Affichage générique d'arbres à l'aide de la géométrie hyperbolique

Alexandre Miquel[1]

1: INRIA Rocquencourt
78153 Le Chesnay CEDEX, France
`Alexandre.Miquel@inria.fr`

Résumé

Nous présentons ici un afficheur générique d'arbres utilisant des techniques issues de la géométrie hyperbolique. Paramétré par des structures abstraites très générales, cet afficheur écrit en Objective Caml traite des arbres de profondeur arbitraire (voir infinis), d'une manière indépendante du périphérique de sortie. Enfin, ce programme montre comment l'utilisation de la géométrie hyperbolique permet de résoudre le problème de la répartition des noeuds dans l'espace d'affichage disponible plus simplement et plus efficacement qu'avec la géométrie euclidienne.

1. Introduction

S'il est assez facile de représenter graphiquement de petits arbres (typiquement les arbres d'une dizaine de noeuds), le problème de la répartition des noeuds dans l'espace graphique devient rapidement ardu dès qu'il s'agit d'afficher *in extenso* de plus grosses arborescences.

Pourtant, ce problème de représentation n'est pas anodin si l'on songe que de nombreuses structures informatiques se présentent naturellement sous une forme d'arbre. Il suffit de penser aux arborescences de fichiers des disques durs modernes, qui comprennent des milliers de fichiers. Par ailleurs, sur le réseau Internet, l'ossature de la plupart des sites WEB est construite sur le modèle d'une arborescence (même si les liens croisés entre les pages que ces sites hébergent donnent à l'ensemble une structure de graphe enchevêtrée).

Nous décrivons une application de la géométrique hyperbolique qui permet d'afficher (et de naviguer dans) des arborescences bien plus volumineuses que celles que les techniques classiques — qui travaillent toutes en géométrie euclidienne — permettent de traiter actuellement.

La méthode de représentation que nous exposons ici n'est pas nouvelle. Elle a déjà fait l'objet de recherches au Palo Alto Research Center (PARC) de Xerox [1]. Depuis quelques années, la société Inxight Software commercialise des outils de navigation dans les systèmes de fichiers ou les sites WEB basés sur le même principe.

Nous montrons que cette méthode de représentation des arbres, efficace mais peu connue, s'écrit naturellement dans un langage fonctionnel (ici, en Objective Caml [2, 3]), et qu'elle met en valeur les capacités d'abstraction de cette famille de langages (en particulier, le système de modules et de foncteurs d'Objective Caml [3, 4, 5]).

Notre algorithme est très général, puisqu'il manipule n'importe quelle structure d'arbre à branchement fini, y compris les arborescences infinies si le langage d'implémentation dispose du mécanisme d'évaluation paresseuse. En effet, grâce à une abstraction très générale de la notion d'arbre, l'algorithme est capable non seulement de manipuler des arbres finis ou rationnels représentés par des types concrets, mais aussi des arbres paresseux infinis non rationnels. Un bel exemple d'affichage et de navigation dans un arbre infini non rationnel est donné au paragraphe 4.2, puisque l'algorithme affiche l'arbre de tous les mots : le dictionnaire des dictionnaires.

2. Affichage d'arbres en géométrie hyperbolique

L'algorithme que nous présentons trace les arbres dans le plan hyperbolique complexe, en tirant parti des propriétés géométriques de cet espace non euclidien pour répartir de manière efficace les noeuds dans l'espace dont il dispose. Nous commençons donc par quelques rappels sur la géométrie du plan hyperbolique, en passant en revue les concepts et résultats élémentaires indispensables pour la compréhension de l'algorithme. Toutefois, nous n'entrerons pas dans le détail de la théorie mathématique sous-jacente dont on trouvera un exposé approfondi dans [6].

2.1. Le plan hyperbolique complexe

La tortue LOGO revisitée Il est commode d'introduire la géométrie du plan hyperbolique complexe en décrivant le comportement d'une tortue LOGO placée dans cet espace. Pour cela, il est nécessaire de définir :

1. la notion de point;
2. la métrique de l'espace;
3. la notion de direction prise par la tortue;
4. les déplacements et changements de cap de la tortue.

Le plan hyperbolique complexe, noté $\mathcal{B}$, est le disque unité ouvert du plan complexe $\mathbb{C}$ constitué par l'ensemble des nombres complexes de module strictement inférieur à 1, soit

$$\mathcal{B} \;=\; \{z \in \mathbb{C}; \quad |z| < 1\}.$$

Une direction dans cette espace est représentée par un nombre complexe unitaire, c'est-à-dire un élément de l'ensemble $\mathcal{U}$ défini par

$$\mathcal{U} \;=\; \{z \in \mathbb{C}; \quad |z| = 1\}.$$

Une tortue est donc la donnée d'une position (un élément de $\mathcal{B}$) et d'une direction (un élément de $\mathcal{U}$). On la représente en Caml par le type `turtle` défini par :

```
type coord = float * float ;;

type turtle = {
  pos : coord ;  (* with |pos| < 1 *)
  dir : coord    (* with |dir| = 1 *)
} ;;
```

Notons que la tortue définie ainsi est utilisée de manière purement fonctionnelle, puisque les champs de l'enregistrement qui la décrivent ne sont pas mutables.

Métrique du plan hyperbolique La distance entre deux points a et b du plan hyperbolique est définie par

$$d(a,b) \;=\; \operatorname{ath}\left|\frac{a-b}{1-\bar{a}b}\right|,$$

où ath désigne la fonction arctangente hyperbolique donnée par

$$\operatorname{ath}(x) \;=\; \frac{1}{2}\ln\Big(\frac{1+x}{1-x}\Big).$$

Remarquons que lorsque l'un des deux points a ou b tend vers le bord du disque, la distance $d(a,b)$ tend vers l'infini. Autrement dit, bien que le plan hyperbolique soit pour l'observateur un sous-ensemble de $\mathbb{C}$ borné, ce même espace, vu par les yeux de la tortue, constitue un espace infini.

Insistons sur le fait que l'ensemble $\mathcal{B}$ peut être vu soit comme le disque unité de $\mathbb{C}$ muni de sa structure euclidienne (le point de vue de l'observateur), soit comme l'espace métrique $(\mathcal{B}, d)$ muni de la géométrie non-euclidienne que la distance d lui confère (le point de vue de la tortue).

Isométries positives du plan hyperbolique Afin d'être en mesure de définir les modes de déplacement de notre tortue hyperbolique, nous devons d'abord répertorier l'ensemble des isométries positives du plan hyperbolique complexe, c'est-à-dire les isométries (pour la distance d) préservant l'orientation. Celles-ci sont engendrées par deux familles de transformations [6] :

1. Les rotations r_u, paramétrées par un nombre complexe unitaire $u \in \mathcal{U}$, et définies par :

$$r_u(z) \;=\; uz.$$

2. Les "translations" t_a, paramétrées par un point $a \in \mathcal{B}$, et définies par :

$$t_a(z) \;=\; \frac{a+z}{1+\bar{a}z}.$$

Notons que dans le plan hyperbolique complexe, la notion de "translation" n'a que peu de rapports avec la translation usuelle des espaces euclidiens, bien qu'il soit commode de reprendre le terme puisqu'il correspondra au mouvement effectué par notre tortue lorsqu'elle avance.

Proposition 2.1 — *L'ensemble des fonctions de la forme $t_a \circ r_u$, lorsque a et u décrivent $\mathcal{B}$ et $\mathcal{U}$ respectivement, constitue un groupe, qui est le groupe des transformations conformes de $\mathcal{B}$ dans lui-même.*

(On trouvera une preuve de ce résultat dans [7].)

Rappelons qu'une transformation conforme est une transformation conservant les angles. Cette proposition a son importance, car elle signifie que la notion d'angle est la même dans le plan hyperbolique que dans le plan euclidien.

Déplacement de la tortue Considérons dans un premier temps une tortue partant de l'origine à l'instant 0 dans la direction du nombre 1. La position de cette tortue à l'instant t est donnée par

$$\gamma(t) \;=\; \operatorname{th} t \;=\; \frac{e^t - e^{-t}}{e^t + e^{-t}}.$$

(où $\operatorname{th} t$ désigne la tangente hyperbolique de t). Remarquons que $\gamma(t)$ tend vers 1 lorsque t tend vers l'infini, ce qui signifie que la tortue mettra un temps infini pour atteindre le bord du disque. Cette trajectoire est une géodésique du plan hyperbolique, c'est-à-dire une courbe minimisant la distance parcourue entre deux points quelconques de celui-ci. Par ailleurs, cette géodésique correspond à un mouvement rectiligne uniforme de vitesse égale à 1 pour la tortue, puisque pour tous t_1, t_2 on a

$$d\big(\gamma(t_1), \gamma(t_2)\big) \;=\; |t_1 - t_2|.$$

Pour passer au cas général et obtenir ainsi toutes les géodésiques du plan hyperbolique, il suffit de composer cette trajectoire de référence avec les transformations du plan hyperbolique décrites précédemment. Soit donc a un point de $\mathcal{B}$ et $u \in U$ une direction. La trajectoire d'une tortue partant de a dans la direction u est donnée par

$$\gamma_{a,u}(t) \;=\; t_a \circ r_u \circ \gamma(t) \;=\; \frac{a + u \operatorname{th} t}{1 + \bar{a}u \operatorname{th} t}.$$

Dans le cas général, la trajectoire de la tortue n'est pas rectiligne du point de vue de l'observateur. Par conséquent, la direction suivie par la tortue change au cours de son déplacement, ce qui nécessite l'introduction d'une fonction $\delta_{a,u}(t)$ donnant cette direction à l'instant t. La fonction $\delta_{a,u}$ est définie comme étant la dérivée normalisée de la fonction $\gamma_{a,u}$, soit

$$\delta_{a,u}(t) \;=\; \frac{\gamma'_{a,u}(t)}{|\gamma'_{a,u}(t)|} \;=\; \frac{u + a\,\mathrm{th}\,t}{1 + \bar{a}u\,\mathrm{th}\,t}.$$

Nous pouvons maintenant définir les fonctions Caml de déplacement et de rotation de la tortue.

La fonction de déplacement utilise les fonctions $a, u, t \mapsto \gamma_{a,u}(t)$ et $a, u, t \mapsto \delta_{a,u}(t)$ (notées `gamma` et `delta` en Caml) afin de mettre à jour la position et la direction de la tortue.

Quant à la rotation d'un angle de d degrés, elle s'effectue de la manière habituelle en multipliant le vecteur de direction de la tortue par le nombre complexe $e^{id\pi/180}$. Cette dernière utilise une fonction `expi` calculant l'exponentielle circulaire, un opérateur infixe `*&` de multiplication des nombres complexes, ainsi qu'une constante `pi_over_180` permettant le passage des degrés aux radians.

Le code Caml correspondant à ces fonctions est le suivant :

```
let advance turt step =
  { pos = gamma turt.pos turt.dir step ;
    dir = delta turt.pos turt.dir step } ;;

let turn_left turt deg =
  { pos = turt.pos ;
    dir = turt.dir *& expi(deg *. pi_over_180) } ;;

let turn_right turt deg =
  { pos = turt.pos ;
    dir = turt.dir *& expi(-.deg *. pi_over_180) } ;;
```

Notons qu'ici, on a fait figurer deux fonctions de rotation, une pour "tourner à gauche" et l'autre pour "tourner à droite", afin de rester le plus près possible de l'esprit du LOGO.

Droites du plan hyperbolique complexe Par définition, la trajectoire d'une tortue dessine une demi-droite du plan hyperbolique complexe, lorsque t décrit l'intervale $[0; \infty[$. Ainsi une droite du plan hyperbolique complexe est-elle formée par la réunion des trajectoires de deux tortues partant d'un même point dans des directions opposées. Les droites du plan hyperbolique complexe sont alors caractérisées par la proposition suivante [6] :

Proposition 2.2 (Droites de $\mathcal{B}$) — *Les droites de $\mathcal{B}$ sont :*

1. *les diamètres ouverts du disque unité complexe;*
2. *les traces sur le disque unité ouvert des cercles othogonaux au cercle unité.*

Notons qu'un diamètre du disque unité peut être vu comme un cas particulier d'arc de cercle orthogonal au cercle unité dont le rayon serait infini. Il est par ailleurs clair que ces droites, en tant que géodésiques de $\mathcal{B}$, sont invariantes par les transformations de $\mathcal{B}$ citées précédemment.

Une géométrie non-euclidienne Il est intéressant de rappeler que, historiquement, la géométrie hyperbolique a été introduite afin de montrer l'indépendance de l'axiome d'Euclide (appelé également *axiome des parallèles*) par rapport aux autres axiomes de la géométrie euclidienne.

En effet, la géométrie du plan hyperbolique $\mathcal{B}$ réfute l'axiome des parallèles puisque par un point donné de cet espace, il passe une infinité de droites parallèles à une droite donnée. Pour s'en convaincre,

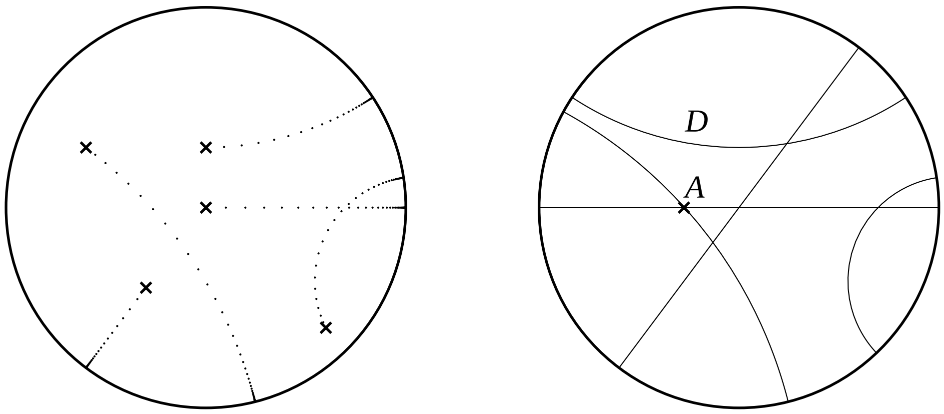


FIG. 1 – *Les trajectoires de cinq tortues dans $\mathcal{B}$ et les droites qu'elles dessinent*

il suffit d'observer la figure 1 ou l'on a fait figurer deux droites parallèles à la droite D, passant toutes les deux par un même point A. En revanche, les autres axiomes de la géométrie euclidienne restent vérifiés.

2.2. L'algorithme de répartition des noeuds

L'algorithme de répartition des noeuds d'un arbre dans le plan hyperbolique complexe repose sur une propriété géométrique liée aux demi-plans de cet espace. Comme dans le plan euclidien, chaque droite de $\mathcal{B}$ sépare cet espace en deux zones disjointes, appelées demi-plans, ou encore demi-espaces.

Une des propriétés élémentaires du plan euclidien est l'impossibilité de disposer dans cet espace plus de deux demi-plans sans que ceux-ci se rencontrent. En revanche, dans le plan hyperbolique complexe, il est tout à fait possible de disposer n demi-plans deux à deux disjoints, et cela quel que soit l'entier n. Cette propriété qui est à la base de l'algorithme de répartition des noeuds est illustrée dans la figure 2 par le "paradoxe des n dictateurs".

Considérons à présent un arbre t ayant n fils $t_1, \ldots, t_n$. On commence d'abord par placer la racine de t en un point quelconque a du plan hyperbolique complexe. Pour placer les racines de ses fils, on effectue les étapes suivantes :

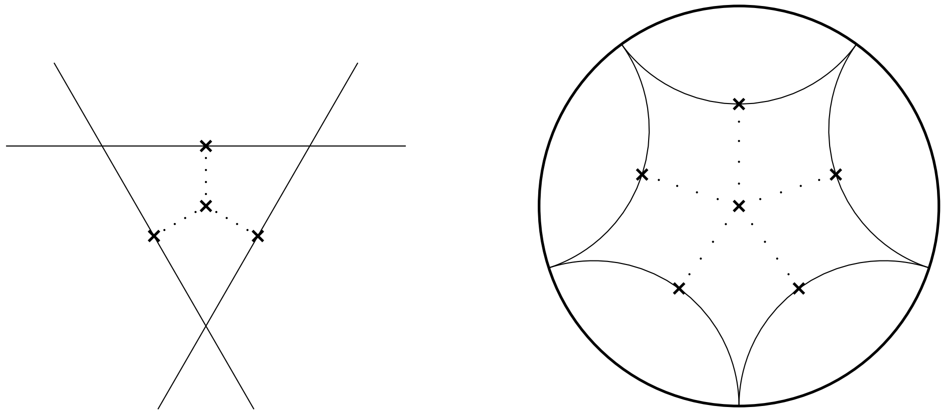
1. On dispose n tortues au point a, que l'on oriente dans des directions espacées de $360/n$ degrés, de façon à répartir régulièrement les directions prises par les tortues autour du point de départ.
2. On fait avancer les n tortues d'une certaine distance d.
3. Pour tout $i = 1 \ldots n$, on dessine la racine de l'arbre t_i au point a_i où s'est arrêtée la i-ème tortue. On note alors u_i la (nouvelle) direction prise par cette tortue, et E_i le demi-espace qui lui fait face, délimité par la droite passant par a_i orthogonale à la trajectoire de la tortue.

L'intérêt d'une telle construction vient de ce que les demi-espaces E_i ainsi définis sont deux à deux disjoints lorsque la distance d parcourue par chacune des n tortues est suffisamment grande (voir figure 2).

Par conséquent, il est possible de répéter récursivement la même construction pour chacun des arbres t_i dans les demi-espaces E_i à partir des points a_i, en modifiant l'étape 1 de façon à prendre en compte le fait que l'espace à partager est un demi-espace, et non plus le plan hyperbolique complexe tout entier. Concrètement, il suffit d'espacer les directions prises par les n_i tortues suivantes (correspondant aux n_i fils de l'arbre t_i) de $180/n_i$ degrés, autour de la direction u_i. Une nouvelle itération de l'étape 3 pour l'arbre t_i permet alors de construire n_i espaces $E_{i,j}$ $(1 \leq j \leq n_i)$, tous

FIG. 2 – *Le paradoxe des n dictateurs*

Dans un espace infini à deux dimensions, n dictateurs situés en un même point désirent se partager le monde de façon à régner chacun sur un demi-espace. Pour ce faire, ils se mettent en marche dans n directions régulièrement espacées de $360/n$ degrés. Lorsqu'ils ont tous parcouru une distance d, les n dictateurs s'arrêtent, et chacun d'entre eux s'approprie le demi-espace en face de lui (et dont la frontière est une droite imaginaire passant par ses deux pieds).



Dans le plan euclidien, cette méthode de partage mène rapidement à un conflit généralisé dès que $n \geq 3$, puisque les zones que les dictateurs se sont attribuées ne sont pas deux à deux disjointes (figure de gauche, avec $n = 3$).

En revanche, dans le plan hyperbolique complexe, quel que soit n, il existe une distance d à partir de laquelle les demi-espaces attribués aux dictateurs sont deux à deux disjoints (figure de droite, avec $n = 5$).

inclus dans le demi-espace E_i, et ainsi de suite.

Bien entendu, le bon fonctionnement de l'algorithme dépend de la distance d choisie. Si cette distance est trop courte, les demi-espaces E_i ne sont pas deux à deux disjoints (on est alors ramené à une situation proche de celle du plan euclidien). Si en revanche cette distance est trop grande, les demi-espaces E_i risquent d'être trop espacés, et la représentation de l'arbre trop clairsemée.

En pratique, la distance optimale d_n (qui dépend du nombre de fils à répartir dans le plan hyperbolique complexe), est donnée par :

$$d_n \;=\; \mathrm{ath}\Big(\tan\Big(\frac{\pi}{4} - \frac{\pi}{2n}\Big)\Big), \qquad (n \geq 2)$$

(Cette distance correspond à la situation où les n arcs de-cercles orthogonaux au cercle unité qui délimitent les espaces E_i sont tangents deux à deux consécutivement, comme dans la figure 2.)

Dans le cas où il s'agit de disposer les n fils dans un demi-espace E au lieu du plan hyperbolique complexe entier, on utilisera la distance d_{2n}, ce qui revient à placer $2n$ fils dans le plan hyperbolique complexe $\mathcal{B}$, n fils étant situés dans E et les n autres fils (virtuels) dans le complémentaire de E.

Remarquons également que la distance d_n n'est définie que pour $n \geq 2$, et qu'elle est nulle lorsque $n = 2$. Afin d'assurer un affichage correct d'un arbre même lorsque le nombre de ses successeurs est égal à 1 ou à 2, on utilisera la distance d'_n définie par

$$d'_n \;=\; d_{\max(n,3)}.$$

Affichage d'arbres infinis L'algorithme ci-dessus permet naturellement de disposer dans le plan hyperbolique complexe tous les noeuds d'un arbre à branchement fini, même si l'arbre en question comporte des branches infinies. En pratique, il faut pourtant décider d'un critère permettant d'arrêter l'algorithme à un certain moment si on veut que l'affichage se fasse en un temps fini. Pour cela, on définit dans le disque unité un rayon limite $R < 1$ au delà duquel on arrête de tracer l'arbre. Plus précisément, on introduit une "couronne d'arrêt" $C \subset \mathcal{B}$ définie par

$$C \;=\; \{z \in \mathcal{B}; \quad R < |z| < 1\}$$

et on décide d'arrêter récursivement le tracé de l'arbre t dès que le demi-espace E dans lequel le tracé de t doit s'effectuer est entièrement inclus dans C.

3. Implémentation de l'algorithme

3.1. Structure du module `Htree`

L'algorithme d'affichage d'arbre est implémenté dans le module `Htree`, qui constitue le coeur du programme proprement dit.

À la différence de l'algorithme que nous avons exposé ci-dessus, l'implémentation effectuée dans `Htree` commence le tracé de l'arbre à partir d'un demi-espace et non à partir du plan hyperbolique complexe entier. Ceci évite d'une part de traiter à part le cas de la racine (puisque le tracé des sous-arbres stricts s'effectue dans des demi-espaces), mais aussi, ceci permet visuellement de repérer plus rapidement la racine de l'arbre dans le dessin final.

Afin de conserver une très grande généralité, le module `Htree` repose sur deux abstractions, l'une servant à encapsuler la notion d'arbre, l'autre servant à encapsuler la notion de moteur d'affichage (afin de rester indépendant du périphérique de sortie).

Abstraction de la notion d'arbre Lors de l'affichage d'un arbre, seule l'opération de destructuration d'arbre est utilisée par la fonction d'affichage. Du point de vue de l'afficheur, un arbre est donc essentiellement un type muni d'un opérateur de destructuration permettant d'accéder aux descendants d'un noeud. Cette abstraction est codée dans `Htree` par la signature suivante :

```
module type TREE = sig
  type t
  type label
  val children : t -> t list
  val label : t -> label
end ;;
```

Dans cette signature sont définis quatre champs, qui sont :

- un type abstrait `t` représentant les arbres sur lesquels on désire travailler (ou le type de ses noeuds);
- un type abstrait `label` représentant les "étiquettes" attachées aux noeuds des arbres de type `t`, lesquelles sont susceptibles de contenir toutes les informations que l'utilisateur désire associer aux noeuds de l'arbre lors de l'affichage;
- une fonction `children` associant à tout arbre la liste de ses descendants directs;
- une fonction `label` associant à tout noeud son "étiquette".

Abstraction de la notion de moteur graphique Afin d'effectuer son travail, l'afficheur générique d'arbres a besoin d'un ensemble de routines de bas niveau lui permettant essentiellement d'afficher les arcs reliant les noeuds de l'arbre et les étiquettes. Cet ensemble de procédures est encapsulé dans l'enregistrement `driver` défini par :

```
type coord = float * float ;;

type driver = {
  rlimit : float ;
  moveto : coord -> unit ;
  lineto : coord -> unit ;
  curveto : coord -> coord -> coord -> unit ;
  draw_label : label -> coord -> float -> unit ;
  init_edge_pass : unit -> unit ;
  init_label_pass : unit -> unit ;
  finalize : unit -> unit
} ;;
```

Cette structure contient :

- Un champ `rlimit` indiquant le rayon $R_{\max}$ à partir duquel il convient d'arrêter le tracé. Cette valeur doit être un nombre flottant compris strictement entre 0 et 1. En pratique, $0,95$ semble être une bonne valeur pour ce paramètre.
- Des procédures `moveto`, `lineto` et `curveto` permettant d'effectuer le tracé des arcs reliant les noeuds. La procédure `moveto` positionne le curseur graphique à l'endroit désiré, et la procédure `lineto` tire un trait reliant l'ancienne position du curseur graphique à la nouvelle position donnée en argument. La procédure `curveto` trace les courbes de Bézier construites par l'afficheur pour représenter les arcs reliant les noeuds de l'arbre. Elle prend trois positions en argument, qui sont les positions des deux points de contrôle et la position du point d'arrivée, la position du point de départ étant fournie par l'ancienne position du curseur graphique.
- Une procédure `draw_label` permettant d'afficher les étiquettes attachées aux noeuds de l'arbre. Cette procédure prend en argument une étiquette, une position ainsi qu'un *facteur de réduction*.

Ce facteur de réduction est un nombre flottant compris entre 0 et 1 permettant de simuler visuellement la "fuite vers l'infini" d'un objet s'approchant du bord du plan hyperbolique complexe, en le faisant apparaître de plus en plus petit. Notons que le concepteur du moteur graphique fournissant la procédure `draw_label` est libre de tenir compte ou d'ignorer ce paramètre supplémentaire.

– Des procédures d'initialisation `init_edge_pass` et `init_label_pass` ainsi qu'une procédure de finalisation `finalize`. L'affichage de l'arbre étant réalisé en deux passes — tracé des arcs puis tracé des étiquettes —, chacune des procédures d'initialisation est invoquée avant la passe correspondante, afin d'effectuer tous les changements nécessaires à l'état du moteur graphique (typiquement, un changement de la couleur courante). Quant à la procédure de finalisation, elle est appelée une fois que le tracé est terminé (pour permettre par exemple la restauration d'un ancien état du moteur graphique).

Toutes les coordonnées passées aux procédures de tracé sont exprimées dans le repère du plan hyperbolique complexe (c'est-à-dire par couples de nombres compris entre -1 et 1). Il est donc de la responsabilité du concepteur du moteur graphique d'effectuer les changements d'échelle nécessaires afin d'adapter ces coordonées aux dimensions de la fenêtre dans laquelle s'effectue le tracé.

Par ailleurs, nous avons fait le choix de représenter les arcs reliant les noeuds par des courbes de Bézier et non par des arcs de cercle. Ce choix qui peut sembler paradoxal a été motivé par plusieurs raisons d'ordre pratique. La première d'entre-elles est qu'il est plus facile de calculer les paramètres d'une courbe de Bézier approchant la trajectoire d'une tortue que de déterminer avec précision le rayon de cette trajectoire (lequel peut être très grand, voire infini) et les coordonnées angulaires des points de départ et d'arrivée. De plus, le tracé d'une courbe de Bézier est plus facile à implémenter et plus efficace que le tracé d'un arc de cercle. Enfin, la plupart des systèmes d'affichage modernes — dont le langage PostScript — proposent des primitives pour tracer ces courbes. À ce titre, signalons que les implémentations actuelles de PostScript utilisent des approximations à base de courbes de Bézier afin de tracer les cercles et les arcs de cercle.

Le foncteur d'extension `Htree.Make` Dans le module `Htree` est défini un foncteur d'extension `Make` dont la signature est :

```
module Make(T : TREE) : HTREE with type t = T.t and type label = T.label
```

Ce foncteur sert essentiellement à étendre le module `T` de signature `TREE` passé en argument en un nouveau module de signature `HTREE` dans lequel sont définies les fonctions d'affichage pour les arbres de type `T.t`. La signature `HTREE` est la suivante :

```
module type HTREE = sig
  type t
  type label
  val children : t -> t list
  val label : t -> label

  type coord = float * float

  type driver = ...

  val shrink_factor : coord -> float
  val drag_origin : coord -> coord -> coord -> coord

  val draw_linear_tree : driver -> t -> coord -> float -> unit
  val draw_curved_tree : driver -> t -> coord -> float -> unit
end
```

Cette signature reprend les quatre champs définis dans le module `T` donné en argument au foncteur `Make`. Les deux champs suivants `coord` et `driver` définissent le type des coordonées ainsi que le type des moteurs d'impression évoqué précédemment.

Enfin, cette signature ajoute quatre nouveaux champs, qui sont :

- Une fonction `shrink_factor` retournant le facteur de réduction associé à un point a du plan hyperbolique complexe. Ce facteur de réduction vaut $1 - |a|^2$.
- Une fonction `drag_origin` prenant en argument trois points a, b et c du plan hyperbolique complexe, et qui retourne l'image de a par l'unique "translation" de $\mathcal{B}$ transformant b en c. Cette fonction est utilisée en mode interactif pour recalculer le point de départ du tracé de l'arbre lorsque l'utilisateur "tire" sur une partie de l'arbre afin de l'amener à un autre endroit de l'écran. Dans ce cas, a contient l'ancien point de départ, b le point sur lequel l'utilisateur a cliqué et c le point vers lequel l'utilisateur a fait glisser la souris.
- Deux fonctions de tracé `draw_linear_tree` et `draw_curved_tree`. La première effectue un tracé rapide où les arcs reliant deux noeuds sont représentés par des segments de droites, tandis que la seconde effectue un tracé plus long où les mêmes arcs sont représentés par des courbes de Bézier. Ces deux fonctions prennent en argument un moteur d'impression, un arbre, la position de la racine dans $\mathcal{B}$ ainsi que la direction (exprimée en degrés) dans laquelle le tracé commence (celle-ci servant essentiellement à définir le demi-espace initial de tracé).

Là encore, toutes les coordonnées sont données dans le repère du plan hyperbolique complexe, sous la forme de couples de nombres compris entre -1 et 1.

3.2. Interfaçage avec la fenêtre graphique de Caml

L'interfaçage de l'algorithme générique de tracé avec la fenêtre graphique de Caml est effectué dans un module annexe `Gtree`.

Contrairement au module `Htree`, le module `Gtree` fournit une procédure d'affichage et de navigation "prête à l'emploi". Pour cela, le module `Gtree` redéfinit la signature `TREE` représentant les arbres génériques (déjà définie dans le module `Htree`) en lui adjoignant un champ `string_of_label` qui associe à chaque étiquette sa représentation sous forme d'une chaîne de caractères :

```
module type TREE = sig
  type t
  type label
  val children : t -> t list
  val label : t -> label
  val string_of_label : label -> string
end
```

À l'instar du module `Htree`, le module `Gtree` définit un foncteur `Make` dont la signature est :

```
module Make(T : TREE) : GTREE with type t = T.t and type label = T.label
```

Le foncteur d'extension `Gtree.Make` prend en argument un module `T` de signature `TREE` et construit un module de signature `GTREE`, laquelle est définie par :

```
module type GTREE = sig
  type t
  type label
  val children : t -> t list
  val label : t -> label
  val string_of_label : label -> string
  val show_tree : t -> int -> int -> unit
end
```

La signature `GTREE` reprend les cinq champs de la signature `TREE` définie précédemment, et ajoute un champ supplémentaire définissant une procédure `show_tree`, laquelle permet d'afficher un arbre et de naviguer de manière interactive dans sa structure. Les trois arguments passés à cette procédure sont l'arbre à afficher ainsi que les dimensions de la fenêtre graphique dans laquelle on désire afficher l'arbre.

De manière interne, le foncteur `Gtree.Make` est construit à partir du foncteur `Htree.Make`, et la procédure `show_tree` utilise la fonction d'affichage générique définie dans le module `Htree`. Notons que pour afficher l'arbre dans la fenêtre graphique, la procédure `show_tree` utilise deux moteurs d'impression (passés en argument aux procédures `draw_linear_tree` et `draw_curved_tree` définies dans le module `Htree`), l'un pour le tracé de l'arbre, et l'autre pour l'effacement de l'arbre.

4. Applications

Dans sa version actuelle, le programme `htree`[1] permet d'afficher deux structures d'arbres très différentes, qui sont :

- l'arborescence des répertoires d'un système de fichiers Unix;
- l'arbre infini des mots construits sur l'alphabet latin — le "dictionnaire des dictionnaires".

4.1. Affichage d'une arborescence de répertoires

La structure représentant les arborescences de répertoires est implémentée dans le module `DirTree`. La signature de ce module est compatible avec la signature `Gtree.TREE` de manière à ce qu'il puisse être passé directement en argument au foncteur `Gtree.Make`. Dans ce module, le type des arbres est défini par le type enregistrement suivant :

```
type t = {
  name : string ;
  children : t list Lazy.t
} ;;
```

Le champ `name` contient le nom du répertoire correspondant, tandis que le champ `children` contient une valeur paresseuse susceptible de s'évaluer sur la liste des descendants directs du répertoire en question. Notons que l'utilisation d'une liste évaluée au besoin est indispensable pour éviter un parcours récursif complet du système de fichiers avant de commencer à tracer l'arbre.

Enfin, le module `DirTree` fournit une fonction `from_dir` prenant en argument un chemin d'accès à un répertoire, et retournant la structure de type `t` correspondante.

4.2. Le dictionnaire des dictionnaires

L'autre exemple d'arborescence traitée par notre afficheur d'arbres est le dictionnaire de tous les mots construits sur l'alphabet latin, implémenté dans le module `DictTree`. Cet ensemble de mots a naturellement la structure d'arbre infini suivante :

- sa racine est le mot vide;
- les fils d'un mot u sont les mots de la forme ux où x décrit les 26 lettres de l'alphabet latin.

Dans ce module, le type `t` des arbres est simplement défini par

```
type t = string
```

1. Accessible sur le WEB à l'adresse `http://pauillac.inria.fr/~miquel/`

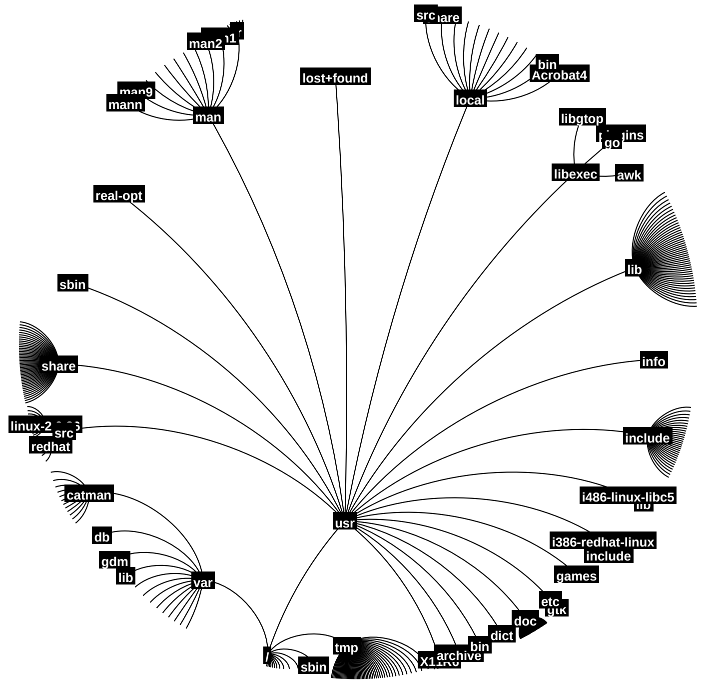


FIG. 3 – *Affichage d'une arborescence de répertoires*

et la fonction `children` se contente de recalculer à chaque fois la liste des successeurs du mot donné en argument.

5. Améliorations du logiciel

Concernant l'implémentation, l'utilisation d'un langage fonctionnel tel que Objective Caml s'est avérée parfaitement adaptée à l'écriture de ce type de programme, principalement grâce à la richesse des structures de programmation offertes au programmeur. Par ailleurs, le code écrit est assez compact, puisqu'il fait moins de 700 lignes.

Cependant, des modifications doivent être apportées à ce programme-jouet dans la perspective d'une utilisation plus professionnelle. La plus urgente d'entre-elles — et sans doute la plus difficile à réaliser — concerne la représentation des points du plan hyperbolique complexe par des couples de nombres flottants. Cette représentation montre rapidement ses limites dès que la racine de l'arbre se rapproche du bord du disque, ce qui se produit typiquement lorsque la partie de l'arbre affichée au centre de l'écran est éloignée de la racine. Dans cette situation, la racine se trouve quasiment sur le bord du disque, et la moindre erreur d'arrondi sur sa position se répercute de manière importante sur les positions des noeuds de l'arbre, qui peut alors s'afficher de façon incorrecte, voire pas du tout. Une solution à ce problème consiste sans doute à passer à une représentation plus algébrique en utilisant l'un des pavages standards du plan hyperbolique complexe.

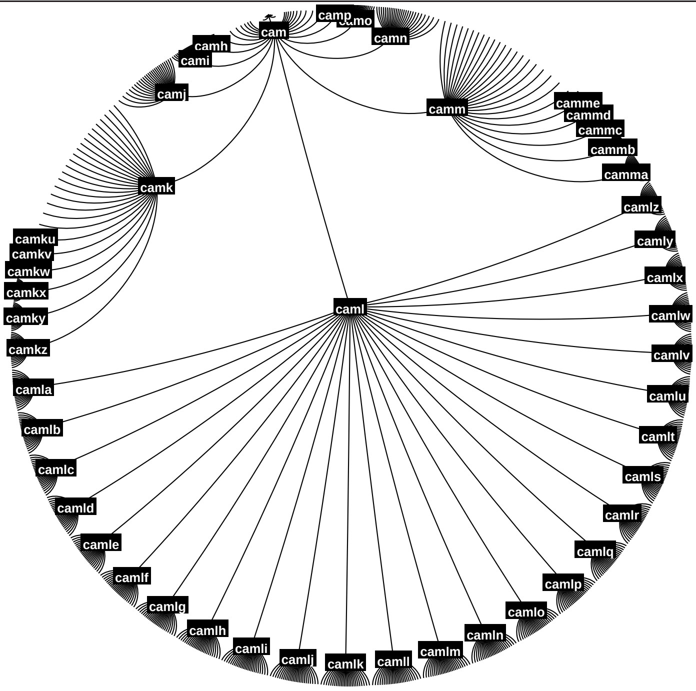


FIG. 4 – *Un extrait du dictionnaire des dictionnaires*

6. Conclusion

Nous avons présenté une application permettant d'afficher des arborescences volumineuses — voire infinies — en nous servant des propriétés géométriques d'un espace non euclidien.

Dans la pratique, cette méthode de représentation des arbres s'est révélée commode et agréable à utiliser. Bien que la géométrie du plan hyperbolique complexe soit peu familière à nos sens habitués à un monde euclidien, la prise en main de l'interface s'effectue très rapidement, et la navigation dans les structures arborescentes affichées de cette façon s'avère facile et intuitive.

Un des avantages de cette méthode de représentation vient de ce qu'elle tend naturellement à privilégier la lisibilité des éléments situés vers le centre de l'écran, en les espaçant davantage que les éléments situés au bord du disque. Il devient ainsi très naturel d'aller chercher la partie de l'arbre qui nous intéresse pour la placer au centre de l'écran, reléguant par le même geste d'autres parties moins intéressantes vers la périphérie.

Un autre avantage de cette représentation est que le passage d'une vue à une autre s'effectue avec une grande continuité, sans qu'il y ait de changement brusque dans la disposition des éléments affichés à l'écran. Au cours de la navigation dans l'arbre, les nouveaux éléments apparaissent naturellement dans la direction du déplacement, tandis que les éléments déjà visités s'éloignent vers la périphérie pour disparaître à leur tour.

Enfin, ce programme montre au cours de son utilisation qu'il est capable d'afficher à l'écran bien plus de noeuds que ne le permettent les techniques conventionnelles de tracé d'arbre en géométrie euclidienne, sans pour autant que cela nuise à la lisibilité.

Une application prometteuse de cette méthode de représentation pourrait être la réalisation d'un plan de site WEB présenté sous la forme d'un arbre hyperbolique. Une telle présentation permettrait à l'utilisateur d'avoir une vue globale du site dès la première page, et les facilités de navigation qui lui seraient offertes lui permettraient d'effectuer une première orientation dans sa recherche avant de s'engouffrer finalement dans le dédale des pages WEB.

Une autre application possible est l'affichage des valeurs d'un langage fonctionnel tel que le langage Caml. Grâce à cette méthode, il serait possible de traiter le cas des valeurs cycliques sans avoir à limiter la profondeur d'affichage de manière arbitraire comme le font les algorithmes actuels. On pourrait également raffiner la méthode d'affichage de manière à traiter de manière spécifique les valeurs paresseuses, par exemple en dégelant les glaçons qui les contiennent au fur et à mesure que ceux-ci s'approchent du centre de l'écran.

Références

[1] J. Lamping, R. Rao, P. Pirolli, A focus+context technique based on hyperbolic geometry for visualizing large hierarchies, in *Proceedings of ACM* SIGCHI Conference on Human factors in Computing systems. ACM, May 1995.

[2] P. Weis, X. Leroy, Le langage Caml (2^e édition). Dunod, 1999.

[3] X. Leroy, The Objective Caml system (release 2.04), Documentation and user's manual. `http://caml.inria.fr/ocaml/htmlman/`

[4] X. Leroy, Manifest types, modules, and separate compilation. POPL 94.

[5] X. Leroy, Applicative functors and fully transparent higher-order modules. POPL 95.

[6] M. Berger, Géométrie (tome 2). Nathan, 1990.

[7] P. Dolbeault, Analyse complexe. Collection Maîtrise de mathématiques pures. Masson, 1990.