everson-Roadmap_SMP]*, cette solution semble viable.

(x) Intérêt pour la compatibilité avec les logiciels de traitement d'image

D'un point de vue traitement d'image, le principal avantage de l'approche par masque de bits est de permettre une description floue des reconnaissances issues des systèmes de traitement d'image, qui ne sont pas toujours très précises ou complètes.

Les approches existantes comme SWML nécessitent par contre d'avoir identifié avec précision les 6 paramètres de chaque symbole composant un signe.

Mais s'il est possible de ne positionner des bits que pour les paramètres ayant été identifiés par un système de reconnaissance d'image, il est alors possible de disposer d'un format d'encodage flou des données : les algorithmes de rechercher/ remplacer sur des textes SignWriting restent fonctionnels, si les paramètres auxquels ils s'intéressent sont renseignés.

Un tel format peut même être utilisé pour l'annotation de vidéos : un système de menus ou d'aide à la numérisation peut alors toujours demander à l'utilisateur des précisions sur les paramètres non reconnus et donc non renseignés.

(xi) Conclusion

L'encodage Unicode d'un symbole à l'intérieur d'un signe doit faire appel à :

- un code pour le choix du symbole sur 13 bits
- un code pour le modifieur du symbole sur 10 bits
- un code par élément de grammaire (nombre et nature à déterminer)
- un code pour le positionnement géométrique du symbole sur un canevas de 128x128, soit 14 bits

Cette segmentation de la complexité et de la combinatoire du problème permet à la fois de l'exprimer de manière compatible avec Unicode, et d'envisager un support des autres formalismes d'écriture bidimensionnels en reprenant la grammaire et le positionnement géométrique, ainsi que le modifieur dans le cas de formalisme d'écriture envisageant des orientations spatiales.

La possibilité d'un positionnement d'un ensemble de signes dans un plan bidimensionnel, quelque soit le mécanisme d'encodage, est pour sa part évident.

Les éléments de grammaire peuvent être réduit au minimum à 3 codes.

Le postulat d'encodage Unicode est donc validé.

(xii) Critique

Bien que le postulat d'encodage Unicode soit ainsi validé, les différents éléments de cet encodage n'ont pas été implémentés, vu la nécessité de nombreux travaux, notamment avec des linguistes.

Les calculs proposés correspondent donc plus à une méthodologie d'implémentation, avec définition d'une formule de calcul de répartition de l'espace, et à une évaluation qu'à une implémentation proprement dite.

Toutefois, les points principaux de cette théorie d'encodage ont été modélisés, mesurés et évalués, afin d'orienter les choix logiciels et de les valider.

Un avantage supplémentaire est de pouvoir appliquer ces éléments à une version future de SignWriting, différente de SSS-2004, ou à un autre formalisme d'écriture de la langue des signes.

3-6. Applicabilité au delà de SignWriting, plan de mise en œuvre

Le plan de mise en œuvre suivant peut alors être proposé, vu la progression logique de la problématique retracée dans l'état de l'art et dans les travaux précédents.

(i) Définition des éléments du formalisme

Une première étape nécessaire est la définition des éléments du formalisme.

Dans le cas de SignWriting, cette définition est faire par SSS-2004, mais pour tout autre formalisme, il faut définir l'ensemble des symboles à encoder.

(ii) Transformation de certains symboles en modifieurs

Suivant la taille de l'ensemble des symboles, un certain nombre de paramètres permettant de les encoder doit être transformé en modifieurs.

Ceci est nécessaire pour suivre la requête du consortium Unicode de n'encoder les nouveaux formalismes d'écriture que sous forme décomposée (équivalent de NFD) pour des raisons de minimalisation de la redondance.

(iii) Attribution de codes aux symboles et aux modifieurs

Suivant les possibilités, des codes sont alors attribués, idéalement selon l'approche multi-champs, et sinon selon l'approche mono-champs.

(iv) Dessin d'une police de caractères

Une police de caractère est ensuite dessinée, permettant pour chaque symbole d'afficher le glyphe correspondant.

À ce stade, aucune composition n'est possible : les symboles ne sont que transformés en glyphes par un mécanisme bijectif simple.

(v) Définition de règles de composition

Cette police de caractère se voit alors complétée par la description d'une table de modification, selon le format choisi pour la police (OpenType, AAT...)

Cette table permet de gérer les modifieurs, pour pouvoir transformer les glyphes des symboles selon ces règles.

(vi) Définition d'opérateurs

Les modifieurs sont alors complétés par des opérateurs, permettant d'envisager la composition graphique multi-symboles. La grammaire d'articulation de ces opérateurs doit aussi être définie à ce moment, les opérateurs ne pouvant être définis isolément des éléments sur lesquels ils s'appliquent.

(vii) Description de la norme d'encodage de ce formalisme

Les codes attribués aux symboles et aux modifieurs se voient donc complétés, et une norme finale rédigée est établie.

Dans le souci d'une intégration par défaut la plus vaste possible, cette norme devrait alors être proposée au consortium Unicode.

À défaut d'une reconnaissance par Unicode, la norme pourrait être proposée au registre alternatif ConScript qui cherche à harmoniser les utilisations non officielles des espaces privés d'Unicode.

Toutefois, l'absence de standardisation de la norme d'encodage signifierait que les objectifs de diffusion et d'intégration à tous les systèmes d'exploitation ne pourraient être atteints.

(viii) Implémentation de la grammaire

La grammaire de fonctionnement de ces opérateurs est implémentée dans le mécanisme de typographie avancée d'un système d'exploitation donné, conformément à la norme acceptée.

À ce moment, il n'est pas contre pas encore possible de saisir des codes en entrée, si ce n'est de manière isolée pour tester cette implémentation de la grammaire.

(ix) Écriture d'une interface d'entrée

Une interface permettant aux utilisateurs d'entrer des codes conformément à cette grammaire est alors rajoutée, par un système quelconque (menus au clavier, menus à la souris...) d'entrée de données.

(x) Validation au niveau du système d'exploitation

Test du système global auprès d'utilisateurs finaux, vérification des contraintes de lisibilité au niveau de l'interface graphique, localisation du système d'exploitation...

(xi) Critique

Le plan de mise en œuvre proposé est arbitraire. Toutefois, pour tout implémenteur ne pouvant examiner en détail tous les aspects du problème, il présente l'avantage d'une suite de travaux à effectuer en série pour reproduire les différents éléments envisagés dans cette thèse.

Vu l'ampleur du travail, ce plan de mise en œuvre n'a pas encore été réalisé. Il nécessite un long développement logiciel.

L'imbrication des différents choix à faire, notamment lors de la répartition entre symboles et modifieurs, aurait intérêt à tirer parti d'un travail multidisciplinaire avec des linguistes, comme précédemment rappelé.

La couche de rentrée des données devrait pour sa part faire appel aux compétences de chercheurs en interface homme-machine, afin de définir des métriques utilisables pour l'évaluation des performances des utilisateurs novices, même dans le cas très spécifiques de formalismes d'écriture bidimensionnels.

Pour améliorer les performances d'entrée d'un point de vue traitement d'image, elle pourrait bénéficier aussi d'un outil d'acquisition par un flux vidéo.

De nombreux travaux sur ce sujet existent, et ont bien mentionné les difficultés existantes, notamment pour la détection de segments corporels, le suivi de ces derniers, la segmentation temporelle et sémantique, la contextualisation... *[2003-Gianni-Pose_reconstruction], [2004-Lenseigne-Gesture_representation].*

De telles questions n'entrent toutefois pas dans le sujet de cette thèse, qui analyse les méthodes possibles d'informatisation de la langue des signes, en envisageant particulièrement les contraintes existantes au niveau du système d'exploitation, et les aspects d'encodage nécessaires pour les différents éléments des couches d'entrée/sortie, par exemple pour faciliter l'interaction avec les systèmes de traitement d'image.

4. Conclusion

Les travaux réalisés se sont étendus sur des domaines très variés.

Certaines restrictions du sujet ont donc du être rajoutées au fur et à mesure des recherches, principalement pour les domaines correspondant à la linguistique.

D'un point de vue informatique, de nombreux apports ont toutefois été réalisés.

Ce chapitre présente donc successivement :

- une synthèse présentant le déroulement logique des travaux

- un rappel des originalités de l'approche suivie, avec les avancées pour l'échange de données, le traitement des langues et la langue des signes

- les perspectives de développement

4-1. Synthèse

Ont été respectivement étudiés l'état de l'art des formalismes d'écriture existants, des formalismes d'écriture des langues signées, puis les encodages précédents des formalismes d'écritures des langues dites naturelles et des langues signées.

Le fonctionnement d'Unicode a alors été analysé, ainsi que son implémentation pratique au niveau d'un système d'exploitation facilement examinable.

À partir de cet état de l'art, le sujet de cette thèse a été redéfini comme l'informatisation d'une forme écrite de la langue des signes, en respectant les contraintes spécifiques nécessaires à une généralisation de ce support au niveau des systèmes d'exploitation, en tenant donc particulièrement compte des contraintes d'Unicode et de la problématique particulière du traitement d'image.

Suite à cette refocalisation du sujet, plusieurs travaux ont été entrepris.

Tout d'abord, une étude sur site a été réalisée, afin de comprendre l'utilisation qui est faite de SignWriting et de voir quels problèmes pratiques se posent.

Cette étude sur le terrain a bien confirmé les difficultés liées à la mise en œuvre informatique de SignWriting vu l'utilisation faite des logiciels existants.

Un premier travail a donc été de resegmenter le problème du support d'une forme écrite de la langue des signes, traité habituellement comme un « tout », en suivant les apports de l'état de l'art.

Cette resegmentation a nécessité le postulat d'une possibilité d'encodage Unicode, qui a fait l'objet d'une proposition non détaillée d'encodage.

Cette proposition a toutefois permis de préciser la resegmentation, avec notamment une couche d'entrée, une couche de rendu, et une couche de polices.

Suite à cette resegmentation, des problèmes particuliers sont apparus, au niveau de la gestion de propriétés spécifiques de l'écriture de la langue des signes.

Ces propriétés spécifiques ont été mise en évidence et analysées. On a pu ainsi définir trois niveaux de variabilités : inter-personelle, intra-personnelle, de positionnement dans un canevas.

L'intérêt d'une conservation ou d'une neutralisation de chaque variabilité a été discuté, et un algorithme de neutralisation et de conservation ont alors été proposés.

Le postulat d'encodage Unicode a donc été repris. Ont été successivement analysés ses deux sous problèmes : encodage des symboles, et encodage de leur position géométrique.

Une modélisation de chaque aspect a été réalisée, pour mesurer les contraintes d'un point de vue temps de calcul et espace de stockage nécessaire, et ces mesures théoriques ont ensuite été validées par une expérimentation pratique.

Différentes articulations du choix du symbole et de son positionnement géométrique ont alors pu être discutées.

La dualité du problème a alors du être redéfinie, vu l'impossibilité d'un support en Unicode en suivant ces contraintes.

En examinant les différentes possibilités pratiques, une méthodologie est apparue, en séparant alors trois éléments : choix du symbole, modificateurs, positionnement géométrique, et l'intérêt d'opérateurs relationnels a pu être discuté, pour une sémantisation des relations au-delà du simple positionnement.

L'extensibilité du support Unicode à d'autres formalismes a ensuite été analysée.

Finalement, une méthodologie de support d'autres formalismes d'encodage a été proposée.

4-1-1. Apport sur la compréhension du problème

Le principal apport de cette thèse est son état de l'art, qui en examinant de manière détaillée le fonctionnement des différents aspects de support de l'écriture dans les systèmes d'exploitation permet de faire émerger une compréhension différente du problème.

Ceci a d'ailleurs donné lieu à une précision du sujet mettant bien en évidence l'importance du support Unicode vu les contraintes d'encodage, puisque la définition d'une grammaire n'existant pas encore pour les formes écrites de la langue des signes rend une approche purement linguistique très complexe.

Le problème du support Unicode a aussi été mieux compris grâce à la réalisation de tests pratiques mettant en évidence l'impossibilité de suivre certaines approches proposées.

4-1-2. Avantages de l'encodage proposé

L'encodage proposé en conclusion de ces mesures et expérimentations présente l'avantage d'encoder de manière compatible avec l'esprit de la norme Unicode un formalisme d'écriture bidimensionnel aussi complexe que SignWriting.

Il présente aussi la possibilité d'être appliqué, du moins en partie, à d'autres formalismes similaires, c'est à dire plus complexes que les formalismes mono-dimensionnels faciles à gérer avec Unicode et les différents mécanismes existants.

4-2. Originalité

La principale originalité de ce travail de thèse est de reconsidérer le problème de l'encodage de SignWriting comme un problème d'encodage de caractères, et non comme un problème de linguistique pure.

En conséquence, l'encodage proposé dispose d'une grammaire définissant des relations spatiales plutôt que sémantiques, en attendant que de telles règles soient définies. La possibilité de créer des opérateurs sémantiques a été étudiée, mais a été laissée à des travaux de linguistes, dans le cadre d'une extension future de l'encodage.

4-2-1. Avancées pour l'export et le partage des données

La principale originalité de l'approche proposée d'encodage est qu'elle établit une compatibilité logicielle, permettant l'export et le partage de donnée au-delà des applications spécifiquement conçues pour le support de SignWriting.

Pour celà, elle déplace les différents problèmes au niveau du système d'exploitation, en en respectant donc les contraintes spécifiques.

Ainsi, elle simplifie le développement logiciel en minimisant le degré de redondance des développements.

4-2-2. Avancées pour la langue des signes

Au-delà de l'aspect purement informatique, l'originalité de cette approche est de mettre la langue des signes sur un pied d'égalité avec les autres langues écrites.

Il devient donc envisageable de traduire complètement un système d'exploitation et ses applications en langue des signes française.

La quantité de travail serait bien sûr considérable, mais bien moindre que ce qu'il aurait été nécessaire selon une autre approche, puisque la gestion des formalismes bidimensionnels n'était alors pas envisageable sans des modifications lourdes et spécifiques à chaque logiciel.

4-2-3. Accélération du problème d'extraction de paramètres

Le problème d'extraction de paramètres des symboles, notamment dans le cas d'algorithmes de rechercher/remplacer était jusqu'à maintenant un problème de recherche.

L'accélération mesurée des méthodes proposées, de l'ordre d'un facteur 6 000, permet d'envisager la diffusion de telles technologies de rechercher/remplacer à des utilisateurs finaux, n'ayant qu'à attendre quelques secondes les résultats.

4-2-4. Algorithme de conservation et de neutralisation des variabilités

L'algorithme de conservation et de neutralisation des variabilités permet de définir des formes standards, et de mesurer les variations qui existent.

Sans prétendre en arriver à une standardisation de la langue des signes, il permet toutefois de mieux comprendre les difficultés jusqu'alors relevées dans l'écriture des langues signées.

Il présente l'originalité d'offrir le choix de conserver ou de neutraliser ces variabilités, sans forcer une approche ou l'autre pour les nombreuses raisons qui font que l'une ou l'autre peuvent être nécessaires à un moment donné (ex : rechercher/remplacer : sur une forme standard, afficher: la forme variable que veut l'utilisateur).

4-3. Perspectives de développement

Les perspectives de développement tirent parti des avancées réalisées.

4-3-1. Implémentation de la méthodologie proposée

Une première perspective de développement est le suivi de la méthodologie proposée, pour apporter un support informatique adéquat de SignWriting ou de tout autre formalisme d'écriture de la langue des signes.

4-3-2. Localisation des applications et du système

Comme précédemment mentionné, la localisation complète d'un système d'exploitation devient envisageable, pour permettre aux sourds de disposer de plus de logiciels dans leur langue.

4-3-3. Support d'ordinateurs peu rapides

En faisant particulièrement attention aux problèmes de temps de traitement logiciel, l'approche proposée ouvre le support de la langue des signes à des ordinateurs moins véloces que ceux utilisés actuellement.

Il s'agit d'une approche à contre-courant de la recherche habituelle, qui propose plutôt des solutions applicables dans le futur, lorsque le matériel aura évolué.

Une des raisons de l'intérêt du support d'ordinateurs peu rapides est la diffusion actuelles d'ordinateurs à bas coûts pour les pays en voie de développement, où se situe une grande partie des utilisateurs de SignWriting.

Signalons ainsi l'existence du projet OLPC, qui veut offrir pour le coût le plus réduit possible un ordinateur portable équipé d'un microprocesseur AMD à chaque enfant.

4-3-4. Amélioration des connaissances sur les variabilités

En appliquant l'algorithme proposé de manière à conserver et identifier les variabilités, il devient possible d'éditer un dictionnaire des « formes locales » de la LSF.

En effet, la manière de signer un même concept peut varier en France selon les régions, et il est important de conserver l'originalité de ces expressions (patrimoine linguistique).

La recherche en linguistique sur la langue des signes peut ainsi bénéficier des apports réalisés par l'informatique.

4-3-5. Extension aux autres formes écrites

La normalisation des mécanismes d'encodages spatiaux proposés permettrait d'envisager un support d'autres formalismes d'écriture pour l'instant eux-aussi non supportés par Unicode.

Lors d'une évaluation des manques d'Unicode, il a ainsi été retenu que 8% des écritures actuellement non supportées nécessiteraient de tels mécanismes avancés de rendu *[2002-Everson-Unicode]*.

Des travaux sont en cours pour le support de ces écritures, principalement pour les langues historiques *[2004-Anderson-SEI-Unicode]*.

Ils pourraient bénéficier *à minima* des mécanismes de positionnement proposés en cas de leur normalisation par Unicode.

4-3-6. Compatibilité des recherche sur la langue des signes

Jusqu'à maintenant, de nombreuses méthodes d'informatisation de la langue des signes existaient.

Ainsi, la compatibilité des travaux de recherche pâtissait de ces différences souvent très importantes.

La possibilité d'un support standardisé, de référence, ouvre la porte à une meilleure diffusion des résultats de recherche entre les équipes.

4-4. Perspectives de recherche

De multiples questions ont été posées tout au long de cette thèse, notamment d'un point de vue statistique et linguistique.

Elle mettent en évidence le manque de connaissances actuelles sur la langue des signes, et donc les difficultés rencontrées lors de cette thèse.

En effet, il n'a pas été possible de répondre à toutes les interrogations, et bien souvent des hypothèses ont du être suivies.

Une meilleure connaissance de la langue des signes, en terme de syntaxe mais aussi de statistiques liées à son usage, permettrait de passer du stade théorique de la possibilité d'informatisation correcte d'une forme écrite de la langue des signes, vers le stade pratique.

Cette thèse part en effet sur l'hypothèse du support Unicode de la langue des signes.

En testant et en discutant différents aspects, elle explore cette hypothèse. Par la mesure de différents points particulièrement problématiques, dont par exemple l'espace nécessaire au sein d'Unicode, elle vérifie la validité de cette hypothèse.

On peut donc parler d'une théorie du support Unicode de la LSF.

Toutefois, il est encore nécessaire de passer de la théorie à la pratique, ce qui vu la quantité de problèmes soulevés nécessitera encore plusieurs années de travaux.

4-4-1. Problème du mécanisme d'entrée

Un autre problème est l'absence d'une généralisation de mécanismes logiciels pour gérer les dispositifs d'entrée autres que le clavier.

Toutefois, tout ne dépend pas de l'encodage clavier, et l'approche proposée ne saurait s'y limiter. Il semble crucial de disposer de plusieurs interfaces d'entrée de donnés, afin au moins de laisser le choix de la métaphore d'entrée à l'utilisateur.

Mais excepté sur les Tablet PC ou les PDA qui disposent de systèmes de reconnaissance de l'écriture et de claviers virtuels, les systèmes d'exploitation n'incluent pas de logiciels adaptés à une saisie rapide des données, si ce n'est dans le cas particuliers des systèmes d'exploitation destinés au marché asiatique.

Seuls quelques logiciels comme « Table des caractères » sont sinon inclus à titre d'exemple pour montrer les symboles non disponibles sur les claviers usuels.

Ceci rend nécessaire de concevoir des techniques d'intégration particulières, reposant sur des mécanismes qui n'ont pas été spécifiquement prévus pour l'entrée de données (ex : communications inter applications / OLE / Corba, presse papier...).

Certains systèmes d'exploitation comme Linux permettent toutefois une injection directe des flux de données dans le kernel par des pilotes « Input », ce qui simplifie la conception logicielle de la gestion de l'entrée.

Ainsi, l'utilisateur pourrait choisir la métaphore qui lui convient le mieux pour entrer des données en langue des signes (clavier virtuel, reconnaissance d'écriture, système de menus, reconnaissance de gestes...).

Après avoir exploré le fonctionnement du mode d'entrée au niveau du système d'exploitation, cette thèse ne s'est pas étendue sur les détails du mécanisme d'entrée à prévoir pour répondre aux besoins de la langue des signes.

Des travaux de recherche supplémentaire seront donc nécessaires.

4-4-2. Problème de l'évolution du formalisme d'écriture

Les linguistes français travaillent actuellement sur d'autres formalismes d'écriture, dans le but de mieux supporter certaines spécificités de la langue des signes française que SignWriting [2007-Garcia-LSSCRIPT].

Ces derniers pourraient implémenter [2004-Lejeune-Analyse_LSF] les phases d'iconicité (technique de description spatiale précise d'éléments ou d'actions en langue des signes en faisant appel à leur aspect audiovisuel) [1996-Cuxac-Iconicité] ou les transferts personnels (technique de citation de personnes en langue des signes, à travers l'instanciation en une position spatiale) *[1999-Sallandre-Transferts]*.

Pour celà, certaines propositions utilisent des formes tridimensionnelles de représentation de l'espace de signation, initialement développées pour représenter les mouvements articulaires *[2004-Lenseigne-Gesture_representation]*.

À l'image des signes standards précédemment évoqués, ce support pourrait être réalisé par l'intermédiaire de méta-données. D'autres informations pourraient aussi se voir intégrées sous forme de méta-données, afin de faciliter des applications spécifiques comme l'annotation de vidéos *[2004-Brafford-Annotation].*

L'encodage présenté pourrait supporter assez facilement de telles modifications, en rajoutant simplement aux paramètres encodés par des codes Unicode ces données supplémentaires.

En leur réservant dès maintenant des plages de valeurs, il est possible d'obtenir une compatibilité ascendante : les anciens systèmes, dont l'implémentation en serait absente au niveau de leur moteur Unicode, pourraient les ignorer, et représenter tous les autres paramètres, alors que les systèmes connaissant cette notation pourraient à partir du même flux de données Unicode représenter l'intégralité de l'information, à l'instar de MdC *[2007-Everson-Hieroglyphs]*.

Toutefois, une bonne utilisation de ces méta-données pour des fonctions de manipulations et de comparaison des symboles pourrait parfois nécessiter une modification du code source des logiciels employés.

Une telle évolution du formalisme d'écriture est à envisager, sous la forme de travaux spécifiques et plus pratique que cette thèse.

4-4-3. Questions ouvertes

De nombreuses questions ont été rajoutées par cette thèse, et persistent :

Quelle est la tolérance de SignWriting au rétrécissement des documents ? Jusqu'à quelle taille restent-t-ils lisibles ?

Faut-il en tenir compte dans les polices ?

Quelle est la taille des polices préférées par les utilisateurs de SignWriting ?

Est-il possible, vu cette taille, d'utiliser SignWriting dans les menus des logiciels sans modification particulière ?

Faut-il sinon, pour conserver la lisibilité, modifier de manière non proportionelle les espacements entre les groupes de symboles, ou les tailles des symboles ?

Est-ce qu'un outil de reconnaissance d'image peut isoler des zones dans un document où se trouvent des signes en SignWriting, puis les reconnaître comme le fait un OCR ?

Comment réaliser des requêtes sur des symboles ou des signes ?

Quels sont les paramètres importants pour de telles requêtes ?

Quelle taxonomie des signes peut être établie ?

Peut-on en bénéficier pour offrir à l'utilisateur un choix par ressemblance des signes, ou appartenance à un même phylum sémantique ?

Est-ce que l'approche statistique du choix et du positionnement des symboles présente un avantage réel pour les utilisateurs ?

Est-ce qu'il y a ainsi la possibilité de représenter en SignWriting ce qui relève des propriétés spécifiques de la LSF, comme les trajectoires des mains de la grande iconicité, ou la logique des transferts personnels ?

Est-ce que de nouveaux opérateurs et symboles pour ces aspects spécifiques de la LSF doivent être définis ?

Quelles sont les relations sémantiques possibles entre les signes ? Quelles sont les groupes sémantiques de signes possibles ?

Est-ce qu'un nouveau formalisme d'écriture aura la possibilité de s'imposer ?

Quelle sera la possibilité et la facilité de compréhension de ce formalisme par rapport à SignWriting ?

Est-il possible de réaliser un avatar signant sans ajouter d'élément au formalisme d'écriture ?

Est-ce que le rajout des relations sémantiques et des méta-informations sémantiques rend l'encodage suffisant pour réaliser un signeur virtuel ?

Quels symboles inutilisés, issus de symboles de base croisés avec certains paramètre, doit-on supprimer ?

À partir de toutes ces questions, de nombreuses extensions du sujet peuvent être envisagées.

Au delà de ces perspectives et des questions ouvertes, nous avons proposé plusieurs actions de développement ou de recherche, décrites en annexe.

5. Bilan

Cette thèse a exploré de manière la plus vaste possible le sujet, et n'a donc pas toujours pu se pencher précisément sur tous les détails, ce que je regrette.

Néanmoins, vu l'ampleur de la tâche, la précision de tous les aspects n'est pas possible pour une seule personne dans le cadre d'une thèse, sauf à négliger plusieurs parties du sujet, et donc se montrer moins exhaustif.

Cette thèse est principalement un travail exploratoire : plusieurs travaux supplémentaires doivent être envisagés dans le cadre d'un plan de mise en œuvre, que ce soit pour SignWriting ou tout autre formalisme d'écriture.

Toutefois, de multiples apports ont été réalisés, à la fois par un état de l'art très complet pouvant bénéficier à tout autre chercheur devant aborder ce même problème, et par des mesures et évaluations précises dont les résultats supportent la théorie d'un encodage Unicode de SignWriting, malgré toutes les contraintes de complexité notées dans l'état de l'art.

Au delà de SignWriting, il s'agit maintenant d'avancer vers d'autres formalismes d'écriture, que ce soit pour la langue des signes ou pour toute autre langue, et d'implémenter.

À ce titre, plusieurs questions et sujets ont été posées, pour inspirer d'autres travaux futurs.

Après avoir passé plusieurs années à travailler sur ce problème, j'ai acquis et j'espère avoir transmis une meilleure connaissance de gestion de l'écrit, surtout sur un aspect encodage, et d'apports à science informatique sur le cas bien particulier de l'écriture de la langue des signe, ainsi que, au-delà de la langue des signes, de l'écriture des langues bidimensionnelles.

Le travail qui reste à faire est immense - néanmoins, les perspectives sont à la hauteur de la tâche, et principalement pour les sourds, qui ont tout à gagner d'un meilleur support de leur si belle langue.

Je suis très fier d'avoir pu réaliser une thèse d'informatique, et c'est un peu - même beaucoup - grâce à vous tous. Merci donc d'avoir encouragé et soutenu ces travaux de recherche.

J'espère qu'ils pourront être utiles à la communauté scientifique, et surtout à la communauté sourde, en lui permettant de se doter d'outils de communications informatiques d'une perfection comparable à la beauté de la Langue des Signes.

Tables des références

Il est parfois complexe de se procurer la bibliographie, surtout avec les politiques tarifaires des éditeurs scientifiques et les budgets à la baisse des services de documentation de nos universités.

Toutefois, la validation d'un travail passe par la vérification de ses références bibliographiques.

À cet effet, pour faciliter ce travail sinon pénible, toutes les références de cette thèse sont disponibles sous forme numérique par demande auprès de l'auteur, dans le cadre du droit à la copie privée

De plus, certaines références ont servi à fournir plusieurs illustrations, à titre didactique : il convient donc de les signaler séparément pour faciliter l'attribution des crédits qui sont dus.

Les illustrations de LSF du §1 sont issues de *[1987-Moody-LSF]*, les tables d'encodage du §2 sont issues de *[2005-Dik-Codes]* et les illustrations des plans Unicode et de composition du §2 sont issues de *[2002-Andries-Unicode]*.

[1825-Bébian-Mimographie] : Bébian A., « Mimographie ou essai d'écriture mimique, propre à régulariser le langage des sourds-muets », p. 9, L. Colas, Paris

[1853-Berthier-Éducation] : Berthier F., « Observations sur la mimique considérée dans ses rapports avec l'enseignement des sourds-muets », lettre adressée le 13 juin 1853 à l'Académie Impériale de Médecine, Paris

[1931-Hessen-SocioEconomic_Science] : Hessen B., "The socio-economics roots of Newton's Principia". In N. Bukharin (Ed.), Science at the Cross Roads. Papers presented to the International Congress of the History of Science and Technology, 1931, by the delegates of the USRR. London: Frank Cass.

[1954-Fitts-Law] : Fitts P. M., "The information capacity of the human motor system in controlling the amplitude of movement", in Journal of Experimental Psychology, volume 47, number 6, June 1954, pp. 381-391. (Reprinted in Journal of Experimental Psychology: General, 121(3):262--269, 1992).

[1960-Stokoe-Structure] : Stokoe W.C., "Sign language structure: An outline of the visual communication systems of the American deaf", Occasional Paper 8, pp. 21-28, University of Buffalo

[1979-Mandel-Constraints] : Mandel M.A.,"Natural constraints in sign language phonology: Data from anatomy", In: Sign Language Studies 8: 24,

[1986-Ecma-latin1_4] : ECMA, "Standard ECMA-94 : 8-Bit Single Byte Coded Graphic Character Sets - Latin Alphabets No. 1 to No. 4", 2e édition, juin 1986

[1987-Moody-LSF] : Moody B., « La langue des signes », Éd. International Visual Théâtre, Tome 1, Édition 1998, ISBN 2-904-6411-73, Paris

[1987-Prillwitz-HamNoSys] : Prillwitz, S. et al., "HamNoSys. Hamburg Notation System for Sign Languages. An introduction", Ed. Zentrum für Deutsche Gebärdensprache, Hamburg, 1987

[1990-Buxton-Chord_Keyboards] : Boxton B., "Pragmatics of Haptic Input", Tutorial 26 Notes of CHU'90, 6.1-6.9, ISBN: 0-521-40409-6, Édition 2002

[1991-Cuxac-Problème-Écrit] : Cuxac C, « L'éducation des sourds en France et le problème de l'accès à l'écrit », in A. Bentolila et al., « La lecture, apprentissage, évaluation, perfectionnement », Éd. Nathan, Paris, pp 255-258

[1991-Unicode-Standard] : Unicode Consortium, "The Unicode Standard", ISBN 0-321-18578-1

[1992-Hummel-Guide_programmeur] : Hummel R., « Guide de réference du programmeur pc: processeur et coprocesseur », Ed Ziff-Davis, ISBN 2-100-01405-6

[1992-Lagally-ArabTex] Lagally K., "ArabTe{X} : typesetting arabic with vowels and ligatures", EuroTeX92, Proceedings, Prague

[1993-ISO-UCS] : International Standard Organisation, "ISO/IEC 10646-1:1993(E) Information Technology-Universal Multiple-octet Coded Character Set (UCS)"

[1995-Brouwer-Keyboard] : Brower A., "The Linux Keyboard Howto", The Linux Documentation Project, http://tldp.org/HOWTO/Keyboard-and-Console-Howto.html

[1995-Jouison-LSF] : Jouison P., « Écrits sur la langue des signes française », Éd L'Harmattan, ISBN 2-7384-3535-1

[1995-Sutton-SignWriter] : Sutton V., Gleaves R., "SignWriter - The world's first sign language processor" Ed. Center for Sutton Movement Writing, La Jolla

[1996-Baynton-Forbidden_Signs] : Bayton D. C., "Forbidden Signs : American Culture and the Campaign against Signs", University of Chicago Press, ISBN 0-226-03964-1, pp.136-139.

[1996-Bray-XML] : Bray T., Sperberg-McQueen C.M., "eXtensible Markup Language", w3c working draft WD-XML-961114

[1996-Cuxac-Iconicité] : Cuxac C., « Fonctions et structures de l'iconicité dans les langues des signes ; analyse descriptive d'un idiolecte parisien de la Langue des Signes Française », Thèse de Doctorat d'État, Université René Descartes Paris V

[1996-Hasan-Sanskrit_Chinese] : Hasan M., "Using Chinese Text Processing Technique for the Processing of Sanskrit Based Indian Languages: Maximum Resource Utilization and Maximum Compatibility. ", International Conference on Chinese Computing (ICCC), Proceedings

[1996-Horstmann-Word_predictin] : Horstmann Koester, H. & Levine, S., "Effect of a word prediction feature on user performance", Augmentative and Alternative Communication, 12, p. 155-168.

[1997-Aznar-Francophones] : Aznar G., « Le Francophones HOWTO : Linux & la langue française », The Linux Documentation Project, http://tldp.org/HOWTO/ Francophones-HOWTO.html, Version 3.2.2, Décembre 2001

[1997-Cuxac-Spatial] : Cuxac C., « Expression des relations spatiales et spatialisation des relations sémantiques en Langue des Signes Française », Diversité des langues et représentations cognitives, C. Fuchs et S. Robert (éds.), Éd Ophrys, Paris, pp. 150-164

[1997-Cuxac-Iconicité] : Cuxac C., « Iconicité et mouvement des signes en langue des signes française » in Actes de la sixième École d'été de l'Association pour la recherche cognitive, pp. 205-218

[1997-Vergé-Regard] : Vergé F., "Structure grammaticale de la langue des signes. Rôle et fonctions du regard dans la Langue des Signes Française", Mémoire de DEA, Université Toulouse Le Mirail

[1997-Wilcox-Dynomite] : Wilcox L. D., Schilit B. N., Sawney N., "Dynomite : A Dynamically Organised In and Audio Notebook", in CHI 97, Proceedings, Atlanta

[1998-André-Euro] : André J., « Iso-Latin9, euro et typographie française » in Document Numérique volume 2 n°2, pp. 231-240, Éd Lavoisier, Cachan

[1998-Chiu-Dynamic-Grouping] : P. Chiu, and L. Wilcox, "A Dynamic Grouping. Technique for Ink and Audio Notes", In the Proc. ACM. UIST, pp. 195-202.

[1998-Fanton-FSA_Arabic] : Fanton M. "Finite State Automata and Arabic Writing", Proceedings of the Workshop on Computational Approaches to Semitic Languages (COLING-ACL'98), Montréal

[1998-Gillot-Rapport_Sourds] : Gillot D. et al, « Le droit des sourds », Rapport remis à Lionel Jospin, Association Nationale des Parents d'Enfants Sourds

[1998-Delsarte-Levenshtein] : Delsarte P., Levenshtein V.I., "Association schemes and coding thory", in Information Theory, IEEE Transactions on, Vol. 44 Issue 6, ISSN: 0018-9448, October 1998

[1998-Schilit-Active_Reading] : Schilit B., Golovchinsky G., Price M., "Beyond Paper: Supporting Active Reading with Free Form Digital Ink Annotations", in ACM CHI 98, Proceedings, v.1 249-256

[1998-ISO-8850-4] : ISO, "ISO/CEI 8859-4:1998 - 8-bit single-byte coded graphic character sets, Part 4: Latin alphabet No. 4" (draft dated February 12, 1998, published July 1, 1998)

[1998-ISO-8859-10] : ISO, "ISO/CEI 8859-10:1998 - 8-bit single-byte coded graphic character sets, Part 10: Latin alphabet No. 6" (draft dated February 12, 1998, published July 15, 1998)

[1998-ISO-8859-15] : ISO, "ISO/CEI 8859-15:1998 - 8-bit single-byte coded graphic character sets, Part 15: Latin alphabet No. 9" (draft dated August 1, 1997; superseded by ISO/CEI 8859-15:1999, published 15 mars 1999)

[1998-ISO-8859-1] : ISO, "ISO/CEI 8859-1:1998 - 8-bit single-byte coded graphic character sets, Part 1: Latin alphabet No. 1" (draft dated February 12, 1998, published April 15, 1998)

[1998-Savage-Computation] : Savage J.E., "Models of Computation: Exploring the Power of Computing" p. 160, Ed. Addison-Wesley, ISBN 0-201-89538-0

[1999-Aznar-Euro] : Aznar G., "Linux and the Euro Currency : Toward a Global Solution", in Linux Journal volume 1999, Issue 59es, Article 23, ISSN 1075-3583, March 1999

[1999-ECMA-8859-9] : ECMA, "Standard ECMA-128 : 8-Bit Single-Byte Coded Graphic Character Sets - Latin Alphabet No. 5" 2e édition (december 1999)

[1999-Ecma-8859-5] : ECMA, "Standard ECMA-113 : 8-Bit Single-Byte Coded Graphic Character Sets - Latin/Cyrillic Alphabet" 3e édition (december 1999)

[1999-Everson-Hieroglyphs] : Everson M., "Encoding Egyptian Hieroglyphs in Plane 1 of the UCS", ISO/IEC JTC1/SC2/WG2 WorkGroup document N1944

[1999-Ferraiolo-SVG] : Ferraiolo J., "Scalable vector graphics (SVG) specification" w3c working draft W D-SVG-19991203

[1999-Sallandre-Transferts] : Sallandre M.A., "La dynamique des transferts de personne en Langue des Signes Française", Mémoire de DEA sous la direction de Christian Cuxac, Paris VIII, Saint-Denis, 1999

[1999-ISO-8859-11] : ISO, "ISO/CEI 8859-11:1999 - 8-bit single-byte coded graphic character sets, Part 11: Latin/Thai character set" (draft dated June 22, 1999; superseded by ISO/CEI 8859-11:2001, published Dec 15, 2001)

[1999-ISO-8859-13] : ISO, "ISO/CEI 8859-13:1998 - 8-bit single-byte coded graphic character sets, Part 13: Latin alphabet No. 7" (draft dated April 15, 1998, published 15 octobre 1998)

[1999-ISO-8859-7] : ISO, "ISO/CEI 8859-7:1999 - 8-bit single-byte coded graphic character sets, Part 7: Latin/Greek alphabet" (draft dated 10 juin 1999; superseded by ISO/CEI 8859-7:2003, published 10 octobre 2003)

[1999-Liefke-XMill] : Liefke H., Suciu D., "XMill: an Efficient Compressor for XML Data", University of Pennsylvania Technical Report MSCIS p. 153-164

[1999-Martin-Wap_XML] : Martin B., Jano. B., "Wap binary xml content format" w3c recommendation wbxml

[1999-Mudur-Shaping_Indian] : Mudur S.P, Nayak N, Shanbhag S., Joshi R.K., "An architecture for the shaping of Indic texts", Computers & Graphics 23 (1999) 7— 24

[1999-Rosenberg-Writing_ASL] : Rosenberg, A., "In Support of Adopting an ASL Writing System", Writing Signed Language, Dept. of Linguistics, Univ. of Kansas

[2000-Babcock-CHA] : Babcock C., "Introduction to CHA - Current Ideas on How to Proceed", in http://www.kanji.com/kc/cha/introduction-to-cha.html, 2000

[2000-ECMA-8859-8] : ECMA, "Standard ECMA-121 : 8-Bit Single-Byte Coded Graphic Character Sets - Latin/Hebrew Alphabet" 2e édition (december 2000)

[2000-Ecma-8859-6] : ECMA, "Standard ECMA-114 : 8-Bit Single-Byte Coded Graphic Character Sets - Latin/Arabic Alphabet" 2nd édition (december 2000)

[2000-Hummels-Design] : Hummels C., "Gestural design tools: prototype, experiments and scenarios", Thèse de Doctorat, Technische Universiteit Delft, ISBN 90-9014013-1

[2000-ISO-8859-16] : ISO, "ISO/CEI 8859-16:2000 - 8-bit single-byte coded graphic character sets, Part 16: Latin alphabet No. 10" (draft dated november 15, 1999; superseded by ISO/CEI 8859-16:2001, published july 15, 2001)

[2000-Losson-Signeur-Virtuel] : Losson O., «Modélisation du geste communicatif et réalisation d'un signeur virtuel de phrases en Langue des Signes Française», Thèse de Doctorat, Université des Sciences et Technologies de Lille, janvier 2000

[2000-Ward-Dasher] : Ward D. J, Blackwell A.F, MacKay D.J.C, "Dasher - a data entry interface using continuous gestures and language models", UIST-00. pp. 129-137.

[2001-Atkin-BiDi-Implementations] : Atkin S. Stansifer R., "Implementations of Bidirectional Reordering Algorithm", 18th International Unicode Conference, Proceedings, Hong Kong, April 2001

[2001-Butler-SW_Ordering] : Butler C., "An Ordering System for SignWriting", The SignWriting Journal, Number 1, Article 1, August 2001

[2001-Crasborn-Signphon] : Crasborn O., Van Der Hulst H. Van Der Kooij E., "SignPhon : A phonological database for sign languages", in Sign language & linguistics, vol 4, n°1-2, pp. 215-228, ISSN 1387-9316

[2001-DaRocha-SW_Similarity] : Da Rocha Costa A., Dimuro G. P., "Supporting Deaf Sign Languages in Written Form on the Web", The SignWriting Journal, Number 0, Article 1, July 2001

[2001-DaRocha-SWML] : Da Rocha Costa A., Dimuro G. P., "A SignWriting-Based Approach to Sign Language Processing", Gesture Workshap 2001, Londres (Royaume Uni)

[2001-DaRocha-SignWriting] : Da Rocha Costa A., Dimuro G. P., "A SignWriting-Based Approach to Sign Language Processing",GestureWorkshap 2001,Londres

[2001-Flood-Learning_to_write_in_SW] : Flood C. M., "How do deaf and hard of hearing students experience learning to write using SignWriting, a way to read and write signs", Thèse de doctorat, University of New Mexico, Albuquerque

[2001-Hoiting-BTS_For_Understanding] : Hoiting N, Slobin D. I., "Transcription as a tool for understanding : the Berkeley Transcription System for Sign Languge Research (BTS)", published in G. Morgan & B. Woll (Eds.), Directions in sign language acquisition (pp. 55-75).

[2001-Herledan-Kanjis_Histoire] : Herledan J., « Historique des Kanjis », Nihon-No-Tomodachi, http://kanji.free.fr/download/pdf/historique.pdf

[2001-Ratheesh-Linux_Console_Indian] : Ratheesh K, "Console based Indian language support for the Linux operating system", Project report for the Master of Technology degree, Indian Institute of Technology, Madras, January 2001

[2001-Taentzer-Graphs_Exchange_Format] : Taentzer G., "Towards Common Exchange Formats for Graphs and Graph Transformation Systems", Uniform Approaches to Graphical Process Specifications Techniques 2001 (UNIGRA01), Proceedings

[2001-Taub-Iconicity] : TAUB S. "Language from the Body. Iconicity and Metaphor in American Sign Language", Ed Cambridge University Press, ISBN 0-521-77062-9

[2002-André-Codage] : André J., « Caractères, codage et normalisation – de Chappe à Unicode », Éditions Lavoisier+Hermès, ISBN 2-7462-0594-7, vol. 6, no 3-4, 2002, p. 13-49.

[2002-Andries-Unicode] : Andries P., « Introduction à Unicode et à l'ISO 10646 », éditions Lavoisier Document numérique 2002- 3/4 (Volume 6), ISSN 1279-5127

[2002-Cuxac-LSCOLIN] : Cuxac C. et al, « Rapport de fin de recherche Langage & Cognition LSCOLIN », Projet LACO 39, Université de Paris 8, Septembre 2002

[2002-Everson-Unicode] : Everson M, "Leaks in the Unicode pipeline: script, script, script...", 21st International Unicode Conference, Proceedings, Dublin

[2002-Huenerfauth-ASL_Generation] : Huenerfauth M., "Natural Language Generation and Machine Translation for ASL", CIS-899 Independent Study Report, Philadelphia, University of Pennsylvania

[2002-Knoblich-Predicting_Strokes] : Knoblich G., Seigerschmidt E., Flach R, Printz W., "Authorship effects in the prediction of handwriting strokes", Q. J. Exp. Psychol. A 55, 10271046.

[2002-MacKenzie-KPC] : MacKenzie I. S., "KSPC (keystrokes per character) as a characteristic of text entry techniques. Proceedings of the Fourth International Symposium on Human-Computer Interaction with Mobile Devices, pp. 195-210. Heidelberg, Germany, Ed. Springer-Verlag

[2002-Norrie-Web-Integration] : Norrie M. C., Signer B., "Web-Based Integration of Printed and Digital Information", DIWeb'02 - 2nd Windowshop on Data Integration over the Web, Proceedings, Toronto, 2002

[2002-Poser-Athabaskan] : Poser W.J., "Making Athabaskan Dictionaries Usable", in Proceedings of the Athabaskan Languages Conference, G. Holton, Fairbanks, Alaska Native Language Center, University of Alaska, pp 126-147

[2002-Séguillon-Échec] : Séguillon D., « Du language des signes à l'apprentissage de la parole, ou l'échec d'une réforme », Rapport de Recherche, Revue Staps 2002 n°58, ISSN 0247-106X

[2002-Torchelsen-SWEdit] : Torchelsen R. P., Da Rocha Costa A., Dimuro G. P., "Editor para Língua de Sinais Escritas em SignWriting " in XII Simpósion Brasileiro de Informática na Educaçaõ - SBIE 2002, São Leopoldo, 2002

[2003-Anderson-Unicode_Historic_Scripts] : Anderson D., "Unicode and Historic Scripts", in Ariadne Issue 37, October 2003

[2003-Aznar-Émergence] : Aznar G., « Conditions et critères pour l'émergence de l'icônicité dans la Langue des Signes Française », mémoire de DEA, Université Paul Sabatier, Toulouse

[2003-Aznar-Zaurus_C700] : Aznar G., "Review: the Sharp Zaurus SL-C700", in Linux Journal, Issue 110 (June 2003), p13, ISSN:1075-3583

[2003-Boutet-Formalisation_Graphique_LSF] : Boutet D., Garcia B., « Vers une formalisation graphique de la Langue des Signes Française (LSF) : Éléments d'un programme de recherche », La Nouvelle revue de l'adaptation et de la scolarisation, ISSN 1957-0341, n°23 pp 49-62, 2003

[2003-DaRocha-SignWriting] : Da Rocha Costa A., Dimuro G., "SignWriting and SWML: Paving the way to sign language processing", Actes de TALN-2003, Batz-sur-Mer, juin 2003

[2003-Everson-Roadmap_SMP] : Everson M., McGowan R., Whistler K., "Roadmap to the SMP", version 4.2, Unicode Consortium Publication, september 2003

[2003-Garcia-Notation_Transcription_LS] : Garcia B., « Notation et transcription des langues des signes », Support de cours du Diplôme de Premier Cycle Universitaire de Linguistique appliquée aux langues des signes, Université Paris 8, avril 2003

[2003-Gianni-Pose_reconstruction] : Gianni F, Lenseigne B., Dalle P. "3d Pose Reconstruction of a Human Arm using a Single Camera and a Biomecanic Model", 5th International Gesture Workshop 2003, Gênes, avril 2003

[2003-DeLanghe-Traitement_sémantique_LSF] : DeLanghe O., Guitteny P. Portine H., Retoré C., « Vers un traitement informatique de la syntaxe et de la sémantique de la langue des signes », Actes de TALN-2003, Batz-sur-Mer, juin 2003

[2003-Tayon-Technosceptique] : Tayon J., «Comment passer pour un techno sceptique quand on aime l'informatique », in http://julbox.net/text/ainfo.shtml, 2003

[2003-Rocha-Graffiti-SW] : Rocha F.Z., "Proposta de dum Padrão Manuscrito para Reconhecimento Automático dos Simbolos do Sistema SignWriting (SW)", Projeto de Graduação do Curso Ciência da Computação, Universidade Católica de Pelotas, 2003

[2003-Sallandre-Unites_Discours] : Sallandre M.A., « Les unités du discours en Langue des Signes Française. Tentative de catégorisation dans le cadre d'une grammaire de l'iconicité », Thèse de doctorat en sciences du language de l'Université Paris 8, décembre 2003

[2003-Slobin-Acquisition] : Slobin D. I., "Sign Language Acquisition : Typical and Atypical Development", Society for Research in Child Development, Biennial Meeting, Proceedings, Tampa, april 2003

[2003-Stumpf-SW] : Stumpf M. R., "Aprendizagem de Escrita de Língua de Sinais Pelo Sistema SignWriting : Comunidades Virtuais de Surdos", Thèse d'Informatique de l'Éducation de l'Université Fédérale de Porto Alegre, décembre 2003

[2003-Ward-Taking_Class_Notes] : Ward N., Tatsukawa H., "Software for Taking Notes in Class", in 33th ASEE/IEEE Frontiers in Education Conference, Proceedings, Boulder

[2004-Aerts-Searching-SignWriting] : Aerts S., Braem B.,Van Mulders K., De Weerdt L., "Searching SignWriting Signs", Actes du Workshop RPSL, LREC 2004, p79-81

[2004-Anderson-SEI-Unicode] : Anderson D., "Unicode in Multilingual Text Projects : A Status Report from the Script Encoding Initiative, UC Berkeley", ALLC/ ACH Conference 2004, Proceedings

[2004-Aznar-FREU] : Aznar G, Dalle P., "Computer Support for SignWriting Written Form of Sign Language", Actes du Workshop RPSL, LREC 2004, p. 109-110, Lisbon

[2004-Aznar-PDA_Linux] : Aznar G., "Why Run Free Software on a PDA?" in http://www.oreillynet.com/pub/a/linux/2004/01/29/zaurus.html, Ed. O'Reilly

[2004-Bayardo-Binary_XML] : Bayardo R., Gruhl D., Josifovski V., Myllymaki J. , "An Evaluation of Binary XML Encoding Optimizations for Fast Stream Based XML Processing", WWW2004, ACM 1-58113-844-X/04/0005. May 17-22, 2004, New York

[2004-Brafford-Annotation] Braffort A., Choisier A., Collet C., Dalle P., Gianni F., Lenseigne B., Segouat J., "Toward an annotation software for video of Sign Language", Actes de LREC 2004

[2004-Constable-ZWI] : Constable P., "Proposal on Clarification and Consolidation of the Function of ZERO WIDTH JOINER in Indic Scripts ", Public Review Issue #37 L2/04-279, Unicode Consortium, June 2006

[2004-Crasbord-Sharing_data] Crasborn O., Kooij E., Broeder D., Brugman H. , "Sharing sign language corpora online:proposals for transcription & metadata",Actes du Workshop RPSL, LREC 2004, p20-23

[2004-DaRocha-Sign_Matching] : Da Rocha A., Dimuro G., De Freitas J., "A sign matching technique to support searches in sign language texts", Actes du Workshop RPSL, LREC 2004, p32-34

[2004-Fusellier-Apparition-LS] : Fusellier-Souza I., « La création gestuelle des individus sourds isolés. De l'édification conceptuelle et linguistique à la sémiogénèse des langues des signes », AILE (Acquisition et Interaction en Langue Étrangère), n°15, pp. 6-96, Paris

[2004-Hepsos-Conceptual_Precursors] : Hepsos S.J., Spelke E., "Conceptual precursors to language", Nature, 430, 453-456.

[2004-Herrero-SEA] : Herrero A., "A Practical Writing System for Sign Languages", Actes du Workshop RPSL, LREC 2004, p37-42

[2004-Larson-Recognition] : Larson K., "The sicence of Word Recognition", Microsoft Corporation, Microsoft Typography Website, June 2004, http:/ /www.microsoft.com/typography/ctfonts/WordRecognition.apsx

[2004-LaViola-Mathpad2] : LaViola J., Zeleznik R., "MathPad2 : A System for the Creation and Exploration of Mathematical Sketches", ACM Transactions on Graphics, Proceedings of SIGGRAPH 2003, 23(3):432:440, August 2004

[2004-Lejeune-Analyse_LSF] : Lejeune F, "Analyse sémantico-cognitive d'énoncés en langue des signes française pour une génération automatique de séquences gestuelles", thèse de docteur en informatique de l'Université de Paris-XI Orsay, novembre 2004

[2004-Lenseigne-Gesture_representation] : Lenseigne B., Gianni F., Dalle P., "A new Gesture Representation for Sign language analysis", Actes de LREC 2004 p85-90, Lisbonne

[2004-McDermott-Occluding_contour] : McDermott J., Adelson E. H., "The geometry of the occluding contour and its effect on motion interpretation ", Journal of Vision (2004) 4, 944-954.

[2004-PrélazGirod-Accès_Écrit] : Prélaz-Girod A.C., "Accès à l'écrit grâce à SignWriting: une solution pour les enfants sourds ?", Travail d'approfondissement en Orthophonie, Université de Neuchâtel, février 2004

[2004-Roald-Making_dictionnaries] : Roald I., "Making Dictionaries of Technical Signs: from Paper and Glue through SW-DOS to SignBank", Actes du Workshop RPSL, LREC 2004, p75-78

[2004-Rodd-Semantic_ambiguity_modelling] : Rodd J. M., "Modelling the effets of semantic ambiguity in word recognition", Cognitive Science, Vol. 28, No. 1, pp. 89-104

[2004-Spagnolo-Superposed_Strokes] : Spagnolo G. S., Simonetti C., Cozzella L., "Superposed strokes analysis by conoscopic holography as an aid for a handwriting expert", Institute of Physics Publishing, Journal Of Optics A : Pure And Applied Optics, J.Opt.A:PureAppl.Opt.6(2004)869–874

[2004-Sutton-SSS] : Sutton V., "International Movement Writing Alphabet", Ed. Center for Sutton Movement Writing, La Jolla (États-Unis), ISBN: 0-914336-83-5

[2004-Vandamme-Websourd] : Vandamme E., « Cahier des charges de développement et intégration des applications web, Sites Websourd V1 », Agence Conseil Insite, janvier 2004

[2005-Aerts-Semantic-Searching] : Aerts S., Braem B., Van Mulders K., De Weerdt K., "Semantic Searching for SignWriting", in : TALN 2005 - atelier Traitement Automatique de la Langue des Signes, Dourdan, Actes, LIMSI, juin 2005

[2005-Aznar-Préserver] : Aznar G., Dalle P., « Informatisation de formes écrites des langues des signes », dans : 2e Congrès de l'International Society for Gesture Studies (ISGS) : Interacting Bodies / Corps en interaction, Lyon, École Normale Supérieure Lettres et Sciences Humaines, juin 2005

[2005-Aznar-Approche] : Aznar G., Dalle P. « Une nouvelle approche pour informatiser une forme écrite de la langue des signes », dans : Colloque des Doctorants, ÉDIT 05, Université Paul Sabatier, avril 2005

[2005-Aznar-Variabilité] : Aznar G., Dalle P., « Variations dans la représentation écrite d'un signe en Signwriting », dans : TALN 2005 - atelier Traitement Automatique de la Langue des Signes, Dourdan, Actes volume 2 pp. 381-384, LIMSI, juin 2005

[2005-Bray-Dictionnary] : Bray P., "Building an On-line New Zealand Sign Language Dictionnary", Graduation Thesis, p. 34, Victoria University of Wellington

[2005-Brewbaker-Genetic_Keyboard] : Brewbaker C. R., "Optimizing stylus keyboard layouts with a genetic algorithm: customization and internationalization", State University of Iowa, http://www.public.iastate.edu/crb002/ie574final

[2005-Cook-Hieroglyphs] : Cook R., Everson M., McGowan R., Richmond R., "Revised proposal to encode Egyptian hieroglyphs in Plane 1 of the UCS", Unicode Consortium, Working Group Document L2/05-311, october 2005

[2005-Dik-Codes]:Dik W.T.,"Codes",http://homepages.cwi.nl/~dik/english/codes July 2005

[2005-Leroy-Pédagogie] : Leroy É., « Didactique de la LSF. Fondement pédagogique dans l'éducation de l'enfant sourd », mémoire de DEA de l'Université de Paris 8, Sciences du Languages

[2005-Skliar-Bilingual_Brazil] : Skliar C., Quadros R. M., "Bilingual Deaf Education in the South of Brazil" in DeMejía A. M., "Bilingual Education in South America", Ed. Multilingual Matters, ISBN 1-853-59819-4, pp. 40-46

[2005-Snoek-Video_Indexing] : Snoek C. G. M., Worring M., "Multimodal Video Indexing: A Review of the State-of-the-art", Multimedia Tools and Applications, 25, 5–35, 2005

[2005-Unicode-4.1] : Unicode Consortium, "The Unicode Standard Version 4.1", The Unicode Consortium

[2005-Watson-Pixels] : Watson B., Luebke D., "The Ultimate Display: Where Will All the Pixels Come From?", IEEE Computer 38(08) : 54-61, August 2005

[2005-Zhai-Sokgraph] : Zhai S., Kristensson P.O., Smith B.A., "In search of effective text input interfaces for off the desktop computing", Interacting With Computers, Volume 17 Issue 3, May 2005, pp-229-250

[2006-Aznar-Algorithm] : Aznar G., Dalle P., Ballabriga C., "Analysis of the differents methods to encode SignWriting in Unicode", in Workshop on the Representation and Processing of Sign Language:Lexicographic Matters and Didactic Scenarios (LREC 2006), Genoa, Italy, 28/05/06-28/05/06, Evaluations and Language resources Distribution Agency (ELDA), p. 59-63, 2006.

[2006-Boutet-Enjeux] : Boutet D., Garcia B., « Finalités et enjeux linguistiques d'une formalisation graphique de la Langue des signes Française (LSF) » in Glottopol, n°7, janvier 2006, Université de Rouen, ISSN 1769-7425

[2006-Chaparro-Legibility] : Chaparro B. S., Shaikh A. D., Chaparro A., "The legibility of Cleartype Fonts", in Human Factors and Ergonomics Society Annual Meeting Proceedings, pp 1829-133

[2006-Davis-BiDi] : M. Davis, "The Bidirectional Algorithm - Unicode Standard Annex #9", Unicode Consortium, Revision 17

[2006-Freytag-Breaking] : Freytag A., "Unicode Standard Annex #14 - Line Breaking Properties", Unicode Consortium, Revision 19

[2006-Trager-International_Layout] : Trager E. H., "International Text Layout & Typography: The Big And Future Picture", Gnome Live, Proceedings, Boston, October 2006

[2007-Boutet-Structuration_LSF] : Boutet D., Garcia B., « Structuration morpho-phonétique de la langue de signes française (LSF) ; étude à partir d'une base de données », Actes de TALN 2007, Toulouse, 12-15 juin 2007

[2007-Garcia-LSSCRIPT] : Garcia B., Brugeille J.L., Mercier H., Dalle P., Braffort A., « Projet LS-SCRIPT : Rapport Final », Projet RIAM, Université Paris 8, Iris, Websourd, IRIT, LIMSI-CNRS,

[2007-Haralambous-Encodings] : Haralambous Y., "Fonts & Encodings", Ed O'Reilly, ISBN 0-596-10242-9

[2008-Davis-Segmentation] : Davis M., "Unicode Standard Annex #29 Proposed Update : Unicode Text Segmentation", Unicode Consortium, Revision 12

6. Annexe : extensions du sujet

Définissons le plus précisément possible quelques unes de ces extensions du sujet, réalisables par différent profils professionels et pour différents buts, sous forme d'une liste de sujets.

Choisissons par exemple :

- mise en oeuvre d'une partie de la solution : travail de recherche informatique

- réponse à une des questions posées : travail de développement

- mise en oeuvre un peu plus complète de la solution : travail de développement

- amélioration des connaissances sur la langue des signes : travail de recherche linguistique

6-1. Proposition d'un sujet de travail pour la mise en oeuvre d'une partie de la solution

Les logiciels existants placent habituellement par défaut les signes rentrés sur le canevas à la position du curseur, ou à l'angle supérieur gauche de la fenêtre d'édition.

Dans tous les cas, l'utilisateur doit déplacer ce signe jusqu'à la position correcte requise, ce qui nécessite un grand nombre d'interactions clavier et souris fastidieuses.

L'approche probabiliste proposée au problème de positionnement d'un symbole peut définir une fonction p(s,s',s'x,s'y) du symbole courant s et du symbole précédent s' placé dans le canevas de saisie du signe à des coordonnées s'x et s'y.

Pour ces 4 paramètres en entrée, cette fonction p donne en sortie sx et sy, les coordonnées par défaut proposées pour le symbole s courant.

La fonction p est établie par exemple par appretissage utilisant un corpus de texte SignWriting

La validation doit alors se faire en mesurant la différence entre la position x,y choisie par l'utilisateur après un repositionnement éventuel du symbole courant s et sa position sx,sy telle que prédite initialement par la fonction p.

Pour celà, on utilise soit un autre corpus de texte SignWriting, soit une validation auprès d'utilisateurs habitués à SignWriting.

6-2. Proposition d'un sujet de travail pour répondre à une des questions posées

Évaluation par des utilisateurs des tailles préférées et minimales de signes SignWriting

(i) Objectif

Préciser quelle est la taille des signes SignWriting que préfèrent les utilisateurs, et la taille minimale à laquelle ils restent visibles, afin d'utiliser cette taille optimale pour l'affichage d'une forme écrite de la langue des signes. De telles données, disponibles pour les langues écrites en alphabet romain, sont disponibles pour les polices les plus courantes. Elles font totalement défaut pour l'écriture de la langue des signes.

(ii) Matériel

Un évaluateur maîtrisant bien SignWriting, N signes choisis par cet évaluateur, et N textes comprenant ces signes. 3 écrans d'ordinateurs (TFT, LCD, OLPC). Des utilisateurs à tester.

(iii) Méthode

Un écran d'ordinateur affiche un signe SignWriting en commençant par la taille la plus petite.

Cette taille peut être augmentée par le sujet testé. Dès qu'il pense pouvoir décrypter le signe, l'utilisateur le réalise gestuellement à l'évaluateur. Ce dernier détermine s'il y a réussite ou échec.

En cas d'échec, l'évaluateur augmente la taille du signe affiché à l'écran, et demande au sujet testé de reessayer, sans lui donner aucun autre indice.

Une fois le signe trouvé, la taille minimale pour une lisibilité est notée.

L'utilisateur peut alors librement régler la taille du signe affiché à une taille qui lui semble confortable tout en restant lisible au sein d'un texte.

Afin de suggérer à l'utilisateur de ne pas choisir une taille trop grande, le contexte du signe est affiché à l'écran, et augmente de taille de la même manière que le signe testé.

Si la taille de l'écran est atteinte, tout le texte n'est pas affiché, et des barres de défilement apparaissent à l'écran. Une fois l'utilisateur satisfait de son confort de lecture, la taille préférée est notée.

Le processus recommence, pour un série de N signes (et N contextes) par utilisateur. L'expérience est réalisée consécutivement avec 3 écrans : un écran TFT, un écran LCD, un écran d'OLPC, dont les paramètres résolutions (DPI) et taille réelle (en pouces) sont connus.

(iv) Traitement des données

Un tableau est réalisé pour chaque utilisateur testé, prenant en ligne le signe testé, et en colonne l'âge du sujet testé, note la taille préférée notée en pixels, en pouces, et en rapport taille en pouce/taille de l'écran.

Les paramètres (composition du signe testé, position x-y des éléments) sont aussi saisis.

Les données sont alors soumises à une analyse statistique.

6-3. Proposition d'un sujet de travail de développement de la solution

Écriture d'une nouvelle forme graphique de la langue des signes française

(i) Objectif

Implémenter une forme informatique d'un nouveau formalisme d'écriture de la langue des signes française défini par ailleurs, en commençant dans un premier temps par se limiter aux configurations de la main, pour à la fois valider les méthodes proposées et offrir un outil de saisie utilisable au quotidien

(ii) Matériel

Ordinateur, clavier, souris, système d'exploitation unicode, language C

(iii) Description

Un utilisateur non nécessairement habitué à l'informatique doit pouvoir saisir le plus rapidement possible, mais en n'utilisant que des périphériques habituels (comme un clavier et une souris) la configuration tridimensionnelle de la main. Ensuite, il doit pouvoir visualiser le résultat sous un formalisme différent, dit écriture.

Pour la saisie, un système de menus est d'abord utilisé, afin de guider initialement l'utilisateur dans des configurations de la main les plus fréquentes et les plus éloignées les unes des autres, dites « poses ».

Une fois une pose choisie, ses propriétés supplémentaires sont saisies avec un nombre limité d'événements clavier et souris. Le temps de saisie et le nombre de ces événements sont conservés, afin de pouvoir comparer les performances de différentes approches.

Vu l'absense de formalisme définitif, le stockage informatique du résultat obtenu utilise les propriétés articulaires, et non un quelconque système de codage, pour permettre une plus grande flexibilité des travaux.

La notion articulaire différencie plusieurs états (flexion/extension/neutre) pour chaque articulation, et si nécessaire (état non neutre), elle donne le degré exact de flexion de l'articulation :

- Les états neutres sont définis comme involontaires. Ils prennent en compte la propagation de contraintes anatomiques (ex : difficulté de fléchir le petit doigt en gardant le doigt précédent étendu).

- Pour les états non neutres, les degrés de liberté de chaque articulation sont considérés, en étant simplifiés à la réalité anatomique (ex : interphalangienne proximale et distale : trochlée : 1 degré de liberté : 0/45°, métacarpo-phalangienne : énarthrose : 3 degrés de liberté sur deux axes : -45/+45 et -10/+90)

Le résultat en sortie lorsqu'il est fourni en entrée au logiciel d'interface de saisie permet donc de reconstituer une configuration tridimensionnelle de la main.

Il s'agit de la première étape : la saisie.

Elle est distinguée d'une deuxième étape : l'encodage

En effet, la même séquence, lorsqu'elle est fournie à un autre logiciel dit interpréteur, permet d'obtenir une séquence Unicode.

À partir de la forme de stockage informatique, libre, un encodage utilisant nécessairement Unicode est généré, de manière à expérimenter les capacités d'Unicode.

Il est recommandé de suivre la méthode proposée dans cette thèse, mais les règles d'encodages pour générer un flux Unicode sont laissées libres.

Il pourrait ainsi être choisi de ne différentier que les propriétés qui donnent une forme graphique stylisée différente, ce qui est d'ailleurs recommandé pour des raisons de simplicité.

Ainsi, l'interpréteur est un logiciel séparé de l'interface de saisie, et des séquences légèrement différentes d'un point de vue articulaire/interface de saisie peuvent avoir une même forme graphique stylisée, conformément aux simplifications faites dans les spécifications de ce nouveau formalisme d'écriture pour des formes visuellement très proches. De telles simplifications sont réalisées par l'interprétateur à partir de la forme articulaire.

Ceci crée toutefois une confusion entre les représentations de la configuration de la main, qui font parties d'un formalisme de notation pour transcrire, et une écriture - il conviendra donc à terme de remplacer l'interpréteur par un mécanisme logiciel plus évolué.

La séquence Unicode ainsi obtenue, lorsqu'elle est passée au système d'exploitation, donne une forme graphique stylisée correspondant à une écriture de la langue des signes.

Il s'agit de la troisième étape : l'interprétation système. Pour cela interviennent les règles de composition, et une police de caractères.

Règles et police doivent être rajoutées au moteur Unicode du système d'exploitation choisi, afin que tout logiciel de ce système d'exploitation puisse tirer parti de cette séquence Unicode, si elle est intégrée dans un document.

Les règles de composition à partir d'un flux Unicode donné permettent d'obtenir la définition du dessin, dit glyphe, pour une configuration de la main. Ce glyphe est une sorte de canevas, qui en utilisant la police de caractères permet d'obtenir le dessin définitif. Un glyphe est donc composé d'un ou plusieurs caractères.

(iv) Implémentation

La description de technique d'encodage de Adamono-Villani est utilisée comme référence pour la méthode d'entrée. Les périphériques de saisie sont exclusivement le clavier voire la souris. La saisie sans souris reste possible.

Les configurations initiales dites "poses" sont issues des HandShapes de Paola Laterza. La réalité anatomique est spécifiée par un ouvrage qui donne les degrés de libertés et les angulations articulaires possibles généralement acceptées par les anatomistes.

À partir de cette forme de stockage, un interprétateur donne une forme écrite conforme aux spécifications de ce nouveau formalisme d'écriture. Cette forme écriture utilise un encodage binaire dans le plan privé Unicode.

À partir de cet encodage binaire, une forme graphique est générée par le moteur Pango du système d'exploitation GNU/Linux. Les règles de composition sont donc rajoutées à Pango, et une police de caractères simplifiée est utilisée par le système.

(v) Chronologie d'implémentation

Définition exhaustive des caractères à partir des références d'encodage : cette définition donne un exemple graphique de ce caractère et une description textuelle en français. Par exemple : 0=neutre 1=doigt droit ; 2=doigt crochu ; 3=doigt fléchi. Un code arbitraire est donc associé à chaque caractère.

Dessin d'une police de caractères sachant représenter chaque caractère. À ce stade, un glyphe n'est constitué que d'un caractère (ex : 1 -> dessin d'un doigt ouvert)

Définition d'une grammaire de composition d'un glyphe à partir des références d'encodage : cette grammaire, à partir de l'association de caractères, doit permettre d'obtenir la description un seul glyphe.

Par exemple : 22222 -> les cinq doigts crochus ; 333341 les quatre doigts fléchis au niveau de la métacarpophalangienne et le pouce droit. Des codes arbitraires sont rajoutés au besoin pour définir cette grammaire (ex : code 4=métacarpophalangienne)

Rédaction de la liste des codes correspondant aux caractères (doigt ouvert; doigt crochu ...) et à la grammaire de composition (pouce, index ...) : description de la fonction de chaque code (ex : définir le rôle du code 1 au code 4)

Écriture logicielle des règles de grammaire pour le moteur Unicode, afin de pouvoir composer graphiquement un glyphe à partir d'une suite de caractères.

Par exemple, la main droite étant représentée le pouce à droite, si le pouce est fléchi ou crochu, il est graphiquement représenté en dessous et à droite des autres doigts .

Par contre, s'il est droit, il est représenté à droite des autres doigts.

(Évaluation du travail à mi parcours)

Écriture d'un logiciel de saisie des configurations articulaires, ou utilisation au besoin de logiciels de référence donnant un format articulaire.

Écriture d'un transcripteur, sachant à partir du format articulaire produire la séquence grammaticale de caractères.

Validation au sein du système d'exploitation : saisie d'une forme articulaire par l'interface, représentation immédiate dans la forme graphique en utilisant toutes les étapes précédentes.

6-4. Proposition d'un sujet de travail pour améliorer les connaissances sur la LSF

Réalisation de requêtes pour la recherche, l'indexation et la labellisation du contenu en langue des signes.

Définition d'une syntaxe, utilisant des critères d'importance du groupement perceptuel (cotemporalité, répétitivité, saillances...).

Définition des critères d'importance pour la requête.

Transformation d'une requête en paramètres.

Test de la requête, et validation de l'approche.