Sphères, grands cercles et parallèles *

Denis Roegel

LORIA
Campus scientifique
BP 239
54506 Vandœuvre-lès-Nancy cedex
roegel@loria.fr

Résumé. Chaque domaine a ses archétypes graphiques et les sphères représentent des éléments incontournables de domaines tels que la géographie ou l'astronomie. Or, en examinant de nombreuses publications, il nous est apparu que les sphères étaient souvent incorrectement dessinées, eu égard à leurs composantes comme les grands cercles et les parallèles. Cet article examine quelques techniques METAPOST simples pour remédier à ces problèmes.

Abstract. *Each domain has its graphical archetypes and spheres are unavoidable components of domains such as geography or astronomy. However, when perusing a number of publications, we noticed that spheres were often incorrectly drawn, with respect to their features such as great circles and parallels. This article examines several simple METAPOST techniques that remedy these problems.*

1. Introduction

Les sphères et leurs composantes (grands cercles, méridiens, parallèles) forment des illustrations typiques dans certaines disciplines telles que la géographie ou l'astronomie. Ainsi, pour représenter le déplacement du soleil sur la voûte étoilée, on représentera souvent une sphère avec l'équateur céleste et l'écliptique, la trajectoire apparente du soleil sur cette sphère. Dans certains domaines, les sphères illustrent des projections, que ce soit en cartographie, en gnomonique, ou ailleurs. La représentation figurée d'une sphère dans une publication est bien sûr elle-même une projection.

Nous examinons ici le cas le plus simple qui est celui des sphères représentées en projection parallèle sur un plan. Dans ce cas, la projection se fait selon des

* Cet article est issu d'une présentation faite lors de la journée GUTenberg du 9 octobre 2006 à Paris. L'auteur remercie Jacques André pour ses remarques et corrections.

parallèles. Nous supposerons aussi, pour simplifier, que le plan de projection est orthogonal à la direction de projection, bien qu'une partie de nos conclusions soient indépendantes de cette hypothèse.

Nous considérons plus précisément le problème suivant : il s'agit de dessiner une sphère, avec un équateur, des méridiens, d'autres grands cercles, des parallèles, le tout avec des pointillés corrects.

Pour bien comprendre les éventuelles difficultés de cette tâche, il est utile de revoir les principes généraux des projections usuellement employées.

2. Projections

Les principales projections sont illustrées dans la figure 1. On y a représenté la projection de l'équateur, du pôle nord et de l'un des points dont la projection se fait selon une droite tangente à la sphère.

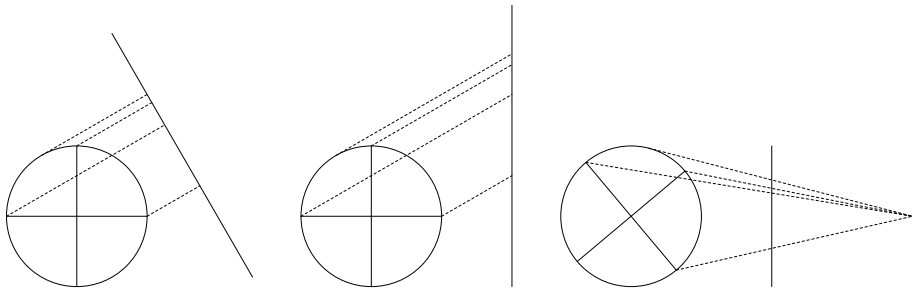


FIG. 1 – Projections orthogonale (à gauche), oblique (au milieu) et perspective (à droite).

3. Le traitement du problème dans la littérature

Un parcours de la littérature, que ce soit sur papier ou sur internet, est une source de surprises. Il apparaît qu'une majorité des ouvrages consultés représentent les sphères d'une manière contradictoire, si tant est que l'on suppose que la projection se fait sur un plan et soit selon des parallèles, soit de manière perspective. Ces hypothèses sont tout à fait naturelles.

Les problèmes sont tous confinés à des figures n'ayant pas été dessinées par projection. Ainsi, outre le fait que nombre de ces figures ne représentent pas

les cercles projetés selon des ellipses, les problèmes manifestés par la plupart des figures imprimées concernent la position de certains points, notamment les pôles. Ainsi, dans le cas des projections de la figure 1, lorsque l'équateur est transformé en une ellipse, les pôles ne devraient pas être placés à la périphérie de la sphère projetée, alors que cela est malheureusement souvent le cas sur les représentations imprimées.

Nous donnons à titre indicatif quelques références d'ouvrages où les sphères posent problème, avec un exemple de page, ce qui permettra au lecteur intéressé de les retrouver :

- W. M. Smart : *Celestial Mechanics*, New York : Longmans, 1953, p. 24.
- Derek J. Price : *The equatorie of the planetis*, Cambridge : the University press, 1955, p. 96.
- John D. North : *Richard of Wallingford*, Oxford : Clarendon Press, 1976, vol. 3, p. 152.
- René R. J. Rohr : *Sundials : History, Theory, and Practice*, New York : Dover publications, 1996, p. 25.
- Gianni Pascoli : *Éléments de mécanique céleste*, Paris : Masson, 1997, p. 12.
- Raymond d'Hollander : *L'astrolabe : histoire, théorie et pratique*, Paris : Institut océanographique, 1999, p. 26.
- Denis Savoie : *La gnomonique*, Paris : Les Belles lettres, 2001, p. 44.
- Denis Savoie : *Cosmographie*, Paris : Belin-Pour la science, 2006, p. 17.

Cela dit, certains ouvrages prennent tout de même garde à ne pas placer les pôles sur le cercle limite, c'est notamment le cas dans l'œuvre classique d'Otto Neugebauer : *A History of Ancient Mathematical Astronomy*, New York : Springer, 1975, p. 1408.

De nombreux sites ne sont pas en reste, y compris des pages dépendant de l'Observatoire de Paris-Meudon (`http://media4.obspm.fr`) ou de l'Institut de Mécanique Céleste et de Calcul des Éphémérides (`http://www.imcce.fr`) qui arborent des représentations critiquables.

Les raisons de la perpétuation de ces erreurs ne sont pas totalement claires et il semble qu'il s'agisse à la fois d'une habitude, d'une certaine forme de facilité, et — dans certains cas — du fruit de la sous-traitance des figures par les auteurs.

4. Une approche du problème avec METAPOST

Quoique notre application soit très simple, il ne nous a pas semblé qu'elle ait déjà été traitée avec le logiciel METAPOST, ou avec d'autres outils graphiques

de TEX comme `PSTricks`.[1] Les extensions de ce dernier système possèdent un certain nombre de facilités pour la représentation d'objets tridimensionnels, mais la représentation d'objets volumiques efface les parties cachées par superposition, et ne réalise pas un véritable calcul de visibilité.

L'une des difficultés de la représentation des sphères a trait aux pointillés. Ceux-ci sont traditionnellement utilisés pour représenter les parties cachées. Il est donc nécessaire de faire commencer et s'achever ces pointillés aux bons endroits, ceci impliquant le plus souvent le calcul d'intersections.

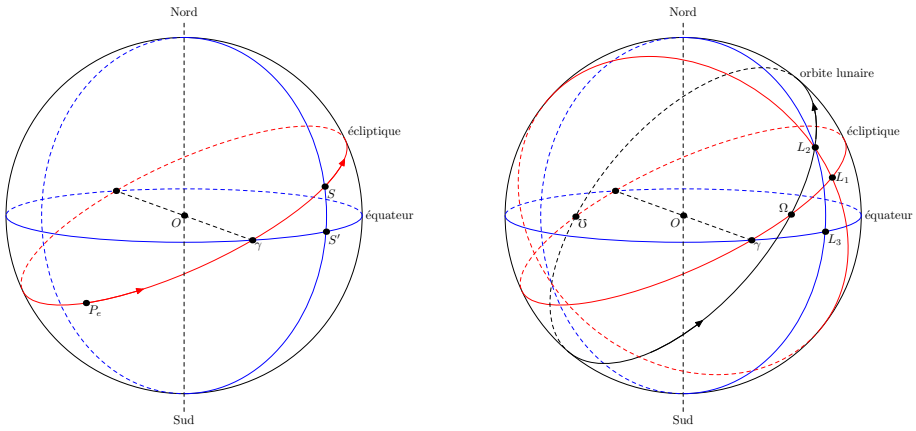


FIG. 2 – Deux tracés de sphères violant les propriétés des projections parallèles sur un plan. Les pôles sont ici placés à la périphérie des sphères, alors qu'ils devraient figurer légèrement à l'intérieur, étant donné l'angle sous lequel le plan de l'équateur est vu.

C'est en concevant une figure pour une conférence en astronomie que nous avons, dans un premier temps, commis la même erreur que nos prédécesseurs, tant le réflexe des « pôles sur le cercle » semble être ancré dans les habitudes. La figure 2 représente ces premiers essais, typiques des figures que l'on trouve un peu partout. La figure 3 illustre comment les sphères auraient dû être représentées. La position des pôles est ici calculée de manière exacte, tant pour les pôles de l'équateur (N et S) que pour ceux de l'écliptique (N^* et S^*). De plus l'angle entre les plans de l'équateur et de l'écliptique est lui aussi correctement représenté (23.5). Par contre, dans le cas de l'orbite lunaire, nous avons volontairement accru l'angle que fait celle-ci avec le plan de l'écliptique.

1 Les Cahiers GUTenberg ont publié deux numéros spéciaux consacrés à METAPOST (numéro 41, novembre 2001) et à PSTricks (numéro 16, février 1994).

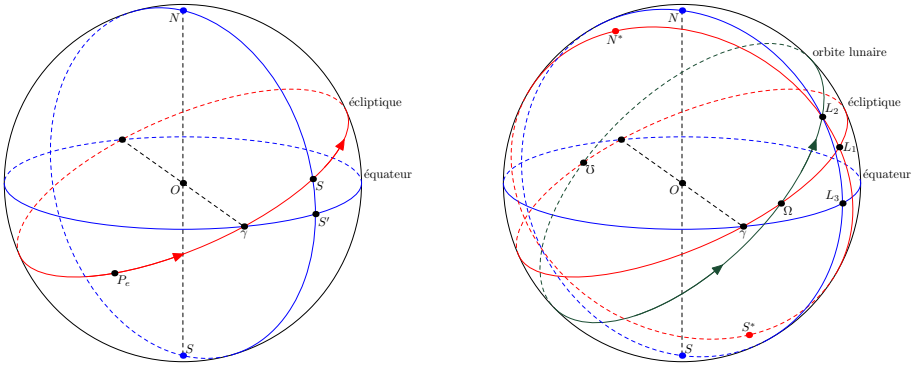
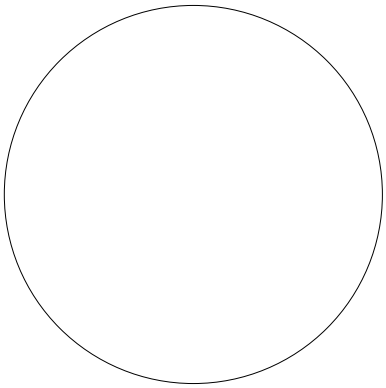


FIG. 3 – Deux tracés corrects des plans de l'équateur, de l'écliptique, des pôles et des méridiens. L'inclinaison de l'orbite lunaire a été volontairement exagérée.

Nous allons maintenant détailler comment les figures correctes ont été obtenues et nous nous confinerons au cas des projections orthogonales. Les constructions seront réalisées en METAPOST, mais rien n'empêche la transposition des techniques utilisées ici à d'autres langages.

4.1. La projection de la sphère

La projection orthogonale de la sphère est un cercle dont le diamètre est celui de la sphère. Nous supposerons pour simplifier que le cercle est centré à l'origine.



```
r=5cm;
draw fullcircle scaled 2r;
```

4.2. Définition de vecteurs

Afin de contrôler précisément la projection, nous définissons tout d'abord un type vecteur. METAPOST ne comporte pas de type vecteur, mais un type `color` à trois composantes numériques que nous déguisons en vecteur. L'accès aux composantes des vecteurs est réalisée par `Xp`, `Yp` et `Zp`. Nous définissons ensuite quelques opérations élémentaires sur ces vecteurs comme le produit scalaire (`dotproduct`), le produit vectoriel (`vecproduct`) et la construction d'un vecteur unitaire.

```
let vector=color;
let Xp=redpart; let Yp=greenpart; let Zp=bluepart;

def dotproduct(expr Vi,Vj)=
  (Xp(Vi)*Xp(Vj)+Yp(Vi)*Yp(Vj)+Zp(Vi)*Zp(Vj))
enddef;

def vecproduct(expr Vi,Vj)= (Yp(Vi)*Zp(Vj)-Zp(Vi)*Yp(Vj),
                             Zp(Vi)*Xp(Vj)-Xp(Vi)*Zp(Vj),
                             Xp(Vi)*Yp(Vj)-Yp(Vi)*Xp(Vj))
enddef;

def norm(expr V)= sqrt(dotproduct(V,V)) enddef;
def normed(expr V)= (V/norm(V)) enddef;
```

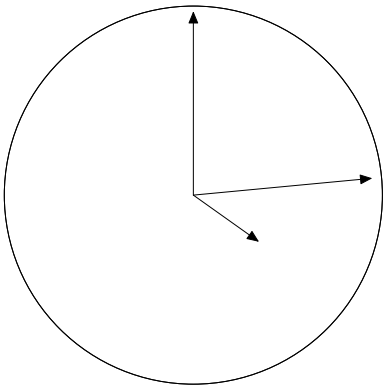
4.3. Orientation dans l'espace

Avant d'effectuer la projection, nous orientons la sphère dans l'espace. Plus précisément, nous construisons trois vecteurs $\overrightarrow{V_1}$, $\overrightarrow{V_2}$, $\overrightarrow{V_3}$ à partir des vecteurs du repère orthonormé. Nous n'employons que deux angles, et conservons ainsi le caractère vertical de la projection de l'un des vecteurs. θ est l'angle par lequel $\vec{\imath}$ est tourné autour de $\vec{k}$, ce qui produit $\overrightarrow{V_1}$. ϕ est l'angle par lequel $\vec{k}$ est tourné autour de $\overrightarrow{V_1}$, ce qui produit $\overrightarrow{V_2}$. $\overrightarrow{V_3}$ est le produit vectoriel de $\overrightarrow{V_1}$ et $\overrightarrow{V_2}$ et est orienté en direction de l'observateur. En définitive, $\overrightarrow{V_1}$ représente le vecteur du plan de projection orienté vers la droite et $\overrightarrow{V_2}$ celui orienté vers le haut. Les figures suivantes ont été obtenues avec $\theta = 70$ et $\phi = -15$.

```
vector V[]; % tableau de vecteurs
theta=70;phi=-15;
V1=(cosd theta,sind theta,0);
V2=(sind(phi)*sind(theta),-sind(phi)*cosd(theta),cosd(phi));
V3=vecproduct(V1,V2);
```

4.4. La projection

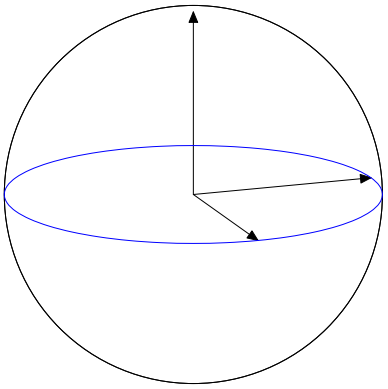
La projection proprement dite est très simple à réaliser, puisqu'il suffit de déterminer les composantes d'un vecteur de l'espace dans la base $(\overrightarrow{V_1}, \overrightarrow{V_2}, \overrightarrow{V_3})$, ce que le produit scalaire nous donne immédiatement. Seules les deux premières composantes nous intéressent, puisque $\overrightarrow{V_3}$ est parallèle à la direction de projection. Une fonction `project` nous permet d'écrire naturellement cette projection qui n'utilise donc pas le troisième vecteur :



```
def project(expr V,Va,Vb)=
  (dotproduct(V,Va),
   dotproduct(V,Vb))
enddef;
z0=(0,0);
z1=project((r,0,0),V1,V2);
z2=project((0,r,0),V1,V2);
z3=project((0,0,r),V1,V2);
drawarrow z0--z1;
drawarrow z0--z2;
drawarrow z0--z3;
```

4.5. Construction de l'équateur

Nous pouvons maintenant tracer un grand cercle, celui de l'équateur. Son équation est très simple : ce sont tous les points $(r\cos t, r\sin t, 0)$ pour $0 \leqslant t < 360$, t étant exprimé en degrés. La macro `f_equ` correspond à cette expression et nous formons la courbe projetée en reliant les projections des points à intervalles réguliers, ici de 10 en 10 degrés.



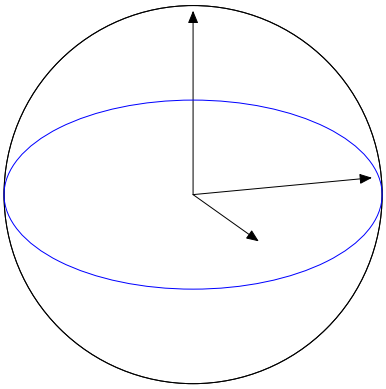
```
def f_equ(expr r,t)=
  (r*cosd(t),r*sind(t),0)
enddef;

path equateur;
equateur=
  project(f_equ(r,0),V1,V2)
  for t=10 step 10 until 350:
   ..project(f_equ(r,t),V1,V2)
  endfor ..cycle;
draw equateur withcolor blue;
```

4.6. Simplification de l'équateur

L'équateur est maintenant représenté par une courbe construite à partir d'un grand nombre de points. Toutefois, cette courbe est une ellipse et nous pouvons en obtenir une très bonne approximation en construisant l'ellipse à partir de `fullcircle`. (Il ne s'agit que d'une approximation, car `fullcircle` n'est pas exactement un cercle.)

La construction d'une ellipse à partir d'un cercle se fait de la manière suivante, où nous donnons le demi grand-axe, le demi petit-axe et l'angle de l'ellipse. Le tracé correct de l'ellipse nécessite la connaissance de la valeur des deux axes, ce qui n'est pas ici le cas.



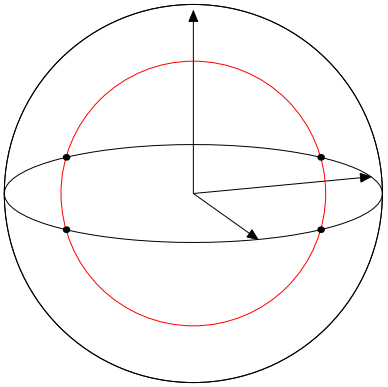
```
def ellipse(expr ra,rb,an)=
  (fullcircle xscaled 2ra
              yscaled 2rb
              rotated an)
enddef;

draw ellipse(r,.5r,0);
```

4.7. Détermination des éléments de l'ellipse

Pour obtenir les éléments de l'ellipse (axes et orientation), les paramètres de la projection peuvent être utilisés, ou bien l'on peut tout simplement se contenter de mesurer ces éléments sur l'ellipse construite point par point. Ceci peut se faire de la manière suivante :

- un cercle est d'abord superposé à l'ellipse ;
- les quatre intersections de ce cercle avec l'ellipse sont déterminées, ce qui peut nécessiter le redimensionnement du cercle ;
- les intersections fournissent facilement les directions des axes ;
- ces axes sont ensuite mesurés ;
- enfin, l'ellipse est reconstruite de manière plus économique.



Nous allons maintenant examiner la réalisation de cette procédure de manière plus détaillée.

4.7.1. Orientation de l'ellipse

Pour déterminer l'orientation de l'ellipse, nous nous servons de la macro `ellipse_major_angle` ci-dessous qui prend un chemin `p` représentant une ellipse de demi-grand axe `a` centrée à l'origine. Une simple dichotomie cherche un demi-cercle de rayon `rc` ayant une intersection non nulle avec l'ellipse. Ensuite, deux intersections (`pi1`, `pi2`) sont obtenues avec `intersectionpoint`, en découpant soigneusement le demi-cercle. Ces deux intersections en fournissent deux autres (`pi3`, `pi4`) par symétrie.

L'orientation de l'ellipse est obtenue en cherchant deux intersections, `pi5` et `pi6`. L'une de ces intersections l'est avec le grand axe, l'autre avec le petit axe.

```
vardef ellipse_major_angle(expr p,a)=
  save pa,pc,pi,ra,rb,rc,an;path pc[];pair pa,pi[];ra=.5a;rb=a;
  forever: %================ dichotomie ===============
    rc:=.5[ra,rb];pc0:=subpath(0,4) of fullcircle scaled 2rc;
    pa:=pc0 intersectiontimes p;exitif pa<>(-1,-1);ra:=rc;
  endfor;
  %======= calcul de deux intersections ======
  pi1=p intersectiontimes pc0;
  pc1=subpath(0,ypart(pi1)-0.01) of pc0;
  pc2=subpath(ypart(pi1)+0.01,length(pc0)) of pc0;
  pi1:=p intersectionpoint pc0;pi2:=p intersectiontimes pc1;
  if pi2=(-1,-1):pi2:=p intersectionpoint pc2;
  else:pi2:=p intersectionpoint pc1;fi;
  pi3=pi1 rotated 180;pi4=pi2 rotated 180; % autres intersections
  %======= orientation ======
  pi5=p intersectionpoint (origin--(unitvector(pi2-pi1)*2a));
```

```
  pi6=p intersectionpoint (origin--(unitvector(pi1-pi4)*2a));
  if arclength(origin--pi5)>arclength(origin--pi6):an=angle(pi1-pi2);
  else:an=angle(pi1-pi4);fi;
  an % résultat de la macro
enddef;
```

4.7.2. Le petit axe de l'ellipse

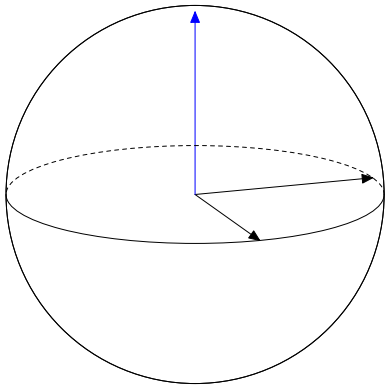
La macro `ellipse_minor_axis` prend un chemin `p` représentant une ellipse de demi-grand axe `a` centrée à l'origine et dont le grand axe est orienté suivant l'angle `an`. La macro détermine simplement l'intersection située à angle droit du grand axe et mesure sa distance du centre de l'ellipse.

```
vardef ellipse_minor_axis(expr p,a,an)=
  save pa;pair pa;
  pa=p intersectionpoint (origin--(dir(an+90)*2a));
  arclength(origin--pa) % résultat
enddef;
```

Ces deux macros permettent donc de déterminer tous les paramètres nécessaires au tracé économique (c'est-à-dire non point par point) d'une ellipse.

4.8. Les pointillés de l'équateur

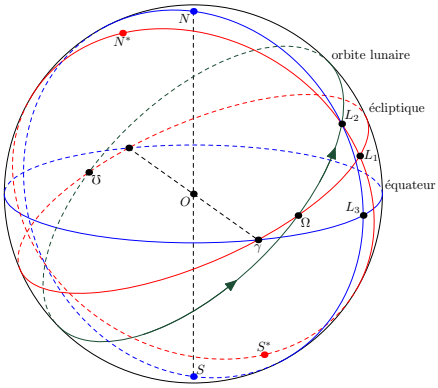
Les pointillés de l'équateur correspondent à l'une des moitiés de l'ellipse et les deux moitiés sont jointes par le grand axe. Il suffit donc de découper l'ellipse en deux parties et de tracer l'une en traits pleins, l'autre en pointillés. L'ellipse renvoyée par la macro `ellipse` est une courbe paramétrée où le paramètre va de 0 à 8 (le cercle de base comporte huit points) et les chemins de 0 à 4 et 4 à 8 en sont extraits.



```
path pa,pb,pc;
pa=ellipse(r,rb,0);
pb=subpath(0,4) of pa;
pc=subpath(4,8) of pa;
draw pb dashed evenly; % caché
draw pc; % visible
```

4.9. Grands cercles

Le même principe est utilisé pour tous les grands cercles, la seule difficulté étant la détermination d'une équation pour ces grands cercles. Les macros utilisées sont paramétrées pour pouvoir choisir le côté des pointillés.



Certains cercles sont déterminés par des contraintes. Ainsi, dans le dessin ci-contre, nous nous sommes donnés le point L_1 sur l'écliptique, le méridien de l'écliptique passant par L_1 a ensuite été construit, conduisant à la détermination du point L_2 sur l'orbite lunaire. Ces intersections ont été obtenues par intersection des projections, mais l'intersection dans l'espace a ensuite été retrouvée par la connaissance des courbes. Enfin, le méridien équatorial passant par L_2 a été tracé, permettant d'obtenir L_3.

4.9.1. Contraintes

La prise en compte de contraintes utilise avantageusement la macro `rotatearound` qui permet de tourner un vecteur autour d'un autre.

```
% tourne Va autour de Vb de l'angle a
vardef rotatearound(expr Va,Vb,a)=
  save v;
  vector v[];
  v0=normed(Vb);
  v1=dotproduct(Va,v0)*v0;
  v2=Va-v1;
  v3=vecproduct(v0,v2);
  v4=v2*cosd(a)+v3*sind(a)+v1;
  v4 % résultat
enddef;
```

Ainsi, pour le cas de la courbe représentant l'écliptique, dont l'équation est déterminée par la fonction

```
def f_ecliptic(expr t)=
  (a*(cosd t,sind t * cosd ec_angle,sind t * sind ec_angle))
enddef;
```

où `ec_angle` est l'obliquité du plan de l'écliptique (23.5), nous commençons par déterminer le pôle Nord (N^*) de l'écliptique :

```
vector North,North_Ec;
North=a*(0,0,1);
North_Ec=rotatearound(North,(1,0,0),ec_angle);
```

Le point L_1 étant choisi sur l'écliptique, le méridien passant par L_1 et N^* est déterminé par la donnée des deux vecteurs $\overrightarrow{ON^*}$ et $\overrightarrow{OL_1}$, tout point du méridien étant obtenu par rotation de $\overrightarrow{OL_1}$ autour du vecteur perpendiculaire à $\overrightarrow{OL_1}$ et $\overrightarrow{ON^*}$. La macro suivante, paramétrée par le point A de l'écliptique et un angle t, permet donc de parcourir ce méridien :

```
def f_ec_meridian(expr t,A)=
  (A*cosd(t)+North_Ec*sind(t))
enddef;
```

4.9.2. Projection inverse

Le principe de la « projection inverse » est très simple et nous ne ferons que l'esquisser. Ainsi, pour déterminer L_2 à partir de L_1 dans la figure précédente, nous avons construit d'une part le grand cercle passant par L_1 et N^* comme expliqué plus haut (`ec_meridian`), et d'autre part l'orbite lunaire (`moon`) en appliquant des principes analogues. L'intersection de ces deux courbes projetées a été calculée tout à fait normalement :

```
Lp2=moon intersectionpoint ec_meridian;
```

Nous avons, pour simplifier, supposé qu'`intersectionpoint` renvoyait ici la bonne intersection, ce qui n'est pas toujours le cas.

Maintenant, le point de l'espace L_2 est une combinaison linéaire déterminée de deux vecteurs formant une base du plan de l'orbite lunaire. Ces deux vecteurs peuvent être déterminés à partir de l'équation de l'orbite lunaire et nous les appelons `moon_x` et `moon_y`. On a donc

```
L2=m_x*moon_x+m_y*moon_y;
```

où `m_x` et `m_y` sont des inconnues scalaires. Celles-ci peuvent être déterminées en projetant l'équation précédente, car une projection parallèle est une transformation linéaire :

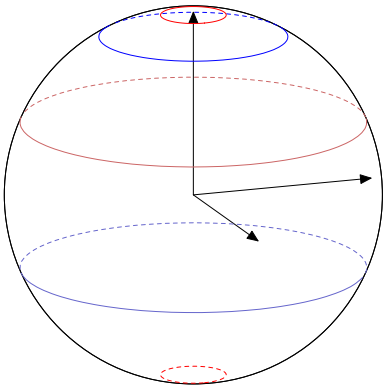
```
Lp2=m_x*project(moon_x,V1,V2)+m_y*project(moon_y,V1,V2);
```

Cette dernière équation définit `m_x` et `m_y` à partir de `L2p` (point du plan) et définit donc du même coup `L2` (point de l'espace).

Une fois que L_2 était connu, nous avons pu l'utiliser pour obtenir L_3 d'une manière analogue.

4.10. Parallèles

Le cas des grands cercles était relativement simple, car ceux-ci étaient toujours à moitié visibles et à moitié invisibles, la limite se situant sur le grand axe de l'ellipse. Il n'en est plus de même pour les autres cercles des sphères. Nous examinerons ici uniquement le cas des parallèles.



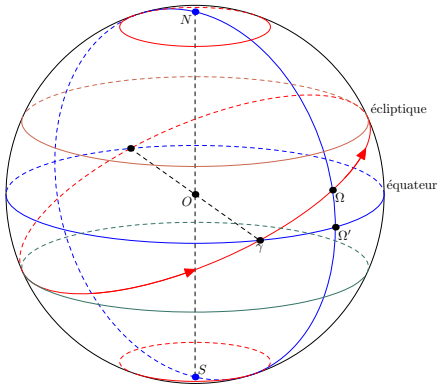
Les parallèles :
- ne sont pas forcément autant visibles que cachés ;
- peuvent être entièrement visibles ou cachés ;
- ont une limite visible/caché qui ne se situe pas sur le grand axe.

Pour dessiner correctement les parallèles, il faut donc déterminer les limites entre la partie visible et la partie cachée d'un parallèle.

Les limites de visibilité sont déterminées par l'intersection entre le plan perpendiculaire à la direction de visée ($\overrightarrow{V_3}$), et le cercle représentant le parallèle. Cette intersection peut consister en :

- 0 point : le parallèle est alors totalement visible ou totalement caché ;
- 2 points : il y a à la fois une partie cachée et une partie visible ;

– 1 point : c'est le cas limite entre les deux précédents.



Une fois les intersections obtenues dans l'espace, celles-ci sont traduites en angles et les deux arcs sont tracés séparément à partir de ces angles. La macro `draw_parallel` est définie dans la figure 4.

L'équation d'un parallèle à la latitude ϕ sera la suivante :

```
def f_parallel(expr r,theta,phi)=
  (r*cosd phi * cosd theta,r*cosd phi * sind theta,r*sind phi)
enddef;
```

5. Conclusion

Après avoir constaté que de nombreuses sphères étaient incorrectement représentées dans la littérature, nous avons analysé le problème plus finement et avons écrit quelques commandes METAPOST pour produire des dessins corrects. Il ne nous reste qu'à espérer que ce travail pourra contribuer, fut-ce indirectement, à une amélioration du réalisme des sphères telles qu'elles sont employées en cosmographie et ailleurs.

Il serait d'autre part intéressant d'étendre d'autres *packages* graphiques par de telles fonctionnalités, toujours dans le but de favoriser leur utilisation. Le *package* `PSTricks` pourrait ainsi être utilement complété, ce qui aurait en outre l'avantage de permettre une comparaison avec notre propre réalisation.

```
% phi=latitude, col=couleur, side=1 ou -1 suivant les pointillés
vardef draw_parallel(expr phi,col,side)=
  save p;path p[];p0=project(f_parallel(a,0,phi),V1,V2)
  for t=0 step 10 until 360 :..project(f_parallel(a,t,phi),V1,V2) endfor;
  % on cherche maintenant les intersections de ce parallèle
  % avec le plan de projection :
  % plan :      V3x*x+V3y*y+V3z*z=0
  % parallèle : x=r*cos(phi)*cos(theta), y=r*cos(phi)*sin(theta), z=r*sin(phi)
  % on cherche theta :
  save A,B,C,X,Y,ca,cb,cc,delta,nx,tha,thb;
  numeric X[],Y[];ca=Xp(V3);cb=Yp(V3);cc=Zp(V3);
  if cb=0:X1=-(cc/ca)*sind(phi)/cos(phi);nx=1;
  else:
    A=1+(ca/cb)**2;B=2*ca*cc*sind(phi)/(cb*cb);
    C=((cc/cb)*sind(phi))**2-cosd(phi)*cosd(phi);delta=B*B-4A*C;
    if delta<0:nx=0;% pas d'intersection
    else:
      X1=((-B-sqrt(delta))/(2A))/cosd(phi); % = cos(theta)
      X2=((-B+sqrt(delta))/(2A))/cosd(phi); % = cos(theta)
      Y1=-((ca*X1+cc*sind(phi)/cosd(phi))/cb); % = sin(theta)
      Y2=-((ca*X2+cc*sind(phi)/cosd(phi))/cb); % = sin(theta)
      tha=angle(X1,Y1);thb=angle(X2,Y2);nx=2;
    fi;
  fi;
  if nx=0: % parallèle entièrement (in)visible
    if side=1:draw p0 withcolor col;
    else:draw p0 withcolor col dashed evenly;fi;
    message "PAS D'INTERSECTION";
  elseif nx=1:X10=angle(X1,1+-+X1);X11=360-X10;
  else: % cas général
    if tha<thb:X10=tha;X11=thb;else:X10=thb;X11=tha;fi;
  fi;
  if nx>0: % détermination des deux chemins
    p1=project(f_parallel(a,X10,phi),V1,V2)
    for t=X10+1 step 10 until X11:..project(f_parallel(a,t,phi),V1,V2)
    endfor;
    p2=project(f_parallel(a,X11,phi),V1,V2)
    for t=X11+1 step 10 until X10+360:..project(f_parallel(a,t,phi),V1,V2)
    endfor;
    % tracé des deux chemins
    if side=1:draw p1 withcolor col;
    else:draw p1 withcolor col dashed evenly;fi;
    if side=1:draw p2 withcolor col dashed evenly;
    else:draw p2 withcolor col;fi;
  fi;
enddef;
```

FIG. 4 – Code de tracé d'un parallèle.