Spheres, great circles and parallels*

Denis Roegel

Abstract

Each domain has its graphical archetypes. In particular, 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

The spheres and their components (great circles, meridians, parallels) make up the typical illustrations in certain fields such as geography or astronomy. For instance, the motion of the Sun in the sky will often be represented as a sphere with the celestial equator, the ecliptic and the apparent path of the Sun on this sphere. In certain fields, spheres illustrate projections, be it in cartography, gnomonics, or elsewhere. The representations of spheres in publications are themselves projections.

Here we examine the simplest case: spheres represented in parallel projection on a plane. In that case, the projection is done along parallel lines. We will also assume, for simplification, that the projection plane is orthogonal to the projection direction, although part of our conclusions are independent of this assumption.

More precisely, the problem we consider is that of drawing a sphere, with an equator, meridians, other great circles, parallels, all of them with correct dashed lines.

In order to get a good understanding of the possible difficulties of this task, it is useful to review the general principles of the projections which are commonly used.

2 Projections

The main projections are illustrated in figure 1. We have represented the projections of the equator, of the North pole and of one of the points whose projection follows a line which is tangent to the sphere.

3 How the problem is handled in the literature

A perusal of the literature, be it on paper or the Internet, is a source of surprises. Assuming that the projections are done on a plane and either along parallels or in a perspective manner, two totally natural assumptions, it appears that the majority of the books consulted represent the spheres in a contradictory way.

The problems are all confined to figures which have not been drawn by projection. For instance, aside from the fact that many of these figures do not represent the projected circles as ellipses, the problems displayed in most of the printed figures concern the position of certain points, in particular the poles. For instance, in the case of the projections of figure 1, when the equator is transformed in an ellipse, the poles should not be positioned at the periphery of the projected sphere, but this is unfortunately often the case on the printed representations.

To support our claim, we give a list of a few books where the spheres are problematic, with a page example, which will allow the interested reader to locate them:

- 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.

That said, some books take care not to put the poles on the limit circle, and this is in particular the case in Otto Neugebauer's classic *A History of Ancient Mathematical Astronomy*, New York: Springer, 1975, p. 1408.

A number of web sites are also faulty, for instance those of the Paris-Meudon observatory or of the *Institut de Mécanique Céleste et de Calcul des Éphémérides* (`http://www.imcce.fr`) which display objectionable representations.

The reasons for perpetuating these errors are not totally clear; it seems that it is a certain habit, perhaps a kind of laziness, and — in some cases — the result of the subcontracting of figures by the authors.

* Translation of "Sphères, grands cercles et parallèles," *Les Cahiers GUTenberg*, number 48, April 2007, pages 7–22. Reprinted with permission.

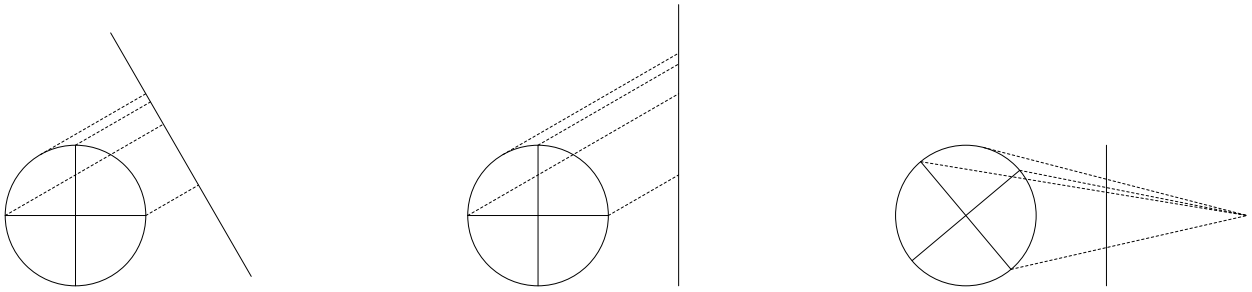


Figure 1: Orthogonal (left), oblique (center) and perspective (right) projections on a vertical plane.

4 A METAPOST approach

Although our application is very simple, it doesn't seem to have been handled with the METAPOST software, or with other graphical TEX tools such as `PSTricks`. The extensions of the latter system already provide a number of facilities for the representation of 3-dimensional objects, but the representation of objects in space obscures the hidden parts by overlaying them and therefore doesn't involve the computation of boundaries between visible and invisible parts.

One of the difficulties of the representation of spheres is related to dashed lines. Dashed lines are traditionally used for representing the hidden parts. It is therefore necessary to ensure that these lines start and end at the right places, and this task usually requires the computation of intersections.

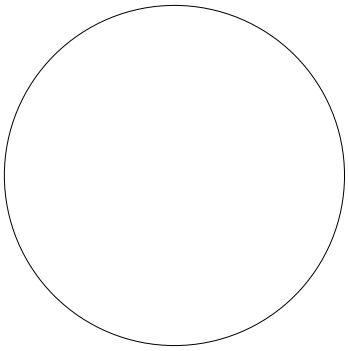
It is when designing a figure for a lecture in astronomy that we have, in the first place, made the same error as that of our predecessors; the reflex of "poles on the circle" was rooted in our habits. Figure 2 represents these first attempts, typical of the figures which are found almost everywhere. Figure 3 illustrates how the spheres should have been represented. The positions of the poles are here computed in an exact way, for the poles of the equator (N and S) as well as for those of the ecliptic (N^* and S^*). Moreover, the angle between the planes of the equator and the ecliptic is also correctly displayed ($23.5°$). In the case of the lunar orbit, however, we have intentionally increased the angle between that orbit and the plane of the ecliptic.

We will now describe how the correct figures were obtained, and we will restrict ourselves to the case of orthogonal projections. Our constructions will be in METAPOST, but nothing prevents the transposition of our techniques to other languages.[1]

4.1 The projection of the sphere

The orthogonal projection of the sphere is a circle whose diameter is that of the sphere. We will assume for simplification that the circle is centered at the origin.

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



4.2 Definition of vectors

In order to precisely control the projection, we first define a vector type. METAPOST does not provide such a type, but it has a `color` type with three numerical components which we disguise as a vector. Accessing the components of the vectors is done with `Xp`, `Yp` and `Zp`. We then define a few elementary operations on these vectors, like the dot product (`dotproduct`), the vector product (`vecproduct`) and the construction of a unit vector.

```
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))
```

[1] For an introduction to METAPOST, one can readily consult various tutorials on the web, the documentation available in most TEX distributions, or the second edition of the *LATEX Graphics Companion*.

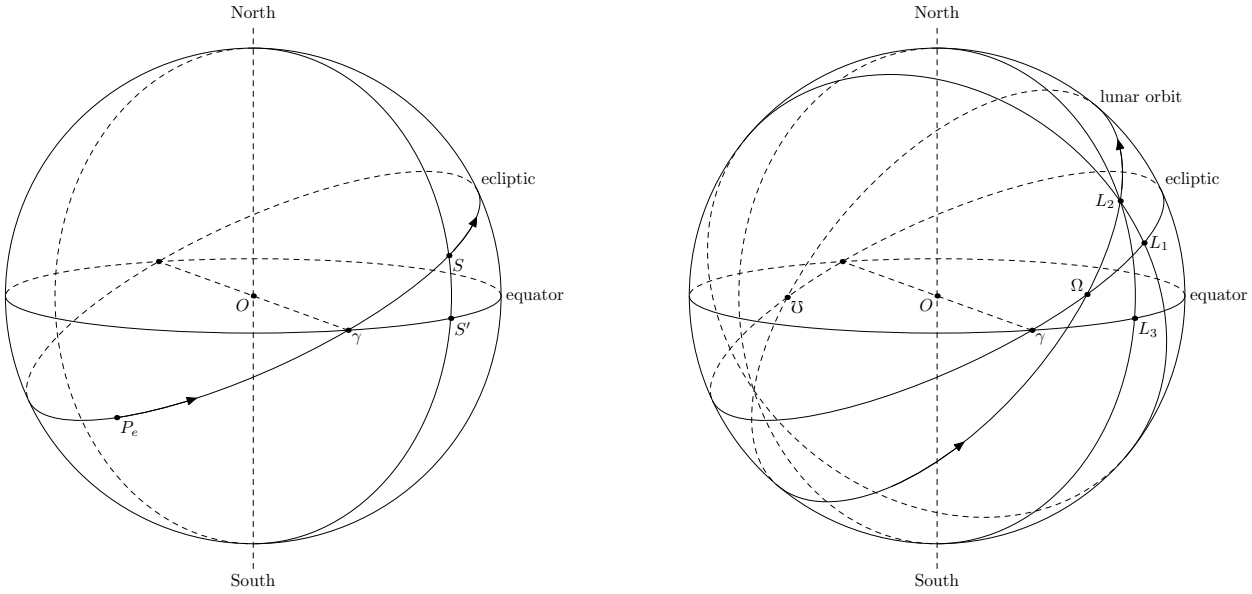


Figure 2: Two sphere drawings violating the properties of parallel projections on a plane. The poles are here put at the periphery of the spheres, although they should be located slightly inside of the spheres, given the angle under which the plane of the equator is seen.

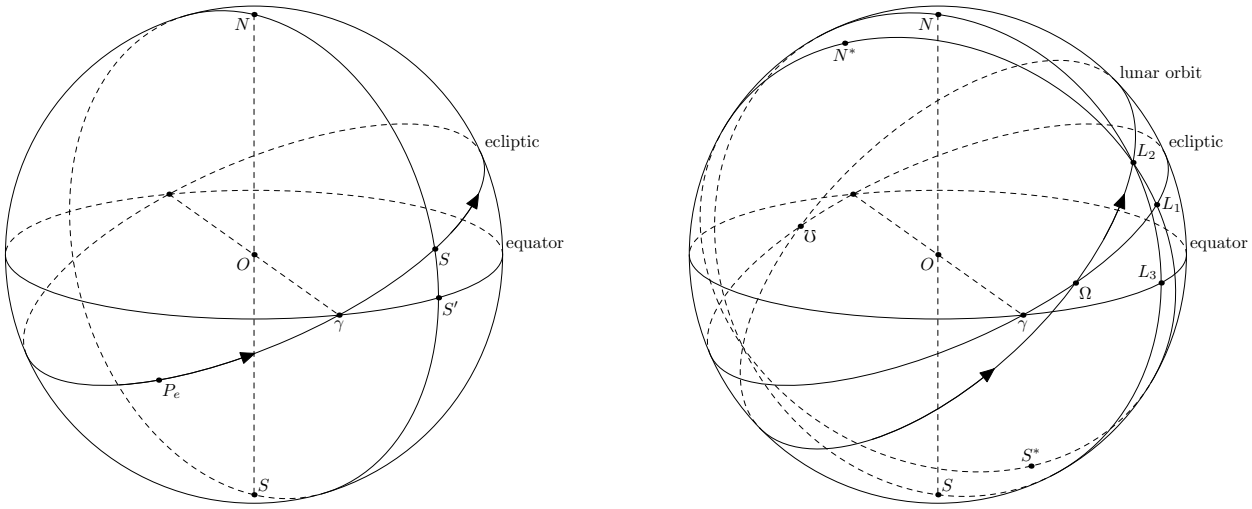


Figure 3: Two correct drawings of the planes of the equator and of the ecliptic, of the poles and of the meridians. The inclination of the lunar orbit has intentionally been magnified.

```
enddef;

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

4.3 Orientation in space

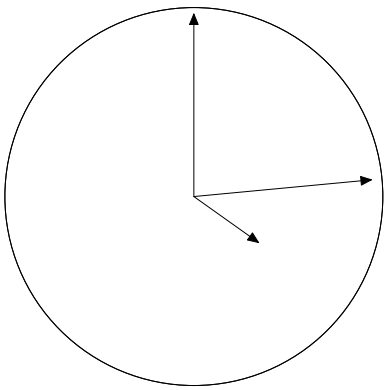
Before performing the projection, the sphere is oriented in space. More precisely, we construct three vectors $\overrightarrow{V_1}$, $\overrightarrow{V_2}$, $\overrightarrow{V_3}$ using the vectors of the orthonormal basis. We employ only two angles, and in that manner we maintain the vertical character of the projection of one of the vectors. θ is the angle by which $\vec{\imath}$ is rotated around $\vec{k}$, which produces $\overrightarrow{V_1}$. ϕ is the angle by which $\vec{k}$ is rotated around $\overrightarrow{V_1}$, which produces $\overrightarrow{V_2}$. $\overrightarrow{V_3}$ is the vector product of $\overrightarrow{V_1}$ and $\overrightarrow{V_2}$ and is oriented towards the observer. Finally, $\overrightarrow{V_1}$ represents the vector of the projection plane directed towards the right and $\overrightarrow{V_2}$ the one directed towards the top. The figures in the sequel were obtained with $\theta = 70$ and $\phi = -15$.

```
vector V[]; % vector array
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 The projection

The projection itself is very simple to achieve, as it is sufficient to determine the components of a vector in space in the $(\overrightarrow{V_1}, \overrightarrow{V_2}, \overrightarrow{V_3})$ base, something which is immediate with the dot product. Only the first two components are of interest to us, since $\overrightarrow{V_3}$ is parallel to the projection direction. A `project` function allows us to write this projection naturally, and this function therefore doesn't use the third vector:

```
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;
```

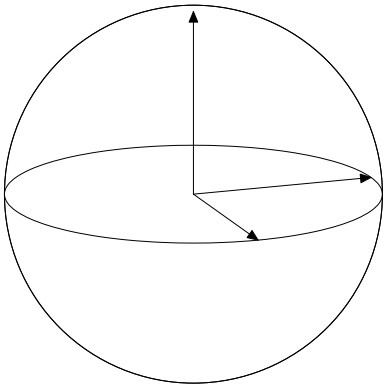


4.5 Construction of the equator

We can now draw a great circle, for instance the circle of the equator. Its equation is very simple: it is the set of points $(r\cos t, r\sin t, 0)$ for $0 \leq t < 360$, t being expressed in degrees. The `f_equ` macro corresponds to this expression and the projected curve is obtained by connecting the projections of points at regular intervals, here from 10 to 350 degrees.

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

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

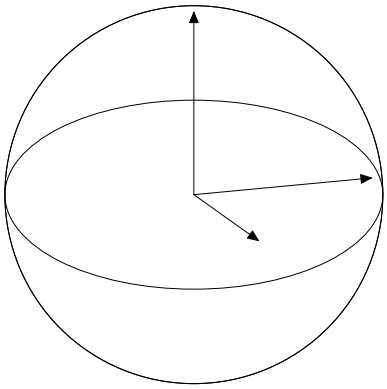


4.6 Simplification of the equator

The equator is now represented by a curve constructed from a large number of points. However, this curve should be an ellipse and we can obtain a very good approximation of it by constructing it using `fullcircle` instead. (It is only an approximation since `fullcircle` is not exactly a circle.)

The construction of an ellipse from a circle is done as follows, using the semi-major axis, the semi-minor axis and the angle of the ellipse. The correct drawing of the ellipse requires the knowledge of its two axes, which are not yet known in the above construction.

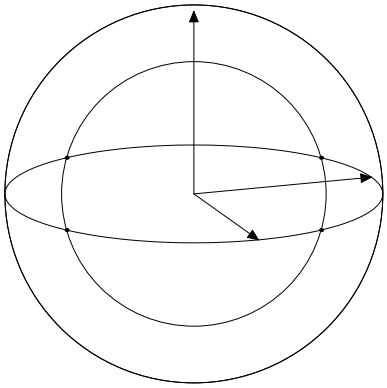
```
def ellipse(expr ra,rb,an)=
  (fullcircle xscaled 2ra yscaled 2rb rotated an)
enddef;
draw ellipse(r,.5r,0);
```



4.7 Determination of the elements of the ellipse

In order to obtain the elements of the ellipse (axes and orientation), the projection parameters can be used, or we can merely measure these elements on the ellipse as constructed pointwise. This can be done as follows:

- first, a circle is superimposed to the ellipse;
- the four intersections of this circle with the ellipse are determined (this may require the circle to be resized);
- the intersections easily provide the directions of the axes;
- these axes are then measured;
- finally, the ellipse is constructed in a more economical way.



We will now examine in more detail how this procedure is realized.

4.7.1 Orientation of the ellipse

In order to determine the orientation of the ellipse, we make use of the `ellipse_major_angle` macro below, which takes a path `p` representing an ellipse of semi-major axis `a` centered at the origin. A simple dichotomy looks for a half circle of radius `rc` with a non-void intersection with the ellipse. Then, two intersections (`pi1`, `pi2`) are obtained with the help of `intersectionpoint`, by carefully splitting the half-circle. By symmetry, these two intersections give two other intersections (`pi3`, `pi4`).

The orientation of the ellipse is obtained by locating two intersections, `pi5` and `pi6`. One of these intersections is with the major axis, the other with the minor axis.

```
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: %======== dichotomy ==========
    rc:=.5[ra,rb];
    pc0:=subpath(0,4) of fullcircle scaled 2rc;
    pa:=pc0 intersectiontimes p;
    exitif pa<>(-1,-1);ra:=rc;
  endfor;
  %=== computation of two 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; % other 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 % result of the macro
enddef;
```

4.7.2 The minor axis of the ellipse

The `ellipse_minor_axis` macro takes a path `p` representing an ellipse of semi-major axis `a` centered at the origin, and whose major axis is oriented according to the angle `an`. The macro merely determines the intersection of `p` and a line located at a right angle to the major axis and measures its distance from the center of the ellipse.

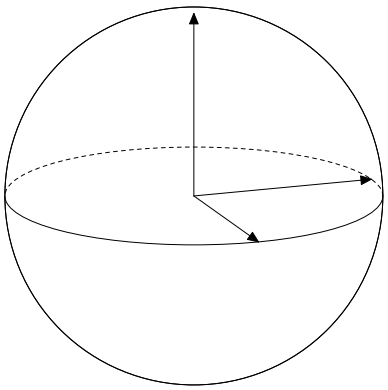
```
vardef ellipse_minor_axis(expr p,a,an)=
  save pa;pair pa;
  pa=p intersectionpoint (origin--(dir(an+90)*2a));
  arclength(origin--pa) % result
enddef;
```

These two macros make it therefore possible to determine all the parameters necessary to the economical drawing (that is, not pointwise) of an ellipse.

4.8 The dashes on the equator

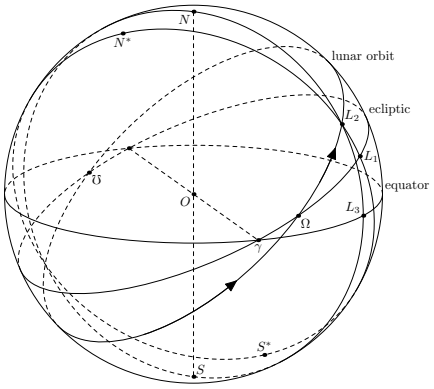
The dashes on the equator correspond to one half of the ellipse and the two halves are joined by the major axis. It is therefore sufficient to cut the ellipse in two parts and draw one in plain lines, the other in dashed lines. The ellipse returned by the `ellipse` macro is a parametric curve where the parameter goes from 0 to 8 (the base circle contains eight points), 0 being on the major axis, and the paths from 0 to 4 and from 4 to 8 are excerpted from it.

```
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; % hidden
draw pc; % visible
```



4.9 Great circles

The same principle is used for all the great circles. The only difficulty is the determination of an equation for these great circles. The macros used are parameterized in order to be able to choose which of the two parts is dashed.



Some of the circles are determined by certain constraints. For instance, in the above drawing, we were given point L_1 on the ecliptic, then the ecliptic meridian going through L_1 was constructed, leading to the determination of point L_2 on the lunar orbit. These intersections were obtained by the intersection of projections, but the intersection in space was then found again using the knowledge of the curves. Finally, the equatorial meridian going through L_2 was drawn, making it possible to obtain L_3.

4.9.1 Constraints

We can easily take such constraints into account by using the `rotatearound` macro which rotates a vector around another one.

```
% rotates Va around Vb by the 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 % result
enddef;
```

Therefore, for the case of the curve representing the ecliptic, whose equation is determined by the function

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

where `ec_angle` is the obliquity of the ecliptic plane (23.5°), we begin by determining the North pole (N^*) of the ecliptic, assuming $\gamma = (1, 0, 0)$:

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

Since point L_1 is chosen on the ecliptic, the meridian going through L_1 and N^* is determined by the two vectors $\overrightarrow{ON^*}$ and $\overrightarrow{OL_1}$, each point of the meridian being obtained by the rotation of $\overrightarrow{OL_1}$ around a vector orthogonal to $\overrightarrow{OL_1}$ and $\overrightarrow{ON^*}$. The following macro, parameterized by the point A (in space) on the ecliptic and an angle t, makes it possible to describe this meridian:

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

This function is then used to define the projected path `ec_meridian`, using `project`, as we did above when defining the `equator` path.

4.9.2 Inverse projection

The principle of the "inverse projection" is very simple and we will only sketch it. For instance, in order to determine L_2 from L_1 in the previous figure, we have on the one hand constructed the great circle going through L_1 and N^* as explained above (`ec_meridian`), and on the other hand the lunar orbit (`moon`) applying analogous principles. The intersection of these two projected curves was computed in the usual way:

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

Here, the `intersectionpoint` macro was assumed to return the correct intersection, which is not always the case.

Now, point L_2 in space is a linear combination determined by two vectors forming a basis of the plane of the lunar orbit. These two vectors can be determined by the equation of the lunar orbit and we call them `moon_x` and `moon_y`. We have therefore:

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

where `m_x` and `m_y` are scalar values. These unknowns can be determined by projecting the above equation, because a parallel projection is a linear transformation:

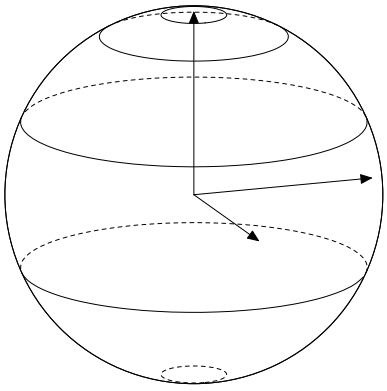
```
Lp2=m_x*project(moon_x,V1,V2)
   +m_y*project(moon_y,V1,V2);
```

The latter equation defines `m_x` and `m_y` from `Lp2` (point of the plane) and therefore defines at the same time `L2` (point in space).

Once L_2 is known, we are able to use it to obtain L_3 in an analogous way.

4.10 Parallels

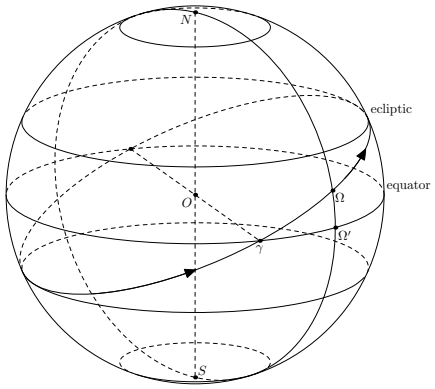
The case of the great circles was relatively simple, because these circles were always half visible and half invisible, the limit of visibility being on the major axis of the ellipse. This is not the case for the other circles of the spheres. We examine here only the case of the parallels to the equator.



Parallels have a number of distinctive features: they do not necessarily have as much visible as they have hidden; they can be totally visible or totally hidden; they have a visible/hidden limit which is not on the major axis.

In order to draw the parallels correctly, it is necessary to determine the limits between the visible part and the hidden part of a parallel.

The limits of visibility are determined by the intersection between the plane orthogonal to the viewing direction ($\overrightarrow{V_3}$), and the circle representing the parallel. This intersection can consist in zero points (the parallel is then totally visible, or totally hidden), two points (there is both a hidden and a visible part), or one point (this is the limit case between the two previous cases).



Once the intersections are obtained in space, they are converted into angles and the two arcs are drawn separately using these angles. The macro `draw_parallel` is defined in figure 4.

The equation of a parallel at latitude ϕ will be the following:

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

5 Conclusion

Having observed that many spheres were not correctly represented in the literature, we have analyzed the problem in detail and have written a few METAPOST commands to produce correct drawings. We now only hope that this work will contribute, even indirectly, to an improvement of the realism of the spheres in the way they are employed in cosmography and elsewhere.

Moreover, it seems interesting to extend other graphical packages with such functionalities, always for the purpose of fostering their use. The `PSTricks` package might benefit from such an extension, which would moreover allow for a comparison with our own implementation.

◇ Denis Roegel
LORIA — BP 239
54506 Vandœuvre-lès-Nancy cedex
France
roegel (at) loria dot fr
http://www.loria.fr/~roegel

```
% phi=latitude, col=color, side=1 or -1 depending on the dashes
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;
  % we now search for the intersections of this parallel
  % with the projection plane:
  % plane:      V3x*x+V3y*y+V3z*z=0
  % parallel: x=r*cos(phi)*cos(theta), y=r*cos(phi)*sin(theta), z=r*sin(phi)
  % we search 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;% no 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: % totally (in)visible parallel
    if side=1:draw p0 withcolor col;
    else:draw p0 withcolor col dashed evenly;fi;
    message "NO INTERSECTION";
  elseif nx=1:X10=angle(X1,1+-+X1);X11=360-X10;
  else: % general case
    if tha<thb:X10=tha;X11=thb;else:X10=thb;X11=tha;fi;
  fi;
  if nx>0: % determination of the two paths
    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;
    % drawing the two paths
    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;
```

Figure 4: Code for drawing a circle parallel to the equator.