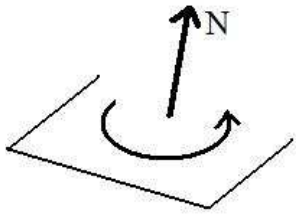
Ombres et lumière.
Faces visibles et faces cachées

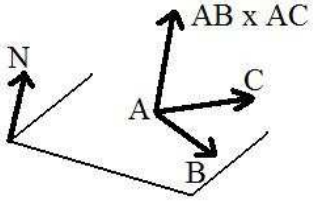
Revenons en trois dimensions. Comme nous l'avons déjà vu dans le cas de la sphère, avec ses méridiens et ses parallèles, nous allons découper les objets suivant de petites facettes carrées, auxquelles nous attribuerons une certaine coloration, suivant la disposition des sources lumineuses. Pour les calculs, nous aurons besoin de définir un vecteur normal –perpendiculaire – à chaque facette, et c'est là qu'intervient ce que l'on appelle le produit vectoriel de deux vecteurs.

Produit vectoriel de deux vecteurs

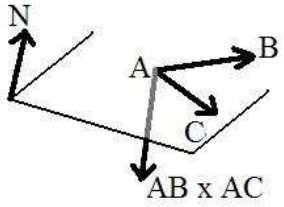
Rappelons qu'un plan, dans un repère orthonormé, a une équation de la forme :
$ax + by + cz + d = 0$, avec comme vecteur normal (perpendiculaire au plan) le vecteur (a, b, c), ou si l'on préfère ($-a$, $-b$, $-c$) selon qu'on prend ce vecteur d'un côté ou de l'autre du plan.



Choisissons un vecteur ***N*** normal. Dans ces conditions, un sens direct (positif) est donné au plan (suivant la règle du tire-bouchon).



Prenons maintenant deux vecteurs dans ce plan et donnons-leur la même origine, soit les vecteurs ***AB*** et ***AC***. Par définition le produit vectoriel ***AB*** x ***AC*** de ces deux vecteurs est un vecteur perpendiculaire au plan, de longueur $AB\ AC \sin a$, où a est l'angle non orienté (entre 0 et 180°) entre ces deux vecteurs, et orienté dans le sens de ***N*** si l'angle orienté de ***AB*** vers ***AC*** est entre 0 et π (d'où un sinus positif) et dans le sens contraire de ***N*** si l'angle orienté est entre 0 et – π.



Remarquons que le sens du vecteur ***AB*** x ***AC*** s'obtient aussi par la règle du tire-bouchon, en s'imposant de tourner de moins de 180° pour aller de ***AB*** à ***AC***, et cela quelle que soit l'orientation du plan. Cette règle est aussi appelée règle des trois doigts (de la main droite) ou règle du bonhomme d'Ampère.

En termes de coordonnées, avec deux vecteurs ***V*** (X, Y, Z) et ***V'*** (X', Y', Z'), le produit vectoriel ***V*** x ***V'*** a pour coordonnées ($YZ' - Y'Z$, $ZX' - Z'X$, $XY' - X'Y$). Cette formule nous servira ultérieurement pour avoir un vecteur ***N*** perpendiculaire à une facette.

Action d'une source lumineuse

Lorsqu'un objet – ou une facette de cet objet, est éclairé par une source lumineuse, on distingue trois types de luminosité sur la facette

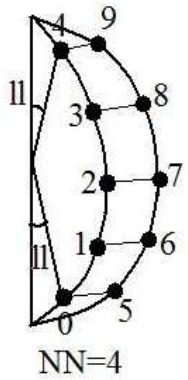
- la lumière ambiante, qui éclaire uniformément le décor
- la lumière diffusée par la facette dans toutes les directions, et de façon uniforme. Autrement dit, d'où que l'on regarde la facette, cette luminosité diffusée est la même. Son amplitude dépend uniquement de l'angle d'incidence θ entre les rayons provenant de la source lumineuse et la normale à la facette. C'est de là que proviennent les ombres, qui sont les mêmes où que soit placé l'œil de l'observateur. L'intensité de cette lumière diffusée est proportionnelle à $\cos\theta$, maximale pour un angle nul, quand la lumière frappe la facette perpendiculairement, et devenant nulle pour un angle de 90°, avec une lumière rasante). Au-delà de 90° la surface tourne le dos à la lumière et elle n'est plus éclairée.
- la réflexion spéculaire, liée à un effet miroir que peut avoir une facette. Les rayons incidents issus de la source lumineuse frappent la facette et se réfléchissent dans la direction ***R***. Si l'œil se trouve dans un cône étroit autour de cette direction, on observe un fort miroitement, sous l'aspect d'une tache de lumière localisée sur l'objet.

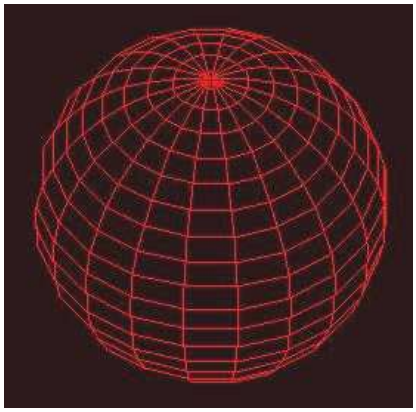
Exemple de la sphère

Reprenons la perspective cavalière qui permet de passer simplement des coordonnées en trois dimensions à celles de l'écran en deux dimensions (voir *chapitre 4*). Cela donne les déclarations préliminaires :

```
alpha=pi/4. ;  /* angle de l'œil avec le plan horizontal */
c=sqrt(2.)*tan(alpha);A=zoom/sqrt(2.);B=zoom*sin(alpha)/sqrt(2.);C=zoom*cos(alpha);
```

Facettes : Nous avons vu aussi comment découper la sphère selon ses méridiens et ses parallèles, avec les angles *phi* et *lambda*. On obtient ainsi des facettes ayant en général une forme rectangulaire, sauf celles situées aux pôles qui sont triangulaires. On découpe la sphère en *N* tranches, ce qui fait *N* méridiens. Puis chaque méridien est découpé verticalement en *NN* intervalles, non pas entre $-\pi/2$ et $\pi/2$, mais entre $-\pi/2+ll$ et $\pi/2-ll$, où *ll* est l'angle pris par la zone polaire (de l'ordre de quelques degrés). Chaque méridien compte ainsi *NN*+1 points, d'où un total de *NNN*= *N**(*NN*+1) points sur la sphère. Chacun de ces *NNN* points constitue le coin en bas à gauche d'une facette, soit rectangulaire, soit triangulaire au pôle nord.

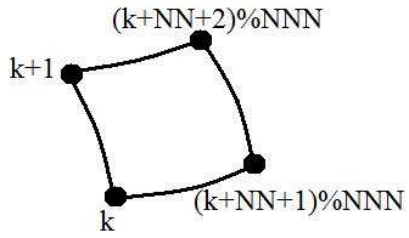




Les coordonnées de ces points, numérotés de 0 à *NNN*-1, sont enregistrées dans les tableaux *x*[], *y*[] et *z*[], et sur l'écran *xe*[] et *ye*[]. On obtient cette première partie du programme :

```
phi=0.; k=0;  dp=2.*pi / (float)N;   dl=(pi - 2.*ll) / (float)NN;
for (i=0; i<N; i++)
{   lambda= -pi/2.+ll;
   for (j=0;j<=NN;j++)
   {  x[k]=R*cos(lambda)*cos(phi);  y[k]=R*cos(lambda)*sin(phi);  z[k]=R*sin(lambda);
      dist[k]=x[k]+y[k]-c*z[k]; if (dist[k]<0.)  /* si dist[k]<0 on a une facette visible */
      xe[k]=xorig +A*(x[k]-y[k]);  ye[k]=yorig-B*(x[k]+y[k])- C*z[k];
      lambda+=dl;  k++;
   }
   phi+=dp;
}
facettes();
```

La fonction *facettes*() est charge de dessiner les quatre (ou trois) côtés des facettes visibles, à partir des quatre (ou trois) points concernés. Pour une facette rectangulaire, les numéros des sommets, à partir du sommet *k* en bas à gauche :




```
void facettes(void)
{  int k;
   for (k=0;k<NNN;k++)
   if ( (k+1) % (NN+1) !=0  && dist[k]<0.)  /* facettes carrées visibles */
      {  ligne( xe[k],ye[k],xe[k+1],ye[k+1],rouge);
         ligne( xe[k],ye[k],xe[(k+NN+1)%NNN],ye[(k+NN+1)%NNN],rouge);
         ligne(xe[(k+NN+1)%NNN],ye[(k+NN+1)%NNN],
                xe[(k+NN+2)%NNN],ye[(k+NN+2)%NNN],rouge);
         ligne(xe[k+1],ye[k+1],xe[(k+NN+2)%NNN],ye[(k+NN+2)%NNN],rouge);
      }
    else if (dist[k]<0)   /* facettes triangulaires du pôle nord */
      {  ligne( xe[k],ye[k],xorig,yorig-C,rouge);
         ligne(xe[(k+NN+1)%NNN],ye[(k+NN+1)%NNN],xorig,yorig-C,rouge);
      }
}
```

Remplissage des facettes : Maintenant nous allons enlever la fonction précédente *facettes*() pour passer à notre objectif principal, le remplissage des facettes, c'est-à-dire les ombres. On commence par se donner la source lumineuse, envoyant des rayons parallèles entre eux. Il suffit de se donner un vecteur ***S*** de longueur unité, pour nous dirigé vers la source lumineuse, de coordonnées *sx*, *sy*, *sz* telles que :

```
sx=0.; sy=-1.; sz=1.; ls=sqrt(sx*sx+sy*sy+sz*sz); sx/=ls; sy/=ls; sz/=ls;
```

On choisit aussi de colorier en rouge la sphère, suivant diverses nuances, grâce à la couleur *co*[] définie ainsi :

```
for (i=0;i<205;i++) co[i]=SDL_MapRGB(ecran->format,i,0,0);
for (i=205;i<256;i++) co[i]=SDL_MapRGB(ecran->format,i,5*i,5*i);
```

Cela donne un rouge qui évolue du sombre (i = 0) au vif pour i = 205, avec au-delà jusqu'à i = 255 un rouge qui vire vers le blanc.

Puis on passe à la fonction *remplissagefacettes*(). Chaque facette est numérotée par son sommet numéro k, grâce au programme précédent. Pour une facette rectangulaire visible (*dist*[*k*]<0) et qui n'est pas triangulaire, soit ((*k*+1) % (*NN*+1) !=0, on prend deux vecteurs, le premier ***V*** entre *k* et (*k*+*NN*+1)%*NNN*, le deuxième ***VV*** entre *k* et *k*+1. Grâce au produit vectoriel, on en déduit le vecteur normal ***N*** (*nx*, *ny*, *nz*) perpendiculaire à la facette, dirigé vers l'extérieur de la sphère, et de longueur unité. Grâce aux vecteurs unitaires, le cosinus de l'angle entre la source lumineuse et la normale est égal au produit scalaire *nx***sx*+*ny***sy*+*nz***sz*.

Pour l'intensité lumineuse, ayant choisi comme couleur d'ambiance celle obtenue pour i = 20 dans *co*[], qui est aussi la couleur du fond d'écran *couleurfond*, on lui ajoute la couleur diffusée, avec comme indice 180. *(*nx***sx*+*ny***sy*+*nz***sz*), d'où une variation entre 20 et 205 (d'un rouge sombre à un rouge vif). L'indice de cette couleur est mis dans *icolor*. Mais si cet indice *icolor* devient inférieur à 20, ce qui correspond à une facette complètement à l'ombre, on la met à *couleurfond*.

Enfin on peut ajouter la réflexion spéculaire pour ajouter un effet de miroitement. Pour cela on reprend le produit scalaire ***S.N*** donnant le cosinus de l'angle entre la source lumineuse et la normale, puis on calcule le rayon réfléchi de vecteur unité R, selon la formule vue au chapitre 2, soit $\boldsymbol{R} = 2\ \boldsymbol{S.N}\ \ N - \boldsymbol{S}$. On détermine aussi le vecteur unité allant vers l'œil, soit ***O*** de coordonnées (- cos(*alpha*)$\sqrt{2}/2$, - cos(*alpha*)$\sqrt{2}/2$, sin(*alpha*)). Le produit scalaire ***R.O*** donne le cosinus de l'angle entre l'œil et le rayon réfléchi. Pour provoquer le miroitement, imposons qu'il soit supérieur à 0,92, ce qui donne un petit angle, et faisons en sorte que l'indice *icolor* augmente de (***RO*** - 0.92)*55./ 0.08, soit une variation comprise entre 0 et 55 (on est dans la zone où le rouge vif vire au blanc). Il ne reste plus qu'à appliquer la fonction *remplirquadri*() vue au chapitre précédent, à condition de définir les variables *xq*[], *yq*[] qu'elle nécessite. Et ce que l'on a fait pour les facettes carrées, on le fait aussi pour les facettes triangulaires du pôle nord.

```
void remplissagefacettes(void)
{
   int i,j,k,icolor; float vx,vvx,vvy,vy,vz,vvz;
   float nx,ny,nz,ln,SN,RO,rx,ry,rz;
    for (k=0;k<NNN;k++) if ( dist[k]<0.)
    if ( (k+1) % (NN+1) !=0 )
        {  vx=x[(k+NN+1)%NNN]-x[k]; vy=y[(k+NN+1)%NNN]-y[k];
           vz=z[(k+NN+1)%NNN]-z[k];
           vvx=x[k+1]-x[k]; vvy=y[k+1]-y[k]; vvz=z[k+1]-z[k];
           nx=vy*vvz-vvy*vz; ny=vz*vvx-vvz*vx; nz=vx*vvy-vvx*vy;
           ln=sqrt(nx*nx+ny*ny+nz*nz);
           nx/=ln; ny/=ln; nz/=ln;  /* la normale à la facette */
           icolor=25.+180. *(nx*sx+ny*sy+nz*sz);
           xq[0]=xe[k];yq[0]=ye[k];
           xq[1]=xe[(k+NN+1)%NNN];yq[1]=ye[(k+NN+1)%NNN];
```

```
            xq[2]=xe[(k+NN+2)%NNN]; yq[2]=ye[(k+NN+2)%NNN];
            xq[3]=xe[k+1];yq[3]=ye[k+1];
            if (icolor>=20)  /* 20 est aussi l'indice de la couleur du fond */
            {
               SN=sx*nx+sy*ny+sz*nz;
               rx=2.*SN*nx-sx; ry=2.*SN*ny-sy; rz=2.*SN*nz-sz;
               RO=-rx*0.707*cos(alpha)-ry*0.707*cos(alpha)+rz*sin(alpha);
               if(RO>0.92) icolor+=(RO-0.92)*55./0.08;
               remplirquadri(co[icolor]);
            }
            else remplirquadri(couleurfond);
         }
        else   /* facettes triangulaires */
         {  vx=x[(k+NN+1)%NNN]-x[k];     vy=y[(k+NN+1)%NNN]-y[k];
            vz=z[(k+NN+1)%NNN]-z[k];
            vvx=-x[k]; vvy=-y[k]; vvz=1.-z[k];
            nx=vy*vvz-vvy*vz; ny=vz*vvx-vvz*vx; nz=vx*vvy-vvx*vy;
            ln=sqrt(nx*nx+ny*ny+nz*nz);nx/=ln; ny/=ln; nz/=ln;
            icolor=20.+180. *(nx*sx+ny*sy+nz*sz);
            xq[0]=xe[k];yq[0]=ye[k];
            xq[1]=xe[(k+NN+1)%NNN];yq[1]=ye[(k+NN+1)%NNN];
            xq[2]=xorig; yq[2]=yorig-C;
            if (icolor>=20) remplirtriangle(co[icolor]);    else remplirtriangle(couleurfond);
         }
 }
```

Derniers aménagements : Nous avons ajouté deux choses.

- un rectangle vert représentant le sol, de sommets B_0 (2, 3, -1), B_1 (-2, 3, -1), B_2 (-2, -2, -1), B_3 (2, -2, -1), le rayon de la sphère étant R = 1.

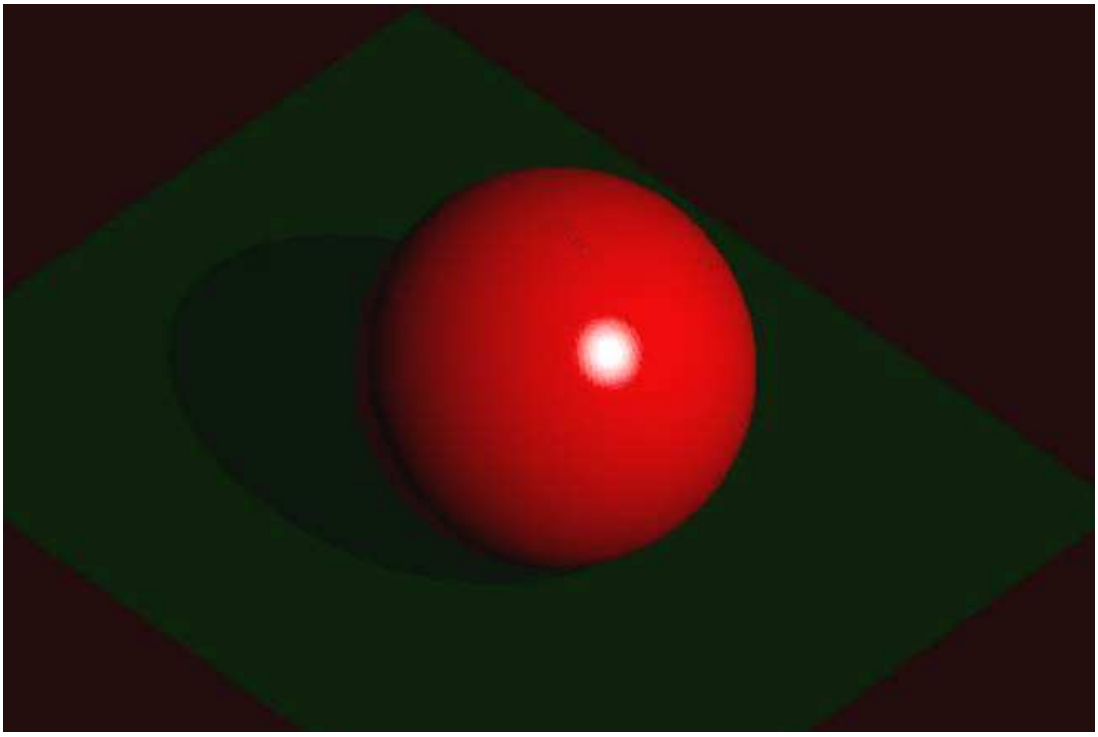
```
void socle(void)
{  float bx[4],by[4],bz[4]; int k;

   bx[0]=2.; by[0]=3.;bz[0]=-1.;
   bx[1]=-2.; by[1]=3.;bz[1]=-1.;
   bx[2]=-2.; by[2]=-2.;bz[2]=-1.;
   bx[3]=2.; by[3]=-2.;bz[3]=-1.;
   for(k=0;k<4;k++)
     { xq[k]=xorig +A*(bx[k]-by[k]);  yq[k]=yorig-B*(bx[k]+by[k])- C*bz[k];}
    remplirquadri(vert);
}
```

- une ombre portée, créée par la sphère sur le sol. Pour cela prenons les points $B(x',y',z')$ situés sur le sol (z'= -1). Pour chaque point B' prenons la droite passant par B' et dirigée par le vecteur S de la source lumineuse. Cette droite coupe le plan passant par O (centre de la sphère) et perpendiculaire à $\boldsymbol{S}$ en un point D. Il suffit de calculer la distance OD^2. Si elle est inférieure à R^2, soit 1, cela signifie que le point B est dans l'ombre créée par la sphère.

Les équations paramétriques de la droite ($B'D$) sont $X = x' + q\ sx$, $Y = y' + q\ sy$, $Z = z' + q\ sz$, avec q réel quelconque. Le plan perpendiculaire à $\boldsymbol{S}$ a pour équation : $sx\ X + sy\ Y + sz\ Z = 0$. Le point d'intersection D correspond à une valeur

$Q = -(sx*x' + sy*y' + sz*z')$, d'où D $(x'+Q\,sx\,,\ y'+Q\,sy,\ z'+Q\,sz)$. Il suffit de tester si $OD^2 < 1$

```
void ombreportee(void)
{ float xx,yy,zz,QQ; int xxe,yye;
   zz=-1.;
   for(xx=-1.;xx<2.;xx+=0.01) for(yy=-1;yy<3.;yy+=0.01)
   {
      Q=-(sx*xx+sy*yy+sz*zz);
      if((xx+Q*sx)*(xx+Q*sx)+(yy+Q*sy)*(yy+Q*sy)
         +(zz+Q*sz)*(zz+Q*sz)<R*R)
       {  xxe=xorig +A*(xx-yy);  yye=yorig-B*(xx+yy)- C*zz;
          cercle(xxe,yye,1,vertfonce);
       }
   }
 }
```

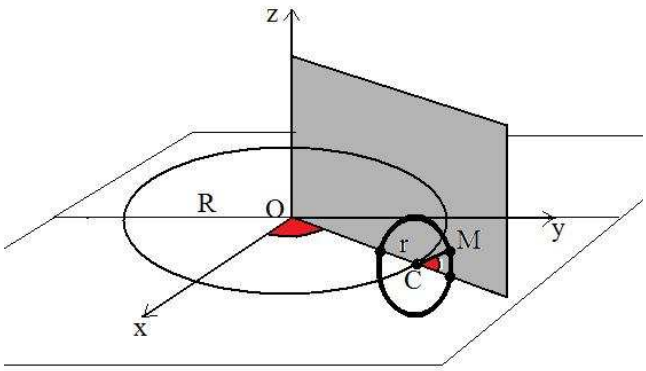


ici nous avons pris $N = NN = 150$

Nous venons de traiter la sphère sous tous ses angles. Mais il s'agit là d'un cas idéal. Lorsque l'on prend d'autres formes, ou un paysage constitué de plusieurs objets, le découpage en facettes, chacune avec sa couleur spécifiée, ne suffit pas. Encore faut-il tracer les facettes en distinguant celles qui sont visibles et celles qui ne le sont pas. Comment faire ? Quand une facette se trouve derrière une autre, elle doit être dessinée en premier. On est amené à trier les facettes des plus lointaines aux plus proches en les dessinant dans cet ordre. Nous allons voir cela dans l'exemple suivant, celui du tore.

Le tore

Un tore n'est rien d'autre qu'un pneu. Pour le fabriquer, on commence par dessiner un *grand* cercle de rayon R et de centre O dans un plan horizontal. Puis on fait tourner



un plan vertical autour de l'axe du cercle. Ce plan coupe le grand cercle en un point *M*. On trace alors dans ce plan vertical un *petit* cercle de rayon *r* et de centre *C*. Lorsque le plan tourne, le petit cercle qu'il contient engendre une forme qui est un tore. On va en déduire les équations du tore. La position d'un point *M* (*x*, *y*, *z*) sur le tore dépend uniquement de deux angles, φ et θ, le premier correspondant à la rotation du plan vertical, et le deuxième à la rotation sur le petit cercle. Ces deux angles varient de 0 à 2π. La projection du point *M* sur le plan horizontal donne un point *m* tel que $Om = R + r \cos\theta$. Puis on projette sur les axes du repère :

$x = (R + r \cos\theta) \cos\varphi$
$y = (R + r \cos\theta) \sin\varphi$
$z = r \sin\theta$

Quadrillage du tore : On divise le grand cercle en *N* angles égaux, soit une variation angulaire de $d = 2\pi / N$ d'un angle au suivant. On fait de même avec le petit cercle, découpé en *NN* angles, avec une variation $dd = 2\pi / NN$ à chaque fois. On obtient ainsi $NNN = N \times NN$ points sur le tore, et l'on numérote chacun de ces points de 0 à $NNN - 1$, de façon qu'à chaque tour complet sur les petits cercles successifs, le numéro augmente de *NN*. A partir des coordonnées *x*[*k*], *y*[*k*], *z*[*k*] de chacun de ces points *k*, on passe à leurs coordonnées *xe*[*k*], *ye*[*k*] sur l'écran. On utilise pour cela les formules de passage vues au *chapitre* 4 (*Géométrie* 3*d*), en se donnant l'angle *alpha* que fait l'œil avec le plan horizontal. Par la même occasion on calcule leur distance *dist*[*k*] par rapport au plan de l'écran.

Distance d'un point à un plan : Rappelons que l'équation d'un plan dans l'espace est $ax + by + cz + d = 0$ avec pour vecteur normal $\boldsymbol{N}$ (*a*, *b*, *c*) supposé non nul. La *distance* d'un point *A* (x_0, y_0, z_0) au plan est égale à :

$a\,x_0 + b\,y_0 + c\,z_0 + d$ lorsque le vecteur $\boldsymbol{N}$ a une longueur 1
(et sinon, s'il a une longueur *L* la distance est $(a\,x_0 + b\,y_0 + c\,z_0 + d) / L$.
Plus précisément il s'agit d'un nombre positif lorsque le point *A* est du côté de $\boldsymbol{N}$ et d'un nombre négatif dans le cas contraire. La distance à proprement parler serait la valeur absolue de ce nombre. [1]

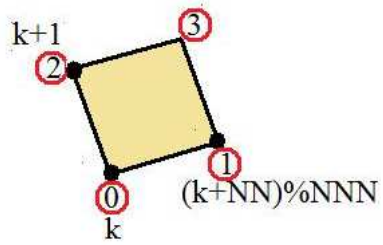
Dans le cas présent, le plan qui fait office d'écran a pour équation $x + y - c\,z = 0$, et son vecteur normal (1, 1, -*c*) est placé de l'autre côté de l'œil. A un facteur près (la longueur du vecteur normal), la distance avec un signe en plus, d'un point M_k (*x*[*k*], *y*[*k*], *z*[*k*]) au plan est : *dist*[*k*] = *x*[*k*] + *y*[*k*] – *c* * *z*[*k*]. Pour classer les points des plus lointains aux plus proches, on prend *dist*[] de sa valeur la plus grande (positive) jusqu'à sa valeur la plus faible (négative)

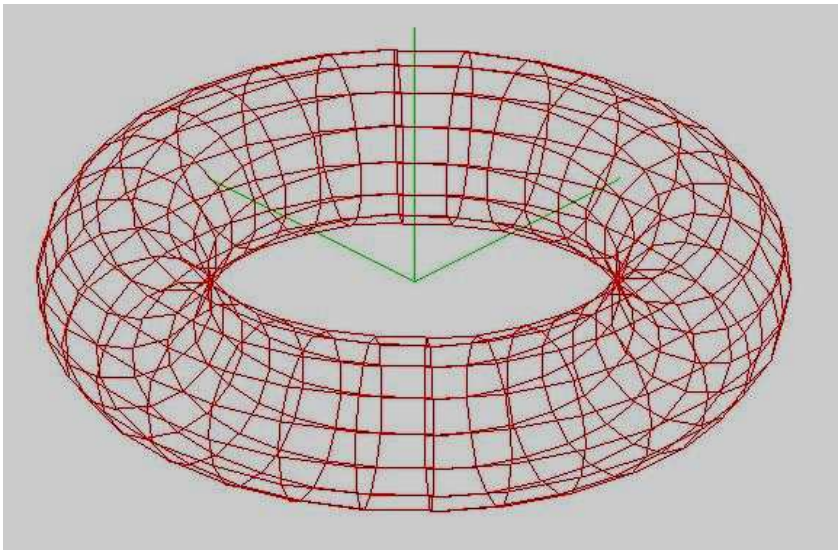
[1] Pour démontrer cette formule, il suffit d'appliquer la définition du produit scalaire.

On aboutit à cette première partie du programme :

```
phi=0.; k=0;  d=2.*M_PI/(float)N; dd=2.*M_PI/(float)NN;
for(i=0;i<N;i++)
 {  theta=0.;
   for(j=0;j<NN;j++)
    { x[k]=(R+r*cos(theta))*cos(phi); y[k]=(R+r*cos(theta))*sin(phi); z[k]=r*sin(theta);
      dist[k]=x[k]+y[k]-c*z[k];
      xe[k]=xorig +A*(x[k]-y[k]);  ye[k]=yorig-B*(x[k]+y[k])- C*z[k];
      theta+=dd; k++;
     }
   phi+=d;
 }
```

Dessin en fil de fer du tore : A chaque point construit précédemment correspond une facette, ce point étant le coin en bas à gauche de la facette. Il y a autant de points que de facettes. Sur chaque facette, le point noté 0 est le point numéro *k*, celui situé à sa droite, noté 1, correspond au point numéro (*k*+*NN*)%*NNN*, le modulo étant indispensable à cause du dernier point, et de même le point au-dessus, noté 2, a pour numéro *k*+1, sauf lorsque *k*+1 ramené modulo *NN* atteint 0, au terme de chaque tour complet, auquel cas le numéro est *k* + 1 – *NN*. En dessinant les jonctions entre les points 0 et 1, ainsi qu'entre 0 et 2, on aura le dessin en style fil de fer du tore, avec toutes ses facettes.





Avec le programme correspondant :

```
for(k=0;k<NNN;k++)
{  xe0[k]=xorig +A*(x[k]-y[k]);  ye0[k]=yorig-B*(x[k]+y[k])- C*z[k];
```

```
   xe1[k]=xorig +A*(x[(k+NN)%NNN]-y[(k+NN)%NNN]);
   ye1[k]=yorig-B*(x[(k+NN)%NNN]+y[(k+NN)%NNN])- C*z[(k+NN)%NNN];
   if (k%NN==NN-1)
     {xe2[k]=xorig +A*(x[k+1-NN]-y[k+1-NN]);
      ye2[k]=yorig-B*(x[k+1-NN]+y[k+1-NN])- C*z[k+1-NN];
     }
  else
     {xe2[k]=xorig +A*(x[k+1]-y[k+1]);  ye2[k]=yorig-B*(x[k+1]+y[k+1])- C*z[k+1];   }
  ligne(xe0[k],ye0[k],xe1[k],ye1[k],rouge);
  ligne(xe0[k],ye0[k],xe2[k],ye2[k],rouge);
}
```

Ombres sur les facettes : Comme les points, les facettes sont indexées par leur numéro *k*. Pour chacune, on détermine leur vecteur normal unité ***N*** en faisant le produit vectoriel des vecteurs joignant les points 0 1 et 02. A son tour, le produit scalaire entre ce vecteur et le vecteur ***S*** dirigé vers le soleil (que l'on s'est donné) permet d'avoir le cosinus de leur angle, ce qui permettra de déterminer le niveau d'ombre de la facette.

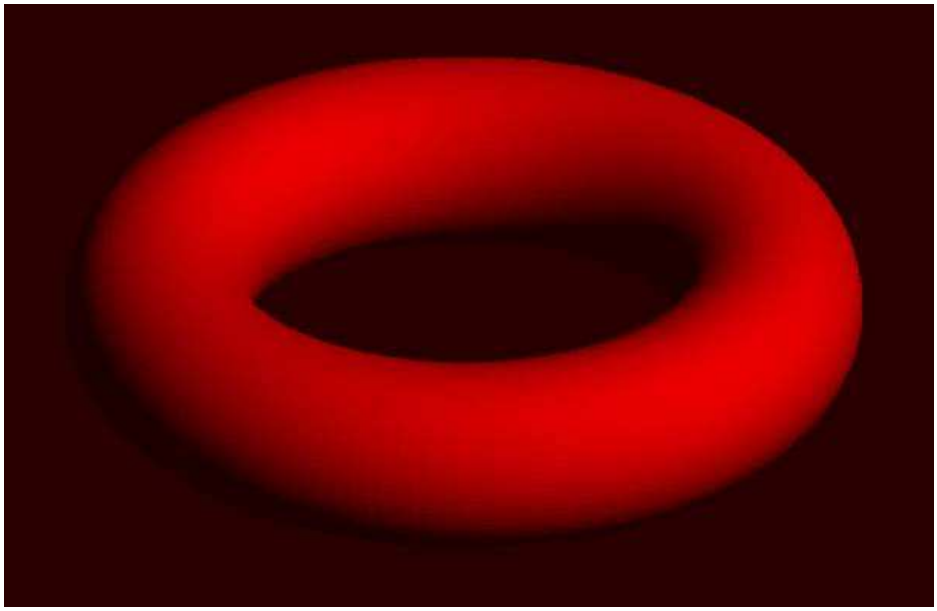
```
for(k=0;k<NNN;k++)
 {  v01x=x[(k+NN)%NNN]-x[k]; v01y=y[(k+NN)%NNN]-y[k];
    v01z=z[(k+NN)%NNN]-z[k];
    if (k%NN==NN-1)
     { v02x=x[k+1-NN]-x[k];  v02y=y[k+1-NN]-y[k];  v02z=z[k+1-NN]-z[k];}
   else
     { v02x=x[k+1]-x[k]; v02y=y[k+1]-y[k]; v02z=z[k+1]-z[k];}
   Nx=v01y*v02z-v01z*v02y; Ny=v01z*v02x-v01x*v02z; Nz=v01x*v02y-v01y*v02x;
   longN=sqrt(Nx*Nx+Ny*Ny+Nz*Nz);   Nx=Nx/longN;Ny=Ny/longN;Nz=Nz/longN;
  cosinus[k]=Nx*Sx+Ny*Sy+Nz*Sz;     if (cosinus[k]<0.) cosinus[k]=0.;
 }
```

Dessin des facettes, de la plus lointaine à la plus proche : On commence par ranger les facettes par leur numéros de 0 à *NNN*-1 dans un tableau *b*[]. Puis on procède à un tri en les classant selon les valeurs décroissantes de leur distance *dist*[*k*]. Il suffit de parcourir le tableau *b*[] trié en dessinant les facettes l'une après l'autre, en leur donnant une couleur qui ajoute à la couleur d'ambiance (ici 30), la couleur diffusée, proportionnelle à *cosinus*[*b*[*i*]]. Pour cela on s'est donné en conditions initiales une couleur *color*[*j*], l'indice *j* allant de 0 (rouge foncé) à 255 (rouge vif). On utilise enfin la fonction *remplirquadri()* pour colorier la facette. [2] D'où la fin du programme :

```
for(i=0;i<NNN;i++) b[i]=i;
for(i=0;i<NNN-1;i++) for(j=i+1; j<NNN;j++)
if (dist[b[j]]>dist[b[i]]) { aux=b[i]; b[i]=b[j]; b[j]=aux; }
for(i=0;i<NNN;i++)
 {  ic=30.+(235.-30.)*cosinus[b[i]];  /* indice de couleur */
    jmax=b[i];
    xe3=xe1[jmax]+xe2[jmax]-xe0[jmax];
    ye3=ye1[jmax]+ye2[jmax]-ye0[jmax];
    xq[0]=xe0[jmax];yq[0]=ye0[jmax];xq[1]=xe1[jmax];yq[1]=ye1[jmax];
```

[2] Rappelons que cette fonction nécessite d'entrer non seulement la fonction *remplirquadri*() mais aussi les fonctions *line*() et *lignehori*(), et de déclarer les variables globales *xq*[4],*yq*[4],*xxq*[10000],*yyq*[10000],*kq*,*xgauche*[10000],*ygauche*[10000], les quatre points du quadrilatère facette étant entrés dans *xq*[4] et *yq*[4].

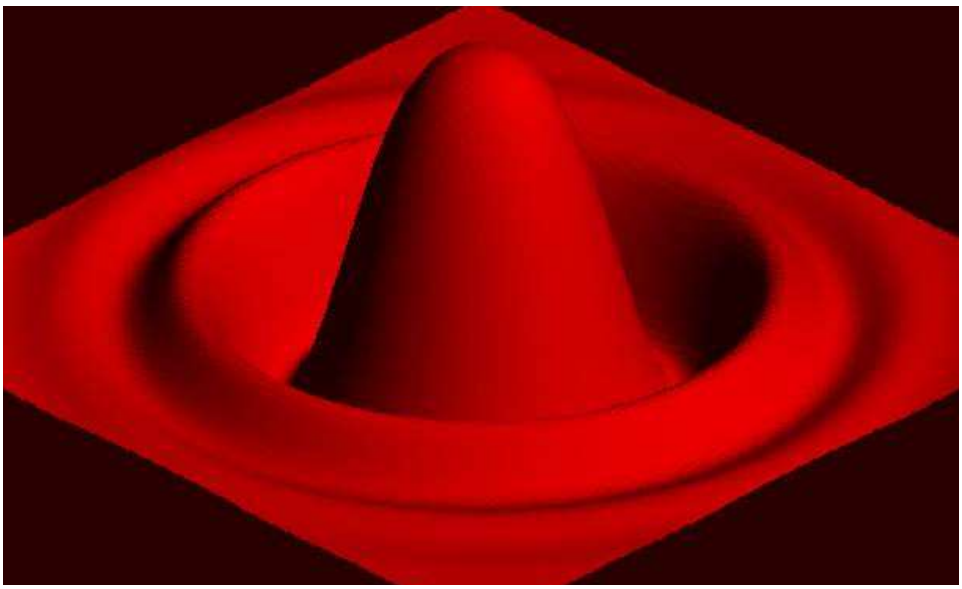
```
    xq[2]=xe3;yq[2]=ye3;xq[3]=xe2[jmax];yq[3]=ye2[jmax];
    remplirquadri(color[ic]);
  }
```



Exercices complémentaires

Exercice 1 : Tracé d'une surface de révolution avec ses ombres

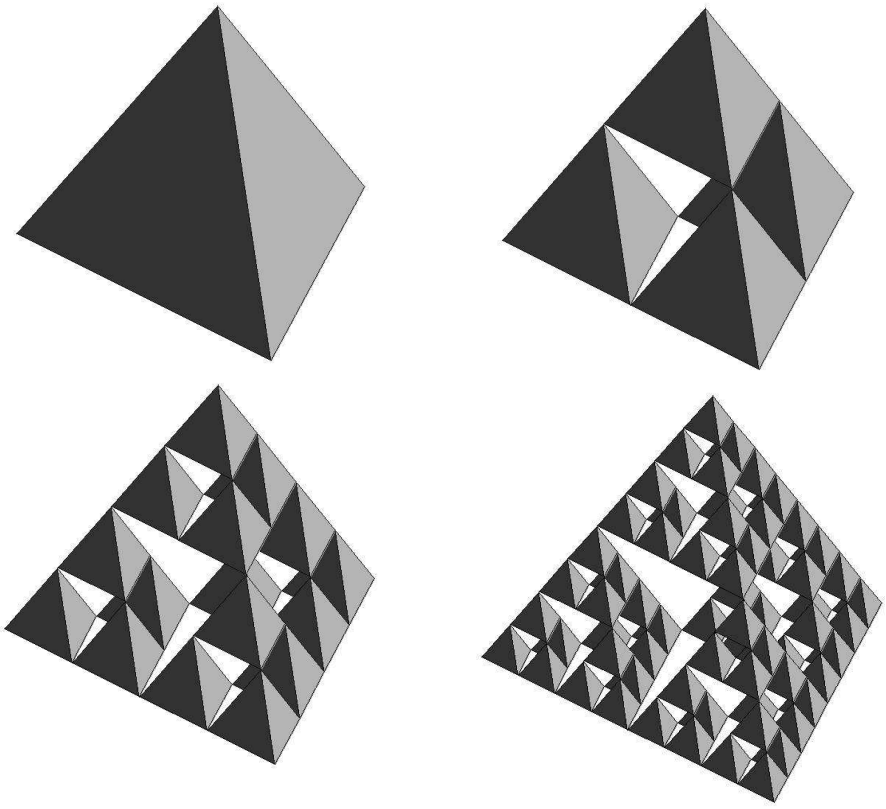
Une surface en trois dimensions a une équation de la forme $z = f(x, y)$, et c'est une surface de révolution lorsque x et y interviennent uniquement dans le bloc $x^2 + y^2$. Voici ce que l'on obtient par exemple avec $z = exp(– x*x – y*y) \cos(0.5*(x*x+y*y))$



Puis, en faisant varier les paramètres, fabriquer plusieurs images permettant de simuler la chute d'une goutte sur une surface d'eau,

Exercice 2 : La pyramide des pyramides

Commencer par dessiner une pyramide à base de triangles équilatéraux (un tétraèdre régulier) en prenant comme centre de gravité de sa base horizontale l'origine *O* du repère, avec un des sommets sur l'axe *Ox*. Puis dessiner des pyramides analogues mais avec un centre de gravité de leur base situé en un point quelconque (*xg*, *yg*, *zg*) avec une certaine longueur *L* entre ce centre de gravité et un sommet de la base. Enfin, dessiner un certain nombre de pyramides emboîtées les unes dans les autres, comme sur les dessins suivants :

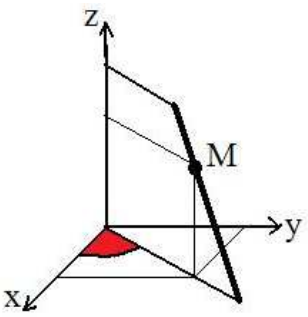


Il conviendra de faire un programme récursif où la fonction *pyramide* (*xg*, *yg*, *zg*, *L*, *n*) où *n* est l'indice de la récursivité, se rappelle quatre fois. Une simple gestion de la récursivité permet un affichage des facettes des plus lointaines aux plus proches. On utilisera aussi la fonction de remplissage des triangles vues précédemment, en se contentant de deux couleurs.

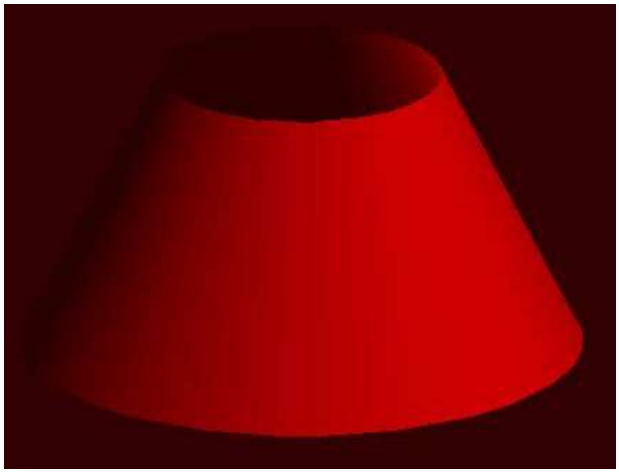
Exercice 3 : Dessin d'un tronc de cône (style abat-jour)

Plaçons-nous dans le plan vertical *xOz*, et traçons un segment de droite, entre les deux points $x = 1$, $z = 0$, et $x = 0.5$, $z = 1$. La droite correspondante a pour équation $x = 1 - 0.5\ z$. En faisant tourner ce segment autour de l'axe vertical Oz, ce segment engendre un tronc de cône. En appelant φ l'angle correspondant à la longitude, un point *M* appartient au tronc de cône si et seulement si :

$x = (1 - 0.5\ z) \cos \varphi$, $y = (1 - 0.5\ z) \sin \varphi$, avec z variant entre 0 et 1.



Pour les facettes, on peut se contenter de prendre des tranches verticales s'étendant sur toute la hauteur du cône. On obtient le résultat ci-dessous, avec une légère erreur, car une partie de l'intérieur du cône est aussi colorié. Pour l'éviter, on peut placer un couvercle horizontal en haut du tronc de cône. Ou bien en séparant les facettes visibles extérieures et intérieures, on peut donner aux facettes intérieures un vecteur normal de sens opposé. On obtiendra alors des ombres dans la zone intérieure.



L'algorithme du peintre

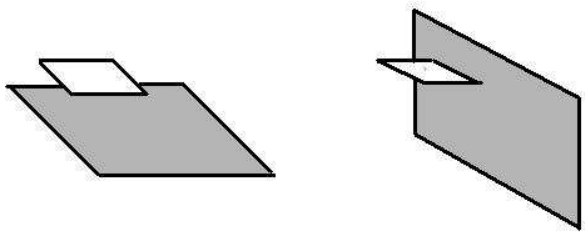
Dans ce qui précède, nous nous sommes arrangés pour dessiner les facettes lointaines avant les facettes proches. Il s'agit d'une version simplifiée de l'algorithme spontanément utilisé par les peintres. Un peintre est en effet amené à peindre plusieurs couches l'une par-dessus l'autre, en fonction de la proximité des formes à tracer.

Pour nous, dans le cas le plus simple, le paysage ne comporte qu'un objet unique fermé convexe, à facettes planes. Il possède un volume intérieur, et sa convexité fait qu'il se divise en deux parties, avant et arrière, séparées par une frontière, une où les facettes sont visibles, et l'autre où elles sont cachées par d'autres facettes. C'est le cas du cube, plus généralement des polyèdres réguliers, et aussi de la sphère que nous avons vue, mais pas du tore. Dans les cas simples, la frontière entre les faces visibles et celles qui sont cachées, auto-cachées dans le sens où c'est l'objet lui-même qui les cache, est facile à déterminer. En général, on peut considérer qu'une face est visible lorsque son extérieur contient l'œil de l'observateur. Pour connaître les facettes visibles, on détermine l'angle entre le vecteur normal à la facette et dirigé vers l'extérieur, et le vecteur allant de la facette vers l'œil. Si cet angle est aigu (produit scalaire positif), la facette est visible.

Dans le cas d'un paysage comportant plusieurs objets, cet algorithme qui élimine les faces auto-cachées reste valable, comme première étape, car les faces auto-cachées d'un objet ne cachent rien de plus. Il ne reste qu'à s'occuper des facettes visibles de chaque objet, mais celles-ci peuvent à leur tour cacher des facettes visibles d'autres objets. De là découle l'algorithme du peintre :

- éliminer les faces auto-cachées de chaque objet
- trier les faces restantes dans l'ordre de la plus lointaine à la plus proche
- projeter cheque facette dans cet ordre sur l'écran avec sa couleur

Mais cet algorithme ne convient que si la notion de distance de la facette à l'œil est un critère véritable. En fait, il n'est sûr de fonctionner que si toutes les facettes sont dans des plans perpendiculaires à la direction de l'œil. Dès que le contexte se complique, il ne marche plus, comme dans les exemples simples ci-dessous, avec deux facettes de dimension différente :



Aussi est-on amené à utiliser d'autres méthodes, comme celles des arbres *BSP*, particulièrement adaptés à un paysage formé de murs et de couloirs, comme dans de nombreux jeux. C'est ce que nous allons voir dans le chapitre suivant.