Transformations générales

Transformations générales

- La forme générale d'une telle matrice est la suivante:

$$\begin{bmatrix} a & b & m \\ c & d & n \\ 0 & 0 & 1 \end{bmatrix}$$

- Pour des opérations en 2D, on retrouve les cas particuliers suivants :

Mise à l'échelle

$$\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

rotation

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

translation

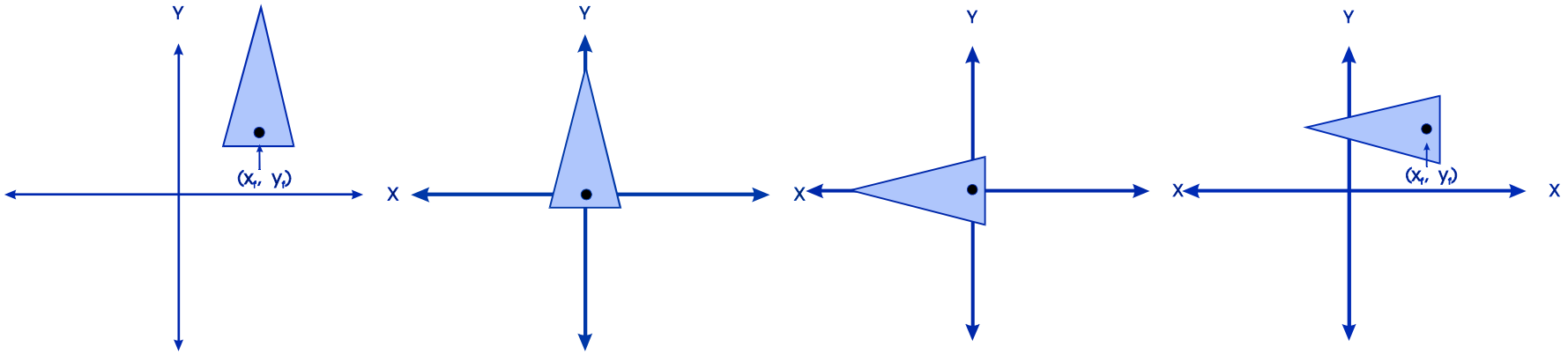
$$\begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix}$$

Transformations générales

- On souhaite enchaîner les transformations élémentaires (mise à l'échelle autour de l'origine, rotation autour de l'origine, translation) pour produire des transformations plus générales
- Ces transformations générales sont représentées par une suite de produits matriciels utilisant les matrices de transformations élémentaires
- Il est alors possible de définir plusieurs transformations plus générales. Par exemple :
  - Rotation autour d'un point arbitraire
  - Homothétie autour d'un point arbitraire
  - Réflexion par rapport à une droite arbitraire
  - Rotation 3D autour d'un axe arbitraire

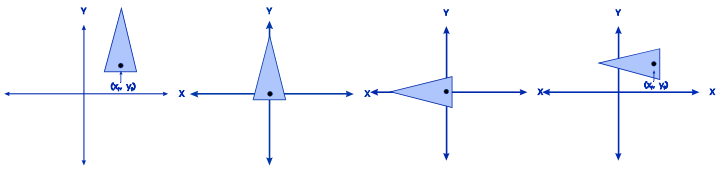
Rotation autour d'un point arbitraire

- Une rotation autour d'un point arbitraire $P_f=(x_f, y_f)$ comprend un déplacement, une rotation autour de l'origine et finalement un autre déplacement



Rotation autour d'un point arbitraire

- Une rotation autour d'un point arbitraire $P_f=(x_f, y_f)$ comprend un déplacement, une rotation autour de l'origine et finalement un autre déplacement
- Mathématiquement, ceci s'écrit :



$$T_{P_f} \cdot R(\theta) \cdot T_{-P_f}$$

$$= \begin{bmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta & (1-\cos\theta)x_f + y_f \sin\theta \\ \sin\theta & \cos\theta & (1-\cos\theta)y_f - x_f \sin\theta \\ 0 & 0 & 1 \end{bmatrix}$$

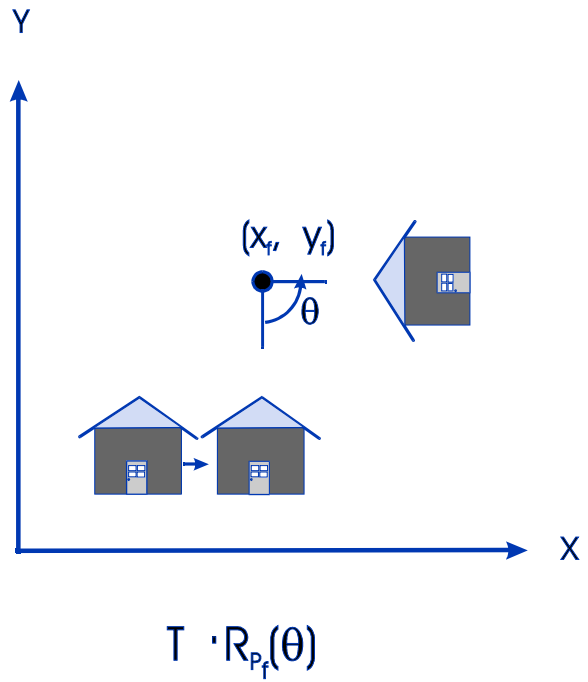
Rotation autour d'un point arbitraire

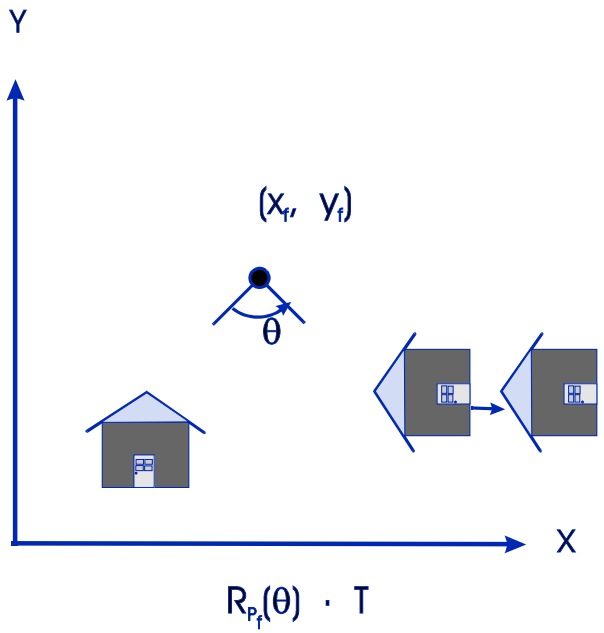
- ou avec les équations suivantes:

$$x^* = x_f + (x - x_f)\cos\theta - (y - y_f)\sin\theta$$
$$y^* = y_f + (x - x_f)\sin\theta + (y - y_f)\cos\theta$$

Rotation autour d'un point arbitraire

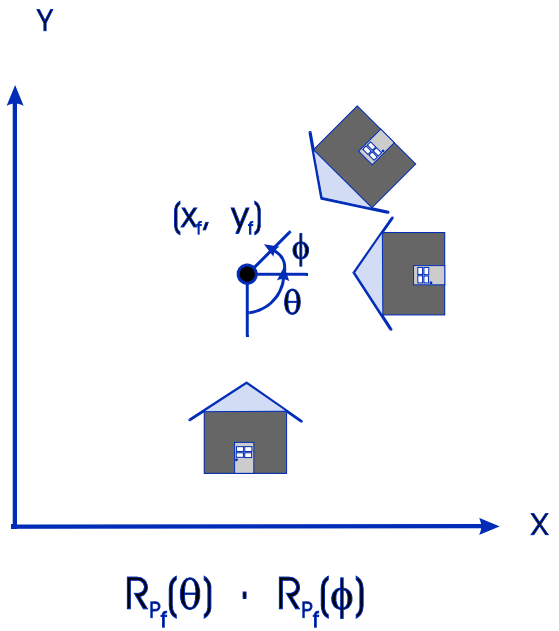
- De façon générale, le produit de matrices ne commutent pas.



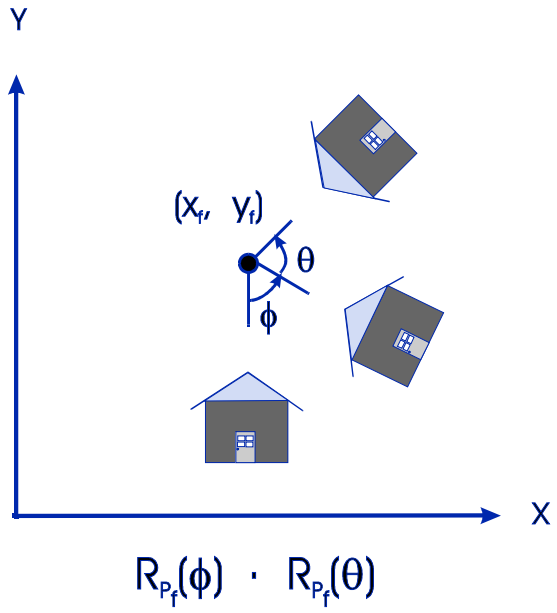


Rotation autour d'un point arbitraire

- Pour certains cas particuliers, le produit de matrices commute. C'est le cas de deux matrices correspondant à la même transformation.
- Par exemple, deux rotations peuvent être inversées :



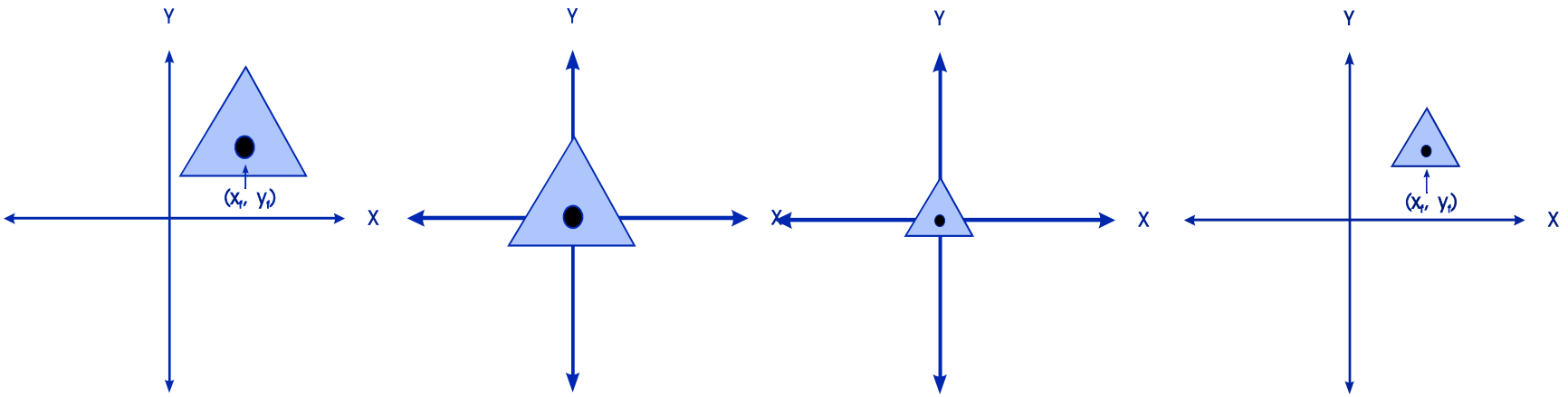
$R_{P_f}(\theta) \cdot R_{P_f}(\phi)$



$R_{P_f}(\phi) \cdot R_{P_f}(\theta)$

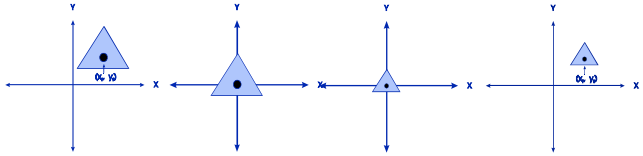
Homothétie autour d'un point arbitraire

- Une homothétie (mise à l'échelle) par rapport à un point arbitraire $P_f=(x_f, y_f)$ comprend une translation du point vers l'origine, une mise à l'échelle et une translation pour ramener le point à sa position initiale



Homothétie autour d'un point arbitraire

- Une homothétie (mise à l'échelle) par rapport à un point arbitraire $P_f=(x_f, y_f)$ comprend une translation du point vers l'origine, une mise à l'échelle et une translation pour ramener le point à sa position initiale
- Mathématiquement, ceci s'écrit :



$$T_{P_f} \cdot S \cdot T_{-P_f}$$

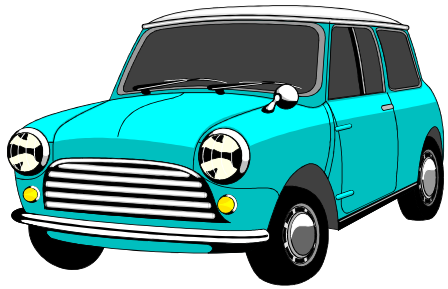
$$= \begin{bmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} S_x & 0 & (1-S_x)x_f \\ 0 & S_y & (1-S_y)y_f \\ 0 & 0 & 1 \end{bmatrix}$$

Homothétie autour d'un point arbitraire

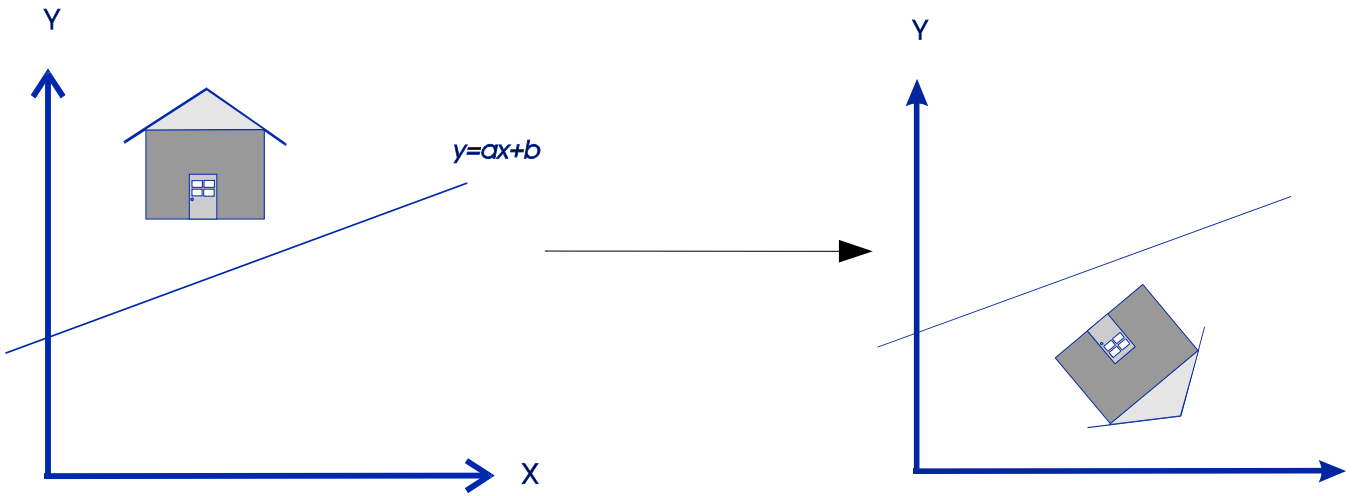
- Ou avec les équations suivantes :

$$x^* = (x - x_f) S_x + x_f$$
$$y^* = (y - y_f) S_y + y_f$$

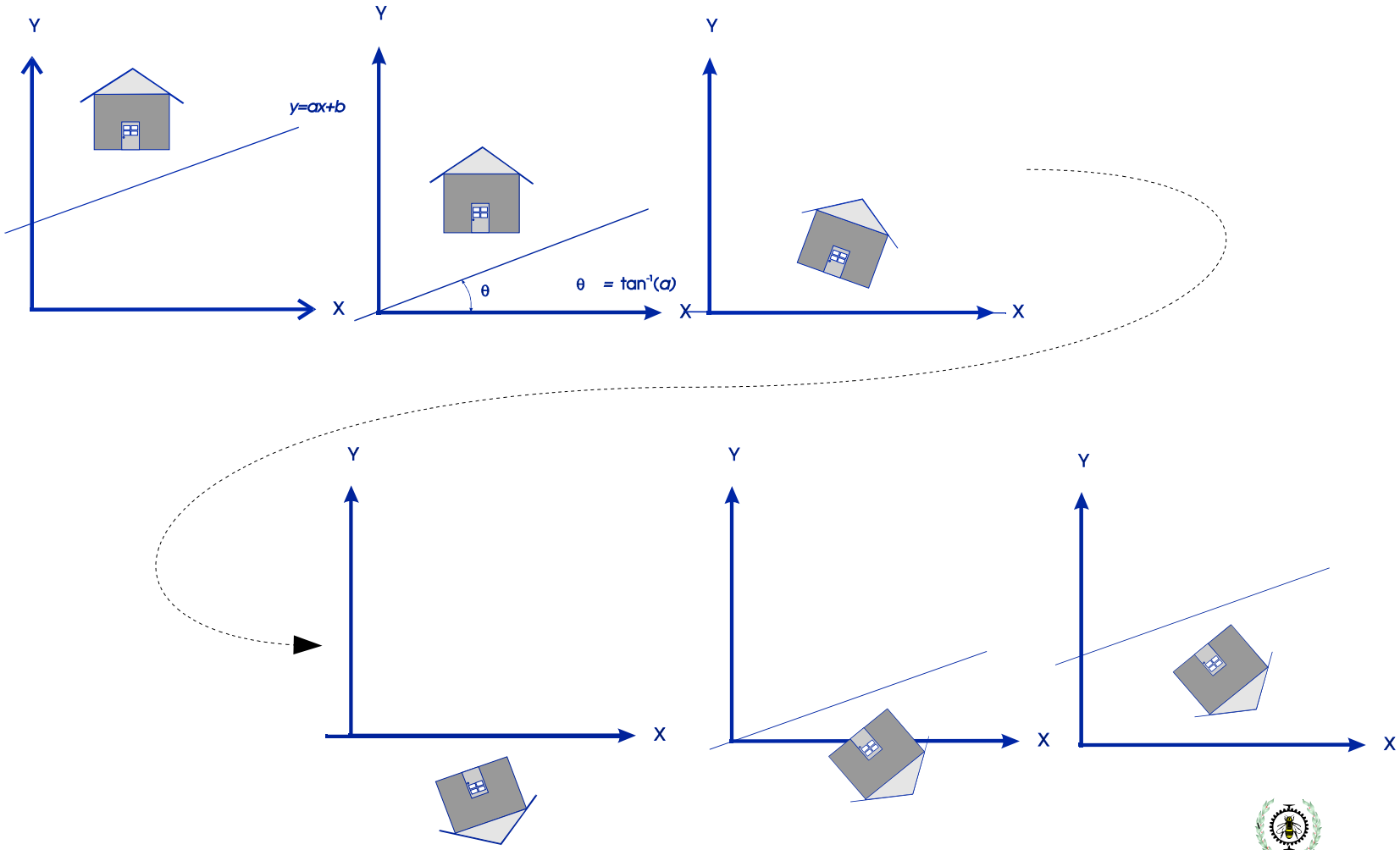


Réflexion par rapport une droite arbitraire

- Une réflexion par rapport une droite arbitraire :
  - Translation de la droite et de l'objet de sorte que la droite passe par l'origine,
  - Rotation pour faire coïncider la droite avec un axe principal,
  - Réflexion par rapport à cet axe,
  - Rotation inverse,
  - Translation inverse.



Réflexion par rapport une droite arbitraire



Réflexion par rapport une droite arbitraire

- Mathématiquement, l'enchaînement des transformations est

$$R_{ax+b} = T^{-1}_{(0,-b)} \cdot R_x^{-1}(\theta) \cdot R_x \cdot R_x(\theta) \cdot T_{(0,-b)}$$

où

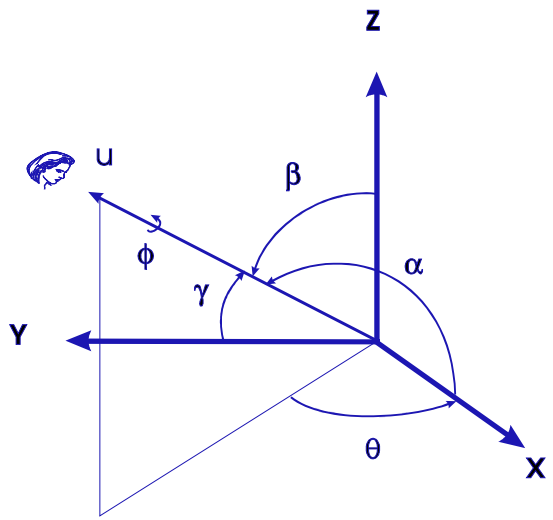
$$T_{(0,-b)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -b \\ 0 & 0 & 1 \end{bmatrix}$$

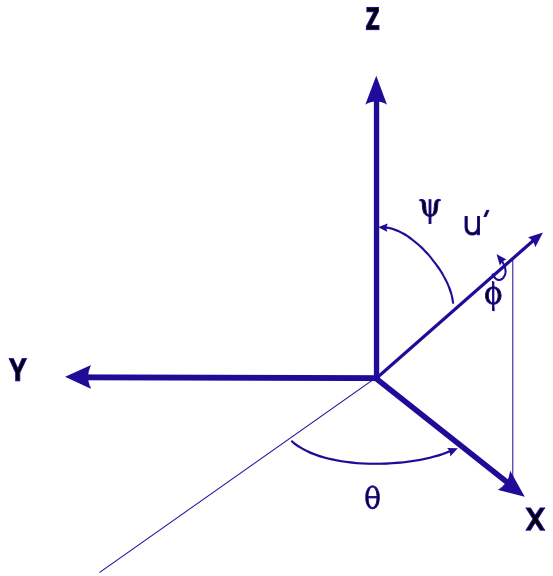
et $R_x(\theta)$: rotation par rapport à l'origine pour faire coïncider la droite $y=ax+b$ avec l'axe des x,

et $\theta = tan^{-1}(a)$, R_x : la réflexion par rapport à l'axe des x.

Rotation 3D autour d'un axe arbitraire

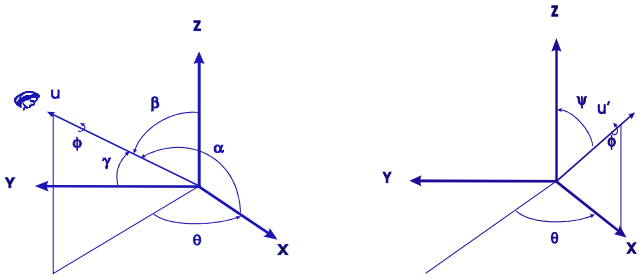
- Une rotation 3D autour d'un axe arbitraire peut être décomposé ainsi :
  - Effectuer une transformation d'alignement de façon à aligner le vecteur ***u*** sur l'axe des z,
  - Effectuer une rotation d'un angle ϕ autour de l'axe des z,
  - Effectuer la transformation d'alignement inverse de façon à repositionner le vecteur ***u*** à sa position originale





Rotation 3D autour d'un axe arbitraire

- La transformation d'alignement $A_z(u)$ du vecteur u avec l'axe des z: une rotation autour de l'axe des z d'un angle $-\theta$ pour former un nouvel axe u' dans le plan x-z, suivie d'une rotation d'un angle $-\psi$ autour de l'axe des y ou sous forme mathématique $A_z(u) = R_z(-\theta).\ R_y(-\psi).$



$$R_u(\varphi) = R_z^{-1}(-\theta) \bullet R_y^{-1}(-\psi) \bullet R_z(\varphi) \bullet R_y(-\psi) \bullet R_z(-\theta)$$

Transformations inverses

- Pour chaque transformation, il existe un inverse qui ramène l'objet au point de départ. Mathématiquement,

$$PTT^{-1} = PI = P$$

  - Une translation d'un vecteur V=(Tx, Ty, Tz)

$$T_V^{-1} = T_{-V}$$

  - Une rotation d'un angle θ autour d'un axe u

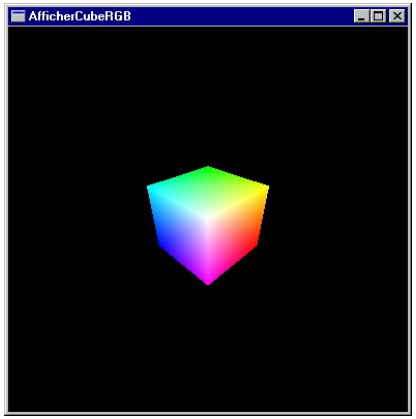
$$R_u^{-1}(\theta) = R_u(-\theta)$$

  - Une homothétie

$$S^{-1}_{(S_x, S_y, S_z)} = S_{\left(\frac{1}{S_x}, \frac{1}{S_y}, \frac{1}{S_z}\right)}$$

Afficher un CUBE

- Fonctions d'affichage :




```
#define AXE_X      0
#define AXE_Y      1
#define AXE_Z      2
void Afficher( )
{
  glClear ( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();
  glRotatef( theta[AXE_X], 1.0, 0.0, 0.0 );
  glRotatef( theta[AXE_Y], 0.0, 1.0, 0.0 );
  glRotatef( theta[AXE_Z], 0.0, 0.0, 1.0 );
  TracerCube();
  glutSwapBuffers();
}
```

Exemple
POSITIONNEMENT DE LA CAMÉRA

```
typedef struct {
        Point3D Observateur;
        Point3D PtVise;
        Point3D VUP; /* View UP Vector */
} Camera_T;
```

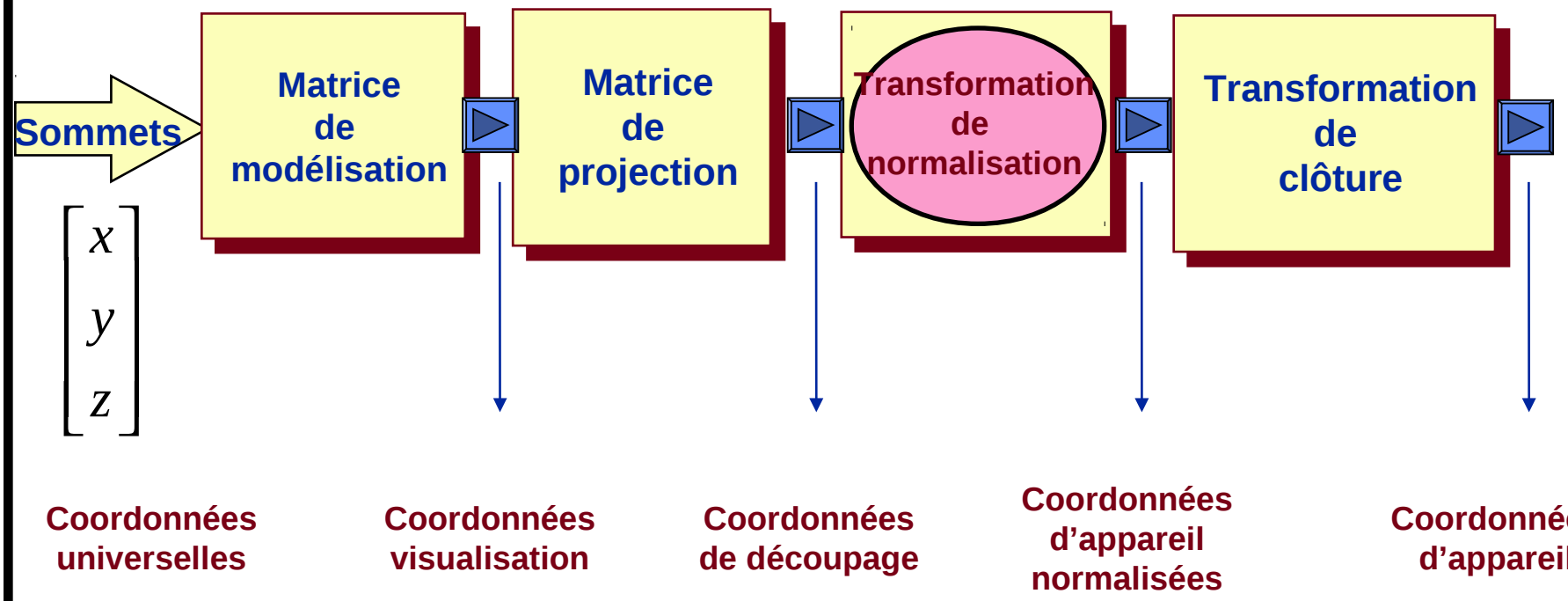
```
void PositionnerObservateur( Camera_T Camera )
{
   glMatrixMode( GL_MODELVIEW );
   glLoadIdentity( );
   gluLookAt( Camera.Observateur.x, Camera.Observateur.y, Camera.Observateur.z,
              Camera.PtVise.x, Camera.PtVise.y, Camera.PtVise.z,
              Camera.VUP.x, Camera.VUP.y, Camera.VUP.z );
}
```

Exemple
POSITIONNEMENT DE LA CAMÉRA

```
void Dessiner( )
{
   glMatrixMode ( GL_MODELVIEW );
   glTranslate ( 0.0, 0.0, -10.0 );
   glRotate ( 90.0, 0.0, 1.0, 0.0 );
   glBegin(...)
   ...
}
```

Découpage du volume de visualisation

PIPELINE DES TRANSFORMATIONS OpenGL



Découpage du volume de visualisation

Trois approches possibles:

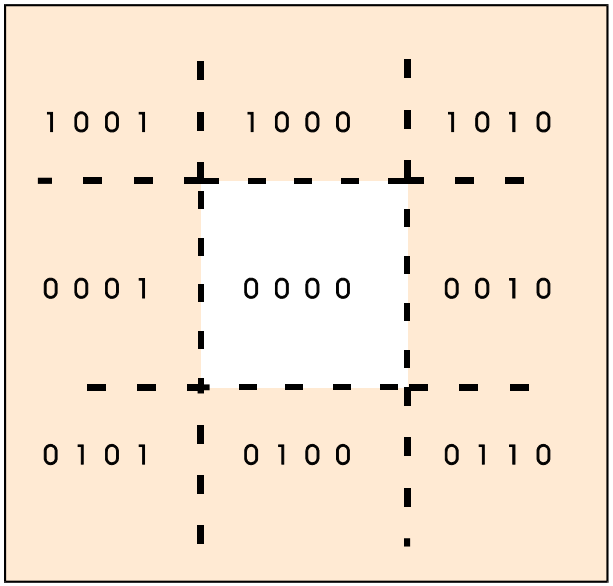
- *AVANT* LA CONVERSION en pixels: calcul analytique des intersections avec la frontière de la région de découpage
  - abordable pour des formes simples (algorithme Cohen-Sutherland pour segment découpé par région rectangulaire)
- *PENDANT* LA CONVERSION en pixels: on construit un masque de la région puis on vérifie que chaque pixel devant être « allumé » y figure
  - permet de représenter des régions de découpage arbitrairement complexes (stencil d'OpenGL)
- *APRÈS* LA CONVERSION en pixels: on convertit tout l'objet virtuel dans un espace mémoire temporaire puis on copie le sous-ensemble d'intérêt (clôture)
  - efficace si on fait face à un objet complexe coûteux à convertir et qu'on déplace la région de découpage de façon interactive (image ou texture OpenGL)

Algorithme Cohen-Sutherland

Découpage d'un segment par une région rectangulaire :

- définit 9 zones identifiées par des codes de 4 bits
- Soient code1 et code2, les codes de chacune des deux extrémités courantes du segment

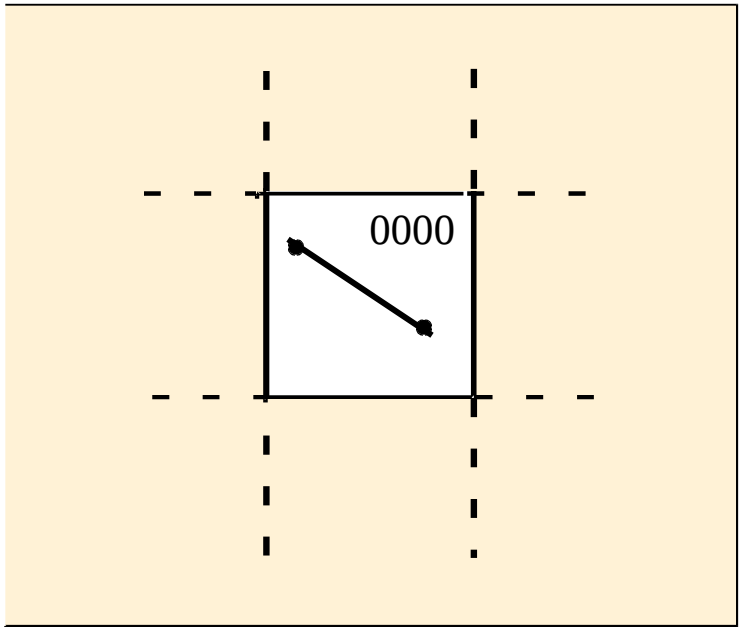
# bit	Description
1	le point se trouve à gauche du côté gauche
2	le point se trouve à droite du côté droit
3	le point se trouve en bas du bas
4	le point se trouve en haut du haut



Algorithme Cohen-Sutherland

- Si *code1* = 0000 et *code2* = 0000 alors

  le segment est trivialement à l'intérieur

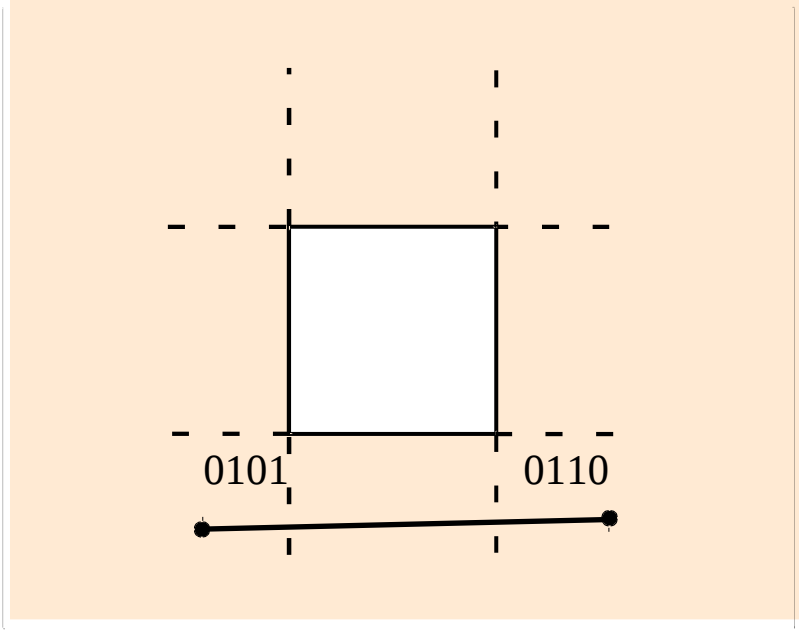


Algorithme Cohen-Sutherland

- Si (*code1* AND *code2*) ≠ 0000 alors

  le segment est trivialement à l'extérieur

$$0101 \wedge 0110 = 0100 \neq 0000$$



Algorithme Cohen-Sutherland

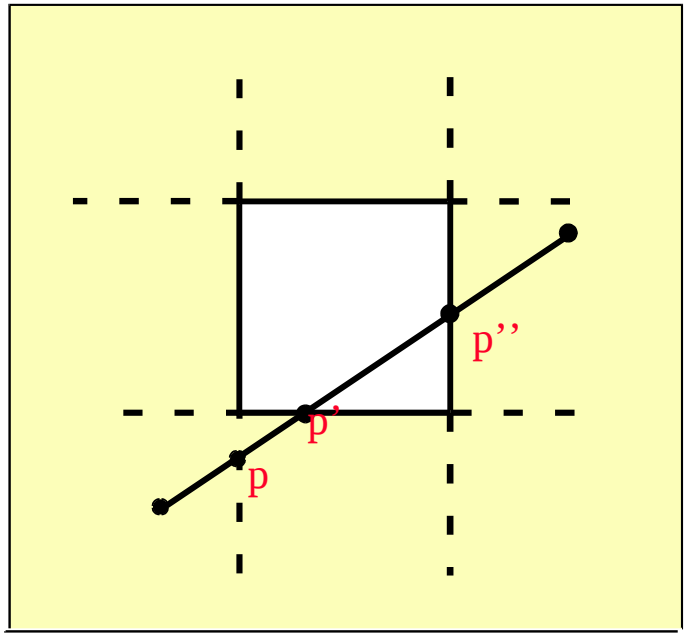
a) choisir une extrémité à l'extérieur (dont le code ≠ 0000)

b) choisir son bit allumé le plus à droite (ou selon autre convention)

c) calculer l'intersection *p* entre le segment et la frontière correspondante de la région

d) *p* remplace l'extrémité à l'extérieur; calculer son code

e) appliquer à nouveau l'algorithme



Découpage du volume normalisé de visualisation

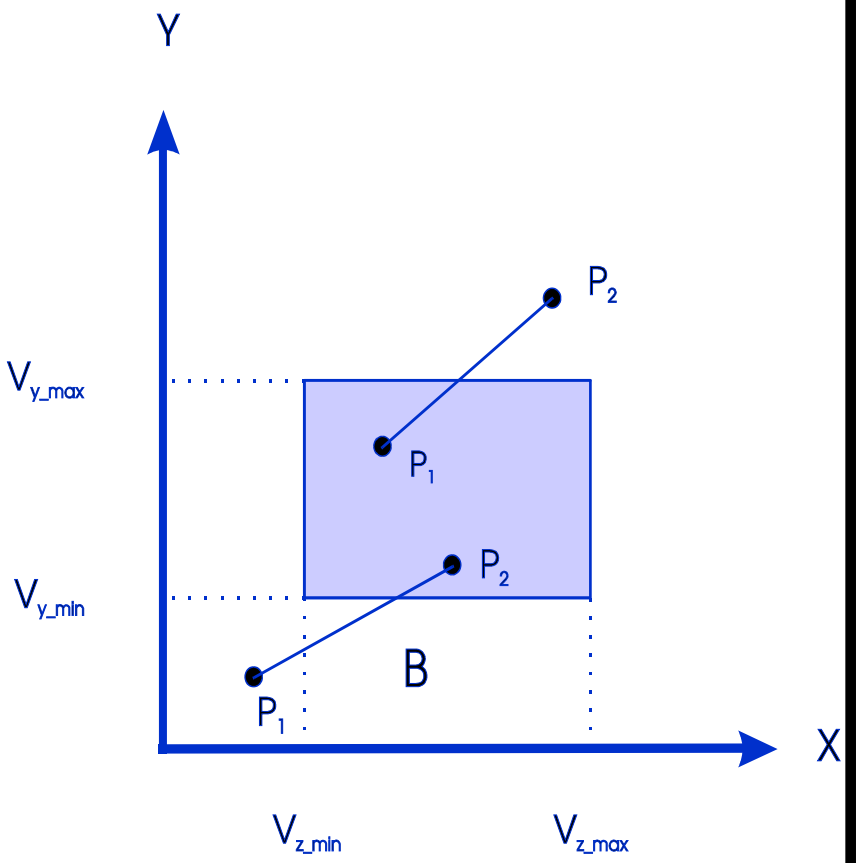
Une généralisation 3D de l'algorithme de Cohen-Sutherland

- Le découpage de segments de lignes en 3D est simplement une généralisation de l'algorithme de Cohen-Sutherland pour le découpage de segments de lignes en 2D.
- En prolongeant les frontières de la fenêtre 3D, on assigne à chacune de ces régions un code d'extrémité de six bits numérotés de droite à gauche et définis ainsi:

# bit	Condition	
1	si $x < V_{x\_min}$	(gauche)
2	si $x > V_{x\_max}$	(droite)
3	si $y < V_{y\_min}$	(dessous)
4	si $y > V_{y\_max}$	(dessus)
5	si $z < V_{z\_min}$	(avant)
6	si $z > V_{z\_max}$	(arrière)

Découpage du volume normalisé de visualisation

- Si les deux points sont à l'intérieur, alors ils ont le code d'extrémité 000000 et le segment de deux segments de lignes à droite est visible.
- Si un des deux points n'a pas un code d'extrémité 000000, alors on effectue l'opération logique et sur les deux codes d'extrémité.
- Le résultat de cette opération sera différent de zéro si le segment est complétement à l'extérieur du volume de visualisation.
- Dans le cas contraire, on trouve l'intersection des segments de droite avec les plans definissant le volume de visualisation.



Découpage du volume normalisé de visualisation

- L'intersection avec les plans doit être calculée efficacement. L'équation paramétrique d'un segment de droite entre deux points $P_1 = (x_1, y_1, z_1)$ et $P_2 = (x_2, y_2, z_2)$ s'écrit:

$$\vec{P}(u) = P_1 + (P_2 - P_1)u$$

- ou encore, pour chacune des composantes:

$$x = x_1 + (x_2 - x_1)\,u$$
$$y = y_1 + (y_2 - y_1)\,u$$
$$z = z_1 + (z_2 - z_1)\,u$$

  - On obtient P_1 pour $u = 0$ et P_2 pour $u = 1$, et les points entre P_1 et P_2 pour $0 < u < 1$
  - Pour trouver l'intersection avec un plan, il suffit de substituer par exemple pour $z = V_{z\_min}$,

$$u = \frac{V_{z\_\min} - z_1}{z_2 - z_1}$$

Découpage du volume normalisé de visualisation

– Si u n'est pas dans l'intervalle $[0, 1]$, ceci signifie que le segment de droite ne coupe pas le plan entre les points P_1 et P_2. Si $0 \leq u \leq 1$, alors on calcule l'intersection pour les coordonnées en x et y

$$x_I = x_1 + (x_2 - x_1)\left(\frac{V_{z\_min} - z_1}{z_2 - z_1}\right)$$

$$y_I = y_1 + (y_2 - y_1)\left(\frac{V_{z\_min} - z_1}{z_2 - z_1}\right)$$

– Si x_I ou y_I ne sont pas dans la plage des frontières de la clôture, ceci signifie que l'intersection est en dessous des frontières du volume.

Utilisation de tableaux de sommets

Primitives géométriques -- *OpenGL :* utilisation de tableaux

1) Au lieu de transmettre les sommets un à un avec `glVertex()`, on peut utiliser plutôt un tableau de sommets, afin de réduire le nombre d'appels OpenGL pour tracer une primitive

- Pour tracer les six faces d'un cube avec la primitive `GL_QUADS`, on aurait toutefois encore besoin de définir

6 faces * 4 sommets/face = tableau de 24 sommets (3 coo./sommet)

2) Afin d'éliminer la redondance des sommets, on peut définir plutôt un tableau de sommets uniques et un autre tableau de connectivité qui indique comment relier ces sommets

- Pour tracer les six faces d'un cube avec cette stratégie, on aurait alors besoin d'un tableau de 8 sommets et d'un autre tableau de

6 faces * 4 indices/face = 24 indices

Primitives géométriques -- *OpenGL :* tableau de sommets

- 1) Exemple d'utilisation d'un tableau de sommets:

```
// définir les sommets
GLfloat sommets[24*3] = {...}; // les 24 sommets (x,y,z)

// activer et spécifier un pointeur vers les sommets
glEnableClientState( GL_VERTEX_ARRAY );
glVertexPointer( 3, GL_FLOAT, 0, sommets );

// tracer la primitive
glDrawArrays( GL_QUADS, 0, 24 ); // 24=sizeof(sommets)/(3*sizeof(GLfloat))

// désactiver l'utilisation des tableaux de sommets
glDisableClientState( GL_VERTEX_ARRAY );
```

Primitives géométriques -- *OpenGL :* tableau de sommets

- Activer ou désactiver l'utilisation de tableaux :

```
void glEnableClientState( GLenum cap );
void glDisableClientState( GLenum cap );
```

- Où `cap` est une des constantes :
  - `GL_VERTEX_ARRAY`
  - `GL_COLOR_ARRAY`
  - `GL_EDGE_FLAG_ARRAY`
  - `GL_INDEX_ARRAY`
  - `GL_NORMAL_ARRAY`
  - `GL_TEXTURE_COORD_ARRAY`

Primitives géométriques -- *OpenGL :* tableau de sommets

Définir un tableau de sommets :

```
void glVertexPointer( GLint taille, GLenum type,
                      GLsizei pas, const GLvoid *ptr );
```

`taille` : nombre de coordonnées par sommet (2,3, ou 4)

`type` : type des données `(GL_FLOAT, GL_SHORT, GL_INT, GL_DOUBLE)`

`pas` : nombre d'octets entre deux sommets consécutifs

(si 0, alors il n'y a rien de plus entre chaque sommet)

`ptr` : pointeur sur les sommets

Primitives géométriques -- *OpenGL :* tableau de sommets

Effectuer le rendu de la primitive :

```
void glDrawArrays( GLenum mode,
                   GLint debut, GLsizei nombre );
```

`mode` : choix de la primitive.

Les mêmes modes que pour `glBegin()/glEnd()` :

`GL_POINTS, GL_LINE_STRIP, GL_LINE_LOOP, GL_LINES,`
`GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, GL_TRIANGLES,`
`GL_QUAD_STRIP, GL_QUADS, GL_POLYGON.`

`debut` : la position de la première valeur

`nombre` : nombre de valeurs à utiliser pour tracer

Primitives géométriques -- *OpenGL :* tableau de sommets et connectivité

- 2) Exemple d'utilisation d'un tableau de sommets et d'une connectivité:

```
// définir les sommets
GLfloat sommets[8*3] = {...}; // les sommets : 8 sommets * 3 coord.(x,y,z)
GLuint connec[6*4] = {...}; // la connectivité : 6 faces * 4 indices

// activer et spécifier un pointeur vers les sommets
glEnableClientState( GL_VERTEX_ARRAY );
glVertexPointer( 3, GL_FLOAT, 0, sommets );

// tracer la primitive
// glDrawArrays( GL_QUADS, 0, 24 );
glDrawElements( GL_QUADS, sizeof(connec)/sizeof(GLuint),
                GL_UNSIGNED_INT, connec );

// désactiver l'utilisation des tableaux de sommets
glDisableClientState( GL_VERTEX_ARRAY );
```

Primitives géométriques -- *OpenGL :* tableau de sommets et connectivité

Effectuer le rendu de la primitive avec un tableau de connectivité :

```
void glDrawElements( GLenum mode, Glsizei nombre,
                     Glenum type, const Glvoid * connec );
```

`mode` : choix de la primitive

```
(GL_POINTS, GL_LINE_STRIP, GL_LINE_LOOP, GL_LINES,GL_TRIANGLE_STRIP,
GL_TRIANGLE_FAN, GL_TRIANGLES, GL_QUAD_STRIP, GL_QUADS, GL_POLYGON)
```

`nombre` : nombre d'éléments à utiliser pour tracer

`type` : le type des indices

```
(GL_UNSIGNED_BYTE, GL_UNSIGNED_SHORT, GL_UNSIGNED_INT)
```

`connec` : un pointeur sur la position de la première valeur

Primitives géométriques -- *OpenGL :* tableaux sur le serveur

- Plutôt que de conserver les tableaux sur le client, on peut aussi envoyer les valeurs sur la carte graphique (le serveur graphique).

- Les *Vertex Buffer Object (VBO)* sont conservés sur le serveur et permettent d'améliorer l'efficacité de l'affichage
- On crée deux *VBO :*
  - un *VBO* pour les sommets et
  - un *VBO* pour la connectivité

- La création d'un *VBO* se fait en trois étapes:
  - Générer un id d'objet tampon avec `glGenBuffers()`.
  - Lier l'objet tampon avec `glBindBuffer()`.
  - Copier les données dans le tampon avec `glBufferData()`.

Primitives géométriques -- *OpenGL :* tableaux sur le serveur

Générer ou supprimer un(des) identificateur(s) de tampon :

```
void glGenBuffers( GLsizei n, GLuint* tampons );
void glDeleteBuffers( GLsizei n, const GLuint* tampons );
```

n : nombre désiré de noms

`tampons` : l'adresse d'un tableau où seront stockés les n noms

Primitives géométriques -- *OpenGL :* tableaux sur le serveur

Lier un objet tampon :

```
void glBindBuffer( GLenum type, GLuint tampon );
```

`type` : le type de tampon par l'une des quatre constantes
`GL_ARRAY_BUFFER` *(pour les sommets)*,
`GL_ELEMENT_ARRAY_BUFFER` *(pour la connectivité)*,
`GL_PIXEL_PACK_BUFFER, GL_PIXEL_UNPACK_BUFFER`

`tampon` : le nom de l'objet tampon

Primitives géométriques -- *OpenGL :* tableaux sur le serveur

Copier les données dans le tampon :

```
void glBufferData( GLenum type, GLsizeiptr taille,
                   const GLvoid * donnees, GLenum usage );
```

- `type` : le type de l'objet tampon par l'une des mêmes quatre constantes
- `taille` : la taille en octets
- `donnees` : les données à copier dans l'objet tampon
  - (ou `NULL` si on ne copie rien)
- `usage` : l'utilisation attendue de ce tampon

  `GL_STREAM_DRAW, GL_STREAM_READ, GL_STREAM_COPY,`
  `GL_STATIC_DRAW``, GL_STATIC_READ, GL_STATIC_COPY,`
  `GL_DYNAMIC_DRAW, GL_DYNAMIC_READ, GL_DYNAMIC_COPY`

Primitives géométriques -- *OpenGL :* tableaux sur le serveur

Création d'un tableau de sommets sur le serveur (VBO):

```
// le tableau de sommets
GLfloat* sommets = new GLfloat[nsommets*3];
...
// générer le VBO
GLuint vboId1;
glGenBuffers( 1, &vboId1 );
// lier l'objet tampon afin de pouvoir l'utiliser
glBindBuffer( GL_ARRAY_BUFFER, vboId1 );
// charger le tableau de sommets sur le serveur
glBufferData( GL_ARRAY_BUFFER, sizeof(sommets), sommets,
              GL_STATIC_DRAW );
// si on veut, on peut effacer les données après la création du VBO
delete [] sommets;
...
// supprimer le VBO à la fin du programme
glDeleteBuffers( 1, &vboId1 );
```

Primitives géométriques -- *OpenGL :* tableaux sur le serveur

Création d'un tableau de connectivité sur le serveur (VBO):

```
// le tableau de connectivité
GLuint* connec = new GLuint[nfaces*4];
...
// générer le VBO
GLuint vboId2;
glGenBuffers( 1, &vboId2 );
// lier l'objet tampon afin de pouvoir l'utiliser
glBindBuffer( GL_ELEMENT_ARRAY_BUFFER, vboId2 );
// charger le tableau de connectivité sur le serveur
glBufferData( GL_ELEMENT_ARRAY_BUFFER, sizeof(connec), connec,
              GL_STATIC_DRAW );
// si on veut, on peut effacer les données après la création du VBO
delete [] connec;
...
// supprimer le VBO à la fin du programme
glDeleteBuffers( 1, &vboId2 );
```

Primitives géométriques -- *OpenGL :* tableaux sur le serveur

Affichage d'une primitive avec VBO:

```
// lier VBOs pour les tableaux de sommets et de connectivité
glBindBuffer( GL_ARRAY_BUFFER, vboId1 );         // sommets
glBindBuffer( GL_ELEMENT_ARRAY_BUFFER, vboId2 ); // connectivité
// activer et spécifier un pointeur 0 vers les sommets
glEnableClientState( GL_VERTEX_ARRAY );
glVertexPointer( 3, GL_FLOAT, 0, 0 ); // ptr nul
// tracer la primitive: 6 faces * 4 indices/face = 24 indices
glDrawElements( GL_QUADS, 24, GL_UNSIGNED_BYTE, 0 ); // ptr nul
// désactiver l'utilisation des tableaux de sommets
glDisableClientState( GL_VERTEX_ARRAY );
// défaire le lien avec les VBO
glBindBuffer( GL_ARRAY_BUFFER, 0 );
glBindBuffer( GL_ELEMENT_ARRAY_BUFFER, 0 );
```