

METAFONT

GUIDE PRATIQUE

Bernard Desgraupes



METAFONT

Guide pratique



VUIBERT INFORMATIQUE

Conception couverture : Jean Widmer

© Vuibert, Paris, 1999

ISBN 2-7117-8642-0

Toute représentation ou reproduction intégrale ou partielle, faite sans le consentement de l'auteur, ou de ses ayants droit, ou ayants cause, est illicite (loi du 11 mars 1957, alinea 1^{er} de l'article 40). Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait une contrefaçon sanctionnée par les articles 425 et suivants du Code pénal. La loi du 11 mars 1957 n'autorise, aux termes des alinéas 2 et 3 de l'article 41, que les copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective d'une part, et d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration.

Table des matières

Introduction	1
à K., Z., F. et S.	1
1. Introduction à la TATTOO	2
1.1. L'anthropologie culturelle	2
1.2. Anthropologie de l'art	2
1.3. Précisions sur les termes	3
2. Anthropologie du tatouage	4
2.1. Les éléments culturels	4
2.1.1. Le tatouage comme moyen de communication	4
2.1.2. Propriétés culturelles des tatouages	5
2.1.3. Signification des tatouages	5
2.1.4. Culture et tatouage	6
2.1.5. Tatouage et culture	6
2.2. Les éléments politiques	7
2.2.1. Les politiques	7
2.2.2. Le système Polynésien	7
2.2.3. Le système Polynésien et le tatouage	8
2.2.4. Les deux éthiques dans le tatouage	8
2.2.5. Le tatouage comme moyen de communication	9
2.2.6. Le tatouage comme moyen d'expression	9
2.2.7. Le tatouage comme moyen de communication	9
2.2.8. Communication et tatouage parmi les Polynésiens	10
2.2.9. Le tatouage comme moyen d'expression	10

Table des matières

Introduction	xiii	
1.1	La typographie numérique	1
1.2	Historique	2
1.3	Premiers pas avec METAFONT	4
2.1	Les éléments nécessaires	7
2.2	Principe de fonctionnement	8
2.2.1	Premières épreuves	10
2.2.2	Épreuves définitives	11
2.2.3	relax	11
2.2.4	Familles de polices ou « métapolices »	12
2.2.5	Tester des polices	14
2.2.6	La police <i>gray</i>	14
2.2.7	La police <i>black</i>	16
2.2.8	Utilisation d'une police avec T <small>E</small> X ou L <small>A</small> T <small>E</small> X	17
2.2.9	Autres utilisations de METAFONT	19
2.3	Constitution du format <i>plain</i>	20
2.3.1	Les bases de METAFONT	20
2.3.2	Le fichier <code>local.mf</code>	20
2.3.3	Compilation du format <i>plain</i>	21
2.3.4	Utilisation d'autres bases	21

3 Les macros METAFONT	23
3.1 Les objets	23
3.1.1 Opérateurs booléens	24
3.1.2 Opérateurs numériques	25
3.1.3 Images	30
3.1.4 Chaînes	30
3.1.5 Transformations	31
3.1.6 Chemins	34
3.1.7 Stylos	37
3.1.8 Paires	38
3.2 Instructions de programmation	40
3.2.1 Commandes non conditionnelles	40
3.2.2 Commandes conditionnelles	43
3.3 Commandes de dessin	44
3.3.1 Définition des chemins	44
3.3.2 Choix d'un stylo	44
3.3.3 Tracés et contours	46
3.3.4 Manipulation de tracés	47
3.4 Exécution d'une tâche	48
3.5 Pixellisation	50
3.6 Gestion des caractères et des polices	54
3.7 Affichage écran	57
3.8 Épreuves	58
3.9 Diagnostics	60
3.10 Divers	62
4 Utilisation de METAFONT	65
4.1 Les courbes	65
4.1.1 Courbes de Bézier	65
4.1.2 Placement des points	66
4.1.3 Premiers coups de crayon	67
4.1.4 Tangentes imposées	70
4.1.5 Tension	71

4.1.6	Taille et forme des crayons	73
4.1.7	Figures de base	76
4.2	Calligraphie	78
4.3	Chemins et images	82
4.3.1	Cheminement sur les courbes	82
4.3.2	Transformations	84
4.3.3	Sélection de pixels	87
4.3.4	Manipulation des images	90
4.3.5	Points d'intersection et points de contact	93
4.3.6	Instructions répétitives	95
4.4	Usage étendu de METAFONT	97
4.4.1	Création de macros	97
4.4.2	METAFONT solveur d'équations	102
4.4.3	Utilisation du hasard	105
4.4.4	Faire parler METAFONT	109

5	Construire une métapolice	111
5.1	Construction d'une police	111
5.1.1	Créer des caractères	112
5.1.2	Organisation du fichier pilote	113
5.1.3	Préambule du fichier pilote	114
5.1.4	Définitions des lettres	116
5.1.5	Ligatures et approches de paires	139
5.1.6	Informations globales	140
5.1.7	Fichier de paramètres	140
5.2	Modifier globalement une police	141
5.2.1	Utilisation de <i>currenttransform</i>	142
5.2.2	Modification d'un fichier de paramètres	143
5.3	Forme et série	143
5.3.1	Caractères penchés	144
5.3.2	Caractères gras	144
5.4	Empattements	145
5.4.1	Style romain	145

5.4.2	Style courrier	150
5.4.3	Style flamboyant	153
5.5	Variations	154
5.5.1	Contours	154
5.5.2	Négatifs	156
5.5.3	Motifs	157
5.6	Questions d'approximation	159
5.7	Lisibilité des caractères	161
5.8	Vers de nouvelles métapolices	162
6	METAFONT au jour le jour	165
6.1	Les différents niveaux d'utilisation	165
6.2	Trois exemples pratiques	166
6.2.1	Le symbole de l'euro	166
6.2.2	Symboles musicaux	167
6.2.3	Hiéroglyphes	170
6.3	Les limitations de METAFONT	175
A	Les polynômes de Bernstein	177
B	Tables de conversion	179
B.1	Unités de mesure	179
B.2	Proportions relatives des caractères	180
C	Les variables internes	183
D	Les messages d'erreur	187
D.1	Messages d'aide	188
D.1.1	Comment réagir à une erreur	188
D.1.2	Erreurs de calcul	189
D.1.3	Erreurs de langage	191
D.2	Erreurs de syntaxe	194
D.2.1	Erreurs sur les variables logiques	194
D.2.2	Erreurs sur les équations	195

D.2.3	Erreurs sur les chemins	196
D.2.4	Erreurs sur les transformations	198
D.2.5	Erreurs sur les images	198
D.2.6	Erreurs sur les polices	199
E	Tables de caractères	201
F	<i>logocomplet</i> : le programme	215
G	Les modes disponibles	231
H	Comment se procurer METAFONT?	241
H.1	Distributions	241
H.2	Utilisateurs francophones	243
H.3	Liste de diffusion	244
I	Bibliographie	245
Liste des tableaux		251
Liste des illustrations		253
Index		259

deuxième édition de METAFONTbook, il est nécessaire de rappeler que ce livre n'est pas destiné à l'usage des débutants : il suppose une connaissance préalable de la programmation en langage C et quelques notions de la théorie des automates et des langages formels. Il n'est pas destiné à l'usage des utilisateurs de METAFONT qui ne connaissent pas les fondamentaux de la typographie et de la conception de caractères. Il est destiné à l'usage des utilisateurs expérimentés qui souhaitent approfondir leurs connaissances et leur maîtrise de METAFONT.

Introduction

METAFONT est un programme conçu conjointement avec T_EX et dont le rôle est à la fois de produire des polices de caractères et d'agir sur les polices existantes pour en modifier l'aspect en jouant sur d'innombrables paramètres. Mais METAFONT est aussi un langage : il fournit ainsi toute la richesse et la flexibilité que l'informatique peut apporter afin de réaliser les tâches pour lesquelles il a été conçu. Ce langage permet de construire d'autre part des ensembles de commandes destinées à l'utilisateur pour accomplir des tâches particulières : chacun peut se constituer son propre réservoir de commandes (appelées aussi macros).

Cet ouvrage présente l'intégralité des macro-commandes qui constituent ce qu'on appelle le format *plain* de METAFONT. Ces commandes fournissent des outils extrêmement puissants pour la création de polices et de « métapolices » de caractères destinées à des documents élaborés avec T_EX ou L_AT_EX : d'infimes modifications permettent de faire varier à l'infini les formes des caractères d'une police aussi bien que les symboles créés.

Les outils sont ici regroupés par type, et leur usage ainsi que leur syntaxe sont expliqués afin, d'une part, de constituer un outil de référence et, d'autre part, d'atteindre progressivement par la pratique l'objectif de tout utilisateur de METAFONT : créer facilement des symboles nouveaux ou de nouvelles polices de caractères et même des familles de polices de caractères.

The METAFONTbook de D. Knuth, auteur de METAFONT et de T_EX, est le livre de référence sur le sujet. Il reste néanmoins de lecture difficile car sa présentation est déroutante : l'information s'y trouve extrêmement dispersée et le lecteur doit frayer son chemin à travers un itinéraire touffu comportant des notes parfois très spécialisées approfondissant tel ou tel détail (les fameux « virages »). On y trouve tout mais il en reste une impression de confusion qui a sans doute desservi METAFONT si l'on pense, par ailleurs, au succès rencontré par T_EX qui lui est pourtant intimement lié. Nous adoptons ici, dans un but de clarification, une méthode d'exposé systématique des macros de METAFONT, assorties d'illustrations, en rassemblant toute l'information dispersée dans The METAFONTbook. Le lecteur curieux qui aura déjà fait l'apprentissage de METAFONT et assimilé le présent ouvrage se plongera certainement avec délices dans les arcanes du METAFONTbook afin d'approfondir les techniques.

Dans le chapitre 1, on trouvera une introduction résumant les apports de la typographie.

graphie numérique aux techniques d'impression et de composition et le rôle révolutionnaire joué par METAFONT et par TEX en ce domaine. Le chapitre 2 présente l'installation puis la mise en œuvre de METAFONT : tout ce qu'il faut savoir pour faire tourner des programmes et rendre utilisables les polices de caractères que l'on aura créées (ou obtenues sur l'Internet). Dans le chapitre 3, toutes les macros sont regroupées en fonction de l'objet auquel elles se rapportent : ce chapitre doit servir de référence pour retrouver rapidement une macro avec sa syntaxe et dans son environnement. Les chapitres 4, 5 et 6 sont consacrés à la pratique de METAFONT. Le chapitre 4 concerne l'apprentissage de base de METAFONT : pas à pas on verra, avec de très nombreux exemples et illustrations, comment s'organisent et s'appliquent toutes les macros. Le chapitre 5 est intégralement consacré à la tâche de construction d'une métapolice avec toutes les variantes que cela suppose : il est la démonstration de la puissance de ce concept et de METAFONT lui-même. Enfin le chapitre 6 présente des exemples ponctuels correspondant à un usage « au quotidien » de METAFONT. Pour terminer, un certain nombre d'annexes regroupent toutes les informations pratiques : modes existants, tables de conversion, encodages, variables internes, adresses où se procurer METAFONT, etc.

Comment lire ce livre ? Le lecteur pressé de commencer et qui dispose d'une installation de METAFONT en état de marche dont il connaît le principe de fonctionnement, pourra aller directement lire le chapitre 4 et enchaîner avec le chapitre 5. Si METAFONT n'est pas encore installé sur votre ordinateur ou bien si vous n'en connaissez pas encore le principe (production d'un fichier *gf*, converti ensuite en fichier *dvi*, etc.), il faudra commencer par lire le chapitre 2. Enfin, le lecteur qui souhaite avoir une vue synthétique du langage METAFONT et se faire une idée d'ensemble du corpus d'instruction qui est à sa disposition trouvera tout cela dans le chapitre 3 qui servira par la suite de référence pour retrouver rapidement une macro.

L'apprentissage de METAFONT est beaucoup plus facile que celui de TEX et c'est un investissement qui décuple les possibilités de TEX lui-même.

Bernard Desgrauves

Paris, mai-septembre 1998.

Chapitre 1

Introduction à METAFONT

1.1 La typographie numérique

Avant l'avènement de l'ère informatique, la typographie et les techniques d'impression en général ont connu plusieurs révolutions, toutes liées d'une manière ou d'une autre à des évolutions matérielles : utilisation de nouveaux supports (pierre, papier, bois, calcaire, cuivre, etc.), de nouveaux outils pour la gravure (plumes, stylos, poinçons, laser), de nouvelles encres et de nouveaux produits chimiques (acides, résines, vernis), mise en application des propriétés physiques de certains corps (héliogravure).

La typographie est donc une technique, mais c'est aussi un art : celui de dessiner les caractères, celui de les juxtaposer et de les disposer sur une surface. Cet art a connu lui aussi ses révolutions, ou peut-être devrait-on dire dans ce cas ses évolutions. Les quatre derniers siècles ont vu apparaître successivement de nouvelles familles de caractères typographiques depuis les *grecs du Roi* conçus par Claude Garamond (1499-1561) pour François Ier ou les *romains du Roi* commandités par Louis XIV pour l'Imprimerie nationale.

La typographie allie donc deux aspects — technique et esthétique — qui correspondent à des métiers différents : le fondeur et le graveur. Garamond ou bien encore la dynastie des Didot ont montré qu'on pouvait même concilier les deux, ce qui témoigne de leur étroite imbrication.

L'informatique a réalisé une double révolution qui a frappé simultanément ces deux aspects de la typographie. Elle a profondément modifié la chaîne de production d'un document imprimé et la conception des caractères. Les contraintes liées à la fonte de caractères de plomb à la durée de vie éphémère ont totalement disparu : les caractères sont numérisés et transmis sous forme numérique directement aux photocomposeuses qui fabriquent les plaques pour la reproduction en grand nombre, ou aux imprimantes personnelles pour la production à l'unité.

On peut distinguer, du point de vue qui nous intéresse ici, c'est-à-dire celui de la

création des caractères, deux phases : dans un premier temps, les caractères ont été conçus comme des images numérisées par avance (*bitmaps*) et sur lesquelles l'utilisateur n'avait aucune possibilité d'intervention. Dans un deuxième temps, tirant profit des progrès accomplis dans le domaine du dessin informatique et de l'augmentation considérable de la puissance de calcul des machines, les images ont été remplacées par des instructions décrivant l'image en question. Pour prendre un exemple simple (et extrêmement schématique !), un O a d'abord été une image représentant un rond et conservée en mémoire sous forme d'un tableau de points élémentaires (pixels) blancs ou noirs alors que, dorénavant, un O doit être conçu comme une instruction qui dit (approximativement) « *tracer un rond* ».

Cette évolution a consacré le retour de la calligraphie. En effet, le processus de l'écriture manuscrite consiste à redessiner un rond chaque fois que la lettre O se présente avec la possibilité à tout moment de modifier notre façon de le faire. Le comportement de l'ordinateur est intermédiaire entre l'approche figée des fichiers d'images numérisées et l'éternel recommencement de l'écriture manuscrite et, en ce sens, il est optimal. Modifier la forme d'un caractère signifie désormais intervenir sur les instructions de dessin qui le décrivent : ainsi pour obtenir un *O* en italique, l'instruction « *tracer un rond* » sera modifiée (toujours schématiquement) en « *tracer un rond et l'incliner vers la droite* ». Tant qu'on ne demande pas de modification, le programme qui assure la composition d'un texte ou d'un ouvrage utilise les mêmes images pour chaque caractère. En revanche, si une modification (de corps, de style, de graisse, etc.) est demandée, l'ordinateur est capable de recalculer tous les caractères en fonction des nouveaux critères imposés et de produire une nouvelle série d'images de type *bitmap* : cette étape de recalculation correspond à ce qui était anciennement la fonte de nouveaux caractères. Cette opération requiert désormais quelques secondes, voire même quelques fractions de seconde : plus besoin de créer des moules dans lesquels on versait le plomb fondu pour produire des milliers de poinçons répartis par la suite dans des casses avant d'être assemblés sur des plaques. La fonte précisément est l'affaire de METAFONT et la composition celle de T_EX.

1.2 Historique

De même que gravure, fonte et composition sont quasiment inséparables, METAFONT et T_EX sont des programmes intimement liés. Leur conception remonte à 1977 pour s'achever en 1984. Ils sont l'œuvre d'un seul homme, le mathématicien et informaticien Donald Knuth, qui a ainsi créé un outil dont la hardiesse de conception et la perfection n'ont encore jamais été égalées. Dès le départ, il s'était fixé comme objectif la plus haute qualité typographique (en particulier pour la composition de textes mathématiques). Mais les mérites de ces deux programmes ne reposent pas uniquement sur le degré de précision et de qualité (reconnus par tous les typographes) des documents produits. Il faut mentionner quelques caractéristiques qui ont également été déterminantes dans le succès rencontré auprès des utilisateurs :

- ces programmes sont indépendants des plates-formes de travail : leur source en Pascal est publié (ce qui est déjà un fait rarissime et particulièrement remar-

uable). C'est ce qui a permis d'adapter ce programme aux différents systèmes d'exploitation et plates-formes existants ;

- les programmes sont des compilateurs : ils prennent un fichier source et l'exécutent pour produire le résultat escompté. Les fichiers sources sont rédigés en format texte ce qui assure une échangeabilité parfaite : des utilisateurs travaillant sur des plates-formes différentes peuvent s'échanger des fichiers sources et les compiler, chacun sur son système ;
- les fichiers sources sont donc conçus comme des ensembles de tâches à exécuter qui sont décrites par des commandes ;
- ces programmes sont aussi des langages avec toute la richesse et la puissance que l'informatique peut apporter en la matière : pour rédiger les commandes que l'on veut faire exécuter, on a à sa disposition un langage et des possibilités syntaxiques adaptées à l'objectif. METAFONT possède des instructions pour le dessin, pour la manipulation des images, pour le choix des stylos ; TeX possède des instructions pour l'assemblage des mots, la conception d'ouvrages d'envergure, la création d'index, de tables des matières, etc. ;
- en tant que langages, ils offrent des possibilités d'extension qui permettent à chacun d'écrire à son tour des commandes (on dit aussi des macros) adaptées à tel ou tel usage particulier : c'est la notion de *base* (pour METAFONT) ou de *format* (pour TeX) qui sont des ensembles de commandes d'ordre plus ou moins général dont on dispose pour la création de caractères ou la composition de textes ;
- enfin — considération loin d'être négligeable — ces programmes sont mis gratuitement à la disposition du public : ils sont en particulier distribués par voie électronique (voir l'annexe H) ou par le biais de groupes d'utilisateurs.

Cette architecture ouverte a suscité l'apparition de formats qui ont considérablement contribué au succès de TeX. Le plus connu d'entre eux est L^AT_EX dont la version la plus récente, L^AT_EX 2_E, a permis de réaliser le présent ouvrage : L^AT_EX a lui aussi largement contribué à la propagation de TeX car il offre une étonnante flexibilité, par le mécanisme des *packages*, et permet l'émergence d'innombrables apports contributifs. Il faut reconnaître que le succès fulgurant de TeX a un peu relégué METAFONT dans l'ombre : toujours présent et indispensable, METAFONT est cependant souvent devenu un outil annexe travaillant en quelque sorte en coulisses pour assurer la fabrication — commandée par TeX — de quelques formes dérivées des polices disponibles. Pourtant, METAFONT offre les mêmes possibilités extraordinaires d'extension que TeX et ces deux programmes présentent des analogies de conception que les familiers de TeX reconnaîtront sans peine.

On attend encore l'émergence d'un avatar qui serait à METAFONT ce que L^AT_EX a été à TeX lui-même. Comme on le verra dans les chapitres qui suivent, l'apprentissage de METAFONT est beaucoup plus facile que celui de TeX. Les utilisateurs de TeX

pourront se rendre compte de l'immense richesse ainsi mise à leur disposition pour compléter les potentialités de *TeX* et diversifier les documents produits.

Il est absurde de voir le mal que certains se donnent (et les contorsions auxquelles il faut se livrer, en particulier pour ce qui est des symboles mathématiques) pour utiliser des polices PostScript en liaison avec *TeX* à la place des polices créées par METAFONT, se privant ainsi de toutes les possibilités offertes par METAFONT¹. Il faut aussi ajouter que l'on souhaiterait disposer d'un choix plus large dans les polices disponibles en METAFONT. L'offre n'est sans doute pas encore suffisante mais, là aussi, on attend la multiplication des contributions de la part de la communauté des utilisateurs : il y a encore du travail à accomplir !

1.3 Premiers pas avec METAFONT

La meilleure façon de comprendre ce qui vient d'être dit, et en particulier la notion de caractères définis par des instructions de tracé et non en tant qu'images figées, est de se pencher sur un exemple. Sans chercher pour le moment à comprendre toutes les subtilités de METAFONT, ce premier contact avec le langage devrait permettre de concrétiser les idées de base de la typographie informatique récente. La lettre grecque bêta représentée sur la figure 1.1 est le résultat de l'exécution par METAFONT de la série d'instructions suivante :

```

1      u#:=4/9pt#;
2      define_pixels(u);
3      beginchar(66,13u#,16u#,5u#); "Lettre bêta";
4      x1=2u; x2=x3=3u;
5      bot y1 = -5u; y2=8u; y3= 14u;
6      x4=6.5u; top y4 = h;
7      z5 = (10u,12u);
8      z6 = (7.5u,7.5u); z8=z6;
9      z7 = (4u,7.5u);
10     z9 = (11.5u,2u);
11     z0 = (5u,u);
12     penpos1(2u,20);
13     penpos2(.5u,0);
14     penpos3(u,-45);
15     penpos4(.8u,-90);
16     penpos5(1.5u,-180);
17     penpos6(.4u,150);
18     penpos7(.4u,0);
19     penpos8(.4u,210);
20     penpos9(1.5u,-180);
21     penpos0(.3u,20);
22     pickup pencircle;
23     penstroke z1e..z2e..z3e..z4e..z5e..z6e..(up)z7e..z8e..z9e..(up)z0e;

```

¹ Ce qui n'ôte rien aux mérites évidents de PostScript !

```

24 labels(range 1 thru 9);
25 endchar;
26 end

```

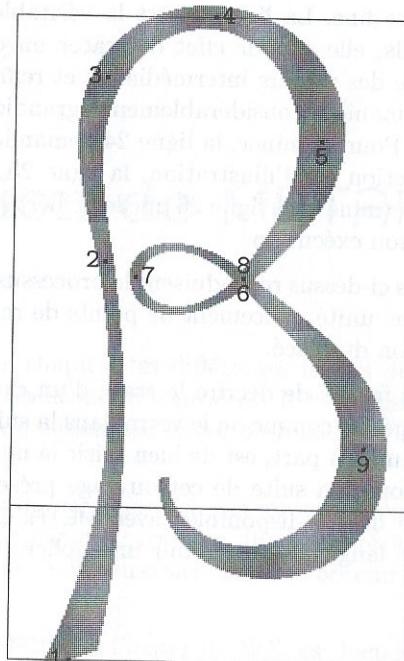


FIG. 1.1 – La lettre grecque β

Ces instructions indiquent au programme comment il doit procéder pour tracer le caractère bêta : il précise quel type de stylo doit être utilisé, comment il doit être incliné et appuyé, par quels points il passe. Voyons cela plus en détail² : la première ligne définit une unité de mesure dans laquelle seront exprimées toutes les grandeurs (coordonnées des points, tailles de stylos, etc.). La ligne 2 commande à METAFONT de faire ses calculs en nombres (entiers) de pixels et non en grandeurs géométriques. La ligne 3 indique non seulement que l'on va commencer le tracé d'un nouveau caractère, mais elle en précise le numéro de code (ici 66) ainsi que trois dimensions caractéristiques — largeur, hauteur et profondeur — qui se trouvent matérialisées sur la figure 1.1 par le cadre et la ligne horizontale qui est la ligne de base du caractère. Le caractère bêta aura donc une largeur de 13 unités, une hauteur de 16 unités au dessus de la ligne de base et il descendra de 5 unités sous cette ligne de base. Les lignes numérotées de 4 à

2. Les lignes sont ici numérotées uniquement pour la commodité de l'exposé. Les instructions d'un fichier programme de METAFONT ne sont en réalité jamais numérotées.

11 indiquent le placement des points z_1 à z_9 qui servent de guides au tracé du stylo. Les instructions 12 à 21 apportent une information supplémentaire : elles indiquent en chacun des points de construction la largeur de la plume du stylo et son inclinaison par rapport à l'horizontale. C'est ce qui permet d'obtenir les pleins et les déliés. La ligne 22 donne alors l'instruction de s'emparer d'un stylo particulier : en l'occurrence un stylo à pointe circulaire fine. La ligne 23 est la véritable instruction de dessin : sans entrer dans les détails, elle a pour effet de tracer un contour qui respecte les valeurs imposées, interpole des valeurs intermédiaires et remplit le tout, aboutissant au résultat représenté de manière considérablement agrandie sur la figure 1.1 et en définitive, au caractère β . Pour terminer, la ligne 24 demande à METAFONT de montrer les points de construction sur l'illustration, la ligne 25 indique que le tracé du caractère est maintenant terminé et la ligne 26 que le fichier est terminé : l'interpréteur METAFONT peut arrêter son exécution.

En résumé, les instructions ci-dessus reproduisent le processus naturel de tracé manuel d'un caractère : choix d'une unité, placement de points de repère, prise en main d'un stylo approprié et exécution du tracé.

Il existe bien sûr d'autres façons de décrire le tracé d'un caractère, d'un symbole et d'un dessin de manière générale comme on le verra dans la suite de cet ouvrage : ce qui importe ici, toute rigueur mise à part, est de bien saisir la notion de caractères définis par un corpus d'instructions. La suite de cet ouvrage présentera systématiquement toutes les commandes de dessin disponibles avec METAFONT ainsi que toutes les possibilités offertes par ce langage pour définir une police complète de caractères et de symboles.

Il existe plusieurs manières d'obtenir ce programme. La première est de le trouver dans une distribution Linux ou Mac OS X. Il existe aussi des sites Web qui proposent des versions de METAFONT pour diverses plates-formes. La deuxième manière consiste à télécharger la version source de METAFONT et à la compiler soi-même. C'est ce que nous allons faire dans ce chapitre.

Chapitre 2

Installation de METAFONT

Nous allons voir dans ce chapitre les différentes façons de faire fonctionner METAFONT et d'exploiter les résultats qu'il produit. Il y a plusieurs niveaux d'utilisation de METAFONT qui correspondent au degré de qualité et au type de résultat qu'on attend de lui.

La création d'une police de caractères se fait progressivement : METAFONT permet d'obtenir des épreuves de chaque caractère afin d'apprecier les formes et les tracés et de faire les corrections indispensables avant d'obtenir des sorties définitives et directement exploitables.

La grande force de METAFONT, à l'instar de *TEX*, est bien sûr de pouvoir fonctionner sur toutes sortes de plates-formes. Les polices sont décrites dans des fichiers d'instructions au format texte : pour obtenir une police, il suffit d'exécuter METAFONT sur ce fichier d'instructions. Les créateurs peuvent échanger leurs fichiers sources sans se soucier du système d'exploitation. Ces fichiers étant codés sur 7 bits, ils peuvent en particulier voyager sans problème par voie électronique.

2.1 Les éléments nécessaires

De quoi a-t-on besoin pour faire fonctionner METAFONT ? Les distributions de METAFONT comportent toutes plus ou moins les mêmes éléments, quels que soient la plate-forme et le système d'exploitation sur lesquels ce programme a été porté¹. À strictement parler, il faut au minimum être en possession de deux fichiers : le programme lui-même et les macros décrites dans la chapitre 3. Celles-ci sont regroupées dans un fichier intitulé `plain.mf`². C'est à partir de ce fichier qu'est fabriqué le format *plain* comme expliqué au paragraphe 2.3. Ces deux fichiers sont tout ce dont on a réellement

1. Voir l'appendice H pour savoir où se procurer METAFONT.

2. Il est instructif d'aller lire de temps en temps comment et de quoi il est fait. Ce fichier se trouve

besoin pour commencer à travailler.

En réalité, une distribution ordinaire de METAFONT comporte également d'autres fichiers : ceux-ci sont répartis entre deux répertoires. Le premier s'intitule en général MFBASES et le second MFINPUTS. Il faut également avoir les programmes gftodvi et gftopk.

- Dans le répertoire MFBASES se trouve le fichier `plain.mf` ; on y trouve souvent aussi un fichier `cmbase` qui contient des macros supplémentaires nécessaires pour fabriquer les polices de la famille Computer Roman³ et un fichier `logobase` qui regroupe les macros utilisées pour la fabrication de la police *logo* (celle qui comporte les sept lettres permettant d'écrire le mot METAFONT). C'est dans ce répertoire que METAFONT recherche le fichier `plain.base`, résultat de la compilation de `plain.mf`.
- Dans le répertoire MFINPUTS on place habituellement tous les fichiers `*.mf` dits « fichiers sources ». Ce sont les fichiers d'instructions en langage METAFONT évoqués plus haut. Ils portent tous l'extension « `.mf` ». On y trouve normalement les fichiers sources de la famille Computer Roman. Il peut s'y trouver aussi ceux de la police *logo* et les fichiers sources des polices *lasy* (symboles complémentaires pour L^AT_EX). C'est dans ce répertoire (et dans des sous-répertoires) que l'on placera les fichiers sources que l'on télécharge sur des sites ftp — en particulier sur les CTAN (Comprehensive T_EX Archive Network) — ainsi que ceux que l'on aura écrits soi-même bien évidemment.

Deux séries de fichiers se révèlent très utiles pour faciliter le travail : les fichiers des polices *gray* qui servent à visualiser en grand les caractères que l'on a créés (*Cf.* paragraphe 2.2.6) et les fichiers *tests* pour étudier à l'écran une lettre en cours de création et contrôler les modifications qu'on y apporte (*Cf.* paragraphe 2.2.5).

Signalons encore aux lecteurs du METAFONTbook de D. Knuth que les fichiers de démonstration `io.mf` et `expr.mf` utilisés dans les chapitres 5 et 8 sont en général fournis dans les distributions de METAFONT et qu'il est donc inutile de les retaper.

Si METAFONT n'est pas déjà installé et en état de marche, il faut impérativement passer par une première étape (très simple mais incontournable) qui consiste à compiler la base « plain ». Tout cela est expliqué au paragraphe 2.3.

2.2 Principe de fonctionnement

Nous présentons ici les différentes étapes dans la création de symboles ou d'une police complète de caractères avec METAFONT. Au moyen des macros présentées au chapitre 3, on aura créé (avec un éditeur de texte) un fichier portant l'extension `.mf` :

³ C'est la famille de polices de caractères créée par D. Knuth et bien connue des utilisateurs de T_EX.

appelons-le par exemple `zozo.mf`. C'est un fichier au format texte comportant les instructions nécessaires à METAFONT pour réaliser le travail souhaité.

On fait alors interpréter ce fichier par METAFONT. Plusieurs cas de figure se présentent :

- dans un premier temps, on fait des essais : au vu des résultats, on sera immédiatement amené à apporter des corrections. Cette première phase se déroule en mode *proof*. Cela signifie qu'il est inutile pour le moment de produire les fichiers `pk` et `tfm` qui seront exploités par la suite par TeX. En mode *proof*, la résolution est de 36 pixels par point d'imprimeur ;
- une fois satisfait du résultat obtenu, on demande à METAFONT de produire un résultat définitif. Il faut alors travailler dans un mode qui corresponde au périphérique sur lequel on fera les impressions. Les modes sont tous répertoriés à l'annexe G.

La puissance de METAFONT réside (outre dans l'efficacité des outils qu'il fournit) dans le fait qu'il est indépendant des périphériques de sortie. Un fichier contient non pas un dessin mais des *instructions* de dessin. C'est le même fichier source qui sera utilisé sur toutes les installations ; il est indépendant du matériel utilisé et en particulier de la résolution. C'est sur chaque installation particulière que METAFONT calculera des fichiers images à la bonne résolution : les informations qui lui sont nécessaires — pour s'adapter à un périphérique particulier — sont définies par le mode (résolution et variables de correction : `blacker`, `fillin`, `o_correction`...).

Nous allons supposer dans la suite de ce chapitre que l'on dispose de deux installations, l'une à basse résolution (300 dpi par exemple) et l'autre à haute résolution (1200 dpi par exemple). Ces deux imprimantes ont des modes que nous baptiserons respectivement *rustaud* et *finaud*. Cela signifie qu'il doit y avoir dans le fichier `modes.mf` les définitions des modes⁴ *rustaud* et *finaud*. Voici, à titre d'exemple, à quoi elles pourraient ressembler⁵ :

```
mode_def rustaud =
  mode_param (pixels_per_inch, 300);
  mode_param (blacker, 0);
  mode_param (fillin, .2);
  mode_param (o_correction, .6);
enddef;

mode_def finaud =
  mode_param (pixels_per_inch, 1200);
  mode_param (blacker, .2);
```

4. Ou bien qu'il faut les y ajouter. Plus de deux cent cinquante modèles sont néanmoins répertoriés : il est bien rare de ne pas trouver quelque chose d'approchant quitte par la suite à ajuster les variables `blacker`, `fillin`, `o_correction`...

5. Les valeurs données ici sont arbitraires puisque *rustaud* et *finaud* n'existent pas !

```
mode_param (fillin, .1);
mode_param (o_correction, .85);
enddef;
```

2.2.1 Premières épreuves

En mode *proof*, la syntaxe pour que METAFONT traite le fichier `zozo.mf` est simplement :

```
mf zozo
```

On ne précise pas que le mode est *proof*, car c'est justement le mode adopté par défaut en dehors de toute précision. METAFONT produit alors un fichier `zozo.2602gf.gf` qui signifie *generic font* et le chiffre de 2602 désigne la résolution. On a vu en effet que le mode *proof* correspond à 36 pixels par point d'imprimeur. Or il y a 72,27 points d'imprimeur dans un pouce, d'où une résolution de :

$$36 \times 72,27 = 2601,72 \text{ dpi} \approx 2602 \text{ dpi}^6.$$

METAFONT produit simultanément un fichier `zozo.log` qui stocke toutes les informations relatives au déroulement de son travail : c'est comme un carnet de bord que l'on peut consulter en cas de problème.

En utilisant le programme `gftodvi`, on convertit le fichier `zozo.2602gf` en `zozo.dvi` que l'on pourra visualiser grâce à un outil de pré-visualisation, un *previewer*, tel que `dvipreview` ou `xpdf`. Le fichier `zozo.dvi` sera imprimé comme n'importe quel fichier `dvi` produit par `TEX`. Pour avoir une pré-visualisation et une impression satisfaisantes, il faut avoir des polices *gray* convenables : pour plus de détails, lire le paragraphe 2.2.6.

L'intérêt de cette démarche est d'avoir une sortie imprimée de chaque caractère dans une taille très agrandie (environ 300 fois) afin de mieux juger des détails : ce sont des épreuves à examiner de près mais aussi à accrocher au mur et à regarder de loin !

Les épreuves obtenues en mode *proof* comportent un caractère par page. En haut à gauche de chaque page on trouve une ligne d'informations comportant en particulier la date et l'heure de production du fichier `gf`, le numéro de la page et éventuellement le code du caractère (s'il y avait une instruction `beginchar` dans le fichier) ainsi que le titre qui aura pu être donné à ce caractère. Par exemple, si le fichier comporte la description d'une lettre B majuscule commençant par :

```
beginchar({"B", larg#, haut#, prof#}); "To be or not to be";
```

l'épreuve obtenue en mode *proof* dira en préambule :

```
METAFONT output 1998.07.08:1857 Page 1
Character 66 "To be or not to be"
```

⁶ Cf. annexe B : les tables de conversion, `dpi` signifie « points par pouce » (*dots per inch*).

2.2.2 Épreuves définitives

Si l'on est satisfait des résultats obtenus en mode *proof*, on peut passer à la phase de production définitive de la police de caractères. À partir de ce moment on doit tenir compte du matériel utilisé, et la compilation par METAFONT se fait pour une résolution particulière. On invoquera la syntaxe suivante :

```
mf \mode=rustaud; input zozo.mf
```

ou bien

```
mf \mode=finaud; input zozo.mf
```

suivant que l'on travaille avec l'imprimante de mode *rustaud* à 300 dpi ou l'imprimante de mode *finaud* à 1200 dpi. METAFONT produit alors d'une part un fichier *zozo.tfm* (*TeX font metrics*), d'autre part un fichier *zozo.300gf* dans le cas de *rustaud* ou *zozo.1200gf* dans le cas de *finaud*. Le programme *gftopk* doit alors être utilisé pour les compacter, ce qui produit *zozo.300pk* et *zozo.1200pk* respectivement⁷. C'est le fichier *tfm* qui est attendu par *TeX* ou *LATEX*. Le fichier *pk* sert à visualiser les caractères : à ce moment-là, il n'y a plus d'indépendance vis-à-vis du matériel utilisé et d'ailleurs le fichier *zozo.1200pk* sera environ quatre fois plus volumineux que le fichier *zozo.300pk*.

2.2.3 \relax

Il y a une manière directe d'utiliser METAFONT comme un interpréteur de commandes en ligne. En effet si l'on se contente d'appeler METAFONT par la commande *mf*, celui-ci répond, dans une fenêtre, en manifestant sa présence par une double astérisque : **.

Cette double astérisque signifie que METAFONT attend qu'on lui donne le nom du fichier à traiter. Si, au lieu d'une instruction

input nom de fichier,

on se contente de donner l'instruction *\relax* (avec une barre contre-oblique) METAFONT n'affichera plus qu'une seule astérisque : *.

Il est alors prêt à fonctionner en mode direct. Il attend une instruction ou une série d'instructions qu'il traitera dès qu'elles auront été validées par un retour-chariot avant d'afficher à nouveau l'astérisque simple et d'attendre de nouvelles instructions.

Aucun fichier de sortie n'est constitué dans ce cas à moins que cela ne soit explicitement demandé par une instruction *ad hoc* (*shipout* ou *shipit*). Pour mettre fin à ce mode de fonctionnement, il faut donner l'instruction *end* (sans barre contre-oblique à

⁷. *pk* signifie *packed*, compacté.

la différence de TeX) à la suite des autres instructions éventuelles. Rappelons d'autre part que toutes les instructions doivent être suivies d'un **point-virgule** « ; »⁸.

D'autre part, si l'on souhaite que le travail exécuté par METAFONT à la suite d'une ligne d'instructions soit stocké dans un fichier, il faut ajouter la commande **shipit**. Si aucun nom de fichier n'a encore été donné, METAFONT donne par défaut le nom **mfput**.

Si nous tapons par exemple la ligne suivante :

```
draw (10,10)..(50,20)..(90,10)--(10,10);showit;shipit; end
```

nous verrons à l'écran le dessin correspondant (grâce à la commande *showit*) et nous aurons un fichier **mfput.2602gf** contenant ce dessin comme caractère unique portant le code 0 (cela résulte de la commande *shipit*). Ce fichier pourra être transformé en fichier **mfput.dvi** par le logiciel *gftodvi* afin d'être visualisé par *dvipreview* ou *xdvi* comme nous l'avons déjà vu au paragraphe 2.2.1.

2.2.4 Familles de polices ou « métapolices »

Une police est susceptible évidemment d'être utilisée dans plusieurs corps. Mais un changement de corps ne devrait pas être simplement un redimensionnement proportionnel : pour s'en convaincre, il suffit de voir sur l'exemple suivant une phrase écrite successivement en corps 12 points puis en corps 10 points magnifié de 20% :

La vie est belle.

La vie est belle.

Pour des questions de lisibilité, si le corps diminue, l'espacement entre les lettres est augmenté et l'épaisseur des traits également. Les paramètres entre plusieurs corps d'une même police sont donc modifiés un par un et pas tous dans les mêmes proportions. Néanmoins les instructions pour dessiner chaque caractère restent les mêmes (c'est ce qui fait la cohésion de la famille). On en arrive ainsi à la notion de « métaplace » qui se traduit par une hiérarchie particulière des fichiers METAFONT : on distingue les fichiers de paramètres et les fichiers pilotes. Les fichiers pilotes⁹ contiennent toutes les instructions de dessin, caractère par caractère : **beginchar...endchar**. Ces instructions sont écrites algébriquement en fonction de paramètres dont les valeurs sont fixées dans les fichiers de paramètres. Par exemple, pour la police cmr, il existe un fichier pilote **roman.mf** et pour chaque corps particulier il y aura un fichier de paramètres différent : **cmr8.mf**, **cmr9.mf**, **cmr10.mf**, **cmr12.mf**, etc. Ces derniers fichiers comportent les valeurs des paramètres et une instruction :

```
input roman
```

⁸. Ce point-virgule sera omis dans tout le chapitre 3 par souci de clarté mais ce n'est syntaxiquement pas correct.

⁹. *driver files*.

qui a pour effet de passer les valeurs des paramètres au fichier `roman.mf`¹⁰.

L'erreur très fréquente chez le débutant est de croire qu'on crée la police cmr en exécutant METAFONT sur le fichier roman.mf alors que l'on doit opérer sur le fichier de paramètres correspondant au corps souhaité.

Cette technique est extrêmement puissante puisqu'elle permet des variations infinies sur une même police de caractères en modifiant simplement les paramètres dans un nouveau fichier de paramètres. Prenons un exemple fantaisiste pour s'en convaincre pleinement : nous avons modifié les valeurs du fichier paramètre de `cmr10.mf` (voir tableau 2.1) et l'avons rebaptisé `cmrfun10.mf`. Voici les caractères qui sont alors obtenus :

a b e d e f g h i j k l m n o p q r s t u v w x y z
 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
 0 1 2 3 4 5 6 7 8 9

Paramètre	<i>cmr10</i>	<i>cmrfun10</i>
body_height	270/36pt	270/18pt
asc_height	250/36pt	250/24pt
cap_height	246/36pt	246/18pt
fig_height	232/36pt	232/18pt
x_height	155/36pt	155/24pt
comma_depth	70/36pt	70/24pt
desc_depth	70/36pt	100/36pt
tiny	8/36pt	15/36pt
stem	25/36pt	32/36pt
flare	33/36pt	45/36pt
dot_size	38/36pt	48/36pt
jut	28/36pt	1pt
beak_jut	10/36pt	18/36pt
o	8/36pt	18/36pt
apex_o	8/36pt	18/36pt
slant	0	-.1
superness	1/sqrt2	.8

TAB. 2.1 – Valeurs comparées des paramètres dans les polices *cmr* et *cmrfun*

10. La réalité est un peu plus complexe puisque, dans le cas de la famille cmr, les caractères sont en fait répartis dans plusieurs sous-fichiers pilotes qui regroupent les caractères par type : les majuscules (*uppercase*), les minuscules (*lowercase*), les chiffres (*digits*), etc. Ces fichiers s'appellent `romanu.mf`, `romani.mf`, `greeku.mf`, `romand.mf`, `romanp.mf`, `romspl.mf`, `romspu.mf`, `punct.mf`, `accent.mf`.

2.2.5 Tester des polices

Le répertoire MFINPUTS contient habituellement des fichiers test. Ce sont des utilitaires destinés à faciliter le travail du métafondeur et lui permettre de préciser les détails de tel ou tel caractère d'une police sans avoir à chaque fois à traiter la police entière avec ses 128 ou 256 caractères. Ces fichiers sont les suivants :

– btest.mf	– mtest.mf	– rtest0.mf	– xtest.mf	– 3test0.mf
– btest0.mf	– mtest0.mf	– stest.mf	– xtest0.mf	– 5test.mf
– ctest.mf	– qtest.mf	– stest0.mf	– ztest.mf	– 5test0.mf
– ftest.mf	– qtest0.mf	– ttest.mf	– ztest0.mf	– 6test.mf
– ftest0.mf	– rtest.mf	– ttest0.mf	– 3test.mf	– 6test0.mf

Le principe de fonctionnement est simple. Les caractères (ou simplement les routines) que l'on souhaite tester devront être isolés dans un fichier que l'on créera et que l'on baptisera `test.mf` puis on lancera METAFONT sur le fichier `rtest.mf` par exemple, qui est destiné à une police de caractères « romains » (ni obliques, ni gras). En écrivant la commande :

`mf rtest`

on obtiendra des épreuves du ou des caractères testés. Le fichier `rtest.mf` est programmé de telle sorte que, s'il y a plusieurs caractères à tester, il s'arrête entre chaque caractère et attend une instruction de la part de l'utilisateur.

Il y a ainsi des fichiers test pour les différentes formes de caractères : `btest.mf` pour les caractères gras, `stest.mf` pour les caractères italiques, `ttest.mf` pour les polices à espaceissement fixe (type Courier ou cmtt), `ctest.mf` pour les caractères sans empattement (famille cmss), `mtest.mf` pour les caractères en mode mathématique (famille cmmi), `ftest.mf` pour la police Funny Roman à inclinaison négative, `qtest.mf` pour la famille cmssqi (inclinée pour les citations), `xtest.mf` pour les symboles mathématiques.

À l'origine, ces fichiers `rtest`, etc., ont été créés pour l'étude de la famille Computer Roman : ils font donc appel à la base `cmbase` et utilisent les paramètres et variables propres de cette famille. Il est possible (et même hautement probable) qu'on doive les adapter à ses propres besoins. Le fichier `ztest.mf` sert précisément à cela : il passe des paramètres nouveaux — les paramètres particuliers de l'utilisateur correspondant aux besoins du moment — et lance le fichier `test.mf`. Ces paramètres auront été réunis dans un fichier que l'on baptisera `z.mf`.

2.2.6 La police *gray*

La police *gray* est indispensable pour le rendu des caractères créés en mode *proof*. Elle utilise des algorithmes astucieux pour faire le remplissage en grisé des caractères (comme dans les exemples que l'on peut voir au long du chapitre 4). Il faut en fait parler non pas de la police *gray* mais des polices *gray*, car le problème est qu'une police *gray* peut ne pas convenir à tous les types de périphériques : soit elle provoque

un gris trop foncé rendant quasiment illisibles les points z_i de construction, soit au contraire elle fait un gris trop clair et donc des contours imprécis, soit encore elle donne un résultat inégal, bariolé ou bigarré.

METAFONT offre la possibilité de fabriquer très simplement sa propre police *gray* ou bien d'ajuster celle qui accompagne habituellement les distributions du programme. Sans entrer complètement dans les détails, une police *gray* comporte essentiellement deux caractères :

- le premier (caractère de code 0) est le point noir «.» qui marque l'emplacement des points z_i que l'on souhaite faire figurer sur les épreuves¹¹ ;
- le deuxième est le caractère «..» (de code 1) qui, démultiplié, fait le remplissage et l'effet de grisé: █;
- tous les caractères suivants (de code 2 à 122) sont des composés du caractère de code 1¹².

La police *gray* fonctionne sur un principe analogue à celui des fichiers de paramètres ce qui permet de les adapter très facilement à une installation particulière, de les « localiser ». Un fichier *gray.mf* fait appel à un fichier local *graylocalfont.mf* (*localfont* est à préciser) dans lequel on peut fixer des paramètres ou des instructions particulières et qui fait à son tour appel au fichier *grayf.mf*, fichier générateur de la police *gray* elle-même.

Supposons que l'on dispose d'une imprimante à 600 dpi dont le mode s'appelle canonex. On devra créer un fichier *grcanonex.mf* contenant des instructions locales. Sous sa forme la plus simple, il contiendra trois lignes :

```
if mode<>canonex: errmessage "Ce fichier est pour canonex seulement"; fi
font_identifier "GRAYCANONEX";
input grayf
```

On verra plus loin quelles autres instructions peuvent éventuellement figurer dans ce fichier. Le fichier *grayf.mf* est celui qui génère effectivement la police *gray*. Il est modifiable lui aussi. L'enchaînement est le suivant : lorsqu'un fichier *zozo.mf* a été interprété par METAFONT en mode *proof*, on obtient un fichier *zozo.2602gf* que gftodvi transforme en *zozo.dvi* (tout cela a été vu au paragraphe 2.2.1). Ce fichier *zozo.dvi* et le programme de pré-visualisation réclament la police *gray*. Si jamais celle-ci n'existe pas déjà, alors le processus habituel se déclenche : appel à METAFONT pour que celui-ci fabrique la police en question (en mode canonex bien sûr dans notre exemple) et donc recherche un fichier *gray.mf*. Ce fichier *gray.mf* contiendra une seule ligne d'instruction :

```
input grcanonex
```

11. Résultats d'une instruction **labels(*i*)**.

12. Cf. le tableau E.12 p. 214.

ce fichier `grcanonex.mf` appelant à son tour `grayf.mf` comme on l'a vu. Il en résultera tout naturellement deux fichiers : le fichier métrique `gray.tfm` et le fichier image `gray.600gf` transformé ensuite en `gray.600pk` par le programme `gftopk`.

Il y a un nombre limité d'instructions qui peuvent être passées en paramètres par le biais du fichier `grcanonex.mf` : **large_pixels**, **pix_picture**, **pix_wd**, **pix_ht**, **rep**, **lightweight**, **dotsize** et **font_identifier**. Voici leur syntaxe et leur signification :

1. Si la variable **large_pixels** est de type booléen, alors seulement quinze caractères seront générés au lieu de 123. Il suffit d'écrire dans le fichier `grcanonex.mf` :

```
boolean large_pixels ;
```

2. Si la variable **pix_picture** est de type « picture », le caractère employé pour le grisé sera le motif défini par **pix_picture**. Sinon un motif par défaut est employé. Les variables **pix_wd** et **pix_ht** représentent la largeur et la hauteur en pixels. Voici un exemple de syntaxe définissant un **pix_picture** particulier :

```
picture pix_picture;
for z=(0,0),(2,0),(0,2),(2,2):
  fill unitsquare shifted z; endfor
pix_picture := currentpicture scaled 2;
pix_wd := pix_ht := 6;
```

3. **rep** est une variable numérique entière. Si une valeur autre que 1 est donnée, le motif par défaut sera agrandi *rep* fois et l'épreuve sera d'autant plus grande¹³ ;
4. Si la variable **lightweight** est de type booléen alors le motif par défaut sera seulement moitié moins noir et l'épreuve sera d'autant plus claire¹³

```
boolean lightweight;
```

5. La valeur de la variable **dotsize** peut, elle aussi, être fixée. Elle représente la taille en pixels du point noir qui marque l'emplacement des z_i ;
6. Pour mémoire, rappelons qu'il faut préciser la variable **font_identifier** comme dans l'exemple ci-dessous ;

```
font_identifier "GRAYCANONEX";
```

2.2.7 La police *black*

Outre la police *gray*, il existe aussi une police *black* qui sert pour les épreuves faites en mode **smoke**. Ce mode est expliqué à la fin de l'annexe G : les épreuves sont un

¹³ Si `pix_picture` est définie explicitement, alors les instructions éventuelles concernant `rep` et `lightweight` seront ignorées.

peu plus petites qu'en mode *proof* et imprimées en noir (voir la lettre R à la page 239 ou la lettre E à la page 149).

Cette police *black* se trouve définie dans le fichier *black.mf* et le processus de fabrication est rigoureusement le même que pour la police *gray*: un fichier *blcanonex* sera créé de la même façon que l'on a créé un fichier *grcanonex*.

2.2.8 Utilisation d'une police avec T_EX ou L_AT_EX

Les polices de caractères créées avec METAFONT sont destinées à être utilisées dans des documents préparés avec T_EX. Les utilisateurs de T_EX connaissent (plus ou moins bien) l'existence de METAFONT à travers les procédures automatisées de fabrication — par des utilitaires tels que *maketexpk* — des polices au format voulu. Rares sont ceux qui en font un usage direct pour faire face à des besoins particuliers en termes de symboles et d'effets typographiques. Ce paragraphe est destiné plus aux utilisateurs de METAFONT qu'à ceux de T_EX ou de L_AT_EX pour lesquels ces mécanismes sont certainement déjà connus.

Format L_AT_EX

La gestion des polices dans L_AT_EX 2_E se fait au moyen du système dit NFSS (*New Font Selection Scheme*). Il y a deux façons de déclarer une police: soit directement dans le document (de préférence dans le préambule), soit dans un fichier séparé qui porte l'extension *.fd*. L'avantage de cette dernière est que le travail se fait une bonne fois pour toutes ; en revanche la syntaxe en est plus délicate. Nous donnons ici le minimum. Il faut d'abord déclarer une nouvelle famille

```
\DeclareFontFamily{encodage}{famille}{}{}
```

puis ensuite déclarer, dans cette famille, les diverses variantes (en fonction des caractéristiques telles que : gras, italique, penché, etc.):

```
\DeclareFontShape{encodage}{famille}{série}{forme}
{définitions de taille}{options}
```

Prenons un exemple concret : supposons qu'une police du nom de *zozo10.mf* ait été créée au moyen de METAFONT. Elle représentera le corps 10 d'une famille qui pourrait s'appeler *zozo*. Supposons enfin qu'elle soit en encodage OT1 comme le sont les polices de la famille Computer Roman¹⁴ et qu'elle corresponde à la forme normale et à la série *medium*. On écrira alors :

```
\DeclareFontFamily{OT1}{zozo}{}{}
\DeclareFontShape{OT1}{zozo}{m}{n}{<10> zozo10}{}
```

¹⁴ Voir la table de la page 203.

Pour utiliser effectivement cette police dans un document, on écrira alors¹⁵:

```
\fontencoding{OT1}\fontfamily{zozo}\fontseries{m}
    \fontshape{n}\fontsize{10}{12}\selectfont
```

Si la police est utilisée parmi d'autres ou bien occasionnellement, on aura intérêt à définir une macro qui lui corresponde — par exemple `\zo` — en posant une définition:

```
\newcommand{\zo}{\fontencoding{OT1}\fontfamily{zozo}\fontseries{m}
    \fontshape{n}\fontsize{10}{12}\selectfont}
```

Les quatre lignes d'instruction ci-dessus figureront en général dans le préambule du fichier `tex`. Mais on peut aussi constituer, comme il a été dit plus haut, un fichier à part qui devra impérativement s'appeler dans notre cas `ot1zozo.fd` et être rangé dans le répertoire destiné aux fichiers `fd` (*font definition*). Ces fichiers sont recherchés en priorité par L^AT_EX lorsque celui-ci rencontre pour la première fois une commande concernant une nouvelle police de caractères : il comporte la description des formes et tailles disponibles dans la famille `zozo`. Voici, par exemple, ce que ce fichier `ot1zozo.fd` pourrait contenir :

```
\ProvidesFile{ot1zozo.fd}
    [1998/07/14 version auteur]
\DeclareFontFamily{OT1}{zozo}{}{%
    \DeclareFontShape{OT1}{zozo}{m}{n}{%
        <->zozo10%
    }{}%
    \DeclareFontShape{OT1}{zozo}{m}{sl}{%
        <->zozosl10%
    }{}%
    \DeclareFontShape{OT1}{zozo}{m}{it}{%
        <->zozoit10%
    }{}%
    \DeclareFontShape{OT1}{zozo}{bx}{n}{%
        <->zozobx10%
    }{}%
% Instructions de substitution :
\DeclareFontShape{OT1}{zozo}{b}{n}{<->ssub * zozo/bx/n}{}%
\DeclareFontShape{OT1}{zozo}{bx}{sc}{<->ssub * zozo/b/sc}{}%
\DeclareFontShape{OT1}{zozo}{bx}{sl}{<->ssub * zozo/b/sl}{}%
\DeclareFontShape{OT1}{zozo}{bx}{it}{<->ssub * zozo/b/it}{}%
\endinput

End of file 'ot1zozo.fd'.
```

¹⁵ Dans cet exemple, nous avons écrit `\fontsize{10}{12}` ce qui signifie qu'on demande un corps

Ce fichier appelle quelques remarques :

- il comporte une première instruction *ProvidesFile* destinée à passer à L^AT_EX des informations relatives à la date, la version et l'auteur ;
- vient ensuite la déclaration *DeclareFontFamily* d'une nouvelle famille de polices ;
- on trouve alors les déclarations concernant la forme. L'exemple ci-dessus suppose que l'on dispose de fichiers **zozo10.mf**, **zozos110.mf**, **zozoit10.mf** et **zozobx10.mf** correspondant respectivement aux formes standard, sl (*slanted*), it (*italic*) et bx (*bold extended*) en corps 10 ;
- <-> signifie que, dans notre exemple, quel que soit le corps demandé il sera engendré à partir du fichier en taille 10 correspondant ;
- enfin les instructions de substitution indiquent à L^AT_EX quelle police utiliser si on lui demande une forme qui n'a pas été définie précédemment.

Il y a beaucoup d'autres cas de figure à envisager et il n'est pas question de résumer ici, en quelques lignes, tout le système NFSS. On devra impérativement consulter un ouvrage sur L^AT_EX 2 _{ε} pour approfondir la question.

Format T_EXplain

T_EX, dans le format « plain », est moins flexible que L^AT_EX 2 _{ε} et offre moins de possibilités en matière de gestion de polices de caractères mais la syntaxe y est plus simple. On définira une macro **\zo** comme on l'a vu pour L^AT_EX 2 _{ε} au paragraphe précédent et la syntaxe, dans ce cas, est :

```
\font\zo=zozo10
```

Si on souhaite un corps différent — 12pt par exemple — et qu'on n'a pas de fichier **zozo12.mf**, la syntaxe sera :

```
\font\zo=zozo10 at 12pt
```

2.2.9 Autres utilisations de METAFONT

Les fonctionnalités offertes par METAFONT en tant que langage de programmation permettent de l'utiliser également à d'autres fins que la seule fabrication de nouvelles polices de caractères. METAFONT peut être utilisé comme solveur d'équations : on se reportera au paragraphe 4.4.2 p. 102 pour des exemples pratiques. METAFONT peut aussi servir de traceur de figures géométriques ou simplement d'outil de dessin pour des illustrations. Les courbes tracées par METAFONT sont exclusivement des courbes de Bézier reposant sur la théorie des polynômes de Bernstein expliquée à l'annexe A. Cela peut paraître limitatif mais ces courbes permettent d'obtenir d'excellentes approximations de courbes plus complexes. Enfin METAFONT peut tout simplement

être utilisé pour accomplir des calculs, bien qu'en la matière on risque rapidement d'excéder ses capacités (voir néanmoins le paragraphe 4.4.4 p. 109).

2.3 Constitution du format *plain*

2.3.1 Les bases de METAFONT

Si METAFONT est installé pour la première fois, il faut impérativement passer par cette phase initiale de compilation du format, faute de quoi il sera incapable de comprendre toutes les macros décrites au chapitre 3. Il est très facile de vérifier si cette initialisation a été effectuée ou non : si c'est le cas, il doit y avoir dans le répertoire MFBASES un fichier intitulé *plain.base* ou parfois *mf.base*¹⁶. Les fichiers **.base* ont été compilés afin d'être lus à grande vitesse par le programme qui en charge toutes les données en mémoire. Le processus est analogue à celui de la création des formats avec TEX.

2.3.2 Le fichier *local.mf*

Avant de fabriquer le format, il faut aussi « localiser » son installation, ce qui signifie qu'il est possible de fournir quelques caractéristiques supplémentaires pour optimiser le fonctionnement de METAFONT. On a vu que le fichier *modes.mf* contenait les définitions de très nombreux modes : sur une installation particulière la plupart de ces modes seront inutiles, surchargeant la mémoire d'informations relatives à des matériels dont on ne dispose pas. La première chose à faire est d'éditer le fichier *modes.mf*, de le renommer *local.mf* par exemple et d'y faire le ménage afin de ne garder que les modes correspondants au(x) périphérique(s) sur le(s)quel(s) on travaille. Attention : ne supprimez pas les modes *proof*, *smoke* et *nullmode*.

Deux autres paramètres peuvent être fixés dans le fichier *local.mf* :

1. On recherchera dans le fichier une ligne du type *localfont:=quelque chose*. La variable **localfont** se verra assigner le nom du mode préférentiel (s'il y a plusieurs périphériques sur l'installation, ce sera le mode de celui qu'on utilise le plus souvent). Si on utilise une imprimante à 600 dpi dont le mode s'appelle *canonex*, on déclarera :

```
localfont:=canonex;
```

Cette procédure fait qu'un utilisateur extérieur qui ne connaît pas forcément le nom du mode de l'imprimante pourra tranquillement faire tourner ses fichiers en tapant simplement :

```
mf \mode=localfont; input zozo.mf
```

¹⁶ Pour les systèmes qui tronquent les extensions des noms de fichier au-delà de trois lettres, ce

2. On peut aussi dans le fichier `local.mf` préciser le nombre de lignes et de colonnes de la fenêtre dans laquelle METAFONT montre les caractères produits. Il faut rechercher dans le fichier `local.mf` les variables `screen_rows` et `screen_cols`. Par défaut, leurs valeurs sont 400 et 500 respectivement. On peut les modifier à volonté.

2.3.3 Compilation du format *plain*

La procédure est la suivante : il faut lancer METAFONT non en appelant `mf`, mais en invoquant `inimf`. La base de macros s'appelle en principe `plain.mf`. On tapera la ligne d'instructions suivante :

```
inimf plain; input local; dump
```

ou bien, si on n'a pas constitué de fichier `local.mf` :

```
inimf plain; input modes; dump
```

ou bien encore, si on ne veut pas incorporer les modes dans la base¹⁷ :

```
inimf plain; dump
```

METAFONT constitue alors un fichier `plain.base` que l'on devra déplacer dans le répertoire MFBASES.

2.3.4 Utilisation d'autres bases

Les procédures définies dans les paragraphes précédents peuvent évidemment être appliquées à d'autres bases telles que `cmbase.mf` qui contient des macros particulières pour les polices Computer Roman. Néanmoins, il est fréquent de ne pas compiler les autres bases éventuelles car presque toutes les bases utilisent les macros du format *plain*, autrement dit sont bâties au dessus de ce format. Si une base particulière est nécessaire pour un fichier, elle sera en général plutôt appelée dans le préambule du fichier par une ligne d'instruction du type :

```
if unknown cmbase: input cmbase fi
```

Il y a une syntaxe qui permet de demander directement à METAFONT de charger une base supplémentaire en mémoire. Supposons que l'on ait fabriqué une base appelée `lulu.base`, l'instruction pour la charger est :

```
mf &lulu
```

¹⁷. Mais on aurait bien tort.

ou `mf &lulu` sur les systèmes où le caractère & pourrait avoir une autre signification. Pour le cas où METAFONT serait déjà lancé et attendrait une instruction en affichant une double astérisque **, tapez simplement :

`&lulu` et appuyez sur la touche Entrée

puis validez par un retour-chariot. On aura alors à sa disposition à la fois les macros de la base *plain* et celles de la base *lulu*.

Il existe des méthodes pour faire fonctionner METAFONT avec d'autres bases que l'origine *plain*. Ces méthodes sont détaillées dans les sections suivantes. La méthode la plus simple consiste à écrire une macro qui contient les commandes nécessaires pour faire fonctionner METAFONT avec une autre base. Par exemple, si l'on souhaite utiliser la base *lulu*, il suffit d'écrire une macro nommée `lulu.mf` qui contient les commandes nécessaires pour faire fonctionner METAFONT avec la base *lulu*. Puis, lorsque l'on lance METAFONT, il suffit de spécifier l'option `-l lulu` pour que METAFONT utilise la base *lulu*.

2.3.2 La fonction `lulu.mf`

Quand j'aurais fini de lire tout ce que

Il existe une autre méthode pour faire fonctionner METAFONT avec une autre base que l'origine *plain*. Cette méthode consiste à écrire une macro nommée `lulu.mf` qui contient les commandes nécessaires pour faire fonctionner METAFONT avec la base *lulu*. Puis, lorsque l'on lance METAFONT, il suffit de spécifier l'option `-l lulu` pour que METAFONT utilise la base *lulu*.

Il existe une autre méthode pour faire fonctionner METAFONT avec une autre base que l'origine *plain*. Cette méthode consiste à écrire une macro nommée `lulu.mf` qui contient les commandes nécessaires pour faire fonctionner METAFONT avec la base *lulu*. Puis, lorsque l'on lance METAFONT, il suffit de spécifier l'option `-l lulu` pour que METAFONT utilise la base *lulu*.

Il existe une autre méthode pour faire fonctionner METAFONT avec une autre base que l'origine *plain*. Cette méthode consiste à écrire une macro nommée `lulu.mf` qui contient les commandes nécessaires pour faire fonctionner METAFONT avec la base *lulu*. Puis, lorsque l'on lance METAFONT, il suffit de spécifier l'option `-l lulu` pour que METAFONT utilise la base *lulu*.

Il existe une autre méthode pour faire fonctionner METAFONT avec une autre base que l'origine *plain*. Cette méthode consiste à écrire une macro nommée `lulu.mf` qui contient les commandes nécessaires pour faire fonctionner METAFONT avec la base *lulu*. Puis, lorsque l'on lance METAFONT, il suffit de spécifier l'option `-l lulu` pour que METAFONT utilise la base *lulu*.

Il existe une autre méthode pour faire fonctionner METAFONT avec une autre base que l'origine *plain*. Cette méthode consiste à écrire une macro nommée `lulu.mf` qui contient les commandes nécessaires pour faire fonctionner METAFONT avec la base *lulu*. Puis, lorsque l'on lance METAFONT, il suffit de spécifier l'option `-l lulu` pour que METAFONT utilise la base *lulu*.

Il existe une autre méthode pour faire fonctionner METAFONT avec une autre base que l'origine *plain*. Cette méthode consiste à écrire une macro nommée `lulu.mf` qui contient les commandes nécessaires pour faire fonctionner METAFONT avec la base *lulu*. Puis, lorsque l'on lance METAFONT, il suffit de spécifier l'option `-l lulu` pour que METAFONT utilise la base *lulu*.

Il existe une autre méthode pour faire fonctionner METAFONT avec une autre base que l'origine *plain*. Cette méthode consiste à écrire une macro nommée `lulu.mf` qui contient les commandes nécessaires pour faire fonctionner METAFONT avec la base *lulu*. Puis, lorsque l'on lance METAFONT, il suffit de spécifier l'option `-l lulu` pour que METAFONT utilise la base *lulu*.

Il existe une autre méthode pour faire fonctionner METAFONT avec une autre base que l'origine *plain*. Cette méthode consiste à écrire une macro nommée `lulu.mf` qui contient les commandes nécessaires pour faire fonctionner METAFONT avec la base *lulu*. Puis, lorsque l'on lance METAFONT, il suffit de spécifier l'option `-l lulu` pour que METAFONT utilise la base *lulu*.

Il existe de nombreux outils permettant d'élaborer des polices de caractères et de faire évoluer les existantes. METAFONT offre une base de macros qui facilite l'écriture de ces outils. Ces macros sont regroupées dans un seul fichier et peuvent être utilisées pour écrire des programmes qui décrivent des polices de caractères ou pour écrire des programmes qui décrivent des polices de caractères.

Chapitre 3

Les macros METAFONT

Une base pour METAFONT est un ensemble de macros écrites au moyen des facilités de programmation offertes par METAFONT et regroupées dans un même fichier. L'usager n'a pas à connaître les commandes de bas niveau qui sont mises en œuvre mais dispose ainsi d'instructions prédéfinies.

METAFONT est fourni avec une base dite *plain* regroupant une grande variété d'outils destinés à l'élaboration et la construction de polices de caractères. Cette base est habituellement incorporée au format (voir le paragraphe 2.3). On peut par ailleurs constituer ses propres bases pour compléter (ou remplacer) les macros de la base *plain*. Par raccourci, nous parlerons dorénavant des « macros de METAFONT » pour désigner les macros du format *plain* de METAFONT.

Dans ce chapitre, les outils sont regroupés par type et leur usage ainsi que leur syntaxe sont expliqués un par un. C'est un chapitre de référence destiné au lecteur qui veut rapidement retrouver une macro ou avoir une vue synthétique du format *plain*. Sa lecture peut être remise à plus tard et on peut très bien commencer l'apprentissage de METAFONT en se reportant directement aux chapitres 4 et 5 qui sont constitués d'exemples pratiques.

3.1 Les objets

Il existe huit types de variables reconnus par METAFONT : **boolean**, **numeric**, **pair**, **path**, **pen**, **picture**, **string** et **transform**.

La syntaxe pour déclarer un nouvel objet est par exemple :

```
boolean toto;  
path chem;
```

Cette commande signifie qu'il existera désormais deux variables intitulées respecti-

ment *toto* et *chem* : la première sera de type booléen et la deuxième sera un chemin¹. Tant qu'une valeur ne leur a pas été assignée, METAFONT considère qu'elles sont inconnues : en revanche au moment de leur assigner une valeur, cette valeur devra obligatoirement être du type déclaré. Toutefois, lorsque METAFONT rencontre une variable qu'il ne connaît pas encore et qui n'a pas été déclarée, il la considère par défaut comme étant de type *numeric*.

3.1.1 Opérateurs booléens

Valeurs logiques

- **true**
- **false**
- **known** (*expression*)
- **unknown** (*expression*)
- **cycle** (*expression*)
- **odd** (*numérique*)
- **charexists** (*numérique*)

Les variables de type booléen peuvent prendre ou se voir attribuer les valeurs **true** ou **false**. Les cinq autres commandes — **known**, **unknown**, **cycle**, **odd**, **charexists** — servent respectivement à tester si une expression a été déclarée précédemment ou pas, si un chemin est un cycle, si une variable numérique est impaire et si un caractère particulier existe. Les valeurs renvoyées sont **true** ou **false**.

Symboles relationnels

Pas de surprise avec les symboles *<*, *≤*, *=*, *<>*, *≥*, *>* qui ont la signification attendue : *inférieur strictement*, *inférieur ou égal*, *égal*, *different de*, *supérieur ou égal*, *supérieur strictement*.

Relations logiques

- **not**
- **and**
- **or**

not, **and** et **or** entrent dans les expressions logiques avec la signification évidente de *négation*, *et* et *ou* respectivement.

1. Les instructions sont toujours suivies par un point-virgule. C'est une erreur fréquente que de l'oublier, surtout en fin de ligne! Néanmoins, dans ce chapitre, on omittra le point-virgule puisque les macros y sont présentées hors contexte.

3.1.2 Opérateurs numériques

Voici la liste de toutes les macros de type numérique :

- eps	- good.y	- +, -, *, /
- epsilon	- xpart	- + +, + - +, **
- infinity	- ypart	- mod, div
- sqrt	- xxpart	- dotprod
- sind	- xy part	- max
- cosd	- yxpart	- min
- mlog	- yy part	- abs
- mexp	- ASCII	- length
- floor	- oct	- directiontime
- round	- hex	- solve
- hround	- byte	- turningnumber
- vround	- normaldeviate	- totalweight
- ceiling	- uniformdeviate	- tolerance
- lft	- randomseed	- join_radius
- rt	- whatever	- displaying
- top	- angle	
- bot	- incr	
- good.x	- decr	

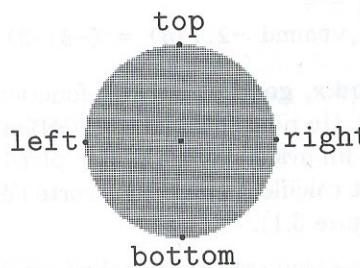


FIG. 3.1 – Étalement d'un point

1. **eps, epsilon et infinity** sont des constantes. **epsilon** et **infinity** sont les quantités extrêmes utilisées par METAFONT : $\text{epsilon} = \frac{1}{65536} = 2^{-16}$ et $\text{infinity} = 4096 - \text{epsilon} = 2^{12} - 2^{-16}$. METAFONT ne connaît pas de nombres inférieurs à epsilon hormis 0. Tout nombre supérieur ou égal à 4096 sera rejeté (pour cause de *arithmetic overflow*) par METAFONT ou bien sera réduit d'office dans les calculs ! METAFONT effectue par ailleurs tous ses calculs en multiples de epsilon .

2. **eps** est un nombre très petit : il vaut 0.00049×2^{-11}

3. **sqrt**, **sind**, **cosd**, **mlog**, **mexp** représentent les fonctions *racine carrée*, *sin*, *cos*, *logarithme* et *exponentielle*. Le *d* dans **sind** et **cosd** est là pour indiquer que tous les angles doivent être exprimés en degrés. D'autre part **mlog** et **mexp** ne sont pas le logarithme et l'exponentielle habituels puisqu'ils sont définis comme suit :

$$\text{mlog } x \stackrel{\text{def}}{=} 256 \ln x$$

$$\text{mexp } x \stackrel{\text{def}}{=} e^{x/256}$$

Voici quelques exemples :

```
sqrt 2 = 1,41422
mlog 2 = 177,44568
mexp -2 = 0,99222
cosd 120 = -0,5
sind 30 = 0,5
```

4. **floor**, **round**, **hround**, **vround**, **ceiling** sont des fonctions qui effectuent des arrondis en valeurs entières. **floor** est la fonction *partie entière*; **round** calcule la valeur entière la plus proche; **hround**, **vround** font de même mais en agissant sur les coordonnées horizontale et verticale respectivement²; enfin **ceiling** arrondit systématiquement à l'entier supérieur. En voici quelques exemples :

```
floor      3.141592 = 3
round      2.718 = 3
floor      -3.141592 = -4
round      -3.141592 = -3
ceiling    3.141592 = 4
(hround -3.141592,vround -2.718) = (-3,-3)
```

5. **lft**, **rt**, **top**, **bot**, **good.x**, **good.y** sont des fonctions très utiles pour définir le placement des points. Un point pour METAFONT a une épaisseur : il se présente comme un disque ou un ovale avec un centre placé aux coordonnées exactes du point. **lft**, **rt**, **top**, **bot** calculent en quelque sorte l'étalement du point autour de son centre (voir la figure 3.1).

6. **good.x** et **good.y** permettent de placer « approximativement » un point :

$$x_1 = \text{good}.x \alpha$$

signifie

$$\text{lft } x_1 = \text{round}(\text{lft } \alpha)$$

Cela fait que x_1 sera approximativement égal à α et que $\text{lft}(x_1)$ sera une valeur entière (ce qui peut permettre d'éviter de petites imperfections au moment de la pixellisation, c'est-à-dire de la transformation des tracés théoriques des courbes en nombres entiers de pixels). Ces questions d'approximation sont très importantes et se trouvent discutées au paragraphe 5.6.

² C'est bien sûr la coordonnée *cartesienne* absente et non sur la paire elle-même.

7. **xpart, ypart, xxpart, xypart, yxpart et yypart** retournent les six valeurs numériques au moyen desquelles METAFONT décrit et effectue toute transformation t . Il faut savoir que les transformations connues de METAFONT sont les transformations usuelles du plan : translations, rotations, symétries, homothéties, etc. autrement dit, du point de vue mathématique, les transformations affines (voir paragraphe 3.1.5 p. 31).

L'instruction

`(x,y) transformed t`

désigne le point de coordonnées

$$(t_x + xt_{xx} + yt_{xy}, t_y + xt_{yx} + yt_{yy}).$$

La transformation t est ainsi décrite par le sextuplet de nombres

$$(t_x, t_y, t_{xx}, t_{xy}, t_{yx}, t_{yy})$$

La commande **xypart t**, par exemple, renvoie la valeur de la quatrième coordonnée t_{xy} du sextuplet.

Signalons au passage que les macros **xpart** et **ypart** ont aussi une autre signification lorsqu'elles sont appliquées à une paire : elles en donnent l'abscisse et l'ordonnée respectivement.

8. **ASCII** s'applique à une chaîne et renvoie le code ASCII du premier caractère de la chaîne³. **oct** et **hex** calculent la valeur décimale d'un nombre écrit dans le système octal ou dans le système hexadécimal :

```
ASCII "ABC" = 65
oct "123" = 83
hex "123" = 291
```

byte a la même signification que **ASCII**:

```
byte "a" = 97
byte "bcd" = 98
```

9. **normaldeviate** et **uniformdeviate** sont des fonctions qui permettent de calculer des nombres au hasard. **uniformdeviate** fournit un nombre distribué selon la loi normale de Gauss $\mathcal{N}(0, 1)$ tandis que **normaldeviate** prend un nombre n comme argument : **normaldeviate n** renvoie un nombre au hasard entre 0 et n . On peut fixer le hasard en imposant, grâce à la macro **randomseed**, la première valeur choisie. En donnant une valeur numérique particulière à **randomseed**, on aura à chaque fois la même suite de nombres aléatoires. On verra au paragraphe 4.4.3 des applications possibles de ces nombres aléatoires pour faire des polices « aléatoires »⁴.

3. C'est un nombre entre 0 et 127. Par convention ASCII " " = -1.

4. Sinon ces nombres aléatoires dépendent de la date et de l'heure puisque par défaut **randomseed** prend la date et l'heure actuelle comme valeur.

10. **whatever** permet de ne pas spécifier une valeur dans une formule ou dans une déclaration : cette valeur sera précisée par la suite ou bien résultera des équations et des contraintes imposées.

Par exemple `z1=whatever [z2,z3]` signifie que le point z_1 est situé quelque part sur la droite passant par z_2 et z_3 . On en verra de nombreuses applications au chapitre 4.

11. **angle** calcule⁵ la direction d'un vecteur donné par rapport à l'axe des abscisses. Par exemple, on aura :

```
angle(1,1) = 45
angle(2,1) = 26,56505
angle(110,50) = 24,44395
angle up = 90
angle down = -90
angle left = 180
```

12. **turningnumber** est une instruction qui s'applique à un chemin cyclique et qui sert à METAFONT à contrôler la forme du chemin. Lorsqu'on parcourt un chemin cyclique, il y a en chaque point de construction un changement de direction que l'on peut mesurer par un angle. En cumulant les valeurs de ces angles on obtient, en revenant au point de départ, une valeur multiple de 360° . C'est ce multiple qui est calculé par *turningnumber* : lorsqu'il est nul, METAFONT se plaint qu'il a affaire à un chemin étrange (*strange path*).

13. **totalweight** s'applique à un objet de type *picture*. C'est la somme de toutes les valeurs de pixels divisée par $65536 = 2^{16}$. C'est, dans la plupart des cas, une valeur numérique inférieure à 0,5 : en divisant par *epsilon*, on retrouve le nombre de pixels de la figure.

À titre d'exemple, un disque de diamètre 100 pixels aura un *totalweight* de 0,11957 correspondant à 7836 pixels.

14. **incr** et **decr** s'appliquent à une variable numérique et l'incrémentent ou la décrémentent d'une unité.

15. Les fonctions **max** et **min** s'appliquent à des valeurs numériques mais aussi à des paires ou à des chaînes (voir paragraphes 3.1.4 et 3.1.8).

16. Outre les opérations habituelles `+, -, *, /`, le format *plain* de METAFONT définit les opérations suivantes :

$$a + + b = \sqrt{a^2 + b^2}$$

$$a + - + b = \sqrt{a^2 - b^2}$$

$$a * * 2 = a^2$$

$$a \bmod b = \text{reste dans la division de } a \text{ par } b$$

5. METAFONT effectue tous les calculs sur les angles en degrés.

$$x \text{ div } y = \text{floor}(x/y)$$

$$(a, b) \cdot \text{dotprod}(c, d) = ac + bd$$

Voici quelques exemples numériques⁶:

```
3++4 = 5
12++5 = 13
5+-+4 = 3
(1.414)**2 = 1,9994
(sqrt 2)**2 = 2
7 mod 4 = 3
7 div 4 = 1
(1,1) dotprod (-2,-3) = -5
```

17. **abs** et **length** ont en réalité la même signification et sont interchangeables mais on utilisera plus volontiers **abs** pour des variables de type *numérique* ou de type *paire* et **length** pour des *chemins* (cela donne le nombre de pas du chemin) ou des *chaînes* (pour obtenir le nombre de caractères qui constituent la chaîne). Voici quelques exemples:

length (1,2)	2,23607
length (3,4)	5
length (-1,1)	1,4142
length "to be or not to be"	18
abs -5	5

18. L'instruction **solve** est extrêmement importante car elle permet de résoudre des systèmes d'équations complexes et de trouver des valeurs approchées dans la recherche de certains points de construction. Le degré de précision des calculs est fixé par la variable **tolerance**. Celle-ci vaut 0,1 par défaut : plus elle est petite et plus les calculs seront précis⁷. On en verra des exemples au paragraphe 4.4.2.
19. **join_radius** et **displaying** sont des constantes assez spécialisées qui interviennent, comme définitions intermédiaires, dans la définition de certaines macros : Cf. The METAFONTbook p. 266 et p. 269. L'utilisateur n'a pas à s'en soucier.
20. Il faut mentionner pour terminer un certain nombre de quantités numériques internes disponibles dans METAFONT : date, heure, variables permettant de suivre à la trace l'exécution des instructions... On en trouvera la liste complète et l'explication à l'annexe C, p. 183.

6. Les valeurs décimales doivent obligatoirement être notées « à l'américaine » avec un point et non une virgule. Nous noterons cependant, tout au long de cet ouvrage, les résultats numériques « à la française » lorsqu'aucune ambiguïté n'est à craindre.

7. Et plus ils seront longs aussi ! Ne jamais fixer une valeur inférieure à *epsilon* sous peine de lancer METAFONT dans des boucles sans fin.

3.1.3 Images

- `currentpicture`
- `nullpicture`
- `blankpicture`
- `unitpixel`

Il y a seulement quatre macros dans cette catégorie mais elles sont de première importance :

1. **currentpicture** est une variable de type *picture* qui contient l'état actuel de ce que METAFONT est en train de confectionner : on peut la passer comme valeur à une autre variable de type *picture*.
2. **nullpicture** est l'image dont tous les pixels sont nuls. **blankpicture** est également une variable qui permet d'afficher une image entièrement blanche. Il y a une nuance entre *nullpicture* et *blankpicture* : *nullpicture* est une expression tandis que *blankpicture* est une variable. La distinction est importante par exemple pour la commande *display* qui attend une variable de type *picture* et non pas une expression : écrire *display blankpicture* est correct ; *display nullpicture* ne l'est pas.
3. **unitsquare** est le carré dont les sommets ont pour coordonnées (0,0), (1,0), (1,1), (0,1).
4. **unitpixel** est la constante de type *picture* contenant ce carré.

3.1.4 Chaînes

- | | | |
|--------------------------|---|---------------------------|
| - " | " | - <code>char</code> |
| - <code>ditto</code> | | - <code>decimal</code> |
| - <code>substring</code> | | - <code>str</code> |
| - & | | - <code>jobname</code> |
| - <code>max</code> | | - <code>readstring</code> |
| - <code>min</code> | | |

Une chaîne est toute série de caractères encadrés par des guillemets : " "

1. **ditto** est une constante de type *chaîne* : c'est le caractère " ⁸ .
2. **substring** (*i*, *j*) **of** *c* renvoie les *j* – *i* caractères de la chaîne *c* à partir du (*i* + 1)-ième caractère jusqu'au *j*-ième.

```
substring (3,13) of "to be or not to be" = "be or not "
```

⁸ Le mot anglais *ditto* est l'équivalent de notre *idem*.

3. **&** sert à concaténer des chaînes entre elles.
4. Les fonctions **max** et **min**, appliquées à des chaînes, se comprendront plus facilement sur des exemples :

```
max("a", "b", "c")           "c"
max("ab", "b")              "b"
max("b", "bc", "bcd", "bcc") "bcd"
```

5. **char** *n* renvoie le caractère de code ASCII *n*. **decimal** *x* représente une valeur numérique *x* sous forme de chaîne. De la même manière, un suffixe de variable (le *i* de *x_i* par exemple) est transformé en chaîne par **str** :

```
char 69 = "E"
decimal 1954 = "1954"
length decimal 1954 = 4
```

6. La variable **jobname** contient le nom du premier fichier appelé par l'instruction **input** (lorsque l'on demande à METAFONT d'exécuter une tâche). Voir au paragraphe 3.4).
S'il n'y a pas d'instruction **input** passée à METAFONT, **jobname** vaut par défaut **mfput** : dans ce cas, une fois son travail effectué, METAFONT produit le résultat dans un fichier intitulé **mfput.2602gf**⁹.
7. Lorsque METAFONT rencontre l'instruction **readstring**, il interrompt son travail et attend qu'une ligne de commandes ou tout simplement une valeur soit saisie au clavier et validée. La valeur de *readstring* pourra être passée à une variable de type *string*. **readstring** contiendra cette ligne. **readstring** est en général associé à une instruction **message** qui explique à l'utilisateur ce qu'on attend de lui (*Cf.* paragraphe 3.2.1).

3.1.5 Transformations

– identity	– slanted	– zscaled
– currenttransform	– scaled	
– inverse	– xscaled	– reflectedabout
– transformed	– yscaled	
– rotated	– shifted	– rotatedaround

Comme on a pu le voir à la page 27, les transformations de METAFONT sont définies au moyen de six nombres. Les formules données en 3.1.8 montrent qu'il s'agit de composées de translations et de transformations linéaires :

⁹ Voir l'explication de ce nombre 2602 au paragraphe 3.2.1.

$$\begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} t_{xx} & t_{xy} \\ t_{yx} & t_{yy} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

1. La syntaxe, quand on a défini une transformation t , est :

(x, y) **transformed** t

2. **identity** est, comme son nom l'indique, la transformation *identité*. Son intérêt est de remettre à zéro si nécessaire des variables du type transformation.

3. Voici les transformations de base prédéfinies ainsi que leur signification :

(x, y) shifted (a, b)	$= (x + a, y + b)$
(x, y) scaleds	$= (sx, sy)$
(x, y) xscaleds	$= (sx, y)$
(x, y) yscaleds	$= (x, sy)$
(x, y) slanteds	$= (x + sy, y)$
(x, y) rotatedθ	$= (x \cos \theta - y \sin \theta, x \sin \theta + y \cos \theta)$
(x, y) zscaled (u, v)	$= (xu - yv, xv + yu)$

L'interprétation géométrique de ces transformations est immédiate :

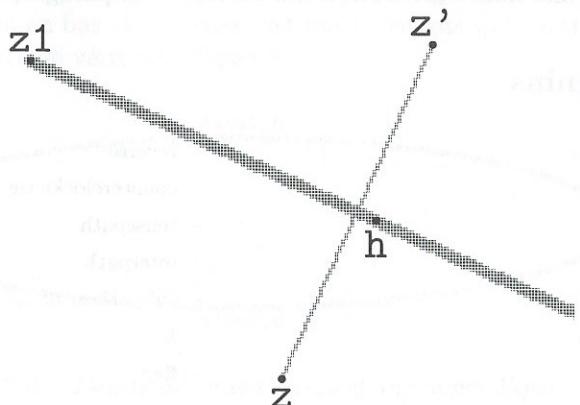
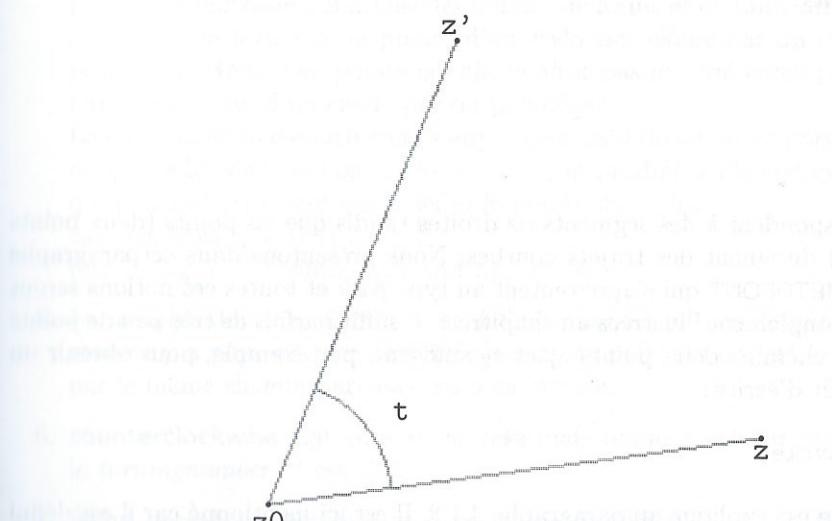
- **shifted** correspond à une translation de vecteur (a, b) ;
- **scaled** correspond à une homothétie de rapport s (à partir de l'origine);
- **xscaled** correspond à une « dilatation » de rapport s pratiquée sur l'abscisse ;
- **yscaled** correspond à une « dilatation » de rapport s pratiquée sur l'ordonnée ;
- **slanted** correspond à une inclinaison ;
- **rotated** correspond à une rotation par rapport à l'origine.

Quant à **zscaled**, il s'agit d'une opération binaire qui correspond à la multiplication des nombres complexes : géométriquement, cela correspond à une rotation ayant pour angle l'argument du nombre complexe $u + iv$ composée avec une homothétie ayant pour rapport le module de ce nombre complexe.

4. **inverse** t définit la transformation qui défait ce que t fait : c'est la fonction réciproque.

5. **reflectedabout** et **rotatedaround** entrent très souvent dans la définition de transformations comme on le verra au chapitre 4 :

- (x, y) **reflectedabout**(z_1, z_2) trouve le point symétrique de (x, y) par rapport à la droite passant par z_1 . Voir la figure 3.2.

FIG. 3.2 – *Instruction reflectedabout*FIG. 3.3 – *Instruction rotatedaround*

6. **currenttransform** est une variable qui affecte toute commande **fill** ou **draw** en les soumettant à une certaine transformation. En général elle est égale à l'identité et son action est alors totalement neutre. En revanche, en lui donnant une autre valeur on affecte toutes les commandes **fill** et **draw** d'un fichier ou d'une police. Par exemple, si **currenttransform** a été préalablement définie comme une rotation d'angle 30°, alors tous les dessins produits par METAFONT dans un fichier particulier seront inclinés de 30° : si le fichier est un fichier de police, tous les caractères produits se trouveront inclinés de 30°... C'est une macro

très puissante dont on trouvera des exemples au paragraphe 5.2.1.

3.1.6 Chemins

- quartercircle
- halfcircle
- fullcircle
- unitsquare
- makepath
- superellipse
- superness
- reverse
- counterclockwise
- tensepath
- interpath
- subpath ... of ...
- &
- flex

Les chemins ont pour but d'indiquer à METAFONT où faire passer le (ou les) stylo(s) qu'il utilise. Ils se présentent comme une suite de points z_i séparés par un des quatre symboles suivants :

--

..
...

Les tirets correspondent à des segments de droites tandis que les points (deux points ou trois points) désignent des trajets courbes. Nous présentons dans ce paragraphe les macros de METAFONT qui s'apparentent au type *path* et toutes ces notions seront développées et amplement illustrées au chapitre 4. Il suffit parfois de très peu de points pour définir un chemin : deux points z_1 et z_2 suffisent, par exemple, pour obtenir un cercle¹⁰ ; il suffit d'écrire :

`z1 .. z2 .. cycle;`

1. **unitsquare** est expliqué au paragraphe 3.1.3. Il est ici mentionné car il est défini comme un chemin qui passe par les points $(0,0)$, $(1,0)$, $(1,1)$, $(0,1)$. C'est une constante de type *path*.
2. **quartercircle**, **halfcircle** et **fullcircle** sont des chemins prédéfinis qui tracent respectivement un quart de cercle, un demi cercle et un cercle entier¹¹. Voir les figures 4.20 à 4.22 au paragraphe 4.1.7.
3. **superellipse** est un chemin de forme ovale (voir figure 3.4). Cette fonction attend cinq arguments. On écrit :

`superellipse(point_d, point_h, point_g, point_b, superness)`

10. On obtient en fait une approximation très acceptable d'un cercle.

11. Il n'y a pas d'instruction *circle*.

où *point_d*, *point_h*, *point_g* et *point_b* sont quatre sommets situés à droite, en haut, à gauche et en bas et *supereness* est une constante qui contrôle de combien l'ovale s'écarte d'une véritable ellipse¹².

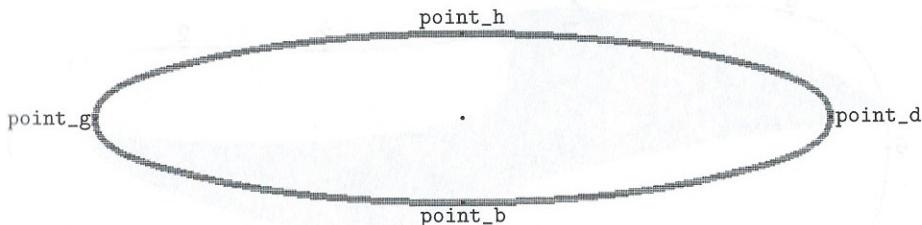


FIG. 3.4 – Points de construction d'une superellipse

4. METAFONT utilise des « stylos » pour exécuter les commandes de dessin. Les divers stylos sont caractérisés par la forme et la taille de leur pointe tout comme pour un calligraphe qui utilise des plumes de forme et de taille différentes. Il faut savoir que la forme de la pointe d'un stylo est définie par un chemin cyclique polygonal. Même une pointe circulaire n'est pas un vrai cercle ; c'est en réalité l'approximation d'un cercle par un polygone !
La commande **makepath** *stylo* s'applique à un stylo de forme polygonaire dont on désignera les sommets par z_0, z_1, \dots, z_{n-1} et produit le *chemin* cyclique suivant, qui pourrait être celui qui a défini la pointe du stylo :


```
z0 .. controls z0 and z1 .. z1 ..  
(etc.) .. zn-1 .. controls zn-1 and z0 .. cycle
```
5. La manière dont un chemin *p* est défini implique un sens de parcours.
reverse *p* a pour effet de renverser le sens de parcours de *p* et le remplace donc par le même chemin parcouru en sens inverse.
6. **clockwise** agit comme **reverse** mais uniquement sur des chemins dont le *turningnumber*¹³ est ≤ 0 .
7. Étant donné un chemin *c* et deux nombres positifs *t*₁ et *t*₂ tels que $t_1 \leq t_2$, **subpath** (*t*₁, *t*₂) **of** *c* est l'ensemble des points au temps *t* de *c* avec *t* variant de *t*₁ à *t*₂. On considère en effet que le chemin est parcouru par un point courant : cela correspond à une définition paramétrique du chemin en fonction d'un paramètre *t* (Cf. annexe A sur les polynômes de Bernstein). Le paramètre *t* varie de 0 à *n* si le chemin a pour longueur *n* : au « temps » *t* = 0 on se trouve au point de départ et au « temps » *t* = *n* au point d'arrivée. La figure 3.5 montre une ellipse sur laquelle on a marqué douze points numérotés de 0 à 11. L'instruction **subpath** permet donc de sélectionner des portions d'un chemin comme on peut le voir

12. Elle varie de 0,5, qui produit un losange, à 1, qui donne un carré. Les valeurs les meilleures sont autour de 0,75. La véritable ellipse correspond à $\sqrt{2}/3 \approx 0,707$. Voir les figures 4.23 à 4.26 au paragraphe 4.1.7.

13. Cf. paragraphe 3.1.2 p. 28.

sur la figure 3.6 où ont été retenues les portions de t_0 à t_4 , de t_5 à t_8 et de t_9 à t_{11} .

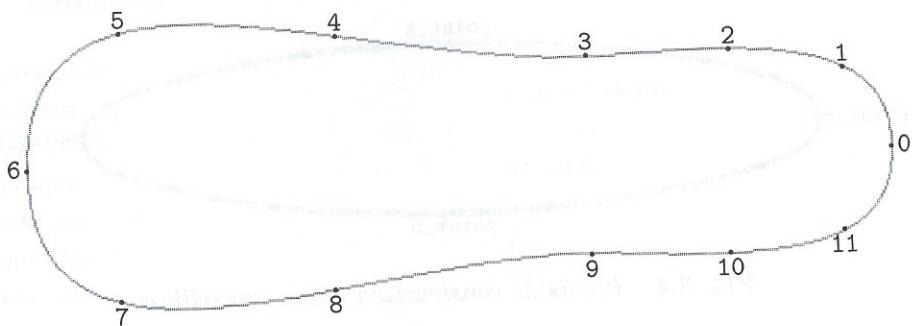


FIG. 3.5 – Douze points intermédiaires sur une ellipse

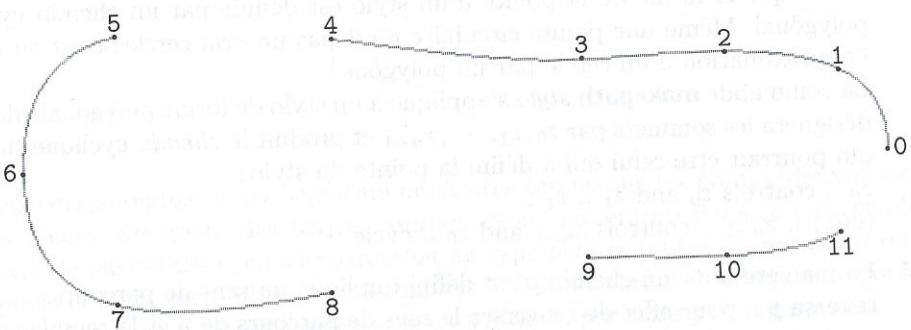


FIG. 3.6 – Instruction subpath

8. **tensepath** s'applique à des chemins ayant des pas de la forme $z_i -- z_{i+1}$ et les transforme en $z_i --- z_{i+1}$, ce qui a pour effet d'arrondir les angles. On en verra des exemples au chapitre 4.
9. **interpath** permet de faire des interpolations entre deux chemins p et q de même longueur n :
 $\text{interpath}(\alpha, p, q)$ crée un chemin dont les points sont :

$\alpha [\text{point } t \text{ of } p, \text{point } t \text{ of } q]$ où t varie de 0 à n

10. **&** sert à concaténer des chemins. On écrira $p \& q$ pour réunir les chemins p et q en un seul chemin.
11. Pour terminer, **flex** est un chemin particulier défini de la manière suivante :

et illustré au chapitre 4, figure 4.27 ou bien sur la figure 3.7.

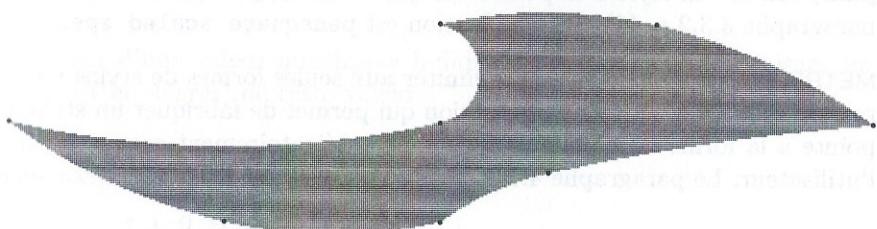


FIG. 3.7 – *Chemin de type flex*

3.1.7 Stylos

- **pencircle**
- **pensquare**
- **penrazor**
- **penspeck**
- **nullpen**
- **currentpen**
- **makepen**

Les stylos sont des instruments dont le choix, on le verra, est déterminant pour la forme des tracés. Il n'y a cependant que sept macros prédéfinies les concernant :

1. **nullpen** est une constante : c'est simplement le point (0,0). C'est un stylo de pointe invisible. Chaque commande **beginchar** qui commence un nouveau caractère initialise *currentpen* en lui donnant la valeur *nullpen*.
2. **currentpen** est la variable qui garde en mémoire les caractéristiques d'un stylo qui vient d'être déclaré par une instruction **pickup** (*Cf.* paragraphe 3.3).
3. **pencircle**, **pensquare**, **penrazor** sont des stylos prédéfinis dont la pointe a une forme particulière : circulaire, carrée ou fine comme un rasoir (de largeur 1 et de hauteur nulle).

On peut modifier la forme de la pointe en modifiant les valeurs horizontale ou verticale. Par exemple¹⁴ :

```
pickup penrazor scaled 10
pickup pencircle scaled 20
pickup pencircle xscaled 40 yscaled 25
pickup pensquare xscaled .7pt yscaled .3pt rotated 60
```

¹⁴ Cf. aussi figures 4.10 p. 75.

Ce dernier stylo est illustré par la figure 3.8 qui représente à la fois la forme de la pointe et le tracé d'un segment exécuté avec ce stylo.

4. **penspeck** est un stylo à la pointe minuscule utilisé par la macro **drawdot** (voir paragraphe 3.3.2 p. 44). Sa signification est **pensquare scaled eps**.
5. METAFONT permet de ne pas se limiter aux seules formes de stylos vues auparavant. **makepen** *c* est une instruction qui permet de fabriquer un stylo dont la pointe a la forme d'un chemin cyclique *c* (obligatoirement convexe) défini par l'utilisateur. Le paragraphe 4.1.6 propose un exemple de pointe hexagonale.



FIG. 3.8 – Un pensquare incliné

3.1.8 Paires

- left	- max	- point
- right	- lft	- precontrol
- up	- rt	- postcontrol
- down	- top	- direction
- origin	- bot	- intersectionpoint
- dir	- good.lft	- intersectiontimes
- unitvector	- good.rt	- penoffset
- round	- good.top	- directionpoint
- min	- good.bot	

Les objets que METAFONT appelle des paires correspondent suivant les cas à des points ou à des vecteurs.

1. **left**, **right**, **up**, **down**, **origin** sont des abréviations pour désigner respectivement les paires $(-1,0)$, $(1,0)$, $(0,1)$, $(0,-1)$, $(0,0)$. Lorsque METAFONT les rencontre, il les traite effectivement comme des variables de type *pair*. Par exemple, si on écrit :

`z1 shifted right;`

cela signifie que le point z_1 subit une translation de vecteur $(1,0)$.

2. **unitvector** permet de normer un vecteur :

`unitvector (1,1) = (0.7071,0.7071)`

ois la forme de
drawdot (voir
ps.

los vues aupar-
un stylo dont la
exe) défini par
hexagonale.

ol
rol
n
lonpoint
lontimes
t
point
les cas à des
espectivement
encontre, il les
le, si on écrit :

3. **round** appliqué à une paire revient à faire **hround** sur l'abscisse et **vround** sur l'ordonnée :

$$\text{round} (2.718, -3.14159) = (3, -3)$$

4. **dir** suivi d'une valeur numérique indique un vecteur unitaire faisant un angle (mesuré en degrés) de cette valeur :

$$\text{dir } 45 = (0.7071, 0.7071)$$

$$\text{dir } 120 = (-0.5, 0.86603)$$

$$\text{dir } 210 = (-0.86603, -0.5)$$

$$\text{dir } 360 = (1, 0)$$

5. **Ift**, **rt**, **top**, **bot** ont la même signification sur une paire que sur chacune des coordonnées de la paire. Elles ont été expliquées au paragraphe 3.1.2. De la même façon **good.Ift**, **good.rt**, **good.top** et **good.bot** se comportent comme des paires de manière analogue à *good.x* et *good.y*. Ces macros font en sorte que le sommet respectivement gauche, droit, supérieur ou inférieur de la portion de stylo soit placé à une valeur entière : l'instruction $z_3=\text{good.top}(\alpha, \beta)$ signifie que z_3 sera proche de (α, β) et que le stylo sera dans une « bonne » position du point de vue du tracé de la courbe en un point (α, β) qui serait à taux horizontale. On y reviendra par la suite (voir le paragraphe 5.6 p. 159).

6. Voici quelques exemples de la façon dont **min** et **max** agissent sur des paires :

$$\text{max}((2,3), (1,4)) \quad (2,3)$$

$$\text{max}((1,3), (1,4)) \quad (1,4)$$

$$\text{max}((-2,3), (0,1)) \quad (0,1)$$

$$\text{max}((1,2), (1,0)) \quad (1,2)$$

7. **point**, **precontrol**, **postcontrol** et **direction** ont des syntaxes similaires :

point *t* **of** *p*

precontrol *t* **of** *p*

postcontrol *t* **of** *p*

direction *t* **of** *p*

Le première instruction, **point** *t* **of** *p*, calcule les coordonnées du point correspondant au « temps » *t* sur le chemin *p* calculé au moyen des polynômes de Bernstein. Si le chemin passe par $n+1$ sommets, il est de longueur *n* et la variable *t* varie de 0 à *n*¹⁵. Ce chemin *p* utilise, en plus des points de construction, des points de contrôle¹⁶ : ce sont eux précisément que fournissent les commandes **precontrol** et **postcontrol**.

La quatrième instruction, **direction** *t* **of** *p*, fournit la direction de la tangente au « temps » *t* : voir la figure 3.9.

15. Ces notions ont été illustrées au paragraphe 3.1.6 p. 34 avec l'instruction **subpath**.

16. Pour plus de détails, voir à l'annexe A.

8. **directiontime** (*paire*) **of** (*chemin*). Cette instruction permet de calculer à quel moment un chemin pointe dans une direction donnée en supposant le chemin parcouru uniformément de son origine à son extrémité : t varie, comme toujours, de 0 à n si le chemin est de longueur n .
C'est l'instruction réciproque de **direction t of p** vue ci-dessus.
9. p **intersectionpoint** q fournit un point d'intersection des deux chemins p et q .
- La figure 4.51 à la page 94 illustre cette macro.
10. p **intersectiontimes** q fournit un couple de valeurs (t, u) signifiant que le point d'intersection éventuel de p et q se trouve (approximativement) au « temps » t sur p et au « temps » u sur q . S'il y a plusieurs points d'intersection, METAFONT en choisit un à sa manière¹⁷.

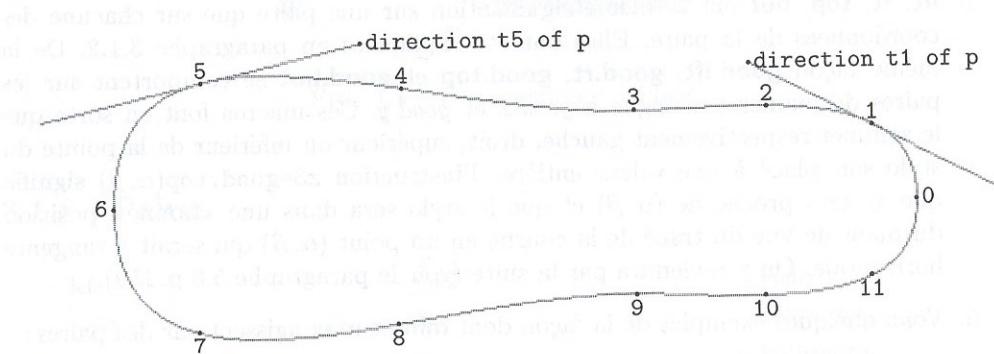


FIG. 3.9 – Instruction direction t of p

3.2 Instructions de programmation

3.2.1 Commandes non conditionnelles

- begingroup... endgroup
- save
- interim
- let
- def... endef
- vardef... endef
- primarydef... endef
- secondarydef... endef
- tertiarydef... endef
- showit
- shipout
- shipit
- cullit
- openit
- clearit
- clearxy
- clearpen
- stop
- show

1. **begingroup... endgroup** est une structure qui fonctionne de la même façon que les environnements pour L^AT_EX ou les groupes pour T_EX. Elle permet d'iso-

une partie de programme du reste du programme : à l'intérieur d'un groupe, un élément symbolique peut avoir une signification différente de celle qu'il avait à l'extérieur. On utilise pour cela les commandes **save** et **interim**. Grâce à cette instruction, des variables vont pouvoir être redéfinies à l'intérieur d'un groupe : la signification et les valeurs qu'elles avaient auparavant sont stockées en lieu sûr et elles les retrouveront à la sortie du groupe (après l'instruction *endgroup*). En attendant, elles peuvent être entièrement redéfinies. Par exemple, en écrivant l'instruction

```
save x,y
```

toutes les variables en *x* et *y* (quel que soit leur indice) peuvent figurer dans de nouvelles équations et dans de nouveaux calculs sans que METAFONT proteste par des messages d'erreur. Leurs anciennes valeurs sont momentanément inaccessibles. **interim** se comporte comme **save** mais s'applique uniquement aux variables internes de METAFONT (celles dont la liste est donnée à l'annexe C p. 183). On pourrait, par exemple, demander que provisoirement plus aucune information ne soit dirigée vers l'écran en créant un groupe et en écrivant :

```
begingroup
interim tracingonline:=0;
..... endgroup
```

2. **let** permet de rebaptiser un élément symbolique ou de redéfinir une macro. On a vu plus haut qu'il n'y a pas de macro *circle* : pour tracer un cercle, la macro s'appelle *fullcircle*. On peut y remédier en posant :

```
let circle = fullcircle;
```

3. **def...endef** permet de définir un nouvel élément symbolique. C'est une instruction très importante puisque c'est grâce à elle qu'on peut créer ses propres macros. La syntaxe est :

def élément symbolique = texte de remplacement **enddef**

4. **vardef...endef** présente une nuance par rapport à **def...endef**. Il faut savoir que METAFONT distingue des « sparks » (qui sont en quelque sorte des macros de bas niveau) et des « tags » (qui sont des éléments symboliques que l'on peut réutiliser dans d'autres définitions)¹⁸. Un élément *t* défini par **def** *t* = ... sera un « spark » tandis que, défini par **vardef** *t* = ... il sera un « tag ». Dans le premier cas, *t* ne pourra jamais être utilisé dans un suffixe. D'autre part il faut savoir que *vardef* enferme automatiquement ses définitions dans un groupe donc les entoure d'un *begingroup* et d'un *endgroup*. C'est une précaution utile qui peut pourtant, dans certains contextes, se révéler gênante.

¹⁸ On peut les assimiler respectivement à des primitives et des macro-définitions.

5. **showit** est la commande qui provoque l'affichage de l'image contenue dans la variable *currenpicture*.
6. **shipout image** est la commande qui envoie les pixels positifs de cette image dans un fichier générique (gf). **shipit** exécute la commande **shipout** sur l'image contenue dans la variable *currenpicture*.
7. **cullit**¹⁹ a pour effet d'éliminer les valeurs de pixels négatives (qui peuvent résulter de la soustraction de figures entre elles) et les valeurs de pixels supérieures ou égales à 2. Il ne restera plus alors que des valeurs 0 ou 1. C'est un cas particulier de la commande **cull** expliquée au paragraphe 3.3.
8. **openit** a pour effet d'ouvrir la fenêtre courante. Les fenêtres sont expliquées au paragraphe 3.7.
9. **clearit** a pour effet de remplacer l'image contenue dans la variable *currenpicture* par *nullpicture*.
10. **clearxy** intervient au sein d'un groupe (encadré par des instructions *begin-group* et *endgroup*) et a la signification de **save** *x,y*: les variables *x* et *y* sont réinitialisées à l'intérieur du groupe et reprendront leurs valeurs antérieures une fois le groupe terminé.
11. **clearpen** réinitialise *currentpen*.
12. **stop** donne l'ordre à METAFONT d'interrompre son travail. Une instruction *stop* devrait toujours être accompagnée d'un message (voir l'instruction **message** au paragraphe 3.9) pour indiquer à l'utilisateur ce qu'on attend de lui, sinon il risque de ne pas savoir ce qui se passe, ni pourquoi le programme s'arrête.
13. **show** est l'instruction qui demande à METAFONT d'expliciter le contenu d'une variable quelle qu'en soit la nature : toutefois les valeurs concernant les chemins, stylos et images ne sont montrées que dans le fichier de transcription (fichier .log) à moins que la variable *tracingonline* ne soit positive (Cf. annexe C). Cette instruction *show* est très pratique pour faire parler METAFONT lorsqu'on cherche à comprendre un comportement imprévu ou à identifier l'origine d'une erreur de programmation²⁰.

Citons aussi quelques macros qui ont un caractère beaucoup plus spécialisé et dont l'usage concerne davantage les programmeurs qui veulent construire de nouvelles bases.

1. **primarydef... endef**, **secondarydef... endef**, **tertiarydef... endef** sont à utiliser pour les définitions d'opérateurs binaires. Il y a trois niveaux dans les macros

19. Du verbe anglais *to cull* qui signifie trier, sélectionner. On verra au paragraphe 4.3.3 l'intérêt qu'il y a à sélectionner les pixels.

20. Voir au paragraphe 4.4.4 p. 109.

du format *plain* de METAFONT, des définitions pouvant ainsi contenir d'autres définitions. Voici, par exemple, la définition de l'opération *modulo*²¹ :

```
primarydef x mod y = (x-y*floor(x/y)) enddef;
```

2. Les instructions **showvariable**, **showtoken**, **showdependencies**, **showstats**, **message**, **errmessage**, et **errhelp**, sont expliquées au paragraphe 3.9.

3.2.2 Commandes conditionnelles

- | | | |
|---------------------------|------------------------|--------------|
| – if...elseif...else...fi | – upto | – exitunless |
| – for...endfor | – downto | |
| – forever...endfor | – forsuffixes...endfor | |
| – ...step...until | – exitif | |

if...elseif...else...fi est la structure conditionnelle habituelle pour aiguiller un programme en fonction d'une condition.

Il est possible aussi de programmer des boucles. Celles-ci peuvent être de quatre sortes :

1. **for** $x = \alpha_1, \alpha_2, \alpha_3$: *texte*(x) **endfor**

2. **for** $x = \mu_1$ **step** μ_2 **until** μ_3 : *texte*(x) **endfor**

3. **forsuffixes** $s = \sigma_1, \sigma_2, \sigma_3$: *texte*(s) **endfor**

4. **forever** *texte* **endfor**

La dernière boucle semble infinie et elle le sera à moins qu'une condition de sortie ne soit présente dans la partie *texte*. Ce sont les macros **exitif** et **exitunless** qui permettent cela. Elles permettent de passer une condition à tester pour sortir éventuellement de la boucle. C'est une manière de redéfinir l'instruction classique *while* qui n'existe pas dans METAFONT. **exitunless** *expr* correspond à **exitif** *not expr*.

La construction **...step...until** permet d'incrémenter une variable d'un certain pas et jusqu'à une certaine valeur. **upto** et **downto** en sont des cas particuliers très commodes : **upto** est une abréviation pour **step 1 until** et **downto** pour **step -1 until**.

²¹ On trouve une première mention au paragraphe 4.4.9.

3.3 Commandes de dessin

- penpos
- penstroke
- pickup
- `:=savepen`
- `clear_pen_memory`
- pen_lft
- pen_rt
- pen_bot
- pen_top
- draw
- undraw
- fill
- unfill
- filldraw
- unfilldraw
- drawdot
- undrawdot
- erase
- cutoff
- cutdraw
- addto...also
- addto...contour...withpen
- addto...contour...withweight
- addto...doublepath...withpen
- addto...doublepath...withweight
- cull...keeping
- cull...dropping

3.3.1 Définition des chemins

Les chemins sont définis, comme il a été dit au paragraphe 3.1.6 p. 34, au moyen de points de construction et de points de contrôle. METAFONT fournit de nombreux moyens pour déterminer ces divers points : ceux-ci peuvent être définis directement par leurs coordonnées ou bien indirectement par des équations que le programme se charge de résoudre.

Le tracé du chemin entre deux points successifs peut être de différentes sortes (*Cf.* paragraphe 3.1.6). Le chapitre 4 présente les diverses possibilités.

3.3.2 Choix d'un stylo

Tous les tracés sont faits au moyen de stylos ayant des pointes de formes diverses. La première chose à faire avant de donner une instruction de dessin est de choisir un stylo.

1. **pickup** est la commande (indispensable avant tout tracé) qui permet de définir le stylo qui sera utilisé par les commandes de dessin. L'instruction `nom:=savepen` permet d'attribuer un nom au stylo qui vient d'être défini. METAFONT attribue un numéro interne à ce stylo qui se trouve ainsi mémorisé avec toutes ses caractéristiques et pourra être réutilisé par la suite. L'instruction `clear_pen_memory` annule tous les styles qui ont été mis en mémoire. Si aucune instruction `pickup` n'est donnée, METAFONT utilise un style par défaut.

2. L'instruction **penpos** sert à préciser la position de la pointe du stylo au cours d'un tracé. **penpos** porte un indice qui est celui du point en lequel le stylo aura la position décrite : lorsqu'on a déclaré $\text{penpos}_k(b, d)$ la pointe du stylo au point z_k se trouve inclinée d'un angle d et a une largeur de b pixels. De plus METAFONT se trouve alors en possession de deux points supplémentaires z_{kl} et z_{kr} situés symétriquement de part et d'autre de z_k , séparés d'une distance de b pixels, la droite $z_{kl}z_{kr}$ faisant un angle d par rapport à l'horizontale. Les coordonnées de z_{kl} et z_{kr} sont calculées automatiquement par METAFONT et ces deux points sont donc directement utilisables dans les calculs. Le dessin de la figure 3.10 résulte des instructions :

```
penpos1(60,120);
penpos2(40,45);
```

En modifiant les valeurs de b et d à chaque point de contrôle, on obtient des effets de calligraphie (pleins et déliés).

3. **penstroke** vient compléter les commandes précédentes. Les points gauche et droit qui résultent d'une instruction **penpos** peuvent conduire à des instructions de tracé assez fastidieuses à écrire. **penstroke** fournit une abréviation. On en comprendra mieux la syntaxe sur un exemple concret²² :

```
penstroke z1e..z2e{right}..{right}z3e
```

est une manière abrégée d'écrire

```
fill z1l .. z2l{right} .. {right}z3l --
z3r{left} .. {left}z2r .. z1r -- cycle
```

On notera que l'instruction *cycle* est incluse dans la définition de *penstroke*.

4. **pen_lft**, **pen_rt**, **pen_bot** et **pen_top** sont les quantités utilisées pour le calcul des variables *lft*, *rt*, *top* et *bot*. C'est aussi un moyen de localiser précisément où passera le bord de la pointe d'un stylo. Par exemple, si on a choisi un stylo à pointe elliptique inclinée de 30°,

```
pickup pencircle xscaled .5pt yscaled .3pt rotated 60;
```

on aura les valeurs :

```
pen_lft = -8
pen_rt = 8
pen_bot = -6,5
pen_top = 6,5
```

²² La lettre *e* en indice signifie *edge*, le bord du stylo.

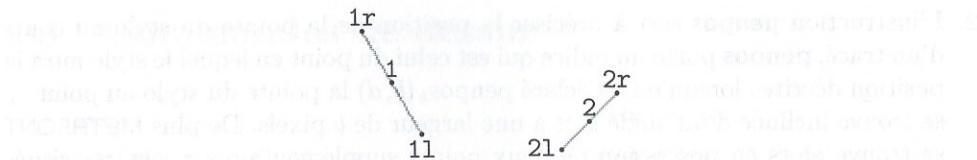


FIG. 3.10 – Position de la pointe du stylo

3.3.3 Tracés et contours

Toutes les commandes qui suivent sont amplement détaillées et illustrées au chapitre 4.

1. **draw** s'applique à un chemin et en effectue le tracé au moyen du stylo défini par la commande **pickup**. Le résultat de ce tracé dépend aussi de la façon dont le chemin est défini. **undraw** est la commande opposée.
2. **fill** s'applique à un chemin cyclique et noircit l'intérieur. **unfill** fait l'opération inverse. La commande **fill** trace le chemin, qui a une certaine épaisseur (de la taille de la pointe du stylo courant) puis remplit la zone ainsi délimitée. En procédant ainsi, les points de construction du chemin cyclique ne sont donc pas sur le pourtour de la région remplie : celle-ci « déborde ».
3. **filldraw** est une combinaison des commandes *draw* et *fill* : il faut que le chemin soit cyclique. Il n'y a pas avec *filldraw* de débordement contrairement à la commande *fill* toute seule : les points de construction seront sur la frontière de la zone remplie. **unfilldraw** combine *draw* et *unfill*.
4. **drawdot** utilise aussi le stylo courant. Cette commande s'applique en un point unique et trace un chemin de un pixel autour du point (le point obtenu aura donc la forme de la pointe du stylo augmentée de un pixel). **undrawdot** a l'effet inverse.

On écrira `drawdot scaled 10` par exemple pour obtenir un point plus gros.

5. **erase** s'utilise conjointement avec les commandes **fill**, **draw**, **filldraw** et **drawdot** et joue le rôle d'une gomme. On écrit :

```
erase fill p
erase draw p
erase filldraw p
erase drawdot p
```

6. **cutoff(z, θ)** supprime la moitié de la pointe du stylo courant au point z : tout ce qui se trouve entre les directions $(\theta - 90)^\circ$ et $(\theta + 90)^\circ$ est supprimé, ce qui fait l'effet d'une coupe nette au lieu d'une terminaison de la forme de la pointe.

7. La commande **cutdraw** fait usage de **cutoff** et trace un chemin dont les extrémités sont coupées net perpendiculairement aux directions de départ et d'arrivée.

3.3.4 Manipulation de tracés

L'intérêt des commandes qui suivent est de construire un dessin en plusieurs morceaux qu'on peut par la suite combiner.

- La commande **addto** V **also** P permet d'ajouter une expression P de type *image* à une variable V de type *image* elle aussi. V doit évidemment contenir une image connue²³. C'est une erreur courante que d'essayer d'appliquer la commande **addto...also** à des variables qui ne sont pas de type *image* mais de type *chemin*: METAFONT réagit dans ce cas par un message d'erreur.
- Il y a aussi des formes plus évoluées de la commande **addto** pour permettre de traiter le cas des chemins :
 - addto** V **contour** c où c est un contour;
 - addto** V **doublepath** p où p est un chemin cyclique.
- On peut, d'autre part, ajouter deux instructions optionnelles :
 - withpen** p ²⁴;
 - withweight** w . Le poids w vaut 1 par défaut. Sinon, il peut prendre les valeurs suivantes : -3, -2, -1, +1, +2 ou +3²⁵.
- La commande **cull** V **keeping** (a, b) **withweight** w est une généralisation de la commande **cullit** (voir au paragraphe 3.2.1). Elle agit de la manière suivante pour deux nombres a et b tels que $a \leq b$, toute valeur de pixel v comprise entre a et b sera remplacée par w et toutes les autres par 0. Si **withweight** w n'est pas précisé, alors $w = 1$.
La commande **cullit** en particulier a pour effet d'éliminer tous les pixels négatifs. Elle a pour signification :
cull currentpicture keeping $(0, \infty)$.
On verra des illustrations des possibilités ainsi offertes au paragraphe 4.3.3 : *Couleurs et figures* 4.43 p. 88 à 4.46 p. 89.
- La commande **cull** V **dropping** (a, b) **withweight** w remplace toute valeur v extérieure à l'intervalle $[a, b]$ par w et les autres par 0. C'est le contraire de **cull keeping** (a, b) .

23. Attention de bien distinguer *expression* et *variable*.

24. Si aucun style n'est précisé, **nullpen** sera choisi par défaut.

25. Pas la valeur 0.

3.4 Exécution d'une tâche

- \mode=
- mag=
- screenchars
- screenstrokes
- imagerules
- gfcorners
- nodisplays
- input

\mode: il s'agit là d'une instruction que l'on passe à METAFONT au moment d'exécuter un fichier²⁶. Le mode est le nom attribué au périphérique de sortie que l'on va utiliser : ce mode fixe certaines valeurs numériques spécifiques au matériel utilisé (les variables *blackter*, *fillin*, *o_correction*, etc.). Tous les renseignements sont contenus dans le fichier *mode.mf* qui est chargé par METAFONT dans la base au moment de l'initialisation (*Cf.* paragraphe 2.3). Une imprimante laser Apple à 300 dpi ou les Canon CX, SX, LBP-LX ont un mode qui est baptisé : *CanonCX* ou bien *cx*. Pour une laser Apple à 600 dpi, le mode s'intitule *canonex*. Enfin, pour une Varityper 4200 B-P à 1800 dpi, le mode s'appelle *vtftzz*. Le fichier *mode.mf* répertorie ainsi un très grand nombre de modèles²⁷. On doit obligatoirement communiquer à METAFONT une déclaration du type : *\mode=canonex*; avant d'exécuter un fichier. Si cette instruction est absente, METAFONT considère que le mode est *proof*. Celui-ci permet de fabriquer des épreuves préliminaires des caractères mais ne fabrique pas une police définitive : en particulier il ne produit pas de fichier *.tfm* (*TeX font metrics*).

Il existe une variable intitulée **localfont** qui contient, sur une installation particulière, le nom du mode associé au périphérique de sortie : un utilisateur qui ne connaît pas le nom exact du mode à employer se contente de déclarer *\mode=localfont*;

On peut, immédiatement après la déclaration de mode, faire exécuter un fichier *.mf* : *input nom du fichier* (sans l'extension *mf*).

METAFONT commence alors son travail et, si tout va bien, autrement dit si aucune erreur de syntaxe n'est détectée, il produit un fichier *.gf* (*generic font*) et un fichier *.tfm*. Le fichier sera transformé en fichier *.pk*²⁸ grâce à un programme tel que *gftopk*, la police sera alors disponible pour être utilisée par *TeX*. Toutes ces procédures sont expliquées en détail au chapitre 2.

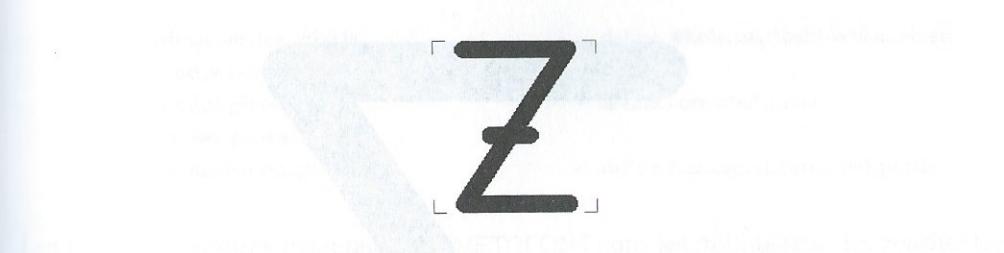
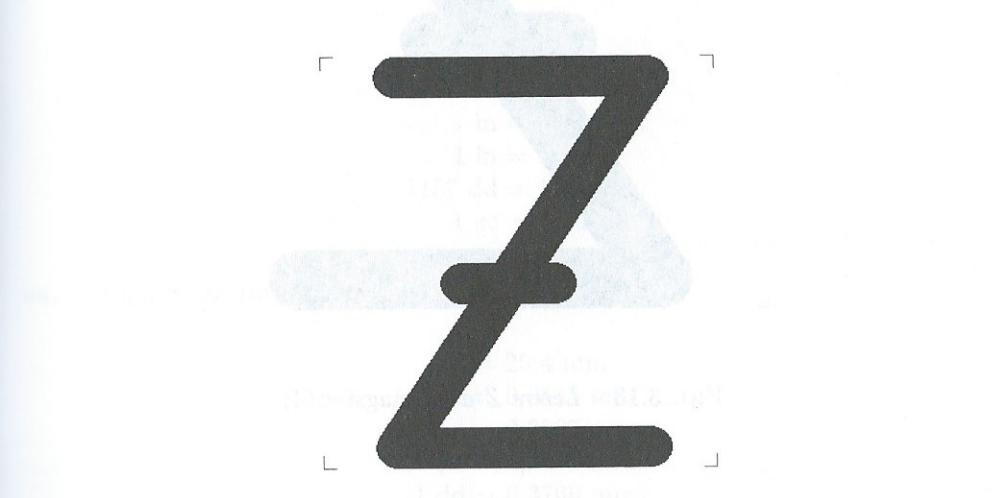
Entre les instructions **\mode** et **input**, il est possible de passer des instructions particulières :

1. **mag** et **magstep**. On peut demander à METAFONT de fabriquer la police avec un certain taux d'agrandissement : par exemple en écrivant *mag=1.2* on obtient une police agrandie de 20% par rapport à la taille pour laquelle elle a été conçue. Les taux d'agrandissement sont souvent choisis parmi les puissances de 1,2 :

26. Le ** est indispensable pour que METAFONT ne croie pas qu'il s'agit du nom d'un fichier car l'instruction se place en début de ligne.

27. Se reporter à l'annexe G.

28. *pk* signifie *packed* car les fichiers *pk* sont des fichiers *gf* compactés afin d'économiser de la place.

FIG. 3.11 – Lettre Z avec *mag=.6*FIG. 3.12 – Lettre Z avec *magstep(2)*

$1,2^2 = 1,44$; $1,2^3 = 1,728$; $1,2^4 = 2,0736$, etc. **magstep(n)** est synonyme de **mag**= $1,2^n$. Les figures 3.11 à 3.13 ont été obtenues respectivement en imposant les commandes suivantes au moment de lancer METAFONT :

```
mag=.6
mag=magstep(2)
mag=magstep(4)
```

2. **screenchars** demande à METAFONT de dessiner à l'écran chaque caractère traité dès qu'il rencontre l'instruction **endchar**.
3. **nodisplays** est l'instruction opposée à **screenchars**.
4. **screenstrokes** demande à METAFONT de dessiner à l'écran chaque caractère trait par trait, autrement dit après chaque instruction du type **draw** ou **fill** au fur et à mesure qu'il les rencontre.

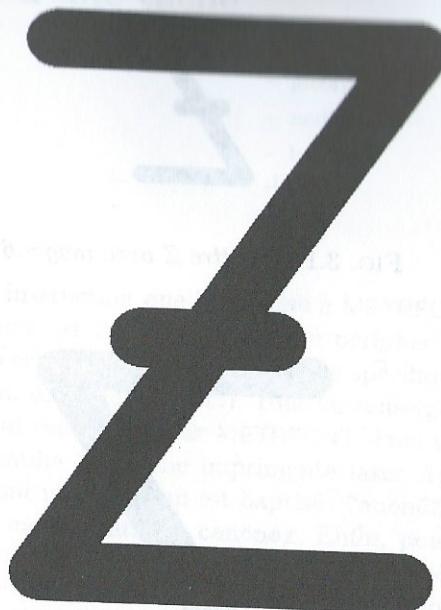


FIG. 3.13 – Lettre Z avec magstep(4)

5. **imagerules** fait que la boîte qui entoure les caractères est incluse avec eux :

M E T A F O N T
M E T A P O S T

6. Avec la commande **gfcorners**, les coins des boîtes entourant les caractères seront dessinés et envoyés dans le fichier .gf. Cela ne fonctionne qu'en mode *smoke*.

3.5 Pixellisation

- mm#, cm#, pt#, pc#, dd#, cc#, bp#, in# - blacker
- mm, cm, pt, pc, dd, cc, bp, in - fillin
- mode_setup - o.correction
- pixels_per_inch - define_pixels
- fix_units - define_whole_pixels

- define_whole_vertical_pixels
- define_good_x_pixels
- define_good_y_pixels
- define_blacker_pixels
- define_whole_bigger_pixels
- define_whole_vertical_bigger_pixels
- define_corrected_pixels
- define_horizontal_corrected_pixels

Les unités de mesures reconnues par METAFONT sont les millimètres, les centimètres, les points d'imprimeur, les gros points, les picas, les didots, les ciceros et bien sûr les pouces (en abrégé : mm, cm, pt, bp, pc, dd, cc, in). En voici les valeurs :

$$\begin{aligned}1 \text{ in} &= 72,27 \text{ pt} \\1 \text{ in} &= 2,54 \text{ cm} \\1 \text{ in} &= 25,4 \text{ mm} \\1 \text{ in} &= 72 \text{ bp} \\1157 \text{ dd} &= 1238 \text{ pt} \\1 \text{ cc} &= 12 \text{ dd}\end{aligned}$$

ou, en rapportant tout en millimètres :

$$\begin{aligned}1 \text{ in} &= 25,4 \text{ mm} \\1 \text{ pt} &= 0,3515 \text{ mm} \\1 \text{ bp} &= 0,3527 \text{ mm} \\1 \text{ pc} &= 4,2175 \text{ mm} \\1 \text{ dd} &= 0,3760 \text{ mm} \\1 \text{ cc} &= 4,5128 \text{ mm}\end{aligned}$$

En fait, METAFONT connaît deux sortes d'unités : les unités vraies (elles sont « diées ») et les unités relatives à la résolution. Les variables `mm#`, `cm#`, `pt#`, `pc#`, `dd#`, `cc#`, `bp#`, `in#` sont les valeurs réelles tandis que la quantité `pt`, par exemple, désigne le nombre de pixels contenus dans un point d'imprimeur. Il faut considérer les variables non diées comme des nombres de pixels et non comme des unités de mesures. Ce nombre est donc variable en fonction de la résolution du périphérique de sortie (écran ou imprimante) : pour une imprimante à 300 dpi, `pt` représente $\frac{300}{72,27} \approx 4,15$ pixels (donc 4 pixels). Pour une imprimante à 600 dpi, ce sera le double. Le processus de conversion des variables en nombres de pixels repose sur un certain nombre de commandes :

- La commande **mode_setup** donne l'ordre à METAFONT d'aller lire les valeurs des paramètres spécifiques : ces valeurs font partie de la définition du mode choisi²⁹ et servent à apporter des correctifs permettant d'obtenir de meilleures impressions en fonction du matériel que l'on utilise. C'est l'ensemble de ces valeurs qui constitue ce que l'on appelle le mode. On y trouve en particulier la

²⁹ Cf. paragraphe 3.4.

variable **pixels_per_inch** qui contient la résolution du périphérique de sortie³⁰ de même que des valeurs appropriées pour les variables correctives **blacker**, **fillin** et **o_correction** qui sont expliquées plus bas. La commande **mode_setup** effectue donc les réglages commandés par le mode que l'on choisit. Notons que c'est *mode_setup* qui établit *proof* comme mode par défaut.

2. La commande **fix_units** définit les facteurs de conversion entre les différentes unités de mesure à partir des nombres de pixels par pouce³¹. Sa définition dans le format *plain* est :

```
def fix_units =
  mm:=pixels_per_inch/25.4;
  cm:=pixels_per_inch/2.54;
  pt:=pixels_per_inch/72.27;
  pc:=pixels_per_inch/6.0225;
  dd:=1238/1157pt;
  cc:=12dd;
  bp:=pixels_per_inch/72;
  in:=pixels_per_inch;
  hppp:=pt;
  vppp:=aspect_ratio*hppp;
enddef;
```

hppp et **vppp** sont les nombres de pixels horizontaux ou verticaux par point : les pixels n'étant pas toujours parfaitement circulaires, ces valeurs peuvent différer et leur rapport s'appelle *aspect_ratio*.

3. La commande **define-pixels** est extrêmement importante car c'est elle qui fait la conversion des variables diésées en nombres de pixels. On l'écrit après que **mode_setup** a été invoquée. Elle agit comme une fonction portant sur les variables qui ont été définies précédemment par l'utilisateur. Imaginons à titre d'exemple que l'on a défini trois variables sous les noms de **larg#**, **graisse#** et **k#** au moyen de définitions telles que **larg#=5/7pt#**, etc. Ce sont des variables diésées, ce qui signifie qu'à la sortie la variable *larg* mesurera réellement³² cinq septièmes de point d'imprimeur. Or l'imprimante attend de savoir combien cela fait de pixels à placer. Il est donc indispensable de donner la commande **define_pixels(larg,graisse,k)** pour que se fasse la conversion en pixels, selon la formule :

$$\text{larg} := \text{larg\#} * \text{hppp}$$

hppp est le nombre de pixels par pouce, dans la direction horizontale, du périphérique de sortie.

30. Il s'agit en général des imprimantes, mais un moniteur peut, lui aussi, avoir un mode.

31. Cf. annexe B p. 179

32. À peu de choses près, compte tenu des erreurs rencontrées au moment de la mise en page.

4. **blacker**, **fillin** et ***o_correction*** sont donc des variables définies avec le mode et qui sont utilisées par les neuf commandes ci-dessous. Celles-ci sont des variantes de la commande **define_pixels**. L'utilisateur n'a pas, en principe, à se préoccuper des valeurs de *blacker*, *fillin* et **o_correction**. Mais il peut être sûr que ces corrections permettront d'obtenir de meilleurs rendus à l'impression sur tel ou tel matériel.

- (a) **define_blackter_pixels(larg#)** revient à dire :

$$\text{larg} := \text{larg\#} * \text{hPPP} + \text{blacker}$$

- (b) **define_whole_pixels** fait un arrondi sur la coordonnée horizontale (en se servant de la fonction *hround*) et fournit un nombre entier.
- (c) **define_whole_vertical_pixels** fait un arrondi sur la coordonnée verticale (en se servant de la fonction *vround*) et fournit un nombre entier.

- (d) **define_whole_blackter_pixels** est une combinaison des commandes précédentes : *define_blackter_pixels* et *define_whole_pixels*.
- (e) **define_whole_vertical_blackter_pixels** est une combinaison des commandes *define_blackter_pixels* et *define_whole_vertical_pixels*.

- (f) Les commandes **define_good_x_pixels** et **define_good_y_pixels** doivent être appelées après qu'un stylo a été sélectionné (par une commande *pickup*). Elles se servent respectivement des fonctions *good.x* et *good.y* vues au paragraphe 3.1.2. Par exemple :

define_good_x_pixels(k#) est synonyme de : $k := \text{good.x}(k\# * \text{hPPP})$

- (g) **define_corrected_pixels(larg#)** est synonyme de :

$$\text{larg} := \text{vround}(\text{larg\#} * \text{hPPP} * \text{o_correction}) + \text{eps}$$

- (h) **define_horizontal_corrected_pixels(larg#)** est synonyme de :

$$\text{larg} := \text{hround}(\text{larg\#} * \text{hPPP} + \text{o_correction}) + \text{eps}$$

La question est de savoir maintenant laquelle choisir de toutes ces commandes : cela dépend du type de variable que l'on a à convertir. *define-whole-pixels* sera choisie lorsqu'il est souhaitable que la variable soit arrondie avant d'être pixellisée. *define-blackter-pixels* convient pour des variables désignant par exemple l'épaisseur d'un jambage ou les dimensions de la pointe d'un stylo. *define-good-x-pixels* et *define-good-y-pixels* conviennent aux variables qui doivent être du type *good.x* ou *good.y*, c'est-à-dire bien placées par rapport aux tangentes verticales (bords de la boîte par exemple). *define-corrected-pixels* concerne des variables mesurant ou fixant le débordement (*overshoot*) d'un trait par rapport à la ligne de base ou aux bords de la boîte qui entourent les caractères : celui-ci doit être modéré sur les imprimantes à basse résolution ³³.

³³ *o_correction* vaudra de l'ordre de 0,4 sur une imprimante à basse résolution tandis que l'on pourra tranquillement prendre la valeur 1 — autrement dit annuler la correction — lorsque la résolution est élevée.

3.6 Gestion des caractères et des polices

- beginchar
- endchar
- italcorr
- change_width
- extra_beginchar
- extra_endchar
- font_size
- font_slant
- font_normal_space
- font_normal_stretch
- font_normal_shrink
- font_x_height
- font_quad
- font_extra_space
- fontdimen
- headerbytes
- font_identifier
- font_coding_scheme
- ligtable
- charlist
- extensible

beginchar et **endchar** sont certainement les commandes les plus importantes dans la constitution d'une police de caractères. Pour chaque caractère, elles définissent les dimensions de la boîte englobante et indiquent le code qui sera attribué à ce caractère dans la police. Les instructions concernant le tracé d'un caractère sont toujours comprises entre ces deux commandes. **beginchar** attend donc quatre arguments :

beginchar(*code,largeur#,hauteur#,profondeur#*)

où *code* désigne le code du caractère dans la police, *largeur#* désigne la largeur³⁴ et *hauteur#* et *profondeur#* désignent la hauteur et la profondeur par rapport à la ligne de base de la boîte qui contient ce caractère. Tout caractère est inscrit virtuellement dans une boîte et c'est en tant que boîte qu'il est traité par TeX³⁵.

largeur#, hauteur# et *profondeur#* sont exprimées impérativement en unités diésées. Leurs valeurs sont envoyées dans le fichier *.tfm* au moment de la fabrication de la police par METAFONT.

Toutes les variables ou instructions décrites dans cette section contiennent des informations qui sont importantes pour caractériser une police de caractères. Ce sont des informations qui se retrouveront, elles aussi, dans le fichier *.tfm* afin d'être exploitées par TeX. Il ne faut surtout pas les négliger lors de la constitution d'une police de caractères.

1. **italcorr(valeur numérique#)** définit un blanc supplémentaire à ajouter à la suite d'un caractère en italique pour tenir compte du fait qu'il déborde de la boîte qui le contient et risque de se superposer au caractère qui le suit. La commande

34. Les typographes disent la *chasse*.

35. Rien n'interdit que l'*image* du caractère sorte cependant de la boîte. C'est le cas, en général, des caractères penchés.

n'est utilisée que pour les polices italiques et penchées. Si la *valeur numérique* est négative, la correction sera nulle. Cette *valeur numérique* doit être exprimée en unité diésée.

2. **change-width** est une instruction qui a pour effet de modifier la valeur de la largeur w du caractère (exprimée en pixels) lorsque celle-ci n'est pas une bonne valeur : w est remplacé par $w - 1$ ou $w + 1$ (celle de ces deux valeurs qui sera la plus proche de la valeur réelle de w). Par exemple si la largeur est de 15,6 METAFONT l'aura arrondie à 16. Si l'on donne une instruction **change-width** elle sera remplacée par 15 ou 17: 15 est plus proche de 15,6 que ne l'est 17 et on aura alors $w = 15$. Cette macro en définitive change la parité de w : une illustration de son emploi est donnée au chapitre 5 dans la définition de la lettre T (paragraphe 5.1.4 p. 116).
3. La commande **beginchar** vue plus haut définit un certain nombre d'actions telles que dessiner un cadre autour des caractères lorsque le mode est *proof*, expédier le caractère tracé dans le fichier approprié, etc. Avec la commande **extra_beginchar**, on peut ajouter des instruction à faire exécuter à tout appel de la commande **beginchar**. **extra_beginchar** est une variable de type chaîne et se voit assigner une valeur sous forme de chaîne :
`extra_beginchar "chaîne d'instruction"`
extra_endchar joue le même rôle vis-à-vis de la commande **endchar**.
4. **font_size** désigne la taille de la police de caractères. Il s'agit de la taille pour laquelle elle est dessinée et censée être optimale. Si aucune commande **font_size** n'est présente dans le fichier, METAFONT la fixe à 128 pt par défaut.
Les sept paramètres suivants sont connus de TEX sous les noms respectifs de `\fontdimen1` à `\fontdimen7`. Pour METAFONT, ils portent les noms suivants :
 - (a) **font_slant** mesure l'inclinaison par point. TEX l'utilise pour le placement des accents qui sont amenés à être élevés ou abaissés en fonction de la hauteur de la lettre sur laquelle ils portent. Cette même quantité est aussi utilisée par `gftodvi` pour faire des sorties inclinées.
 - (b) **font_normal_space** est l'espacement entre les mots.
 - (c) **font_normal_stretch** est l'extensibilité de l'espacement entre les mots.
 - (d) **font_normal_shrink** est la quantité maximale dont on peut diminuer l'espacement entre les mots.
 - (e) **font_x_height** est la quantité que TEX désigne par `ex`. C'est, en principe, la hauteur du caractère x qui sert, entre autres, au placement des accents.
 - (f) **font_quad** est la quantité que TEX désigne par `em`. C'est en principe la largeur du caractère m .
 - (g) **font_extra_space** est l'espacement supplémentaire ajouté entre les phrases à l'espacement normal entre mots.

5. METAFONT a une commande **fontdimen** qui correspond à la commande de TeX `\fontdimen`, mentionnée ci-dessus. Voici un exemple de la syntaxe :

```
fontdimen 3 : 2.5, 6.5, 0, 4x
```

qui attribue ces valeurs aux variables `fontdimen3`, `fontdimen4`, `fontdimen5` et `fontdimen6` respectivement.

6. **litable** introduit toute l'information concernant les ligatures et les approches de paires (espacement particulier entre certaines paires de caractères). La syntaxe en est très simple, comme on peut le voir sur l'exemple suivant :

```
litable "f" : "i" =: oct"014", "f" =: "oct013", "l" =:  
oct"015", ' ' kern .5pt#, "?" kern .55pt#, "!" kern .5pt#;
```

Cela signifie que lorsque TeX rencontre un f suivi d'un i, il devra substituer le caractère de code octal "014" (donc 12 en décimal) pour obtenir « fi » au lieu de « fi ».

`kern .5pt#` signifie que TeX devra ajouter un espace supplémentaire de 0,5pt# entre un f et une apostrophe ou un point d'interrogation ou d'exclamation³⁶. La quantité attribuée à `kern` peut-être négative lorsque l'on souhaite au contraire rapprocher des lettres comme c'est le cas avec A et V : comparer AV avec AV.

7. **charlist** lie ensemble plusieurs caractères qui sont susceptibles d'être utilisés par TeX en diverses tailles suivant le contexte. Cela concerne principalement les délimiteurs (parenthèses, crochets, etc.) ou des accents de taille variable ou encore des symboles mathématiques qui existent en plusieurs tailles (comme \sum par exemple). La syntaxe est par exemple :

```
charlist oct"000": oct"020": oct"022": oct"040": oct"060"
```

Cet exemple correspond à la parenthèse gauche dans des tailles croissantes :

8. **extensible** concerne les symboles mathématiques extensibles (grandes parenthèses, grandes accolades devant des séries d'équations) qui sont le résultat de l'assemblage de plusieurs morceaux. La syntaxe est la suivante par exemple :

```
extensible oct"060": oct"060", 0, oct"100", oct"102"
```

correspondant aux éléments  qui sont assemblés ensuite par TeX pour faire de grandes parenthèses comme celles qui figurent dans la marge ci-contre. La seconde valeur (nulle dans l'exemple ci-dessus) indique le code d'un éventuel élément central tel que les pointes des accolades correspondant aux valeurs octales oct"074" et oct"075": 

- Les premiers octets d'un fichier .tfm contiennent des informations importantes pour TeX. Les octets 1-4 contiennent un nombre dénommé « checksum » qui sert à vérifier l'intégrité du fichier ³⁷. Les octets 5-8 contiennent la taille de la police multipliée par 2^{20} . **headerbytes** permet de compléter ces informations. La syntaxe est analogue à celle de **fontdimen** sauf que les valeurs passées doivent être des octets. On écrira par exemple :

```
headerbyte 33: 0, 214, 0, "c"
```

ce qui signifie que les octets 33 à 36 du fichier .tfm auront les valeurs 0, 214, 0 et 99 (code de la lettre c). Cette commande **headerbytes** est utilisée pour des programmes autres que TeX qui font usage des fichiers .tfm. En particulier **font_identifier** et **font_coding_scheme** sont des informations (codées respectivement sur les octets 9-48 et 49-68) pour décrire de grandes familles de polices ³⁸.

8.7 Affichage écran

- currentwindow
- screen_rows
- screen_cols
- openwindow...from...to...at
- display...inwindow

La commande **openwindow k from** (l_0, c_0) **to** (l_1, c_1) **at** (x, y) définit une zone de l'écran calculée en nombre de lignes et de colonnes à partir d'un sommet supérieur gauche situé au point de coordonnées (x, y) lui-même repéré à partir du point supérieur gauche de l'écran (l'axe des y est orienté vers le bas). On pourra alors édicter une commande :

```
display currentpicture inwindow k
```

pour que l'image fabriquée par METAFONT soit affichée dans la k -ième fenêtre.

³⁷ Le principe est qu'il doit correspondre à une valeur identique dans le fichier .gf. Il peut se faire effet que la police soit modifiée entre deux utilisations sans que le fichier .tfm le soit. checksum permet de détecter cet état de fait.

³⁸ Cf. The METAFONTbook, p. 320.

On peut ainsi définir seize fenêtres numérotées de 0 à 15. La commande **openit** ouvre la fenêtre *currentwindow*, variable qui vaut 0 par défaut mais à laquelle on peut assigner une autre valeur : **currentwindow** = *k*.

Enfin les variables **screen_rows** et **screen_cols** contiennent le nombre de lignes et de colonnes de l'écran. Par défaut, ces valeurs sont fixées respectivement à 400 et 500 mais peuvent être modifiées en fonction de l'installation en éditant un fichier *local.mf* ou directement dans le fichier *modes.mf* (*Cf.* paragraphe 2.3.2 p. 20).

3.8 Épreuves

- labels
- penlabels
- makelabel
- titlefont
- labelfont
- grayfont
- slantfont
- proofrule
- screenrule
- makegrid
- proofrulethickness
- proofoffset

Pour obtenir le maximum d'information à partir des épreuves des caractères que l'on est en train de créer³⁹, il est utile de pouvoir repérer graphiquement où se trouvent les points de construction qui ont permis de faire les divers tracés. La convention veut qu'on les désigne par une lettre (souvent *z*) avec un indice entier : par exemple *z1*, *z25*, *z[2]*, *z[126]*, etc. Les commandes suivantes ne fonctionnent que si *proofing* > 1.

1. Si on a défini des points z_1, \dots, z_n et que l'on souhaite qu'ils apparaissent sur les épreuves réalisées en mode *proof*, on utilise la commande **labels** à l'intérieur du groupe de définition du caractère (c'est-à-dire avant l'instruction **endchar**) :

```
labels(1, ..., n)
```

Les points z_1, \dots, z_n seront matérialisés sur l'épreuve par un point noir auprès duquel figure l'indice. S'il y a beaucoup de points, il existe une syntaxe générique que l'on comprendra facilement avec l'exemple ci-dessous :

```
labels(1, range 5 thru 12, range 101 thru 112, 153)
```

2. L'instruction **penlabels** montre non seulement les points z_1, \dots, z_n comme précédemment mais aussi les positions gauches z_{1l}, \dots, z_{nl} et les positions droites z_{1r}, \dots, z_{nr} associées à z_1, \dots, z_n et induites par les commandes *penpos*. À titre d'exemple, la figure 3.10 p. 46 a été obtenue simplement en écrivant :

```
draw z1l -- z1r;
draw z2l -- z2r;
penlabels(1,2);
```

3. **makelabel** permet de faire un marquage plus précis des épreuves. On a vu que **labels** écrit l'indice i du point z_i . Si l'on souhaite écrire autre chose, on aura recours à **makelabel** :

```
makelabel("marquage", zi)
```

Le programme gftodvi choisit automatiquement l'emplacement du marquage mais il reconnaît aussi huit autres options :

- (a) **makelabel.top** : point noir et marquage au dessus ;
 - (b) **makelabel.lft** : point noir et marquage à gauche ;
 - (c) **makelabel.rt** : point noir et marquage à droite ;
 - (d) **makelabel.bot** : point noir et marquage en dessous ;
 - (e) **makelabel.top.nodot** : comme .top mais pas de point noir ;
 - (f) **makelabel.lft.nodot** : comme .lft mais pas de point noir ;
 - (g) **makelabel.rt.nodot** : comme .rt mais pas de point noir ;
 - (h) **makelabel.bot.nodot** : comme .bot mais pas de point noir.
4. **titlefont**, **labelfont**, **grayfont**, **slantfont** sont quatre polices de caractères aux-
quelles METAFONT puis gftodvi ont recours pour faire les épreuves du mode
proof qui sont expliquées en détail au paragraphe 2.2.1 p. 10 : la première sert
à écrire le titre en haut de la page sur les épreuves (c'est, par défaut, la police
cmr8), la deuxième est celle qui marque les indices des points demandés par
une instruction *labels* par exemple (par défaut, c'est la police cmtt10). Les deux
autres servent à faire le « remplissage » et à visualiser les caractères en grisé.
slantfont sert évidemment au tracé des obliques. On trouvera de plus amples
détails au paragraphe 2.2.6 p. 14. Pour modifier ces quatre polices, on utilise la
commande **special** expliquée au paragraphe 3.10 p. 62.
5. **proofrule**(z_1, z_2) permet de tracer une droite sur les épreuves (pour matérialiser
la ligne de base par exemple ou tout autre repère utile). **proofrulethickness**
contrôle l'épaisseur de ces droites. Par exemple :

```
proofrulethickness 1.2mm#
```

6. **proofoffset**(x, y) permet de décaler l'emplacement du dessin sur l'épreuve.
7. **screenrule** a la même fonction que **proofrule** mais pour tracer des droites sur
les sorties à l'écran.
8. Enfin **makegrid** permet de tracer une grille pour mieux visualiser le placement
des caractères dans leur boîte.

La syntaxe est :

```
makegrid(liste d'abscisses)(liste d'ordonnées)
```

et a pour effet de tracer tous les segments allant d'une des abscisses de la *liste
des abscisses* à l'une des ordonnées de la *liste des ordonnées*. Alors la liste des
abscisses et la liste des ordonnées doivent être de mêmes longueurs.

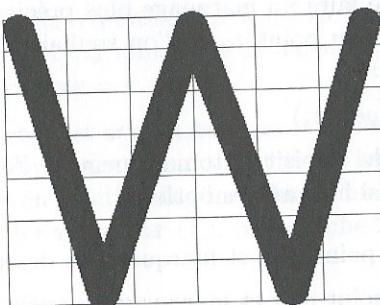


FIG. 3.14 – Instruction makegrid

a été obtenue en ajoutant la ligne suivante avant l'instruction *endchar* dans la définition du caractère W qui a pour dimensions 18u en largeur et 13,5u en hauteur :

```
makegrid(0,3u,6u,9u,12u,15u,18u)(0,2u,4u,6u,8u,10u,12u,13.5u);
```

3.9 Diagnostics

- message
- errmessage
- errhelp
- showvariable
- showtoken
- showdependencies
- showstats
- tracingall
- tracingnone
- interact
- loggingall
- hide
- ???

Dans cette section figurent les instructions relatives au traitement des erreurs.

1. L'instruction **message** *s* a pour effet d'afficher la chaîne de caractères *s* à l'écran pour communiquer un message ou un commentaire destiné à l'utilisateur.
2. L'instruction **errmessage** *s* se distingue de la précédente en ce que le message *s* est précédé d'un point d'exclamation et suivi d'un point. À la suite de ce point pourra figurer un message d'explication d'une éventuelle erreur. Ce message, qui peut avoir plusieurs lignes, apparaîtra si l'utilisateur frappe la touche H (H pour *help*) et est contenu dans la variable **errhelp**.
Tout comme pour **TEX**, il y a plusieurs modes d'exécution pour **METAFONT**

défaut), **scrollmode**, **nonstopmode** et **batchmode**. Ces trois derniers sont utilisés lorsque, en réponse à un arrêt du programme à la suite d'une erreur, on répond au point d'interrogation de METAFONT en frappant la touche S ou R ou Q respectivement⁴⁰.

- (a) **scrollmode** demande à METAFONT de continuer après chaque message d'erreur : cela revient à taper un retour-chariot à chaque message d'erreur affiché, ce qui fait que METAFONT ignorera la faute signalée et continuera sa lecture du fichier. Cependant, il y a quand même certaines erreurs qui l'arrêteront : s'il ne trouve pas un fichier appelé par une instruction *input*, il ne pourra pas poursuivre tant qu'on ne lui donnera pas le nom d'un autre fichier à ouvrir.
 - (b) **nonstopmode** est plus drastique encore puisque METAFONT ne s'arrêtera cette fois sous aucun prétexte. Il ignorera toute instruction fautive (même un *input* fautif) et on verra les messages d'erreur défiler très vite à l'écran. On pourra les lire tranquillement dans le fichier *log*.
 - (c) Enfin **batchmode** agit comme *nonstopmode* et, de plus, n'envoie aucun message d'erreur à la console. Ils sont dirigés seulement sur le fichier *log*.
3. La commande **showvariable** donne la structure de toutes les variables commençant par un symbole donné ainsi que leurs valeurs. En demandant à METAFONT :
- ```
showvariable x,y
```
- on obtient toutes les coordonnées qui ont été définies depuis le dernier **beginchar**.
4. **showtoken** donne la signification courante d'un symbole ou d'un élément de syntaxe. D'autre part, toute variable numérique non connue mais qui est dépendante des autres sera montrée si on utilise la commande **showdependencies**. Ces deux instructions sont utiles en phase de débogage.
5. **showstats** affiche des informations sur l'état de la mémoire utilisée par le programme METAFONT.
6. METAFONT fournit un certain nombre de variables permettant de contrôler le fonctionnement des programmes et de suivre leur déroulement. Ces commandes commencent toutes par le mot *tracing*. Ce sont en fait des variables à qui on doit assigner une valeur non nulle pour les activer. Lorsque **tracingall** = 1 toutes les commandes commençant par *tracing* (et répertoriées à l'annexe C) sont activées, de même que *showstopping*. **loggingall** agit comme **tracingall** avec cette différence qu'elle n'active pas *tracingonline* et *showstopping*. Enfin **tracingnone** fait l'inverse : elle met fin à toutes les commandes *tracing*.
7. **interact** correspond à *tracingonline:=showstopping:=1*.

<sup>40</sup> On peut aussi taper I pour insérer du texte, H pour obtenir de l'aide, X pour quitter pendant un cours ou E pour stopper et sortir.

8. La commande **hide** sert à introduire des commandes à l'intérieur d'une expression. METAFONT lit et exécute en premier lieu ce qui se trouve à l'intérieur de la commande **hide**. On en verra un exemple concret au chapitre 4 p. 101. Les commandes cachées par **hide** sont entre parenthèses et ne sont pas suivies d'un point-virgule :

```
hide(...)
```

9. Pour terminer, on tape **I ????** lorsqu'il se produit une erreur sur une variable et que METAFONT s'arrête en attendant qu'on lui donne une instruction. Cette instruction a pour effet de lui faire afficher toutes les relations actuelles entre variables numériques et permet de repérer plus facilement sur quoi portent les problèmes.

### 3.10 Divers

- **capsule\_def**
- **numtok**
- **gobble**
- **special**
- **numspecial**

Ces dernières instructions sont citées ici uniquement par souci d'exhaustivité. Elles interviennent uniquement dans la définition de certaines macros du format *plain* et ne concernent pas directement le dessin de caractères ou la constitution de polices. Elles ne seront utiles qu'au programmeur qui construit une base pour METAFONT :

1. **capsule\_def** sert essentiellement à prédéfinir des stylos. Si un stylo est prédéfini, il sera appelé beaucoup plus rapidement que si on le définit directement au moment de s'en servir avec l'instruction **pickup** : **pencircle**, **pensquare** et **penrazor** sont des exemples de stylos prédéfinis<sup>41</sup>. Or une variable ne peut pas être un stylo prédéfini tandis qu'une capsule le peut et il faut donc utiliser **capsule\_def** au lieu de **vardef**. Voici, par exemple, la définition du stylo *pensquare* :

```
capsule_def(pensquare) makepen(unitsquare shifted -(.5,.5));
```

2. **numtok** sert à transformer un suffixe en variable numérique :

```
def numtok suffix x=x enddef;
```

3. **gobble** signifie en anglais « engloutir ». Cette macro demande à METAFONT de ne rien faire ! Elle est utilisée, dans le format *plain*, uniquement pour la définition de l'instruction **stop** suivie d'un message : dans ce cas METAFONT s'arrête et

il faudra lui donner une instruction qu'il « engloutira » avant de reprendre son travail. En voici la définition<sup>42</sup> :

```
def gobble primary g = enddef;
def stop expr s = message s; gobble readstring enddef;
```

4. Les commandes **special** et **numspecial** ont pour but de transmettre au fichier gf (*generic font*) des commandes particulières qui ne sont pas interprétées par METAFONT mais le seront éventuellement par le programme qui traitera ce fichier gf, par exemple gftodvi (voir au paragraphe 2.2.1 p. 10). Cette commande est analogue à la commande `\special` de *TEX*. La commande **special** doit être suivie par une chaîne entre guillemets puis, éventuellement, suivie par des commandes **numspecial** qui transmettent des valeurs numériques associées aux instructions contenues dans la chaîne. Le programme gftodvi sait reconnaître les commandes *special* : si la chaîne a un sens pour lui, il réagira en conséquence, sinon elle sera purement et simplement ignorée. Les commandes *special* fonctionnent lorsque la variable *proofing* est  $\geq 0$ .

Si on souhaite, par exemple, qu'une police particulière appelée *fontmaison* soit employée au titre de police *gray* pour faire des épreuves des caractères (voir au paragraphe 2.2.6 p. 14), on pourra écrire :

```
special "grayfont fontmaison"
```

Si on veut que les titres soient écrits, en haut des épreuves, en corps 10, on écrira :

```
special "titlefont cmr10"
```

La macro **proofrule**<sup>43</sup>, qui trace sur les épreuves un segment de droite allant du point  $(x_1, y_1)$  au point  $(x_2, y_2)$ , est transmise au fichier gf sous la forme suivante :

```
special "rule";
numspecial x1; numspecial y1;
numspecial x2; numspecial y2;
```

42. *readstring* est une primitive de METAFONT.

43. Voir au paragraphe 3.8 p. 58.

## Chapitre 4

# Utilisation de METAFONT

METAFONT peut être utilisé non seulement comme outil destiné à produire des polices de caractères — ce qui est sa vocation première — mais également comme solveur d'équations, comme traceur de courbes, comme outil de dessin pour produire des figures qui pourront être utilisées aussi bien avec TeX qu'avec n'importe quel traitement de texte, ou encore comme langage de programmation. Nous donnerons un aperçu des possibilités d'extension des capacités de METAFONT par la création de nouvelles bases<sup>1</sup>.

### 4.1 Les courbes

#### 4.1.1 Courbes de Bézier

METAFONT offre des outils très puissants pour le tracé des courbes : il suffit de lui indiquer les points par lesquels on veut qu'une courbe passe. Ces points s'appellent des points de construction. METAFONT a sa propre idée de ce qui sera la meilleure courbe passant par les points qui lui sont imposés : elle repose sur la notion de courbes de Bézier et de polynômes de Bernstein (pour plus de détails, voir l'appendice A).

D'autre part, en plus des points de construction, nous allons voir qu'on peut imposer des points de contrôle qui définissent la direction de la tangente à la courbe en certains points de construction. Sur la figure 4.1 est dessinée une courbe de Bézier passant par les points  $z_1$  et  $z_4$  ayant  $z_2$  et  $z_3$  comme points de contrôles : la tangente en  $z_1$  est la droite  $z_1 z_2$  et la tangente en  $z_4$  est la droite  $z_3 z_4$ .

<sup>1</sup> Les bases complémentaires sont en quelque sorte à METAFONT ce que les packages sont à TeX.

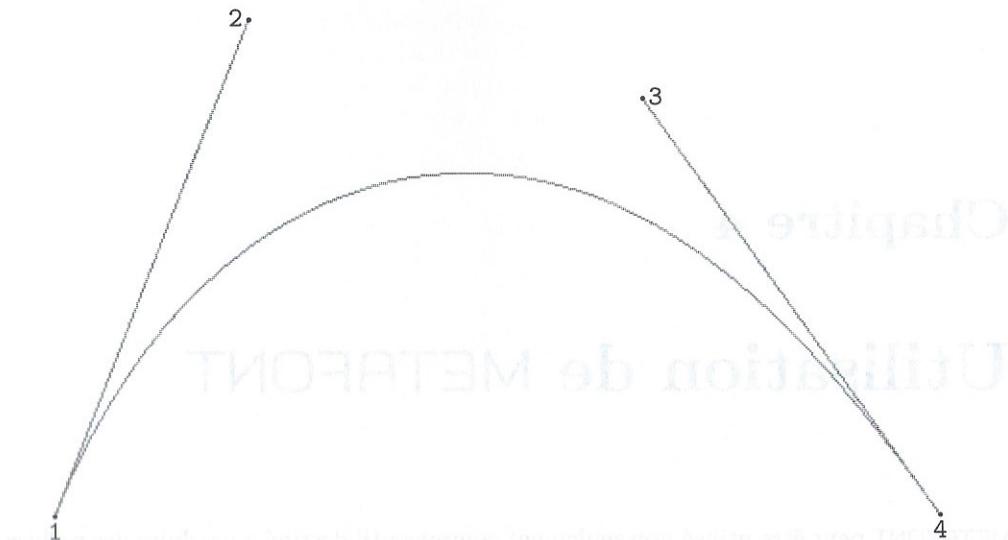


FIG. 4.1 – Courbe de Bézier: points de construction et de contrôle

### 4.1.2 Placement des points

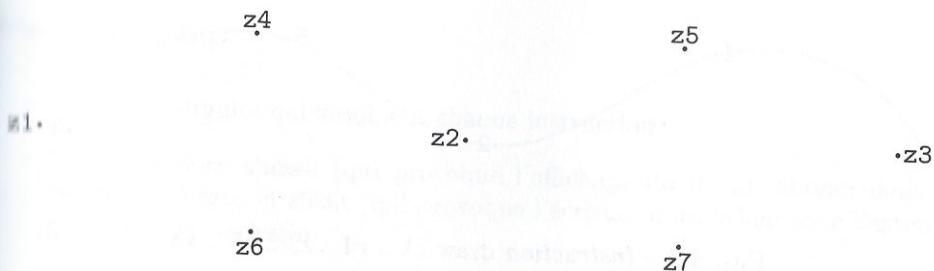
Nous allons dans ce chapitre voir progressivement comment mettre en œuvre les macros du format *plain*: tous les exemples qui vont suivre sont bâtis sur une structure unique faite de sept points  $z_1$  à  $z_7$  (figure 4.2). Ces points sont définis de la manière suivante :

```
z1=(0,0);
z2=(220,0);
z3=(440,0);
z4=(110,50);
z5=(330,50);
z6=(110,-50);
z7=(330,-50);
```

Comme ils sont définis sans préciser l'unité de mesure, METAFONT considère dans ce cas qu'il s'agit de pixels. Ces points sont représentés sur la figure 4.2.

Les points sont repérés par leurs coordonnées cartésiennes. Ils sont habituellement désignés par la lettre *z* puisque le format *plain* définit *z\$* comme étant la paire (*x\$*, *y\$*) où \$ désigne n'importe quel suffixe<sup>2</sup>. Cela nous conduit à préciser ce que l'on entend par *suffixe*: METAFONT est capable de traiter des variables ayant un suffixe composé

<sup>2</sup> Il résulte de cela qu'il n'est pas nécessaire de déclarer *z* comme une variable de type *pair*. En revanche, si on veut utiliser une autre lettre, il faudra faire une déclaration préalable.

FIG. 4.2 - Placement des sept points  $z_1$  à  $z_7$ 

telles que  $x_{1r}$ . L'indice peut être une valeur numérique mais peut aussi être une formule entre crochets comme par exemple  $x[2k+1]$  ce qui permet de traiter les variables de manière générique (dans des boucles conditionnelles par exemple) : si  $k$  varie de 0 à 2,  $x[2k+1]$  désignera successivement  $x_1, x_3, x_5$ .

D'autre part, les points de construction n'ont pas obligatoirement à être définis de manière explicite comme les sept points donnés. METAFONT est capable de résoudre des équations linéaires directement (ou même non linéaires au moyen de l'instruction *solve* expliquée au paragraphe 4.4.2). Ainsi les sept points auraient aussi bien pu être définis par les équations suivantes :

```
y1=y2=y3=0;
y4+y6=y5+y7=0;
y4=50; x1=0; x3=440;
x4=x6=x5-x7=0;
x3+x2=2x5;
x2=2x6=.5x3;
y6=y4;
```

### 4.1.3 Premiers coups de crayon

L'instruction la plus importante est **draw** qui permet de tracer aussi bien des courbes de Bézier que des segments de droites. Comparons les deux lignes d'instructions suivantes dont les résultats sont montrés sur les figures 4.3 et 4.4 respectivement :

```
draw z1..z4..z2..z5..z3;
draw z1--z4--z2--z5--z3;
```

Les points de construction sur ces exemples (de même que sur tous ceux qui suivent dans ce chapitre) sont maintenant marqués uniquement par un point noir.

FIG. 4.3 – Instruction `draw z1 .. z4 .. z2 .. z5 .. z3`FIG. 4.4 – Instruction `draw z1--z4--z2--z5--z3`

indice :  $i$  et non  $z_i$ . On obtient cela en écrivant, à la fin du fichier, une commande telle que :

```
makelabel("z1",z1);;makelabel("z7",z7);
```

Notons que sur la figure 4.2 le marquage des points a été obtenu en ajoutant des instructions **makelabel** :

```
makelabel("z1",z1);;makelabel("z7",z7);
```

En résumé, nous donnons ci-dessous, pour ce premier exemple, le fichier complet. Par la suite, il sera sous-entendu que les instructions illustrées sont incluses dans un fichier semblable que l'on ne prendra pas la peine de réécrire chaque fois *in extenso* :

```

z1=(0,0); z2=(220,0);
z3=(440,0); z4=(110,50);
z5=(330,50); z6=(110,-50);
z7=(330,-50);

pickup pencircle;
draw z1..z4..z2..z5..z3;

labels(range 1 thru 5);
showit;shipit;
end

```

Noter en particulier :

- les points-virgules qui terminent chaque instruction ;
- les instructions **showit** (qui provoque l'affichage du dessin obtenu dans une fenêtre à l'écran) et **shipit** (qui provoque l'écriture d'un fichier avec l'extension gf contenant ce dessin) ;
- l'instruction **labels** qui doit obligatoirement être placée avant le **shipit** faute de quoi elle restera sans effet ;
- le mot **end** qui termine impérativement le fichier.

Lorsqu'une courbe se referme sur elle-même on obtient un cycle et METAFONT possède une syntaxe particulière afin que les tangentes au point de départ (et donc d'arrivée) coïncident. On écrira :

```
draw z1..z4..z2..z5..z3..z7..z6..cycle;
```

qui donne la courbe cyclique de la figure 4.5. Attention, il ne suffit pas qu'un chemin se referme sur soi-même en revenant à son point de départ pour qu'il soit considéré comme un cycle, et s'il n'est pas un cycle, l'instruction *fill* ne marchera pas.

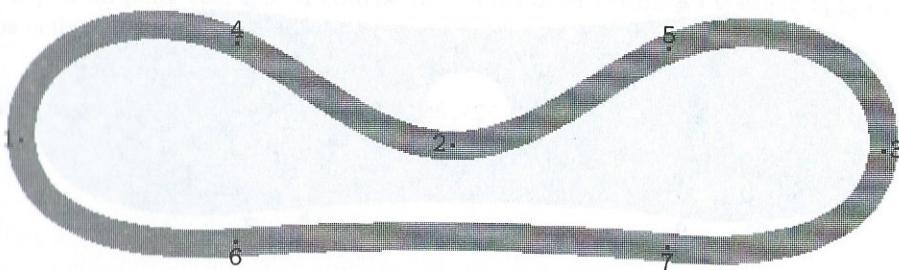


FIG. 4.5 – Ajout d'un cycle

En complément de la commande **draw** existe une commande **filldraw** qui s'applique uniquement à des chemins cycliques et remplit l'intérieur du cycle :

```
filldraw z1..z4..z2..z5..z3..z7..z6..cycle;
```

comme on peut le voir sur la figure 4.6.

Pour toutes ces instructions, il existe des instructions opposées qui ont pour effet d'effacer ce qui a été dessiné. Lorsque METAFONT dessine quelque chose il stocke le résultat dans la variable *currentpicture*. Le contenu de cette variable évolue au fur et à mesure que METAFONT réalise le travail qui lui est demandé. Les deux commandes

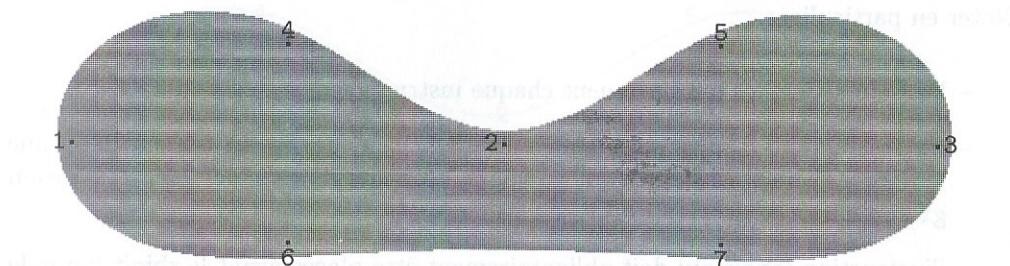


FIG. 4.6 – Instruction fill

conséquent leur valeur modifiée soit par addition soit aussi par soustraction. Voyons par exemple l'instruction **unfill**:

```
fill z1..z4..z5..z3..z7..z6..cycle;
unfill .9[z1,z2].. .1[z2,z3]..cycle;
```

Dans cet exemple, ".9[z1,z2]" désigne un point qui se trouve situé aux neuf dixièmes du segment  $z_1z_2$  tandis que ".1[z2,z3]" désigne son symétrique par rapport à  $z_2$ . Le résultat est montré sur la figure 4.7.

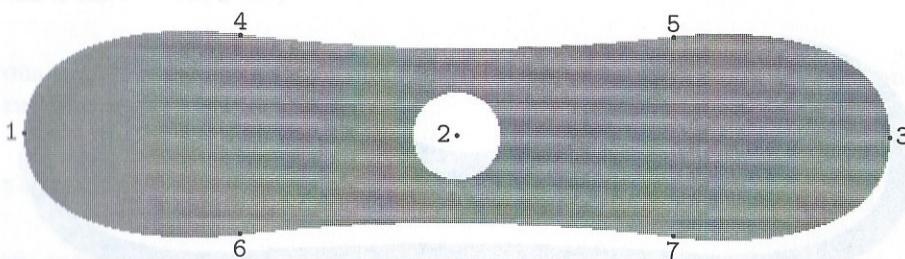


FIG. 4.7 – Instruction unfill

#### 4.1.4 Tangentes imposées

Dans les exemples des figures 4.8 et 4.9, METAFONT a choisi lui-même la forme de la courbe lorsque celle-ci passe par les points de construction. Il est possible d'intervenir dans ce choix et de donner des indications quant à la direction des tangentes. On utilise pour cela les commandes **up**, **down**, **left**, **right** ou bien on donne directement les coordonnées du vecteur tangent comme dans les deux exemples suivants qui correspondent aux figures 4.8 et 4.9 :

```
draw z1{up}..z4..z2..{down}z5..{right}z3;
```

```
draw z1{sqrt(3)/2,1}..z4{(1,-2)}..z2..z5{(-1,1)}..{right}z3;
```

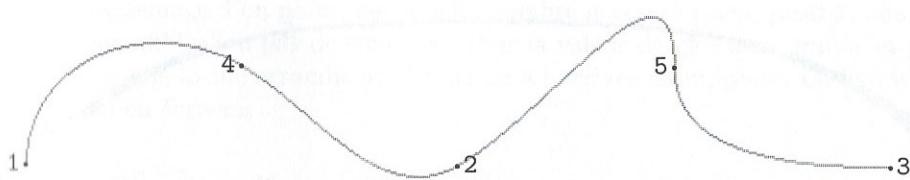


FIG. 4.8 – Tangentes verticales et horizontales

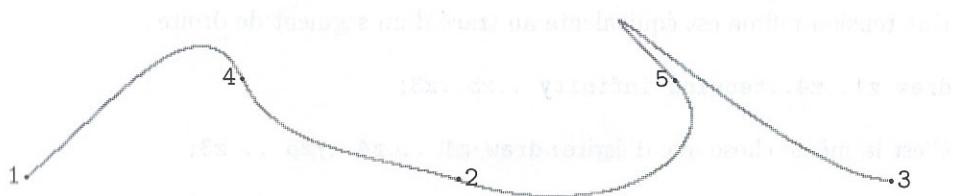


FIG. 4.9 – Tangentes quelconques

Dans le deuxième exemple, la tangente en  $z_1$  est inclinée de  $30^\circ$  par rapport à l'horizontale puisque son vecteur directeur a pour coordonnées  $(\sqrt{3}/2, 1)$ . Il y a une autre façon d'imposer les tangentes, comme il a été dit plus haut: c'est en fournissant les points de contrôle. Il est temps d'en étudier la syntaxe. L'exemple de la figure 4.10 — sur lequel on peut voir que la courbe part dans la direction du vecteur  $z_1 z_4$  et arrive dans celle du vecteur  $z_2 z_5$  — est obtenu de la manière suivante:

```
draw z1.. controls z4 and z2 .. z5;
```

ou encore

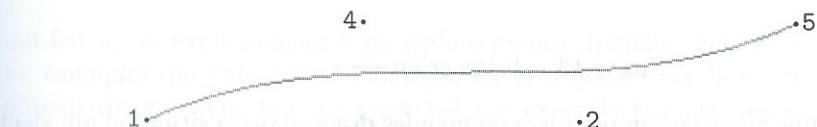


FIG. 4.10 – Instruction controls

#### 4.1.5 Tension

Voici maintenant un autre paramètre qui permet d'influer sur la forme des courbes: c'est la **tension**. Par défaut elle vaut 1. Si on l'augmente, la courbe a tendance à s'aplatisir. Ainsi la courbe de la figure 4.11 est obtenue avec une tension de 1,4:

```
draw z1..z4 tension 1.4 .. z5;
```

courbe  
de 1 inc  
courbe  
été obt

draw z

FIG. 4.11 – Courbe avec une tension de 1,4

Une tension infinie est équivalente au tracé d'un segment de droite:

draw z1..z4..tension infinity ..z5..z3;

C'est la même chose que d'écrire: draw z1 .. z4 --z5 .. z3;

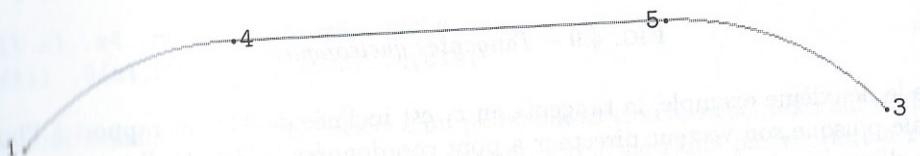


FIG. 4.12 – Instruction -- (équivalente à tension infinity)

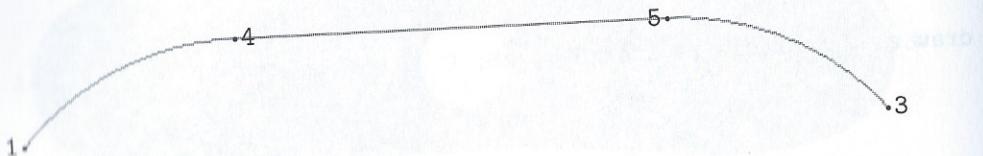


FIG. 4.13 – Instruction ---

Signalons enfin une autre option pour les commandes **draw**: dans la situation qui vient d'être évoquée on peut voir sur la figure 4.12 que l'on a des angles abrupts en chacun des deux points  $z_4$  et  $z_5$ . METAFONT offre la possibilité « d'arrondir les angles » grâce à la commande **---** (trois tirets<sup>3</sup> au lieu de deux):

draw z1 .. z4 --- z5 .. z3;

qui est illustrée par la figure 4.13.

Enfin il faut citer aussi l'instruction **curl** qui est une primitive de METAFONT (et non pas une macro du format *plain*) permettant de donner le degré de courbure d'une

#### 4.1.6

Nous  
nous a  
pour l  
a prév  
figur  
trait  
ce cas  
ajouta

picku

qui fa  
les ex  
obten  
filldr

pick  
fill

Le r  
on re  
la fig  
bien

pick  
drav

3. La définition de **---** est **.. tension infinity ..**

courbe au voisinage d'un point. *curl* est un nombre nécessairement positif : une valeur de 1 indique qu'il n'y a pas de courbure. Plus la valeur de *curl* sera grande et plus la courbe aura une forme arrondie au départ ou à l'arrivée en un point. La figure 4.14 a été obtenue en écrivant :

```
draw z1{curl 1} .. z4 .. {curl 10}z5;
```



FIG. 4.14 – Instruction *curl*

#### 4.1.6 Taille et forme des crayons

Nous ne nous sommes pas préoccupés jusqu'à présent de « l'instrument » avec lequel nous avons dessiné. Or le choix d'un stylo (ou crayon ou pinceau) est déterminant pour le travail d'un calligraphe. Il en va de même pour le métafondeur et METAFONT a prévu les outils indispensables. On aura peut-être remarqué que l'exemple de la figure 4.5 comportait un trait épais tandis que ceux des figures 4.3 et 4.4 avaient un trait fin. Dans le premier cas aucun stylo n'avait été spécifié et METAFONT, dans ce cas, en avait choisi un par défaut. Les deux autres ont été obtenus en réalité en ajoutant une instruction :

```
pickup pencircle;
```

qui fait appel explicitement à un stylo à pointe circulaire fine. Nous allons voir dans les exemples qui vont suivre comment on peut jouer avec la pointe des stylos pour obtenir divers effets. Que ce passe-t-il par exemple si nous combinons l'instruction *filldraw* avec un stylo à point circulaire fine :

```
pickup pencircle;
filldraw z1..z4..z2..z5..z3..z7..z6..cycle;
```

Le résultat, comme on le voit sur la figure 4.15, est différent de celui de la figure 4.6 : on remarque en effet que les points de construction sont cette fois-ci sur le bord de la figure tandis que précédemment la figure débordait au delà de ces points. On peut bien évidemment choisir la grosseur de la pointe en écrivant par exemple :

```
pickup pencircle scaled .3pt;
draw z1..z4..z2..z5..z3..
```

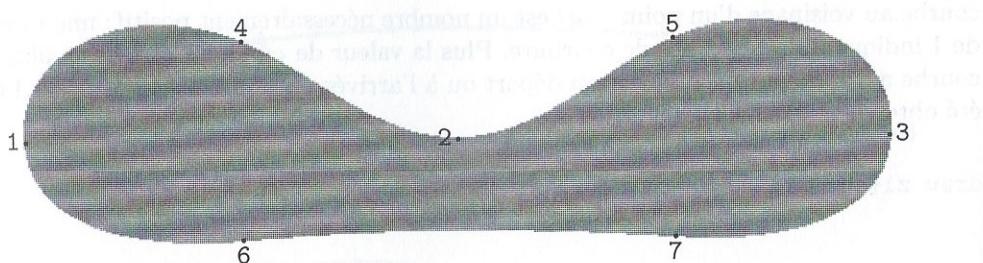
FIG. 4.15 – Instruction `filldraw`

FIG. 4.16 – Taille de la pointe du stylo égale à 0,3 pt

Supposons que nous voulions faire un cercle d'ouverture constant, mais dont la taille de la pointe du stylo varie au cours de la courbe. Nous pouvons alors écrire la commande suivante qui donne la courbe de la figure 4.16.

On obtient un résultat plus raffiné en donnant des tailles différentes en  $x$  et en  $y$ . La figure 4.17 est obtenue en écrivant :

```
pickup pencircle xscaled .3pt yscaled.05pt;
draw z1..z4..z2..z5..z3;
```

FIG. 4.17 – Dimension de pointe différente suivant  $x$  et  $y$ 

Mais il est aussi possible de donner une autre forme à la pointe du stylo : par exemple une forme carrée avec l'instruction `pensquare`. En écrivant :

```
pickup pensquare xscaled .3pt yscaled.1pt;
```

on obtient le tracé de la figure 4.18.

On verra au paragraphe 4.2 qu'il y a aussi une instruction `penrazor` mais nous allons maintenant comment on peut définir soi-même une forme particulière de



FIG. 4.18 – Instruction pensquare

pointe de stylo. On dispose pour cela de l'instruction **makepen**. Il faut définir un chemin cyclique *c* (obligatoirement convexe sinon il sera refusé par METAFONT). Nous allons créer ici un stylo de pointe hexagonale, que nous appellerons *hexapen* avec les instructions suivantes :

```
path c;
c:=(4,2)--(0,4)--(-4,2)--(-4,-2)--(0,-4)--(4,-2)--cycle;
def hexapen = makepen c enddef;
```

Pour conserver ce stylo en mémoire pour un usage ultérieur, on rajoutera la ligne :

```
hexapen:=savepen;
```

On peut voir sur la figure 4.19 les tracés que l'on obtient avec ce nouveau stylo en écrivant :

```
pickup hexapen scaled 4;
draw z1--z2; draw z4--z6;
drawdot z5; drawdot z7;
```

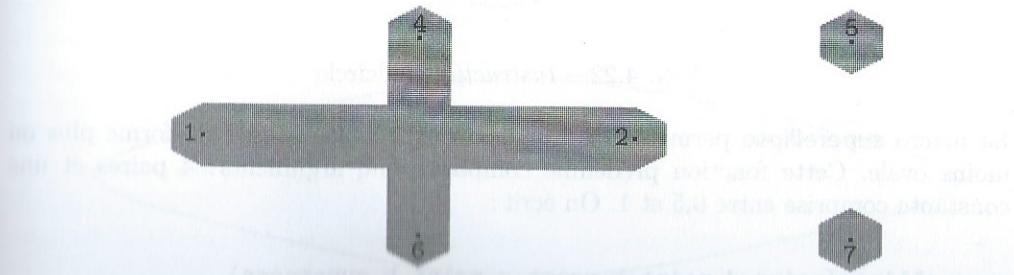


FIG. 4.19 – Pointe de stylo hexagonale

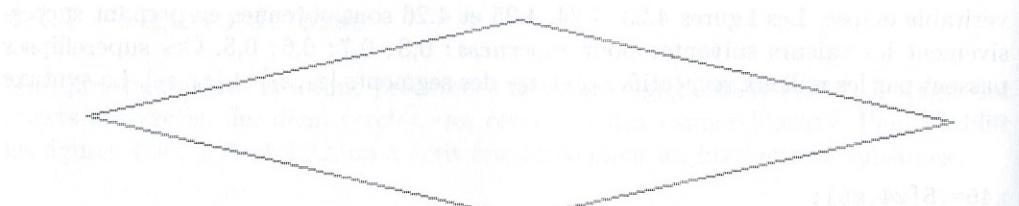


FIG. 4.26 – Superellipse de paramètre 0,5

Il existe enfin une macro peu ordinaire appelée **flex** qui a pour effet de tracer des courbes complexes. L'exemple de la figure 4.27 est obtenu avec :

```
draw flex(z1,z4,z2,z5,z3,z7,z1);
```

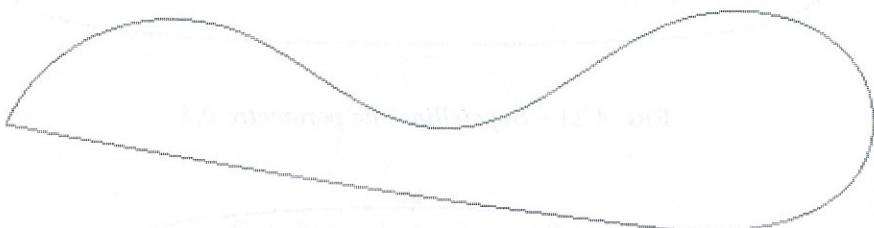


FIG. 4.27 – Instruction flex

La définition exacte de *flex* est donnée au paragraphe 3.1.6.

## 4.2 Calligraphie

La calligraphie, on le sait, est un art et la beauté d'un trait ne dépend pas seulement du matériel utilisé mais également de la façon dont on s'en sert. Ainsi la position de la plume sur la feuille de papier, l'angle d'attaque, la pression, la vitesse sont autant de paramètres sur lesquels joue le calligraphe professionnel. METAFONT est capable de reproduire ces différents aspects et fournit en particulier un procédé et une syntaxe très efficaces pour placer le stylo utilisé sous un certain angle et pour réaliser des pleins et des déliés comme avec une plume et de l'encre. Ce sont essentiellement les commandes **penpos** et **penstroke** qui, utilisées conjointement, permettent d'atteindre ce résultat. La syntaxe a été expliquée à la page 45 et en voici un exemple (voir figure 4.28) :

```
pickup pencircle;
```

```

penpos1(1pt,150);
penpos2(1.5pt,45);
penpos3(1pt,120);
penpos4(.5pt,90);
penpos7(.5pt,60);
draw z1r..z4r..z2r..z7r..z3r--z31..z71..z21..z41..z11--cycle;

```

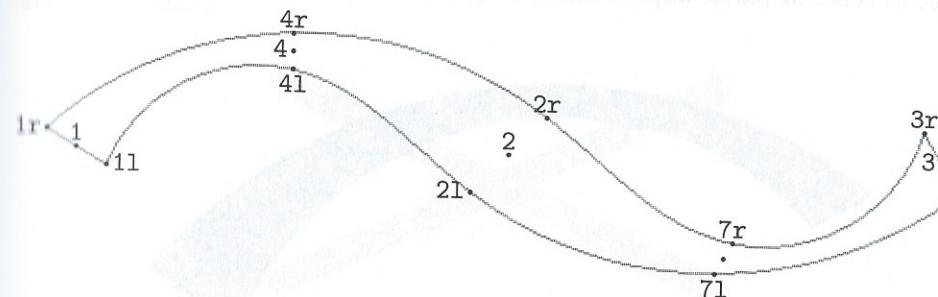


FIG. 4.28 – Instruction penpos avec draw

Les instructions *penpos* indiquent la largeur de la plume et son angle par rapport à l'horizontale<sup>4</sup>.

Ces instructions s'accordent très bien de la commande **filldraw** comme on peut voir sur la figure 4.29 obtenue grâce à :

```
filldraw z1r..z4r..z2r..z7r..z3r--z31..z71..z21..z41..z11--cycle;
```

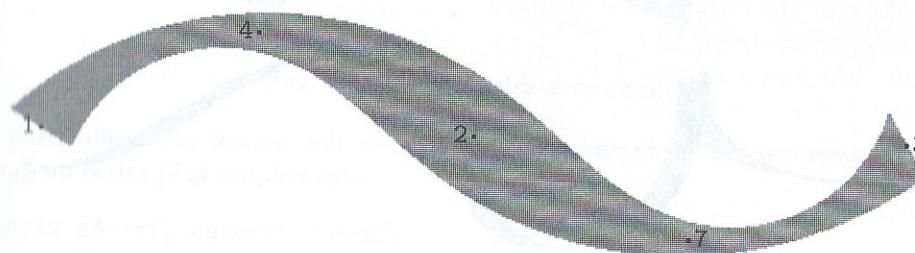


FIG. 4.29 – Instruction penpos avec filldraw

Le défaut de cette méthode est que les chemins sont assez fastidieux à écrire. Il existe une façon plus élégante d'arriver au même résultat : c'est la commande *penstroke*. La figure 4.30 en est une illustration ; elle a été obtenue en écrivant :

```
pickup pencircle;
```

<sup>4</sup> Le marquage des points avec les extrémités gauche *zu* et droite *zd* est obtenu au moyen d'instructions *pickup*.

```
penpos1(1pt,150);
penpos4(.5pt,90);
penpos7(1.5pt,30);
penstroke z1e..z4e..z7e ;
```

L'indice *e* est une abréviation pour *edge*. À noter qu'il est inutile de terminer le chemin par *cycle* : cela fait partie implicitement de *penstroke*.

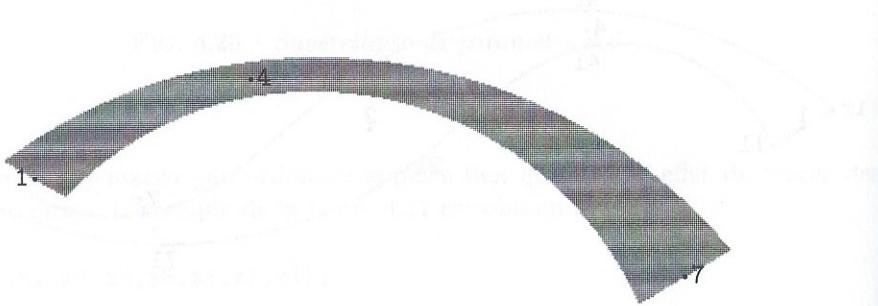


FIG. 4.30 – *Instruction penstroke*

On peut enfin obtenir des terminaisons de traits ayant la finesse du rasoir grâce au stylo intitulé **penrazor** au moyen duquel on réalisera des arabesques comme sur la figure 4.31 obtenue de la manière suivante :

```
pickup penrazor scaled 15;
draw z1{right}..{up}z4{down}..z2{up}..z5..{right}z3;
```



FIG. 4.31 – *Instruction penrazor*

Il est bien entendu possible d'utiliser plusieurs stylos au cours d'un même travail. Chaque stylo, avec ses particularités de forme et de taille de la pointe, peut recevoir un nom et être conservé en mémoire grâce à l'instruction **savepen** (voir au paragraphe 3.3).

Pour terminer cette section, nous illustrerons maintenant une autre façon d'obtenir des terminaisons nettes des tracés. La simple croix de la figure 4.32 est obtenue par :

```
draw z4--z7; draw z5--z6;
```

Le trait est épais (METAFONT a utilisé le stylo par défaut, aucun stylo n'ayant été spécifié) et les extrémités sont arrondies. Dans la figure 4.33, les traits sont coupés net.

```
cutoff(z4,135);
cutoff(z5,45);
cutoff(z7,-45);
cutoff(z6,225);
```

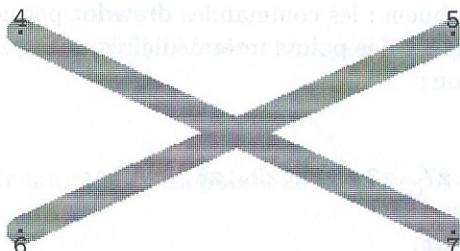


FIG. 4.32 – Extrémités sans cutoff

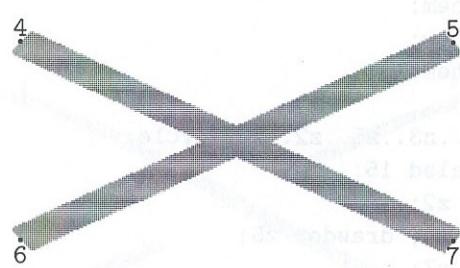


FIG. 4.33 – Extrémités avec cutoff

En particulier, si la section doit se faire *perpendiculairement* aux segments  $[z_4 z_7]$  et  $[z_6 z_7]$ , on écrira plus simplement :

```
cutdown z4--z7; cutdraw z5--z6;
```



FIG. 4.34 – Instruction cutdraw

## 4.3 Chemins et images

### 4.3.1 Cheminement sur les courbes

En dehors des points de construction par lesquels passent les courbes, METAFONT offre la possibilité de parcourir une courbe dans son intégralité. Si  $p$  est un chemin, l'instruction *point  $t$  of  $p$*  désigne un point courant sur la courbe — en supposant que la variable  $t$  varie de 0 à  $n$  où  $n$  est la longueur du chemin. Le programme suivant se comprend aisément ; les commandes **drawdot** permettent de visualiser sur la figure 4.35 l'emplacement des points intermédiaires, tous placés à mi-chemin entre les points de construction :

```
path chem;
chem := z1..z4..z2..z7..z3..z5..z2..z6..cycle;
z14= point .5 of chem;
z42= point 1.5 of chem;
z27= point 2.5 of chem;
z73= point 3.5 of chem;
z35= point 4.5 of chem;
z52= point 5.5 of chem;
z26= point 6.5 of chem;
z61= point 7.5 of chem;
pickup pencircle;
draw z1..z4..z2..z7..z3..z5..z2..z6..cycle;
pickup pencircle scaled 15;
drawdot z1; drawdot z2;
drawdot z3; drawdot z4; drawdot z5;
drawdot z6; drawdot z7;
pickup pencircle scaled 10;
drawdot z14; drawdot z42;
drawdot z27; drawdot z73;
drawdot z35; drawdot z52;
drawdot z26; drawdot z61;
```

En conservant les mêmes points intermédiaires, nous allons quelque peu modifier le programme ci-dessus en introduisant une commande **erase** :

```
pickup pencircle xscaled .4pt yscaled .1pt;
draw z1..z4..z2..z7..z3..z5..z2..z6..cycle;
pickup pencircle xscaled .6pt yscaled .3pt;
erase draw z52..z2..z26;
pickup pencircle xscaled .4pt yscaled .1pt;
draw z1{up}..z4..z2..z7..{up}z3;
```

La commande *erase* à la quatrième ligne efface une partie de l'arc entre  $z_7$  et  $z_4$  d'où la nécessité de retracer cet arc dans la sixième ligne. Le résultat est la figure 4.36. La

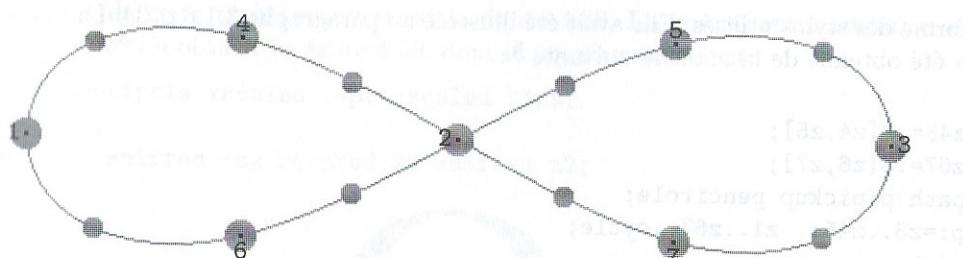


FIG. 4.35 – *Instruction point t of p<sub>1..6,0..1</sub> dessine cette*  
*: u à (0.5,0) depuis un*

figure 4.37 quant à elle matérialise les points de coupure par deux gros points obtenus en rajoutant simplement :

```
pickup pencircle scaled 20;
drawdot z52;
drawdot z26;
```

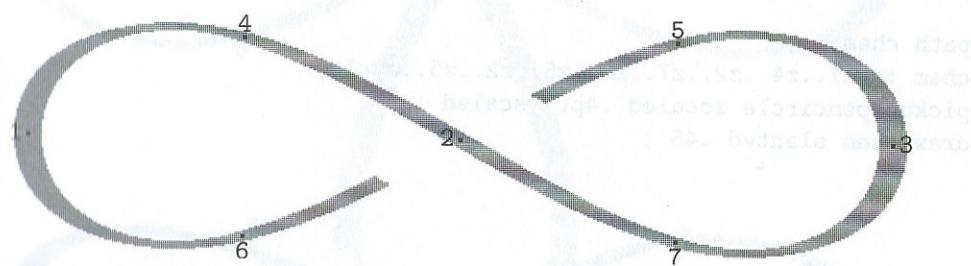


FIG. 4.36 – *Instruction erase*

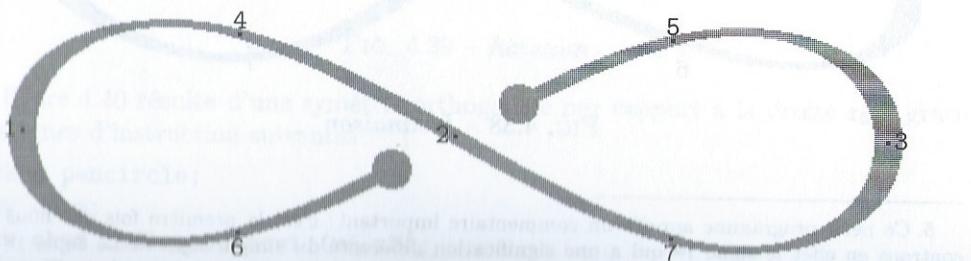


FIG. 4.37 – *Instruction drawdot*

En conclusion, citons l'instruction **subpath** qui aurait pu être utilisée pour obtenir ce genre d'effet mais qui aurait fait une courbure énormément difficile à contrôler.

forme des stylos utilisés. Elle avait été illustrée au paragraphe 3.1.6 p. 34. La figure 3.6 a été obtenue de la manière suivante<sup>5</sup> :

```

z45=.8[z4,z5];
z67=.8[z6,z7];
path p;pickup pencircle;
p:=z3..z45.. z1..z67..cycle;
pickup pencircle;
draw subpath (0,1.33) of p;
draw subpath (1.66,2.66) of p;
draw subpath (3,3.66) of p;

```

### 4.3.2 Transformations

Les transformations que METAFONT est capable d'effectuer sont toutes des composées de transformations linéaires et de translations comme on peut le voir sur la formule de la page 31. Nous allons illustrer maintenant les principales transformations prédéfinies dans le format *plain* : ce sont **slanted**, **rotated**, **reflectedabout**, **rotatedaround** et **shifted**. Nous allons pour cela utiliser la figure introduite dans le paragraphe précédent. C'est un chemin nommé *chem* :

```

path chem;
chem := z1..z4..z2..z7..z3..z5..z2..z6..cycle;
pickup pencircle xscaled .4pt yscaled .1pt;
draw chem slanted .45 ;

```

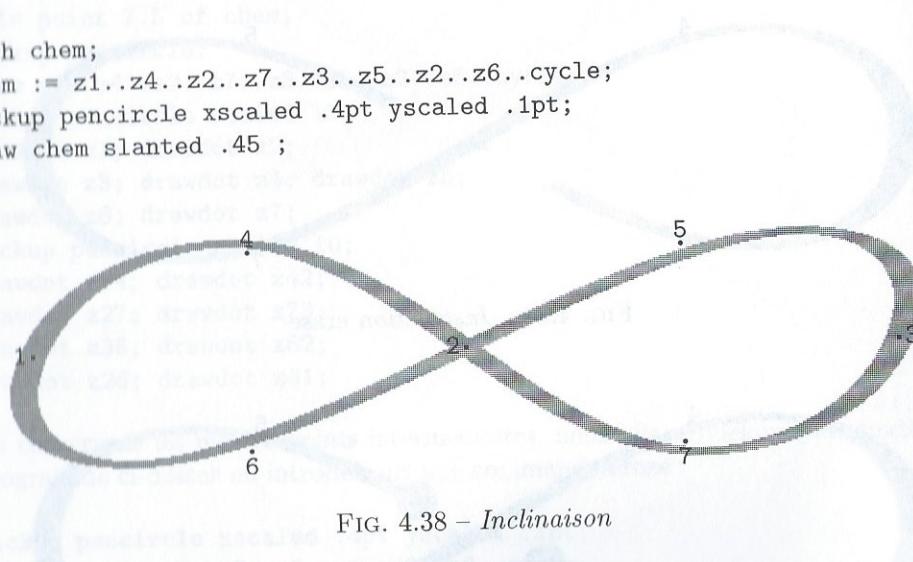


FIG. 4.38 – *Inclinaison*

5. Ce petit programme appelle un commentaire important : c'est la première fois que nous rencontrons en effet le signe `:=` qui a une signification différente du simple signe `=`. Le signe `:=` sert à assigner à une variable une nouvelle valeur qui vient remplacer une ancienne valeur. Si on écrit `a := 1` puis `a := 7` METAFONT comprend que la variable `a` prend d'abord la valeur 1 puis que cette valeur est changée en 7. En revanche si on écrit `a = 1` puis, quelques lignes plus loin, `a = 7` alors METAFONT protestera qu'il a rencontré deux équations contradictoires car une même variable ne peut valoir simultanément 1 et 7. On aura le message d'erreur suivant : ! Inconsistent equation (off by 6).

Le résultat de cette transformation est la figure 4.38. En combinant translation et rotation à 90° on obtient la figure 4.39. dont le programme s'écrit :

```
pickup pencircle xscaled .4pt yscaled .1pt;
draw chem;
draw chem shifted -z2 rotated 90 shifted z2;
```

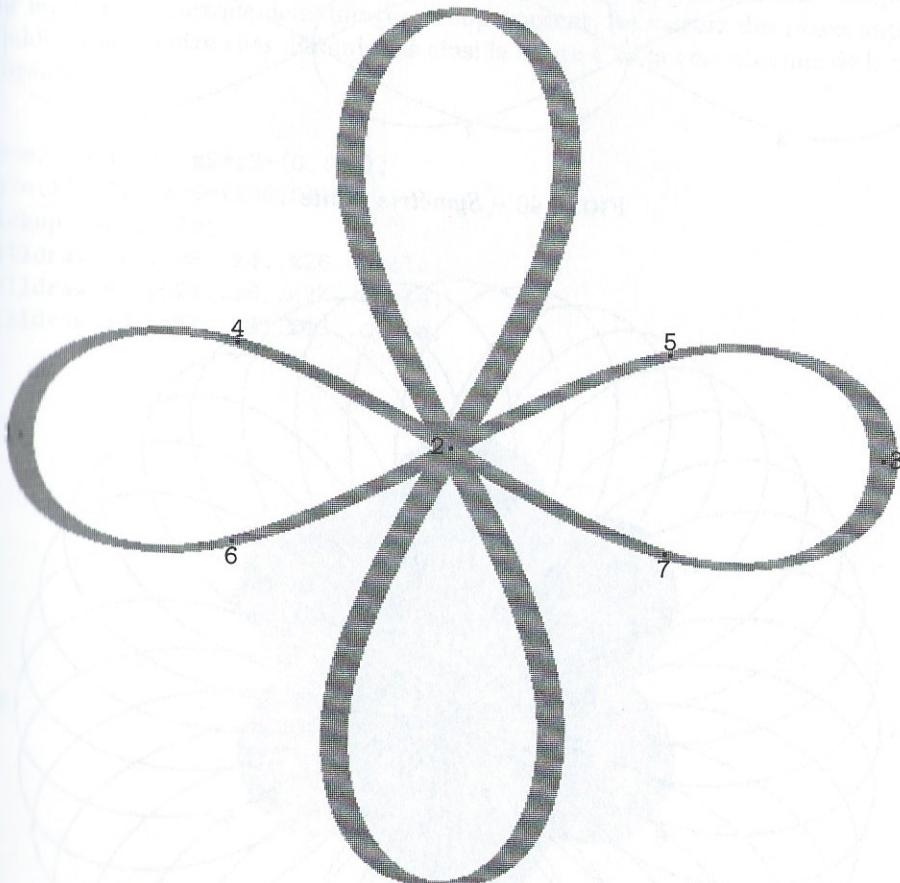


FIG. 4.39 – Rotation

La figure 4.40 résulte d'une symétrie orthogonale par rapport à la droite  $z_5z_7$  grâce aux lignes d'instruction suivantes :

```
pickup pencircle;
draw chem;
draw chem reflectedabout (z5,z7);
```

L'instruction *rotated* effectue une rotation par rapport à l'origine d'où la nécessité de la combiner avec des translations dans l'exemple de la figure 4.39. En revanche l'instruction *rotatedaround* permet de préciser le point autour duquel se fait la rotation. La figure 4.41 a été obtenue en écrivant simplement :

```

pickup pencircle;
for i = 0 step 10 until 170 : draw chem rotatedarround (82,1); endfor

```

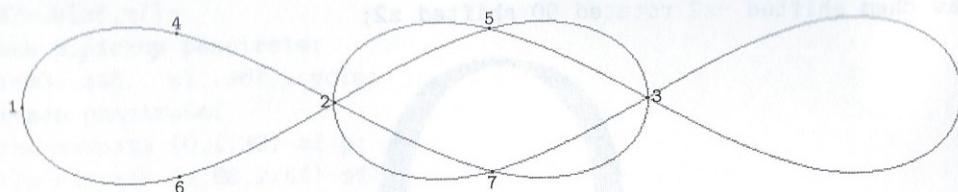


FIG. 4.40 – Symétrie droite

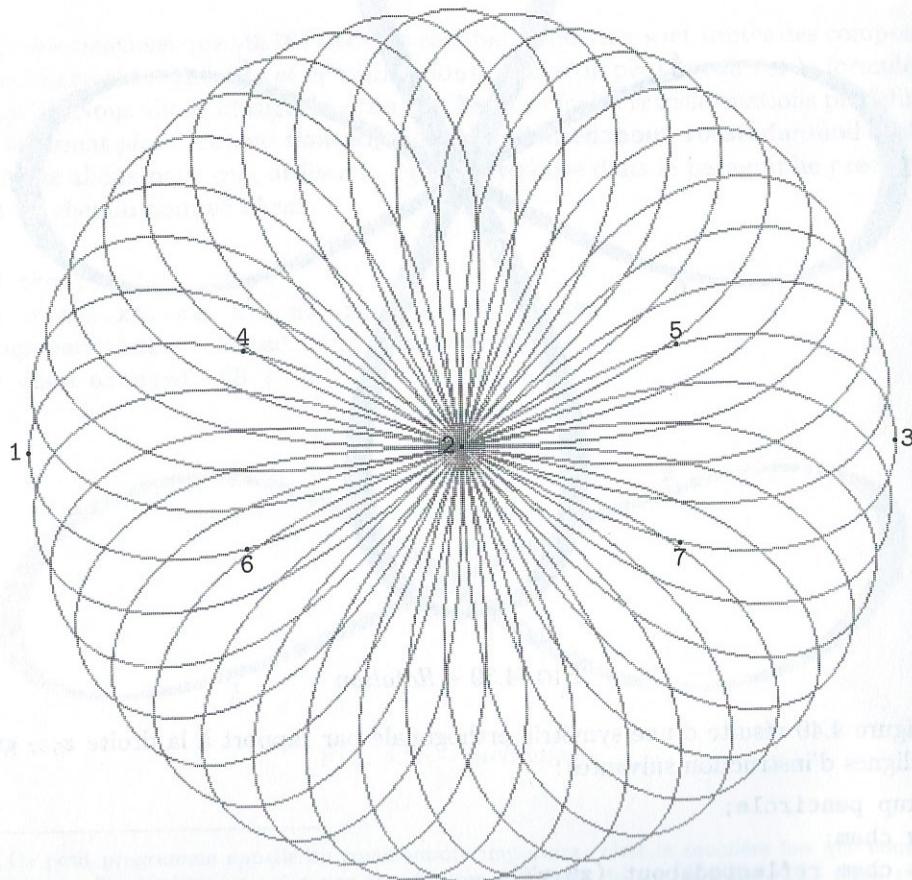
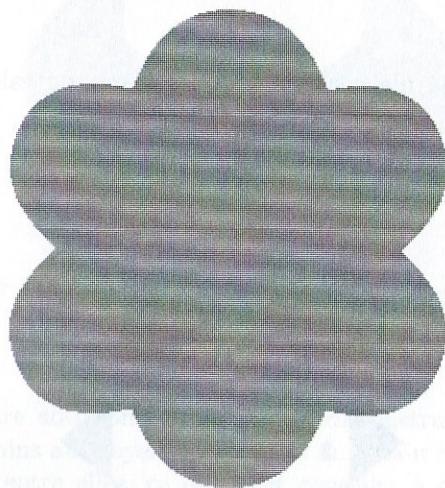


FIG. 4.41 – Instruction rotatedarround

### 4.3.3 Sélection de pixels

Les pixels ont des valeurs entières : si un pixel est négatif ou nul il sera invisible, s'il est au contraire strictement positif il sera matérialisé par un point sur l'écran ou sur l'imprimante. Il n'y a aucune différence pour l'œil entre des pixels de valeur 1 ou 2 mais en revanche on peut à partir de ces valeurs procéder à des manipulations sur les images. Lorsque deux images se superposent, les valeurs des pixels superposés s'additionnent entre elles. Examinons ainsi la figure 4.42 qui est obtenue de la manière suivante :

```
z8=z2+(0,120); z9=z2-(0,120);
z57=(160,0); z46=(280,0);
pickup pencircle;
filldraw z7..z25..z4..z26..cycle;
filldraw z5..z24..z6..z27..cycle;
filldraw z9..z57..z8..z46..cycle;
```



Les pixels peuvent être sélectionnés et éliminés à l'aide d'instructions spéciales (par exemple 3.3) en les chemins d'images. La figure 4.42 illustre l'application de cette instruction dans une application d'images.

FIG. 4.42 – Instruction **cull**

Les pixels qui se trouvent près du centre résultent de la superposition de trois images : ils ont pour valeur trois. Ceux qui se trouvent dans les six branches arrondies ont pour valeur 1 et, entre les deux, il y a des pixels de valeur 2.

L'instruction **cull**, expliquée au paragraphe 3.3, permet de trier ces pixels, d'en sélectionner certains et d'annuler les autres. En écrivant :

```
cull currentpicture keeping (1,1);
```

on obtient l'élimination des pixels différents de 1 (figure 4.43).

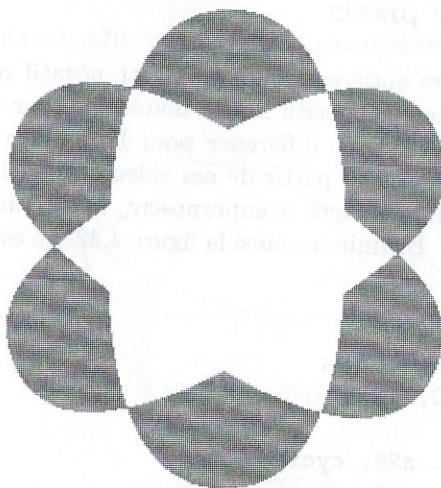


FIG. 4.43 – Résultat de cull keeping (1,1)

Tandis que la figure 4.44 n'a conservé que les pixels de valeur 2 à la suite d'une instruction :

```
cull currentpicture keeping (2,2) ;
```

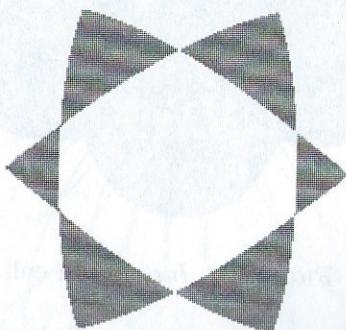


FIG. 4.44 – Résultat de cull keeping (2,2)

et que la figure 4.45 n'a gardé que les pixels égaux à 3 :

```
cull currentpicture keeping (3,3) ;
```

Avec `cull currentpicture keeping (1,2)`, on obtient la figure 4.46.

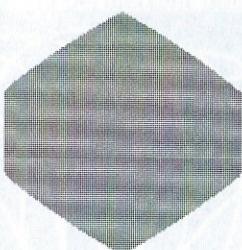


FIG. 4.45 – Résultat de cull keeping (3,3)

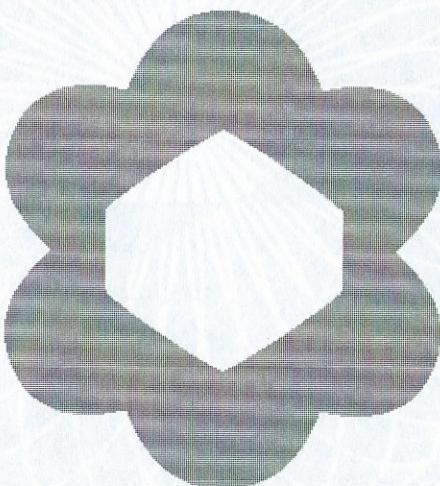


FIG. 4.46 – Résultat de cull keeping (1,2)

Les figures peuvent être additionnées au moyen des instructions **addto** (voir paragraphe 3.3) et les chemins au moyen du symbole **&**. Il faut savoir aussi que l'on peut soustraire des images entre elles, ce qui peut conduire à des effets intéressants de négatifs comme sur la figure 4.47 obtenue comme suit :

```
path chem;
chem := z1..z4..z2..z7..z3..z5..z2..z6..cycle;
picture rosace;
pickup pencircle scaled .07pt;
for i = 0 step 10 until 170 : draw chem rotatedaround (z2,i);
endfor
rosace=currentpicture;
currentpicture=nullpicture;
fill (-10,-220)--(450,-230)--(450,230)--(-10,230)--cycle;
currentpicture:=currentpicture-rosace;
```

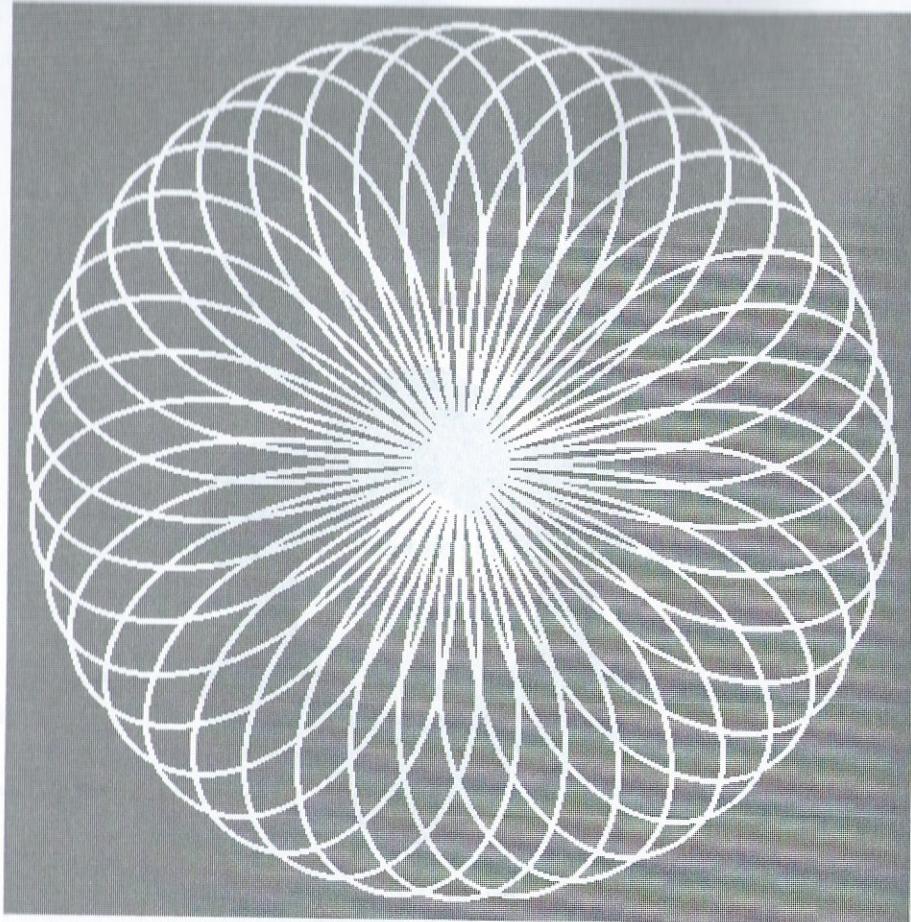


FIG. 4.47 – *Effet de négatif par soustraction*

#### 4.3.4 Manipulation des images

Les principales opérations de combinaison que l'on effectue sur les chemins et sur les images se font avec l'opérateur **&** et l'instruction **addto**. L'opérateur **&** est utilisé pour concaténer des chemins : il ne peut que réunir des chemins contigus ; il faut que l'extrémité du premier chemin coïncide avec l'origine du second. Notons le cas particulier de **cycle** qui peut aussi être considéré comme un chemin et permet d'écrire, étant donnés deux chemins *p* et *q* contigus, des combinaisons telles que :

```
chemin := p & q & cycle;
```

Il y a d'autre part des instructions qui permettent non d'additionner mais de soustraire des figures entre elles. Il s'agit de **erase** qui agit en combinaison avec **fill**, **draw**, **filldraw** et **drawdot**:

```
erase fill p;
erase draw p;
erase filldraw p;
erase drawdot;
```

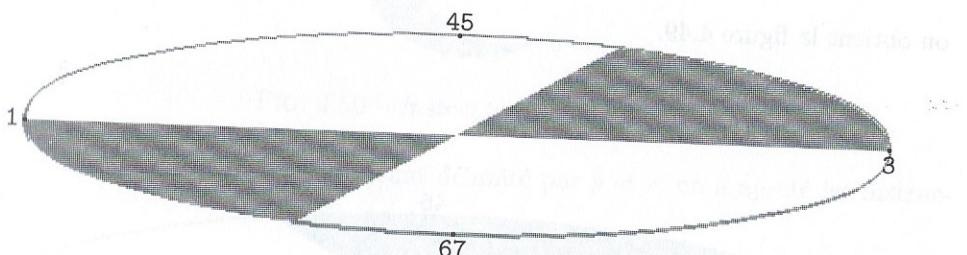


FIG. 4.48 – Addition et soustraction de pixels (1)

L'exemple de la figure 4.48 est obtenu en combinant ces différentes possibilités d'addition, de soustraction et également de sélection de pixels :

```
z45=.5[z4,z5];
z67=.5[z6,z7];
z8=z3 rotatedarround (z2,30);
path p;
pickup pencircle scaled .1pt;
p:= superellipse(z3,z45,z1,z67,.707);

pickup pencircle;
filldraw p;
erase fill z2--z3--z8--cycle;
addto currentpicture also currentpicture rotatedarround (z2,180);
cull currentpicture keeping (1,1);
draw p;
```

Analysons en détail le déroulement de ce programme : les points  $z_{45}$  et  $z_{67}$  sont définis comme les milieux de  $z_4z_5$  et de  $z_6z_7$  respectivement, puis  $z_8$  est obtenu à partir de  $z_3$  par rotation d'angle 30° autour de  $z_2$ .

0,1 pt, METAFONT dessine le pourtour d'une superellipse passant par  $z_3$ ,  $z_{45}$ ,  $z_1$  et  $z_{67}$ . Cette superellipse est remplie à la suite de l'instruction **filldraw** mais le « quart » de forme triangulaire délimité par les points  $z_2$ ,  $z_3$  et  $z_8$  est amputé. Les pixels dans ce quart sont maintenant nuls tandis que dans le reste de la superellipse ils valent 1. La figure est alors pivotée de  $180^\circ$  et ajoutée à elle-même : les pixels s'additionnent. Ainsi les pixels du premier quart et du quart opposé valent maintenant 1 ( $1 + 0$  et  $0 + 1$ ) tandis que les autres valent 2. L'instruction **cull** a pour effet d'annuler tous les pixels différents de 1. Une nouvelle instruction **draw** appliquée remet en place le pourtour de la superellipse.

En modifiant l'instruction *cull* en :

```
cull currentpicture keeping (2,2);
```

on obtient la figure 4.49.

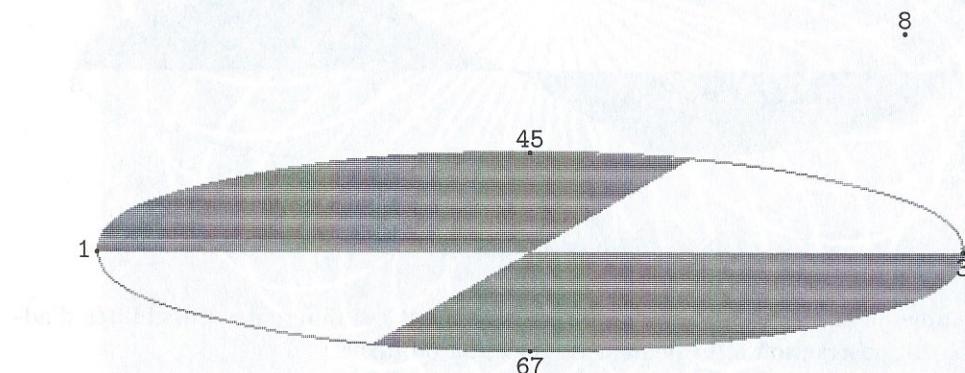


FIG. 4.49 – Addition et soustraction de pixels (2)

METAFONT offre d'autre part des possibilités d'interpolation intéressantes entre des chemins de même longueur. C'est l'instruction **interpath** qui permet de faire des tracés « intermédiaires » à partir de deux chemins donnés par interpolation. La figure 4.50 en servira d'illustration. Voici comment elle a été obtenue : dans un premier temps ont été définis deux chemins  $p$  et  $q$ . Le premier est la courbe passant par les points  $z_4$ ,  $z_1$  et  $z_6$  et le second est le segment qui joint  $z_4$  et  $z_6$  ; ce segment a été défini en réalité comme segment passant par les points  $z_4$ ,  $z_{46}$  et  $z_6$ , où  $z_{46}$  désigne le milieu de  $z_4 z_6$  : cela ne change rien géométriquement mais c'est une astuce pour faire en sorte que  $p$  et  $q$  aient tous les deux la longueur 2 puisque l'instruction *interpath* attend deux chemins de même longueur.  $r$  est le chemin qui interpole les deux chemins  $p$  et  $q$  : la commande **interpath** attend un premier argument numérique qui désigne le coefficient d'interpolation (fixé à 0,25 dans notre exemple). On a ainsi le programme :

```
z46=.5[z4,z6];
```

```
path p,q,r;
```

```
pickup pencircle;
```

```
p=z4..z1..z6;
q=z4--z46--z6;
r=interpath(.25,p,q);
draw p;
draw r;
```

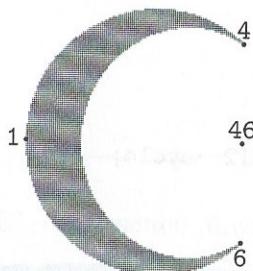


FIG. 4.50 – Instruction interpath

Pour obtenir le remplissage du croissant délimité par  $p$  et  $r$ , on a ajouté les instructions :

```
fill p & reverse r & cycle;
```

On notera d'une part la commande **reverse** — car  $r$  doit repartir de  $z_6$  pour pouvoir employer l'opérateur  $\&$  —, et d'autre part la commande **cycle** — car l'instruction **fill** ne peut s'appliquer qu'à un cycle et que  $p$  & **reverse**  $r$  tout seul n'en est curieusement pas un, bien qu'il s'agisse d'un chemin qui revient à son point de départ.

#### 4.3.5 Points d'intersection et points de contact

Les chemins peuvent se couper et il peut être utile de connaître les coordonnées des points d'intersection. On dispose pour cela d'une instruction **intersectionpoint** utilisée pour réaliser le dessin de la figure 4.51. Celle-ci reprend la même superellipse que dans l'exemple de la figure 4.48 et le point  $z_8$  est obtenu à partir de  $z_3$  par rotation d'angle  $5^\circ$  autour du point  $z_2$ .  $z_9$  est le point d'intersection de la superellipse avec la droite  $z_2z_8$  et la figure est obtenue en remplaçant la superellipse et en lui retirant, au moyen d'une instruction **erase** combinée avec **fill**, une bande horizontale passant par  $z_9$  et par son symétrique par rapport à l'axe horizontal.

Voici le programme de cette figure 4.51 dans lequel on remarquera l'instruction **yscaled -1** qui est une manière astucieuse d'écrire le symétrique par rapport à l'axe horizontal :

```
z8=z3 rotatedarround (z2,5);
path p,q;
```

```

pickup pencircle scaled .1pt;
p:= superellipse(z3,z45,z1,z67,.7);
q:=z2--z8;
z9 = p intersectionpoint q;
z10=(0,ypart z9);
z11 = z10 yscaled -1;
z12 = (xpart z3,ypart z9);
z13 = z12 yscaled -1;

pickup pencircle;
fill p;
erase fill z10--z11--z13--z12--cycle;

```

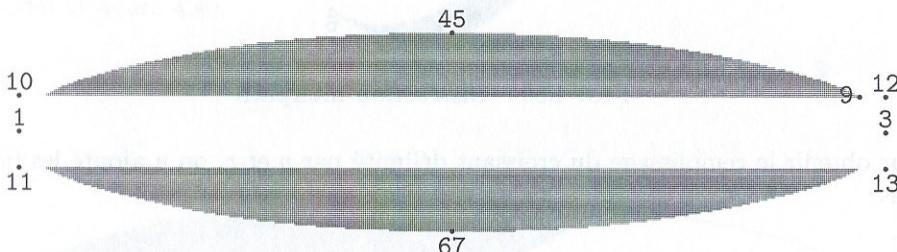


FIG. 4.51 – *Instruction intersectionpoint*

Pour calculer un point de contact d'une courbe avec une de ses tangentes, on utilise l'instruction **directiontime**. Celle-ci indique à quel « moment » un point courant de la courbe aura son vecteur tangent dans une direction donnée. C'est ainsi que l'on a pu trouver sur la figure 4.52 un point *a* situé à la même hauteur que le sommet de la courbe et qui « voit » cette courbe sous un angle donné. Les points *z*<sub>1</sub> et *z*<sub>3</sub> sont les mêmes qui nous servent depuis le début de ce chapitre et le programme devrait maintenant se comprendre aisément :

```

px=.7; py = .6px;
theta= 60 ;
path p;
pickup pencircle;
p:= (z3{up} .. {down}z1 .. cycle) xscaled px yscaled py;
draw p;

t= directiontime (-dir theta) of p;
z4= point t of p;
z5 = z4 shifted (-dir theta * 70);
z1t=point .5 of p;
z6= whatever [z4,z5];
y6=y1t;

```

```

draw p;
t:=directiontime right of p;
k := point t of p;
d:= d+20;
exitif ypart k > 40;
endfor;

```

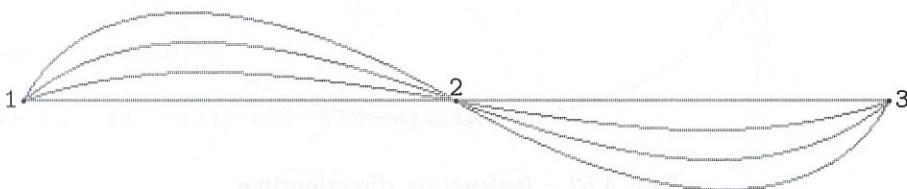


FIG. 4.53 – Instructions forever et exitif

Une autre application des instructions répétitives est maintenant fournie par l'exemple de la figure 4.54 qui utilise également l'instruction **intersectionpoint** pour délimiter des tranches d'angle constant sur une ellipse. La syntaxe est la suivante :

```

path q,p;
pair k,dk;
numeric t;
d:=0;
q:= superellipse(z3,z45,z1,z67,.7);
pickup pencircle;
draw q;
for i = 0 step 20 until 360 :
 k:=z3 rotatedarround (z2,i);
 p:=z2--k;
 dk:= q intersectionpoint p;
 draw z2--dk;
endfor;

```

Cet exemple n'appelle aucune remarque particulière mais c'est une étape pour la construction de la figure 4.55 dans laquelle on a également découpé des tranches triangulaires en en remplies une sur deux. Pour cela une instruction conditionnelle permet de choisir une tranche sur deux en testant une condition arithmétique qui utilise l'opérateur **mod**. Le cycle sera rempli par l'instruction **fill** seulement si l'angle est divisible par 40, c'est-à-dire si son reste dans la division par 40 est nul. On a ainsi :

```

path q,p;
pair k,dk,ek;
numeric t;

```

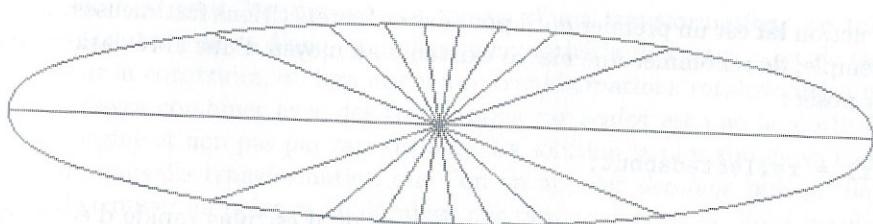


FIG. 4.54 – Découpage répétitif

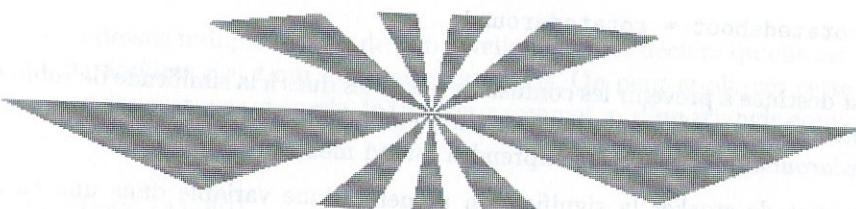


FIG. 4.55 – Exemple de condition arithmétique

```

ek:=z3;
d:=0;
q:= superellipse(z3,z45,z1,z67,.7);
pickup pencircle;
for i = 0 step 20 until 360 :
 k:=z3 rotatedaround (z2,i);
 p:=z2--k;
 dk:= q intersectionpoint p;
 if i mod 40 = 0 : fill z2--ek..dk--cycle; fi
 ek:=dk;
endfor;
```

## 4.4 Usage étendu de METAFONT

### 4.4.1 Crédation de macros

On a pu se convaincre, dans tous les exemples qui précédent, de l'utilité des macros prédéfinies : pourtant celles-ci ne sauraient répondre à tous les besoins rencontrés par l'utilisateur et on est souvent amené, pour éviter les tâches répétitives, à définir soi-même des macros. Pour un travail particulier, on peut même envisager de réunir toutes les macros personnelles dans un même fichier et traiter ce fichier comme une base (voir au chapitre 2).

L'instruction **let** est un premier outil pour éviter les répétitions fastidieuses : il permet, par exemple, de renommer une macro existante au moyen d'une abréviation. Ainsi on pourra poser :

```
let rfl = reflectedabout;
```

L'intérêt de ce changement est tout simplement qu'il est plus rapide d'écrire *rfl* que *reflectedabout*, avec les risques de fautes de frappe que cela comporte. Le format *plain* prévoit la définition amusante qui suit :

```
let rotatedabout = rotatedaround;
```

qui est destinée à prévenir les confusions possibles dues à la similitude de *rotatedaround* et *reflectedabout*. Si jamais l'utilisateur s'embrouille et écrit *rotatedabout* au lieu de *rotatedaround*, METAFONT comprendra quand même !

**let** permet de stocker la signification actuelle d'une variable dans une autre et de redéfinir provisoirement la première. Mais c'est en réalité avec l'instruction **def** que l'on pourra réaliser efficacement de nouvelles macros. Commençons par un exemple simple. Supposons que l'on veuille pouvoir disposer d'une instruction analogue à " --- " afin que les chemins tracés avec **draw** aient systématiquement une tension supérieure ou égale à 1,5. On pourra définir de la manière suivante un symbole " -.- ":

```
def -.- = .. tension atleast 1.5 .. enddef
```

On verra au paragraphe 5.1.1 une macro *beginlogochar* définie dans la police *logo* et qui contient à la fois une instruction pour commencer un nouveau caractère et le choix du stylo pour le tracer. En voici la définition :

```
def beginlogochar(expr code, unit_width) =
beginchar(code,unit_width*u#+2s#,ht#,0);
pickup logo_pen enddef;
```

Le fichier *plain.mf* — qui contient toutes les macros du format *plain* — est construit presque exclusivement à base de définitions. Voici un double exemple :

```
let abs = length;
vardef unitvector primary z = z/abs z enddef;
```

qui définit l'opération **unitvector** associant à tout vecteur *z* un vecteur unitaire de même direction.

Les nouvelles définitions sont surtout intéressantes si elles contiennent des paramètres. Posons le premier problème suivant : étant donnés deux points  $z_1$  et  $z_2$ , comment trouver un point  $z_3$  équidistant de  $z_1$  et  $z_2$  et s'écartant de la droite  $z_1z_2$  d'un angle  $\theta$

donné. La solution peut être trouvée au moyen d'une transformation : on fait une rotation de  $z_2$  autour de  $z_1$  d'angle  $\theta$  puis une homothétie de centre  $z_1$  et de rapport  $1/2 \cos \theta$ . Pour la construire, on fera appel aux transformations *rotatedarround* et *scaled* que l'on devra combiner avec des translations car *scaled* est une homothétie par rapport à l'origine et non pas par rapport à  $z_1$ . La solution la plus simple va consister à définir une nouvelle transformation que l'on va appeler *decalage* puisqu'elle nous permettra de trouver un point  $z_3$  en décalage par rapport à  $z_1$  et  $z_2$ . Voici la solution :

```
def decalage(expr z,t)=
 rotatedarround (z,t) shifted -z scaled 1/(2cosd t) shifted z enddef;
```

La définition ci-dessus indique le nom de la nouvelle macro et déclare qu'elle est fonction de deux paramètres  $z$  et  $t$  qui sont des expressions. On peut appliquer cette nouvelle macro pour trouver, par exemple, le troisième sommet  $z_3$  d'un triangle équilatéral dont on a déjà deux points  $z_1$  et  $z_2$  :

```
z3 = z2 decalage (z1,60)
```

Examinons, à partir de cela, le petit programme suivant :

```
z1=(0,0); z2=(0,200);
z3= z1 decalage (z2,60);
z12 = z2 decalage (z1,-t);
z23 = z3 decalage (z2,-t);
z31= z1 decalage (z3,-t);
pickup pencircle;
filldraw z1--z12--z2--z23--z3--z31--cycle;
end
```

Il conduit, en prenant  $t = 10$ , au symbole de la figure 4.56 ou, en prenant  $t = 20$ , à celui de la figure 4.57.

Les paramètres d'une nouvelle macro peuvent être de trois types: **expr**, **suffix** et **text**. Dans le cas d'un paramètre de type **expr**, l'expression peut être n'importe lequel des objets reconnus par METAFONT : nombre, paire, chemin, image, transformation, etc. La macro *decalage* attend ainsi deux paramètres de type *expr* : un vecteur et un angle, autrement dit une paire et une valeur numérique.

Voici maintenant un exemple, tiré de la police *logo*, de paramètre de type *suffix* dans une macro : dans le préambule du fichier pilote *logo.mf* est définie **super\_half** — macro qui trace une demi superellipse à partir de trois points  $z_i$ ,  $z_j$  et  $z_k$  :

```
def super_half(suffix i,j,k) =
 draw z.i{0,y.j-y.i}
 ... (.8[x.j,x.i],.8[y.i,y.j])(z.j=z.i)
 ... z.j(x.k-x.i,0)
```

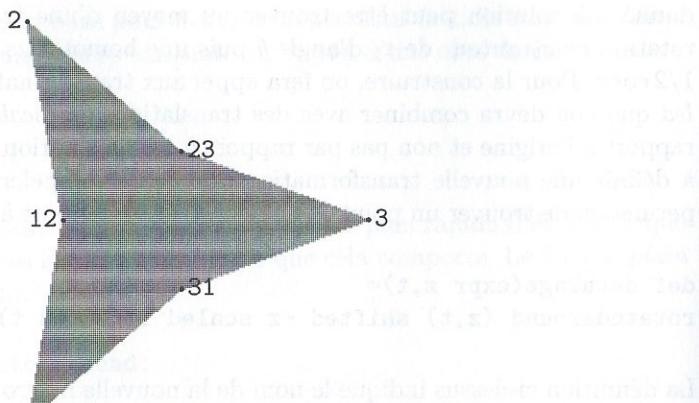


FIG. 4.56 – La macro decalage avec un angle de 10 degrés

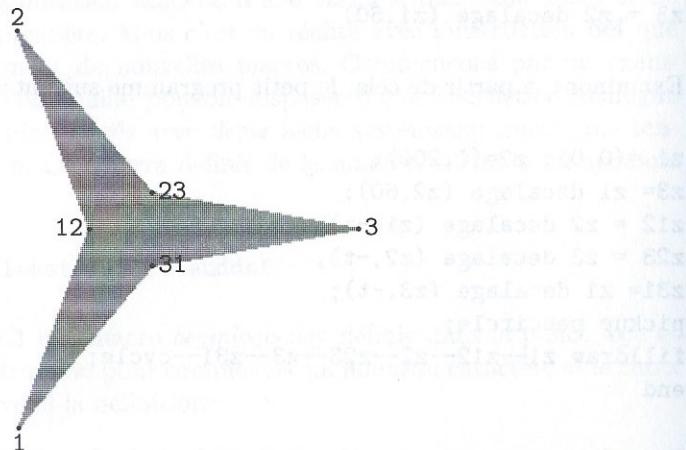


FIG. 4.57 – La macro decalage avec un angle de 20 degrés

Enfin, les paramètres de type *text* permettent de passer un nombre arbitraire (non déterminé à l'avance) de valeurs à une macro. Nous allons construire une macro que nous appellerons **polygone** que l'on utilisera ensuite en écrivant *polygone*(*z*<sub>1</sub>, ..., *z*<sub>*n*</sub>) afin de tracer la ligne brisée (pas nécessairement fermée) qui passe successivement par chacun des points *z*<sub>1</sub> à *z*<sub>*n*</sub>. En voici la syntaxe :

```
def polygone(text t) =
```

```
hide(n_:=0; for z=t: z_[incr n_]:=z; endfor)
z_1 for k=2 upto n_-1: --z_[k] endfor --z_[n_] enddef;
newinternal n_; pair z_[];
```

L'instruction **hide** a pour effet d'isoler (on dit aussi « encapsuler ») une sous-routine qui calcule préalablement le nombre d'éléments de la liste *t* et les charge dans un tableau indexé<sup>6</sup>. En utilisant à nouveau les sept points introduits au paragraphe 4.1.2, on obtient la figure 4.58 en écrivant :

```
pickup pencircle;
draw polygone(z1,z4,z6,z5,z7,z1);
```

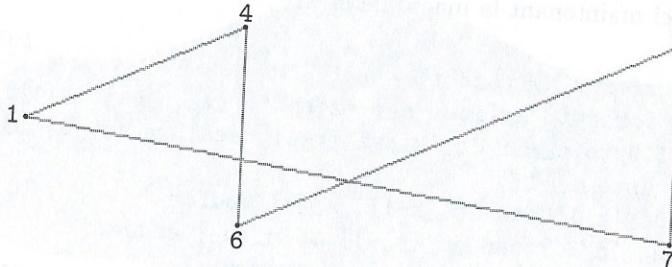


FIG. 4.58 – La macro *polygone*

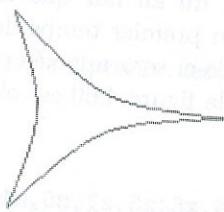


FIG. 4.59 – Contour aux arêtes incurvées

Nous allons maintenant compliquer le problème afin de montrer comment des définitions peuvent s'imbriquer entre elles. L'objectif est d'obtenir facilement des tracés comme celui de la figure 4.59 : un contour dont les arêtes sont déformées ou incurvées. Le principe consiste à construire entre chaque paire de points successifs un point en léger « décalage » qui servira de point de contrôle. Nous allons baptiser **polycreux** cette nouvelle macro et la syntaxe en sera :

```
def polycreux(text t) =
hide(n_:=0; for z=t: z_[incr n_]:=z; endfor; z_[n_+1]=z_[1])
```

<sup>6</sup>. Remarquer que l'instruction **hide** n'a pas à être suivie d'un point-virgule. La paire de accolades constitue un délimiteur suffisant.

```

hide(for k=1 upto n_-1: dc_[k]:=z_[k+1] decalage (z_[k],20);endfor)
z_1 for k=2 upto n_-1:
..controls dc_[k-1] and dc_[k-1]..z_[k] endfor
..controls dc_[n_-1] and dc_[n_-1] .. z_[n_] enddef;
newinternal n_; pair z_[],dc_[];

```

Cette définition fait appel à la macro *decalage* introduite plus haut, avec un angle de 20°. Les points  $dc_k$  sont des points décalés qui sont utilisés comme points de contrôle. Mais on pourrait souhaiter que cet angle soit, lui aussi, une valeur à passer en paramètre : cela ne présente aucune difficulté particulière car on peut combiner des paramètres de type différent dans une même définition. L'angle sera un paramètre de type *expr* tandis que les sommets du « polycreux » sont représentés par une liste de type *text*. Voici maintenant la macro définitive :

```

def polycreux(expr u)(text t) =
hide(n_:=0; for z=t: z_[incr n_]:=z;endfor;z_[n_+1]=z_[1])
hide(for k=1 upto n_-1: dc_[k]:=z_[k+1] decalage (z_[k],u);endfor)
z_1 for k=2 upto n_-1:
..controls dc_[k-1] and dc_[k-1]..z_[k] endfor
..controls dc_[n_-1] and dc_[n_-1] .. z_[n_] enddef;
newinternal n_; pair z_[],dc_[]; numeric u;

```

Il ne faut pas s'étonner de voir que les déclarations de variable figurent à la dernière ligne, après le **enddef** : cela est dû au fait que lorsque METAFONT rencontre une définition, il se contente dans un premier temps de la mettre de côté et n'ira la lire et la développer que lorsque celle-ci sera appelée par la suite dans une instruction à exécuter. À titre d'application, la figure 4.60 est obtenue par l'instruction suivante :

```

pickup pencircle;
filldraw polygone(-10,z1,z4,z5,z3,z7,z6,z1)--cycle;

```

Avec un angle positif de 25°, on obtient un polygone gonflé et non plus déprimé (figure 4.61).

#### 4.4.2 METAFONT solveur d'équations

METAFONT a des facultés de résolution qui ne se limitent pas aux seules équations linéaires (ou systèmes d'équations linéaires). Nous avons vu au paragraphe 4.1.2 qu'on pouvait définir les coordonnées des points au moyen d'équations implicites. Mais nous allons voir maintenant comment obtenir de METAFONT qu'il résolve des équations plus complexes.

Nous allons reprendre le problème posé à la section 4.4.1 : trouver un point  $z_3$  équidistant de deux points  $z_1$  et  $z_2$  et vu depuis ces deux points sous un angle  $\theta$  donné. Il est clair géométriquement que  $z_3$  est le sommet d'un triangle isocèle de base  $z_1 z_2$  et

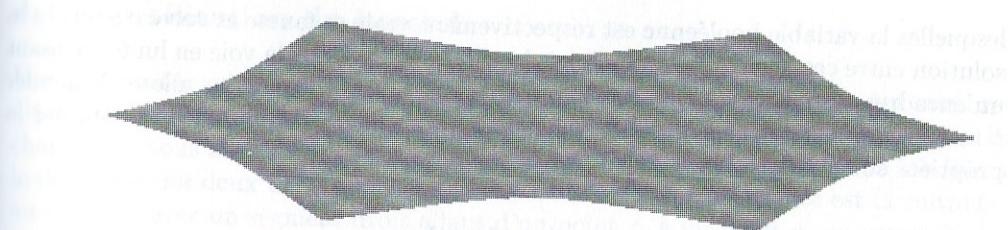


FIG. 4.60 – La macro polycreux avec un angle négatif

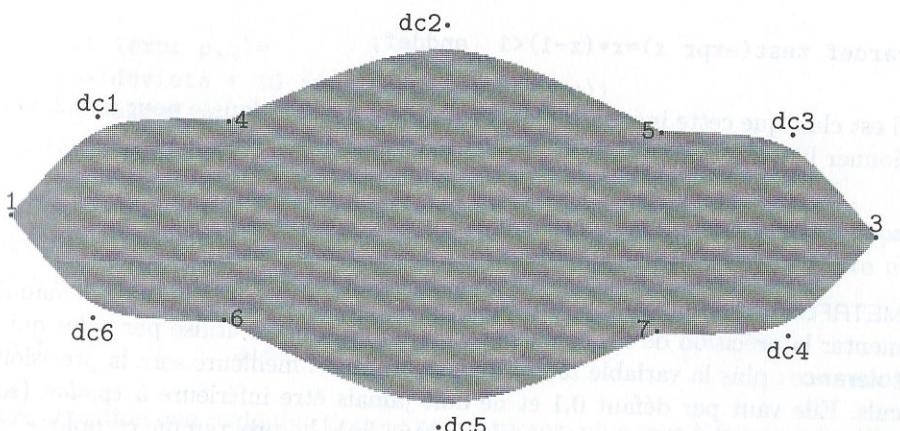


FIG. 4.61 – La macro polycreux avec un angle positif

d'angle à la base  $\theta$ : il appartient donc à la médiane du segment  $z_1 z_2$  c'est-à-dire à la droite perpendiculaire à  $z_1 z_2$  passant par le milieu  $z_4$ . On écrira alors les équations suivantes :

```

z4 = .5[z1,z2]; z5 = z2 rotatedarround(z1,theta);
z3 = whatever[z1,z5];
(z4-z3).dotprod(z2-z1)=0;

```

Ainsi en prenant les deux points  $z_1$  et  $z_2$  qui nous ont servi d'exemple dans tout ce chapitre et un angle  $\theta$  de 5 degrés, METAFONT saura mener les calculs tout seul et donnera comme résultat (110,9.62433).

L'instruction **solve** est d'utilisation plus délicate. Elle est associée à une variable booléenne liée au problème posé : elle fonctionne en quelque sorte comme une routine calculant une valeur limite en dessous de laquelle la variable booléenne sera vraie et au-dessus de laquelle elle sera fausse ; on doit penser à la faire converger vers une valeur

lesquelles la variable booléenne est respectivement vraie et fausse et **solve** cherchera la solution entre ces deux valeurs. On met donc METAFONT sur la voie en lui fournissant un encadrement de la solution à trouver. Prenons un exemple : nous allons demander à METAFONT de calculer la valeur du fameux *nombre d'or*. Celui-ci est défini par la propriété suivante :

$$x(x - 1) = 1$$

On va donc définir une variable booléenne que l'on nommera *test* pour tester la quantité  $x(x - 1)$  :

```
vardef test(expr x)=x*(x-1)<1 enddef;
```

Il est clair que cette inéquation est vraie pour  $x = 0$  et fausse pour  $x = 2$ . On va donc donner l'instruction :

```
solve test(0,2);
```

METAFONT produit comme résultat 1,59375. On va maintenant lui demander d'augmenter la précision de ses calculs grâce à un paramètre utilisé par *solve* qui s'appelle **tolerance** : plus la variable *tolerance* sera petite et meilleure sera la précision des calculs. Elle vaut par défaut 0,1 et ne doit jamais être inférieure à *epsilon* (sous peine de lancer METAFONT dans des calculs sans fin). Le programme complet — avec une tolérance de 0,05 — s'écrirait ainsi :

```
vardef test(expr x)=x*(x-1)<1 enddef;
tolerance:= 0.05;
show solve test(0,2);
end
```

Voici les résultats que l'on obtient pour diverses valeurs de la *tolerance* :

| <i>tolerance</i> | Résultat |
|------------------|----------|
| 0.05             | 1.60938  |
| 0.01             | 1.62111  |
| <i>epsilon</i>   | 1.61804  |

Ce dernier résultat est satisfaisant puisque le nombre d'or vaut en réalité  $(1 + \sqrt{5})/2 = 1,618033989$ .

Un autre exemple, de caractère plus géométrique, d'utilisation de l'instruction *solve* sera donné au chapitre 5 à propos du tracé de la lettre H (voir au paragraphe 5.1.4 p. 129).

### 4.4.3 Utilisation du hasard

Des possibilités amusantes sont offertes par METAFONT par le biais des nombres aléatoires. Les deux macros **uniformdeviate** et **normaldeviate** ont été expliquées au chapitre 3. Nous allons les utiliser pour obtenir des résultats fantaisistes en modifiant la définition des deux tirets " -- " dans les instructions *draw*. L'idée est la suivante : au lieu de tracer un segment droit allant d'un point  $z_i$  à un point  $z_j$  on voudrait que le tracé soit dévié de façon arbitraire. Le milieu  $z_{ij}$  de  $z_i z_j$  sera décalé d'un petit angle choisi au hasard et le tracé ira de  $z_i$  à  $z_{ij}$  puis de  $z_{ij}$  à  $z_j$ . On utilisera donc la commande *decalage* définie au paragraphe 4.4.1 et une nouvelle macro qui sera baptisée *aleatrait* :

```
def aleatrait (expr p,q)=
hide(u:=normaldeviate * 10 ;dc:= q decalage(p,u))
p .. dc .. q enddef ;
pair p,q,dc; numeric u;
```

L'angle  $u$  de décalage est calculé avec *normaldeviate* et aura donc à chaque appel de cet *aleatrait* une valeur différente choisie au hasard. Nous définissons ensuite un opérateur binaire " -- " au moyen de **primarydef** :

```
primarydef p -- q = aleatrait(p,q) enddef;
```

Il faut faire attention que ce double tiret -- ne fonctionne plus tout à fait comme d'habitude : il ne peut lier que deux points entre eux (et pas davantage) car *aleatrait*( $p, q$ ) est un chemin et non une paire. Des instructions telles que :

```
draw z1--z2--z3 ;
```

devront être remplacées par :

```
draw z1--z2; draw z2--z3 ;
```

Cette précaution prise et en appliquant ces nouvelles définitions à toutes les lettres de la police *logo*, voici ce que l'on obtient :

METAFONT

METAFONT

METAFONT

METAFONT

METAFONT

METAFONT

Voici pour terminer un autre exemple amusant d'utilisation des nombres au hasard. Il s'agit de tracer une ligne horizontale d'une main tremblante comme ceci :



ou encore ceci :



Cet exemple va être l'occasion de fabriquer le premier véritable caractère destiné à figurer dans une police et d'introduire toutes les notions qui seront développées dans le chapitre 5. Nous allons attribuer à ce caractère le code 65 qui est celui de la lettre A majuscule. Sa hauteur sera de 6pt, sa largeur de 15pt et sa profondeur de 2pt. Nous utiliserons d'autre part une unité de mesure que nous appellerons  $u$  et qui vaudra 1,5pt. Nous allons découper la largeur du caractère fabriqué en dix sous-intervalles de longueur  $u$  délimités par des points  $z_0$  à  $z_{10}$ . En ayant recours aux nombres au hasard nous allons modifier arbitrairement chacun de ces points  $z_i$  pour en faire des points  $dz_i$  décalés verticalement par rapport aux  $z_i$  et par lesquels passera la ligne à dessiner comme on peut le voir sur la figure 4.62. Il suffit pour cela d'écrire<sup>7</sup> :

```
beginchar(65,10u#,6pt#,2pt#);
pickup pencircle scaled .5pt;
path p;
pair k[],dk[],init;
init:=(0,0);randomseed:=.1954;
k[0]:=init;
for i = 1 upto 10 :
```

7. L'instruction `randomseed` a été ajoutée uniquement pour être sûr d'obtenir le même dessin sur les figures 4.62 et 4.63. En lui passant comme cela une valeur initiale, on force — paradoxalement — le hasard à reproduire le même comportement !

```

k[i]:=k[i-1]+(u,0);
dk[i]:=k[i]+(0,normaldeviate*u/4);
endfor;
p:= k[0] for i = 1 upto 9 : ..dk[i] endfor.. dk[10];
draw p;
endchar;

```

La commande `normaldeviate` nous permet de faire apparaître des décalages aléatoires dans les caractères. On peut alors écrire :

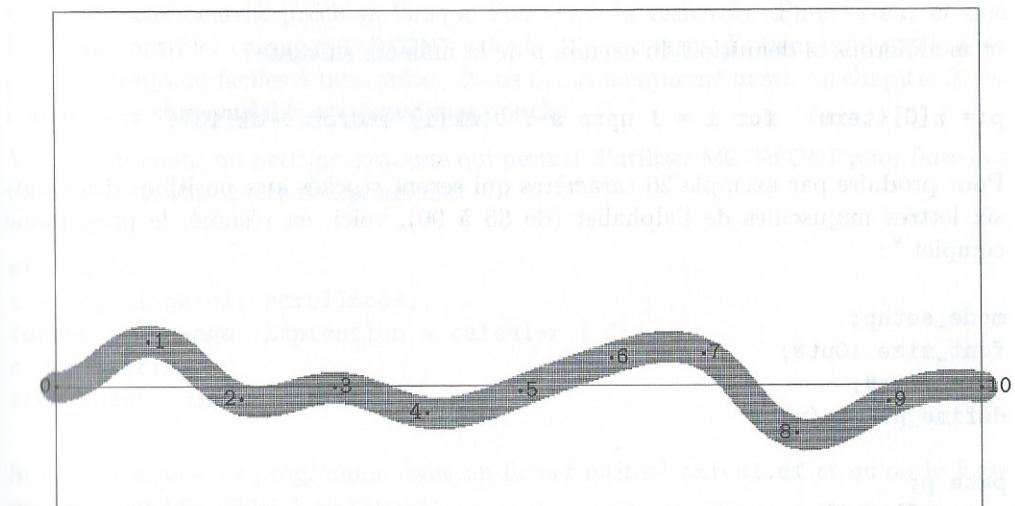


FIG. 4.62 – Points décalés au hasard

On pourra utiliser ce code pour dessiner toutes sortes d'expressions graphiques, mais sans le fonds de la page qui appelle à l'évidence une fonction de type `point[i]` pour chaque caractère. Mais, en termes d'effort, il n'y a rien de plus simple que de faire un caractère à la main et de le modifier par un programme informatique pour ajouter des courbes d'ordre quelconque.



FIG. 4.63 – Ligne ondulée par le hasard

Maintenant, nous allons un peu compliquer le problème, car nous voudrions produire une trentaine de ces caractères, tous différents puisque dessinés au hasard, afin de les

mettre bout à bout pour obtenir une ligne tremblée comme ci-dessus. Il faut donc que ces caractères puissent se raccorder autrement dit que le tracé d'un caractère commence au point où le précédent avait terminé tout en partant dans la même direction. C'est à cela que vont servir les variables `init` (pour point initial) et `term` (pour direction terminale) ; nous ajouterons donc les deux lignes ;

```
term:=direction 10 of p;
init:=(0,ypart dk[10]);
```

et modifierons la définition du chemin *p* de la manière suivante :

```
p:= k[0]{term} for i = 1 upto 9 : ..dk[i] endfor.. dk[10];
```

Pour produire par exemple 26 caractères qui seront stockés aux positions des vingt-six lettres majuscules de l'alphabet (de 65 à 90), voici, en résumé, le programme complet<sup>8</sup> :

```
mode_setup;
font_size 10pt#;
u#:=1.5pt#;
define_pixels(u);

path p;
pair k[],dk[],init,term;
term:=right;
init:=(0,0);
code = 64;

for j = 1 upto 26:
beginchar(incr code,10u#,6pt#,2pt#);
pickup pencircle scaled .5pt;
k[0]:=init;
for i = 1 upto 10 :
k[i]:=k[i-1]+(u,0);
dk[i]:=k[i]+(0,normaldeviate*u/4);
endfor;
p:= k[0]{term} for i = 1 upto 9 : ..dk[i] endfor.. dk[10];
draw p;
term:=direction 10 of p;
init:=(0,ypart dk[10]);
endchar;
endfor;
end
```

<sup>8</sup> Les instructions `mode_setup`, `font_size` et `define_pixels` seront expliquées en détail au chapitre 5.

#### 4.4.4 Faire parler METAFONT

METAFONT est un langage évolué qui peut nous renseigner de diverses façons sur les commandes qu'il est en train de traiter ou sur les grandeurs qu'il est en train de manipuler. Nous avons déjà mentionné les commandes *tracing* répertoriées à l'annexe C et qui demandent à METAFONT de nous renseigner automatiquement au cours de chaque session sur tel ou tel aspect des calculs qu'il exécute. Il y a aussi et surtout l'instruction **show** qui permet à l'utilisateur d'interroger directement METAFONT. Cela est extrêmement précieux lorsque l'on est à la recherche d'une erreur et que l'on veut contrôler ce que METAFONT calcule. Les messages d'erreur qu'il produit ne sont pas toujours faciles à interpréter. Nous avons mentionné aussi, au chapitre 3, les commandes **showvariable** et **showdependencies**.

Voici maintenant un petit programme qui permet d'utiliser METAFONT pour faire des calculs sur toutes sortes d'expressions :

```
string s;
tracingonline:=1; scrollmode;
forever: message "Expression a calculer : ";
s:=readstring;
show scantokens s; endfor
```

Si on sauvegarde ce programme dans un fichier intitulé **calcul.mf** et qu'on le fasse exécuter par METAFONT en tapant :

```
mf calcul
```

on pourra passer à METAFONT toutes sortes d'expressions dont il donnera le résultat ou bien pour lesquelles il répondra, s'il y a lieu, par un message d'erreur.

Ainsi, en tapant  $(1+\sqrt{5})/2$ , on obtiendra 1,61804 qui est, une fois de plus, la valeur du nombre d'or !

## Chapitre 5

# Construire une métapolice

Le but de METAFONT est essentiellement de fabriquer des polices de caractères. En plus des outils de dessin et des facilités de programmation présentés au chapitre 4, il comporte tous les outils nécessaires à la création à la manipulation des polices et des caractères à l'intérieur d'une police.

Toutes ces notions vont être présentées à travers un travail de construction d'une police complète et même d'une « métapolice » comme nous l'expliquerons par la suite : cette idée de métapolice<sup>1</sup> repose sur le principe qui veut que les caractères d'une police soient tous définis de façon générique et que les différentes variantes de la police soient toutes obtenues à partir d'un fichier générique, en lui passant diverses valeurs de paramètres.

### 5.1 Construction d'une police

Le concept de métapolice ne s'arrête pas à la seule obtention de caractères romains, italiques ou gras car nous verrons que l'on peut obtenir de la même façon des caractères avec ou sans empattement (comme dans les polices Times ou Helvetica respectivement), des caractères à espacement fixe du type machine à écrire (comme dans la police Courier), etc.

Nous partirons pour cela de la police logo qui est la police créée par D. Knuth pour servir d'exemple aux notions principales de METAFONT et qui précisément sert à écrire le mot METAFONT : ce mot n'ayant que sept lettres distinctes, la police logo n'est constituée que de ces sept lettres majuscules : A E F M N O T. On ne va évidemment pas très loin avec seulement sept lettres : TOMATE ET ATOME. En réalité la police logo contient en plus les lettres S et P qui y ont été ajoutées par D. Knuth lui-même depuis qu'existe le programme Metapost de John Hobby, langage analogue à METAFONT mais qui produit du code PostScript : nous allons donc compléter cet

alphabet incomplet et construire une véritable métapolice. Le présent chapitre sera consacré entièrement à cette tâche, qui sera l'occasion de passer en revue tous les concepts illustrés au chapitre 4 et de les compléter.

Il faut tout d'abord faire acte de modestie en soulignant d'emblée que la fabrication d'une police de caractères, et *a fortiori* d'une famille de polices, est un art qui requiert un grand savoir-faire : la typographie — à l'instar de tous les métiers graphiques — exige des connaissances que le présent ouvrage ne prétend en aucun cas enseigner. La police que nous allons construire se veut donc résolument plus pédagogique que belle, l'aspect didactique l'emportant de loin sur l'aspect esthétique. D'un autre côté, la connaissance de METAFONT permet de répondre, sans être soi-même un Garamond<sup>2</sup> ou un Didot<sup>3</sup>, aux besoins que peut rencontrer un auteur au cours de la rédaction d'un travail : besoin d'un symbole particulier ou de familles de caractères spéciaux que l'on peut créer en un temps minime, nécessité de modifier une police déjà existante, fabrication d'un logo, etc. METAFONT permet de fournir des solutions très rapides en même temps qu'il est capable de satisfaire à la plus haute exigence en matière de qualité.

### 5.1.1 Créer des caractères

Nous partirons donc du fichier `logo.mf` qui contient les lettres A E F M N O P S T mais nous allons le rebaptiser `logocomplet.mf` puisqu'il va être considérablement modifié et augmenté. Nous appellerons d'ailleurs dorénavant la nouvelle police **logocomplet**. Elle utilise, au départ, un nombre restreint de paramètres :

1. `ht#` est la hauteur de chaque caractère ;
2. `u#` est une unité de mesure adoptée pour exprimer la largeur de chaque caractère ;
3. `s#` est la taille d'un espace supplémentaire ajouté à gauche et à droite de chaque caractère ;
4. `o#` est le débordement des caractères par rapport aux limites inférieure et supérieure de la boîte qui délimite chaque caractère. `ho#` représente une quantité analogue horizontalement ;
5. `xgap#` est la différence de largeur entre la barre horizontale médiane et la barre supérieure des lettres E et F. `ygap#` représente la position du point central de la lettre M par exemple ou mesure de combien descend le trait oblique de la lettre N ;
6. `px#` est l'épaisseur horizontale du stylo. L'épaisseur verticale `py#` lui est liée ( $py = 0,9px$ ) ;

2. Claude Garamond (1499-1561), créateur des caractères qui portent son nom et sont considérés comme le point de départ de la typographie moderne.

3. François-Ambroise Didot (1730-1804), créateur des caractères qui portent son nom et de la

7. `leftstemloc#` est l'abscisse de la barre verticale des lettres D E F P R, etc. (elle est fixée à  $2,5u+s$ ) ;
8. `barheight#` est un paramètre qui fixe la hauteur des barres horizontales dans les lettres A E F P R ;
9. `slant` est un paramètre numérique qui désigne le degré d'inclinaison des caractères obliques.

Voici, par exemple, les valeurs assignées à ces paramètres<sup>4</sup> pour le corps 10 :

```
ht#:=6pt#;
xgap#:=0.6pt#;
u#:=4/9pt#;
s#:=0;
o#:=1/9pt#;
px#:=2/3pt#;
ygap#:=(ht#/13.5u#)*xgap#;
ho#:=o#;
leftstemloc#:=2.5u#+s#;
barheight#:=.45ht#;
py#:=.9px#;
slant:=0;
```

### 5.1.2 Organisation du fichier pilote

Nous allons commencer par passer en revue l'organisation d'un fichier pilote pour une police de caractères. Un fichier pilote comporte habituellement les éléments suivants :

1. des commentaires explicatifs (les commentaires comportent le signe % en début de ligne) ;
2. des déclarations éventuelles de bases à charger en plus de la base *plain*, autrement dit des ensembles de macros spécialement écrites en fonction des besoins ou des habitudes du créateur ;
3. l'instruction **mode\_setup** ;
4. des déclarations de paramètres ;
5. les instructions concernant la pixellisation des paramètres ;
6. des macros éventuelles (`def... enddef`) définies pour la circonstance<sup>5</sup> ;

4. Ces valeurs ont déjà été rencontrées au paragraphe 5.2.2.

5. Si elles ont un caractère plus universel, on en fera plutôt une base à part comme il a été dit plus haut.

7. les définitions des caractères un par un (`beginchar..endchar`) ;
8. les instructions concernant les ligatures de deux caractères et les approches de paires ;
9. les paramètres globaux de la police :
  - `font_quad`
  - `font_normal_space`
  - `font_normal_stretch`
  - `font_normal_shrink`
  - `font_identifier`
  - `font_coding_scheme`

Comme il est expliqué au chapitre 2 (paragraphe 2.2.4), les paramètres sont souvent réunis dans un fichier à part, dit fichier de paramètres, qui fixe les valeurs de ces derniers et fait ensuite appel au fichier pilote. Nous verrons cela en détail au paragraphe 5.2.2 p. 143. Le choix des paramètres est déterminant pour avoir suffisamment de possibilités de modification. Il est aussi possible de répartir les définitions des caractères dans plusieurs fichiers séparés qui seront appelés par des instructions `input` afin de ne pas avoir des fichiers trop lourds à gérer. C'est le cas de la police cmr répartie entre les fichiers romanu.mf, romanl.mf, greeku.mf, romand.mf, romanp.mf, romspl.mf, romspu.mf, punct.mf, accent.mf.

### 5.1.3 Préambule du fichier pilote

Le fichier pilote commence par la commande `mode_setup`: comme son nom l'indique, cette commande a pour effet de faire les réglages de mode, autrement de fixer les valeurs des variables particulières — telles que `pixels_per_inch`, `blacker`, `fillin`, `o_correction` — en fonction du mode choisi. Rappelons que, quand on lance METAFONT sur un fichier particulier de police, on doit tout d'abord indiquer le nom de mode: la commande `mode_setup` va chercher les valeurs qui se trouvent fixées dans la définition du mode telle qu'on peut la trouver dans le fichier `modes.mf`. Ce fichier a été compilé avec le format *plain* et METAFONT a donc en réalité tous les renseignements nécessaires en mémoire. On trouve d'autre part en tout début de fichier une ligne d'instruction concernant la commande `currenttransform` expliquée en détail au paragraphe 5.2.1 p. 142.

Voici donc, *in extenso*, le début du fichier `logocomplet.mf`<sup>6</sup>:

```
mode_setup;
if unknown slant: slant:=0 else: currenttransform:-
 identity slanted slant yscaled aspect_ratio fi;
```

Suivent ensuite des déclarations globales de variables (*boolean*, *numeric*, *pair*, *path*, *pen*, *picture*, *string* et *transform*) et de paramètres: des paramètres, cette fois, que

---

6. Le fichier de paramètres avec les valeurs de paramètres du corps 10 s'appellera dorénavant `logocomplet10.mf`.

l'on n'aura pas souhaité isoler dans un fichier de paramètres car on n'entend pas les rendre modifiables :

```
ygap#:=(ht#/13.5u#)*xgap#;
ho#:=o#;
leftstemloc#:=2.5u#+s#;
barheight#:=.45ht#;
py#:=.9px#;
```

Viennent ensuite les instructions de conversion des variables diésées en nombre de pixels. Ces conversions utilisent précisément les valeurs correctives que **mode\_setup** a établies. Voici à nouveau la liste des instructions disponibles dans le format *plain*, chacune correspondant à un type particulier de variable (comme expliqué au paragraphe 3.5 p. 50) :

```
define_pixels
define_whole_pixels
define_whole_vertical_pixels
define_blacker_pixels
define_good_x_pixels
define_good_y_pixels
define_corrected_pixels
define_horizontal_corrected_pixels
```

Ces instructions ont pour effet de transformer toutes les grandeurs et toutes les valeurs de paramètres d'unités diésées en nombres de pixels. Voici quelles sont les instructions de pixellisation dans le fichier *logo.mf* :

```
define_pixels(s,u);
define_whole_pixels(xgap);
define_whole_vertical_pixels(ygap);
define_blacker_pixels(px,py);
define_good_x_pixels(leftstemloc);
define_good_y_pixels(barheight);
define_corrected_pixels(o);
define_horizontal_corrected_pixels(ho);
```

Les lettres de la police *logocomplet* sont tracées au moyen d'un stylo unique baptisé *logo\_pen*. C'est un stylo à pointe elliptique, plus large dans le sens horizontal que vertical afin de faire des traits plus épais verticalement qu'horizontalement. Il est défini comme suit :

```
pickup pencircle xscaled px yscaled py;
logo_pen:=savepen;
```

On trouve ensuite quelques définitions de macros sur lesquelles nous reviendrons :

- **beginlogochar** qui redéfinit la commande *beginchar* et dont la définition a été donnée au paragraphe 4.4.1 p. 97 ;

- **super\_half** qui permet de tracer des moitiés de superellipse et dont la définition a été donnée au paragraphe 4.4.1 p. 99 ;
- **super\_crescent** qui trace des arcs en forme de croissant et dont la définition est très similaire à celle de *super\_half* :

```
def super_crescent(suffix i,j,k) =
 draw z.i{x.j-x.i,0}
 ... (.8[x.i,x.j],.8[y.j,y.i]) {z.j-z.i}
 ... z.j{0,y.k-y.i}
 ... (.8[x.k,x.j],.8[y.j,y.k]) {z.k-z.j}
 ... z.k{x.k-x.j,0} enddef;
```

Enfin, une fois ce préambule achevé, commencent les routines de définition des caractères eux-mêmes.

### 5.1.4 Définitions des lettres

Les instructions définissant chaque caractère sont introduites par la commande *beginchar*. Étant donné que tous les caractères de cette police logocomplet auront même hauteur, même profondeur et seront tracés par un stylo unique, la commande *beginchar* a été remplacée par une commande *beginlogochar* définie comme suit :

```
def beginlogochar(expr code, unit_width) =
 beginchar(code,unit_width*u#+2s#,ht#,0);
 pickup logo_pen enddef;
```

Cette commande n'attend que deux paramètres (le code du caractère et la largeur de la lettre exprimée en multiple de l'unité u#) et incorpore l'instruction de choix du stylo. Il faut savoir qu'à chaque nouveau caractère le stylo courant (*currentpen*) est réinitialisé (à la « valeur » *nullpen*) et qu'il faut à nouveau le définir : grâce à la macro *beginlogochar* cela se fait automatiquement.

Nous allons maintenant regarder la définition de chacune des vingt-six lettres de la police logocomplet en commençant par celles du mot METAFONT dont la définition est due, on le rappelle, à D. Knuth. Les plus simples sont E et F :

```
beginlogochar("E",14);
x1=x2=x3=leftstemloc;
x4=x6=w-x1+ho;
x5=x4-xgap;
y1=y6; y2=y5; y3=y4;
bot y1=0;
top y3=h;
y2=barheight;
draw z6--z1--z3--z4;
```

```
draw z2--z5;
labels(1,2,3,4,5,6);
endchar;
```

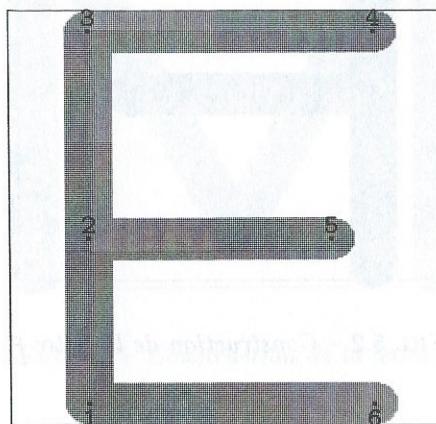


FIG. 5.1 – Construction de la lettre *E*

```
beginlogochar("F",14);
x1=x2=x3=leftstemloc;
x4=w-x1+ho;
x5=x4-xgap;
y2=y5; y3=y4;
bot y1=-o;
top y3=h;
y2=barheight;
draw z1--z3--z4;
draw z2--z5;
labels(1,2,3,4,5);
endchar;
```

Ces définitions se comprennent sans difficulté et sont illustrées sur les figures 5.1 et 5.2 respectivement. Nous voyons qu'il y a en fait deux façon d'attribuer un code à un caractère, c'est-à-dire une position dans la police : on peut le faire en déclarant une valeur numérique mais aussi, comme c'est le cas ici, en indiquant, entre guillemets, le nom de la lettre. Dans ce dernier cas, METAFONT sait où placer le caractère : il se réfère à l'encodage-type de la police cmr qui est donné à l'annexe E dans le tableau E.1. C'est l'encodage connu de TeX sous le nom d'encodage OT1.

Les lettres M et N ne présentent pas non plus de grande difficulté : elles illustrent néanmoins la fonction du paramètre *o#*. Dans le cas de la lettre M, les points *z*<sub>1</sub>, *z*<sub>5</sub> débordent légèrement sous la ligne de base, de même que les points *z*<sub>2</sub> et *z*<sub>4</sub> au dessus de la ligne supérieure. Ce sont des considérations optiques relatives à la lisibilité

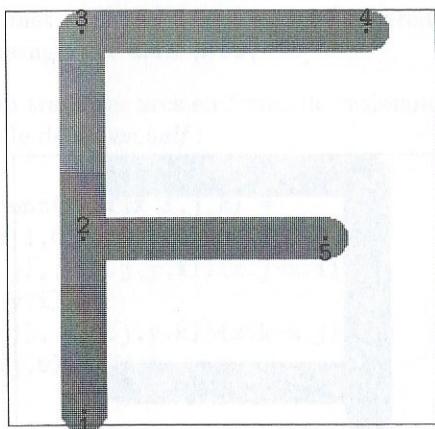


FIG. 5.2 – Construction de la lettre F

des caractères qui commandent ces petites modifications : les questions de lisibilité se trouvent évoquées au paragraphe 5.7 p. 161.

```

beginlogochar("M",18);
x1=x2=leftstemloc;
x4=x5=w-x1; x3=w-x3;
y1=y5; y2=y4;
bot y1=-o; top y2=h+o;
y3=y1+ygap;
draw z1--z2--z3--z4--z5;
labels(1,2,3,4,5);
endchar;

beginlogochar("N",15);
x1=x2=leftstemloc;
x3=x4=x5=w-x1;
bot y1=bot y4=-o;
top y2=top y5=h+o;
y3=y4+ygap;
draw z1--z2--z3;
draw z4--z5;
labels(1,2,3,4,5);
endchar;
```

Les largeurs des caractères sont variables. Elles sont toutes résumées dans le tableau 5.1 : M et W sont les plus larges tandis que i est évidemment la plus étroite.

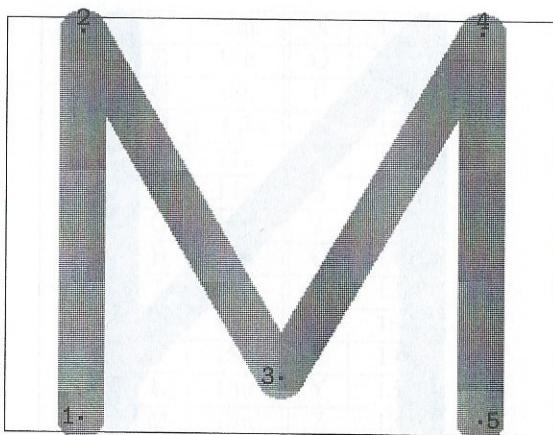


FIG. 5.3 – Construction de la lettre M

Les lettres A et O donnent une autre illustration de cette question de débordement et font appel d'autre part à la macro **super\_half** évoquée plus haut (voir les figures 5.5 et 5.6). Les définitions sont respectivement les suivantes :

```
beginlogochar("A",15);
x1=.5w; x2=x4=leftstemloc;
x3=x5=w-x2;
top y1=h+o;
y2=y3=barheight;
bot y4=bot y5=-o;
draw z4--z2--z3--z5;
super_half(2,1,3);
super_half(2,4,3);
labels(1,2,3,4,5);
endchar;

beginlogochar("O",15);
x1=x4=.5w;
top y1=h+o; bot y4=-o;
x2=w-x3=good.x(1.5u+s);
y2=y3=barheight;
super_half(2,1,3);
super_half(2,4,3);
lettreo:=currentpicture;
labels(1,2,3,4); endchar;
```

Remarquons que dans la définition de la lettre O, nous avons ajouté une variable appelée *lettreo* qui est de type « picture » et qui a pour fonction de garder en mémoire

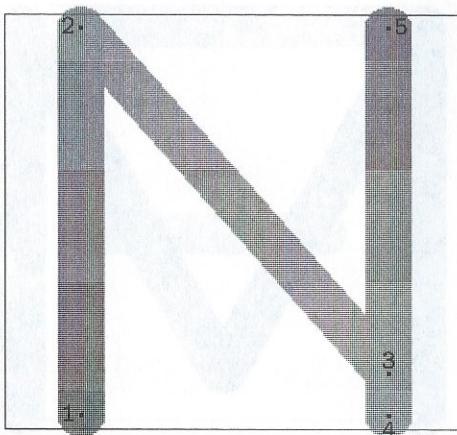


FIG. 5.4 – Construction de la lettre N

l'image représentant cette lettre : l'idée est que, lorsqu'il faudra dessiner la lettre Q, il suffira de rappeler la lettre O et de lui ajouter uniquement le trait supplémentaire en bas à droite. Pour que cette précaution puisse effectivement fonctionner, il faudra ne pas oublier de rajouter, au début du fichier `logocomplet.mf`, une déclaration pour la variable `lettreo` :

```
picture lettreo;
```

Une autre remarque à propos de la lettre O concerne la définition des abscisses des points  $z_2$  et  $z_3$  qui font appel à la commande **good.x** : ces deux points sont des positions à tangente verticale et sont l'exemple type de la situation où l'on doit faire appel à cette macro afin d'avoir des résultats satisfaisants au moment de la pixellisation (voir la discussion du paragraphe 5.6 p. 159).

Des sept lettres distinctes qui forment le mot METAFONT, il ne reste plus qu'à définir le T :

```
beginlogochar("T",13);
italcorr ht##slant + .5u#;
if .5w>good.x .5w: change_width; fi
lft x1=-eps; x2=w-x1;
x3=x4=.5w;
y1=y2=y3;
top y1=h; bot y4=-o;
draw z1--z2;
draw z3--z4;
labels(1,2,3,4);
endchar;
```

|   |       |   |       |
|---|-------|---|-------|
| A | 15u   | N | 15u   |
| B | 15u   | O | 15u   |
| C | 15u   | P | 12,5u |
| D | 14,5u | Q | 15u   |
| E | 14u   | R | 14u   |
| F | 14u   | S | 14u   |
| G | 15u   | T | 13u   |
| H | 14u   | U | 14u   |
| I | 7u    | V | 13u   |
| J | 11u   | W | 18u   |
| K | 15u   | X | 14u   |
| L | 13u   | Y | 13u   |
| M | 18u   | Z | 13u   |

TAB. 5.1 – Largeur des caractères dans la police logocomplet

Cette définition présente deux particularités intéressantes : l'utilisation des macros **italcorr** et **change\_width**. La barre supérieure de la lettre T est tracée sur toute la largeur de la boîte qui contient le caractère. Il y a donc nécessité de prévoir une correction italique : c'est une valeur qui sera transmise dans le fichier **.tfm** associé à la police **logocomplet** et qui sera exploitée par **TEX** pour ajouter un blanc supplémentaire à droite de la lettre T lorsque celle-ci sera utilisée dans la forme italique afin qu'elle n'entre pas éventuellement en collision avec le caractère qui la suivra. En mode *proof*, la présence et la valeur de cette correction est matérialisée par un trait vertical supplémentaire à droite de la boîte qui encadre le caractère (voir figure 5.7). Quant à la commande **change\_width**, elle concerne la barre verticale du T : on souhaite que celle-ci soit parfaitement centrée pour que la lettre soit symétrique ; pour qu'il en soit ainsi, il faut simplement que la largeur du stylo et la largeur de la lettre — exprimées en nombres de pixels — aient la même parité. Cela revient à dire que, si  $l$  désigne la largeur du stylo,  $w - l$  est pair et donc que  $0,5w - 0,5l$  est un nombre entier : or ce nombre est précisément égal à  $\text{lft } .5w$  ; cela est exprimé par la condition

$$0,5w = \text{good.x } 0,5w$$

Si la condition n'est pas réalisée, **change\_width** a pour effet de changer la parité de  $w$  pour la faire coïncider avec celle de  $l$ .

La lettre S est définie principalement au moyen de la macro **super\_crescent** vue au paragraphe 5.1.3 p. 114. On remarquera dans cette définition l'usage de la proportion 0,618 qui est, encore une fois, celle du nombre d'or (égale à son inverse<sup>7</sup>) :

```
beginlogochar("S", 14);
x3=x8=leftstemloc;
x2=x4=.382[x3,w-x3];
```

7. Le nombre d'or vérifie  $x(x - 1) = 1$ .

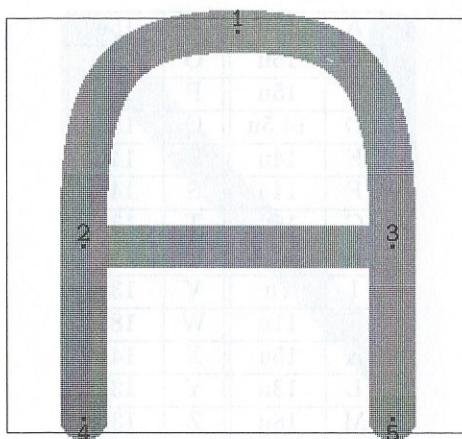


FIG. 5.5 – Construction de la lettre P

```

x1=x2=x3=leftstemloc;
x4=x5=.618[x1,w+1.5u-x1];
y2=y5; y3=.5[y2,y4];
bot y1=-o; top y8=h;
y4=barheight;
y1=y2; y3=.5[y2,y4];
y4=y5; y6=.5[y5,y7];
y7=y8;
draw z1--z2;
draw z4--z5;
draw z7--z8;
super_crescent(2,3,4);
super_crescent(5,6,7);
labels(1,2,3,4,5,6,7,8);
endchar;

```

Pour ce qui est de la lettre P, nous avons quelque peu modifié sa définition afin de mettre en application les idées du chapitre 4 et en prévision des définitions, par la suite, des lettres R et B. Le placement du point  $z_6$  fait appel à la macro **decalage** que nous avons définie au paragraphe 4.4.1 p. 97. Cette définition de macro sera donc à rajouter au préambule du fichier logocomplet.mf. La définition du P est la suivante :

```

beginlogochar("P",12.5);
x1=x2=x3=leftstemloc;
x4=x5=.618[x1,w+1.5u-x1];
y2=y5; y3=y4;
bot y1=-o; top y8=h;

```

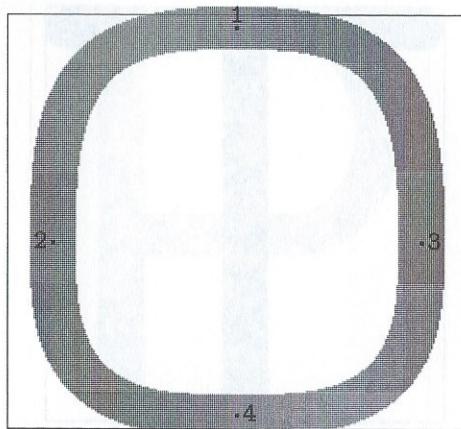
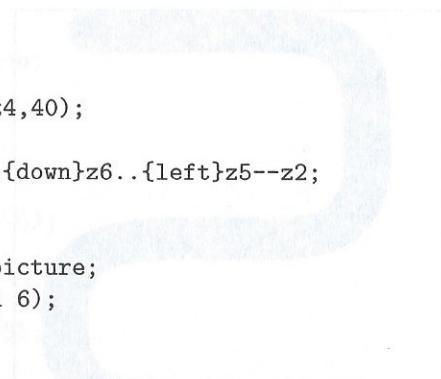


FIG. 5.6 – Construction de la lettre O

```

y2=barheight;
z6 = z5 decalage (z4,40);
path p;
p:= z3--z4{right}..{down}z6..{left}z5--z2;
draw z1--z3;
draw p;
lettrep := currentpicture;
labels(range 1 thru 6);
endchar;
```



On y remarque, comme pour la lettre  $\Sigma$ , l'usage du nombre d'or et, d'autre part, la variable de type « picture » appelée *lettrep* qui conserve en mémoire l'image de ce P. On rajoutera donc, dans le préambule, une déclaration :

```
picture lettrep;
```

La lettre D est obtenue assez simplement en utilisant la macro **super\_crescent** (voir l'illustration à la figure 5.10) :

```

beginlogochar("D",14.5);
x1=x3=leftstemloc;
x4=x5=.5[x1,w-x1];
y1=y5; y3=y4; bot y1=0;
top y3=h; y2=barheight;
draw z5--z1--z3--z4;
x2=.309[w-x1,lft w];
super_crescent(4,2,5);
labels(1,2,3,4,5);
```

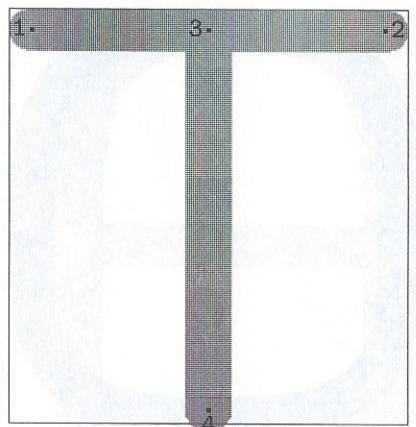


FIG. 5.7 – Construction de la lettre T

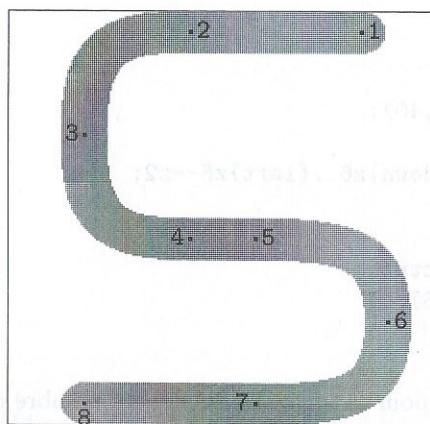


FIG. 5.8 – Construction de la lettre S

Pourquoi la valeur 0,309 dans la définition de D? C'est la moitié de l'inverse du nombre d'or! Voici maintenant quelques lettres aux définitions simples:

```
beginlogochar("H",14);
x1=x2=x3=leftstemloc;
x4=x6=w-x1;
x5=x4; y1=y6;
y2=y5; y3=y4;
```

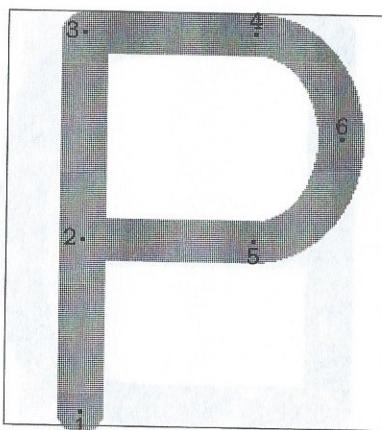


FIG. 5.9 – Construction de la lettre P

```

bot y1=-o; top y3=h+o;
y2=barheight;
draw z1--z3;
draw z4--z6;
draw z2--z5;
labels(range 1 thru 6);
endchar;

beginlogochar("L",13);
x1=x2=leftstemloc;
x3=w-x1+ho;
y1=y3;
bot y1=0; top y2=h+o;
draw z3--z1--z2;
labels(1,2,3);
endchar;

beginlogochar("I",7);
if .5w<>good.x .5w: change_width; fi
x1=x2=x3=.5w;y3=1.2h;
bot y1=-o; top y2 = h+o;
draw z2--z1;
labels(1,2,3);
endchar;

beginlogochar("J",11);

```

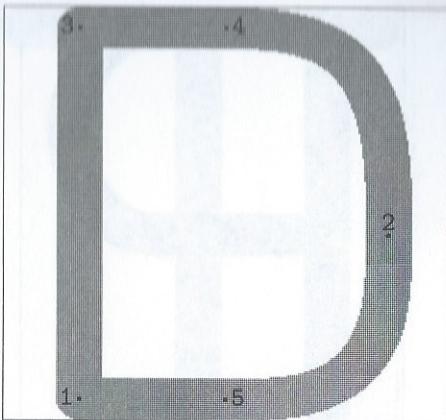


FIG. 5.10 – Construction de la lettre D

```

x1=x5=w-leftstemloc;
bot y2 = bot y3=0; top y1 = h+o;
x3=1.5x4;x2=x5-x3+x4;
x4=leftstemloc;y5=y4=.5barheight;
draw z1--z5{down}..{left}z2--z3{left}..{up}z4;
labels(1,2,3,4);
endchar;

beginlogochar("U",14);
x1=.5w; bot y1=-o;
x2=x4=w-x3=w-x5=good.x(2u);
y2=y3=.618barheight;
top y4 = h+o; y5=y4;
super_half(2,1,3);
draw z2--z4 ;
draw z3--z5;
labels(range 1 thru 6);
endchar;

beginlogochar("V",13);
if .5w>>good.x .5w: change_width; fi
lft x1=-eps; x2=w-x1;
x3=.5w; y1=y2;
top y1=h+o; bot y3=-o;
draw z1--z3;

```

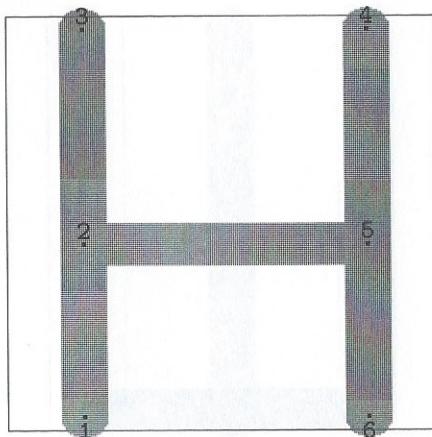


FIG. 5.11 – Construction de la lettre H

```

draw z2--z3;
labels(1,2,3);
endchar;

beginlogochar("W",18);
italcorr ht#*slant + .5u#;
lft x1=0; x3=.5w=2x2;
x5=w-x1;
x4=w-x2; y4=y2;
y1=y3=y5;
top y1=h+o; bot y2=-o;
draw z1--z2--z3--z4--z5;
labels(1,2,3,4,5);
endchar;

beginlogochar("X",14);
x1=x2-ho=.8leftstemloc;
x4=w-x1=x3+ho;
y1=y4; y2=y3;
bot y1=-o; top y2=h+0;
draw z1--z3;
draw z2--z4;
labels(range 1 thru 4);
endchar;

beginlogochar("Y",13);

```

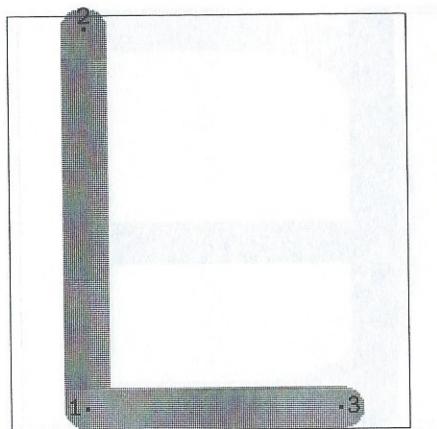


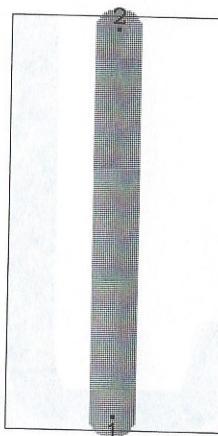
FIG. 5.12 – Construction de la lettre L

```

if .5w<>good.x .5w: change_width; fi
x1=x2=.5w;
x3=leftstemloc;
x4=w-x3; bot y1=-o;
y2=barheight;
top y3 = top y4 = h+0;
draw z3--z2--z4;
draw z1--z2;
labels(range 1 thru 4);
endchar;

beginlogochar("Z",13);
x1=x2=leftstemloc;
x3=x4=w-x1+ho;
y1=y4; y2=y3;
bot y1=0; top y2=h+0;
z5=whatever[z1,z3];
y5=y6=y7=barheight;
x6=x5-1.5u;
x7=x5+1.5u;
draw z2--z3--z1--z4;
draw z6--z7;
labels(range 1 thru 7);
endchar;

```

FIG. 5.13 – Construction de la lettre *l*

On remarquera, dans les définitions des dix lettres ci-dessus, quelques particularités :

- la barre horizontale du *L* est augmentée de la quantité *ho#*;
- les lettres *I* et *Y* font appel, comme on l'a déjà vu avec le *T*, à l'instruction **change\_width**;
- la lettre *W* comporte une correction italique car elle touche les bords de la boîte qui l'entoure ;
- la lettre *X* est un peu plus large en bas qu'en haut pour tenir compte de considérations optiques concernant la lisibilité (voir au paragraphe 5.7 p. 161);
- la lettre *Z* fait usage de la macro *whatever* pour le placement du point *z<sub>5</sub>*.

Pour ce qui est de la lettre *R*, son tracé fait appel à des techniques plus complexes car elle utilise l'instruction **solve** vue au paragraphe 4.4.2. La lettre *R* a une partie commune avec la lettre *P*: il suffira d'appeler cette partie commune, précédemment stockée dans la variable de type « picture » intitulée *lettrep*, et de lui rajouter une « patte » oblique. Comme on peut le voir sur la figure 5.21, cette patte va du point *z<sub>7</sub>* au point *z<sub>8</sub>*: ce point *z<sub>7</sub>* va être choisi de telle sorte que la tangente à l'arc de courbe qui va de *z<sub>4</sub>* à *z<sub>5</sub>* passe par un point *z<sub>12</sub>* imposé. Le point *z<sub>2</sub>* a pour ordonnée *barheight* qui est ce paramètre qui a déjà fixé la hauteur des barres horizontales dans les lettres *A*, *E*, *F* et *P*. Le point *z<sub>12</sub>* découpe le segment *z<sub>1</sub>z<sub>2</sub>* dans la proportion du nombre d'or :

$$z_{12} = .618[z_1, z_2];$$

Le chemin *p* qui fait la boucle du *R* et va de *z<sub>3</sub>* à *z<sub>2</sub>* a pour longueur 4 : la question est de savoir à quel « temps » *t* sur le chemin *p* la tangente aura la propriété voulue :

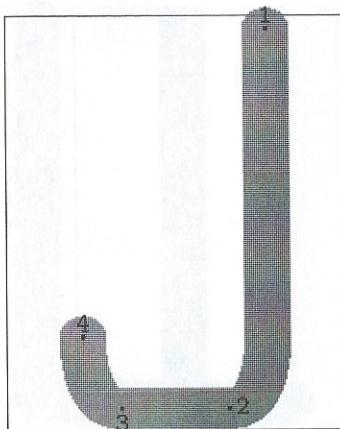


FIG. 5.14 – Construction de la lettre J

le point correspondant **point t of p** sera  $z_7$ . Pour utiliser *solve*, on définit une variable *test*:

```
vardef test(expr t)=
- angle(direction t of p) + angle (z12- point t of p) < 0 enddef;
```

La propriété testée est vraie pour  $t = 2$  (c'est le point  $z_6$ ) et fausse pour  $t = 3$  (c'est le point  $z_5$ ). On écrira donc :

```
t = solve test(2,3);
z7 = point t of p;
```

Une « tolerance » de 0,001 fournit un bon résultat. Pour terminer, le point  $z_8$  est choisi de telle sorte que la patte du R soit normale à la boucle (perpendiculaire à la tangente en  $z_7$ ). Voici le programme complet :

```
beginlogochar("R",14);
x1=x2=leftstemloc;
bot y1=-o;y8=y1;
y2=barheight;
z12=.618[z1,z2];
vardef test(expr t)=
- angle(direction t of p)
+ angle (z12- point t of p) < 0 enddef;
tolerance:=.001;
t = solve test(2,3);
z7 = point t of p;
(z8-z7).dotprod(z12-z7)=0;
```

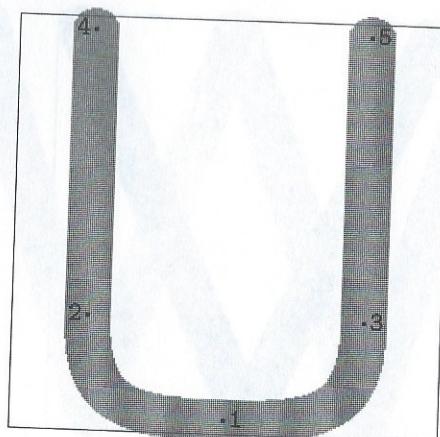


FIG. 5.15 – Construction de la lettre U

```

addto currentpicture also lettrep;
draw z7--z8;
labels(1,2,7,8,12);
endchar;
```

La lettre B elle aussi est définie à partir de la lettre P mais de façon plus simple que R. Voici cette définition (voir la figure 5.22) :

```

beginlogochar("B",15);
x1=leftstemloc;
bot y1=-o;y7=y1;
y5=barheight;
x5=x7=.618[x1,w-x1];
z8 = z7 decalage (z5,50);
addto currentpicture also lettrep;
draw z5{right}..z8..{left}z7;
draw z1--z7;
labels(1,5,7,8);
endchar;
```

La syntaxe pour rappeler la lettre P stockée est impérativement :

```
addto currentpicture also lettrep;
```

et il serait incorrect d'écrire : `draw lettrep;` puisque la variable `lettrep` est de type `*picture*` et non `*path*`. On procède de même pour obtenir la lettre Q à partir de la lettre O :

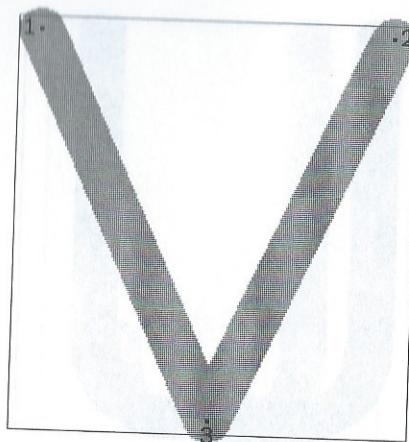


FIG. 5.16 – Construction de la lettre V

```

beginlogochar("Q",15);
x5=.5w; bot y5=-o;
x3=w-good.x(1.5u+s);
y3=barheight;
x6=x3+.5u;y6=y5-.5u;
z53=whatever[z5,z3];
(z3-z5)dotprod(z53-z6)=0;
addto currentpicture also lettreo;
draw z53--z6;
labels(1,2,3,5,6,53);
endchar;

```

Dans cet exemple, l'opérateur **dotprod** et la commande **whatever** sont utilisés conjointement pour trouver le point  $z_{53}$ , projection orthogonale de  $z_6$  sur  $z_3z_5$  (voir la figure 5.23).

La lettre K utilise l'instruction **solve** pour déterminer la position du point  $z_5$  (voir la figure 5.24) : une fois placés les points  $z_1, z_3, z_4$  et  $z_6$ , on souhaite que le point  $z_5$  se trouve à la même hauteur *barheight* que la barre du E et que  $z_5z_6$  soit perpendiculaire à  $z_2z_4$ . Si l'angle  $z_4z_5z_6$  est droit,  $z_5$  se trouve sur le cercle ayant pour centre le milieu de  $z_4z_6$  (de coordonnées  $(x_4, h/2)$ ) et de rayon  $h/2$ : c'est cette condition qui est testée par l'instruction **solve**.

```

beginlogochar("K",15);
x1=x2=x3=leftstemloc;
x4=x6=w-x1; y3=y4;
bot y1=bot y6=-o; top y3=h+o;

```

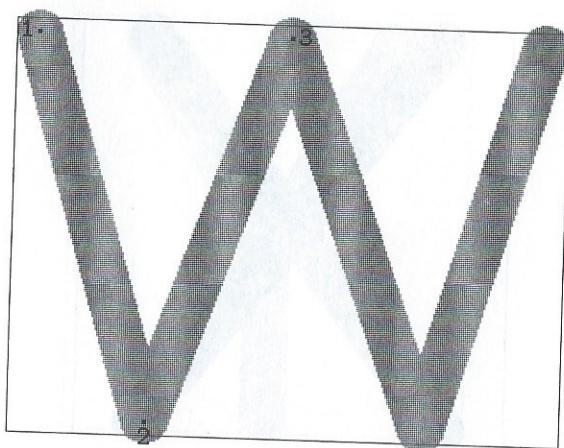


FIG. 5.17 – Construction de la lettre W

```

y5=barheight;
vardef test(expr t)= length((x4,h/2)-(t,y5))>h/2 enddef;
x5=solve test(0,x4);
z2=whatever[z4,z5];
draw z1--z3;
draw z2--z4;
draw z5--z6;
labels(1,2,3,4,5,6);
endchar;
```

Il ne reste que deux lettres pour que l'alphabet soit complet : C et G. Ces deux lettres ont des éléments communs et on pourrait envisager une définition qui fournisse la lettre G à partir de la lettre C. En fait, nous allons procéder différemment : la lettre G va être définie la première et, en cours de définition, nous trouverons C que nous stockerons provisoirement dans une variable de type « picture » :

```

beginlogochar("G",15);
x4=.618leftstemloc;
y8=y4=.5h;
bot y5=bot y6=0;
top y2 = top y3 = h;
x3=x5=x4+.309w;
x2=x6=w-x3;
x8=w-x4;
path chem;
chem:= z8{up}..z2{left}--z3{left}
..z4..{right}z5--z6{right}..{up}cycle;
```

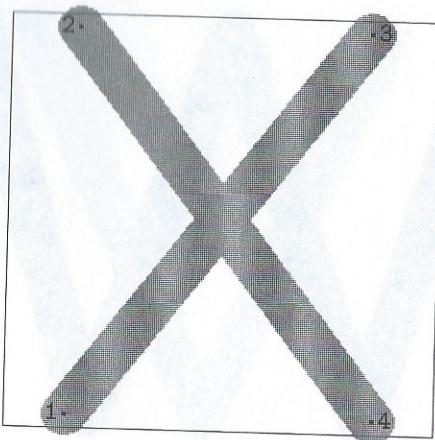


FIG. 5.18 – Construction de la lettre X

```
t1:=directiontime (-1,sqrt 3) of chem;
z1 = point t1 of chem;
t2:=directiontime (1,sqrt 3) of chem;
z7 = point t2 of chem;
x10=x7-2u;x11=x7+1.5u;
y9=y10=y11=.8barheight;
x9=x7;
draw subpath (1,5) of chem;
draw z1..{left}z2; draw z6{right}..z7;
lettrec:=currentpicture;
erase draw z6{right}..z7;
draw z6{right}..{up}z7--z9;
draw z10--z11;
labels(range 1 thru 8);
endchar;
```

Le point  $z_1$  est calculé ici de telle sorte que la tangente en ce point soit inclinée de  $60^\circ$  sous l'horizontale. Il en va de même symétriquement au point  $z_7$ . Autour de ce point  $z_7$ , les lettres C et G diffèrent totalement ; donc, une fois définie la lettre C, l'arc de  $z_6$  à  $z_7$  est effacé et la lettre G est complétée. Cette façon de procéder fait que la définition de C se résume ensuite à trois lignes :

```
beginlogochar("C",15);
addto currentpicture also lettrec;
endchar;
```

L'alphabet logocomplet comporte maintenant les vingt-six lettres majuscules :

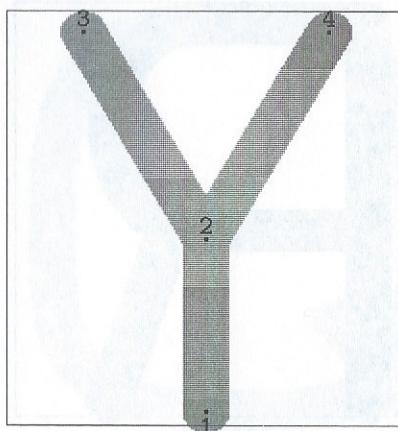


FIG. 5.19 – Construction de la lettre Y

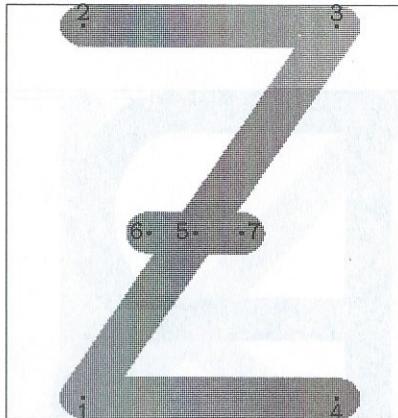


FIG. 5.20 – Construction de la lettre Z

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Le lecteur pourra s'exercer en complétant à son tour la police logocomplet : il faudrait maintenant définir les lettres minuscules, les chiffres, les signes de ponctuation, les accents, etc.

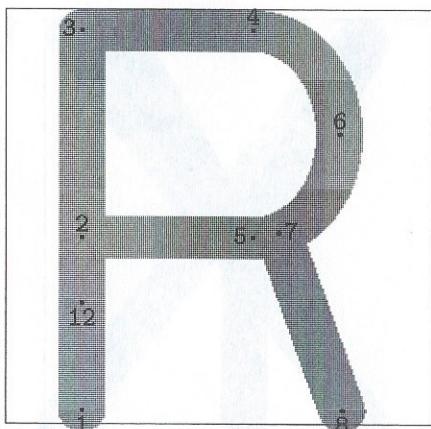


FIG. 5.21 – Construction de la lettre R

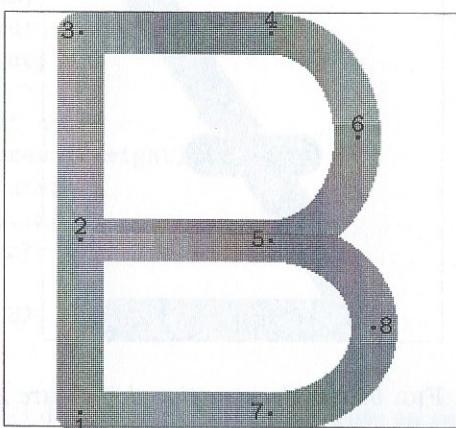


FIG. 5.22 – Construction de la lettre B

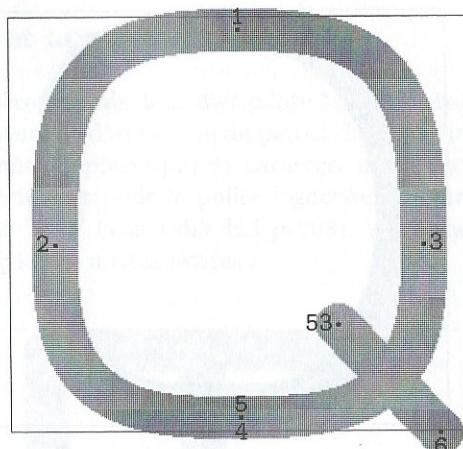


FIG. 5.23 – Construction de la lettre Q

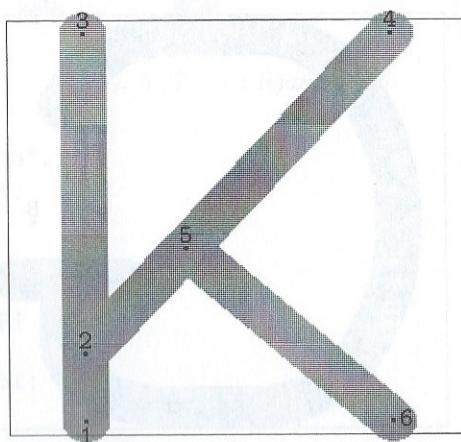


FIG. 5.24 – Construction de la lettre K

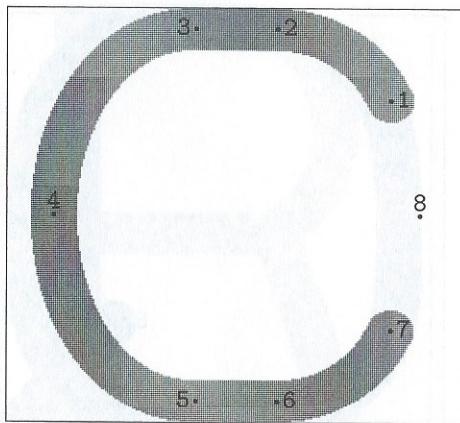


FIG. 5.25 – Construction de la lettre C

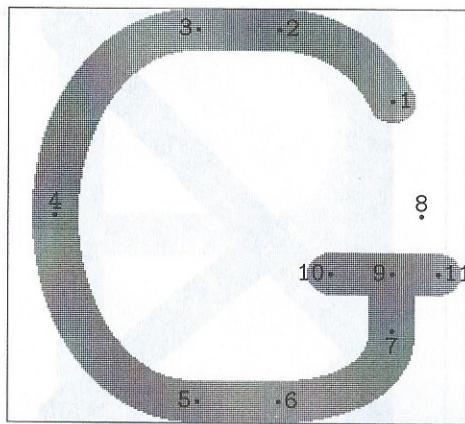


FIG. 5.26 – Construction de la lettre G

### 5.1.5 Ligatures et approches de paires

Une fois tous les caractères définis, le fichier pilote `logocomplet.mf` comporte ensuite des instructions de ligature et d'approche de paires. Les ligatures concernent certains couples de lettres qui sont remplacés par un caractère unique comme dans : fi, ff, fl, ffi, ffi. Nous allons nous contenter pour la police `logocomplet` d'un caractère `\texttt{F}`. Le code octal de ce caractère est "013" (voir table E.1 p. 203). Voici une définition possible de ce caractère, dans le style des autres lettres :

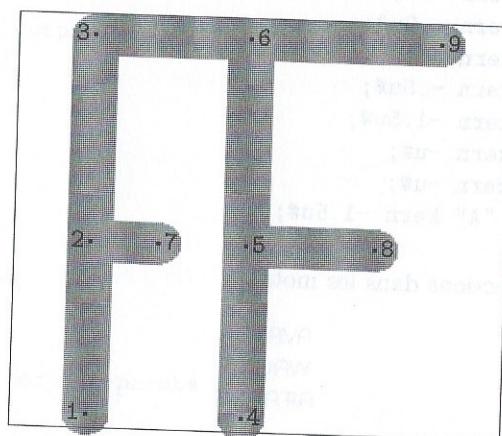


FIG. 5.27 – La ligature `\texttt{F}`

```
beginlogochar(oct"013",16);
x1=x2=x3=leftstemloc;
x4=x5=x6;
x9=w-x1+ho;
z6=.45[z3,z9];
z7(x5-x7)=x9-x8=xgap;
bot y1=-o; y1=y4;
y2=y7=y5=y8=barheight;
top y3=h;
y3=y9;
draw z1--z3--z9;
draw z2--z7;
draw z5--z8;
draw z4--z6;
labels(range 1 thru 9);
endchar;
```

On ajoutera donc au fichier `logocomplet.mf` l'information suivante :

```
ligtable "F"; "F"=oct"013";
```

qui indiquera à TeX par la suite que, lorsque deux lettres F se suivent, il doit les remplacer par le caractère FF. L'instruction **ligtable** accueille aussi, grâce à l'instruction **kern**, l'information concernant les approches de paires, c'est-à-dire des modifications de l'espacement entre certains couples de caractères :

```

ligitable "A": "T" kern -.5u#, "V" kern -1.5u#,
 "W" kern -u#, "Y" kern -u#;
ligitable "F": "F"=:oct"013", "O" kern -u#;
ligitable "L": "V" kern -u#, "W" kern -.8u#,
 "Y" kern -.9u#;
ligitable "P": "O" kern u#;
ligitable "T": "A" kern -.5u#;
ligitable "V": "A" kern -1.5u#;
ligitable "W": "A" kern -u#;
ligitable "Y": "A" kern -u#;
ligitable oct"013": "A" kern -1.5u#;
```

On observera ces corrections dans les mots :

```

AVATAR
WAGON
AIFABLE
```

### 5.1.6 Informations globales

Le fichier logocomplet.mf s'achève avec des instructions globales concernant la police :

```

font_quad:=18u#+2s#;
font_normal_space:=6u#+2s#;
font_normal_stretch:=3u#;
font_normal_shrink:=2u#;
font_identifier:="LOGOCOMPLET" if slant<>0: & "SL" fi;
font_coding_scheme:="MAJUSCULES seulement";
```

**font\_quad** et **font\_normal\_space** sont particulièrement importants : si on néglige de fixer leurs valeurs, il sera impossible d'obtenir des blancs pour séparer les caractères ou les mots !

### 5.1.7 Fichier de paramètres

Les paramètres, comme il a été dit à plusieurs reprises, sont en général regroupés dans des fichiers à part qui fixent leurs valeurs pour chaque corps de caractères et les transmettent ensuite au fichier pilote en appelant ce dernier au moyen d'une instruction **input**. La question peut se poser de savoir si tous les paramètres doivent

être regroupés dans un fichier séparé: il est clair que seuls les paramètres que l'on considère comme modifiables en fonction des différents corps devraient figurer dans le fichier de paramètre et que les autres devraient rester dans le fichier pilote dans lequel ils jouent le rôle de valeurs définissant le style général de la police et garantissant sa cohérence. C'est bien évidemment au créateur qu'il appartient de décider ce qui est intrinsèque de ce qui est modifiable.

Voici deux exemples de fichiers de paramètres, l'un — `logocomplet10.mf` — pour fabriquer le corps 10 et l'autre — `logocomplet8.mf` — pour le corps 8:

```
%LOGOCOMPLET en corps 10-points
font_size 10pt#;
ht#:=6pt#;
xgap#:=0.6pt#;
u#:=4/9pt#;
b#:=0;
o#:=1/9pt#;
px#:=2/3pt#;
input logocomplet
bye

METAFONT

%LOGOCOMPLET en corps 8-points
font_size 8pt#;
ht#:=.8*6pt#;
xgap#:=.8*0.6pt#;
u#:=.82*4/9pt#;
b#:=.2pt#;
o#:=1/12pt#;
px#:=.8*2/3pt#;
input logocomplet
bye
```

On constate par exemple que les paramètres *leftstemloc* ou bien *barheight* n'y figurent pas. Ils sont donc considérés comme caractéristiques de la forme de la police *logocomplet*. Rien n'empêche cependant de les modifier pour tester justement cette forme. Voici plusieurs lettres A avec la barre horizontale placée à diverses hauteurs: A A A A A.

## 5.2 Modifier globalement une police

Une fois qu'une police de caractères a été créée, on peut très facilement la décliner sous des formes variées. METAFONT offre plusieurs techniques pour obtenir ces variations. Certaines portent sur le fichier pilote (*currenttransform*) et d'autres sur les fichiers paramètres (par modification des valeurs).

### 5.2.1 Utilisation de *currenttransform*

La macro **currenttransform** est en général égale à l'identité, ce qui fait que son action est (dans ce cas) invisible. Mais c'est une variable de type *transformation* qui agit sur toutes les instructions *draw*, *filldraw*, etc. On peut donc, en la modifiant, obtenir des modifications globales d'une police de caractères. Faisons l'expérience sur le fichier pilote *logocomplet.mf*. Dans le préambule de ce fichier, on peut lire :

```
if unknown slant: slant:=0 else: currenttransform:=
identity slanted slant yscaled aspect_ratio fi;
```

Nous allons modifier cette ligne de la manière suivante :

```
slant:=0;
currenttransform := identity reflectedabout ((0,0),(0,ht#));
```

Tous les tracés effectués par METAFONT seront donc remplacés par leur symétrique par rapport à l'axe vertical, ce qui produit le résultat suivant :

T H O F A T E M

Toutes les transformations prédéfinies par le format *plain* peuvent ainsi entrer en jeu (au besoin on peut même les combiner entre elles). Voici un certain nombre de variations sur ce thème :



On les a obtenues respectivement en écrivant dans le fichier pilote *logocomplet.mf* :

```
currenttransform := identity slanted .7;
currenttransform := identity rotated 30;
currenttransform := identity scaled .5;
currenttransform := identity xscaled .5;
currenttransform := identity reflectedabout ((0,108),(10,108));
currenttransform := identity slanted -1.5 rotated -45;
```

### 5.2.2 Modification d'un fichier de paramètres

L'autre manière de modifier une police consiste, comme on l'a déjà dit, à modifier les valeurs des paramètres. En voici deux exemples accomplis encore une fois sur le fichier `logocomplet10.mf`. Les valeurs normales des paramètres décrits au paragraphe 5.1.1 sont :

```
ht#:=6pt#
*xgap#:=0.6pt#
u#:=4/9pt#
s#:=0
o#:=1/9pt#
px#:=2/3pt#
```

Nous allons prendre successivement les valeurs suivantes :

```
font_size 20pt#
ht#:=20pt#
*xgap#:=1pt#
u#:=2/9pt#
s#:=2/9pt#
o#:=1/3pt#
px#:=1pt#
slant:=-1/3
font_size 10pt#
ht#:=5pt#
*xgap#:=2pt#
u#:=5/3pt#
s#:=-2/9pt#
o#:=0
px#:=1/3pt#
```

On obtient alors :

METAFONT E FANTOMAS

M E T A F O N T  
M E T A P O S T

## 5.3 Forme et série

Au vu de ce qui a été expliqué précédemment, c'est maintenant un jeu d'enfant que d'obtenir des formes italiques diverses aussi bien que des caractères gras.

### 5.3.1 Caractères penchés

Il faut tout d'abord rétablir la formulation initiale de l'instruction conditionnelle placée au début du fichier `logocomplet.mf` concernant `currenttransform`:

```
if unknown slant: slant:=0 else: currenttransform:=
identity slanted slant yscaled aspect_ratio fi;
```

Elle va en effet nous permettre d'avoir une version italicisée de la police `logocomplet`. Il suffit de définir un fichier de paramètres `logocompletsl10.mf` comportant, en plus des valeurs déjà données plus haut, une valeur non nulle pour le paramètre `slant`. Par exemple :

```
%LOGOCOMPLET SL en corps 10-points
slant:=.2;
input logocomplet10
```

On obtient ainsi :

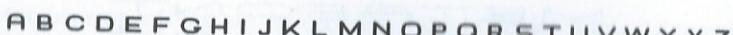


### 5.3.2 Caractères gras

Par modification de certains paramètres, nous obtenons des caractères gras. Il suffit de créer un fichier intitulé `logocompletb10.mf` avec le contenu suivant :

```
LOGOCOMPLET gras en corps 10-points
font_size 10pt#;
ht#:=6pt#;
xgap#:=23/20*0.6pt#;
u#:=23/20*4/9pt#;
s#:=.1pt#;
o#:=1/9pt#;
px#:=44/36pt#;
input logocomplet
```

Avec ces nouvelles valeurs, on obtient les caractères suivants :



et, en ajoutant une légère inclinaison, on a des caractères à la fois gras et penchés<sup>8</sup> :



<sup>8</sup> Ici `slant=0.1`.

## 5.4 Empattements

La police logocomplet définie jusque là a une caractéristique importante : c'est une police sans empattements ni ornements comme le sont, par exemple, les polices Helvetica, Geneva, Monaco ou, dans la famille Computer Roman, cmss et tous ses dérivés. Une police sans empattements est dite, en anglais, « sans serif ». À l'opposé les caractères d'une police avec empattements seront dits « romains » : c'est le cas évidemment de la police cmr mais aussi de la police cmtt, cette dernière différant de la précédente par le fait que tous ses caractères ont même largeur, comme ceux d'une machine à écrire. Une métapolice doit pouvoir offrir de manière unifiée la possibilité de générer toutes ces variantes et le présent paragraphe a pour but de modifier le fichier pilote afin d'obtenir, à volonté, une police avec ou sans empattements, à largeur fixe ou variable, etc. suivant le même concept : un fichier pilote unique et des variations obtenues par le biais des fichiers de paramètres.

### 5.4.1 Style romain

La forme des empattements est bien entendu extrêmement variable et constitue un des éléments forts qui donneront une personnalité à une police. METAFONT va nous offrir la possibilité de définir des empattements types auxquels on fera appel très simplement au moyen de macros. La figure 5.28 montre une forme typique d'empattement. Nous définirons de la sorte un empattement autour d'un certain point de construction  $z_i$  grâce à six points supplémentaires qui vont l'entourer comme on le voit sur cette figure. Ces six points  $z_{ia}, z_{ib}, z_{ic}, z_{id}, z_{ie}, z_{if}$  joueront un rôle analogue à celui des points  $z_{il}$  et  $z_{ir}$  fournis par l'instruction **penpos**<sup>9</sup>.

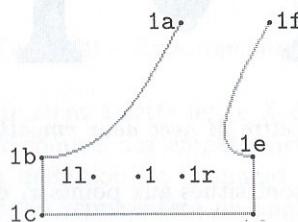


FIG. 5.28 – Empattement de caractère romain

L'empattement sera caractérisé par des paramètres que nous appellerons *empat\_g*, *empat\_d*, *empat\_e*, *empat\_l*, *empat\_h* et *theta* qui représenteront respectivement la largeur de l'empattement vers la gauche (distance  $x_{ixie}$ ) et vers la droite (distance  $x_{ixid}$ ), l'épaisseur de l'empattement (hauteur entre  $x_{ib}$  et  $x_{ic}$ ), la largeur de la tige et la hauteur du point  $z_{ia}$  par rapport à  $z_i$ . *theta* est l'angle d'inclinaison de la tige.

Une macro va résumer les instructions de tracé de l'empattement en un point donné en fonction des valeurs des paramètres qui viennent d'être décrits :

9. Cf. paragraphe 3.3 p. 44.

```

def empattement (suffix $)(expr empat_g,empat_d,
 empat_e,empat_l,empat_h,theta)=
penpos$(empat_l/abs sind theta,0);
x$c=x$-empat_g;
x$d=x$+empat_d;
y$c=y$d=bot y$;
x$b=x$c; x$e=x$d;
y$b-y$c=y$e-y$d=empat_e;
z$a-z$l=z$f-z$r=(empat_h/abs sind theta)* dir theta;
fill z$a{z$l-z$a} ... {left}z$b--z$c--z$d--z$e{left}
... {z$f-z$r}z$f--cycle
enddef;

```

La figure 5.29 nous montre la lettre M dont on a modifié la définition en rajoutant simplement deux lignes :

```

empattement(1,1.6u,1.6u,.5py,px,py,90);
empattement(5,1.6u,1.6u,.5py,px,py,90);

```



FIG. 5.29 – Lettre M avec deux empattements

puisque les deux empattements sont situés aux points  $z_1$  et  $z_5$ .

On peut encore améliorer l'efficacité de la méthode en définissant une macro complémentaire de la précédente qui permettra de ne donner qu'une seule instruction lorsque plusieurs points (en nombre d'ailleurs quelconque) sont amenés à recevoir un même empattement. On définira :

```

def empat (expr a,b,c,d,e,f)(text t)=
forsuffixes i=t:
empattement(i,a,b,c,d,e,f); endfor enddef;

```

Il suffirait alors d'écrire :

```
empat(1.6u,1.6u,.5py,px,py,90,1,5);
```

Sans modifier en quoi que ce soit les macros ci-dessus, on obtient également des empattements inclinés comme illustré sur la figure 3.12. La définition de la lettre X est ainsi modifiée :

```
beginlogochar(0,14); "Lettre X";
x1=x2-ho=.8leftstemloc;
x4=w-x1=x3+ho;
y1=y4; y2=y3;
bot y1=-o; top y2=h+o;
theta=angle (z3-z1);
draw z1--z3;
draw z2--z4;
empattement(1,1.6u,1.6u,.4py,px,py,theta);
empattement(4,1.6u,1.6u,.4py,px,py,180-theta);
labels(2,3);penlabels(1,4);
endchar;
```

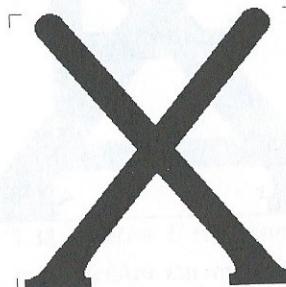


FIG. 5.30 – Empattements obliques

Il manque maintenant clairement à cette lettre X des ornements supérieurs que l'on pourrait très bien obtenir comme des empattements retournés. Il suffit pour cela de modifier à nouveau la définition de la macro *empattement* en lui ajoutant un paramètre supplémentaire qui représentera un angle de rotation, nommé ici *pos*, de l'empattement :

```
def empattement (suffix $)(expr empat_g,
 empat_d,empat_e,empat_l,empat_h,theta,pos)=
 penpos$(empat_l/abs sind theta,0);
 x$c=x$-empat_g;
 x$d=x$+empat_d;
 y$c=y$d=bot y$;
 x$b=x$c; x$e=x$d;
 y$b-y$c=y$e-y$d=empat_e;
 z#a-z$l=z#f-z#r=(empat_h/abs sind theta)* dir theta;
 fill (z#a{z$l-z#a} .. {left}z$b-z$c=z$d-z$e{left}) ..
```

```
{z$f-z$r}z$f--cycle) rotatedarround(z$,pos)
enddef;
```

et la lettre X définitive, telle qu'on peut la voir sur la figure 5.31, est obtenue dorénavant avec les commandes suivantes :

```
empattement(1,1.6u,1.6u,.4py,px,py,theta1,0);
empattement(4,1.6u,1.6u,.4py,px,py,180-theta1,0);
empattement(2,1.6u,1.6u,.4py,px,py,180-theta1,180);
empattement(3,1.6u,1.6u,.4py,px,py,theta1,180);
```

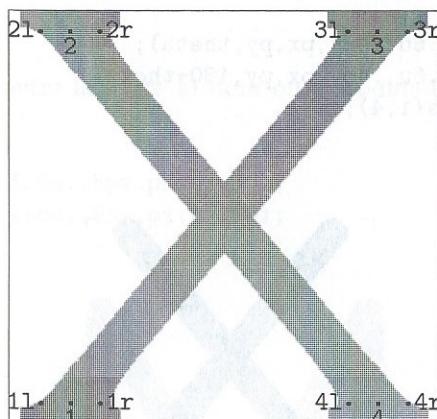


FIG. 5.31 – Ornements inférieurs et supérieurs

Cette nouvelle formulation de la macro nous permet d'ailleurs de faire toutes sortes d'ornements. En particulier, avec une rotation de 90°, on améliore les terminaisons des trois barres horizontales de la lettre E (voir figure 5.32) en écrivant :

```
empattement(4,1.4u,.5py,.4py,.9py,2u,90,90);
empattement(5,1.2u,1.2u,.3py,py,2u,90,90);
empattement(6,.5py,1.4u,.4py,.9py,2u,90,90);
```

À y regarder de près<sup>10</sup>, ces empattements présentent un petit défaut car la forme du stylo provoque un léger dépassement à droite. Pour y remédier, on rajoutera simplement une instruction **cutoff** au début de la définition de la macro *empattement* dont la première ligne de définition sera réécrite ainsi :

```
cutoff(z$,90-pos);penpos$(empat_1/abs sind theta,0);
```

pour obtenir finalement la figure 5.33.

<sup>10</sup> Le lecteur aura tout intérêt à faire l'acquisition d'un compte-fil qui est une loupe spéciale à fort grossissement utilisée par les typographes pour contrôler les impressions. Un compte-fil est muni de ses boutons et saffra la position de l'encre dans la grille des matricules.

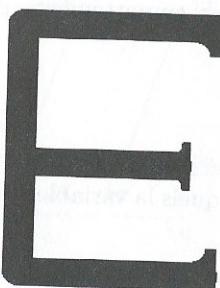


FIG. 5.32 – Empattements verticaux

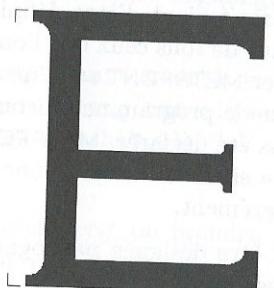


FIG. 5.33 – Lettre E romaine corrigée

Il reste à voir comment on peut mettre en œuvre le concept de métapolice pour obtenir au choix des caractères avec ou sans empattements. La solution passe par la définition d'une variable de type « boolean » (variable logique qui sera soit vraie, soit fausse) dont la valeur sera passée par le fichier de paramètres. Convenons de l'appeler, par exemple, *serif* et écrivons par conséquent, dans le fichier de paramètres, une déclaration :

```
boolean serif;
```

Ainsi la lettre E de la figure 5.33 sera définie avec des instructions d'empattement soumises à condition, de la manière suivante :

```
if serif :
 empattement(4,.1.4u,.5py,.5py,.9py,2u,90,90);
 empattement(5,.1.2u,1.2u,.3py,py,2u,90,90);
 empattement(6,.5py,1.4u,.5py,.9py,2u,90,90);
 empattement(1,.1.6u,1.6u,.4py,px,2py,90,0);
 empattement(3,.1.6u,1.6u,.4py,px,2py,90,180)
fi;
```

et la lettre sera obtenue avec ses empattements pour peu que, dans le fichier de paramètres, on ait indiqué :

```
serif := true;
```

On devra ainsi avoir deux fichiers nommés, par exemple, `logocompletrm10.mf` et `logocompletss10.mf`<sup>11</sup> dans lesquels la variable *serif* aura respectivement les valeurs *true* et *false*.

Enfin, pour éviter tout risque d'erreur, on prendra la précaution, dans le fichier pilote, d'ajouter la ligne suivante :

```
if unknown serif : boolean serif; serif:=false fi;
```

En effet, la variable booléenne *serif* vient d'être définie dans un nouveau fichier de paramètres : elle est donc inconnue de tous ceux que l'on aura pu éventuellement créer auparavant. Or, si on fait tourner METAFONT sur l'un d'eux, on aura immanquablement un message d'erreur lorsque le programme rencontrera la variable *serif* dans le fichier pilote : celle-ci n'ayant pas été déclarée, METAFONT la considérera comme une variable numérique et protestera si on lui attribue une valeur *true* ou *false*. La ligne ci-dessus évite ce genre de désagrément.

En corps 10, la police pourrait être désignée par `logocompletrm` et serait obtenue à partir d'un fichier de paramètres `logocompletrm10` :

### 5.4.2 Style courrier

Il va être maintenant très aisément d'obtenir la variante « courrier » de la métapolice logocomplet : celle-ci va nous amener à modifier la forme des empattements afin d'avoir quelque chose de plus carré (voir figure 5.34) ; d'autre part elle est caractérisée par le fait que tous les caractères ont la même largeur que nous appellerons *larg\_tt#*. Le tableau 5.1 donne les largeurs qui ont été adoptées pour les vingt-six majuscules : la valeur moyenne est de  $14u$  et c'est elle que nous choisirons comme valeur fixe pour cette nouvelle forme de la police. Le passage à la version « courrier » sera contrôlé par une variable booléenne que nous appellerons précisément *courrier*. On ajoute donc au début du fichier de paramètres, la déclaration de variable et la valeur de paramètre comme suit :

```
boolean courrier;
larg_tt#:=14u#;
courrier:=true;
```

---

<sup>11</sup>. rm pour romain, ss pour sans serif.

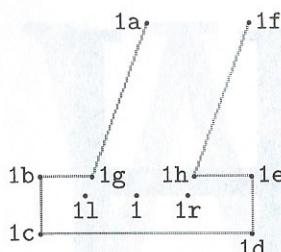


FIG. 5.34 – Empattement pour la famille tt

Par ailleurs, dans le fichier pilote, la macro **beginlogochar** va être légèrement modifiée pour devenir :

```
def beginlogochar(expr code, unit_width) =
beginchar(code,if courrier : larg_tt# else :
 unit_width*u#+2s# fi ,ht#,0);
pickup logo_pen enddef;
```

et, tout comme pour la variable *serif*, on prendra la précaution d'ajouter, dans le préambule du fichier pilote, la ligne suivante :

```
if unknown courrier : boolean courrier; courrier:=false fi;
```

La figure 5.34 montre la forme de l'empattement qui sera adoptée pour la version «courrier». C'est toujours la même macro *empattement* qui fera les empattements des lettres ; simplement nous allons, encore une fois, modifier sa définition afin que, suivant que l'on souhaite obtenir telle ou telle forme de la police, ce soit l'empattement correspondant qui soit tracé :

```
def empattement (suffix $)(expr empat_g,empat_d,
 empat_e,empat_l,empat_h,theta,pos)=
cutoff(z$,90-pos);penpos$(empat_l/abs sind theta,0);
x$c=x$-empat_g;
x$d=x$+empat_d;
y$c=y$d=bot y$;
x$b=x$c; x$e=x$d;
y$b-y$c=y$e-y$d=empat_e;
z$a-z$l=z$f-z$r=(empat_h/abs sind theta)* dir theta;
z$g=whatever[z$a,z$l];y$g=y$b;
z$h=whatever[z$f,z$r];y$h=y$e;
fill (z$a{z$l-z$a} if courrier : -- z$g -- else :
.. {left}fi z$b--z$c--z$d--z$e{left} if courrier :
-- z$h -- else : .. {z$f-z$r}fi
z$f--cycle) rotatedaround(z$,pos)
enddef;
```

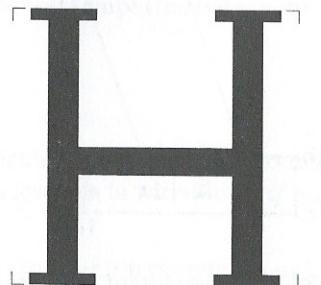


FIG. 5.35 – Empattements machine à écrire

Afin d'alléger l'écriture des commandes d'empattement, nous avons défini deux macros complémentaires, l'une pour les empattements inférieurs et l'autre pour les empattements supérieurs :

```
def empatinf(text t)=
forsuffixes i=t:
 empattement(i,empat_g,empat_d,empat_e,empat_l,
 empat_h,90,0); endfor enddef;

def empatsup(text t)=
forsuffixes i=t:
 empattement(i,empat_g,empat_d,empat_e,
 empat_l,empat_h,90,180); endfor enddef;
```

La figure 5.35 a été obtenue en retenant les valeurs suivantes :

```
courrier:=true;
serif:=true;
empat_g := 1.6u;
empat_d := 1.6u;
empat_e := .4py;
empat_l := px ;
empat_h := 2py ;
```

et en écrivant laconiquement à la fin de la définition de la lettre H :

```
if serif :
 empatinf(1,6);
 empatsup(3,4) fi;
```

La figure 5.36 montre la lettre W, la plus large de la police logocomplet, dans sa déclinaison « courrier ».



FIG. 5.36 – Le W ramené à la largeur 14u

De la sorte, on construit une police qui pourrait s'appeler `logocomplettt` et serait obtenue à partir d'un fichier de paramètres `logocomplettt10`:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

### 5.4.3 Style flamboyant

Libre à chacun de choisir la forme et les dimensions de ses empattements ou autres ornements. On créera sans difficultés un style flamboyant avec, par exemple, cet ornement nouveau que nous baptiserons *flamme*, qui fait appel à plusieurs stylos différents, comme illustré par la figure 5.37:

```
def flamme (suffix $)(expr empat_g,empat_d,empat_p,empat_h,phi)=
pickup pencircle;
penpos$a(0,phi);penpos$(py,90);penpos$b(0,phi);
x$a=x$ -empat_g;
y$a=y$-empat_p;
x$b=x$+empat_d;
y$b=y$+empat_h;
fill z$a{dir phi}..z$1..{dir phi}z$b{-dir phi}
..z$r..{-dir phi}z$a..cycle;
pickup logo_pen enddef;
```

Tout le travail effectué jusque là ne donne bien entendu qu'un tout premier aperçu de tout ce qu'il est possible de faire pour créer des caractères avec METAFONT: il aurait fallu varier les stylos utilisés, soigner les épaisseurs des traits, faire attention aux pleins et aux déliés, etc. Il a été dit plus haut que la police logocomplet avait une valeur didactique plus qu'esthétique mais on aura compris le haut degré de perfection et de raffinement qu'il est possible d'atteindre avec METAFONT qui a le même souci de la qualité poussé à l'extrême que TeX lui-même. L'illustration de la figure 5.38

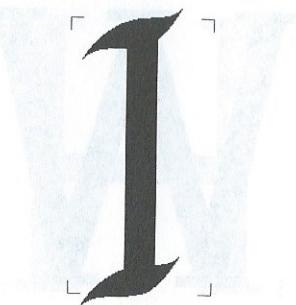


FIG. 5.37 – Style flamboyant

tirée de la police d'initiales *yinit* due à Y. Haralambous<sup>12</sup> est éloquente à ce sujet. Le paragraphe suivant inventorie d'autres possibilités.

## 5.5 Variations

### 5.5.1 Contours

D'autres formes peuvent être facilement obtenues : c'est le cas par exemple des contours. Voici une macro qui remplacera les lettres tracées par leur contour comme sur la figure 5.39 :

```
def trace_contour =
 cull currentpicture keeping (1,infinity);
 image := currentpicture;
 addto currentpicture also image + image shifted 2left
 + image shifted 2right + image shifted 2up
 + image shifted 2down;
 cull currentpicture keeping (1,4);
enddef;
```

On comprend bien comment fonctionne cette petite routine : on translate la figure d'un pixel vers la gauche, puis vers la droite, puis vers le haut et enfin vers le bas en additionnant les pixels à chaque fois. Un pixel intérieur, c'est-à-dire qui serait entouré de pixels dans les quatre directions, verrait sa valeur augmenter de quatre unité. L'instruction **cull** fait disparaître précisément tous les pixels de valeur supérieure ou égale à 5. La première instruction **cull** a pour but de supprimer préalablement

<sup>12</sup>. Voir à l'annexe E.



FIG. 5.38 – Lettrine H de la police yinit

d'éventuels pixels négatifs. La lettre H de la figure 5.39 a ainsi été obtenue en ajoutant l'instruction :

```
trace_contour;
```

avant le **endchar**.

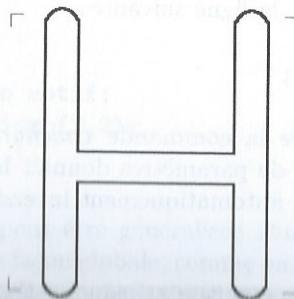


FIG. 5.39 – Contour d'un caractère

### 5.5.2 Négatifs

On peut aussi très facilement inverser les images des caractères pour en obtenir le négatif: lettre blanche sur fond noir. Définissons une macro intitulée *negatif* comme suit :

```
def negatif =
if neg = true : pickup pensquare scaled px;
filldraw (0,0)--(w,0)--(w,h)--(0,h)--cycle;
cull currentpicture keeping (1,1) ; fi enddef;
```

qui agit en fonction d'une variable booléenne nommée *neg*. Il suffira d'écrire l'instruction *negatif*; juste avant la commande *endchar* pour obtenir l'effet souhaité comme sur la figure 5.40.

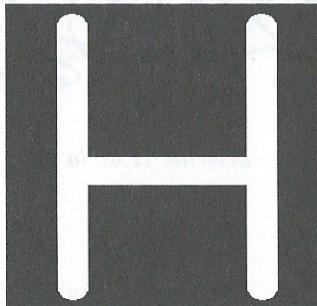


FIG. 5.40 – *Négatif*

Si l'on prévoit que cet effet de négatif devra être appliqué à une police entière, l'instruction **extra\_endchar** va permettre de réaliser cet objectif sans avoir à écrire «*negatif*;» à chaque instruction *endchar*. Il a été en effet expliqué au paragraphe 3.6 p. 54 que l'instruction **extra\_endchar** permettait de passer, sous forme d'une chaîne de caractères, des instructions supplémentaires à la commande **endchar**. On écrira, en préambule du fichier pilote, la ligne suivante :

```
extra_endchar:="negatif";
```

Ainsi, à chaque invocation de la commande *endchar*, la macro *negatif* sera tout d'abord exécutée: si le fichier de paramètres donnait la valeur *true* pour la variable booléenne *neg* alors on aurait automatiquement inversion des pixels blancs et noirs, ce qui conduit à ceci<sup>13</sup>:

**ABCDEFGHIJKLMNPQRSTUVWXYZ**

<sup>13</sup>. Par souci de lisibilité, nous avons ici appliqué un grossissement : *mag=magstep(2)*.

### 5.5.3 Motifs

Outre les contours et les négatifs, il est aussi possible de créer une macro qui réalisera des motifs et permettra de créer un style de caractères fantaisistes comme sur l'exemple de la figure 5.41 où l'intérieur de la lettre est remplacé par la texture du motif. Un motif bien choisi permettra aussi de donner une impression de grisé : c'est le principe même de l'impression en niveaux de gris qui repose sur l'illusion d'optique suscitée par une trame. On pourra ainsi constituer des polices en divers niveaux de gris suivant la trame choisie. Dans un premier temps, fabriquons un motif simple en forme de grille à trame horizontale :

```
u#:=4/9pt#;
v#:=.1u#;
picture motif;
pickup pensquare scaled .05pt;
for i = 0 step 5v until 140v : draw (i,0)--(i,140v);
draw (0,i)--(140v,i); endfor;
motif := currentpicture;
cull motif keeping (1,1);
```

L'exemple de la figure 5.41 est ainsi obtenu en appliquant le motif précédent (contenu dans la variable de type *picture* intitulée *motif*) sur la lettre H et en opérant une sélection de pixels grâce à l'instruction **cull**. Ceci peut être réalisé de façon très simple en modifiant les définitions déjà données de la lettre H de la manière suivante :

```
beginlogochar("H",14);
x1=x2=x3=leftstemloc;
x4=x6=w-x1; x5=x4;
y1=y6; y2=y5; y3=y4;
bot y1=-o; top y3=h+o;
y2=barheight;
draw z1--z3;
draw z4--z6;
draw z2--z5;
cullit ;
addto currentpicture also motif;
cull currentpicture keeping (2,2);
endchar;
```

Bien entendu, la procédure peut être généralisée afin de s'appliquer à tous les caractères d'une même police de façon globale, comme on l'a vu au paragraphe précédent pour les effets de négatifs. Il suffit pour cela de définir une macro qui effectuera le tracé du motif puis de modifier la définition de la commande **endchar** grâce à l'instruction **extra\_endchar**. Le motif (en forme de grille) dépendra de trois paramètres *taille*, *pas*, *dimen* dont la signification est évidente : *taille* est la taille en pixels du stylo, donc

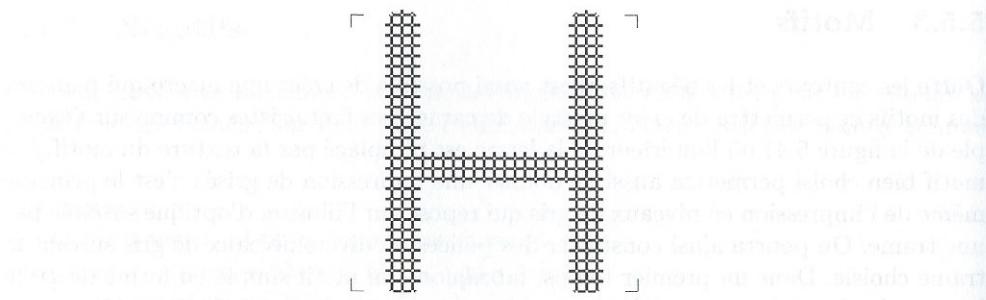


FIG. 5.41 – Lettre avec motif

l'épaisseur des traits de la grille ; *pas* est le pas de la grille, donc en quelque sorte l'espacement des traits ; *dimen* est la dimension du motif qui est ici un carré. Voici donc les instructions concernant le motif que l'on fera figurer parmi les définitions réunies dans le préambule de la police :

```
picture motif,caract;
caract:=currentpicture;
currentpicture=nullpicture;
pickup pensquare scaled taille;
for i = 0 step pas until dimen : draw (i,0)--(i,dimen);
draw (0,i)--(dimen,i); endfor;
motif := currentpicture;
currentpicture:=caract;
cull motif keeping (1,1);
```

La variable de type *picture* intitulée *caract* sert à préserver le travail en cours accompli éventuellement par METAFONT. On n'oubliera pas, d'autre part, de donner dans le préambule une instruction de pixellisation pour les trois paramètres *taille*, *pas* et *dimen*:

```
define_pixels(taille,pas,Dimen);
```

Dans ce même préambule ou bien dans un fichier séparé réunissant toutes les macros (une nouvelle base de macros, comme expliqué à l'annexe F), figurera la définition d'une nouvelle macro baptisée *trace\_motif* :

```
def trace_motif(expr taille,pas,Dimen)=
cullit;
addto currentpicture also motif;
cull currentpicture keeping (2,2);
enddef;
```

Pour terminer, on écrira simplement la ligne suivante avant la première instruction *beginchar* :

```
extra_endchar := "trace_motif(taille,pas,dimen);";
```

De la sorte, chaque fois que METAFONT rencontrera une instruction *endchar*, il appliquera la macro *trace\_motif* au caractère en cours avant de l'expédier vers le fichier de sortie (le fichier *gf*). Voici, à titre d'exemple, deux applications de ce principe obtenues respectivement avec les valeurs suivantes<sup>14</sup> :

```
taille# := .4pt#;
pas# := 10v#;
dimen# := 180v#;
```

puis

```
taille# := .05pt#;
pas# := 5v#;
dimen# := 180v#;
```

A B C D E F G H I J K L M  
N O P Q R S T U V W X Y Z

A B C D E F G H I J K L M  
N O P Q R S T U V W X Y Z

Toutes les possibilités suggérées dans ce chapitre peuvent être combinées entre elles et ne sont certainement qu'une infime partie de tout ce que l'on pourrait encore réaliser : la seule limite est l'imagination du créateur.

## 5.6 Questions d'approximation

Le placement des points de construction est *a priori* dicté par des considérations géométriques : mais la position qui leur est assignée, une fois les calculs accomplis, peut conduire, au moment de la pixellisation, à des distorsions ou à des déformations

14. Ces valeurs peuvent être données dans le fichier paramètres ou bien dans le préambule du fichier pilote comme on l'a vu plus haut.

indésirables. Il peut être parfois judicieux de déplacer un point d'une quantité infinitésimale — ce qui ne modifie pas fondamentalement la forme de la lettre — pour obtenir un tracé meilleur. D'où l'importance de macros telles que **good.x**, **good.y**, **good.lft**, **good.rt**, **good.top** ou **good.bot**.

Cela amène à préciser comment est faite la pixellisation d'une image par METAFONT. Par commodité, on se représente les pixels comme des petits carrés élémentaires et indivisibles : ces carrés seront noirs ou blancs. Il est clair que si le carré est entièrement contenu dans l'image, il sera noir et que s'il est complètement extérieur il sera blanc. La question se pose pour les carrés qui se trouvent sur le pourtour de l'image, autrement dit ceux dont une partie se trouve à l'intérieur tandis que l'autre est à l'extérieur de l'image. La règle est que le pixel sera noir si le centre du carré se trouve à l'intérieur de l'image (ou à la rigueur sur la frontière) ; si le centre est à l'extérieur strictement, alors le pixel sera blanc.

Prenons un exemple concret : supposons que l'on veuille tracer un trait vertical avec un stylo à pointe circulaire de 0,48 points de diamètre. Nous allons considérer, pour avoir des valeurs simples, qu'il s'agit de gros points (*big points*, *bp*) plutôt que de points d'imprimeur (il y en a 72 par pouce et non 72,27) et que nous disposons d'une imprimante à 600 dpi (*dots per inch*, pixels par pouce). L'épaisseur des traits tracés par ce stylo sera, en pixels, de :

$$0,48 \text{ bp} \times 600 / 72 = 4$$

Deux cas de figure se présentent essentiellement :

- si le centre du stylo se trouve placé sur une coordonnée entière en pixels, autrement dit sur le bord d'un de ces petits carrés élémentaires, on aura deux pixels à gauche et deux pixels à droite qui se propageront verticalement. Le trait aura bien pour épaisseur 4 pixels ;
- si l'abscisse du centre, au contraire, se trouve exactement au milieu d'un carré, alors le bord gauche et le bord droit de la pointe du stylo se trouveront eux aussi sur le centre d'un carré (deux carrés plus loin à gauche ainsi qu'à droite). D'après la règle donnée plus haut, ces carrés ayant leur centre sur la frontière du tracé seront noircis : le trait ainsi élargi d'un demi carré à gauche et d'un demi carré à droite aura une épaisseur de 5 pixels et sera donc vingt-cinq pour cent plus épais que ce qui était souhaité : 5 pixels correspondent à une pointe de stylo de diamètre 0,6 *bp*.

Le deuxième cas envisagé ci-dessus ne se présenterait pas si on imposait que la partie gauche du stylo ait une valeur entière en pixels : c'est précisément ce que réalisent les macros **good.x**, **good.lft**, etc. Cette situation se produit aux points par lesquels passent des traits verticaux<sup>15</sup> et plus généralement aux points ayant une tangente verticale (ou horizontale). Ce sont, de manière générale, ces points qui réclament une attention particulière au moment où on les définit.

---

<sup>15</sup>. Ou horizontaux : le raisonnement est bien entendu le même horizontalement.

Dans une situation quelque peu similaire, on a eu recours également à la macro **change\_width** (voir la définition de la lettre T dans la police logocomplet).

On peut se demander d'autre part ce qu'il se passe lorsque la pointe du stylo est inclinée : par exemple s'il s'agit d'un stylo à pointe elliptique et que les axes de cette ellipse se trouvent inclinés par rapport à l'horizontale. Les instructions **good.x**, et apparentées sont intelligentes (ou du moins peuvent être rendues telles grâce à la variable *autorounding*) : si la variable *autorounding* est strictement positive, elles tiendront compte de l'inclinaison et corrigent la coordonnée des points de construction en fonction du point du stylo, gauche ou droit, qui a une tangente verticale pour faire en sorte que cette coordonnée soit un nombre entier en pixels. Si *autorounding* de surcroît est  $> 1$ , alors les points dont les tangentes sont inclinées de  $\pm 45^\circ$  seront aussi pris en compte pour une meilleure pixellisation.

Une autre variable interne dénommée *smoothing* a aussi pour but d'améliorer le rendu des courbes pixellisées : si  $smoothing > 0$ , un algorithme est déclenché qui rend la progression des pixels plus régulière.

En réalité, il faut être conscient que aussi bien *autorounding* que *smoothing* peuvent produire des résultats surprenants et parfois même indésirables. S'ils sont conçus en principe pour améliorer les choses, c'est seulement en faisant des essais qu'on pourra être sûr qu'ils agissent à bon escient.

## 5.7 Lisibilité des caractères

La lisibilité d'un texte doit beaucoup à la police de caractères employée. Certains styles de caractères sont choisis pour la faculté qu'ils ont d'attirer l'œil : ce sont souvent des caractères fantaisistes et hors norme dont la fonction est principalement de capter l'attention du lecteur. D'autres sont choisis pour favoriser une lecture rapide et sont conçus pour occasionner le minimum de fatigue. Dans tous les cas, il faut être conscient de certaines propriétés optiques qui conditionnent notre perception des caractères imprimés. On tiendra compte soit en les acceptant, soit en les rejetant suivant le but recherché. Voici donc quelques observations :

1. à épaisseur égale, un trait horizontal paraîtra plus gras qu'un trait vertical comme on peut le constater sur les deux lettres E de la figure 5.42 (le E de droite est corrigé par une épaisseur moindre des traits dans le sens horizontal) ;
2. à hauteur égale, des lettres de forme triangulaire ou circulaire paraîtront plus petites que des lettres de forme carrée. C'est la raison pour laquelle on prend soin d'ajouter aux premières une petite quantité *o* (pour *overshoot*, débordement). Ce phénomène se trouve illustré par la figure 5.43 sur laquelle le triangle et le cercle du bas ont été corrigés ;
3. si une lettre possède une partie supérieure et une partie inférieure, à taille égale la partie supérieure paraîtra plus grande que la partie inférieure. Les deux lettres X de la figure 5.42 en sont l'illustration : la deuxième a été corrigée. Dans la

police logocomplet développée dans ce chapitre, cette règle a été appliquée pour le X mais a été contournée pour ce qui est de la hauteur des barres horizontales médianes qui ont été volontairement placées — dans A, E, F, H, etc. — en dessous du milieu géométrique ( $\text{barheight} = 0,45 h < 0,5 h$ ) ;

4. les points d'intersection de deux traits dans une lettre ont un poids optique renforcé. Il est habituel de compenser cela en diminuant l'épaisseur des traits à proximité des points d'intersection ;
5. une lettre verticale — telle que F ou P — paraîtra plus stable et plus équilibrée si la partie droite de la base de l'empattement est légèrement plus grande que la partie gauche : avec un empattement symétrique, on a l'impression que la lettre tombe vers la droite ;
6. les proportions respectives des caractères ont aussi une grande importance. Le tableau B.3 dans l'annexe B.2 propose des valeurs qui semblent optimales : ces sont celles adoptées pour la police cmr (Computer Modern Roman).

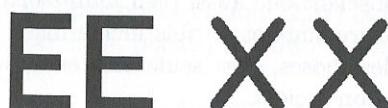


FIG. 5.42 – *Les lettres E et X avant et après correction*

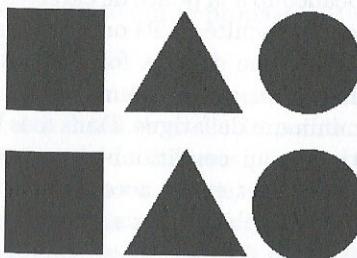


FIG. 5.43 – *Effets optiques*

Encore une fois, ces observations conduisent à des règles de lisibilité que l'on peut aussi bien respecter que contourner : c'est parfois en les prenant à contrepied que l'on donne une personnalité à un ensemble de caractères.

## 5.8 Vers de nouvelles métapolices

Des exemples de polices très diverses sont présentés à l'annexe E. Mais la famille Computer Roman est quasiment la seule métapolice complète disponible depuis des

années. Il est très étonnant de constater qu'il n'existe pas d'alternative véritable. Pourtant l'extrême raffinement qui peut être atteint dans le cadre de la création d'une police de caractères et la puissance de la notion de métapolice devraient susciter l'apparition de nouvelles familles de polices à l'instar de la cmr. Un auteur qui souhaite échapper à cette police cmr n'a guère d'autre possibilité que de se rabattre sur des polices PostScript (Times, Helvetica, Lucida, etc.) qu'il adaptera tant bien que mal à la richesse potentielle et aux exigences de T<sub>E</sub>X (avec non seulement ses caractères alphabétiques mais aussi ses symboles mathématiques<sup>16</sup>). Souhaitons donc que des créateurs imaginatifs auront le courage de relever ce défi en offrant à la communauté des utilisateurs de T<sub>E</sub>X de nouvelles familles complètes.

## 5.8.2 Les différentes méthodes d'optimisation

Il existe deux types de méthodes pour optimiser les polices de caractères : celles qui visent à améliorer la qualité de rendu des caractères et celles qui visent à réduire leur taille. Ces deux types sont étroitement liés, car une police optimisée pour la taille peut également être optimisée pour la qualité de rendu et vice versa. Cependant, il est important de comprendre que ces deux types de méthodes sont souvent en conflit les uns avec les autres.

La première méthode consiste à optimiser les polices de caractères pour la qualité de rendu. Cela peut être fait en utilisant des techniques telles que l'anti-aliasage, qui consiste à ajouter des pixels supplémentaires aux bords des caractères pour les rendre plus fluides et moins épaissis. Cela peut également être fait en utilisant des algorithmes de tracage plus sophistiqués pour créer des contours plus précis.

La deuxième méthode consiste à optimiser les polices de caractères pour la taille. Cela peut être fait en utilisant des techniques telles que la compression de la police, qui consiste à réduire la taille du fichier sans perdre trop de qualité. Cela peut également être fait en utilisant des algorithmes de tracage plus efficaces pour créer des contours plus courts et moins complexes. Cela peut également être fait en utilisant des techniques de codage plus efficaces pour réduire la taille du fichier.

Il existe plusieurs façons de combiner ces deux types de méthodes. Par exemple, on peut utiliser une technique d'optimisation pour la taille pour réduire la taille du fichier, puis une technique d'optimisation pour la qualité de rendu pour améliorer la qualité de rendu. Cela peut également être fait en utilisant une technique d'optimisation pour la taille pour réduire la taille du fichier, puis une technique d'optimisation pour la qualité de rendu pour améliorer la qualité de rendu. Cela peut également être fait en utilisant une technique d'optimisation pour la taille pour réduire la taille du fichier, puis une technique d'optimisation pour la qualité de rendu pour améliorer la qualité de rendu.

Il existe plusieurs façons de combiner ces deux types de méthodes. Par exemple, on peut utiliser une technique d'optimisation pour la taille pour réduire la taille du fichier, puis une technique d'optimisation pour la qualité de rendu pour améliorer la qualité de rendu. Cela peut également être fait en utilisant une technique d'optimisation pour la taille pour réduire la taille du fichier, puis une technique d'optimisation pour la qualité de rendu pour améliorer la qualité de rendu.

16. Le trio Times-Helvetica-Courier par exemple ne constitue pas une métapolice car ces polices n'ont pas été spécialement conçues pour cohabiter et ne suffisent pas à elles trois pour rivaliser avec la famille cmr.

# Chapitre 6

## METAFONT au jour le jour

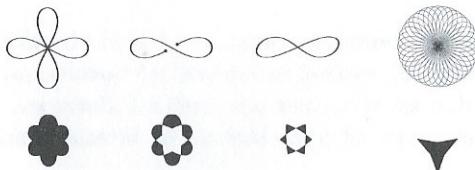
Tout en permettant, comme on l'a vu en détail au chapitre 5, un extrême raffinement dans l'élaboration d'une métapolice, **METAFONT** reste un outil tout à fait simple à mettre en œuvre et, de ce fait, peut être appelé à tout moment pour faire face à des besoins ponctuels. C'est cette utilisation de **METAFONT** au jour le jour que ce court chapitre voudrait mettre en valeur.

### 6.1 Les différents niveaux d'utilisation

**METAFONT** reste relativement méconnu et est en fait peu utilisé dans l'usage quotidien. Il n'est souvent considéré que comme la face cachée de **TeX** alors qu'il est tout aussi riche en fonctionnalités et par ailleurs beaucoup plus facile à utiliser. **METAFONT** permet d'enrichir considérablement les potentialités offertes par **TeX**, quel que soit le format utilisé et tout particulièrement avec **LATEX 2 $\epsilon$**  et le système **NFSS** de gestion des polices de caractères.

**METAFONT** permet en quelques lignes d'instructions de fabriquer les symboles nécessités par un travail particulier. Toutes les disciplines sont concernées par ce genre de problèmes : de l'archéologie et des inscriptions archaïques aux innombrables disciplines scientifiques et techniques, en passant par la musique ou les langues extra-européennes, toutes ont leurs besoins spécifiques en matière de caractères et de symboles particuliers. Ce seul aspect justifie à soi seul l'apprentissage de **METAFONT**.

Voici pour commencer à quoi ressembleront les caractères correspondant aux principaux exemples qui ont été présentés tout au long du chapitre 4 :





## 6.2 Trois exemples pratiques

Pour illustrer ce qui vient d'être dit sur l'utilisation « au quotidien » de METAFONT, les trois exemples proposés seront puisés à trois domaines très variés : la finance, la musique et l'égyptologie. Ils présentent des degrés divers de complexité mais prouvent que le travail de conception et de réalisation d'un symbole isolé peut être effectué en un temps minime.

Avant de se lancer dans la programmation d'un symbole, rappelons quelques conseils pratiques :

- il est tout d'abord recommandé de se munir d'un papier à petits carreaux (voire millimétrique) et d'y faire le croquis du symbole à réaliser afin de placer plus facilement les points de construction ou de contrôle ;
- il faut faire confiance à METAFONT qui sait, avec un minimum de points, tracer des arcs de courbe harmonieux. L'expérience montre qu'il faut laisser une part de la créativité au programme lui-même en ne lui imposant pas trop de points.

### 6.2.1 Le symbole de l'euro

La monnaie qui entre progressivement en usage dans les pays européens de l'U.E.M. (Union Économique et Monétaire) en remplacement des monnaies nationales possède un symbole qui ne figure pas (pas encore !) dans les polices de caractères ordinaires. Avec METAFONT et TeX, on peut créer en quelques lignes d'instructions ce symbole manquant. Il sera stocké dans une police dédiée et appelé, dans un document TeX, par une macro à définir par l'utilisateur et qui pourrait s'intituler « \euro ». Voici le programme dont le résultat est montré, en mode *proof*, sur la figure 6.1.

```

u#:=7/24pt#;
define_pixels(u);
path losange;
losange:=(6,3)--(-3,3)--(-6,-3)--(3,-3)--cycle;
def lospen = makepen losange enddef;
z1 = (18u,0);
z2 = (18u,24u);
z3 = (25u,21u);
z4 = (25u,3u);
z5 = (2u,14u);
ecart:=4.5u;
```

```

theta:=60;
z0 = z3 shifted (-cosd theta,-sind theta);
z6 = whatever [z0,z3];
y6 = ypart z5;
z7 = z5 shifted (0,-ecart);
z8 = whatever [z0,z3];
y8 = ypart z7;
pickup lospen scaled 3;
draw z5--z6;
draw z7--z8;
pickup pencircle scaled 2u;
cutdraw z3..{left}z2..{right}z1..z4;

```

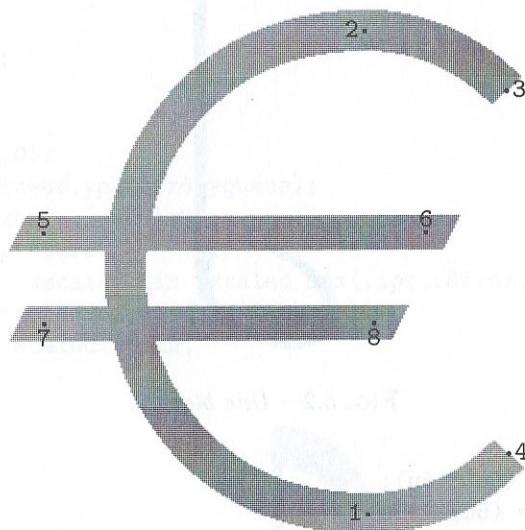


FIG. 6.1 – Le symbole de l'euro

On remarque que les deux barres horizontales de ce symbole ont des extrémités obliques : le moyen le plus simple pour les obtenir a été de créer un stylo spécial à pointe en forme de losange. L'arc en forme de C est obtenu avec seulement quatre points et est dessiné avec un stylo à pointe circulaire : les extrémités ont une coupure nette et non pas arrondie grâce à l'instruction *cutdraw*.

### 6.2.2 Symboles musicaux

Dans un tout autre ordre d'idées, la musique est grande consommatrice de symboles spéciaux. L'auteur d'un ouvrage technique sur le sujet (méthode, analyse, traité, etc.) aura besoin d'insérer ces symboles dans son texte. Or les polices musicales traditionnelles cohabitent habituellement assez mal avec les caractères alphabétiques et ne

sont souvent pas assez riches pour faire face à tous les besoins (en particulier pour la musique contemporaine). Une fois encore METAFONT répondra en quelques lignes de code à l'attente de l'utilisateur.

Voici, à titre d'illustration, comment on peut obtenir des blanches, des noires, des croches et des doubles croches. Pour la blanche tout d'abord (voir la figure 6.2), le problème à résoudre est d'incliner d'un angle  $\theta$  l'ellipse qui représente la tête de la note et de trouver le point de contact d'où partira la hampe, donc le point de l'ellipse à tangente verticale. La méthode utilisée ici consiste à rechercher un point dont la tangente fait un angle  $90 - \theta$ , puis de faire ensuite une rotation d'angle  $\theta$ .



FIG. 6.2 – *Une blanche*

```

z1 = (u,0); z2 = (5u,2.5u);
z3 = (9u,0); z4 = (5u,-2.5u);
z0 = .5[z1,z3];
fin := .2pt;
path chem;
pair incl;
theta:= 60;
incl := (cosd theta,sind theta);
hauteur := 21u;
chem:=z1{up}..{right}z2..{down}z3..{left}z4..{up}z1;
t = directiontime -incl of chem;
z = point t of chem;
z5 = z rotatedarround (z0,90-theta);
z6 = (xpart z5,hauteur);
pickup pencircle xscaled fin yscaled max(.1pt,.6fin);
draw chem rotatedarround (z0,90-theta);
pickup pencircle scaled .8fin;
draw z5--z6;

```

Pour obtenir une noire, il suffit de modifier deux lignes du programme précédent : remplacer la commande **draw** par **filldraw** et, pour que celle-ci fonctionne, transformer le chemin elliptique en cycle en écrivant :

```
chem:=z1{up}..{right}z2..{down}z3..{left}z4..{up}z1..cycle;
```

Pour obtenir ensuite une croche, il suffit d'ajouter une queue à la hampe de la noire. Cette queue resservira pour les doubles, triples, quadruples croches. Elle va donc être stockée en mémoire dans une variable de type *picture* baptisée (qui l'eût cru !) *queue*. Le programme de la figure 6.3 sera le même que précédemment mais complété par les lignes suivantes :

```
...
xqueue := 7u;
yqueue := 16u;
picture queue;
penpos7(2.5u,90);
penpos8(.15u,0);
...
z7 = z6 - (.4fin,0);
z8 = (xpart z6+xqueue,ypart z6-yqueue);
penstroke z7e{right}..{down}z8e;
queue := currentpicture;
pickup pencircle xscaled fin yscaled max(.1pt,.5fin);
filldraw chem rotatedarround (z0,90-theta);
pickup pensquare scaled .8fin;
draw z5--z6;
```



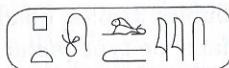
FIG. 6.3 – Une croche

Pour la double croche, on ajoutera à la fin la ligne suivante :

```
addto currentpicture also queue shifted (0,-5u);
```

### 6.2.3 Hiéroglyphes

Plus complexe sera la constitution de polices de hiéroglyphes car ceux-ci ne sont pas toujours des formes géométriques simples. Il s'agit parfois de véritables petits dessins. Nous allons prendre comme exemple un cartouche fameux, celui du roi Ptômélée :



C'est à partir de ce cartouche que Champollion (1790-1832) a pu déchiffrer l'écriture égyptienne hiéroglyphique. Chacun des symboles qui le constituent possède une valeur littérale comme on peut le voir sur le tableau 6.1.

|   |   |
|---|---|
| □ | p |
| ⌇ | t |
| ꙗ | o |
| ꙑ | l |
| ꙓ | m |
| Ꙕ | i |
| ꙕ | s |

TAB. 6.1 – Valeurs littérales des hiéroglyphes

L'intérêt de cet exemple est d'une part de montrer comment METAFONT peut s'acquitter de la tâche de créer un *⌇* et d'autre part de réaliser des constructions du type du cartouche de Ptolémée qui est défini ici comme un seul caractère. La méthode, déjà rencontrée dans cet ouvrage, consiste à stocker progressivement dans des variables de type *picture* les éléments constitutifs pour les assembler ensuite.

Par analogie, METAFONT devrait donc faire également le bonheur des sinologues puisque les idéogrammes chinois sont très souvent des assemblages de caractères élémentaires et pourront donc être construits par ce même procédé.

Voici le programme qui a permis d'obtenir les sept hiéroglyphes ci-dessus (stockés dans une police *hiero* aux positions p, t, o, l, m, i et s) de même que le cartouche de Ptolémée (stocké comme P majuscule) :

```
picture lettrep, lettret, lettrel, lettreo,
 lettrem, lettrei, lettres;
```

```
font_size 24pt#;
```

```
u#:= 2 pt#;
v#:=.5u#;
s#:= u#;
px#:= 3/4pt#;
py#:= .9px#;

define_pixels(s,u,v);
define_blackter_pixels(px,py);

pickup pencircle scaled .2u;
hiero_pen:=savepen;

def beginhierochar(expr code,unit_width,unit_height) =
beginchar(code,unit_width*u#+2s#,unit_height*u#,0);
pickup hiero_pen enddef;

beginhierochar("p",5,5); "Hieroglyphe p";
draw unitsquare scaled 2.5u;
lettrep := currentpicture;
currentpicture := nullpicture;
draw unitsquare scaled 4u shifted (2u,0);
endchar;

beginhierochar("t",5,5); "Hieroglyphe t";
z1= (0,0); z2 = (6u,0);
z3 = (3u,3u); z4 = (4u,0);
z5 = (2u,2u);
draw z1..z5..z4--cycle;
lettret := currentpicture;
currentpicture := nullpicture;
draw z1..z3..z2--cycle;
labels(range 1 thru 3);
endchar;

beginhierochar("o",5,12); "Hieroglyphe o";
z0 = (5u,2u); z1 = (5u,6u);
z2 = (3u,9u); z3 = (1.5u,5.5u);
z4 = (2u,3.5u); z5 = (3u,2u);
z6 = (2u,u); z7 = (1.2u,2u);
z8 = (3.5u,4.5u); z9 = (.5u,5u);
draw z0{z2-z0}..z1..{left}z2..z3..z4..{down}z5{z1}..{right}z6..{left}z7..{up}z8..{left}z4..z9;
lettreo := currentpicture;
labels(range 1 thru 9);
endchar;
```

### 6.1.3 Hiéroglyphes

Il existe une autre manière de dessiner des symboles : à l'aide d'un programme de dessin interactif comme Metapost ou Inkscape. Ces programmes sont très bons pour dessiner des symboles complexes et peuvent être utilisés pour écrire des commandes MF.

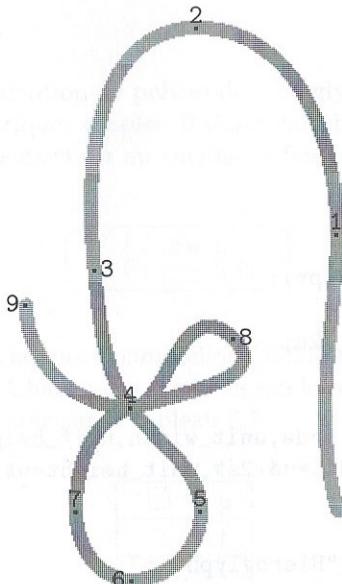


FIG. 6.4 – Hiéroglyphe *o*

```

beginhierochar("l",7,4); "Hieroglyphe 1";
z1 = (0v,0v); z2 = (3v,0v);
z3 = (6.5v,3.5v); z4 = (13v,2v);
z5 = (16v,1v); z6 = (15v,0v);
z7 = (11v,v); z71 = (11v,0);
z8 = (8.5v,0v); z9 = (6v,.3v);
z20 = (14.5v,3.5v);
path p;
pair tangente;
p:= z1--z2{right}..{right}z3..z4..{down}z5..{left}z6..{left}z7
..{left}z8..{left}z9..{left}z2--cycle;
z0=point 1.8 of p; z10=point 2.8 of p;
tangente:=direction 1.8 of p;
z11=round(z0 shifted (v,v));
z12=z11 shifted (-v,1.5v);
z13=z11 shifted (-3v,0);
z14 = z13 shifted (2/3v,-2/3v);
z15=(xpart z12,ypart z11);
z16=.6[z14,z12];
draw z0{tangente}..{up}z11..{left}z12
..{down}z13..z14{z15-z14}..{tangente}z0;
drawdot z16;
draw z12{z12-z13}..{z6-z10}z10;
draw p; draw z6--z71;

```

```

draw z6{z7-z6}..z4..{(1,1)}z20;
lettrel := currentpicture;
labels(range 0 thru 16,20);
endchar;

```

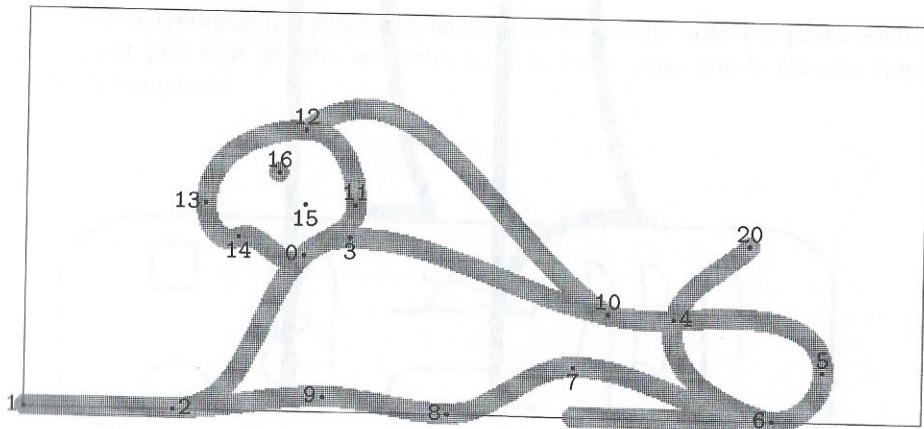


FIG. 6.5 – Hieroglyphe l

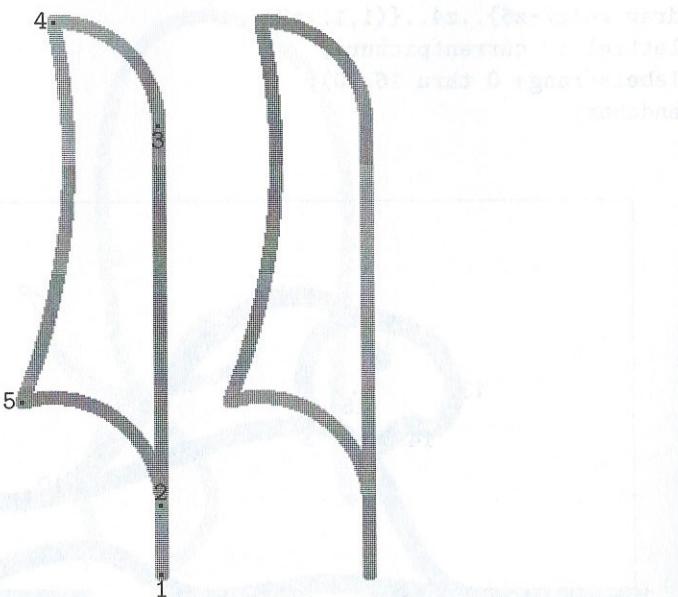
```

beginhierochar("m",6.5,4); "Hieroglyphe m";
z1 = (8u,0); z2 = (u,0);
z3 = (0,.5u); z4 = (2u,2u);
z5 = (8u,2u);
draw z1--z2{left}..{up}z3..{right}z4--z5;
lettrem := currentpicture;
labels(range 1 thru 5);
endchar;

beginhierochar("i",5,12); "Hieroglyphe i";
z1 = (2u,0); z2 = (2u,u);
z3 = (2u,6.5u); z4 = (.5u,8u);
z5 = (0,2.5u);
draw z1--z3{up}..{left}z4;
draw z4{z2-z4}..z5; draw z5..{down}z2;
addto currentpicture also currentpicture shifted (3u,0);
lettrei := currentpicture;
labels(range 1 thru 5);
endchar;

beginhierochar("s",2,12); "Hieroglyphe s";

```

FIG. 6.6 – Hiéroglyphe *i*

```

z1 = (0,0); z2 = (0,6u);
z3 = (u,8u); z4 = (2u,6u);
z5 = (2u,3u);
draw z1-z2{up}..{right}z3..{down}z4--z5;
lettres := currentpicture;
labels(range 1 thru 5);
endchar;

beginhierochar("P",42,16);"Cartouche Ptolemaios";
addto currentpicture also lettrep shifted (5u,8u);
addto currentpicture also lettret shifted (4.5u,2u);
addto currentpicture also lettreo shifted (10u,2u);
addto currentpicture also lettrel shifted (18u,7u);
addto currentpicture also lettrem shifted (18u,2u);
addto currentpicture also lettrei shifted (28u,2u);
addto currentpicture also lettres shifted (38u,2u);
z0 = (0,4u); z1 = (5u,0);
z2 = (38u,0); z3 = (42u,3u);
z4 = (42u,9u); z5 = (38u,12u);
z6 = (5u,12u); z7 = (0,8u);
z8 = (42u,0); z9 = (42u,12u);

```

```

draw z0{down}..{right}z1--z2{right}..{up}z3--z4{up}
 ..{left}z5--z6{left}..{down}z7--cycle;
draw z8--z9;
labels(range 0 thru 9);
endchar;

```

Les lignes d'instructions qui précèdent ne présentent aucune difficulté particulière : on a simplement pris soin de faire en sorte, pour la lettre  $\text{\texttt{A}}$ , que la tête du lion soit tangente à son corps !

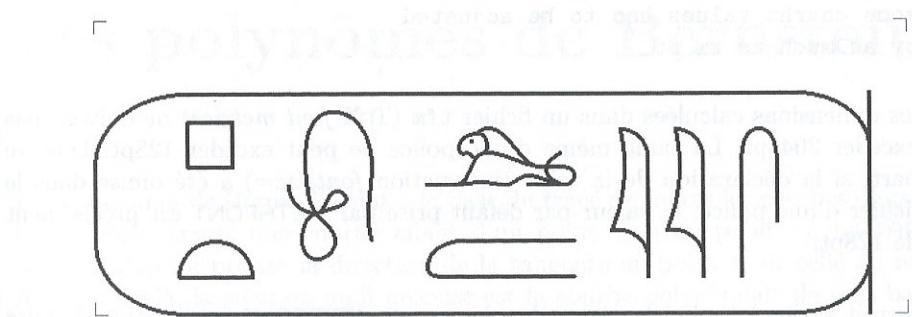


FIG. 6.7 – *Cartouche Ptolemaios*

### 6.3 Les limitations de METAFONT

Malgré toutes ses vertus, METAFONT possède quelques limitations dont il faut être conscient et dont certaines ne vont pas de soi. Dans la pratique, aucune de ces limitations n'est réellement gênante mais il faut néanmoins en tenir compte.

1. METAFONT ne gère pas la couleur. On peut tout au plus, par les astuces expliquées au paragraphe 5.5.3 p. 157, créer des niveaux de gris ;
2. tout nombre doit être inférieur *strictement* à 4096. Si on dépasse cette valeur, METAFONT produit le message d'erreur suivant :
 

! Value is too large
3. il arrive que METAFONT se trouve à cours de mémoire. C'est extrêmement rare et résulte le plus souvent d'une erreur de codage ou bien d'un codage maladroit qui lance METAFONT dans des calculs excessifs. Autrement, il faut vérifier la quantité de mémoire allouée à l'application par le système d'exploitation ;
4. la variable **tolerance** associée à la macro **solve** ne doit jamais être inférieure à **epsilon** qui vaut  $\frac{1}{65536} = 2^{-16}$  ;

5. la variable **withweight** associée à la macro **addto** n'accepte que les valeurs -3, -2, -1, 1, 2 et 3. Si l'on veut obtenir d'autres valeurs de pixels, il faut appliquer la commande *addto* à nouveau ;
6. dans une même police, il ne peut y avoir plus de 15 hauteurs différentes de caractères, plus de 15 profondeurs différentes de caractères et plus de 63 corrections italiques. Si ces limites sont dépassées, METAFONT modifiera certains de ces paramètres de la plus petite quantité possible puis le signalera dans le fichier log par le message suivant :

```
some charht values had to be adjusted
by as much as xx pt
```

7. les dimensions calculées dans un fichier **tfm** (*TEX font metrics*) ne doivent pas excéder 2048pt. La taille même d'une police ne peut excéder 128pt. D'autre part, si la déclaration de la taille (instruction *fontsize=*) a été omise dans le fichier d'une police, la valeur par défaut prise par METAFONT est précisément de 128pt.

On pourrait varier à l'infini les exemples de ce que METAFONT est capable de faire dans toutes sortes de domaines mais le lecteur qui a assimilé cet ouvrage saura lui-même définir ses projets. Tous les apports qui feront croître la bibliothèque de polices déjà disponibles (voir à l'annexe H p. 241 les adresses des archives CTAN) permettront d'étendre l'usage de *TEX* à de nouveaux domaines et ne pourront que contribuer au rayonnement de METAFONT pour le plus grand bénéfice des utilisateurs.

## Annexe A

# Les polynômes de Bernstein

Les polynômes de Bernstein sont à la base du tracé de toutes les courbes dans METAFONT. Pour tracer une courbe allant d'un point  $z_1$  à un point  $z_2$ , METAFONT a besoin qu'on lui précise la direction de la tangente au point  $z_1$  et celle au point  $z_2$ . À partir de là, la solution qu'il propose est la courbe polynomiale de plus bas degré possible satisfaisant à ces contraintes. Ce problème a été étudié par le mathématicien Sergueï Bernstein et la solution s'écrit très simplement sous forme d'équation paramétrique : si  $z_3$  est un point de la tangente à  $z_1$  et  $z_4$  un point de la tangente à  $z_2$  on aura :

$$z(t) = (1-t)^3 z_1 + 3(1-t)^2 t z_3 + 3(1-t)t^2 z_4 + t^3 z_2$$

Cette équation s'appelle polynôme de Bernstein de degré 3 ou encore courbe de Bézier (du nom de Pierre Bézier qui a montré tout l'intérêt que ces courbes pouvaient avoir en informatique). Les points  $z_1$  et  $z_2$  sont des points de construction,  $z_3$  et  $z_4$  sont des points de contrôle.

Ces courbes ont une propriété géométrique remarquable : si on désigne par  $z_{13}$  le milieu du segment  $[z_1, z_3]$ , par  $z_{34}$  le milieu du segment  $[z_3, z_4]$  et par  $z_{42}$  le milieu du segment  $[z_4, z_2]$ , puis par  $z_{134}$  et  $z_{342}$  les milieux respectifs de  $[z_{13}, z_{34}]$  et de  $[z_{34}, z_{42}]$ , le segment  $[z_{134}, z_{342}]$  est tangent *en son milieu*  $z_{1342}$  à la courbe de Bézier. Cette construction peut bien entendu être répétée à l'infini : par exemple avec les points  $z_1$  et  $z_{1342}$  en prenant  $z_{13}$  et  $z_{134}$  comme points de contrôle : voir la figure A.1.

De manière plus générale, un polynôme de Bernstein de degré  $n$  correspond à la donnée de  $n+1$  points  $a_1, \dots, a_{n+1}$  et s'écrit paramétriquement :

$$z(t) = \sum_{i=1}^n \binom{i-1}{n} (1-t)^{n-i} t^{i-1} a_i$$

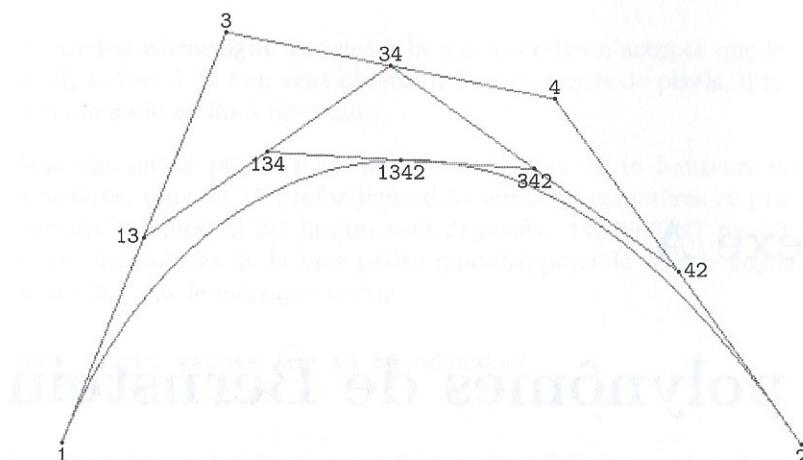


FIG. A.1 – Courbe de Bézier: construction des milieux

Les polynômes de Bernstein peuvent s'obtenir par récurrence. Si on convient de noter, pour deux points  $\alpha$  et  $\beta$ ,

$$t[\alpha, \beta] = t\alpha + (1 - t)\beta$$

on obtient, en fait, le polynôme de Bernstein défini par les seuls points  $\alpha$  et  $\beta$ : c'est l'équation de la droite passant par ces deux points. Le polynôme de Bernstein de degré 2 défini par trois points  $\alpha$ ,  $\beta$  et  $\gamma$  sera  $t[\alpha, \beta, \gamma]$  avec la convention :

$$t[\alpha, \beta, \gamma] \stackrel{\text{def}}{=} t[t[\alpha, \beta], t[\beta, \gamma]]$$

Le polynôme de Bernstein de degré 3 défini, comme ci-dessus, par quatre points  $\alpha$ ,  $\beta$ ,  $\gamma$  et  $\delta$  sera  $t[\alpha, \beta, \gamma, \delta]$  avec la convention :

$$t[\alpha, \beta, \gamma] \stackrel{\text{def}}{=} t[t[\alpha, \beta, \gamma], t[\beta, \gamma, \delta]]$$

et ainsi de suite...

## Annexe B

# Tables de conversion

### B.1 Unités de mesure

Voici les unités de mesure reconnues par METAFONT: pouce, point d'imprimeur, gros point, centimètre, millimètre, didot et cicero. Elles sont liées entre elles par les relations suivantes :

$$\begin{aligned}1 \text{ in} &= 72,27 \text{ pt} \\1 \text{ in} &= 2,54 \text{ cm} \\1 \text{ in} &= 25,4 \text{ mm} \\1 \text{ in} &= 72 \text{ bp} \\1157 \text{ dd} &= 1238 \text{ pt} \\1 \text{ cc} &= 12 \text{ dd}\end{aligned}$$

On aura donc les conversions apparaissant dans le tableau B.1.

On a vu d'autre part que METAFONT ne connaît de nombres strictement positifs qu'entre  $\frac{1}{65536} = 2^{-16}$  et 4095. Le premier est nommé *epsilon* et le second est presque l'infini : la variable dénommée *infinity* vaut  $4096 = 2^{12} - \text{epsilon}$ . METAFONT fonctionne suivant une arithmétique un peu particulière : tous ses calculs sont faits en utilisant des multiples entiers de *epsilon*, ce qui conduit d'ailleurs parfois à des arrondis curieux (qui restent cependant tout à fait négligeables au regard de la taille d'un pixel).

Il est intéressant de se représenter aussi la taille d'un pixel. Cela permet en particulier de mieux comprendre le problème des arrondis que METAFONT effectue dans ses calculs et la difficulté d'avoir des caractères parfaits en basse résolution. La taille réelle d'un pixel dépend bien évidemment de la résolution. Les valeurs sont indiquées dans le tableau B.2 suivant :

| <i>Valeurs en pouces</i> |                           | <i>Valeurs en mm</i> |
|--------------------------|---------------------------|----------------------|
| point d'imprimeur        | $\frac{1}{72,27}$ pouce   | 0,013837 pouce       |
| gros point               | $\frac{1}{72}$ pouce      | 0,013888 pouce       |
| centimètre               | $\frac{1}{2,54}$ pouce    | 0,39370 pouce        |
| millimètre               | $\frac{1}{25,4}$ pouce    | 0,03937 pouce        |
| didot                    | $\frac{1238}{1157}$ point | 0,01481 pouce        |
| cicero                   | 12 didots                 | 0,17767 pouce        |
|                          |                           | 1 in = 25,4 mm       |
|                          |                           | 1 pt = 0,3515 mm     |
|                          |                           | 1 bp = 0,3527 mm     |
|                          |                           | 1 pc = 4,2175 mm     |
|                          |                           | 1 dd = 0,3760 mm     |
|                          |                           | 1 cc = 4,5128 mm     |

TAB. B.1 – *Tables de conversions en pouces et en millimètres*

|            |                                                   |
|------------|---------------------------------------------------|
| à 72 dpi   | 1 ppxl = 0,35277 mm soit environ 3 pixels par mm  |
| à 120 dpi  | 1 ppxl = 0,21167 mm soit environ 5 pixels par mm  |
| à 300 dpi  | 1 ppxl = 0,08467 mm soit environ 12 pixels par mm |
| à 600 dpi  | 1 ppxl = 0,04233 mm soit environ 24 pixels par mm |
| à 1200 dpi | 1 ppxl = 0,02117 mm soit environ 47 pixels par mm |
| à 2400 dpi | 1 ppxl = 0,01058 mm soit environ 95 pixels par mm |

TAB. B.2 – *Nombre de pixels par millimètres à diverses résolutions*

## B.2 Proportions relatives des caractères

Il n'y a pas de règle absolue concernant la taille des caractères de même que leurs proportions (rapport entre la hauteur et la largeur, largeurs ou hauteurs des uns par rapport aux autres, etc.) car tout dépend de l'effet typographique recherché. Si l'objectif est la plus grande lisibilité et le plus grand confort de lecture, il faut rechercher quelles seront les proportions les plus harmonieuses. À titre indicatif nous donnons dans la table B.3 les valeurs absolues et relatives retenues pour la police Computer Roman. Les largeurs des caractères sont exprimées en multiples d'une même unité *u*. Celle-ci vaut par exemple 20/36 point pour la police cmr10. Pour les majuscules, la lettre la plus large est le *W*, pour les minuscules c'est le *m*. La lettre la plus étroite est le *i*. Les hauteurs et profondeurs des caractères sont tout aussi importantes mais sont généralement plus uniformes :

- les lettres b, d, f, h, i, j, k, l, t ont même hauteur (dite *asc.height*) ;
- les lettres a, c, e, g, m, n, o, p, q, r, s, u, v, w, x, y, z ont même hauteur (dite

| Caractères majuscules | Taille absolue | Taille relative |      | Caractère minuscules | Taille absolue | Taille relative |      |
|-----------------------|----------------|-----------------|------|----------------------|----------------|-----------------|------|
|                       |                | à W             | à I  |                      |                | à m             | à i  |
| A                     | 13 u           | 0,72            | 2,16 | a                    | 9 u            | 0,6             | 1,8  |
| B                     | 12,5 u         | 0,69            | 2,08 | b                    | 10 u           | 0,66            | 2    |
| C                     | 13 u           | 0,72            | 2,16 | c                    | 8 u            | 0,53            | 1,6  |
| D                     | 13,5 u         | 0,75            | 2,25 | d                    | 10 u           | 0,66            | 2    |
| E                     | 12 u           | 0,66            | 2    | e                    | 7,25 u         | 0,48            | 1,45 |
| F                     | 11,5 u         | 0,63            | 1,91 | f                    | 5,5 u          | 0,36            | 1,1  |
| G                     | 14 u           | 0,77            | 2,33 | g                    | 9 u            | 0,6             | 1,8  |
| H                     | 13 u           | 0,72            | 2,16 | h                    | 10 u           | 0,66            | 2    |
| I                     | 6 u            | 0,33            | 1    | i                    | 5 u            | 0,33            | 1    |
| J                     | 9 u            | 0,5             | 1,5  | j                    | 5,5 u          | 0,36            | 1,1  |
| K                     | 13,5 u         | 0,75            | 2,25 | k                    | 9,5 u          | 0,63            | 1,9  |
| L                     | 11 u           | 0,61            | 1,83 | l                    | 5 u            | 0,33            | 1    |
| M                     | 16 u           | 0,88            | 2,66 | m                    | 15 u           | 1               | 3    |
| N                     | 13 u           | 0,72            | 2,16 | n                    | 10 u           | 0,66            | 2    |
| O                     | 14 u           | 0,77            | 2,33 | o                    | 9 u            | 0,6             | 1,8  |
| P                     | 12 u           | 0,66            | 2    | p                    | 10 u           | 0,66            | 2    |
| Q                     | 14 u           | 0,77            | 2,33 | q                    | 10 u           | 0,66            | 2    |
| R                     | 12 u           | 0,66            | 2    | r                    | 7 u            | 0,46            | 1,4  |
| S                     | 10 u           | 0,55            | 1,66 | s                    | 7,1 u          | 0,47            | 1,42 |
| T                     | 13 u           | 0,72            | 2,16 | t                    | 6 u            | 0,4             | 1,2  |
| U                     | 13 u           | 0,72            | 2,16 | u                    | 10 u           | 0,66            | 2    |
| V                     | 13 u           | 0,72            | 2,16 | v                    | 9,5 u          | 0,63            | 1,9  |
| W                     | 18 u           | 1               | 3    | w                    | 13 u           | 0,86            | 2,6  |
| X                     | 13 u           | 0,72            | 2,16 | x                    | 9,5 u          | 0,63            | 1,9  |
| Y                     | 13 u           | 0,72            | 2,16 | y                    | 9,5 u          | 0,63            | 1,9  |
| Z                     | 11 u           | 0,61            | 1,83 | z                    | 8 u            | 0,53            | 1,6  |

TAB. B.3 – Les largeurs des caractères Computer Roman

x\_height);

- les lettres g, j, p, q, y ont même profondeur (dite desc\_depth);
- les lettres a, b, c, d, e, f, h, i, k, l, m, n, o, r, s, u, v, w, x, z, t ont une profondeur nulle.

Par exemple, dans la police cmr10, on a les valeurs :

```
body_height# 270/36pt#
asc_height# 250/36pt#
desc_depth# 70/36pt#
```

Ce que l'on désigne en général par la « taille » d'une police de caractères est la hauteur du plus grand d'entre eux : la parenthèse. Mais cette taille n'est qu'indicative : il y a parfois de grandes différences entre plusieurs polices toutes déclarées comme étant de « taille 10 ».

| 0,4  | 00,0 | 0,0  | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
|------|------|------|-----|-----|-----|-----|-----|-----|-----|
| 1,0  | 00,0 | 0,01 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 2,1  | 00,0 | 0,02 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 3,4  | 00,0 | 0,03 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 5,1  | 00,0 | 0,04 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 7,9  | 00,0 | 0,05 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 1,4  | 00,0 | 0,06 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 1,9  | 00,0 | 0,07 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 2,1  | 00,0 | 0,08 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 2,4  | 00,0 | 0,09 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 2,5  | 00,0 | 0,10 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 2,6  | 00,0 | 0,11 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 2,7  | 00,0 | 0,12 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 2,8  | 00,0 | 0,13 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 2,9  | 00,0 | 0,14 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 3,0  | 00,0 | 0,15 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 3,1  | 00,0 | 0,16 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 3,2  | 00,0 | 0,17 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 3,3  | 00,0 | 0,18 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 3,4  | 00,0 | 0,19 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 3,5  | 00,0 | 0,20 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 3,6  | 00,0 | 0,21 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 3,7  | 00,0 | 0,22 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 3,8  | 00,0 | 0,23 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 3,9  | 00,0 | 0,24 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 4,0  | 00,0 | 0,25 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 4,1  | 00,0 | 0,26 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 4,2  | 00,0 | 0,27 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 4,3  | 00,0 | 0,28 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 4,4  | 00,0 | 0,29 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 4,5  | 00,0 | 0,30 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 4,6  | 00,0 | 0,31 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 4,7  | 00,0 | 0,32 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 4,8  | 00,0 | 0,33 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 4,9  | 00,0 | 0,34 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 5,0  | 00,0 | 0,35 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 5,1  | 00,0 | 0,36 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 5,2  | 00,0 | 0,37 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 5,3  | 00,0 | 0,38 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 5,4  | 00,0 | 0,39 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 5,5  | 00,0 | 0,40 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 5,6  | 00,0 | 0,41 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 5,7  | 00,0 | 0,42 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 5,8  | 00,0 | 0,43 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 5,9  | 00,0 | 0,44 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 6,0  | 00,0 | 0,45 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 6,1  | 00,0 | 0,46 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 6,2  | 00,0 | 0,47 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 6,3  | 00,0 | 0,48 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 6,4  | 00,0 | 0,49 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 6,5  | 00,0 | 0,50 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 6,6  | 00,0 | 0,51 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 6,7  | 00,0 | 0,52 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 6,8  | 00,0 | 0,53 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 6,9  | 00,0 | 0,54 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 7,0  | 00,0 | 0,55 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 7,1  | 00,0 | 0,56 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 7,2  | 00,0 | 0,57 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 7,3  | 00,0 | 0,58 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 7,4  | 00,0 | 0,59 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 7,5  | 00,0 | 0,60 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 7,6  | 00,0 | 0,61 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 7,7  | 00,0 | 0,62 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 7,8  | 00,0 | 0,63 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 7,9  | 00,0 | 0,64 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 8,0  | 00,0 | 0,65 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 8,1  | 00,0 | 0,66 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 8,2  | 00,0 | 0,67 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 8,3  | 00,0 | 0,68 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 8,4  | 00,0 | 0,69 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 8,5  | 00,0 | 0,70 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 8,6  | 00,0 | 0,71 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 8,7  | 00,0 | 0,72 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 8,8  | 00,0 | 0,73 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 8,9  | 00,0 | 0,74 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 9,0  | 00,0 | 0,75 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 9,1  | 00,0 | 0,76 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 9,2  | 00,0 | 0,77 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 9,3  | 00,0 | 0,78 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 9,4  | 00,0 | 0,79 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 9,5  | 00,0 | 0,80 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 9,6  | 00,0 | 0,81 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 9,7  | 00,0 | 0,82 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 9,8  | 00,0 | 0,83 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 9,9  | 00,0 | 0,84 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| 10,0 | 00,0 | 0,85 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |

Il y a une règle générale dans les polices de caractères qui éstime que la proportion entre la hauteur et la largeur d'un caractère dépend de sa taille par rapport aux autres, cette taille dépend de l'effet visuel qu'il a sur l'objectif et la plus grande l'effet et le plus grande taille de caractère, et vice versa, quelles sont les lettres les plus importantes. Ainsi, dans la police de caractères sans serif, pour les lettres les plus importantes, il y a une proportion de 1 à 1,5, pour les lettres les moins importantes, il y a une proportion de 1 à 0,8, mais pour les lettres les plus importantes, il y a une proportion de 1 à 1,2, et pour les lettres les moins importantes, il y a une proportion de 1 à 0,9. Il existe également des exceptions, telles que les lettres accentuées, qui ont une hauteur supérieure à celle des lettres non accentuées, et les lettres diacritiques, qui ont une hauteur inférieure à celle des lettres non diacritiques.

Le tableau ci-dessous répertorie les variables internes de METAFONT et leur fonctionnement. Les commandes qui commencent par *tracing* sont placées dans le fichier log à moins que l'on n'ait fixé *tracingonline* = 1 auquel cas les informations produites apparaîtront à la fois dans le fichier log et sur l'écran.

## Annexe C

# Les variables internes

Un certain nombre de constantes ont été mentionnées dans le chapitre 3, en particulier au paragraphe 3.1.2. Ce sont des variables internes de METAFONT. Nous présentons dans cette section la liste complète de ces quarante quantités (ce sont toutes des variables numériques<sup>1</sup>) et leur explication :

- **tracingonline** : les commandes commençant par *tracing* placent leur résultat dans le fichier log à moins que l'on n'ait fixé *tracingonline* = 1 auquel cas les informations produites apparaîtront à la fois dans le fichier log et sur l'écran ;
- **tracingtitles** : sert à montrer à l'écran les titres des caractères lorsqu'ils existent. Ces titres sont des chaînes de caractères entre guillemets après une déclaration **beginchar**. Par exemple :

```
beginchar(65,largeur,hauteur,profondeur); "nom du caractère"
```

- **tracingequations** : sert à montrer chaque variable dès que sa valeur est connue ;
- **tracingcapsules** : sert à montrer les instructions encapsulées et les variables ;
- **tracingchoices** : sert à indiquer les points de contrôle choisis pour le tracé des chemins ;
- **tracingspecs** : sert à indiquer les subdivisions des chemins avant numérisation ;
- **tracingpens** : sert à montrer les sommets des stylos ;
- **tracingcommands** : sert à afficher les opérations et les instructions avant qu'elles ne soient exécutées ;
- **tracingrestores** : sert à indiquer le moment où un symbole ou bien une quantité interne est modifiée ;

1. Pour celles qui commencent par *tracing*, l'action décrite est accomplie si et seulement si la valeur qui leur est attribuée est différente de 0.

- **tracingmacros**: sert à montrer les macros avant qu'elles ne soient développées ;
- **tracingedges**: sert à montrer les pourtours numérisés au moment de leur calcul ;
- **tracingoutput**: sert à montrer les pourtours numérisés au moment de l'écriture du fichier de sortie ;
- **tracingstats**: sert à inscrire à la fin du fichier log des informations sur la quantité de mémoire utilisée pendant l'exécution d'un travail ;
- **pausing**: écrit les lignes d'instruction à l'écran avant qu'elles ne soient lues ;
- **showstopping**: lorsque *showstopping*= 1, on peut intervenir en cours de processus au moyen de commandes **show** ;
- **fontmaking**: pour demander que soit produit un fichier tfm ;
- **proofing**: pour obtenir des épreuves préliminaires (résolution de 36 pixels par point et pas de fichier tfm) ;
- **turningcheck**: pour vérifier les *turningnumbers* (voir p. 28) et signaler les chemins « étranges » (*strange paths*) ;
- **warningcheck**: pour avertir au cas où une variable devient trop grande (*arithmetic overflow*) ;
- **smoothing**: pour obtenir des numérisations plus élégantes. Consulter le paragraphe 5.6 p. 159 ;
- **autorounding**: pour décaler imperceptiblement les chemins afin d'avoir des tangentes verticales passant par des bons points (à coordonnées entières). Voir au paragraphe 5.6 p. 159 ;
- **granularity**: c'est la taille des pixels utilisée par l'instruction *autorounding* ;
- **fillin**: paramètre pour corriger la noirceur des diagonales ;
- **year**: année courante ;
- **month**: mois courant ;
- **day**: jour courant ;
- **time**: nombre de minutes depuis 0 heure au moment du lancement d'un travail ;
- **charcode**: numéro du prochain caractère qui sera produit ;
- **charext**: code d'extension du prochain caractère ;
- **charwd**: largeur en points du prochain caractère ;
- **charht**: hauteur en points du prochain caractère ;
- **chardp**: profondeur en points du prochain caractère ;

- **charic**: correction italique en points du prochain caractère ;
- **chardx**: déplacement en *x* du périphérique pour produire le prochain caractère (exprimé en pixels) ;
- **chardy**: déplacement en *y* du périphérique pour produire le prochain caractère (exprimé en pixels) ;
- **designsize**: taille déclarée de la police fabriquée ;
- **hppp**: nombre de pixels par pouce dans la direction horizontale ;
- **vppp**: nombre de pixels par pouce dans la direction verticale ;
- **xoffset**: décalage en *x* du caractère produit ;
- **yoffset**: décalage en *y* du caractère produit.

Si l'un des deux éléments des commandes précédentes est mal saisi, l'interprèteur affiche un message correspondant à l'erreur. Dès qu'il rencontre une erreur, il s'arrête, il signale le problème et enclenche une procédure d'analyse. Celui-ci peut être soit de type « frappe la touche H pour de l'aide », de type « aide plus précis que nécessaire » ou de type « aide au débogage de vos erreurs ». Généralement il s'agit de types qui sont souvent assez déroutants, les messages d'erreur étant très courtes et peu éloquentes et, en général, faciles à comprendre (au moins si l'on possède suffisamment bien l'anglais). Il n'a pas été écrit pour un utilisateur qui l'utilise régulièrement, mais lorsque quelque chose va mal, les explications de MCLISP sont très courtes et souvent pleines d'humour (« *I proceed, with fingers crossed* »), ce qui peut l'en réveiller habilement dans les moments de panique. On apprendra que les messages s'expliquent de façon plus adéquate lorsque l'on apprendra que « *dear, I can't handle all the questions about the positive* » est la tournure la plus courante de cette phrase pour éviter la forme « *positive question* » de l'anglais. Nous rentrons dans cette partie la fois que nous parlerons de quelles sont les meilleures méthodes pour se faire aider.

Les erreurs les plus fréquemment rencontrées sont les suivantes :

1. quand une parenthèse à la fin d'une expression, il se crée une situation où l'interprète ne sait pas où l'expression commence en premier ou elle doit fermer de la parenthèse et rentrer également, par exemple, une parenthèse de la manière suivante :
2. une ligne manquante dans la déclaration des variables ou dans une ligne d'abstraction d'un point et ou celle-ci a été mal formée. Cela peut arriver lors d'un début de déclaration d'une variable. Une variable ignorée dont le nom n'a pas été déclaré est considérée par défaut comme variable de type autre chose, ce qui

Le langage offre également la possibilité d'interrompre l'exécution d'une instruction pour effectuer une autre. Cela peut être utile pour corriger des erreurs ou pour empêcher une instruction de se dérouler dans un cycle infini. Il existe deux méthodes pour interrompre l'exécution d'une instruction : *stopmode* et *nonstopmode*. La première méthode (*stopmode*) arrête l'exécution de l'instruction courante et renvoie au programmeur une erreur. La deuxième méthode (*nonstopmode*) arrête l'exécution de l'instruction courante et renvoie une erreur sans arrêter le programme.

## Annexe D

# Les messages d'erreur

Quand il s'agit de corriger des erreurs ou de prodiguer des conseils, METAFONT est un langage particulièrement discret. Dès qu'il rencontre une erreur, il s'arrête<sup>1</sup>, signale le problème et en donne une première analyse. Celle-ci peut être suivie (si l'on frappe la touche H suivie de Return) de conseils plus précis qui tireront, dans la plupart des cas, l'utilisateur de son embarras. Contrairement à ceux de TeX qui sont souvent assez hermétiques, les messages d'erreur de METAFONT sont très documentés et, en général, faciles à comprendre (du moins si l'on parle suffisamment bien l'anglais). Cela ne signifie pas en revanche que l'erreur commise soit toujours aisément identifiable.

Les explications de METAFONT sont explicites et souvent pleines d'humour (« *Proceed, with fingers crossed* ») : contrairement à ce que l'on rencontre habituellement dans les programmes informatiques, on appréciera que METAFONT s'exprime de façon profondément humaine (sic!), exprimant tour à tour la gaieté (« *Oh dear. I can't decide if the expression above is positive* ») ou la tristesse la plus tragique (« *One of your faux pas seems to have wounded me deeply...in fact, I'm barely conscious.* »). Nous donnons dans cette annexe la liste des principaux messages que l'on peut rencontrer ainsi que les conclusions qu'on peut en tirer.

Les erreurs les plus fréquemment commises sont les suivantes :

1. oubli du point-virgule à la fin d'une instruction. Il en résulte que METAFONT continue à lire l'instruction suivante en croyant qu'elle fait partie de la précédente et réagit en signalant, par exemple, que l'argument de la première est inapproprié ;
2. oubli de coordonnées dans la déclaration des points de construction : on définit l'abscisse d'un point et on oublie son ordonnée (ou le contraire) ;
3. oubli de déclaration d'une variable. Une variable nouvelle dont le type n'a pas été déclaré est considérée par défaut comme variable de type numérique, ce qui

1. À moins qu'on ne l'ait mis en *scrollmode* ou bien en *nonstopmode*.

- peut conduire à des incohérences là où METAFONT attend un type particulier ;
4. définitions redondantes : on donne une valeur à une variable particulière puis par la suite on lui en donne une autre. Il est important de bien faire la différence entre l'égalisation (signe  $=$ ) et l'assignation (signe  $:=$ ) ;
  5. utilisation d'un même nom pour deux variables différentes.

On a d'autre part inventorié à l'annexe C p. 183 les commandes qui permettent de suivre à la trace l'exécution d'un programme afin de le déboguer. Il s'agit de *tracingtitles*, *tracingequations*, *tracingcapsules*, *tracingchoices*, *tracingspecs*, *tracingpens*, *tracingcommands*, *tracingrestores*, *tracingmacros*, *tracingedges*, *tracingoutput*, *tracingstats*, *tracingonline*, etc.

Enfin, le plus souvent, METAFONT ne se contente pas de diagnostiquer un problème, il propose aussi un remède. On peut accepter ce qu'il propose en frappant la touche **Return** : l'exécution peut reprendre en général sans trop de dégâts mais cela ne dispense aucunement l'utilisateur d'apporter par la suite les corrections nécessaires à son fichier. Les valeurs de remplacement que METAFONT substitue aux valeurs qu'il considère comme défectueuses (comme on le verra dans de très nombreux messages ci-dessous) permettent de produire un résultat quoi qu'il arrive lorsque le travail se fait dans un mode tel que *scrollmode* ou *nonstopmode*.

## D.1 Messages d'aide

Nous allons voir dans cette section que METAFONT propose à l'utilisateur plusieurs façons de rattraper une erreur : il est très rare qu'il n'ait pas de suggestions ou de conseils avisés à donner à la suite d'une erreur.

### D.1.1 Comment réagir à une erreur

- Type « *return* » to proceed, *S* to scroll future error messages, *R* to run without stopping, *Q* to run quietly, *I* to insert something, *E* to edit your file, or *1...* or *9* to ignore the next 1 to 9 tokens of input, *H* for help, *X* to quit.

Il a été expliqué au paragraphe 3.9 p. 60 comment on peut réagir à un arrêt intempestif du programme. Le message ci-dessus rappelle les différentes options.

- That makes 100 errors ; please try again.

METAFONT, en *scrollmode* ou bien en *nonstopmode*, a recensé 100 erreurs (probablement dans une boucle ou une erreur à répétition) et c'est pour lui la limite.

- I have just deleted some text, as you asked. You can now delete more, or insert, or whatever.

En réponse à l'option *I*, on peut insérer directement des instructions supplémentaires pour que METAFONT continue son exécution.

- *Sorry, I don't know how to help in this situation. Maybe you should try asking a human?*

METAFONT diagnostique les erreurs en deux temps : le second diagnostic est obtenu en tapant H après un arrêt. Le second message est en général plus détaillé sauf dans le cas présent où METAFONT baisse les bras en recommandant de s'adresser à un être humain ! La machine capitule.

- *Sorry, I already gave what help I could... An error might have occurred before I noticed any problems.*

Humilité de la part de METAFONT : il n'a pas vu venir l'erreur et avoue qu'il ne sait quel conseil donner.

- *Emergency stop. METAFONT capacity exceeded, sorry. [ If you really absolutely need more capacity, you can ask a wizard to enlarge me]*

Il est rare que la mémoire de METAFONT soit réellement épuisée par l'ampleur de la tâche : c'est le plus souvent une erreur de programmation qui l'aura lancé dans une boucle infinie.

- *I'm broken. Please show this to someone who can fix. One of your faux pas seems to have wounded me deeply...in fact, I'm barely conscious. Please fix it and try again.*

Le plus tragique des messages émis par METAFONT : blessé à mort, il implore l'utilisateur de réparer lui-même ses erreurs ! Cette situation tragi-comique renseigne malheureusement assez peu sur la nature de l'erreur. On aura intérêt à aller lire le fichier log.

- *Missing xxx has been inserted*

Un caractère ou un délimiteur (noté ici xxx) attendu par la syntaxe a été omis et METAFONT a pris de lui-même l'initiative de l'insérer là où il pense que c'est juste.

- *Try to insert some instructions for me (e.g., « I show x »), unless you just want to quit by typing X.*

METAFONT suggère qu'une ou plusieurs instructions soient insérées par l'utilisateur. Ce genre d'arrêts n'est pas nécessairement dû à une erreur de programmation. « I show x » signifie « insérer la commande *show x* ».

## D.1.2 Erreurs de calcul

Dans cette section sont regroupées les erreurs résultant d'un non respect des limitations internes de METAFONT. Les messages d'erreur ci-dessous peuvent être supprimés en donnant la valeur 0 à la variable *warningcheck*.

- *Arithmetic overflow. Uh, oh. A little while ago one of the quantities that I was computing got too large, so I'm afraid your answers will be somewhat askew.*

*You'll probably have to adopt different tactics next time. But I shall try to carry on anyway.*

Les calculs demandés à METAFONT l'ont conduit à traiter des nombres trop grands pour lui. Rappelons qu'il n'accepte pas de nombres supérieurs ou égaux à 4096. Une des solutions possibles consiste souvent à changer l'unité de mesure pour une valeur plus petite.

- *Some number got too big.*

On ne saurait être plus clair ! Voir ci-dessus.

- *Square root of has been replaced by 0. Since I don't take square roots of negative numbers, I'm zeroing this one. Proceed, with fingers crossed.*

METAFONT remplace souvent, de sa propre initiative, les valeurs illicites (racine carrée d'un nombre négatif, logarithme d'un nombre négatif ou nul, etc.) par 0 ou par 1 et poursuit les calculs ainsi. Du coup, le résultat n'est pas toujours ce que l'on souhaite, mais on aura pu traiter le fichier jusqu'au bout.

- *The path that I just computed is out of range. So it will probably look funny. Proceed, for a laugh.*

De même que pour les nombres, un chemin peut devenir trop gros. Il faut revoir la position des points de contrôle ou bien l'unité de mesure choisie.

- *At least one of the coordinates in the path I'm about to digitize was really huge (potentially bigger than 4095). So I've cut it back to the maximum size. The results will probably be pretty wild.*

Autre réaction possible de METAFONT face à un nombre trop grand : il le tronque à la valeur maximale autorisée (à savoir 4095). Le résultat peut être très surprenant et parfaitement imprévisible.

- *The cycle you specified has a coordinate of 4095.5 or more. So I've replaced it by the trivial path '(0, 0)..cycle'.*

C'est parfois un chemin cyclique qui conduit à des valeurs excessives. Il est alors remplacé par le cycle nul.

- *Value is too large. The equation I just processed has given some variable a value of 4096 or more. Continue and I'll try to cope with that big value ; but it might be dangerous.*

C'est à la suite d'un calcul interne cette fois que METAFONT a été amené à traiter un nombre trop grand. En général, en frappant la touche Return, METAFONT réajuste la valeur et l'exécution du programme peut reprendre sans incident.

- *Enormous number has been reduced I can't handle numbers bigger than about 4095.99998 ; so I've changed your constant to that maximum amount.*

Encore un nombre tronqué ! Voir plus haut.

### D.1.3 Erreurs de langage

Dans cette section sont réunis les messages d'erreur dus à une erreur de langage : en général, non respect de la syntaxe requise ou, le plus souvent, oubli d'un délimiteur ou simple erreur de frappe.

- *Incomplete if; all text was ignored after line xx. A forbidden outer token occurred in skipped text. This kind of error happens when you say « if... » and forget the matching « fi ». I've inserted a 'fi'; this might work.*

Une boucle conditionnelle comporte une entrée (if) et un sortie (fi). Si le délimiteur de sortie est omis, METAFONT continue à lire le texte et rencontre un mot qu'il ne sait pas interpréter. De lui-même il insère un *fi* pour poursuivre l'exécution. L'erreur n'a pas été fatale mais il faudra vérifier et corriger le programme.

- *The file ended while I was skipping conditional text.*

METAFONT a rencontré une instruction *end* avant la fin d'une boucle conditionnelle : c'est à nouveau un *fi* de fin de condition qui a été oublié.

- *Forbidden token found while scanning. I suspect you have forgotten an « enddef », causing me to read past where you wanted me to stop.*

Cette fois il s'agit du même type d'erreur que précédemment mais dans un environnement de définition. Celui-ci a une entrée (*def* ou *vardef*) et une sortie (*enddef*).

- *I suspect you have forgotten an « endfor »*

Même chose que précédemment mais dans une boucle répétitive : elle commence par *for* et doit se terminer par *endfor*.

- *A group begun on line xx never ended. I saw a « begingroup » back there that hasn't been matched by « endgroup ». So I've inserted « endgroup » now.*

Même chose que précédemment mais dans un environnement de groupe. Un groupe commencé n'a pas été clos. METAFONT se propose de lui-même d'insérer le *endgroup* manquant mais il faut néanmoins vérifier l'endroit auquel il a l'intention de le faire car ce n'est pas toujours celui qu'on souhaite.

- *I'll try to recover ; but if the error is serious, you'd better type E or X now and fix your file. A previous error seems to have propagated*

Dans ce cas, METAFONT a détecté qu'une erreur a été commise quelque part et s'est propagée mais il n'en a pas trouvé l'origine. L'utilisateur devra lui-même analyser son programme.

- *It seems that a right delimiter was left out*

Message explicite : un délimiteur a été oublié (parenthèse pas refermée par exemple).

- *I've scanned an expression of the form « a/b, c », so a right bracket should have come next.*

Encore un problème de délimiteur manquant. Voir plus haut.

- *Text line contains an invalid character*

Erreur de frappe ou bien syntaxe illicite. Il faut relire soigneusement la ligne incriminée.

- *A funny symbol that I can't read has just been input. Continue, and I'll forget that it ever happened.*

Un symbole inattendu, dû en général à une faute de frappe, a été lu par METAFONT sans provoquer d'erreur de syntaxe. METAFONT choisit ici de l'ignorer. En frappant la touche **Return**, il reprendra l'exécution.

- *Incomplete string token has been flushed. Strings should finish on the same line as they began. I've deleted the partial string; you might want to insert another by typing, e.g., « I new string ».*

Une chaîne doit être écrite sur une seule ligne. METAFONT propose de rattraper l'erreur en tapant à nouveau la chaîne souhaitée. « *I new string* » signifie « Insérez une nouvelle chaîne ».

- *Job aborted, no legal end found*

La fin du fichier a été atteinte et l'instruction *end* a été omise. METAFONT dans ce cas là s'arrête et seul le fichier *log* est produit.

- *Missing parameter type ; « expr » will be assumed. You should've had « expr » or « suffix » or « text » here.*

Il existe trois types de paramètres lorsqu'on définit une macro : *expr*, *suffix* et *text*. On a vu des exemples de chacun d'eux au chapitre 4. Si l'on a oublié de préciser le type en question, METAFONT considère par défaut qu'il a affaire à des paramètres de type *expr*.

- *Extra « endfor ». I'm not currently working on a for loop, so I had better not try to end anything.*

Il arrive, à force de faire des modifications, que l'on place une fin de groupe (*endfor*, *enddef*, *endgroup*, etc.) à deux endroits différents. METAFONT se propose d'ignorer l'instruction superflue.

- *No loop is in progress. Why say « exitif » when there's nothing to exit from ?*

L'instruction *exitif* sert à quitter une boucle. Elle doit nécessairement être placée entre les instructions d'entrée et de sortie de la boucle.

- *After « exitif <boolean exp> » I expect to see a semicolon. I shall pretend that one was there.*

Un point-virgule a été omis pour délimiter l'instruction conditionnelle de sortie d'une boucle.

- *Too many arguments to ...*

Une macro est définie avec un certain nombre d'arguments : lorsque cette macro est appelée, elle doit évidemment comporter le nombre d'arguments qui a été déclaré. Lorsque ce message apparaît et que l'on a pourtant donné le bon nombre d'arguments, c'est le plus souvent qu'un point-virgule a été oublié : METAFONT est allé lire ce qui suit et l'interprète comme un argument supplémentaire.

- *Missing argument to ...*

*That macro has more parameters than you thought. I'll continue by pretending that each missing argument is either zero or null.*

À l'inverse de ce qui précède, METAFONT diagnostique parfois une insuffisance d'arguments. Ce sont des choses qui arrivent ! Mais si l'on est sûr qu'il y a effectivement le bon nombre d'arguments, l'erreur provient alors en général d'un oubli de délimiteurs (les virgules qui séparent les arguments) ou de délimiteurs mal placés.

- *I've finished reading a macro argument and am about to read another; the arguments weren't delimited correctly. I've gotten to the end of the macro parameter list.*

Même problème que précédemment.

- *When you say « for x=a step b until c », the initial value « a » and the step size « b » and the final value « c » must have known numeric values.*

Les boucles répétitives peuvent faire appel à des paramètres ou des variables (tels que *a*, *b* ou *c*) mais ceux-ci doivent avoir des valeurs numériques connues antérieurement. Il se peut que ces valeurs n'aient pas été déclarées ou qu'il y ait un problème de type de variable.

- *I assume you meant to say « until » after « step ». So I'll look for the final value and colon next.*

Voici un autre exemple dans lequel METAFONT prend lui-même l'initiative de corriger une syntaxe défectueuse.

- *I'm afraid I need some sort of value in order to continue, so I've tentatively inserted 0. You may want to delete this zero and insert something else; see Chapter 27 of The METAFONTbook for an example.*

Ce message est typique du comportement de METAFONT face à une valeur qu'il juge incorrecte ou manquante. Il se propose de remplacer par zéro : cela se produit systématiquement lorsqu'une coordonnée d'un point n'a pas été définie. L'allusion au chapitre 27 du The METAFONTbook (voir la bibliographie à la page 1) concerne l'usage de la commande « *!???* » expliquée au chapitre 3 p. 23.

- *Division by zero. I'll pretend that you meant to divide by 1.*

La division par zéro provient en général d'une variable à qui on a oublié d'assigner une valeur numérique.

- *Variable has been obliterated. It seems you did a nasty thing — probably by accident, but nevertheless you nearly hornswoggled me...*

Une variable aura été accidentellement endommagée : cela peut se produire lorsque METAFONT analyse et développe une macro définie par l'utilisateur. Lorsqu'il rencontre la macro, il se reporte à la définition qu'il avait stockée quelque part et la lit : c'est cette définition qui peut malencontreusement modifier une variable qui a été définie précédemment (double emploi d'une même lettre par exemple).

## D.2 Erreurs de syntaxe

Cette section réunit les messages d'erreur concernant les définitions de points, les chemins, les images ou les transformations.

### D.2.1 Erreurs sur les variables logiques

- *Undefined condition will be treated as « false ». The expression shown above should have had a definite true-or-false value. I'm changing it to « false ».*

Une variable de type booléen à laquelle on n'a pas attribué de valeur (*true* ou *false*) reçoit par défaut la valeur *false*.

- *The quantities shown above have not been equated. Oh dear. I can't decide if the expression above is positive, negative, or zero. So this comparison test won't be « true ».*

L'instruction *solve* utilise systématiquement un test de comparaison. Dans le cas présent, METAFONT se trouve face à une condition à tester trop complexe pour lui. Par défaut, il renvoie la valeur *false* comme résultat du test.

- *Unknown relation will be considered false*

Si une relation inconnue est utilisée dans une instruction conditionnelle, METAFONT lui attribue encore une fois par défaut la valeur *false*.

- *Extra tokens will be flushed. I've just read as much of that statement as I could fathom, so a semicolon should have been next. It's very puzzling... but I'll try to get myself back together, by ignoring.*

Erreur classique : un point-virgule a été oublié. METAFONT est prêt à fermer les yeux sur tout ce qu'il juge superflu.

- *Isolated expression. I couldn't find an < ==> or \*: ==> after the expression that is shown above this error message, so I guess I'll just ignore it and carry on.*

Erreur classique : on commence à taper une ligne d'instructions et on va boire un café. De retour à la console, on passe à autre chose et METAFONT se retrouve avec une expression isolée qu'il prend pour une nouvelle variable et dont il cherche en vain la valeur.

### D.2.2 Erreurs sur les équations

- *Internal quantity xxx must receive a known value I can't set an internal quantity to anything but a known numeric value, so I'll have to ignore this assignment.*

Une quantité interne ne peut avoir une valeur définie indirectement ou implicitement. METAFONT attend une valeur numérique explicite.

- *Equation cannot be performed. I'm sorry, but I don't know how to make such things equal.*

Le plus souvent ce message résulte d'une erreur de type de variable : hormis la variable *z* qui est considérée *a priori* comme une paire, toute variable non déclarée sera traitée, par défaut, comme une variable numérique. Ce choix conduit à des conflits lorsque l'on croit égaler deux variables de même type alors que l'une d'elle a été ainsi modifiée.

- *Redundant or inconsistent equation. An equation between already-known quantities can't help. But don't worry; continue and I'll just ignore it.*

Ce message résulte souvent de la réutilisation d'une même variable avec une valeur différente. Supposons que l'on ait une variable numérique nommée *theta* qui désigne un angle. Il se peut qu'on ait besoin d'utiliser cet angle avec une valeur de 30 degrés puis par la suite qu'on souhaite lui donner la valeur 45. Il faut, pour éviter toute erreur, utiliser le symbole «`:`=» et non pas le signe = sinon METAFONT pense que *theta* vaut simultanément 30 et 45 : en rencontrant la deuxième valeur il signalera une erreur de 15 (la différence). Le signe = est traité comme une équation.

- *Inconsistent equation. The equation I just read contradicts what was said before. (off by xxx).*

Même explication que précédemment.

- *Nonnumeric ypart has been replaced by 0. I thought you were giving me a pair «(x, y)»; but after finding a nice xpart «x» I found a ypart «y» that isn't of numeric type. So I've changed y to zero. (The y that I didn't like appears above the error message.)*

Erreur de type de variable : une ordonnée *y* n'a pas le type numérique attendu. En fait il n'est pas obligatoire que *x* ou *y* soient de type numérique ou ne servent qu'à désigner des abscisses ou des ordonnées ; pourtant, pour éviter tout risque de confusion, il est prudent de réservier ces deux lettres à cet usage. Rappelons dans le même ordre d'idée, que la variable *z* est *a priori* considérée par METAFONT comme une variable de type *paire* (il est donc inutile de la déclarer comme paire : c'est la seule exception à la règle qui veut qu'une variable dont le type n'a pas été déclaré soit considérée par défaut comme une variable numérique).

- *Improper subscript has been replaced by zero. A bracketed subscript must have a known numeric value; unfortunately, what I found was the value that appears just above this error message. So I'll try a zero subscript.*

Un indice de variable peut être écrit entre crochets [ ] et représenté sous forme algébrique et non pas sous une forme numérique explicite. En revanche, si la valeur algébrique ne peut être calculée au moment où METAFONT en a besoin, il substitue par défaut la valeur 0 (il y a très peu de chance que ce soit la valeur idoine mais METAFONT a besoin d'une valeur, quelle qu'elle soit, pour continuer).

- *While I was evaluating the suffix of this variable, something was redefined, and it's no longer a variable! In order to get back on my feet, I've inserted « 0 » instead.*

Situation analogue à la précédente : substitution de la valeur 0 à une valeur inconnue.

- *Improper « : = » will be changed to « = ». I didn't find a variable name at the left of the « : = », so I'm going to pretend that you said « = » instead.*

Le symbole d'assignation ne peut être utilisé que pour attribuer une valeur à une variable déclarée. On ne doit pas l'utiliser dans des définitions de macros ou de variables.

- *Declared variable conflicts with previous vardef. You can't use, e.g., « numeric foo[ ] » after « vardef foo ». Proceed, and I'll ignore the illegal redeclaration.*

Si un nom est utilisé pour définir une macro (*foo* dans le cas présent), il ne peut servir ensuite pour désigner une variable, même indexée.

- *Illegal suffix of declared variable will be flushed. Variables in declarations must consist entirely of names and collective subscripts, e.g., « x[ ]a ».*

*x[ ]a* est la forme la plus générale de variable indexée, l'expression entre crochets pouvant être une expression algébrique aussi bien qu'une valeur numérique explicite. C'est la seule syntaxe admise en dehors de la forme explicite *x1*, *x2*, etc.

- *Are you trying to use a reserved word in a variable name? I'm going to discard the junk I found here, up to the next comma or the end of the declaration.*

Les noms des macros du format *plain* décrit dans cet ouvrage sont évidemment des noms réservés et ne peuvent être réutilisés sans risquer de perturber profondément METAFONT.

- *Explicit subscripts like « x15a » aren't permitted.*

Il manque ici les crochets qui permettront à METAFONT de reconnaître un suffixe de variable. En revanche *x15* aurait été accepté.

### D.2.3 Erreurs sur les chemins

Les trois premiers messages indiquent le comportement de METAFONT lorsqu'il rencontre une coordonnée non définie : il la remplace par zéro. Ce n'est évidemment pas

ce que l'on souhaite en général mais c'est un choix par défaut qui permet de poursuivre l'exécution du programme.

- 1. *Undefined coordinates have been replaced by (0, 0). I need x and y numbers for this part of the path. The value I found (see above) was no good; so I'll try to keep going by using zero instead. (Chapter 27 of The METAFONT-book explains that you might want to type « I??? » now.)*
  - 2. *Undefined x coordinate has been replaced by 0. I need a « known » x value for this part of the path.*
  - 3. *Undefined y coordinate has been replaced by 0. I need a « known » y value for this part of the path.*
- *Pen path must be a cycle. I can't make a pen from the given path.*

Un stylo doit être défini à partir d'un chemin cyclique. Rappelons qu'il ne suffit pas qu'un chemin revienne à son point de départ pour faire un cycle. Il doit se terminer par la mention « ... cycle ».

- *I've scanned a direction spec for part of a path, so a right brace should have come next.*

On peut préciser les directions des demi-tangentes en un point de construction : ces directions sont indiquées entre accolades. Ce message indique un oubli d'accolade fermante.

- *Improper curl has been replaced by 1. A curl must be a known, nonnegative number.*

La variable *curl* indique un degré de courbure. Ce doit être un nombre positif ou nul défini explicitement.

- *Paths don't touch ; « & » will be changed to « .. ». When you join paths « p & q », the ending point of p must be exactly equal to the starting point of q. So I'm going to pretend that you said « p..q » instead.*

Pour concaténer deux chemins, il faut évidemment qu'ils soient contigus : si ce n'est pas le cas, METAFONT les joint l'un à l'autre en prolongeant la courbe de l'extrémité du premier chemin à l'origine du second.

- *Not a cycle. That contour should have ended with « .. cycle » or « & cycle ».*  
Là où METAFONT attend un cycle et ne le trouve pas, il fait lui-même la modification nécessaire.

- *Strange path (turning number is zero).*

Message qui traduit l'embarras de METAFONT devant un chemin dont le *turningnumber* est nul (reportez-vous au paragraphe 3.1.2). Cela signifie qu'il ne sait plus distinguer l'intérieur et l'extérieur d'un chemin cyclique (comme par exemple dans le chiffre 8). Avec la commande *draw* cela ne l'empêche pas de fournir le bon résultat, en revanche il ne pourra appliquer la commande *filldraw*.

- Backwards path (turning number is negative). The path doesn't have a counter-clockwise orientation, so I'll probably have trouble drawing it. (See Chapter 27 of *The METAFONTbook* for more help.)

Situation analogue à la précédente : METAFONT confondra l'intérieur avec l'extérieur si le *turningnumber* est négatif.

#### D.2.4 Erreurs sur les transformations

- Not implemented: I'm afraid I don't know how to apply that operation to that particular type. Continue, and I'll simply return the argument (shown above) as the result of the operation.

Toutes les transformations ne s'appliquent pas à tous les types de variables : toutes s'appliquent aux paires évidemment, mais pas nécessairement aux chemins ou aux images.

- Improper transformation argument. The expression shown above has the wrong type, so I can't transform anything using it.

Erreur de syntaxe dans l'argument d'une transformation.

- Transform components aren't all known. I'm unable to apply a partially specified transformation except to a fully known pair or transform.

Dans ce cas, METAFONT se plaint de ne pas connaître la valeur explicite de toutes les six coordonnées d'une transformation. Dans le cas d'une paire, il peut néanmoins s'en accommoder, ce qui signifie qu'il peut traiter le point transformé de manière algébrique en attendant d'avoir des valeurs explicites.

- That transformation is too hard. I can apply complicated transformations to paths, but I can only do integer operations on pictures.

METAFONT n'est pas un logiciel de dessin mais de création de polices de caractères : toutes les opérations sur les images ne sont pas possibles.

- Too far to shift I can't shift the picture as requested

La translation demandée déplacerait les points trop loin.

- Scaled picture would be too big. I can't yscale the picture as requested — it would make some coordinates too large or too small. Proceed, and I'll omit the transformation.

La transformation *scaled* a conduit à des valeurs excessives. Dans ce cas là, la transformation est purement et simplement omise.

#### D.2.5 Erreurs sur les images

- Weight must be -3, -2, -1, +1, +2, or +3

Dans la commande *addto* il est possible de spécifier un poids (autrement dit un nombre de pixels). Celui-ci doit impérativement avoir l'une des valeurs indiquées.

- *I was looking for a « known » picture variable. So I'll not change anything just now.*

Une variable de type *picture* n'a pas été explicitée. METAFONT ignorera l'instruction concernée.

- *Not a suitable variable. At this point I needed to see the name of a picture variable. (Or perhaps you have indeed presented me with one; I might have missed it, if it wasn't followed by the proper token.)*

Erreur de type de variable. METAFONT, en même temps, sait être modeste : peut-être a-t-il lui-même manqué quelque chose !

- *Bad culling amounts. Always cull by known amounts that exclude 0.*

L'instruction *cull* attend deux valeurs entières explicites et non nulles.

## D.2.6 Erreurs sur les polices

- *Font metric dimensions must be less than 2048pt.*

Le fichier *.tfm* (TeX Font Metrics) n'admet pas de dimensions supérieures à 2048 pt.

- *Improper location. I was looking for a known, positive number. For safety's sake I'll ignore the present command.*

Le code d'un caractère doit être un nombre entier positif compris entre 0 et 255.

- *Illegal ligtable step. I was looking for « =:» or « kern » here.*

Erreur de syntaxe dans les tables de césures.

- *Improper kern. The amount of kern should be a known numeric value.*

Erreur de syntaxe concernant les approches de paires.

- *Improper font parameter (some values had to be adjusted by as much as xxx pt)*

Une limitation de METAFONT concerne le nombre de hauteurs différentes de caractères dans une même police qui est limité à 15. Au-delà, METAFONT procède à des ajustements. Ceci est expliqué au paragraphe 6.3 p. 175.

- *Improper font parameter (illegal design size has been changed to 128pt)*

Une limitation de METAFONT : la taille d'une police ne peut excéder 128 pt. Si une valeur illicite est déclarée ou si aucune valeur n'a été déclarée, METAFONT prend par défaut 128 pt.

Pour conclure cette annexe, voici un message qui est davantage un conseil qu'un message d'erreur :

- *See the transcript file for additional information*

En effet, le fichier `zozo.log` associé à un fichier `zozo.mf` contient beaucoup plus d'informations que ce qu'on peut lire à la console. En cas de problème, la lecture de ce fichier (que l'on peut ouvrir avec n'importe quel éditeur de texte) est hautement recommandée.

Le message d'erreur suivant indique que la commande `ldd` a été exécutée avec un argument incorrect :

`ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found`

Le message d'erreur suivant indique que la commande `ldd` a été exécutée avec un argument incorrect :

`ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found`

Le message d'erreur suivant indique que la commande `ldd` a été exécutée avec un argument incorrect :

`ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found`

Le message d'erreur suivant indique que la commande `ldd` a été exécutée avec un argument incorrect :

`ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found`

Le message d'erreur suivant indique que la commande `ldd` a été exécutée avec un argument incorrect :

`ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found`

Le message d'erreur suivant indique que la commande `ldd` a été exécutée avec un argument incorrect :

`ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found`

Le message d'erreur suivant indique que la commande `ldd` a été exécutée avec un argument incorrect :

`ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found`

Le message d'erreur suivant indique que la commande `ldd` a été exécutée avec un argument incorrect :

`ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found`

Le message d'erreur suivant indique que la commande `ldd` a été exécutée avec un argument incorrect :

`ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found  
ldd: /bin/ls: file not found`

Il est à noter que les deux encodages OT1 et T1 sont en étendue très similaire mais que certains caractères sont codés différemment. Ainsi par exemple le caractère `é` (code 231) est codé en `é` dans l'OT1 mais en `é` dans le T1. De plus, certains caractères sont absents dans l'un des deux encodages. C'est le cas par exemple du caractère `à` qui n'existe pas dans l'OT1 mais qui existe dans le T1.

## Annexe E

# Tables de caractères

Dans cette annexe, on trouvera quelques tables de caractères utiles. Tout d'abord la table des caractères de la police cmr10 dans l'encodage OT1 (table E.1) puis la table des caractères de la police ecr10 dans l'encodage T1 (table E.2). Il est important lorsqu'on crée une police de caractères de déterminer quel sera l'encodage c'est-à-dire l'emplacement des caractères. Les polices de symboles fantaisistes n'ont pas d'encodage fixé (leur encodage s'appelle U et ne correspond à rien de normalisé). En revanche les polices de caractères peuvent être en OT1 (128 caractères comme dans l'exemple de la famille Computer Roman) ou en T1 (256 caractères comportant en particulier les caractères accentués ; c'est la norme dite « de Cork ») : mais il existe bien d'autres encodages (OT2 pour les polices cyrilliques de la famille wncyr ou wlcyr : voir le tableau E.3) et chacun peut définir le sien propre mais, sauf nécessité absolue, il est recommandé de se conformer à ceux en vigueur.

Nous donnons ensuite la table des vingt-six lettrines de la police yinit qui sont un remarquable exemple, dû à Yannis Haralambous, du degré de qualité des résultats que l'on peut obtenir avec METAFONT (table E.4). Du même auteur, on verra les tableaux des polices gothiques ygoth et yswab (tables E.6 et E.7). Une police de lettres calligraphiées intitulée calligra est présentée dans le tableau E.8 : elle est codée sur 256 caractères. Le tableau E.5 montre une police complète de caractères grecs avec tous les esprits et accents nécessaires pour écrire le grec ancien : il est vrai que les polices de la famille cmr comportent tous les caractères grecs mais ceux-ci sont destinés à l'usage mathématique et ne conviennent pas bien à la rédaction de textes littéraires. Une police de symboles astrologiques intitulée astrosym présente une série de 29 symboles déclinée, au sein de la même police, en trois versions différentes : une version calligraphiée (codes 0 à 28), une version droite plus épaisse (codes 100 à 128) et une version droite plus fine (codes 200 à 228). Les dix premiers caractères concernent les planètes du système solaire, les 12 suivants sont les signes du zodiaque et les derniers sont des variantes (voir table E.10). Pour terminer, la table E.12 présente les cent vingt-trois caractères de la police gray dont le rôle a été expliqué au paragraphe 2.2.6 : le caractère de code zéro est le point noir `*` qui marque l'emplacement des points sur les

épreuves en mode *proof* et le caractère de code 1 est le symbole «». Les suivants sont des composés de ce «» comme par exemple | ou ||.

Dans un genre plus fantaisiste, voici quelques symboles dus à D. Roegel qui les a repris des polices *wasy* et *gen* de D. Knuth lui-même. Ce sont les sept caractères de la police *drgen* destinés aux passionnés de généalogie :

♀ ♂ + ♂ ♂ ♂ ♂

GEN

La police *cirth* de J. Grant offre soixante-quatre caractères runiques :

◊ Å 1 Þ L A Æ Y Æ Ÿ ß K Æ Y Æ Z K \* X Æ M N Æ  
† M A A Æ k þ A ÿ 1 H Æ Æ M C X H R i F H Æ P A  
Í k Þ Æ B A P K > T Æ A Æ Y H X

Pour les archéologues voici les vingt-neuf caractères, dus à A. Stanier, de la police *SouthArabian*. Ils proviennent de plusieurs langues parlées au sud de l'Arabie au second millénaire av. J.-C. :

· ፩ ፻ ፻ ፻ ፻ ፻ ፻ ፻ ፻ ፻ ፻ ፻ ፻ ፻ ፻ ፻ ፻ ፻ ፻ ፻ ፻ ፻ ፻ ፻ ፻

X + i X

On trouve bien sûr encore beaucoup d'autres exemples sur les sites accessibles par l'Internet et, grâce à METAFONT, la porte est désormais grande ouverte à l'imagination de chacun...

|    |     |    |    |    |   |     |   |
|----|-----|----|----|----|---|-----|---|
| 0  | Γ   | 32 | -  | 64 | @ | 96  | ' |
| 1  | Δ   | 33 | !  | 65 | A | 97  | a |
| 2  | Θ   | 34 | "  | 66 | B | 98  | b |
| 3  | Λ   | 35 | #  | 67 | C | 99  | c |
| 4  | Ξ   | 36 | \$ | 68 | D | 100 | d |
| 5  | Π   | 37 | %  | 69 | E | 101 | e |
| 6  | Σ   | 38 | &  | 70 | F | 102 | f |
| 7  | Τ   | 39 | ,  | 71 | G | 103 | g |
| 8  | Φ   | 40 | (  | 72 | H | 104 | h |
| 9  | Ψ   | 41 | )  | 73 | I | 105 | i |
| 10 | Ω   | 42 | *  | 74 | J | 106 | j |
| 11 | ff  | 43 | +  | 75 | K | 107 | k |
| 12 | fi  | 44 | ,  | 76 | L | 108 | l |
| 13 | fl  | 45 | -  | 77 | M | 109 | m |
| 14 | ffi | 46 | .  | 78 | N | 110 | n |
| 15 | ffl | 47 | /  | 79 | O | 111 | o |
| 16 | ι   | 48 | 0  | 80 | P | 112 | p |
| 17 | ј   | 49 | 1  | 81 | Q | 113 | q |
| 18 | ΄   | 50 | 2  | 82 | R | 114 | r |
| 19 | ΅   | 51 | 3  | 83 | S | 115 | s |
| 20 | ΅   | 52 | 4  | 84 | T | 116 | t |
| 21 | ΅   | 53 | 5  | 85 | U | 117 | u |
| 22 | -   | 54 | 6  | 86 | V | 118 | v |
| 23 | °   | 55 | 7  | 87 | W | 119 | w |
| 24 | ,   | 56 | 8  | 88 | X | 120 | x |
| 25 | ß   | 57 | 9  | 89 | Y | 121 | y |
| 26 | æ   | 58 | :  | 90 | Z | 122 | z |
| 27 | œ   | 59 | ;  | 91 | [ | 123 | - |
| 28 | ø   | 60 | l  | 92 | " | 124 | - |
| 29 | Æ   | 61 | =  | 93 | ] | 125 | " |
| 30 | Œ   | 62 | ѣ  | 94 | ^ | 126 | ~ |

|    |     |    |    |    |   |     |   |     |    |     |    |     |    |     |   |
|----|-----|----|----|----|---|-----|---|-----|----|-----|----|-----|----|-----|---|
| 0  | `   | 32 | ¸  | 64 | @ | 96  | ' | 128 | Ă  | 160 | ă  | 192 | À  | 224 | à |
| 1  | '   | 33 | !  | 65 | A | 97  | a | 129 | Ą  | 161 | ą  | 193 | Á  | 225 | á |
| 2  | ^   | 34 | "  | 66 | B | 98  | b | 130 | Ć  | 162 | ć  | 194 | Â  | 226 | â |
| 3  | ~   | 35 | #  | 67 | C | 99  | c | 131 | Č  | 163 | č  | 195 | Ā  | 227 | ã |
| 4  | ..  | 36 | \$ | 68 | D | 100 | d | 132 | Ď  | 164 | ď  | 196 | Ä  | 228 | ä |
| 5  | ~   | 37 | %  | 69 | E | 101 | e | 133 | Ě  | 165 | ě  | 197 | Å  | 229 | å |
| 6  | °   | 38 | &  | 70 | F | 102 | f | 134 | Ę  | 166 | ę  | 198 | Æ  | 230 | æ |
| 7  | ˇ   | 39 | '  | 71 | G | 103 | g | 135 | Ğ  | 167 | ğ  | 199 | Ҫ  | 231 | ç |
| 8  | ˇ   | 40 | (  | 72 | H | 104 | h | 136 | Ĺ  | 168 | Í  | 200 | È  | 232 | è |
| 9  | -   | 41 | )  | 73 | I | 105 | i | 137 | Ľ  | 169 | ĺ  | 201 | É  | 233 | é |
| 10 | .   | 42 | *  | 74 | J | 106 | j | 138 | Ľ  | 170 | ł  | 202 | Ê  | 234 | ê |
| 11 | ,   | 43 | +  | 75 | K | 107 | k | 139 | Ń  | 171 | ń  | 203 | Ë  | 235 | ë |
| 12 | ,   | 44 | ,  | 76 | L | 108 | l | 140 | Ń  | 172 | ň  | 204 | Ì  | 236 | ì |
| 13 | ,   | 45 | -  | 77 | M | 109 | m | 141 | Ĳ  | 173 | ŋ  | 205 | Í  | 237 | í |
| 14 | <   | 46 | .  | 78 | N | 110 | n | 142 | Ő  | 174 | ő  | 206 | Î  | 238 | î |
| 15 | >   | 47 | /  | 79 | O | 111 | o | 143 | Ŕ  | 175 | ŕ  | 207 | Ï  | 239 | ï |
| 16 | "   | 48 | 0  | 80 | P | 112 | p | 144 | Ř  | 176 | ř  | 208 | Đ  | 240 | đ |
| 17 | "   | 49 | 1  | 81 | Q | 113 | q | 145 | Ś  | 177 | ś  | 209 | Ñ  | 241 | ñ |
| 18 | "   | 50 | 2  | 82 | R | 114 | r | 146 | Š  | 178 | š  | 210 | Ò  | 242 | ò |
| 19 | «   | 51 | 3  | 83 | S | 115 | s | 147 | Ş  | 179 | ş  | 211 | Ó  | 243 | ó |
| 20 | »   | 52 | 4  | 84 | T | 116 | t | 148 | Ť  | 180 | ť  | 212 | Ô  | 244 | ô |
| 21 | -   | 53 | 5  | 85 | U | 117 | u | 149 | Ŧ  | 181 | ŧ  | 213 | Õ  | 245 | õ |
| 22 | —   | 54 | 6  | 86 | V | 118 | v | 150 | Ӯ  | 182 | ú  | 214 | Ö  | 246 | ö |
| 23 |     | 55 | 7  | 87 | W | 119 | w | 151 | Ӱ  | 183 | û  | 215 | Œ  | 247 | œ |
| 24 | ø   | 56 | 8  | 88 | X | 120 | x | 152 | Ӯ  | 184 | ÿ  | 216 | Ø  | 248 | ø |
| 25 | ı   | 57 | 9  | 89 | Y | 121 | y | 153 | Ӷ  | 185 | ż  | 217 | Ù  | 249 | ù |
| 26 | ј   | 58 | :  | 90 | Z | 122 | z | 154 | Ӷ  | 186 | ž  | 218 | Ú  | 250 | ú |
| 27 | ff  | 59 | ;  | 91 | [ | 123 | { | 155 | ӷ  | 187 | ż  | 219 | Û  | 251 | û |
| 28 | fi  | 60 | <  | 92 | \ | 124 |   | 156 | IJ | 188 | ij | 220 | Ü  | 252 | ü |
| 29 | fl  | 61 | =  | 93 | ] | 125 | } | 157 | í  | 189 | í  | 221 | Ý  | 253 | ý |
| 30 | ffi | 62 | >  | 94 | ^ | 126 | ~ | 158 | đ  | 190 | đ  | 222 | Þ  | 254 | þ |
| 31 | ffl | 63 | ?  | 95 | _ | 127 | - | 159 | §  | 191 | ƒ  | 223 | SS | 255 | ß |

TAB. E.2 – Les caractères de la police eer et l'encodage T1

|    |    |    |   |    |   |     |   |
|----|----|----|---|----|---|-----|---|
| 0  | НЬ | 32 | " | 64 | ^ | 96  | ' |
| 1  | Љ  | 33 | ! | 65 | А | 97  | а |
| 2  | Џ  | 34 | " | 66 | Б | 98  | б |
| 3  | Ә  | 35 | Ӯ | 67 | Ҷ | 99  | ң |
| 4  | І  | 36 | ~ | 68 | ҳ | 100 | ҷ |
| 5  | Ҽ  | 37 | % | 69 | Ҽ | 101 | ҽ |
| 6  | Ӯ  | 38 | ' | 70 | Փ | 102 | ֆ |
| 7  | Ҥ  | 39 | , | 71 | Ր | 103 | ր |
| 8  | Ҋ  | 40 | ( | 72 | Ҳ | 104 | ҳ |
| 9  | ڶ  | 41 | ) | 73 | Ӣ | 105 | ӣ |
| 10 | Ҿ  | 42 | * | 74 | Ҵ | 106 | ҵ |
| 11 | Ҹ  | 43 | Ӱ | 75 | һ | 107 | һ |
| 12 | и  | 44 | , | 76 | Ҹ | 108 | Ҹ |
| 13 | ҿ  | 45 | - | 77 | ҹ | 109 | ҹ |
| 14 | ԡ  | 46 | . | 78 | Ҥ | 110 | Ҥ |
| 15 | ԡ  | 47 | / | 79 | Ӯ | 111 | Ӯ |
| 16 | Ҽ  | 48 | 0 | 80 | ҽ | 112 | ҽ |
| 17 | Ҿ  | 49 | 1 | 81 | ҷ | 113 | ҷ |
| 18 | Ҿ  | 50 | 2 | 82 | Ҹ | 114 | Ҹ |
| 19 | Ҽ  | 51 | 3 | 83 | һ | 115 | һ |
| 20 | һ  | 52 | 4 | 84 | ҹ | 116 | ҹ |
| 21 | Ӫ  | 53 | 5 | 85 | һ | 117 | һ |
| 22 | Ҹ  | 54 | 6 | 86 | Ҹ | 118 | Ҹ |
| 23 | Ҽ  | 55 | 7 | 87 | ҽ | 119 | ҽ |
| 24 | ҽ  | 56 | 8 | 88 | ҽ | 120 | ҽ |
| 25 | ҷ  | 57 | 9 | 89 | һ | 121 | һ |
| 26 | Ҹ  | 58 | : | 90 | Ҹ | 122 | Ҹ |
| 27 | Ҹ  | 59 | ; | 91 | [ | 123 | - |
| 28 | һ  | 60 | « | 92 | “ | 124 | — |
| 29 | Ҹ  | 61 | 1 | 93 | ] | 125 | № |
| 30 | Ҹ  | 62 | » | 94 | Ҹ | 126 | Ҹ |
| 31 | Ҹ  | 63 | ? | 95 | Ҿ | 127 | Ҿ |

ТАБ. E.3 – Les caractères de la police *wncyr* et l'encodage OT2

|   |  |   |  |   |  |
|---|--|---|--|---|--|
| A |  | J |  | S |  |
| B |  | K |  | T |  |
| C |  | L |  | U |  |
| D |  | M |  | V |  |
| E |  | N |  | W |  |
| F |  | O |  | X |  |
| G |  | P |  | Y |  |
| H |  | Q |  | Z |  |
| I |  | R |  |   |  |

TAB. E.4 – *Les lettrines de la police yinit*

|    |   |    |   |    |   |     |   |     |   |     |   |     |   |     |   |
|----|---|----|---|----|---|-----|---|-----|---|-----|---|-----|---|-----|---|
| 0  | - | 32 | ~ | 64 | ~ | 96  | ' | 128 | ά | 160 | ǎ | 192 | ā | 224 | ā |
| 1  | " | 33 | ! | 65 | A | 97  | α | 129 | έ | 161 | ξ | 193 | ά | 225 | ā |
| 2  | " | 34 | " | 66 | B | 98  | β | 130 | ή | 162 | ῃ | 194 | ή | 226 | ή |
| 3  |   | 35 | " | 67 |   | 99  | ς | 131 | ό | 163 | օ | 195 | ή | 227 | ή |
| 4  |   | 36 | " | 68 | Δ | 100 | δ | 132 | ώ | 164 | օ | 196 | օ | 228 | օ |
| 5  |   | 37 | % | 69 | E | 101 | ε | 133 | ί | 165 | է | 197 | լ | 229 | օ |
| 6  |   | 38 |   | 70 | Φ | 102 | φ | 134 | ύ | 166 | ն | 198 | ն | 230 | ր |
| 7  |   | 39 | ' | 71 | Γ | 103 | γ | 135 | ᾶ | 167 | ն | 199 |   | 231 |   |
| 8  | ü | 40 | ( | 72 | H | 104 | η | 136 | ὰ | 168 | ὰ | 200 | ă | 232 | ă |
| 9  | ť | 41 | ) | 73 | I | 105 | ι | 137 | ὲ | 169 | ὲ | 201 | ă | 233 | ă |
| 10 | ť | 42 | * | 74 | Θ | 106 | θ | 138 | ὴ | 170 | ὴ | 202 | ὴ | 234 | ὴ |
| 11 | ł | 43 | + | 75 | K | 107 | χ | 139 | օ | 171 | օ | 203 | ń | 235 | ń |
| 12 | ü | 44 | , | 76 | Λ | 108 | λ | 140 | ວ | 172 | ວ | 204 | ວ | 236 | ວ |
| 13 | Ծ | 45 | - | 77 | M | 109 | μ | 141 | լ | 173 | լ | 205 | լ | 237 | լ |
| 14 | Ը | 46 | . | 78 | N | 110 | ν | 142 | ւ | 174 | ւ | 206 | ւ | 238 | ւ |
| 15 | Ծ | 47 | / | 79 | O | 111 | օ | 143 | ն | 175 |   | 207 | Ւ | 239 |   |
| 16 |   | 48 | 0 | 80 | Π | 112 | π | 144 | ա | 176 | ա | 208 | Ճ | 240 | Ճ |
| 17 |   | 49 | 1 | 81 | X | 113 | χ | 145 | է | 177 | է | 209 | Ճ | 241 | Ճ |
| 18 |   | 50 | 2 | 82 | P | 114 | ρ | 146 | ի | 178 | ի | 210 | դ | 242 | դ |
| 19 |   | 51 | 3 | 83 | Σ | 115 | σ | 147 | օ | 179 | օ | 211 | ն | 243 | ն |
| 20 |   | 52 | 4 | 84 | T | 116 | τ | 148 | ວ | 180 | ວ | 212 | Փ | 244 | Փ |
| 21 |   | 53 | 5 | 85 | Y | 117 | υ | 149 | լ | 181 | լ | 213 | Վ | 245 | Վ |
| 22 |   | 54 | 6 | 86 | ~ | 118 |   | 150 | ն | 182 | ն | 214 | Ջ | 246 | Ջ |
| 23 |   | 55 | 7 | 87 | Ω | 119 | ω | 151 | օ | 183 |   | 215 |   | 247 |   |
| 24 |   | 56 | 8 | 88 | Ξ | 120 | ξ | 152 | ڏ | 184 | ڏ | 216 | ڏ | 248 |   |
| 25 |   | 57 | 9 | 89 | Ψ | 121 | ψ | 153 | ڦ | 185 | ڦ | 217 | ڦ | 249 |   |
| 26 |   | 58 | : | 90 | Z | 122 | ζ | 154 | ڦ | 186 | ڦ | 218 | ڦ | 250 |   |
| 27 |   | 59 | . | 91 | [ | 123 | « | 155 | ڦ | 187 | ڦ | 219 | ڦ | 251 | ڦ |
| 28 | ' | 60 | ° | 92 | ~ | 124 | , | 156 | ڦ | 188 | ڦ | 220 | ڦ | 252 | ڦ |
| 29 | ' | 61 | = | 93 | ] | 125 | » | 157 | ڦ | 189 | ڦ | 221 | ڦ | 253 |   |
| 30 | ˇ | 62 | ° | 94 | ~ | 126 | ~ | 158 | ڦ | 190 | ڦ | 222 | ڦ | 254 |   |
| 31 | - | 63 | ; | 95 |   | 127 | — | 159 | ڦ | 191 |   | 223 |   | 255 | ߵ |

TAB. E.5 – Tableau des caractères de la police greek

|    |    |    |    |    |   |     |   |
|----|----|----|----|----|---|-----|---|
| 0  | à  | 32 |    | 64 |   | 96  | ø |
| 1  | é  | 33 | !  | 65 | À | 97  | ä |
| 2  | è  | 34 | "  | 66 | Ã | 98  | â |
| 3  | ç  | 35 | ¤  | 67 | Œ | 99  | ç |
| 4  | ë  | 36 | ¤  | 68 | œ | 100 | đ |
| 5  | ë  | 37 |    | 69 | ¢ | 101 | € |
| 6  | æ  | 38 | ƒ  | 70 | ƒ | 102 | ƒ |
| 7  | œ  | 39 | .  | 71 | ß | 103 | g |
| 8  | œ  | 40 | (  | 72 | ß | 104 | h |
| 9  | à  | 41 | )  | 73 | Ĳ | 105 | i |
| 10 | ÿ  | 42 | ſ  | 74 | Ĳ | 106 | j |
| 11 | ff | 43 | ſſ | 75 | ꝑ | 107 | h |
| 12 | ü  | 44 | ,  | 76 | ꝑ | 108 | I |
| 13 | ñ  | 45 | :  | 77 | ꝑ | 109 | m |
| 14 | ññ | 46 | .  | 78 | ꝑ | 110 | n |
| 15 | ññ | 47 | ññ | 79 | ꝑ | 111 | o |
| 16 | ı  | 48 | o  | 80 | ꝑ | 112 | p |
| 17 | j  | 49 | ı  | 81 | ꝑ | 113 | q |
| 18 | þ  | 50 | 2  | 82 | ꝑ | 114 | r |
| 19 | pa | 51 | 3  | 83 | ꝑ | 115 | t |
| 20 | pe | 52 | 4  | 84 | ꝑ | 116 | t |
| 21 | po | 53 | 5  | 85 | ꝑ | 117 | u |
| 22 | ij | 54 | 6  | 86 | ꝑ | 118 | v |
| 23 | ø  | 55 | 7  | 87 | ꝑ | 119 | w |
| 24 | ñ  | 56 | 8  | 88 | ꝑ | 120 | x |
| 25 | ß  | 57 | 9  | 89 | ꝑ | 121 | y |
| 26 | æ  | 58 | :  | 90 | ꝑ | 122 | ſ |
| 27 | œ  | 59 | ;  | 91 | ä | 123 | - |
| 28 | ø  | 60 | ſt | 92 | ë | 124 | — |
| 29 | ll | 61 | ſt | 93 | ð | 125 |   |
| 30 | œ  | 62 | ñ  | 94 | ü | 126 |   |
| 31 | ñ  | 63 | ?  | 95 | “ | 127 |   |

TAB. E.6 – Tableau des caractères de la police *ygoth*

|    |   |    |   |    |   |     |   |     |   |
|----|---|----|---|----|---|-----|---|-----|---|
| 0  |   | 32 |   | 64 |   | 96  | ' | 128 |   |
| 1  |   | 33 | ! | 65 | ꝑ | 97  | ꝑ | 129 | ꝑ |
| 2  |   | 34 | " | 66 | Ꝓ | 98  | ꝑ | 130 | ꝑ |
| 3  |   | 35 | # | 67 | ꝓ | 99  | ꝑ | 131 | ꝑ |
| 4  |   | 36 |   | 68 | Ꝕ | 100 | ꝑ | 132 | ꝑ |
| 5  |   | 37 | % | 69 | ꝕ | 101 | ꝑ | 133 | ꝑ |
| 6  |   | 38 |   | 70 | Ꝗ | 102 | ꝑ | 134 | ꝑ |
| 7  |   | 39 | , | 71 | ꝗ | 103 | ꝑ | 135 |   |
| 8  |   | 40 | ( | 72 | Ꝙ | 104 | ꝑ | 136 |   |
| 9  |   | 41 | ) | 73 | ꝙ | 105 | ꝑ | 137 | ꝑ |
| 10 |   | 42 | * | 74 | Ꝛ | 106 | ꝑ | 138 | ꝑ |
| 11 |   | 43 | + | 75 | ꝛ | 107 | ꝑ | 139 | . |
| 12 |   | 44 | , | 76 | Ꝝ | 108 | ꝑ | 140 |   |
| 13 |   | 45 | : | 77 | ꝝ | 109 | ꝑ | 141 | ꝑ |
| 14 |   | 46 | . | 78 | Ꝟ | 110 | ꝑ | 142 |   |
| 15 |   | 47 | / | 79 | ꝟ | 111 | ꝑ | 143 |   |
| 16 | ı | 48 | ø | 80 | Ꝡ | 112 | ꝑ | 144 | é |
| 17 | ј | 49 | I | 81 | ꝡ | 113 | ꝑ | 145 | ë |
| 18 | ‘ | 50 | 2 | 82 | Ꝣ | 114 | ꝑ | 146 |   |
| 19 | ‘ | 51 | 3 | 83 | ꝣ | 115 | ꝑ | 147 |   |
| 20 | ‘ | 52 | 4 | 84 | Ꝥ | 116 | ꝑ | 148 |   |
| 21 | ‘ | 53 | 5 | 85 | ꝥ | 117 | ꝑ | 149 |   |
| 22 | ‘ | 54 | 6 | 86 | Ꝧ | 118 | ꝑ | 150 |   |
| 23 | ‘ | 55 | 7 | 87 | ꝧ | 119 | ꝑ | 151 |   |
| 24 | ‘ | 56 | 8 | 88 | Ꝩ | 120 | ꝑ | 152 |   |
| 25 |   | 57 | 9 | 89 | ꝩ | 121 | ꝑ | 153 | ó |
| 26 | ß | 58 | : | 90 | Ꝫ | 122 | ꝑ | 154 | ö |
| 27 |   | 59 |   | 91 | [ | 123 | — | 155 |   |
| 28 |   | 60 | § | 92 | „ | 124 | — | 158 | ú |
| 29 |   | 61 | = | 93 | ] | 125 | ” | 159 | ü |
| 30 |   | 62 |   | 94 | ^ | 126 | ~ | 164 | § |
| 31 |   | 63 | ˇ | 95 | · | 127 |   | 167 | ß |

TAB. E.7 – Tableau des caractères de la police *yswab*

|    |   |    |    |    |   |     |   |     |   |     |   |
|----|---|----|----|----|---|-----|---|-----|---|-----|---|
| 0  |   | 32 | /  | 64 |   | 96  | ' | 152 | Y | 255 | G |
| 1  | . | 33 | /  | 65 | A | 97  | a | 184 | y | 189 | / |
| 2  | - | 34 | "  | 66 | B | 98  | b | 190 | o | 191 | L |
| 3  | - | 35 | ,  | 67 | C | 99  | c | 192 | A | 224 | à |
| 4  | - | 36 | \$ | 68 | D | 100 | d | 193 | A | 225 | í |
| 5  |   | 37 | %  | 69 | E | 101 | e | 194 | A | 226 | é |
| 6  | . | 38 | J  | 70 | F | 102 | f | 195 | A | 227 | á |
| 7  |   | 39 | ,  | 71 | G | 103 | g | 196 | A | 228 | â |
| 8  | - | 40 | /  | 72 | H | 104 | h | 197 | A | 229 | ä |
| 9  |   | 41 | /  | 73 | I | 105 | i | 198 | A | 230 | æ |
| 10 | . | 42 | *  | 74 | J | 106 | j | 199 | C | 231 | ç |
| 11 |   | 43 |    | 75 | K | 107 | k | 200 | E | 232 | è |
| 12 |   | 44 | ,  | 76 | L | 108 | l | 201 | E | 233 | é |
| 13 |   | 45 |    | 77 | M | 109 | m | 202 | E | 234 | ê |
| 14 | , | 46 | .  | 78 | N | 110 | n | 203 | E | 235 | ë |
| 15 | , | 47 | /  | 79 | O | 111 | o | 204 | J | 236 | ì |

TAB. E.8 – Tableau des caractères de la police calligra

|    |    |    |    |    |   |     |    |     |    |     |   |
|----|----|----|----|----|---|-----|----|-----|----|-----|---|
| 16 | "  | 48 | ø  | 80 | P | 112 | p  | 205 | J  | 237 | i |
| 17 | "  | 49 | ø  | 81 | Q | 113 | g  | 206 | J̄ | 238 | é |
| 18 | "  | 50 | æ  | 82 | R | 114 | ø  | 207 | J̄ | 239 | è |
| 19 | "  | 51 | ɔ  | 83 | S | 115 | s  | 208 |    | 240 |   |
| 20 | »  | 52 | ɛ  | 84 | T | 116 | ø  | 209 | N  | 241 | ñ |
| 21 |    | 53 | ɔ  | 85 | U | 117 | uu | 210 | Ø  | 242 | à |
| 22 | —  | 54 | ø  | 86 | V | 118 | v  | 211 | Ø  | 243 | á |
| 23 |    | 55 | ø  | 87 | W | 119 | uw | 212 | Ø  | 244 | ó |
| 24 |    | 56 | ø  | 88 | X | 120 | x  | 213 | Ø  | 245 | ö |
| 25 |    | 57 | ø  | 89 | Y | 121 | y  | 214 | Ø  | 246 | ö |
| 26 |    | 58 | .. | 90 | Z | 122 | z  | 215 | Œ  | 247 | œ |
| 27 |    | 59 | .. | 91 | / | 123 | /  | 216 | Ø  | 248 | ø |
| 28 | ſ  | 60 |    | 92 |   | 124 |    | 217 | U  | 249 | à |
| 29 | ſſ | 61 |    | 93 | / | 125 | /  | 218 | U  | 250 | á |
| 30 |    | 62 |    | 94 | ^ | 126 | ~  | 219 | U  | 251 | ó |
| 31 |    | 63 | ø  | 95 | — | 127 | -  | 220 | U  | 252 | ö |

TAB. E.9 – Tableau des caractères de la police *calligra* (suite)

|           |    |   |     |   |     |   |
|-----------|----|---|-----|---|-----|---|
| Sol       | 0  | ○ | 100 | ○ | 200 | ○ |
| Mercurius | 1  | ☿ | 101 | ☿ | 201 | ♀ |
| Venus     | 2  | ♀ | 102 | ♀ | 202 | ♀ |
| Terra     | 3  | ♂ | 103 | ♂ | 203 | ♂ |
| Mars      | 4  | ♂ | 104 | ♂ | 204 | ♂ |
| Iuppiter  | 5  | ♃ | 105 | ♃ | 205 | ♃ |
| Saturnus  | 6  | ♄ | 106 | ♄ | 206 | ♄ |
| Uranus    | 7  | ♅ | 107 | ♅ | 207 | ♅ |
| Neptunus  | 8  | ♆ | 108 | ♆ | 208 | ♆ |
| Pluto     | 9  | ♇ | 109 | ♇ | 209 | ♇ |
| Luna      | 10 | ☽ | 110 | ☽ | 210 | ☽ |
| Aries     | 11 | ♈ | 111 | ♈ | 211 | ♈ |
| Taurus    | 12 | ♉ | 112 | ♉ | 212 | ♉ |
| Gemini    | 13 | ♊ | 113 | ♊ | 213 | ♊ |
| Cancer    | 14 | ♋ | 114 | ♋ | 214 | ♋ |

TAB. E.10 – Tableau des caractères de la police astrosym

|             |    |  |     |  |     |  |
|-------------|----|--|-----|--|-----|--|
| Leo         | 15 |  | 115 |  | 215 |  |
| Virgo       | 16 |  | 116 |  | 216 |  |
| Libra       | 17 |  | 117 |  | 217 |  |
| Scorpio     | 18 |  | 118 |  | 218 |  |
| Sagittarius | 19 |  | 119 |  | 219 |  |
| Capricornus | 20 |  | 120 |  | 220 |  |
| Aquarius    | 21 |  | 121 |  | 221 |  |
| Pisces      | 22 |  | 122 |  | 222 |  |
| Saturnus 2  | 23 |  | 123 |  | 223 |  |
| Neptunus 2  | 24 |  | 124 |  | 224 |  |
| Pluto 2     | 25 |  | 125 |  | 225 |  |
| Libra 2     | 26 |  | 126 |  | 226 |  |
| Aquarius 2  | 27 |  | 127 |  | 227 |  |
| Aquarius 3  | 28 |  | 128 |  | 228 |  |

TAB. E.11 – Tableau des caractères de la police astrosym (suite)

|    |   |    |   |    |   |    |   |    |   |    |   |     |   |     |   |
|----|---|----|---|----|---|----|---|----|---|----|---|-----|---|-----|---|
| 0  | . | 16 | . | 32 | . | 48 | . | 64 | . | 80 | . | 96  |   | 112 |   |
| 1  | . | 17 | : | 33 | : | 49 | : | 65 | . | 81 | . | 97  |   | 113 |   |
| 2  | . | 18 | : | 34 | : | 50 | : | 66 | . | 82 | . | 98  | . | 114 |   |
| 3  | . | 19 | : | 35 | : | 51 | : | 67 | . | 83 | . | 99  | . | 115 |   |
| 4  | . | 20 | : | 36 | : | 52 | : | 68 |   | 84 |   | 100 | . | 116 |   |
| 5  | : | 21 | : | 37 | : | 53 | : | 69 |   | 85 |   | 101 | . | 117 |   |
| 6  | . | 22 | : | 38 | : | 54 | : | 70 |   | 86 |   | 102 | . | 118 |   |
| 7  | . | 23 | : | 39 | : | 55 | : | 71 | . | 87 |   | 103 | . | 119 |   |
| 8  | . | 24 | . | 40 | : | 56 | . | 72 | . | 88 | . | 104 | . | 120 |   |
| 9  | : | 25 | . | 41 | : | 57 | : | 73 | . | 89 | . | 105 | . | 121 |   |
| 10 | : | 26 | . | 42 | : | 58 | : | 74 | . | 90 | . | 106 |   | 122 | ■ |
| 11 | : | 27 | . | 43 | : | 59 | : | 75 | . | 91 | . | 107 |   |     |   |
| 12 | . | 28 | . | 44 | : | 60 | . | 76 | . | 92 | . | 108 | . |     |   |
| 13 | : | 29 | . | 45 | : | 61 | : | 77 | . | 93 | . | 109 | . |     |   |
| 14 | : | 30 | . | 46 | : | 62 | . | 78 | . | 94 | . | 110 | . |     |   |
| 15 | . | 31 |   | 47 |   | 63 |   | 79 | . | 95 |   | 111 | . |     |   |

TAB. E.12 – Tableau des caractères de la police gray

```

 ; le constructeur de la liste = (d,p,x,y) avec x,y qui sont des
 ; constructeurs de la liste : range de 0 à 255, p,x,y sont
 ; des entiers binaires
 ; constructeur keepage = (g,s-a,s)(d,p,x,y) où x,y,s,t sont
 ; des entiers binaires

```

## Annexe F

# *logocomplet* : le programme

La métapolice que nous avons construite au chapitre 5 sous le nom de *logocomplet* a été élaborée pas à pas et au prix d'ajustements successifs pour obtenir des définitions convenables des empattements, de certaines variables booléennes, etc. Cette annexe en donne le programme complet. La structure adoptée comporte un fichier pilote, un fichier base et des fichiers de paramètres pour toutes les variantes que l'on voudra définir.

Le fichier base est un fichier dans lequel sont réunies les macros spécifiques définies pour l'élaboration de cette métapolice. Il sera baptisé, par exemple, *completbase.mf* et sera appelé en préambule du fichier pilote. Voici son contenu :

```

% Ce fichier contient les macros spécifiques
% de la métapolice logocomplet.
% La déclaration suivante est une
% précaution. Elle signifie que
% si la variable [completbase] est connue,
% c'est que le fichier a déjà été chargé.
completbase:=1;

def beginlogochar(expr code, unit_width) =
beginchar(code,if courrier : larg_tt#
else : unit_width*u#+2s# fi,ht#,0);
pickup logo_pen enddef;

def super_half(suffix i,j,k) =
draw z.i{0,y.j-y.i}
... (.8[x.j,x.i],.8[y.i,y.j]) {z.j-z.i} courrier=false fi;
... z.j{x.k-x.i,0}
... (.8[x.j,x.k],.8[y.k,y.j]) {z.k-z.j} courrier=true fi;
... z.k{0,y.k-y.j} enddef;

```

```

def super_crescent(suffix i,j,k) =
draw z.i{x.j-x.i,0}
... (.8[x.i,x.j],.8[y.j,y.i]) {z.j-z.i}
... z.j{0,y.k-y.i}
... (.8[x.k,x.j],.8[y.j,y.k]) {z.k-z.j}
... z.k{x.k-x.j,0} enddef;

def decalage(expr z,u)=
rotatedaround (z,u) shifted -z scaled 1/(2cosd u)
shifted z enddef;

def empattement (suffix $)(expr empat_g,empat_d,
empat_e,empat_l,empat_h,theta,pos)=
cutoff(z$,90-pos);penpos$(empat_l/abs sind theta,0);
x$c=x$-empat_g;
x$d=x$+empat_d;
y$c=y$d=bot y$;
x$b=x$c; x$e=x$d;
y$b-y$c=y$e-y$d=empat_e;
z$a-z$l=z$f-z$r=(empat_h/abs sind theta)* dir theta;
z$g=whatever[z$a,z$l];y$g=y$b;
z$h=whatever[z$f,z$r];y$h=y$e;
fill (z$a{z$l-z$a} if courrier : -- z$g -- else :
.. {left}fi z$b--z$c--z$d--z$e{left}
if courrier : -- z$h -- else : .. {z$f-z$r}fi
z$f--cycle) rotatedarround(z$,pos)
enddef;

def empat (expr a,b,c,d,e,f,g)(text t)=
forsuffixes i=t:
empattelement(i,a,b,c,d,e,f,g); endfor enddef;

def empatinf(text t)=
forsuffixes i=t:
empattelement(i,empat_g,empat_d,empat_e,empat_l,empat_h,theta,0);
endfor enddef;

def empatsup(text t)=
forsuffixes i=t:
empattelement(i,empat_g,empat_d,empat_e,empat_l,empat_h,theta,180);
endfor enddef;

def trace_contour =
if pourtour = true :
cull currentpicture keeping (1,infinity);

```

```

image := currentpicture;
addto currentpicture also image + image shifted 2left
+ image shifted 2right + image shifted 2up
+ image shifted 2down;
cull currentpicture keeping (1,4);
fi enddef;

def negatif =
if neg = true : pickup pensquare scaled px;
filldraw (0,0)--(w,0)--(w,h)--(0,h)--cycle;
cull currentpicture keeping (1,1) ;
fi enddef;

```

Un fichier de paramètres pour faire une police avec empattements en corps 10 s'appellerait logocompletrm10.mf et contiendrait simplement :

```

boolean serif,courrier;
serif := true;
courrier := false;
font_size 10pt#;
ht#:=6pt#;
xgap#:=0.6pt#;
u#:=4/9pt#;
s#:=0;
o#:= if serif : 0 else :1/9pt# fi;
px#:=2/3pt#;
input logocomplet

```

Il faut noter cette précaution supplémentaire qui consiste à annuler le paramètre *o* (*overshoot*) dans une police avec empattements : si on ne le faisait pas, les empattements ne seraient pas sur la ligne de base mais déborderaient sous cette ligne, ce qui n'est pas souhaitable.

Finalement voici le fichier pilote complet, lettre par lettre :

```

% Routines génériques pour la métapolice << logocomplet >>.
% Préambule :
if unknown completbase : input completbase fi;
mode_setup;
if unknown slant: slant:=0 else: currenttransform:=
identity slanted slant yscaled aspect_ratio fi;
if unknown courrier : boolean courrier:=false fi;
if unknown serif : boolean serif; serif:=false fi;
if unknown pourtour : boolean pourtour; pourtour:=false fi;
if unknown neg : boolean neg; neg:=false fi;

```

```

picture lettrec,lettrep,lettreo;
% Paramètres caractéristiques :
ygap#:=(ht#/13.5u#)*xgap#;
yo#:=o#;
leftstemloc#:=2.5u#+s#;
barheight#:=.45ht#;
py#:=.9px#;
larg_tt#:=14 u#;
empat_g# := 1.6u#;
empat_d# := 1.6u#;
empat_e# := .4py#;
empat_l# := px# ;
empat_h# := 2py# ;
% Instructions de pixellisation :
define_pixels(s,u,larg_tt);
define_pixels(empat_g,empat_d,empat_e,empat_l,empat_h);
define_whole_pixels(xgap);
define_whole_vertical_pixels(ygap);
define_blacker_pixels(px,py);
pickup pencircle xscaled px yscaled py;
logo_pen:=savepen;
define_good_x_pixels(leftstemloc);
define_good_y_pixels(barheight);
define_corrected_pixels(o);
define_horizontal_corrected_pixels(ho);

% Instruction supplémentaire
extra_endchar:="negatif;";

% Début des définitions des lettres :
beginlogochar("A",15);
x1=.5w; x2=x4=leftstemloc;
x3=x5=w-x2;
top y1=h+o; y2=y3=barheight;
bot y4=bot y5=-o;
draw z4--z2--z3--z5;
super_half(2,1,3);
if serif :
theta:=90;
empatinf(4,5) fi;
labels(1,2,3,4,5);
endchar;

% Les lettres P, R et B se suivent dans cet ordre.

```

```

beginlogochar("P",12.5);
x1=x2=x3=leftstemloc;
x4=x5=.618[x1,w+1.5u-x1];
y2=y5; y3=y4;
bot y1=-o; top y3=h;
y2=barheight;
z6 = z5 decalage (z4,40);
path p;
p:= z3--z4{right}..{down}z6..{left}z5--z2;
draw z1--z3;
draw p;
if serif :
theta:=90;
empatinf(1);
empatsup(3) fi;
lettrep := currentpicture;
labels(range 1 thru 6);
endchar;

beginlogochar("R",14);
x1=x2=leftstemloc;
bot y1=-o;y8=y1;
y2=barheight;
z12=.618[z1,z2];
vardef test(expr t)=
- angle(direction t of p)
+ angle (z12- point t of p) < 0 enddef;
tolerance:=.001;
t = solve test(2,3);
z7 = point t of p;
(z8-z7)dotprod(z12-z7)=0;
addto currentpicture also lettrep;
draw z7--z8;
if serif :
theta:=angle (z7-z8);
empatinf(8) fi;
labels(1,2,7,8,12);
endchar;

beginlogochar("B",15);
x1=leftstemloc;
bot y1=-o;y7=y1;
y5=barheight;x5=x7=.618[x1,w-x1];
z8 = z7 decalage (z5,50);
addto currentpicture also lettrep;
draw z5{right}..z8..{left}z7;

```

```

draw z1--z7;
labels(1,5,7,8);
endchar;

% Les lettres G et C se suivent dans cet ordre.
beginlogochar("G",15);
x4=.618leftstemloc;
y8=y4=.5h;
bot y5=bot y6=0;
top y2 = top y3 = h;
x3=x5=x4+.309w;
x2=x6=w-x3;
x8=w-x4;
path chem;
chem:= z8{up}..z2{left}--z3{left}
 ..z4..{right}z5--z6{right}..{up}cycle;
t1:=directiontime (-1,sqrt 3) of chem;
z1 = point t1 of chem;
t2:=directiontime (1,sqrt 3) of chem;
z7 = point t2 of chem;
x10=x7-2u;
x11=x7+1.5u;
y9=y10=y11=.8barheight;
x9=x7;
draw subpath (1,5) of chem;
draw z1..{left}z2;
draw z6{right}..z7;
lettrec:=currentpicture;
erase draw z6{right}..z7;
draw z6{right}..{up}z7--z9;
draw z10--z11;
labels(range 1 thru 8);
endchar;

beginlogochar("C",15);
addto currentpicture also lettrec;
endchar;

beginlogochar("D",14.5);
x1=x3=leftstemloc;
x4=x5=.5[x1,w-x1];
y1=y5; y3=y4;
bot y1=0; top y3=h;
y2=barheight;
draw z5--z1--z3--z4;

```

```

x2=.309[w-x1,lft w];
super_crescent(4,2,5);
labels(1,2,3,4,5);
if serif :
theta := 90;
empatinf(1);
empatsup(3) fi;
endchar;

beginlogochar("E",14);
x1=x2=x3=leftstemloc;
x4=x6=w-x1+ho;
x5=x4-xgap;
y1=y6; y2=y5; y3=y4;
bot y1=0; top y3=h;
y2=barheight;
draw z6--z1--z3--z4; draw z2--z5;
if serif :
theta := 90;
empattement(4,.1.4u,.5py,.5py,.9py,2u,90,90);
empattement(5,.1.2u,.1.2u,.3py,py,2u,90,90);
empattement(6,.5py,.1.4u,.5py,.9py,2u,90,90);
empatinf(1); empatsup(3) fi;
labels(1,2,3,4,5,6);
endchar;

beginlogochar("F",14);
x1=x2=x3=leftstemloc;
x4=w-x1+ho;
x5=x4-xgap;
y2=y5; y3=y4;
bot y1=-o; top y3=h;
y2=barheight;
draw z1--z3--z4; draw z2--z5;
if serif :
theta := 90;
empattement(4,.1.4u,.5py,.5py,.9py,2u,90,90);
empattement(5,.1.2u,.1.2u,.3py,py,2u,90,90);
empatinf(1);
empatsup(3) fi;
labels(1,2,3,4,5);
endchar;

beginlogochar("H",14);
x1=x2=x3=leftstemloc;

```

```

x4=x6=w-x1; x5=x4; ;[x .x1,x-x]00,-0x
y1=y6; y2=y5; y3=y4; ;(3,1,A)macoco_reque
bot y1=-o; top y3=h+o; ;(3,3,3,3,1)aligned
y2=barheight; ;times li
draw z1--z3; ;specif
draw z4--z6; ;specif
draw z2--z5; ;specif
if serif : ;specif
theta := 90; ;specif
empatinf(1,6); ;specif
empatsup(3,4) fi; ;specif
labels(range 1 thru 6); ;specif
endchar;

beginlogochar("I",7);
if .5w<>good.x .5w: change_width; fi
x1=x2=x3=.5w;
y3=1.2h;
bot y1=-o; top y2 = h+o;
draw z2--z1;
if serif :
theta := 90;
empatinf(1);
empatsup(2) fi;
labels(1,2,3);
endchar;

beginlogochar("J",11);
x1=x5=w-leftstemloc;
bot y2 = bot y3=0;
top y1 = h+o;
x3=1.5x4;
x2=x5-x3+x4;
x4=leftstemloc;
y5=y4=.5barheight;
draw z1--z5{down}..{left}z2--z3{left}..{up}z4;
if serif :
theta := 90;
empatsup(1,4) fi;
labels(1,2,3,4);
endchar;

beginlogochar("K",15);
x1=x2=x3=leftstemloc;
x4=x6=w-x1; y3=y4;

```

```

bot y1=bot y6=-o; ;(3,p,S,C,I)aleksi
top y3=h+o; ;radiobase
y5=barheight;
vardef test(expr t)=
length((x4,h/2)-(t,y5))>h/2 enddef;
x5=solve test(0,x4);
z2=whatever[z4,z5];
draw z1--z3; draw z2--z4;
draw z5--z6;
if serif :
theta := 90;
theta1=angle (z4-z5);
theta2=angle (z5-z6);
empattement(4,1.6u,1.6u,.4py,.95px,py,theta1,180); ;(4,t)intreque
empattement(6,1.6u,1.6u,.4py,.95px,py,theta2,0); ;(12,(d,S)quatreque
empatinf(1); ;(3,p,S,C,I)aleksi
empatsup(3) fi; ;radiobase
labels(1,2,3,4,5,6); ;radiobase
endchar;

beginlogochar("L",13);
x1=x2=leftstemloc;
x3=w-x1+ho;
y1=y3;
bot y1=0; top y2=h+o;
draw z3--z1--z2;
if serif :
theta := 90;
empatinf(1);
empatsup(2);
empattement(3,.5py,1.4u,.5py,.9py,2u,90,90) fi;
labels(1,2,3);
endchar;

beginlogochar("M",18);
x1=x2=leftstemloc;
x4=x5=w-x1;
x3=w-x3; y1=y5;
y2=y4; bot y1=-o;
top y2=h+o;
y3=y1+ygap;
draw z1--z2--z3--z4--z5;
if serif :
theta := 90 ;
empatinf(1,5);
empatsup(2,4) fi;

```

```

labels(1,2,3,4,5);
endchar;

beginlogochar("N",15);
x1=x2=leftstemloc;
x3=x4=x5=w-x1;
bot y1=bot y4=-o;
top y2=top y5=h+o;
y3=y4+ygap;
draw z1--z2--z3;
draw z4--z5;
if serif :
theta := 90 ;
empatinf(1,4);
empatsup(2,5) fi;
labels(1,2,3,4,5);
endchar;

% Les lettres O et Q se suivent dans cet ordre.
beginlogochar("O",15);
x1=x4=.5w;
top y1=h+o;
bot y4=-o;
x2=w-x3=good.x(1.5u+s);
y2=y3=barheight;
super_half(2,1,3);
super_half(2,4,3);
lettreo:=currentpicture;
labels(1,2,3,4);
endchar;

beginlogochar("Q",15);
x5=.5w; bot y5=-o;
x3=w-good.x(1.5u+s);
y3=barheight;
x6=x3+.5u;
y6=y5-.5u;
z53=whatever[z5,z3];
(z3-z5).dotprod(z53-z6)=0;
addto currentpicture also lettreo;
draw z53--z6;
labels(1,2,3,5,6,53);
endchar;

beginlogochar("S",14);

```

```

x3=x8=leftstemloc; x4=x9=rightstemloc;
x2=x4=.382[x3,w-x3]; x5=x7=.618[x3,w-x3];
x1=w-x3; x6=.5[w-x3, lft w];
top y1=h; y4=barheight; bot y8=0;
y1=y2; y3=.5[y2,y4];
y4=y5; y6=.5[y5,y7]; y7=y8;
draw z1--z2; draw z4--z5;
draw z7--z8;
super_crescent(2,3,4);
super_crescent(5,6,7);
if serif :
theta := 90 ;
empattement(1,1.2u,1.2u,.5py,.9py,2u,90,90);
empattement(8,1.4u,.5py,.5py,2u,90,270) fi;
labels(1,2,3,4,5,6,7,8);
endchar; ;(2,2,1)elabel ;zinfobase

beginlogochar("T",13);
italcorr ht#*slant + .5u#;
if .5w<>good.x .5w: change_width; fi
lft x1=-eps; x2=w-x1;
x3=x4=.5w;
y1=y2=y3; top y1=h;
bot y4=-o;
draw z1--z2;
draw z3--z4;
if serif :
theta := 90 ;
empattement(2,1.4u,.5py,.5py,.9py,2u,90,90);
empattement(1,.5py,1.4u,.5py,.9py,2u,90,270);
empatinf(4) fi;
labels(1,2,3,4);
endchar; ;(2,2,1)elabel ;zinfobase

beginlogochar("U",14);
x1=.5w; bot y1=-o;
x2=x4=w-x3=w-x5=good.x(2u);
y2=y3=.618barheight;
top y4 = h+o;
y5=y4;
super_half(2,1,3);
draw z2--z4 ; draw z3--z5;
if serif :

```

```

theta := 90;
empatsup(4,5) fi;
labels(range 1 thru 5);
endchar;

beginlogochar("V",13);
italcorr ht#*slant + .8u#;
lft x1=-eps; x2=w-x1;
x3=.5w; y1=y2;
top y1=h+o; bot y3=-o;
draw z1--z3;
draw z2--z3;
if serif :
theta := 90;
theta1 := angle (z2-z3);
empattement(2,1.6u,1.6u,.4py,px,py,theta1,180);
empattement(1,1.6u,1.6u,.4py,px,py,180-theta1,180);
empattement(3,1.6u,1.6u,.4py,px,py,90,0) fi;
labels(1,2,3);
endchar;

beginlogochar("W",18);
italcorr ht#*slant + .8u#;
lft x1=0; x3=.5w=2x2;
x5=w-x1;
x4=w-x2; y4=y2;
y1=y3-y5; top y1=h+o;
bot y2=-o;
draw z1--z2--z3--z4--z5;
theta:= angle (z3-z2);
theta2:=angle (z1-z2);
theta3:=angle (z5-z4);
if serif :
empatinf(2);
empattement(1,1.6u,1.6u,.4py,px,py,theta2,180);
empattement(4,1.6u,1.6u,.4py,px,py,theta3,0);
empattement(5,1.6u,1.6u,.4py,px,py,theta3,180);
empatsup(3) fi;
labels(1,2,3,4,5);
endchar;

beginlogochar("X",14);
x1=x2-ho=.8leftstemloc;
x4=w-x1=x3+ho;
y1=y4; y2=y3;

```

```

bot y1=-o; top y2=h+0; (001,002,003,004,005,006,007,008,009,010) choose if queue
draw z1--z3; draw z2--z4; (001,002,003,004,005,006,007,008,009,010) choose if queue
if serif :
theta := angle(z3-z1); (001,002,003,004,005,006,007,008,009,010) choose if queue
empattement(1,1.6u,1.6u,.4py,.95px,py,theta,0); (001,002,003,004,005,006,007,008,009,010) choose if queue
empattement(4,1.6u,1.6u,.4py,.95px,py,180-theta,0); (001,002,003,004,005,006,007,008,009,010) choose if queue
empattement(2,1.6u,1.6u,.4py,.95px,py,180-theta,180); (001,002,003,004,005,006,007,008,009,010) choose if queue
empattement(3,1.6u,1.6u,.4py,.95px,py,theta,180) fi; (001,002,003,004,005,006,007,008,009,010) choose if queue
labels(range 1 thru 4); (001,002,003,004,005,006,007,008,009,010) choose if queue
endchar;

beginlogochar("Y",13);
if .5w<>good.x .5w: change_width; fi
x1=x2=.5w;
x3=leftstemloc;
x4=w-x3; bot y1=-o;
y2=barheight;
top y3 = top y4 = h+0;
draw z3--z2--z4;
draw z1--z2;
if serif :
theta := 90;
theta1 := angle (z4-z2);
empatinf(1); (001,002,003,004,005,006,007,008,009,010) choose if queue
empattement(3,1.6u,1.6u,.4py,.95px,py,180-theta1,180); (001,002,003,004,005,006,007,008,009,010) choose if queue
empattement(4,1.6u,1.6u,.4py,.95px,py,theta1,180) fi; (001,002,003,004,005,006,007,008,009,010) choose if queue
labels(range 1 thru 4); (001,002,003,004,005,006,007,008,009,010) choose if queue
endchar;

beginlogochar("Z",13);
x1=x2=leftstemloc;
x3=x4=w-x1+ho;
y1=y4; y2=y3;
bot y1=0; top y2=h+0;
z5=whatever[z1,z3];
y5=y6=y7=barheight;
x6=x5-1.5u; x7=x5+1.5u;
draw z2--z3--z1--z4;
if serif : pickup pensquare scaled .8px fi;
draw z6--z7;
if serif :
theta := angle (z3-z1);
cutoff (z7,-90+theta);
cutoff (z6,90+theta);
pickup logo_pen;
empattement(1,1.6u,1.6u,.4py,.95px,py,theta,0); (001,002,003,004,005,006,007,008,009,010) choose if queue

```

```

empattement(3,1.6u,1.6u,.4py,.95px,py,theta,180);
empattement(4,.5py,1.4u,.5py,.9py,2u,90,90);
empattement(2,.5py,1.4u,.5py,.9py,2u,90,270) fi;
labels(range 1 thru 7);
endchar;

beginlogochar(oct"013",16);
x1=x2=x3=leftstemloc;
x4=x5=x6;x9=w-x1+ho;
z6=.45[z3,z9];
.7(x5-x7)=x9-x8=xgap;
bot y1=-o;y1=y4;
y2=y7=y5=y8=barheight;
top y3=h;
y3=y9;
draw z1--z3--z9;
draw z2--z7;
draw z5--z8;
draw z4--z6;
if serif :
theta := 90 ;
empattement(7,1.2u,1.2u,.3py,py,2u,90,90);
empattement(8,1.2u,1.2u,.3py,py,2u,90,90);
empattement(9,1.4u,.5py,.5py,.9py,2u,90,90);
empatinf(1,4);
empatsup(3) fi;
labels(range 1 thru 9);
endchar;

```

% Instructions pour les ligatures et les approches de paires:

```

ligitable "A": "T" kern -.5u#, "V" kern -1.5u#,
 "W" kern -u#, "Y" kern -u#;
ligitable "F": "F"=:oct"013", "O" kern -u#;
ligitable "L": "V" kern -u#, "W" kern -.8u#,
 "Y" kern -.9u#;
ligitable "P": "O" kern u#;
ligitable "T": "A" kern -.5u#;
ligitable "V": "A" kern -1.5u#;
ligitable "W": "A" kern -u#;
ligitable "Y": "A" kern -u#;
ligitable oct"013": "A" kern -1.5u#;

```

% Valeurs globales :

```

font_quad:=18u#+2s#;
font_normal_space:=6u#+2s#;

```

```

font_normal_stretch:=3u#;
font_normal_shrink:=2u#;
font_identifier:="MFLOGOCOMPLET" if slant<>0: & "SL" fi;
font_coding_scheme:="MAJUSCULES seulement";
end % Fin du fichier pilote

```

## Annexe G

# Les modes disponibles

Le logiciel MFLOGO offre plusieurs modes d'écriture que la sélection correspondante dans les pages de configuration permet en toutes circonstances et au moyen des deux méthodes suivantes. Il existe une table contenant des arguments et toutes les propriétés qui sont utilisées pour écrire un mode reporteur.

On peut choisir une méthode d'écriture particulière grâce seulement pour effet de changer de l'option *mode* à l'appelant. L'application de l'option de *mode* *printing* (ou *print*) change le mode d'écriture en mode reporteur. La table d'écriture qui est choisie se compose à partir des valeurs des trois fonctions *mode*, *mode\_type* et des propriétés aux variables *header*, *footer* et *intersections*. Ces deux dernières contiennent respectivement les caractéristiques des ensembles utilisés pour l'écriture des informations supplémentaires dans les fichiers d'écriture qui sont créés ultérieurement.

La table d'écriture qui est choisie se compose à partir des valeurs des trois fonctions *mode*, *mode\_type* et des propriétés aux variables *header*, *footer* et *intersections*. Ces deux dernières contiennent respectivement les caractéristiques des ensembles utilisés pour l'écriture des informations supplémentaires dans les fichiers d'écriture qui sont créés ultérieurement.

Le tableau des modes possibles est le suivant :

|                      |              |               |
|----------------------|--------------|---------------|
| <i>mode</i>          | <i>print</i> | <i>report</i> |
| <i>mode_type</i>     | <i>text</i>  | <i>text</i>   |
| <i>header</i>        | non          | oui           |
| <i>footer</i>        | non          | oui           |
| <i>intersections</i> | non          | oui           |

Le mode *print* (ou *print*) est le mode d'écriture par défaut. Le mode *report* (ou *report*) est le mode d'écriture par rapport à l'ensemble *header*.

Le mode *text* (ou *text*) est le mode d'écriture par rapport à l'ensemble *header*.

Le mode *print* (ou *print*) est le mode d'écriture par rapport à l'ensemble *header*.

Le mode *report* (ou *report*) est le mode d'écriture par rapport à l'ensemble *header*.

Le mode *text* (ou *text*) est le mode d'écriture par rapport à l'ensemble *header*.

## Annexe G

# Les modes disponibles

Voici la liste des modes contenus dans le fichier `modes.mf` ainsi que la résolution correspondante. Tous les noms de mode sont en principe en lettres minuscules et ne comportent pas plus de huit caractères. Ils décrivent principalement des imprimantes de toutes marques mais on verra que quelques moniteurs ont aussi un mode répertorié.

Signalons aussi l'existence d'un mode intitulé **nullmode** qui a seulement pour effet de fabriquer des fichiers `.tfm`: il correspond à une résolution de 101 dpi et fixe *proofing* = -1 et *fontmaking* = 1.

On a déjà vu que les modes ne se contentent pas de préciser les valeurs des résolutions. Ils donnent les valeurs les plus appropriées aux variables *blacker*, *fillin* et *o\_correction*. Ce sont des valeurs qui rendent compte des caractéristiques des matériels utilisés. Citons à titre d'exemple les définitions complètes des modes `cx` et `canonex` qui sont très répandus :

```
mode_def cx =
mode_param (pixels_per_inch, 300);
mode_param (blacker, 0);
mode_param (fillin, .2);
mode_param (o_correction, .6);
mode_common_setup_;
enddef;

mode_def canonex =
mode_param (pixels_per_inch, 600);
mode_param (blacker, .2);
mode_param (fillin, .1);
mode_param (o_correction, .85);
mode_common_setup_;
enddef;
```

TAB. G.1 – *Noms de modes associés à des périphériques*

|                             |               |          |
|-----------------------------|---------------|----------|
| AGFA 400PS                  | 406 dpi       | agfafzz  |
| AGFA P3400PS                | 400 dpi       | agfatfzz |
| Alphatype CRS               | 5333 dpi      | crs      |
| Apple ImageWriter           | 144 dpi       | iw       |
| Apple LaserWriter Pro 630   | 600 dpi       | canonex  |
| Apple LaserWriterPro 810    | 800 dpi       | lwpro    |
| Apple StyleWriter           | 360 dpi       | stylewr  |
| Atari previewer             | 95 dpi        | atarinf  |
| Atari previewer             | 96 dpi        | atarins  |
| Atari ST SLM 804 printer    | 300 dpi       | atariezf |
| Atari ST SM 124 screen      | 101 dpi       | atariotf |
| Autologic APS-Micro5        | 723 dpi       | aps      |
| Autologic APS-Micro6        | 1016 dpi      | apssixhi |
| BBN Bitgraph                | 118 dpi       | bitgraph |
| Canon BJC-600               | 360 dpi       | canonbjc |
| Canon BubbleJet 10ex        | 360 dpi       | bjtenex  |
| Canon CX, SX, LBP-LX        | 300 dpi       | cx       |
| Chelgraph IBX               | 9600 dpi      | ibx      |
| CItoh 310                   | 240x144 dpi   | itohtoz  |
| CItoh 310 landscape         | 144x240 dpi   | itohtozl |
| CItoh 8510A                 | 160x144 dpi   | itoh     |
| CItoh 8510A landscape       | 144x160 dpi   | itolh    |
| Commodore Amiga             | 100 dpi       | amiga    |
| Compugraphic 8600           | 1301x1569 dpi | cg       |
| Compugraphic 8600 landscape | 1569x1302 dpi | cgl      |
| Compugraphic 9600           | 1200 dpi      | cgnzz    |
| DataDisc                    | 70 dpi        | datadisc |
| DataDisc                    | 70x93 dpi     | newdd    |
| DEC 17-inch, 1024 x 768     | 82 dpi        | decsmall |
| DEC 19-inch, 1280 x 1024    | 100 dpi       | declarge |
| DEC LA75                    | 144 dpi       | lasf     |
| DEC LN01                    | 300 dpi       | lnzo     |
| DEC LN03                    | 300 dpi       | ricohlp  |
| DEC LN03R Scriptprinter     | 300 dpi       | lnotr    |
| DEC lps20                   | 300 dpi       | lpstz    |
| EightThree                  | 83 dpi        | eighthre |
| Epson                       | 120x72 dpi    | epsdrft  |
| Epson                       | 72x120 dpi    | epsdrftl |
| Epson                       | 60x72 dpi     | epsfast  |
| Epson                       | 72x60 dpi     | epsfastl |
| Epson                       | 120x216 dpi   | epsonlo  |
| Epson 9-pin MX/FX           | 240x216 dpi   | epson    |
| Epson 9-pin MX/FX landscape | 216x240 dpi   | epsonl   |

TAB. G.1 – *Noms de modes associés à des périphériques*

|                           |             |          |
|---------------------------|-------------|----------|
| Epson Action Laser 1500   | 300 dpi     | epsonact |
| Epson landscape           | 216x120 dpi | epsonlol |
| Epson LQ-500              | 180 dpi     | lqlores  |
| Epson LQ-500              | 360x180 dpi | lqmed    |
| Epson LQ-500 landscape    | 180x360 dpi | lqmedl   |
| Epson SQ 870              | 360 dpi     | epsonsq  |
| Epson Stylus              | 360 dpi     | epstylus |
| Epson Stylus Pro          | 360 dpi     | epstypro |
| Epson Stylus Pro          | 180 dpi     | epstypro |
| Epson Stylus Pro          | 720x360 dpi | epstypmd |
| Epson Stylus Pro          | 720 dpi     | esphi    |
| FourFour                  | 44 dpi      | fourfour |
| G3fax                     | 204x196 dpi | gtfax    |
| G3fax                     | 204x98 dpi  | gtfaxlo  |
| G3fax                     | 200 dpi     | highfax  |
| G3fax landscape           | 196x204 dpi | gtfaxl   |
| G3fax landscape           | 98x204 dpi  | gtfaxlol |
| HP 2680A                  | 180 dpi     | boise    |
| HP DeskJet 500            | 300 dpi     | deskjet  |
| HP Laser Jet IIISi        | 300 dpi     | jetiisi  |
| HP LaserJet               | 150 dpi     | ljlo     |
| HP LaserJet 4             | 600 dpi     | ljfour   |
| HP LaserJet 5             | 600 dpi     | ljfive   |
| HP LaserJet 5MP           | 600 dpi     | ljfivemp |
| HP RuggedWriter 480       | 180 dpi     | hprugged |
| IBM 3179 screen           | 87x65 dpi   | ibmtosn  |
| IBM 3179 screen landscape | 65x87 dpi   | ibmtosnl |
| IBM 3193 screen           | 100 dpi     | ibmtont  |
| IBM 3812                  | 240 dpi     | ibmteot  |
| IBM 3820                  | 240 dpi     | ibmtetz  |
| IBM 38xx                  | 240 dpi     | ibm_a    |
| IBM 38xx                  | 240 dpi     | ibmd     |
| IBM 4019                  | 300 dpi     | ibmfzon  |
| IBM 4029-30, 4250         | 600 dpi     | ibmfztn  |
| IBM 4216                  | 300 dpi     | ricoha   |
| IBM 6154 display          | 118 dpi     | ibmsoff  |
| IBM 6670 Sherpa           | 240 dpi     | sherpa   |
| IBM EGA monitor           | 96x81 dpi   | ibmega   |
| IBM EGA monitor landscape | 81x96 dpi   | ibmegal  |
| IBM Lexmark Optra R 4049  | 1200 dpi    | lexmarkr |
| IBM ProPrinter            | 240x216 dpi | ibmpp    |
| IBM ProPrinter            | 216x240 dpi | ibmppl   |
| IBM VGA monitor           | 110 dpi     | ibmvga   |

TAB. G.1 – *Noms de modes associés à des périphériques*

|                              |             |          |
|------------------------------|-------------|----------|
| LaserMaster                  | 1000 dpi    | lmaster  |
| Linotronic L-300 with RIP-50 | 3386 dpi    | linolttz |
| Linotronic [13]00            | 1270 dpi    | linoone  |
| Linotype Linotronic 300      | 2540 dpi    | linotzzh |
| Linotype Linotronic [13]00   | 635 dpi     | linolo   |
| Mac screen                   | 72 dpi      | mactrue  |
| Mac screens at magstep 1     | 86 dpi      | macmag   |
| NCD 19-inch                  | 95 dpi      | ncd      |
| NEC                          | 180 dpi     | nec      |
| NEC PC-PR201 series          | 160 dpi     | nectzo   |
| NEC PC-PR406LM               | 320 dpi     | neclm    |
| NEC-P6                       | 360 dpi     | nechi    |
| NeXT monitor                 | 100 dpi     | nextscrn |
| NeXT Newgen                  | 400 dpi     | nexthi   |
| NineOne                      | 91x91       | nineone  |
| OCE 6750-PS                  | 508 dpi     | ocessfz  |
| Okidata                      | 240x288 dpi | okidata  |
| Okidata 410e in 600 dpi mode | 600 dpi     | okifte   |
| Okidata landscape            | 288x240 dpi | okidatal |
| OneTwoZero                   | 120/120     | onetz    |
| PC screen preview            | 118 dpi     | pcprevw  |
| Printware 720IQ              | 1200 dpi    | prntware |
| QMS 1700                     | 600 dpi     | qmsoszz  |
| QMS 1725                     | 600 dpi     | qmsostf  |
| QMS 2425                     | 1200 dpi    | qmstftf  |
| QMS Xerox engine             | 300 dpi     | qms      |
| Ricoh sp10ps/lp7200-ux       | 600 dpi     | ricohsp  |
| Sigma L-View monitor         | 118x109 dpi | lview    |
| Star NL-10                   | 240x216 dpi | starnlt  |
| Star NL-10 landscape         | 216x240 dpi | starnltl |
| Sun and BBN Bitgraph         | 85 dpi      | sun      |
| Sun SPARCprinter             | 400 dpi     | sparcptr |
| Suns high-resolution         | 118 dpi     | pcscreen |
| Symbolics LGP-10             | 240 dpi     | canonlbp |
| Tektronix Phaser PXi         | 300 dpi     | phaser   |
| TI Omnilaser                 | 300 dpi     | ricoh    |
| Toshiba 13XX, EpsonLQ        | 180 dpi     | toshiba  |
| Ultre*setter                 | 2400 dpi    | supre    |
| Ultre*setter                 | 1200 dpi    | ultre    |
| Variotype 5060W, APS 6       | 600 dpi     | vtfzszw  |
| Variyper 4200 B-P            | 1800 dpi    | vtftzz   |
| Variyper 4300P               | 2400 dpi    | vtftzzhi |
| Variyper 4300P               | 1200 dpi    | vtftzzlo |

TAB. G.1 – *Noms de modes associés à des périphériques*

|                           |         |          |
|---------------------------|---------|----------|
| Varityper Laser 600       | 600 dpi | vtszz    |
| VAXstation monitor        | 78 dpi  | vs       |
| Xerox 3700                | 300 dpi | xrxtszz  |
| Xerox 4050/4075/4090/4700 | 300 dpi | xrxfzfz  |
| Xerox 8790 or 4045        | 600 dpi | docutech |
| Xerox 8790 or 4045        | 300 dpi | xrxesnz  |
| Xerox 9700                | 300 dpi | xrxnszz  |
| Xerox Dover               | 384 dpi | dover    |

D'autres noms de modes sont reconnus par METAFONT qui correspondent à des dénominations plus anciennes. Ils comportent des lettres majuscules et minuscules et sont maintenus uniquement pour la compatibilité. On en trouvera la liste dans le tableau suivant, qui donne les équivalences de noms :

TAB. G.2 – *Anciens noms de modes*

|                              |          |
|------------------------------|----------|
| AgfaFourZeroZero             | agfafzz  |
| AgfaThreeFourZeroZero        | agfatfzz |
| APSSixMed                    | vtfzszw  |
| aselect                      | ljfour   |
| AtariNineFive                | atarinf  |
| AtariNineSix                 | atarins  |
| AtariSLMEightZeroFour        | atariezf |
| AtariSMOneTwoFour            | atariotf |
| CanonBJCSixZeroZero          | canonbjc |
| CanonCX                      | cx       |
| CanonEX                      | canonex  |
| CanonLBPLX                   | cx       |
| CanonLBPTen                  | canonlbp |
| CanonSX                      | cx       |
| ChelgraphIBX                 | ibx      |
| CItohEightFiveOneZero        | itooh    |
| CItohThreeOneZero            | itohtoz  |
| citohtoz                     | itohtoz  |
| CompugraphicEightSixZeroZero | cg       |
| CompugraphicNineSixZeroZero  | cgnsszz  |
| corona                       | cx       |
| cthreeten                    | itohtoz  |
| DataDiscNew                  | newdd    |
| DD                           | datadisc |
| DEClarge                     | declarge |
| DECsmall                     | decsmall |
| dp                           | cx       |
| EightThree                   | eighthre |

TAB. G.2 – Anciens noms de modes

|                        |                          |
|------------------------|--------------------------|
| elvira                 | declarege                |
| epsdraft               | epsdrft                  |
| epshi                  | epson                    |
| epslo                  | epsonlo                  |
| EpsonAction            | epsonact                 |
| epsonfx                | epson                    |
| epsonlq                | toshiba                  |
| EpsonLQFiveZeroZeroLo  | lqllores                 |
| EpsonLQFiveZeroZeroMed | lqmed                    |
| EpsonMXFX              | epson                    |
| EpsonSQEightSevenZero  | epsonsq                  |
| EpsonStylusPro         | epstypro                 |
| EpsonStylusProHigh     | esphi                    |
| EpsonStylusProLow      | epstypl                  |
| EpsonStylusProMed      | epstypmd                 |
| epstylwr               | stylewri                 |
| FourFour               | fourfour                 |
| gpx                    | vs                       |
| gtfaxhi                | voir GThreefax           |
| GThreefax              | gtfax                    |
| hifax                  | highfax                  |
| HPDeskJet              | deskjet                  |
| hplaser                | cx                       |
| HPLaserJetIIISi        | jetiiisi                 |
| IBMFourTwoFiveZero     | ibmfztn                  |
| IBMFourTwoOneSix       | ricoha                   |
| IBMFourTwoThreeZero    | ibmfztn                  |
| IBMFourZeroOneNine     | ibmfzon                  |
| IBMFourZeroTwoNine     | ibmfztn                  |
| IBMProPrinter          | ibmpp                    |
| IBMSixOneFiveFour      | ibmsoff                  |
| IBMSixSixSevenZero     | sherpa                   |
| IBMUlfHolleberg        | voir IBMThreeEightOneTwo |
| IBMThreeEightOneTwo    | ibmteot                  |
| IBMThreeEightTwoZero   | ibmtetz                  |
| IBMThreeOneNineThree   | ibmtont                  |
| IBMThreeOneSevenNine   | ibmtosn                  |
| imagen                 | cx                       |
| imagewriter            | lw                       |
| kyocera                | cx                       |
| laserjet               | cx                       |
| laserjetfive           | ljfive                   |
| laserjetfivemp         | ljfivemp                 |

TAB. G.2 – *Anciens noms de modes*

|                         |                            |
|-------------------------|----------------------------|
| laserjetfour            | ljfour                     |
| laserjethi              | cx                         |
| laserjetlo              | ljlo                       |
| lasermaster             | lmaster                    |
| laserwriter             | cx                         |
| LASevenFive             | lasf                       |
| LexmarkOptraR           | lexmarkr                   |
| linohalf                | voir LinotypeOneZeroZeroLo |
| linohi                  | voir LinotypeOneZeroZero   |
| linosuper               | linotzzh                   |
| linothree               | linotzzh                   |
| linothreelo             | voir LinotypeOneZeroZero   |
| LinotypeLThreeThreeZero | linoltz                    |
| LinotypeOneZeroZero     | linoone                    |
| LinotypeOneZeroZeroLo   | linolo                     |
| LinotypeThreeZeroZeroHi | linotzzh                   |
| LNOthree                | ricohlp                    |
| LNOthreR                | lnotr                      |
| LNZeroOne               | lnzo                       |
| LNZeroThree             | ricohlp                    |
| lps                     | lnzo                       |
| LPSFourZero             | lnzo                       |
| LPSTwoZero              | lpstz                      |
| lqhiress                | nechi                      |
| lqmedres                | lqmed                      |
| MacTrueSize             | mactrue                    |
| NecTwoZeroOne           | nectzo                     |
| Newgen                  | nexthi                     |
| NeXTprinter             | nexthi                     |
| NeXTscreen              | nextscrn                   |
| nextscreen              | nextscrn                   |
| NineOne                 | nineone                    |
| OCESixSevenFiveZeroPS   | ocessfz                    |
| okifourten              | okifte                     |
| okihi                   | okidata                    |
| OneTwoZero              | onetz                      |
| OneZeroZero             | amiga                      |
| onezz                   | amiga                      |
| PrintwareSevenTwoZeroIQ | prntware                   |
| Prism                   | ultre                      |
| proprinter              | voir IBMProPrinter         |
| qmsesz                  | ljfour                     |
| QMSOneSevenTwoFive      | qmsostf                    |

TAB. G.2 – Anciens noms de modes

|                              |          |
|------------------------------|----------|
| QMSOneSevenZeroZero          | qmsoszz  |
| QMSTwoFourTwoFive            | qmstftf  |
| RicohA                       | ricoha   |
| RicohFortyEighty             | ricoh    |
| RicohFourZeroEightZero       | ricoh    |
| RicohLP                      | ricohlp  |
| SparcPrinter                 | sparcptr |
| StarNLOneZero                | starnlt  |
| stylewriter                  | stylewri |
| varityper                    | vtszz    |
| VarityperFiveZeroSixZeroW    | vtfzszw  |
| VarityperFourThreeZeroZeroHi | vtftzzhi |
| VarityperFourThreeZeroZeroLo | vtftzzlo |
| VarityperFourTwoZeroZero     | vtftzz   |
| VarityperSixZeroZero         | vtszz    |
| VAXstation                   | vs       |
| VTSix                        | vtszz    |
| XeroxDocutech                | docutech |
| XeroxEightSevenNineZero      | xrxesnz  |
| XeroxFourZeroFiveZero        | xrxzfz   |
| XeroxNineSevenZeroZero       | xrxnszz  |
| XeroxThreeSevenZeroZero      | xrxtszz  |

En conclusion de ce chapitre, il faut encore citer l'existence d'un mode **smoke**. Il s'apparente au mode *proof* mais en diffère par les aspects suivants : les caractères seront beaucoup plus noirs que dans le mode *proof* (ils font appel à une police intitulée **black** au lieu de la police *gray*) et les boîtes qui entourent les caractères sont seulement matérialisées par des traits fins aux quatre coins (comme avec l'instruction **gfcorners**). L'utilité de ce mode est de faire des épreuves à épingle au mur, les coins servant à raccorder correctement entre eux les caractères pour pouvoir juger également des espacements. C'est un mode qui fait partie intégrante du format *plain*. En voici la définition :

```
mode_def smoke =
 proof_;
 proofing:=1;
 extra_setup:=extra_setup&"grayfont black";
 let makebox=maketicks;
 enddef;
```

Le fait que *proofing* vaut 1 signifie que l'on fait bien des épreuves (résolution de 36 pixels par point) et non pas une police définitive mais que les points de construction ne seront pas matérialisés (l'instruction **label** ne fonctionne que pour *proofing*  $\geq 2$ ).

## Annexe H

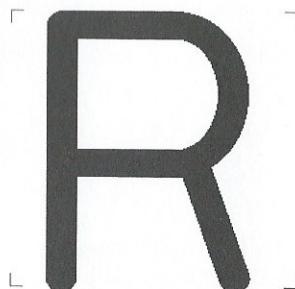


FIG. G.1 – Une lettre en mode *smoke*

## METAFONT ?

### A.1. Distributions

La famille METAFONT<sup>1</sup> dans toute distribution complète de TeX. Il existe un des nombreux sites de téléchargement de logiciels qui fournit programmes qui l'accompagnent, paramètres, guides d'emploi et pertinents. Par exemple dans OpenGroupWare (<http://archives.kermark.org>) sur lesquelles on trouve également de très nombreux fichiers sources. Les sites GRAN sont accessibles de diverses manières avec un moteur de recherche.

#### 1. en Allemagne

Le site de l'Université de Bielefeld<sup>2</sup> propose dans le répertoire `tex/extra/tex/bm` plusieurs autres archives supplémentaires.

Il existe également d'autres sites de téléchargement de logiciels, par exemple <http://www.math.tu-berlin.de/~koch/tex/>, mais il faut faire attention pour choisir. Beaucoup d'installations de logiciels peuvent être dangereuses. Il est préférable de télécharger des logiciels de sites officiels.

#### 2. en Angleterre

Le site de la société CambridgeSoft<sup>3</sup> dans le répertoire `tex/extra/tex/bm` offre plusieurs logiciels et documents de programmation.

Il existe également d'autres sites de téléchargement de logiciels, par exemple <http://www.maths.warwick.ac.uk/~vassilis/tex/>.

TABLEAU H.1. Fréquence d'utilisation par pays

| Pays       | Français | Anglais | Allemand | Autre |
|------------|----------|---------|----------|-------|
| France     | 100 %    | 0 %     | 0 %      | 0 %   |
| Angleterre | 0 %      | 100 %   | 0 %      | 0 %   |
| Allemagne  | 0 %      | 0 %     | 100 %    | 0 %   |
| Autres     | 0 %      | 0 %     | 0 %      | 0 %   |

## Annexe H

# Comment se procurer METAFONT ?

### H.1 Distributions

On trouve METAFONT dans toute distribution complète de TEX. Il en est un des éléments obligés, de même que les autres programmes qui l'accompagnent normalement : gftodvi et gftopk en particulier. Par voie électronique, il existe des archives officielles intitulées CTAN (*Comprehensive TEX Archive Network*) sur lesquelles on trouvera également de très nombreux fichiers sources. Les sites CTAN sont accessibles de diverses manières aux adresses suivantes :

#### 1. en Allemagne

- par ftp anonyme: [ftp.dante.de](ftp://ftp.dante.de) dans le répertoire /tex-archive (ou bien /pub/tex ou bien encore /pub/archive)
- par gopher à l'adresse [gopher.dante.de](gopher://gopher.dante.de)
- sur le Web, accès à l'adresse <http://www.dante.de/>
- pour tout contact par email: [ftpmail@dante.de](mailto:ftpmail@dante.de)
- administrateur: [ftpmaint@dante.de](mailto:ftpmaint@dante.de)

#### 2. en Angleterre

- par ftp anonyme: [ftp.tex.ac.uk](ftp://ftp.tex.ac.uk) dans le répertoire /tex-archive (ou bien /pub/tex ou bien encore /pub/archive)
- par gopher à l'adresse [gopher.tex.ac.uk](gopher://gopher.tex.ac.uk)
- sur le Web, accès à l'adresse <http://www.tex.ac.uk/tex-archive>
- administrateur: [ctan-uk@tex.ac.uk](mailto:ctan-uk@tex.ac.uk)

### 3. aux USA

- par ftp anonyme: ctan.tug.org dans le répertoire /tex-archive (ou bien /pub/archive)
- sur le Web, accès à l'adresse http://ctan.tug.org/ctan/
- administrateur: ftpmaint@mail.tug.org

En plus de ces trois sites, il existe, de par le monde, un certain nombre de sites miroirs qui reprennent plus ou moins complètement le contenu des CTAN. Le tableau H.1 en donne une liste classée par ordre alphabétique de pays.

TAB. H.1 – Principaux sites ftp classés par pays

| Pays       | Adresse ftp          | Répertoire                        |
|------------|----------------------|-----------------------------------|
| Allemagne  | ftp.gwdg.de          | /pub/dante                        |
| Allemagne  | ftp.mpi-sb.mpg.de    | /pub/tex/mirror/ftp.dante.de      |
| Allemagne  | ftp.tu-chemnitz.de   | /pub/tex                          |
| Allemagne  | ftp.uni-augsburg.de  | /pub/tex/ctan                     |
| Allemagne  | ftp.uni-stuttgart.de | /tex-archive                      |
| Angleterre | src.doc.ic.ac.uk     | /packages/tex/uk-tex              |
| Australie  | ftp.cs.rmit.edu.au   | /tex-archive                      |
| Australie  | ctan.unsw.edu.au     | /tex-archive                      |
| Autriche   | ftp.univie.ac.at     | /packages/tex                     |
| Belgique   | ftp.belnet.be        | /packages/TeX                     |
| Canada     | ctan.math.mun.ca     | /tex-archive                      |
| Corée      | ftp.kreonet.re.kr    | /pub/CTAN                         |
| Danemark   | ftp.net.uni-c.dk     | /mirror/ftp.tex.ac.uk/tex-archive |
| Danemark   | sunsite.auc.dk       | /pub/tex/ctan                     |
| Espagne    | ftp.rediris.es       | /mirror/tex-archive               |
| Estonie    | ftp.ut.ee            | /tex-archive                      |
| Finlande   | ftp.funet.fi         | /pub/TeX/CTAN                     |
| France     | ftp.jussieu.fr       | /pub4/TeX/CTAN                    |
| France     | ftp.loria.fr         | /pub/ctan                         |
| France     | ftp.oleane.net       | /pub/mirrors/CTAN/                |
| Grece      | ftp.ntua.gr          | /mirror/ctan                      |
| Hong Kong  | ftp.comp.hkbu.edu.hk | /pub/TeX/CTAN                     |
| Irlande    | ftp.heanet.ie        | /pub/ctan/tex                     |
| Italie     | cis.utovrm.it        | /TeX                              |
| Italie     | ftp.unina.it         | /pub/TeX                          |
| Japon      | ftp.lab.kdd.co.jp    | /CTAN                             |
| Japon      | ftp.riken.go.jp      | /pub/tex-archive                  |
| Japon      | ftp.u-aizu.ac.jp     | /pub/tex/CTAN                     |
| Pays-Bas   | ftp.cs.ruu.nl        | /pub/tex-archive                  |
| Pologne    | sunsite.icm.edu.pl   | /pub/CTAN                         |

TAB. H.1 – Principaux sites ftp classés par pays

|                      |                         |                      |
|----------------------|-------------------------|----------------------|
| République Tchèque   | ftp.cstug.cz            | /pub/tex/CTAN        |
| Russie               | tex.ihep.su             | /pub/TeX/CTAN        |
| Singapour            | ftpserver.nus.sg        | /pub/zi/TeX          |
| Suède                | ftp.nada.kth.se         | /pub/tex/ctan-mirror |
| Suisse               | sunsite.cnlab-switch.ch | /mirror/tex          |
| Taiwan               | dongpo.math.ncu.edu.tw  | /tex-archive         |
| Taiwan               | ftp.ccu.edu.tw          | /pub/tex             |
| USA Côte Ouest       | ftp.cdrom.com           | /pub/tex/cтан        |
| USA Missouri         | wuarchive.wustl.edu     | /packages/TeX        |
| USA New York         | ftp.rge.com             | /pub/tex             |
| USA Caroline du nord | sunsite.unc.edu         | /pub/tex             |

Le tableau H.2 fournit une liste complémentaire de sites qui sont des miroirs seulement partiels.

TAB. H.2 – Quelques sites miroirs partiels

| Pays         | Adresse ftp          | Répertoire               |
|--------------|----------------------|--------------------------|
| Allemagne    | ftp.germany.eu.net   | /pub/packages/TeX        |
| Australie    | ftp.adfa.oz.au       | /pub/tex/cтан            |
| Chili        | ftp.dcc.uchile.cl    | /pub/Mirror/cтан         |
| Corée        | sunsite.snu.ac.kr    | /shortcut/CTAN           |
| Italie       | sunsite.dsi.unimi.it | /pub/TeX                 |
| Japon        | ftp.jaist.ac.jp      | /pub/TeX/tex-archive     |
| Pologne      | ftp.gust.org.pl      | /pub/TeX                 |
| Taiwan       | ftp.fcu.edu.tw       | /pub2/tex                |
| USA Virginie | ftp.uu.net           | /pub/text-processing/TeX |

## H.2 Utilisateurs francophones

Enfin, pour les utilisateurs francophones, il faut citer tout particulièrement l'association GUTenberg qui est le groupe français des utilisateurs de TeX et qui fournit à ses adhérents des distributions francisées de TeX et METAFONT pour des environnements très divers (UNIX, LINUX, MAS-OS, MS-DOS, WINDOWS).

Association GUTenberg

c/o Irisa, Campus universitaire de Beaulieu

35042 Rennes Cedex — France

e-mail : tresorerie.gutenberg@ens.fr et gut@irisa.fr

Tél. et fax : 33 (0)1 30 87 06 25

<http://www.ens.fr/gut/>

L'association distribue aussi « Les Cahiers GUTenberg » dont le site WEB est :

<http://www.univ-rennes1.fr/pub/GUTenberg/publications/index.html>

### H.3 Liste de diffusion

Il existe une liste de diffusion et de discussion entièrement consacrée à METAFONT. On peut s'y inscrire en envoyant un courrier à `listserv@ens.fr` tandis que les messages sont adressés à l'adresse `metafont@ens.fr`

On discute sur cette liste aussi bien de questions liées à METAFONT que du programme Metapost qui est similaire à METAFONT pour ce qui est de la syntaxe mais qui produit du PostScript. C'est une liste qui se veut internationale et sur laquelle l'anglais semble plus utilisé que le français à la différence de la liste de GUTenberg mentionnée plus haut, consacrée à TeX et, elle aussi, hébergée par l'E.N.S. (École Normale Supérieure).

éditions de l'Institut national des sciences appliquées de Paris, 1987.

Griffiths, M.-A. et autres. *Le guide du métamorphisme*. Paris, 1987.

Hallé, J. et al. *Le guide du métamorphisme*. Paris, 1987.

Le Gall, J. et al. *Le guide du métamorphisme*. Paris, 1987.

## Annexe I

# Bibliographie

### 1. Voici les trois ouvrages fondateurs de METAFONT :

- The METAFONTbook — volume C de *Computers and Typesetting*.  
Donald E. Knuth. Addison-Wesley 1984.  
ISBN 0-201-13445-4.
- METAFONT the program — volume D de *Computers and Typesetting*.  
Donald E. Knuth. Addison-Wesley 1984.  
ISBN 0-201-13438-1.
- Computer modern typefaces — volume E de *Computers and Typesetting*.  
Donald E. Knuth. Addison-Wesley 1984.  
ISBN 0-201-13446-2.

Cet ouvrage comporte les programmes exhaustifs de tous les caractères de la famille Computer Roman.

### 2. Autres ouvrages :

- L<sup>A</sup>T<sub>E</sub>X , Band 2, Ergänzungen, mit einer Einführung in METAFONT.  
Helmut Kopka. Addison-Wesley 1995, Bonn.
- T<sub>E</sub>X and METAFONT, New Directions in Typesetting.  
D. E. Knuth. The American Mathematical Society and Digital Press, Bedford, MA, 1979.
- METAFONT for Beginners. Geoffrey Tobin. Document électronique.  
juillet 1994. Disponible sur les archives CTAN.

### 3. Articles :

- Fonts for Digital Halftones. Donald E. Knuth. TUGboat, 8(2), p. 135–160, juillet 1987.
- The Future of T<sub>E</sub>X and METAFONT. Donald E. Knuth. TUGboat, 11(4), p. 489, nov. 1990.

- When  $\text{\TeX}$  and METAFONT Work Together. Alan Hoenig. Proceedings of the 7th European  $\text{\TeX}$  Conference, Prague, p. 1–19, sept. 1992.
- Packed (pk) Font File Format. Tomas Rokicki. TUGboat, 6(3), p. 115–120, sept. 1985.
- Outline fonts with METAFONT. Doug Henderson. TUGboat 10(1), p. 36–38, avril 1989.
- Typesetting old german : Fraktur, Schwabacher, Gotisch and initials. Yannis Haralambous. TUGboat, 12(1), p. 129–138, mars 1991.
- Opening Pandora's Box. Neenie Billawala. TUGboat, 10(4), p. 481–489, déc. 1989.
- MFtool: A description of a METAFONT script-driven processing facility. John Crawford. TUGboat 8(2), p. 131–131, juillet 1987.
- Update: METAFONT mode\_def settings for  $\text{\TeX}$  output devices. Barbara Beeton. TUGboat 8(2), p. 132–134, juillet 1987.
- Update: METAFONT mode\_def settings for  $\text{\TeX}$  output devices. Doug Henderson. TUGboat 8(3), p. 268–270, novembre 1987.
- TANGLE modification causes problems in METAFONT and PK files. Thomas J. Reid. TUGboat 8(3), p. 264–265, novembre 1987.
- Write-white printing engines and tuning fonts with METAFONT. Neenie Billawala. TUGboat 8(1), p. 29–32, avril 1987.
- METAFONT mode\_def settings for various  $\text{\TeX}$  output devices. Barbara Beeton. TUGboat 8(1), p. 33–33, avril 1987.
- $\text{\TeX}82$  and METAFONT84 for the HP1000 A-series. Tor Lillqvist. TUGboat 8(1), p. 47–48, avril 1987.
- The exotic Croatian Glagolitic alphabet. Darko Žubrinić. TUGboat 13(4), p. 470–471, décembre 1992.
- A typewriter font for the Macintosh 8-bit font table. Yannis Haralambous. TUGboat 13(4), p. 476–477, décembre 1992.
- When  $\text{\TeX}$  and METAFONT talk: Typesetting on curved paths and other special effects. Alan Hoenig. TUGboat 12(3), p. 554–557, novembre 1991.
- Partial font embedding utilities for PostScript Type-1 fonts. Basil Malyshev — Michel Goossens. TUGboat 16(1), p. 69–77, mars 1995.
- Public-domain, documented implementations of  $\text{\TeX}$  and METAFONT for VAX/VMS. Brian Hamilton Kelly. TUGboat 12(1), p. 80–83, mars 1991.
- Update on Pascal METAFONT. Scott Kim. TUGboat 2(1), p. 10–10, février 1981.
- A Fortran version of METAFONT. Sao Khai Mong. TUGboat 3(2), p. 25, octobre 1982.
- A course on METAFONT programming. Donald E. Knuth. TUGboat 5(2), p. 105, novembre 1984.

- A Chinese meta-font. John D. Hobby — Gu Guoan. TUGboat 5(2), p. 119, novembre 1984.
- Meta-METAFONT: An exhibit at the Cooper Union, NYC. Alan Hoenig TUGboat 7(2), p. 102–102, juin 1986.
- METAFONT,  $\text{\TeX}$  and the Humanities, Rennes, France. TUGboat 7(3), p. 191–191, octobre 1986.
- Some empirical observations on METAFONT design. Georgia K.M. Tobin. TUGboat 8(1), p. 26–28, avril 1987.
- Updated Computer Modern fonts for the LN03. John Sauter. TUGboat 8(2), p. 129–130, juillet 1987.
- New version(s) of  $\text{\TeX}$  and METAFONT. Barbara Beeton. TUGboat 9(2), p. 121–122, août 1988.
- TurboMETAFONT: A new port in C for UNIX and MS-DOS. Richard Kinch. TUGboat 10(1), p. 23–24, avril 1989.
- The  $\text{\TeX}$  PostScript software package. Stephan v. Bechtolsheim. TUGboat 10(1), p. 25–27, avril 1989.
- A handy little font. Georgia K.M. Tobin. TUGboat 10(1), p. 28–30, avril 1989.
- Typesetting *Concrete Mathematics*. Donald Knuth. TUGboat 10(1), p. 31–36, avril 1989.
- Font news. Dominik Wujastyk. TUGboat 10(1), p. 39–39, avril 1989.
- Another dingbat idea. Georgia K.M. Tobin. TUGboat 10(2), p. 166–169, juillet 1989.
- Chess printing via METAFONT and  $\text{\TeX}$ . Zalman Rubinstein. TUGboat 10(2), p. 170–172, juillet 1989.
- Chess printing via METAFONT and  $\text{\TeX}$ . Erratum. TUGboat 10(3), p. 359–359, novembre 1989.
- Guidelines for creating portable METAFONT code. Don Hosek. TUGboat 10(2), p. 173–176, juillet 1989.
- The new versions of  $\text{\TeX}$  and METAFONT. Donald E. Knuth. TUGboat 10(3), p. 325–328, novembre 1989.
- The new versions of  $\text{\TeX}$  and METAFONT. Erratum. TUGboat 11(1), p. 12–12, avril 1990.
- PublC METAFONT available. Klaus Thull. TUGboat 10(3), p. 328–328, novembre 1989.
- A chess font for  $\text{\TeX}$ . Jan Eric Larsson. TUGboat 10(3), p. 351–351, novembre 1989.
- Typesetting modern Greek with 128 character codes. Yannis Haralambous — Klaus Thull. TUGboat 10(3), p. 354–359, novembre 1989.

- Introduction to METAFONT. Doug Henderson. *TUGboat* 10(4), p. 467–479, décembre 1989.
- Fractal images with TeX. Alan Hoenig. *TUGboat* 10(4), p. 491–498, décembre 1989.
- Design of Oriental characters with METAFONT. Don Hosek. *TUGboat* 10(4), p. 499–502, décembre 1989.
- Thai languages and METAFONT. Bob Batzinger. *TUGboat* 10(4), p. 503–503, décembre 1989.
- A METAFONT-like system with PostScript output. John D. Hobby. *TUGboat* 10(4), p. 505–512, décembre 1989.
- Migration from Computer Modern fonts to Times fonts. R.E. Youngen, W.B. Woolf — D.C. Lattner. *TUGboat* 10(4), p. 513–519, décembre 1989.
- Fine typesetting with TeX using native Autologic fonts. Arvin C. Conrad. *TUGboat* 10(4), p. 521–528, décembre 1989.
- Typesetting modern Greek: An update. Yannis Haralambous. *TUGboat* 11(1), p. 26–26, avril 1990.
- TeX 3.0 and METAFONT 2.0. Nelson Beebe. *TUGboat* 11(3), p. 442–443, septembre 1990.
- LATEX 2.10. Frank Mittelbach. *TUGboat* 11(3), p. 444, septembre 1990.
- Comments on the future of TeX and METAFONT. Nelson H. F. Beebe. *TUGboat* 11(4), p. 490–494, novembre 1990.
- Environment for translating METAFONT to PostScript. Shimon Yanai — Daniel M. Berry. *TUGboat* 11(4), p. 525–541, novembre 1990.
- An improved chess font. David Tofsted. *TUGboat* 11(4), p. 542–544, novembre 1990.
- Labelled diagrams in METAFONT. Alan Jeffrey. *TUGboat* 12(2), p. 227–229, juin 1991.
- When TeX and METAFONT talk: Typesetting on curved paths and other special effects. Alan Hoenig. *TUGboat* 12(3), p. 554–557, novembre 1991.
- Packing METAFONTs into PostScript. Toby Thain. *TUGboat* 13(1), p. 36–38, avril 1992.
- Modern Greek with adjunct fonts. C. Mylonas — R. Whitney. *TUGboat* 13(1), p. 39–50, avril 1992.
- Postnet codes using METAFONT. John Sauter. *TUGboat* 13(4), p. 472–476, décembre 1992.
- Icons for TeX and METAFONT. Donald E. Knuth. *TUGboat* 14(4), p. 387–389, décembre 1993.
- Automatic conversion of METAFONT fonts to Type1 PostScript. Basil Malyshov. *TUGboat* 15(3), p. 200–200, septembre 1994.

- Problems of the conversion of METAFONT fonts to PostScript Type 1. B. Malyshev. *TUGboat* 16(1), p. 60–68, mars 1995.
- Introducing Metapost. Alan Hoenig. *TUGboat* 16(1), p. 45, mars 1995.
- Some METAFONT techniques. Yannis Haralambous. *TUGboat* 16(1), p. 53, mars 1995.
- The program a2ac — Font handling on the PostScript level. Petr Olsufyev. *TUGboat* 16(1), p. 54–59, mars 1995.
- When METAFONT does it alone. Jiří Zlatuška. *TUGboat* 16(3), p. 227–232, septembre 1995.
- MetaFog: converting METAFONT shapes to contours. Richard J. Kinch. *TUGboat* 16(3), p. 233–243, septembre 1995.
- The Poetica family: fancy fonts with TeX and LATEX. Alan Hoenig. *TUGboat* 16(3), p. 244–252, septembre 1995.
- Dotted and dashed lines in METAFONT. Jeremy Gibbons. *TUGboat* 16(3), p. 259–264, septembre 1995.
- A METAFONT–EPS interface. Bogusław Jackowski. *TUGboat* 16(3), p. 333–340, septembre 1995.
- Graphics and TeX — A reappraisal of METAFONT/MetaPost/PostScript. Kees van der Laan. *TUGboat* 17(3), p. 269–279, septembre 1996.

— Beebe.

Yanai —

544, no-

227–229,

nd other  
e 1991.

), p. 36–

TUGboat

472–476,

, p. 387–

Basil Ma-

# Liste des tableaux

|      |                                                                        |     |
|------|------------------------------------------------------------------------|-----|
| 2.1  | <i>Valeurs comparées des paramètres dans les polices cmr et cmrfun</i> | 13  |
| 5.1  | <i>Largeur des caractères dans la police logocomplet</i>               | 121 |
| 6.1  | <i>Valeurs littérales des hiéroglyphes</i>                             | 170 |
| B.1  | <i>Tables de conversions en pouces et en millimètres</i>               | 180 |
| B.2  | <i>Nombre de pixels par millimètres à diverses résolutions</i>         | 180 |
| B.3  | <i>Les largeurs des caractères Computer Roman</i>                      | 181 |
| E.1  | <i>Les caractères de la police cmr et l'encodage OT1</i>               | 203 |
| E.2  | <i>Les caractères de la police ecr et l'encodage T1</i>                | 204 |
| E.3  | <i>Les caractères de la police wncyr et l'encodage OT2</i>             | 205 |
| E.4  | <i>Les lettrines de la police yinit</i>                                | 206 |
| E.5  | <i>Tableau des caractères de la police greek</i>                       | 207 |
| E.6  | <i>Tableau des caractères de la police ygoth</i>                       | 208 |
| E.7  | <i>Tableau des caractères de la police yswab</i>                       | 209 |
| E.8  | <i>Tableau des caractères de la police calligra</i>                    | 210 |
| E.9  | <i>Tableau des caractères de la police calligra (suite)</i>            | 211 |
| E.10 | <i>Tableau des caractères de la police astrosym</i>                    | 212 |
| E.11 | <i>Tableau des caractères de la police astrosym (suite)</i>            | 213 |
| E.12 | <i>Tableau des caractères de la police gray</i>                        | 214 |
| G.1  | <i>Noms de modes associés à des périphériques</i>                      | 232 |
| G.2  | <i>Anciens noms de modes</i>                                           | 235 |

|     |                                              |     |
|-----|----------------------------------------------|-----|
| H.1 | <i>Principaux sites ftp classés par pays</i> | 242 |
| H.2 | <i>Quelques sites miroirs partiels</i>       | 243 |

- 17.4.1 *Méthode style* . . . . .  
17.4.2 *Placement des sept points* . . . . .  
17.4.3 *Instruction reflectedabout* . . . . .  
17.4.4 *Instruction rotatedaround* . . . . .  
17.4.5 *Instruction subpath* . . . . .  
17.4.6 *Position de la pointe du stylo* . . . . .  
17.4.7 *Ajout d'un cycle* . . . . .  
17.4.8 *Instruction fill* . . . . .  
17.4.9 *Instruction unfill* . . . . .

# Liste des illustrations

|      |                                                                                            |    |
|------|--------------------------------------------------------------------------------------------|----|
| 1.1  | <i>La lettre grecque <math>\beta</math></i> . . . . .                                      | 5  |
| 3.1  | <i>Étalement d'un point</i> . . . . .                                                      | 25 |
| 3.2  | <i>Instruction reflectedabout</i> . . . . .                                                | 33 |
| 3.3  | <i>Instruction rotatedaround</i> . . . . .                                                 | 33 |
| 3.4  | <i>Points de construction d'une superellipse</i> . . . . .                                 | 35 |
| 3.5  | <i>Douze points intermédiaires sur une ellipse</i> . . . . .                               | 36 |
| 3.6  | <i>Instruction subpath</i> . . . . .                                                       | 36 |
| 3.7  | <i>Chemin de type flex</i> . . . . .                                                       | 37 |
| 3.8  | <i>Un pensquare incliné</i> . . . . .                                                      | 38 |
| 3.9  | <i>Instruction direction t of p</i> . . . . .                                              | 40 |
| 3.10 | <i>Position de la pointe du stylo</i> . . . . .                                            | 46 |
| 3.11 | <i>Lettre Z avec mag=.6</i> . . . . .                                                      | 49 |
| 3.12 | <i>Lettre Z avec magstep(2)</i> . . . . .                                                  | 49 |
| 3.13 | <i>Lettre Z avec magstep(4)</i> . . . . .                                                  | 50 |
| 3.14 | <i>Instruction makegrid</i> . . . . .                                                      | 60 |
| 4.1  | <i>Courbe de Bézier: points de construction et de contrôle</i> . . . . .                   | 66 |
| 4.2  | <i>Placement des sept points <math>z_1</math> à <math>z_7</math></i> . . . . .             | 67 |
| 4.3  | <i>Instruction draw <math>z_1 \dots z_4 \dots z_2 \dots z_5 \dots z_3</math></i> . . . . . | 68 |
| 4.4  | <i>Instruction draw <math>z_1--z_4--z_2--z_5--z_3</math></i> . . . . .                     | 68 |
| 4.5  | <i>Ajout d'un cycle</i> . . . . .                                                          | 69 |
| 4.6  | <i>Instruction fill</i> . . . . .                                                          | 70 |
| 4.7  | <i>Instruction unfill</i> . . . . .                                                        | 70 |

|      |                                                        |    |
|------|--------------------------------------------------------|----|
| 4.8  | <i>Tangentes verticales et horizontales</i>            | 71 |
| 4.9  | <i>Tangentes quelconques</i>                           | 71 |
| 4.10 | <i>Instruction controls</i>                            | 71 |
| 4.11 | <i>Courbe avec une tension de 1,4</i>                  | 72 |
| 4.12 | <i>Instruction -- (équivalente à tension infinity)</i> | 72 |
| 4.13 | <i>Instruction ---</i>                                 | 72 |
| 4.14 | <i>Instruction curl</i>                                | 73 |
| 4.15 | <i>Instruction filldraw</i>                            | 74 |
| 4.16 | <i>Taille de la pointe du stylo égale à 0,3 pt</i>     | 74 |
| 4.17 | <i>Dimension de pointe différente suivant x et y</i>   | 74 |
| 4.18 | <i>Instruction pensquare</i>                           | 75 |
| 4.19 | <i>Pointe de stylo hexagonale</i>                      | 75 |
| 4.20 | <i>Instruction quartercircle</i>                       | 76 |
| 4.21 | <i>Instruction halfcircle</i>                          | 76 |
| 4.22 | <i>Instruction fullcircle</i>                          | 76 |
| 4.23 | <i>Superellipse de paramètre 0,8</i>                   | 77 |
| 4.24 | <i>Superellipse de paramètre 0,7</i>                   | 77 |
| 4.25 | <i>Superellipse de paramètre 0,6</i>                   | 77 |
| 4.26 | <i>Superellipse de paramètre 0,5</i>                   | 78 |
| 4.27 | <i>Instruction flex</i>                                | 78 |
| 4.28 | <i>Instruction penpos avec draw</i>                    | 79 |
| 4.29 | <i>Instruction penpos avec filldraw</i>                | 79 |
| 4.30 | <i>Instruction penstroke</i>                           | 80 |
| 4.31 | <i>Instruction penrazor</i>                            | 80 |
| 4.32 | <i>Extrémités sans cutoff</i>                          | 81 |
| 4.33 | <i>Extrémités avec cutoff</i>                          | 81 |
| 4.34 | <i>Instruction cutdraw</i>                             | 81 |
| 4.35 | <i>Instruction point t of p</i>                        | 83 |
| 4.36 | <i>Instruction erase</i>                               | 83 |
| 4.37 | <i>Instruction drawdot</i>                             | 83 |
| 4.38 | <i>Inclinaison</i>                                     | 84 |
| 4.39 | <i>Rotation</i>                                        | 85 |

---

|                                                             |     |
|-------------------------------------------------------------|-----|
| 4.40 Symétrie droite . . . . .                              | 86  |
| 4.41 Instruction rotatedarround . . . . .                   | 86  |
| 4.42 Instruction cull . . . . .                             | 87  |
| 4.43 Résultat de cull keeping (1,1) . . . . .               | 88  |
| 4.44 Résultat de cull keeping (2,2) . . . . .               | 88  |
| 4.45 Résultat de cull keeping (3,3) . . . . .               | 89  |
| 4.46 Résultat de cull keeping (1,2) . . . . .               | 89  |
| 4.47 Effet de négatif par soustraction . . . . .            | 90  |
| 4.48 Addition et soustraction de pixels (1) . . . . .       | 91  |
| 4.49 Addition et soustraction de pixels (2) . . . . .       | 92  |
| 4.50 Instruction interpath . . . . .                        | 93  |
| 4.51 Instruction intersectionpoint . . . . .                | 94  |
| 4.52 Instruction directiontime . . . . .                    | 95  |
| 4.53 Instructions forever et exitif . . . . .               | 96  |
| 4.54 Découpage répétitif . . . . .                          | 97  |
| 4.55 Exemple de condition arithmétique . . . . .            | 97  |
| 4.56 La macro decalage avec un angle de 10 degrés . . . . . | 100 |
| 4.57 La macro decalage avec un angle de 20 degrés . . . . . | 100 |
| 4.58 La macro polygone . . . . .                            | 101 |
| 4.59 Contour aux arêtes incurvées . . . . .                 | 101 |
| 4.60 La macro polycreux avec un angle négatif . . . . .     | 103 |
| 4.61 La macro polycreux avec un angle positif . . . . .     | 103 |
| 4.62 Points décalés au hasard . . . . .                     | 107 |
| 4.63 Ligne ondulée par le hasard . . . . .                  | 107 |
| 5.1 Construction de la lettre E . . . . .                   | 117 |
| 5.2 Construction de la lettre F . . . . .                   | 118 |
| 5.3 Construction de la lettre M . . . . .                   | 119 |
| 5.4 Construction de la lettre N . . . . .                   | 120 |
| 5.5 Construction de la lettre A . . . . .                   | 122 |
| 5.6 Construction de la lettre O . . . . .                   | 123 |
| 5.7 Construction de la lettre T . . . . .                   | 124 |
| 5.8 Construction de la lettre S . . . . .                   | 124 |

|      |                                                     |     |
|------|-----------------------------------------------------|-----|
| 5.9  | <i>Construction de la lettre P . . . . .</i>        | 125 |
| 5.10 | <i>Construction de la lettre D . . . . .</i>        | 126 |
| 5.11 | <i>Construction de la lettre H . . . . .</i>        | 127 |
| 5.12 | <i>Construction de la lettre L . . . . .</i>        | 128 |
| 5.13 | <i>Construction de la lettre I . . . . .</i>        | 129 |
| 5.14 | <i>Construction de la lettre J . . . . .</i>        | 130 |
| 5.15 | <i>Construction de la lettre U . . . . .</i>        | 131 |
| 5.16 | <i>Construction de la lettre V . . . . .</i>        | 132 |
| 5.17 | <i>Construction de la lettre W . . . . .</i>        | 133 |
| 5.18 | <i>Construction de la lettre X . . . . .</i>        | 134 |
| 5.19 | <i>Construction de la lettre Y . . . . .</i>        | 135 |
| 5.20 | <i>Construction de la lettre Z . . . . .</i>        | 135 |
| 5.21 | <i>Construction de la lettre R . . . . .</i>        | 136 |
| 5.22 | <i>Construction de la lettre B . . . . .</i>        | 136 |
| 5.23 | <i>Construction de la lettre Q . . . . .</i>        | 137 |
| 5.24 | <i>Construction de la lettre K . . . . .</i>        | 137 |
| 5.25 | <i>Construction de la lettre C . . . . .</i>        | 138 |
| 5.26 | <i>Construction de la lettre G . . . . .</i>        | 138 |
| 5.27 | <i>La ligature FF . . . . .</i>                     | 139 |
| 5.28 | <i>Empattement de caractère romain . . . . .</i>    | 145 |
| 5.29 | <i>Lettre M avec deux empattements . . . . .</i>    | 146 |
| 5.30 | <i>Empattements obliques . . . . .</i>              | 147 |
| 5.31 | <i>Ornements inférieurs et supérieurs . . . . .</i> | 148 |
| 5.32 | <i>Empattements verticaux . . . . .</i>             | 149 |
| 5.33 | <i>Lettre E romaine corrigée . . . . .</i>          | 149 |
| 5.34 | <i>Empattement pour la famille tt . . . . .</i>     | 151 |
| 5.35 | <i>Empattements machine à écrire . . . . .</i>      | 152 |
| 5.36 | <i>Le W ramené à la largeur 14u . . . . .</i>       | 153 |
| 5.37 | <i>Style flamboyant . . . . .</i>                   | 154 |
| 5.38 | <i>Lettrine H de la police yinit . . . . .</i>      | 155 |
| 5.39 | <i>Contour d'un caractère . . . . .</i>             | 155 |
| 5.40 | <i>Négatif . . . . .</i>                            | 156 |

---

|      |                                                     |     |
|------|-----------------------------------------------------|-----|
| 5.41 | <i>Lettre avec motif</i>                            | 158 |
| 5.42 | <i>Les lettres E et X avant et après correction</i> | 162 |
| 5.43 | <i>Effets optiques</i>                              | 162 |
| 6.1  | <i>Le symbole de l'euro</i>                         | 167 |
| 6.2  | <i>Une blanche</i>                                  | 168 |
| 6.3  | <i>Une croche</i>                                   | 169 |
| 6.4  | <i>Hiéroglyphe o</i>                                | 172 |
| 6.5  | <i>Hiéroglyphe l</i>                                | 173 |
| 6.6  | <i>Hiéroglyphe i</i>                                | 174 |
| 6.7  | <i>Cartouche Ptolemaios</i>                         | 175 |
| A.1  | <i>Courbe de Bézier: construction des milieux</i>   | 178 |
| G.1  | <i>Une lettre en mode smoke</i>                     | 239 |

# Index

\* 25  
     prompt, 11  
 \*\* 25  
     prompt, 11  
 + 25  
 ++ 25  
 +, -, \*, / 28  
 +-+ 25  
 - 25  
 --- 72  
 ...step...until 43  
 / 25  
 < 24  
 ≤ 24  
 <> 24  
 = 24  
 ≥ 24  
 ??? 60  
 & 30, 31, 34, 36, 89, 90  
     bases, 21  
     chaînes, 31  
     chemins, 36

## A

lettre, construction, 122  
 abs 25, 29  
 addto 47, 89, 90, 176  
     also, 44  
     contour  
         withpen, 44  
         withweight, 44  
 doublepath  
     withpen, 44  
     withweight, 44  
 aleatrait 105  
 also 47

commençage 10, 131  
 contour 47, 89, 176  
     withpen, 44  
     withweight, 44  
 doublepath 105, 131  
     withpen, 44  
     withweight, 44  
 extraheight 24, 25  
     and 24  
 angle 25, 28, 130  
     and 25  
 approche de paires 139, 140, 199  
 approximation 159  
 ASCII 25, 27  
 aspect\_ratio 52  
 at 57  
 autorounding 161, 184

**B**  
 lettre, construction, 136  
 barheight# 113  
 base 3, 8, 14, 20–23, 42, 48, 62, 65, 97, 113, 158, 215  
     ligne de, 5, 53, 59  
 batchmode 61  
 beginchar 10, 37, 54, 55, 61, 183  
 begingroup 40  
 begingroup...endgroup 40  
 beginlogochar 115, 151  
 Bernstein 65  
 β, construction 5  
 Bézier 65, 67  
 bitmap 2  
 black 16, 238  
 blacker 9, 50, 52, 53, 114  
 blankpicture 30  
 boolean 23, 114  
 bot 25, 26, 38, 39  
 bp 50  
 bp# 50  
 btest.mf 14  
 byte 25, 27



**C**

lettre, construction, 138  
 calligraphie 2, 78  
 capsule\_def 62  
 cc 50  
 cc# 50  
 ceiling 25, 26  
 cercle 76  
 chaîne 27–31, 55, 60, 63, 156, 183, 192  
 Champollion 170  
 change\_width 54, 55, 121, 129, 161  
 char 30, 31  
 charcode 184  
 chardp 184  
 chardx 185  
 chardy 185  
 charexists 24  
 charext 184  
 charht 184  
 charic 185  
 charlist 54, 56  
 charwd 184  
 chemin 24, 28, 29, 34–36, 38–40, 44, 46,  
     47, 69, 75, 80, 82, 84, 90, 92,  
     93, 98, 99, 105, 108, 129, 169,  
     190, 197  
 clear\_pen\_memory 44  
 clearit 40, 42  
 clearpen 40, 42  
 clearxy 40, 42  
 cm 50  
 cm# 50  
 cmr 12  
 cmrfun 13  
 code 54  
 conditionnelle (instruction) 43, 67, 96,  
     144, 191, 192, 194  
 constant 30  
 contour 6, 46, 47, 101, 154, 197, 249  
 controls 35  
 cosd 25, 26  
 counterclockwise 34, 35  
 courbure 73  
 courrier, liste de diffusion 244  
 ctest.mf 14  
 cull 42, 47, 87, 92, 154, 157

dropping, 44  
 keeping, 44  
 cullit 40, 42, 47  
 curl 72  
 currentpen 37, 116  
 currentpicture 30  
 currenttransform 31, 33, 114, 142  
 currentwindow 57, 58  
 cutdraw 44, 47, 167  
 cutoff 44, 46, 47, 148  
 cycle 24, 69, 90, 93

**D**

lettre, construction, 126  
 day 184  
 dd 50  
 dd# 50  
 decalage 99, 122  
 decimal 30, 31  
 decr 25, 28  
 def 41, 98  
 def...edef 40, 41  
 define-pixels 52  
 define\_blacker\_pixels 51, 53, 115  
 define\_corrected\_pixels 51, 53, 115  
 define\_good\_x\_pixels 51, 53, 115  
 define\_good\_y\_pixels 51, 53, 115  
 define\_horizontal\_corrected\_pixels 51,  
     53, 115  
 define\_pixels 50, 53, 115  
 define\_whole\_blacker\_pixels 51, 53  
 define\_whole\_pixels 50, 53, 115  
 define\_whole\_vertical\_blacker\_pixels  
     51, 53  
 define\_whole\_vertical\_pixels 51, 53, 115  
 demi-cercle 76  
 demi-tangente 197  
 designsize 185  
 diagnostics 60, 189  
 Didot 1, 112  
 dir 38, 39  
 direction 38–40, 130  
 directionpoint 38  
 directiontime 25, 40, 94  
 display...inwindow 57

displaying 25, 29  
 ditto 30  
 div 25  
 dotprod 25, 132  
 dotsize 16  
 doublepath 47  
 down 38, 70  
 downto 43  
 dpi 10  
 draw 33, 44, 46, 67, 69, 72, 91, 92, 98, 169  
 drawdot 38, 44, 46, 82, 91  
 dropping 47  
 dump 21  
 dvipreview 12

## E

lettre, construction, 117  
 ellipse, *voir* superellipse  
 else 43  
 elseif 43  
 empat 146  
 empattement 145  
 macro, 145  
 end 11, 69  
 endchar 54, 55, 58, 155–157  
 enddef 40, 41, 102  
 endfor 43  
 endgroup 40  
 épreuve 7, 10, 14–16, 48, 58, 59, 63, 184, 202, 238  
 eps 25  
 epsilon 25, 175  
 équation 19, 28, 29, 41, 44, 56, 65, 67, 84, 102, 177, 178, 195  
 erase 44, 46, 82, 91, 93  
 erase draw 91  
 erase drawdot 91  
 erase fill 91, 93  
 erase filldraw 91  
 erreurs (messages) 41, 42, 47, 60–62, 84, 109, 150, 175, 176, 187–189, 191, 193, 195, 197, 199  
 errhelp 43, 60

errmessage 43, 60  
 errstopmode 60  
 euro 166  
 exitif 43, 95  
 exitunless 43, 95  
 expr 99  
 extensible 54, 56  
 extra\_beginchar 54, 55  
 extra\_endchar 54, 55, 156, 157

## F

lettre, construction, 118  
 false 24  
 .fd 17, 18  
 F(ligature), construction 139  
 fi 43  
 fichier  
 de paramètres, 12  
 pilote, 12  
 source, 3, 7–10, 17, 68, 188, 190  
 fill 33, 44, 46, 91, 93, 96  
 filldraw 44, 46, 69, 73, 79, 91, 92, 169  
 fillin 9, 50, 52, 53, 114, 184  
 fix\_units 50, 52  
 flex 34, 36, 78  
 floor 25, 26  
 font definition 18  
 font\_coding\_scheme 54, 57, 114, 140  
 font\_extra\_space 54, 55  
 font\_identifier 16, 54, 114, 140  
 font\_indentifier 57  
 font\_normal\_shrink 54, 55, 114, 140  
 font\_normal\_space 54, 55, 114, 140  
 font\_normal\_stretch 54, 55, 114, 140  
 font\_quad 54, 55, 114, 140  
 font\_size 54, 55  
 font\_slant 54, 55  
 font\_x\_height 54, 55  
 fontdimen 54, 56, 57  
 fontmaking 184  
 for 43, 95  
 forever 43, 95  
 format, xiii 3, 7, 9, 17, 19–21, 23, 28, 43, 52, 62, 66, 72, 84, 98, 114, 115, 142, 165, 196, 238

forsuffixes 43, 95, 146  
ftest.mf 14  
fullcircle 34

## G

lettre, construction, 138  
Garamond 1, 112  
generic font 10  
.gf 48, 69  
gfcorners 48, 50, 238  
gftodvi 8, 10, 12, 15, 241  
gftopk 8, 11, 16, 48, 241  
gobble 62  
good.bot 38, 39, 160  
good.lft 38, 39, 160  
good.rt 38, 39, 160  
good.top 38, 39, 160  
good.x 25, 26, 120, 160, 161  
good.y 25, 26, 160  
granularity 184  
gray 14  
grayfont 58, 59  
GUTenberg 243

## H

lettre, construction, 127  
halfcircle 34  
hauteur# 54  
headerbytes 54, 57  
hex 25, 27  
hexapen 75  
hide 60, 62, 101  
hiéroglyphes 170  
historique 2  
ho# 112  
hppp 52, 185  
hround 25, 26, 39  
ht# 112

## I

lettre, construction, 129  
identity 31, 32  
if 43  
if...elseif...else...fi 43  
image 2–4, 30, 42, 47, 57, 87, 89, 90, 99, 120, 123, 160, 194, 198  
fichier, 9, 16  
imagerules 48, 50  
in 50  
in# 50  
incr 25, 28  
infinity 25  
input 48, 114, 140  
interact 60, 61  
interim 40, 41  
interpath 34, 36, 92  
intersectionpoint 38, 40, 93, 96  
intersectiontimes 38, 40  
inverse 31, 32  
italcorr 54, 121

## J

lettre, construction, 130  
jobname 30, 31  
join\_radius 25, 29

## K

lettre, construction, 137  
keeping 47  
kern 139, 140  
known 24  
Knuth, Donald 2

## L

lettre, construction, 128  
label 238  
labelfont 58, 59  
labels 15, 58, 59, 69  
large\_pixels 16  
largeur# 54  
LaTeX, xiii 3, 8, 11, 17–19, 40, 245, 248, 249  
left 38, 70  
leftstemloc# 113

length 25, 29  
 let 40, 41, 98  
 lettre  
     À, construction, 122  
     B, construction, 136  
     β, construction, 5  
     C, construction, 138  
     D, construction, 126  
     E, construction, 117  
     F, construction, 118  
     G, construction, 138  
     H, construction, 127  
     I, construction, 129  
     J, construction, 130  
     K, construction, 137  
     L, construction, 128  
     M, construction, 119  
     N, construction, 120  
     O, construction, 123  
     P, construction, 125  
     Q, construction, 137  
     R, construction, 136  
     S, construction, 124  
     T, construction, 124  
     U, construction, 131  
     V, construction, 132  
     W, construction, 133  
     X, construction, 134  
     Y, construction, 135  
     Z, construction, 135  
 lft 25, 26, 38, 39  
 ligature 56, 114, 139  
     FF, construction, 139  
 ligatures 139  
 lightweight 16  
 ligtable 54, 56, 139, 140  
 lisibilité 12, 117, 129, 161, 162, 180  
 local.mf 20, 21, 58  
 localfont 20, 48  
 .log 10, 183, 184  
 loggingall 60, 61  
 logocomplet 112  
 lulu 21

**M**  
     mag 48, 49  
     magstep 48, 49  
     makegrid 58, 59  
     makelabel 58, 59, 68  
     makepath 34, 35  
     makepen 37, 38, 75  
     maketexpk 17  
     max 25, 28, 30, 31, 38, 39  
     métapolice, xiii, xiv 12, 111, 112, 145,  
         149, 163, 215  
     message 31, 42, 43, 60  
     mexp 25, 26  
     .mf 8  
     mf.base 20  
     MFBASES 8, 20, 21  
     MFINPUTS 8, 14  
     mfput 12  
     min 25, 28, 30, 31, 38, 39  
     mlog 25, 26  
     mm 50  
     mm# 50  
     mod 25, 96  
     mode, xiv 9–11, 14–16, 20, 48, 50–53,  
         55, 58, 59, 114, 121, 166, 188,  
         202, 231, 235, 238  
         proof, 14  
     \mode= 48  
     mode\_setup 52  
     mode\_setup 50–52, 113–115  
     modes.mf 20  
     month 184  
     motif 16, 157, 158  
     mtest.mf 14  
     musique 167  
**N**  
     lettre, construction, 120  
     négatif  
         effet de, 89, 156  
         pixel, 47, 87, 155  
     nodisplays 48, 49  
     nombre d'or 104  
     nonstopmode 61

normaldeviate 25, 27, 105

not 24

nullmode 231

nullpen 37,

116

nullpicture 30

numeric 23, 114

numspecial 62, 63

numtok 62

## O

lettre, construction, 123

o# 112, 117

o\_correction 9, 50, 52, 53, 114

oct 25, 27

odd 24

of 39, 40

openit 40, 42, 58

openwindow 57

opérateur

binaire, 42, 105

booléen, 24

numérique, 25

or 24

origin 38

OT1 17

overshoot 161

## P

lettre, construction, 125

packages 3

pair 23, 114

paramètres, fichiers de 13

Pascal (source) 2

path 23, 114

pausing 184

pc 50

pc# 50

pen 23, 114

pen\_bot 44, 45

pen\_lft 44, 45

pen\_rt 44, 45

pen\_top 44, 45

penché

style, 17, 144



pencircle 37, 62, 73

penlabels 58

penoffset 38

penpos 44, 45, 78, 145

penrazor 37, 62, 74, 80

penspeck 37, 38

pensquare 37, 62, 74

penstroke 44, 45, 78

pickup 37, 44, 46, 62

picture 23, 114

pilote, fichier 113

pix\_ht 16

pix\_picture 16

pix\_wd 16

pixellisation 26, 113, 115, 120, 158–161

pixels\_per\_inch 50, 52, 114

.pk 48

plain, xiii 7, 19–21, 23, 43, 52, 62

TEX, 19

plain.base 8, 20, 21

point 38, 39, 130

contact, 94, 168

intersection, 40, 93, 162

placement, 6, 26, 59, 66, 122, 129, 159

point-virgule 12

pointe de stylo 6, 35, 37–39, 45, 46, 53,

73–75, 80, 91, 115, 160, 161, 161,

167

points de construction 65

points de contrôle 65

police de caractères, xiii, xiv 7, 8, 11,

13, 14, 17–19, 23, 54, 55, 59,

65, 111–113, 141, 142, 161, 163,

165, 166, 182, 198

polycreux 101

polygone 100

postcontrol 38, 39

préambule 10, 17, 21, 99, 116, 122, 123,

142, 151, 156, 158, 159, 215

precontrol 38, 39

primarydef 105

primarydef... endef 40, 42

profondeur# 54

proof 9, 14

proofing 184

proofoffset 58, 59  
 proofrule 58, 59, 63  
 proofrulethickness 58, 59  
 pt 50  
 pt# 50  
 Ptolémée 170  
 px# 112  
 py# 112

## Q

lettre, construction, 137  
 qtest.mf 14  
 quart de cercle 76  
 quartercircle 34

## R

lettre, construction, 136  
 randomseed 25, 27, 106  
 range 58  
 readstring 30, 31  
 reflectedabout 31, 32, 84  
 \relax 11  
 rep 16  
 reverse 34, 35, 93  
 right 38, 70  
 romains (caractères) 145  
 romains du Roi 1  
 rotated 31, 84  
 rotatedaround 31, 32, 84  
 round 25, 26, 38, 39  
 rt 25, 26, 38, 39  
 rtest.mf 14

## S

lettre, construction, 124  
 s# 112  
 sans serif 145  
 save 40–42  
 savepen 44, 80  
 scaled 31  
 screen\_cols 57, 58  
 screen\_rows 57, 58  
 screenchars 48, 49  
 screenrule 58, 59  
 screenstrokes 48, 49

scrollmode 61  
 secondarydef... endef 40, 42  
 shifted 31, 84  
 shipit 11, 12, 40, 42, 69  
 shipout 11, 40, 42  
 show 40, 42, 109, 184  
 showdependencies 43, 60, 61, 109  
 showit 40, 42, 69  
 showstats 43, 60, 61  
 showstopping 184  
 showtoken 43, 60, 61  
 showvariable 43, 60, 61, 109  
 sind 25, 26  
 slanted 31, 84  
 slantfont 58, 59  
 smoke 16, 238  
 smoothing 161, 184  
 solve 25, 29, 103, 104, 129, 132, 175  
 solveur 19, 65, 102  
 source (fichier) 8, 241  
 special 59, 62, 63  
 sqrt 25, 26  
 step 43  
 step... until 43  
 stest.mf 14  
 stop 40, 42, 62  
 str 30, 31  
 string 23, 114  
 style  
 courrier, 150–152  
 flamboyant, 153  
 penché, 55  
 romain, 1, 14, 111, 145  
 stylo 62  
 épaisseur, 112, 160  
 forme, 3, 5, 6, 34, 35, 37–39, 44–  
 46, 73, 75, 78, 80, 91, 115, 148,  
 160, 161, 167  
 position, 39, 45, 161  
 taille, 5, 46, 121, 157, 160  
 subpath 35, 83  
 subpath ... of ... 34  
 substring 30  
 suffix 99  
 super\_crescent 116, 121, 123  
 super\_half 99, 116, 119

superellipse 34, 76  
supereness 34, 35, 76

## T

lettre, construction, 124  
tables de caractères 201  
tangente 39, 53, 65, 69–71, 94, 95, 120, 129, 130, 134, 160, 161, 168, 177, 184  
tensepath 34, 36  
tension 71, 72, 98  
tertiarydef... endef 40, 42  
**T****E****X**, xiii, xiv 2–4, 11, 19, 48, 57, 65, 117, 121, 140, 153, 163, 166, 176, 187, 199, 244–247  
text 99  
.tfm 48, 54, 57  
thru 58  
time 184  
titlefont 58, 59  
tolerance 25, 29, 104, 130, 175  
top 25, 26, 38, 39  
totalweight 25, 28  
tracé 4–7, 19, 26, 37–39, 44–46, 54, 55, 58, 59, 65, 72, 74, 75, 80, 92, 98, 101, 104, 105, 108, 129, 142, 145, 157, 160, 177, 183  
tracingall 60, 61  
tracingcapsules 183, 188  
tracingchoices 183, 188  
tracingcommands 183, 188  
tracingedges 184, 188  
tracingequations 183, 188  
tracingmacros 184, 188  
tracingnone 60, 61  
tracingonline 42, 183, 188  
tracingoutput 184, 188  
tracingpens 183, 188  
tracingrestores 183, 188  
tracingspecs 183, 188  
tracingstats 184, 188  
tracingtitles 183, 188  
transform 23, 114  
transformation 27, 31–33, 84, 85, 99, 194, 198

transformed 31, 32  
true 24  
ttest.mf 14  
turningcheck 184  
turningnumber 25, 28, 35, 184  
typographie, xiv 1, 4, 112

## U

lettre, construction, 131  
u# 112  
undraw 44, 46  
undrawdot 44, 46  
unfill 44, 46, 70  
unfilldraw 44, 46  
uniformdeviate 25, 27, 105  
unités 51, 52, 54, 179  
unités diées 115  
unitpixel 30  
unitsquare 30, 34  
unitvector 38, 98  
unknown 24  
until 43  
up 38, 70  
upto 43

## V

lettre, construction, 132  
vardef 41, 62  
vardef... endef 40, 41  
variables internes, xiv 41, 183  
vppp 52, 185  
vround 25, 26, 39

## W

lettre, construction, 133  
warningcheck 184  
whatever 25, 28, 132  
withpen 47  
withweight 47, 176

## X

lettre, construction, 134  
xdvi 12  
xgap# 112  
xoffset 185

---

xpart 25, 27

xscaled 31

xtest.mf 14

xxpart 25, 27

yypart 25, 27

## Y

lettre, construction, 135

year 184

ygap# 112

yoffset 185

ypart 25, 27

yscaled 31, 93

yxpart 25, 27

yypart 25, 27

## Z

lettre, construction, 135

zozo 9, 17

zscaled 31

ztest.mf 14

Achevé d'imprimer  
sur les presses de  
l'Imprimerie France Quercy  
113, rue André Breton  
46001 CAHORS  
d'après montages et gravure  
numériques  
(Computer To Plate)  
Dépôt légal : mars 1999  
Numéro d'impression : 90374



METAFONT, conçu par D.E. Knuth conjointement avec TEX, est un programme aussi riche et flexible que ce dernier, mais dont l'apprentissage est plus facile.

Le rôle de METAFONT est à la fois de créer des polices de caractères – y compris des symboles – et d'agir sur les polices existantes pour en modifier l'aspect. Ce langage permet aussi à l'utilisateur de construire des ensembles de commandes – macros – destinées à des tâches particulières.

- Après un rappel des apports de la typographie numérique aux techniques d'impression et de composition et le rôle révolutionnaire joué par METAFONT et TEX en ce domaine, sont abordées les notions nécessaires pour faire tourner des programmes et rendre utilisables les polices de caractères créées.
- Puis sont exposées et analysées toutes les macros, regroupées en fonction de l'objet auquel elles se rapportent.

• L'auteur guide ensuite l'utilisateur, pas à pas et grâce à de très nombreux exemples et illustrations, dans l'apprentissage de METAFONT, la tâche de construction d'une police, enfin son usage quotidien.

- Les annexes regroupent des informations pratiques telles que modes existants, tables de conversion, encodages, variables internes, etc.

Les débutants comme les utilisateurs initiés seront séduits par la clarté de cet ouvrage (**le premier sur ce sujet disponible en langue française**), mais également son exhaustivité puisque l'intégralité des macro-commandes constituant le format *plain* de METAFONT y est présentée.

Son public est celui de toutes les disciplines scientifiques – professionnels ou étudiants qui rédigent leurs articles et ouvrages au moyen de TEX ou LATEX –, mais aussi les utilisateurs d'alphabets spéciaux (hiéroglyphes, langues anciennes, chinois, etc.).

## L'AUTEUR

*Bernard Desgraupes, ancien élève de l'Ecole Normale Supérieure de la rue d'Ulm et agrégé de mathématiques, est actuellement maître de conférences à l'Université Paris X et travaille en informatique sur la modélisation de problèmes d'analyse harmonique. Il est également l'auteur d'une thèse sur l'homogénéisation des équations aux dérivées partielles.*