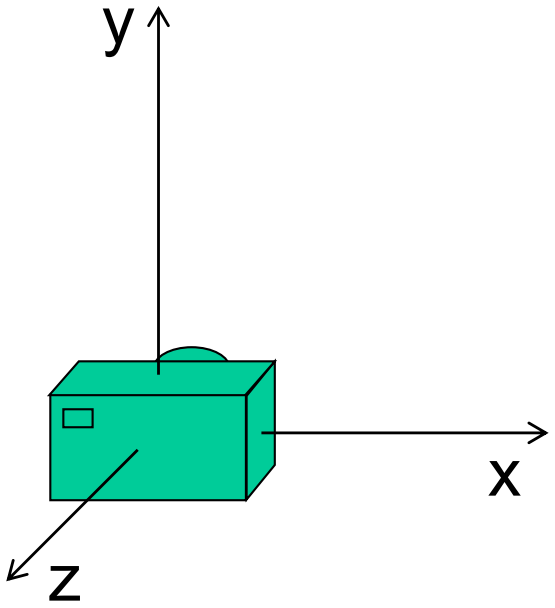
IV.
Primitives graphiques

Préliminaire :

Le repère 3D est direct

La caméra est initialement

- à l'origine
- tournée vers les Z négatifs



1. La couleur

Codage : 4 scalaires

- rouge, vert, bleu
- et … transparence (ou coefficient *alpha*)
- réel dans [0. , 1.] ou entier dans [0 , 255]
- transparence facultative (1. par défaut : opaque)
- blanc (1., 1., 1.), noir (0., 0., 0.)
- bleu (0., 0., 1.), rouge(1., 0., 0.)
- jaune (1., 1., 0.), violet sombre (0.5, 0., 0.5)

Deux **variables d'état** sous openGL :

- couleur du fond
- couleur du tracé
- elles font partie du contexte de tracé

Couleur du fond :

Pourquoi ?

- "support" du dessin
- pour effacer l'ancien tracé (*quasi instantané !*)

Deux commandes openGL :

- définir la couleur du fond (affectation de la variable d'état) :

```
glClearColor(0., 0., 0., 1.);
```

- effacer la fenêtre d'affichage avec cette couleur (utilisation de la variable d'état) :

```
glClear(GL_COLOR_BUFFER_BIT);
```

Couleur des primitives :

`glColor3f(`*`rouge, vert, bleu`*`)`

Exemple : `glColor3f(1., 0., .0) ;`

- le tracé des prochaines primitives sera en rouge
- c'est une variable d'état qui peut être modifiée à tout moment !

`glColor` est polymorphe :

- Notation vectorielle

```
float jaune[3] = {1., 1., 0.};
glColor3fv(jaune);
```

- Notation entière avec chaque coefficients dans [0, 255]

```
glColor3ub(255, 255, 0);
glColor3i(255, 255, 0);
```

- remarques sur un tracé en couleur brute :
  - faces uniformes
  - pas de sensation de « relief »
  - mauvaise perception volumique
  - mais peu coûteux
  - *pour améliorer le rendu, il faudra simuler un éclairage et un matériau qui réfléchit cette lumière vers la caméra…*

- Conseil : séparer si possible la couleur de la construction d'un objet

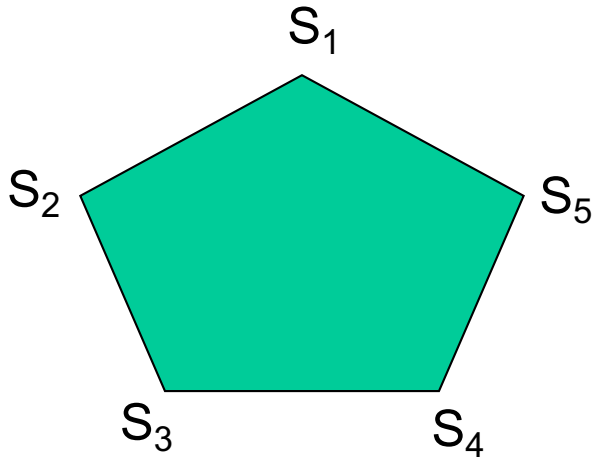
```
glColor3f(1., 0., 0.); /*rouge*/
dessineBicyclette(position, direction);
glColor3f(0., 0., 1.); /*bleu*/
dessineBicyclette(autrePosition, autreDirection);
```

2. Primitives graphiques

Primitives de base : le triangle, la ligne et le point.

Toute surface est décomposée en triangles par OpenGL.

```
glBegin(type de figure);
    suite de sommets associés au tracé
glEnd();
```



Désigner un sommet : `glVertex3type(coordonnées3D)`

`glVertex3f(x, y, z)`

`glVertex3fv(vecteur de 3 float)`

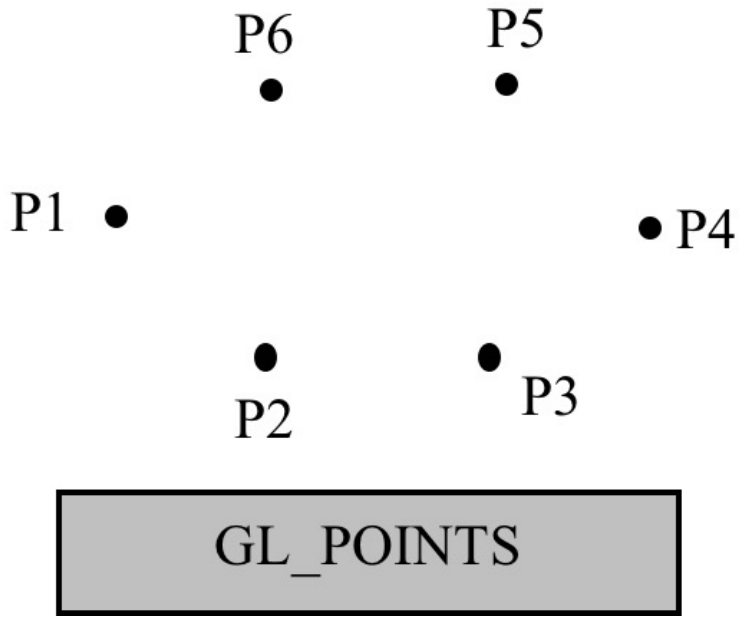
Exemple de tracé de triangles :

```
glColor3f(1., 0., 0.) ; /* crayon rouge */
glBegin(GL_TRIANGLES);
   /* un triangle */
   glVertex3f(x1,y1,z1);
   glVertex3f(x2,y2,z2);
   glVertex3f(x3,y3,z3);

   /* un autre triangle */
   glColor3f(0., 1., 0.) ; /* crayon vert */
   glVertex3f(x4,y4,z4);
   glVertex3f(x5,y5,z5);
   glVertex3f(x6,y6,z6);
   /* etc. */
glEnd();
```

Valeurs du paramètre de `glBegin` :

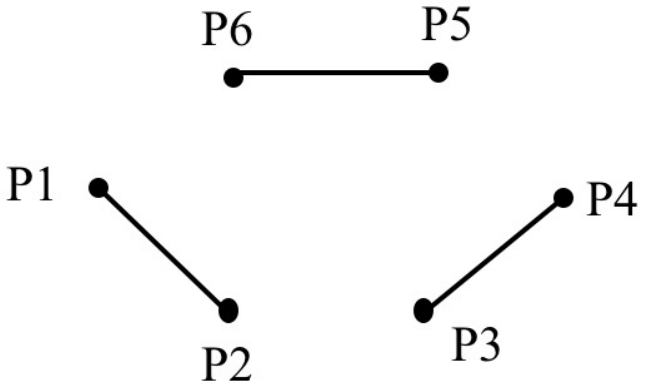
Les points :

```
glBegin(GL_POINTS) ;
  glVertex3fv(P1) ;
  glVertex3fv(P2) ;
  glVertex3fv(P3) ;
  glVertex3fv(P4) ;
  glVertex3fv(P5) ;
  glVertex3fv(P6) ;
glEnd() ;
```

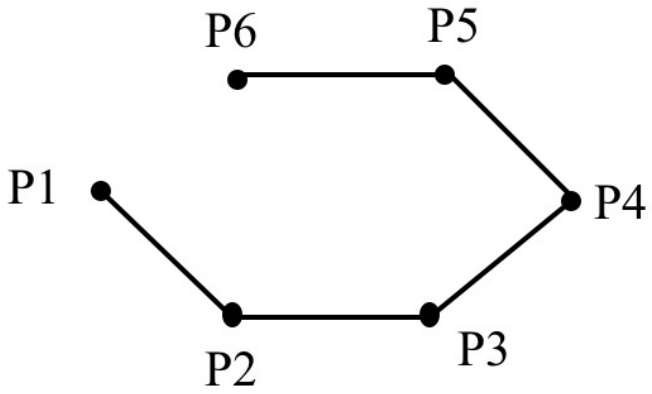


Les lignes :

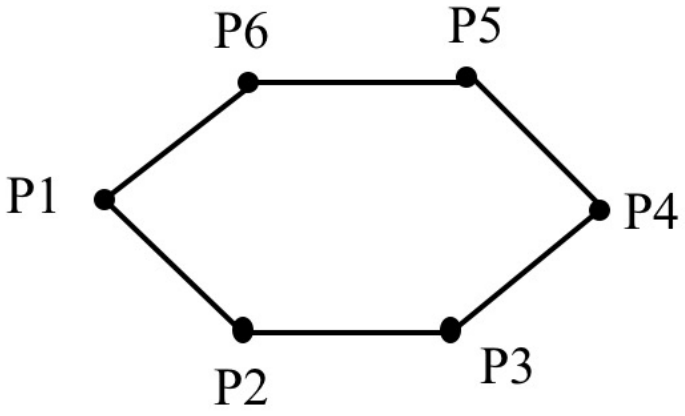
```
glBegin(GL_LINES) ;
  glVertex3fv(P1) ;
  glVertex3fv(P2) ;
  glVertex3fv(P3) ;
  glVertex3fv(P4) ;
  glVertex3fv(P5) ;
  glVertex3fv(P6) ;
glEnd() ;
```



GL_LINES



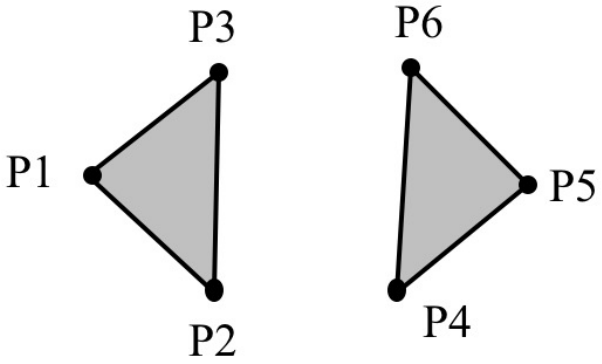
GL_LINE_STRIP



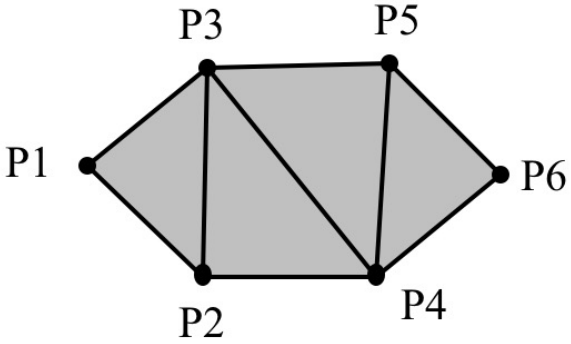
GL_LINE_LOOP

Les triangles :

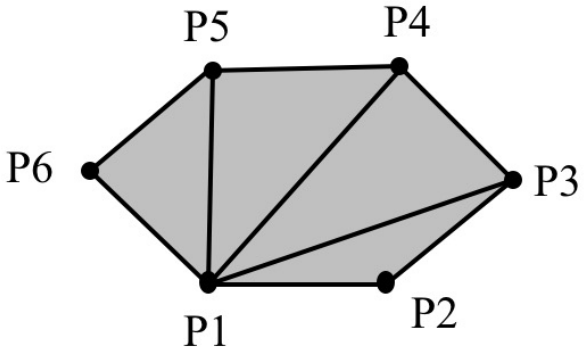
```
glBegin(GL_TRIANGLES) ;
   glVertex3fv(P1) ;
   glVertex3fv(P2) ;
   glVertex3fv(P3) ;
   glVertex3fv(P4) ;
   glVertex3fv(P5) ;
   glVertex3fv(P6) ;
glEnd() ;
```



GL_TRIANGLES



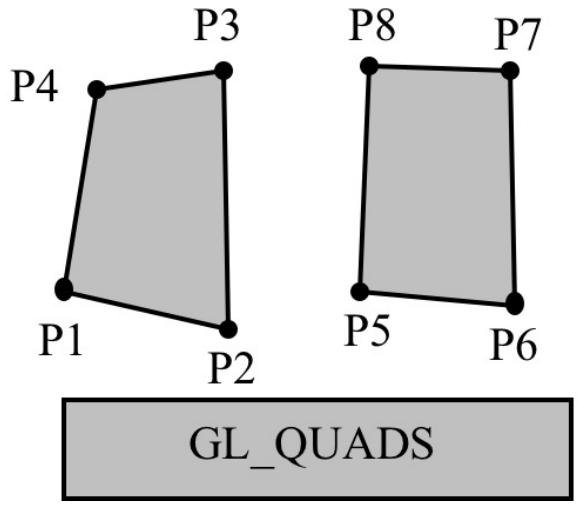
GL_TRIANGLE_STRIP

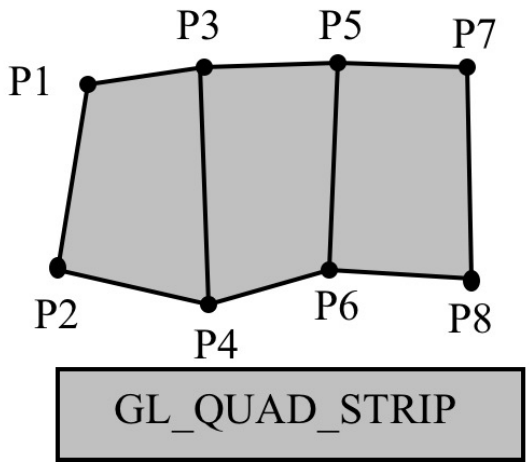


GL_TRIANGLE_FAN

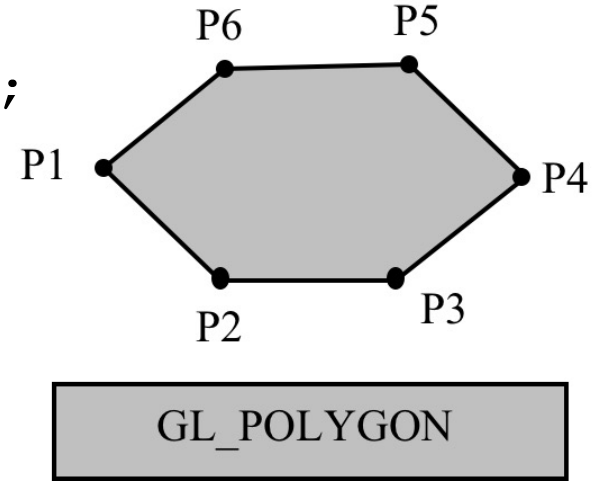
Les polygones :

```
glBegin(GL_QUADS);
   glVertex3fv(P1);
   glVertex3fv(P2);
   glVertex3fv(P3);
   glVertex3fv(P4);
   glVertex3fv(P5);
   glVertex3fv(P6);
   glVertex3fv(P7);
   glVertex3fv(P8);
glEnd() ;
```



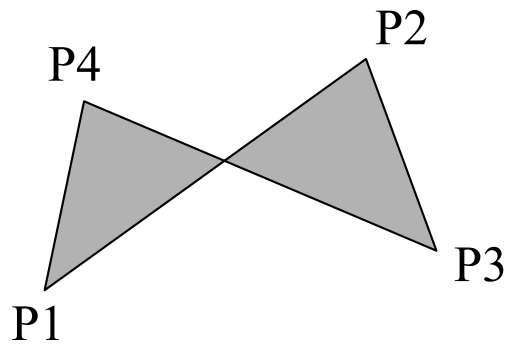



```
glBegin(GL_POLYGON);
   glVertex3fv(P1);
   glVertex3fv(P2);
   glVertex3fv(P3);
   glVertex3fv(P4);
   glVertex3fv(P5);
   glVertex3fv(P6);
glEnd() ;
```



Attention sur le tracé d'un polygone :

- ses sommets doivent être **coplanaires**
- pas d'intersection d'arêtes
  - `glBegin(GL_QUADS);`
  - `  glVertex3fv(P1);`
  - `  glVertex3fv(P2);`
  - `  glVertex3fv(P3);`
  - `  glVertex3fv(P4);`
  - `glEnd() ;`



Remarques sur l'optimisation du code GL :

- GL_TRIANGLES est plus rapide que GL_POLYGON
- un maximum de primitives entre `glBegin()` et `glEnd()`.
- la notation vectorielle (suffixe `v`) est généralement plus rapide.

```
typedef float Point3D[3];
Point3D P1={1,3,0} ;
```

3. Épaisseur des points et des traits

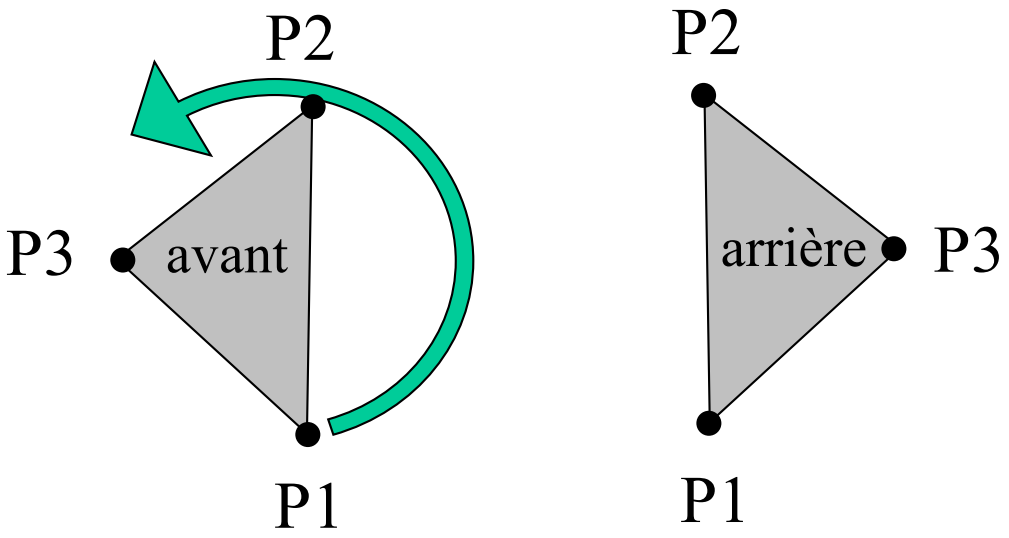
```
void glPointSize(GLfloat size)
void glLineWidth(GLfloat width)
```

- taille de 1 pixel par défaut
- peut-être redéfinie à tout moment (affectation d'une variable d'état)
- taille réelle (float) :
  - ✓ efficace si l'anti-aliasing (anti-crénelage) est activé
  - ✓ arrondie à l'entier le plus proche sinon

3. Face avant/arrière d'un polygone

Parcours trigonométrique des sommets : **face avant**

```
glBegin(GL_TRIANGLES) ;
   glVertex3fv(P1) ;
   glVertex3fv(P2) ;
   glVertex3fv(P3) ;
glEnd() ;
```



Les calculs de rendu sont définis soit sur :

- les faces avants **GL_FRONT**
- les faces arrières **GL_BACK**
- les faces avants et arrières **GL_FRONT_AND_BACK**

Contexte d'affichage des faces (*variables d'état*) :

Polygones pleins, ou seulement leurs contours : **`glPolygonMode`**

```
glPolygonMode(GL_FRONT, GL_FILL); /* faces pleines */
glPolygonMode(GL_BACK, GL_LINE);  /* contours */
```

Définir et ne pas tracer les faces non visibles (gain de temps !)

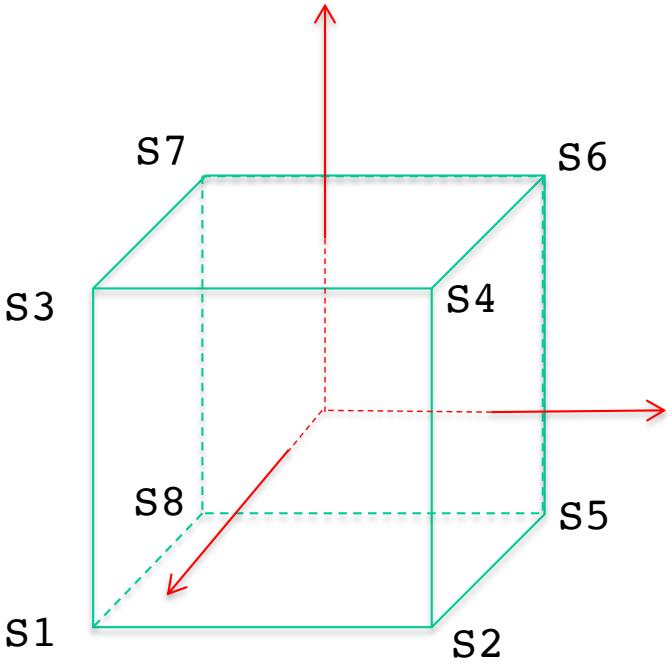
```
glCullFace(GL_BACK); // choix de la face non visible
glEnable(GL_CULL_FACE);  // activer l'optimisation
glDisable(GL_CULL_FACE); // désactiver l'optimisation
```

Ex : contexte de tracé d'un cube "plein" vu de l'extérieur

- *choisir l'orientation des faces extérieures* *(par exemple **`GL_FRONT`**)*
- définir les faces non visibles — **`glCullFace(GL_BACK);`**
- activer l'optimisation — **`glEnable(GL_CULL_FACE);`**
- tracer le cube — ***`tracerCube();`***
- désactiver l'optimisation — **`glDisable(GL_CULL_FACE);`**

Tracé d'un cube unitaire centré sur O :

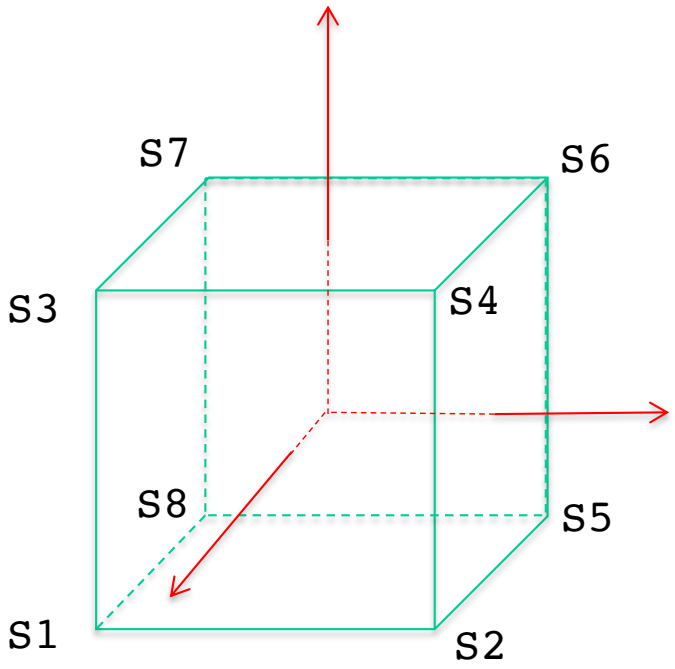
```
Point3D s1={-0.5,-0.5,0.5},
        s2={0.5,-0.5,0.5}
        s3={-0.5,0.5,0.5},
        s4={},  à compléter
        s5={},
        s6={},
        s7={},
        s8={} ;
```




```
dessineCube()  // faces "avants" à l'extérieur
{ glBegin(GL_QUADS) ;
    // devant


    //face droite


    // gauche


    // arrière


    // dessus


    // dessous


  glEnd() ;
}
```

Tracé d'un cube unitaire centré sur O :

```
Point3D s1={-0.5,-0.5,0.5},
        s2={0.5,-0.5,0.5}
        s3={-0.5,0.5,0.5},
        s4={},  à compléter
        s5={},
        s6={},
        s7={},
        s8={} ;
```



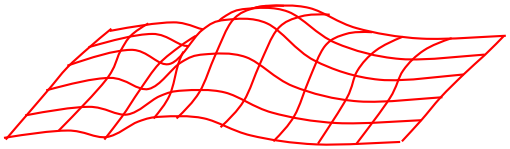

```
dessineCube()  // faces "avants" à l'extérieur
{ glBegin(GL_QUADS) ;
    // devant 3-1-2-4
    glVertex3fv(s3); glVertex3fv(s1);
    glVertex3fv(s2); glVertex3fv(s4);
    //face droite 2-5-6-4
    à compléter

    // gauche 7-8-1-3


    // arrière 8-7-6-5


    // dessus 3-4-6-7


    // dessous 1-8-5-2


  glEnd() ;
}
```

4. Exemple de surfaces « fil de fer » :

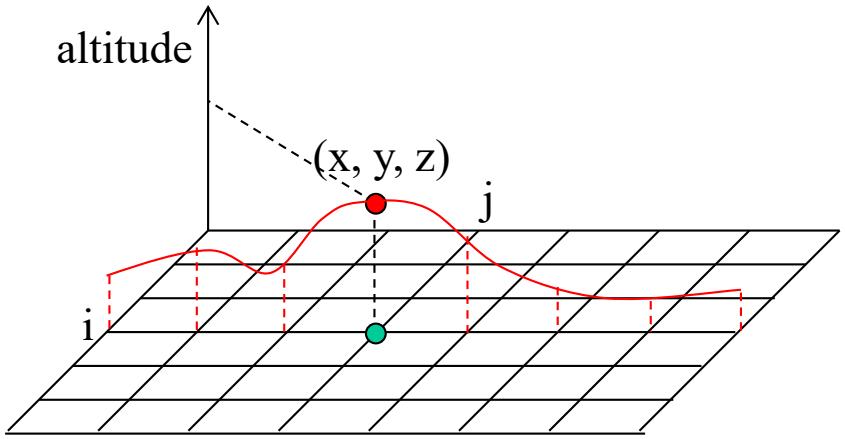
Méthode pour dessiner la surface d'un objet complexe :

- choisir une représentation discrète (taille des mailles)
- obtenir les points qui composent sa surface (calcul, fichier…)
- stocker les coordonnées dans une structure adaptée (matrice, …)
- utiliser cette structure pour afficher l'objet à la demande

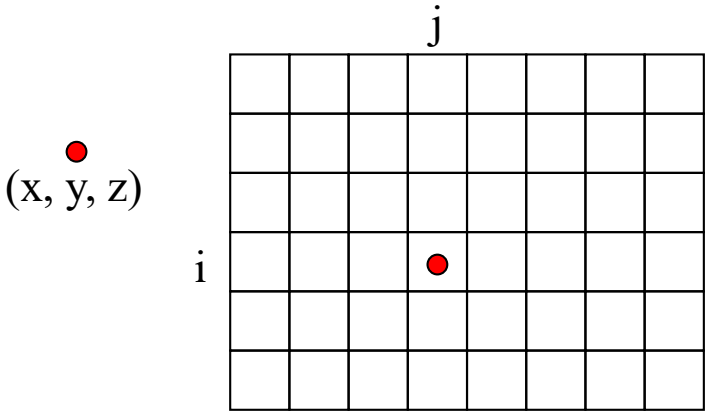
Ex : topographie d'un terrain :



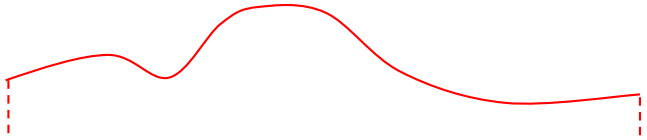
- échantillonnage régulier du plan horizontal
- stockage des altitudes pour chaque point du plan horizontal



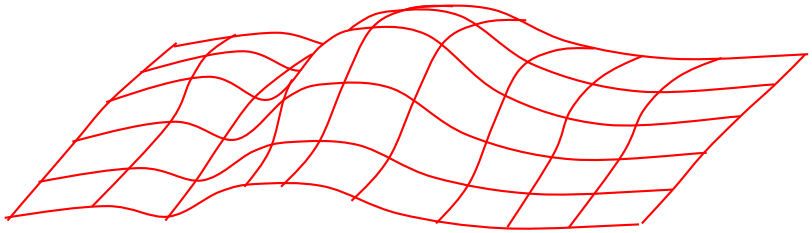
Echantillonage



Stockage des sommets



Fonction qui effectue le tracé
du profil de la ligne i :
ligne brisée qui relie les sommets



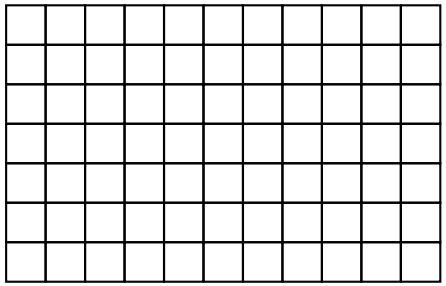
Tracé des profils de ligne et de colonne

```
#define Nblig 256
#define NBcol 512

typedef float Point3D[3];
Point3D mat[NBlig][NBcol]; /* qu'il faudra initialiser */

void tracerProfilLigne(int i)
{ int j;
  glBegin(GL_LINE_STRIP);
    for (j=0 ; j<Nbcol ; j++) glVertex3fv(mat[i][j]);
  glEnd();
}

void tracerSurface(void)
{ int i,j;
  for (i=0 ; i<Nblig ; i++)
    tracerProfilLigne(i);
  for (j=0 ; j<Nbcol ; j++)
    tracerProfilColonne(j);
}
```

Remarque : *On choisira éventuellement la taille de la maille en fonction de la distance au point de vue*

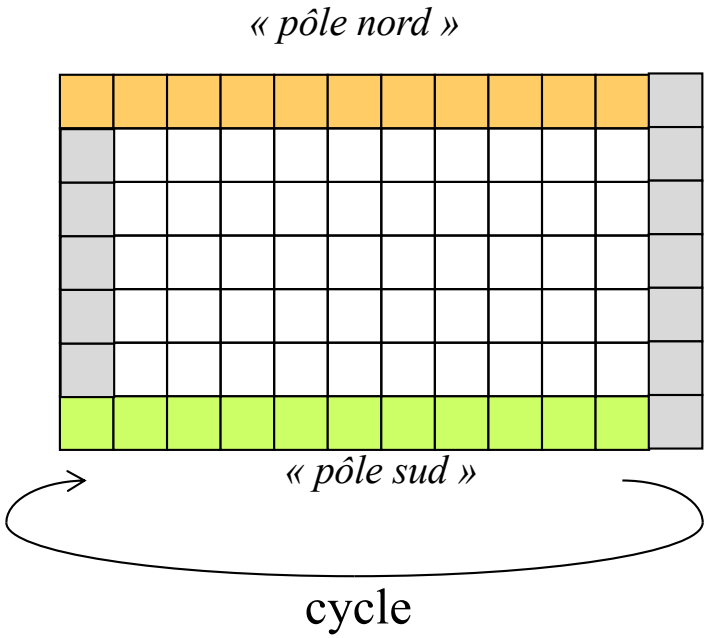
Ex : sphère de rayon 1

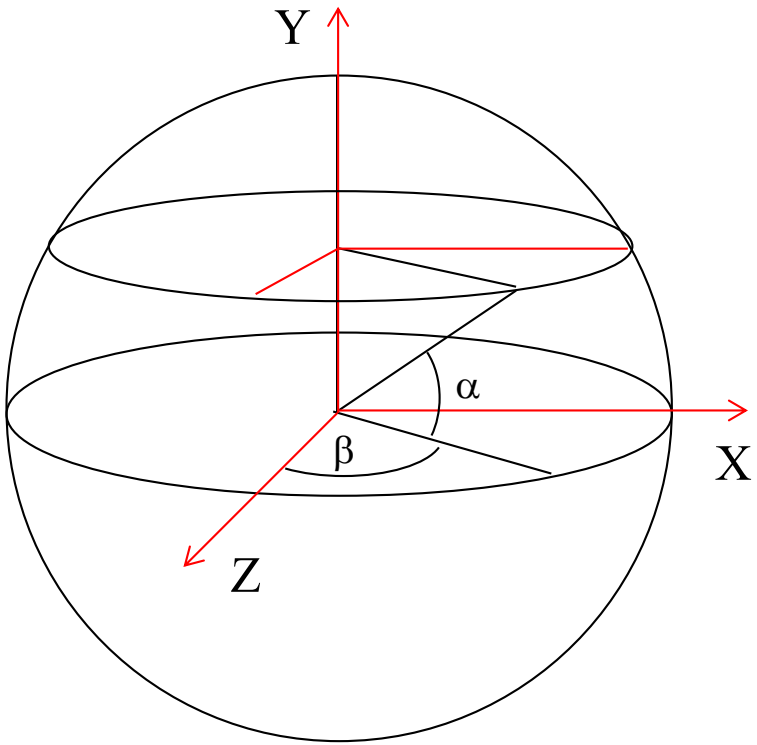
- représentation cylindrique
- avec des parallèles et des méridiens



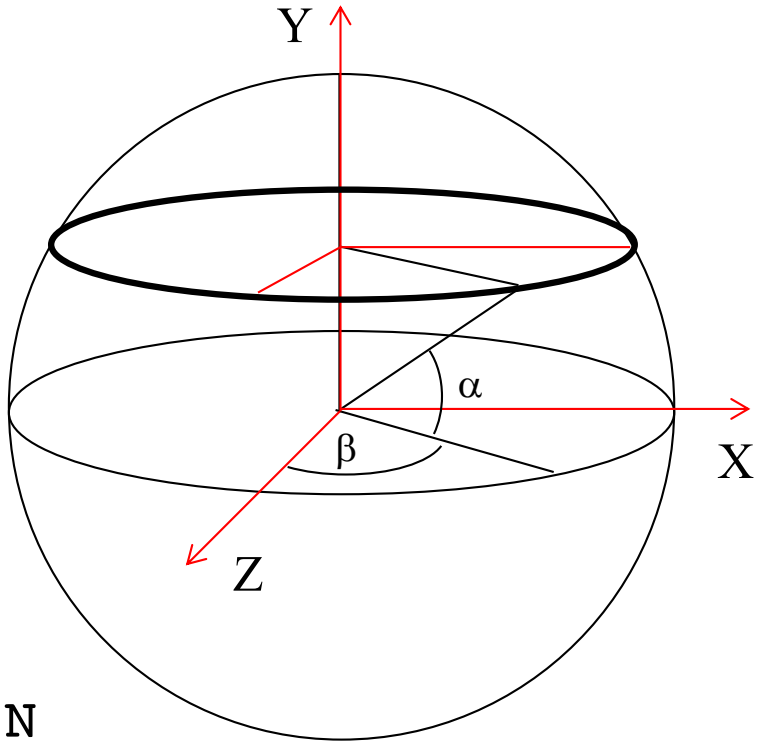
```
#define NBlig 19
#define NBcol 37

typedef float Point3D[3];
Point3D sphere[NBlig][NBcol];

void initSphere(float rayon)
{… }

void tracerSphere(void)
{… }
```

```
void initSphere(void)  /* de rayon 1 */
{ choisir un pas d'échantillon des latitudes
  choisir un pas d'échantillon des longitudes
  Pour chaque latitude du sud au nord
      échantillonner cette latitude
  Fin pour
}
```



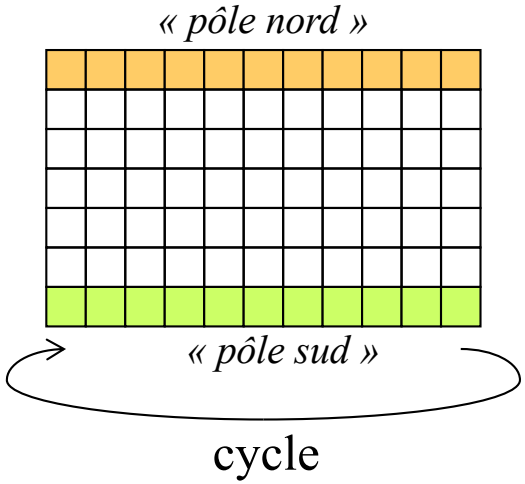


```
void initSphere(void)  /* de rayon 1 */
{int i,j;
 float r, y, alpha, beta, dAlpha, dBeta;
 dAlpha = PI/(NBlig-1);        /* pas de l'angle alpha */
 dBeta = 2*PI/(NBcol-1);       /* pas de l'angle beta */
 // Latitudes
 for (i=0 ; i<NBlig ; i++)
 {alpha = dAlpha*i - PI/2;
  y = sin(alpha);
  r = cos(alpha);
  // Meridiens de la latitude i
  for (j=0 ; j<NBcol ; j++)
  { beta = dBeta*j;
    sphere[i][j][2] = r*cos(beta);
    sphere[i][j][0] = r*sin(beta);
    sphere[i][j][1] = y;
  }
 }
 /*Le meridien 360°  (ou 0) a été
   stocké en derniere colonne
   pour refermer la sphere.
   La ligne 0 contient le pole S
   La ligne NBlig-1 contient le pole N
 */
}
```



```
Action tracerSphere(void)
{ Pour chaque latitude du sud au nord
    tracer les mailles de cette latitude
  Fin pour
}
```

```
void tracerSphere(void)
{int i,j;
 glBegin(GL_QUADS) ;
 for (i=0 ; i<NBlig-1 ; i++)
  for (j=0 ; j<NBcol-1 ; j++)
  { glVertex3fv(sphere[i][j]);
    glVertex3fv(sphere[i][j+1]);
    glVertex3fv(sphere[i+1][j+1]);
    glVertex3fv(sphere[i+1][j]);
  }
 // le meridien 0 a ete dupliqué en
 // derniere colonne de sphere
 // pour refermer la sphere
 glEnd() ;
}
```



Remarque : pour un tracé fil de fer : `glPolygonMode(GL_FRONT, GL_LINE)`

Résumé

Principe d'un tracé

- openGL : machine à états
- Définir d'abord le contexte de tracé (variables d'état)
- Effectuer ensuite le tracé
- Chaque primitive énoncée est projetée immédiatement
- Modification du contexte => reconstruction de la scène

Exemple de variable d'état et utilisation :

- Définir le fond `glClearColor(`*R*`,`*V*`,`*B*`)`
- Effacer l'image `glClear(GL_COLOR_BUFFER_BIT)`
- Définir la couleur avant tracé `glColor3f(`*R*`,`*V*`,`*B*`)`

Résumé

Programmation événementielle :

- Variables d'état spécifiques à openGL
- Variables d'état spécifiques à l'application (variables globales)
- Boucle d'événements :
  - Les fct liées à une interruption modifient un certain nbre de variables d'état
  - La fonction qui trace la scène utilise ces variables d'état pour définir le contexte de tracé et effectue le tracé

Gestion des interruption (GLUT)

- Définir et associer une fonction pour chaque type d'interruption
- C'est le système qui déclenche ces fonctions
- Une seule fonction chargée de tracer la scène
- **`glutPostRedisplay()`** pour exécuter cette fonction
- Modification du contexte d'affichage => **`glutPostRedisplay()`**

Résumé

Canevas de la fonction d'affichage :

```
void afficheMaScene(void)
{ glClear(GL_COLOR_BUFFER_BIT);
  la caméra est en O tournée vers les Z négatifs
  tracerObjet1() ;
  tracerObjet2();
   …
  glutSwapBuffers();
}

void tracerObjet_i(void)
{ glColor3f(r, v, b);
  glBegin(primitives);
    … glVertex3fv(sommet); …
  glEnd();
}
```

Résumé

Tracé d'une primitive

- Primitive = suite de sommets `glVertex3fv(`S_i`)`
- Structure : `glBegin(`*`primitive`*`)`

  suite de sommets

  `glEnd();`

Type de primitives

- `GL_POINTS`
- `GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP`
- `GL_TRIANGLES`
- `GL_QUADS, GL_POLYGON` (S_i coplanaires, arrêtes non sécantes)

Résumé

Orientation d'une primitive (face avant, face arrière)

- Face avant (sens trigo) `GL_FRONT`
- Face arrière (sens horaire) `GL_BACK`
- Face avant et arrière `GL_FRONT_AND_BACK`

- Polygones pleins `glPolygonMode(`*`face`*`, GL_FILL)`
- Polygones vides `glPolygonMode(`*`face`*`, GL_LINE)`

- Définir les faces non visibles (gain de temps !)
  - `glCullFace(GL_BACK)` désignation de la face non visible
  - `glEnable(GL_CULL_FACE)` activée
  - `glDisable(GL_CULL_FACE)` désactivée