

VI.

Visualisation

1. Projection
2. Rastérisation
3. Z-buffer
4. La face cachée d'OpenGL



Laboratoire des Sciences de l'Information et des Systèmes
UMR CNRS 6168

1. La projection

La visualisation d'une scène nécessite deux types de transformation :

- des transformations 3D pour construire la scène :
matrice de modélisation (`GL_MODELVIEW`)
- la projection de cette scène 3D sur un plan de visualisation (2D) :
matrice de projection (`GL_PROJECTION`)

Une seule de ces deux matrices est modifiable à un moment donné

On bascule de l'une à l'autre avec :

- **`glMatrixMode(GL_PROJECTION)`**
- **`glMatrixMode(GL_MODELVIEW)`**

Modification des matrices de modélisation et de projection :

- GL_MODELVIEW : très fréquemment
- GL_PROJECTION : rarement

➔ on privilégie l'activation de la matrice de modélisation

➔ on active le mode projection uniquement le temps de définir une nouvelle projection

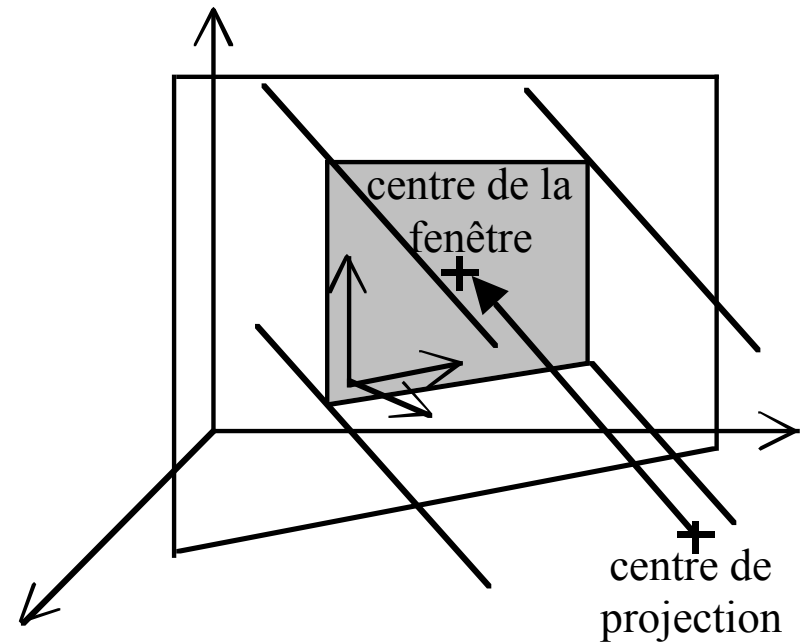
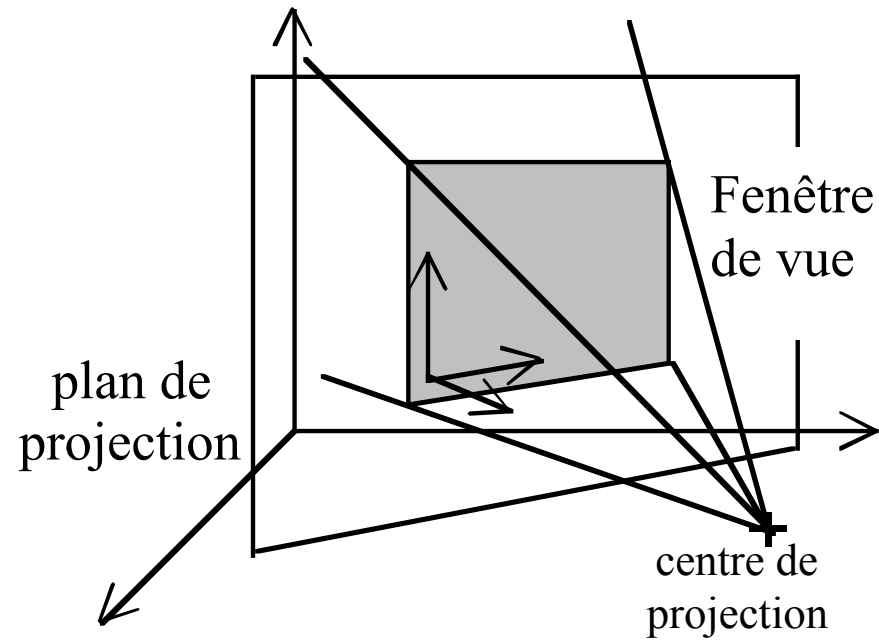
```
glMatrixMode(GL_PROJECTION) ;
```

définir la projection

```
glMatrixMode(GL_MODELVIEW) ;
```

Deux types de projection :

- **perspective**
- **parallèle**



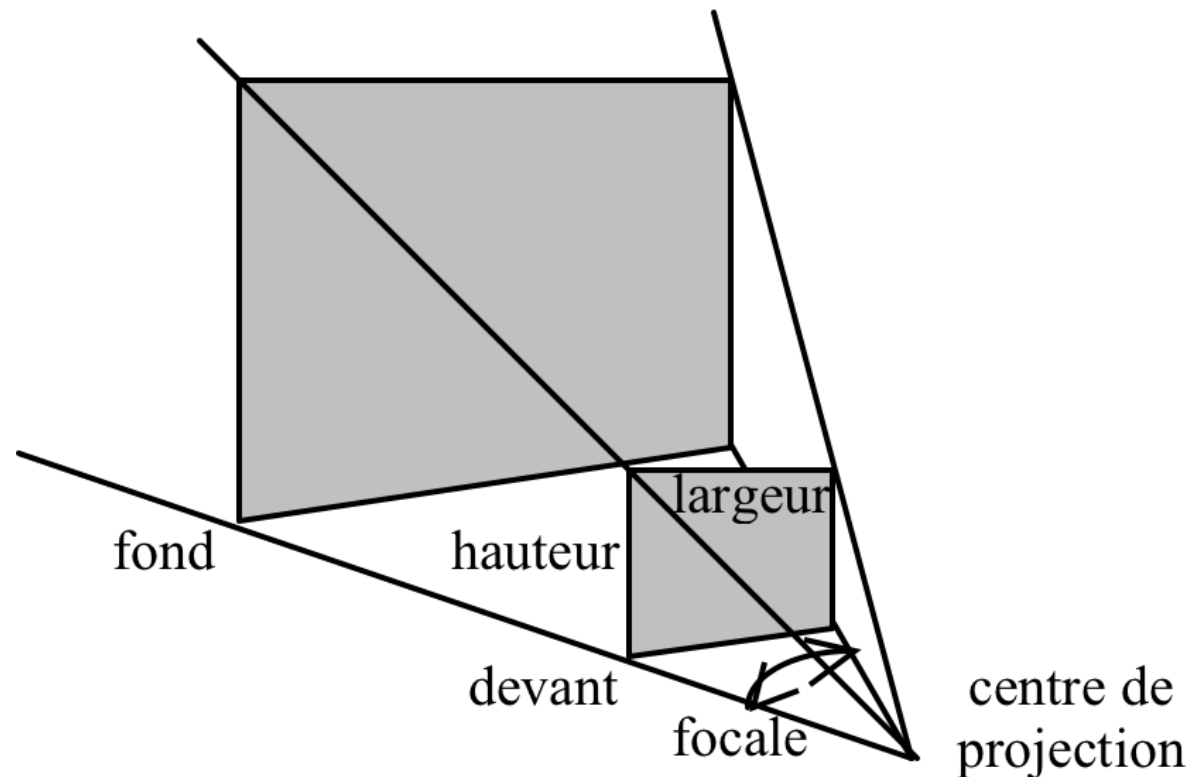
`gluPerspective(GLdouble focale, GLdouble aspect,
GLdouble devant, GLdouble fond)`

focale : angle du champ de vision (dans $[0^\circ, 180^\circ]$)

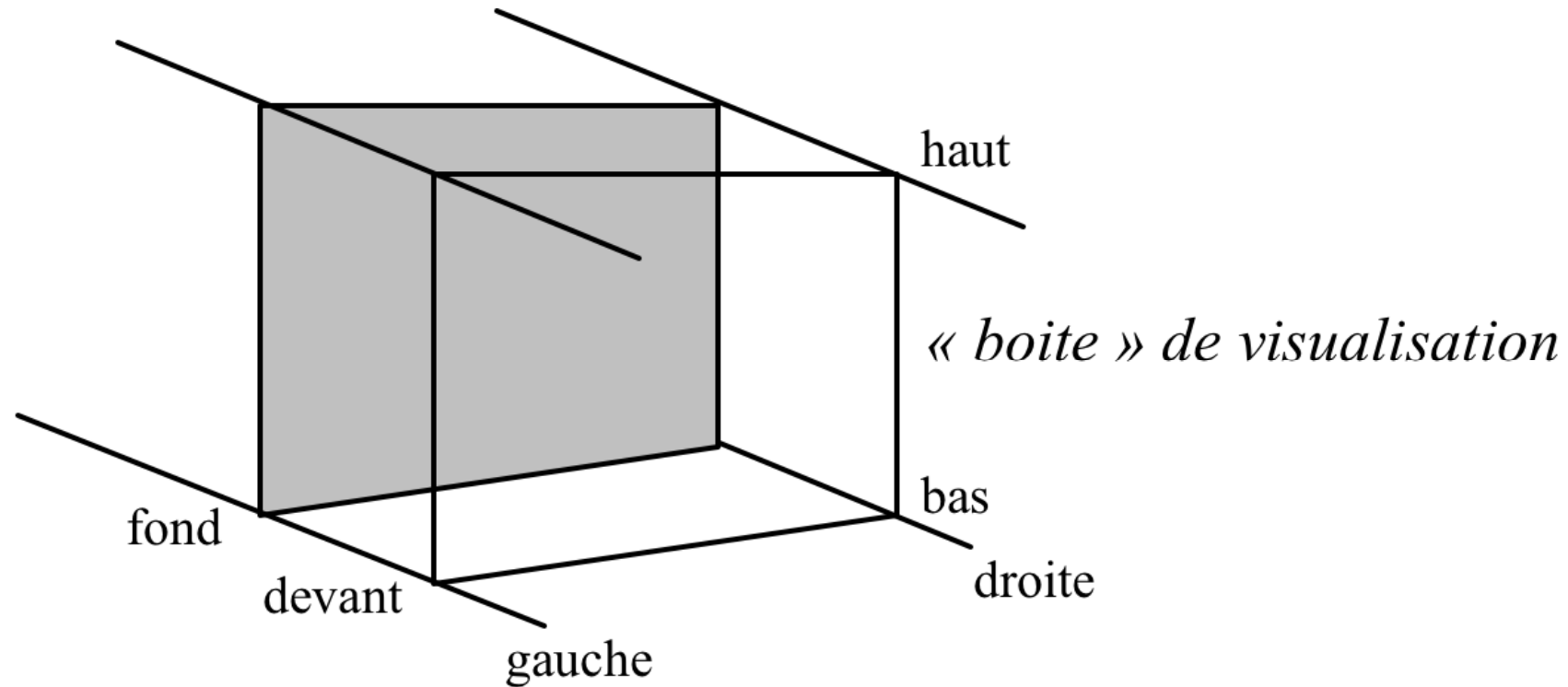
aspect : rapport largeur/hauteur du plan avant

devant : distance au plan de projection (**plan de clipping avant**)

fond : distance du point de vue au **plan de clipping arrière**



```
glOrtho(GLdouble gauche, GLdouble droite,  
        GLdouble bas, GLdouble haut,  
        GLdouble devant, GLdouble fond)
```



➔ Remarque : limiter l'écart fond-devant pour plus de précision

Exemple de définition d'une perspective avec :

- focale : 90°
- proportion de la fenêtre : 16/9ème

```
/* définition de la projection */  
glMatrixMode(GL_PROJECTION) ;  
    glLoadIdentity() ;  
    gluPerspective(90, 16./9., 5, 20) ;  
glMatrixMode(GL_MODELVIEW) ;  
}
```

Remarques :

- la projection doit être définie au moins une fois avant le tracé de la scène
- Elle peut-être définie une fois pour toute dans le `main` et avant `glutMainLoop`

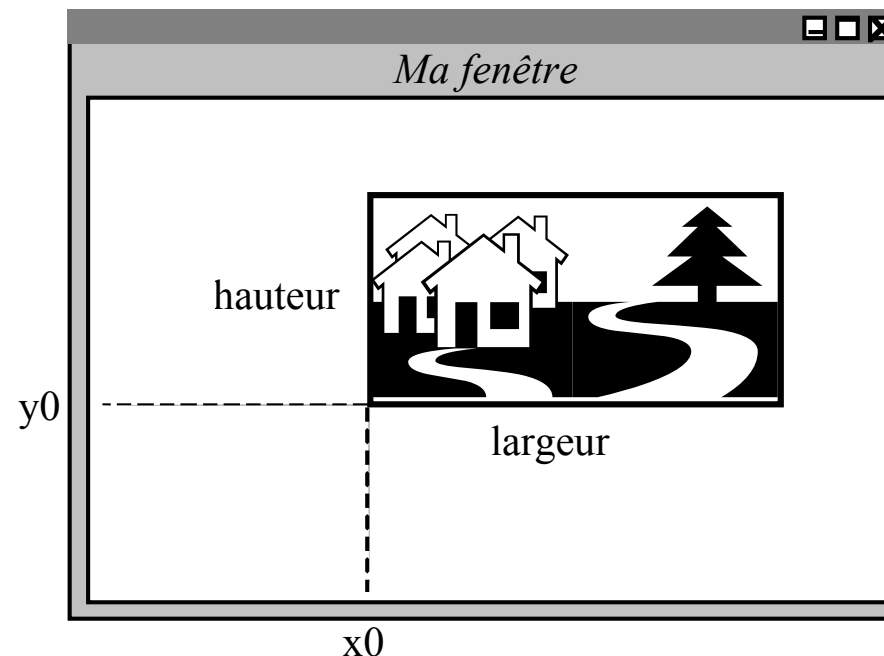
2. Rastérisation

Fenêtre du plan de projection -> cadre de pixels

Sommets 2D -> lignes de pixels

```
glViewport(GLint x0, GLint y0, GLint large, GLint haut)
```

- unité : le pixel
- déformation si le ratio largeur/hauteur n'est pas conservé.



pour ne pas déformer l'image suivant la forme de la fenêtre

```
void monFenetrage(int large, int haut)
{ /* taille du cadre d'affichage */
    glViewport(0, 0, large, haut) ;
    /* définir la projection */
    glMatrixMode(GL_PROJECTION) ;
    glLoadIdentity() ;
    gluPerspective(90, (float)large/haut, 5, 20) ;
    glMatrixMode(GL_MODELVIEW) ;
}

int main (int argc, char **argv)
{ ...
    glutReshapeFunc (monFenetrage) ;
    ...
    glutMainLoop () ;
    return 0 ;
}
```

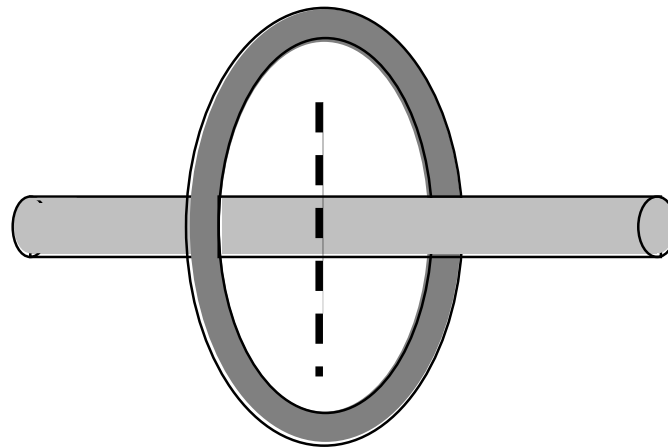
3. Z-buffer

Un objet peu partiellement ou totalement en cacher un autre :

- dépend du point de vue qui peut varier,
- indépendant de l'ordre de construction.
- en pratique, le dernier dessiné « écrase » l'existant.

L'algorithme du peintre : une solution qui n'est plus guère utilisée.

- trier les objets dans l'ordre décroissant de la distance au point de vue
- les dessiner dans cet ordre
- fragmenter éventuellement les objets (relation d'ordre totale)

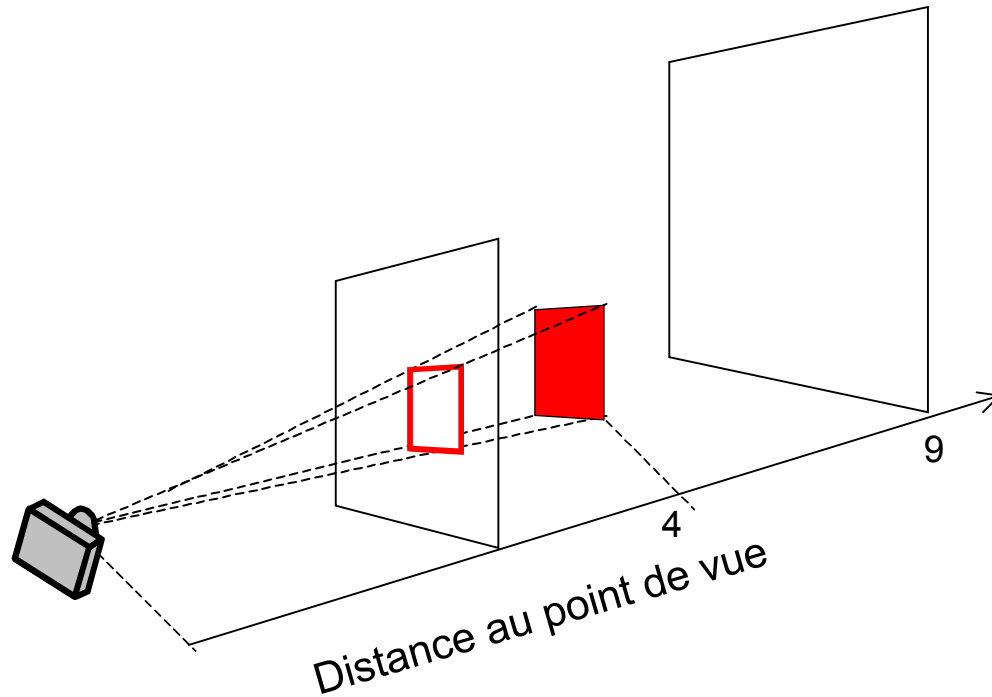


Le Z-buffer : tampon des distances au point de vue

➔ s'affranchir de l'ordre de construction des objets.

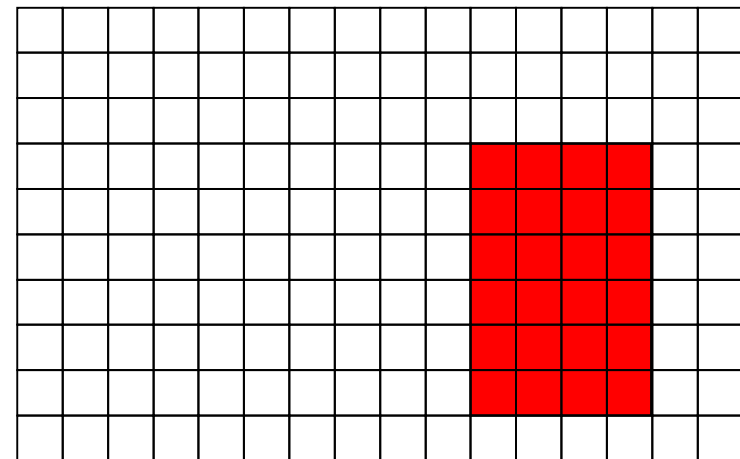
Exemple de tracé d'un premier objet :

- se trouvant à une distance 4 du point de vue
- avec un plan arrière éloigné de 9



9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
9	9	9	9	9	9	9	9	9	9	9	4	4	4	4	9
9	9	9	9	9	9	9	9	9	9	9	4	4	4	4	9
9	9	9	9	9	9	9	9	9	9	9	4	4	4	4	9
9	9	9	9	9	9	9	9	9	9	9	4	4	4	4	9
9	9	9	9	9	9	9	9	9	9	9	4	4	4	4	9
9	9	9	9	9	9	9	9	9	9	9	4	4	4	4	9
9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9

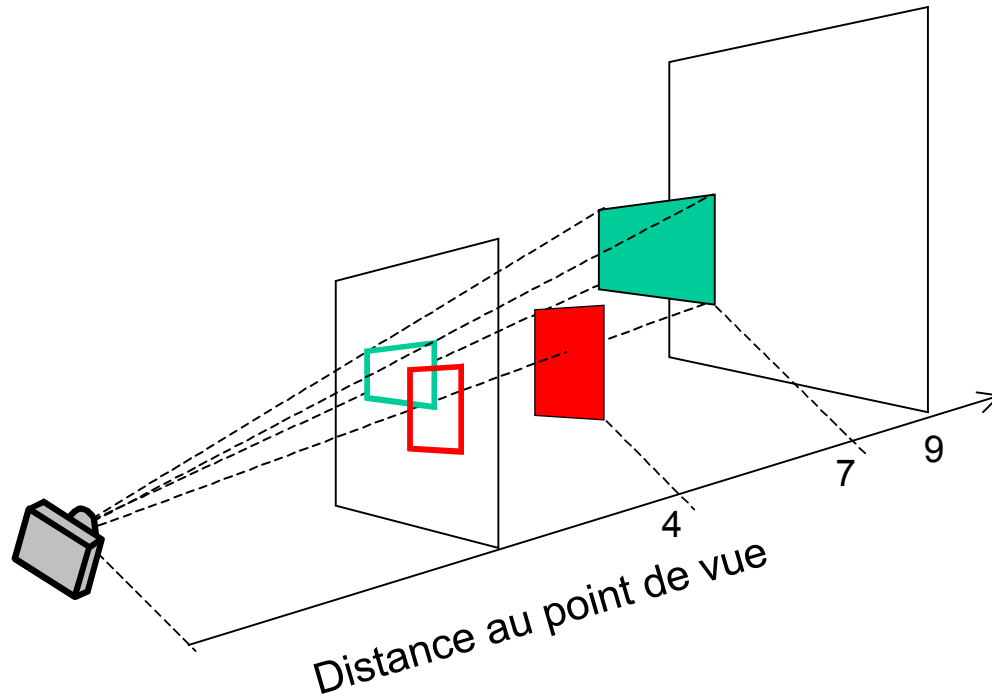
Z-buffer



Buffer image

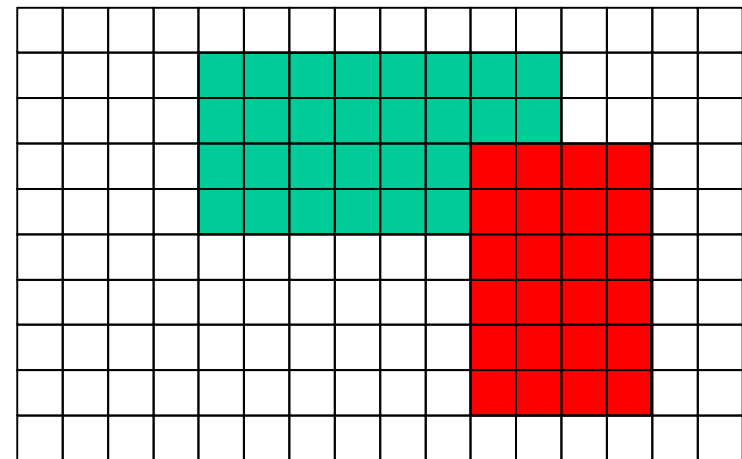
Tracé d'un deuxième objet :

- se trouvant à une distance 7 du point de vue
- et partiellement occulté par le premier objet



9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
9	9	9	9	7	7	7	7	7	7	7	7	9	9	9	9	9	9
9	9	9	9	7	7	7	7	7	7	7	7	9	9	9	9	9	9
9	9	9	9	7	7	7	7	7	7	4	4	4	4	9	9	9	9
9	9	9	9	7	7	7	7	7	7	4	4	4	4	9	9	9	9
9	9	9	9	9	9	9	9	9	9	4	4	4	4	9	9	9	9
9	9	9	9	9	9	9	9	9	9	4	4	4	4	9	9	9	9
9	9	9	9	9	9	9	9	9	9	4	4	4	4	9	9	9	9
9	9	9	9	9	9	9	9	9	9	4	4	4	4	9	9	9	9
9	9	9	9	9	9	9	9	9	9	4	4	4	4	9	9	9	9
9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9

Z-buffer



Buffer image

Algorithme du Z-buffer :

- Z-buffer : dimension en pixel de la fenêtre,
- initialisé avec la plus grande profondeur possible (distance au plan de clipping arrière)

soit une primitive à afficher

pour tous les pixels qui lui correspondent dans le plan image

- évaluer la profondeur (distance au point de vue)
- **si** profondeur < à la valeur stockée dans le Z-buffer
 - le Z-buffer reçoit la profondeur
 - le pixel reçoit ses valeurs chromatiques

fin-si

fin-pour

Mise en œuvre sous OpenGL :

Contexte d'affichage :

1 : déclarer l'utilisation du Z-buffer dans le `main()` à l'initialisation :

```
glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH) ;
```

2 : activer/désactiver le mode Z-buffer avec :

```
glEnable(GL_DEPTH_TEST) ; /* algo. du Z-buffer */
```

```
glDisable(GL_DEPTH_TEST) ; /* algo. du peintre */
```

Dessin d'une scène :

3 : Débuter une scène par une réinitialisation du Z-buffer :

```
glClear (GL_DEPTH_BUFFER_BIT) ;
```

ou encore :

```
glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT) ;
```

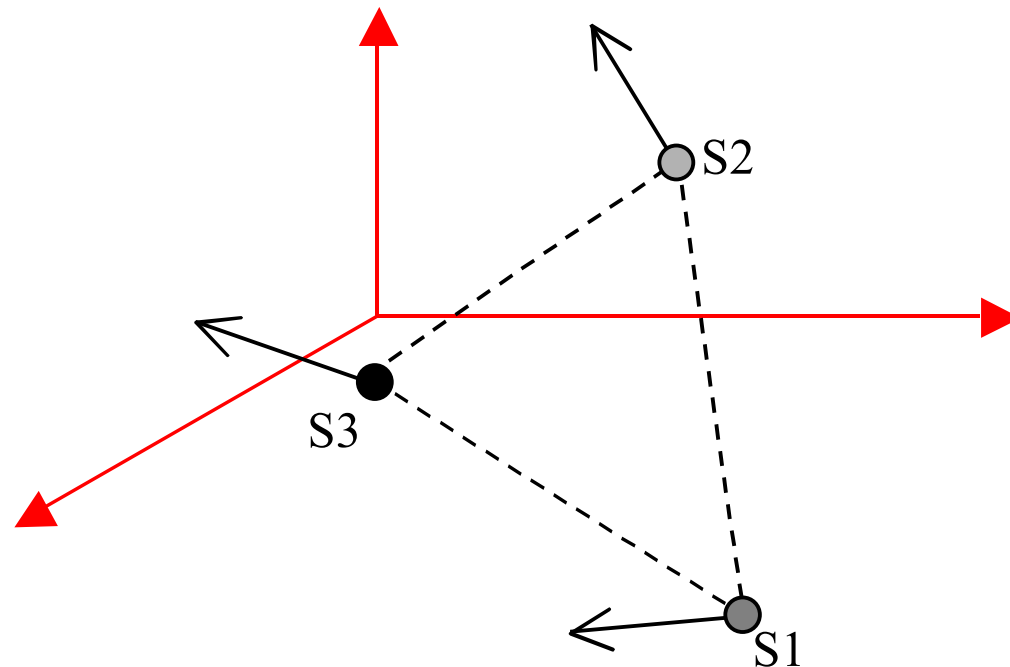
4. La « face cachée » d'OpenGL

Rappel :

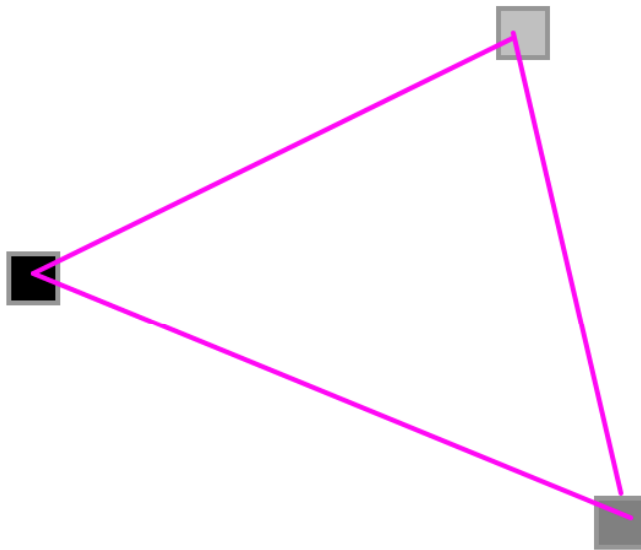
- la scène 3D n'est pas mémorisée
- l'image 2D est construite pendant la description de la scène 3D
- le contexte de tracé doit être défini avant l'énumération des sommets
- les primitives sont tracées « à la volée »

Exemple du tracé d'une facette triangulaire :

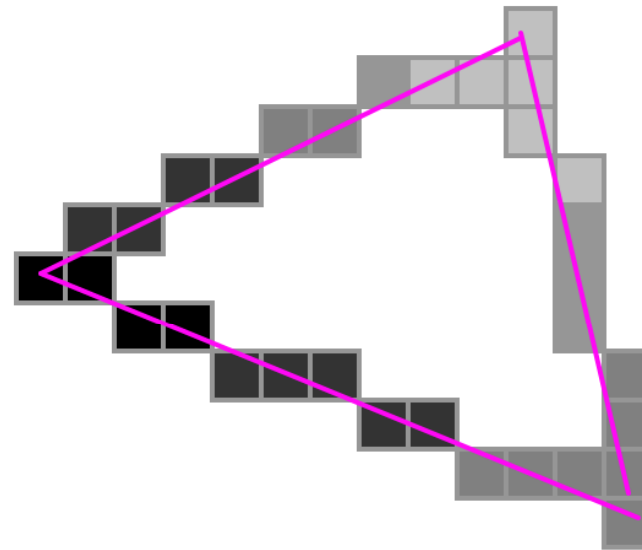
1. énumération des coordonnées des trois sommets (`glVertex`)
2. produit des sommets par la matrice de modélisation (`GL_MODELVIEW`)
3. évaluation de la distance des sommets au point de vue (profondeur)
4. évaluation du rendu de chaque sommet en fonction de la position, des normales, des lumières et de l'observateur



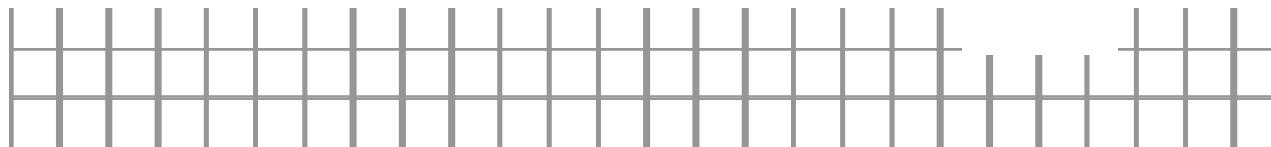
5. projection des sommets sur le plan de projection
6. pixelisation
7. calcul des pixels constituant les 3 arêtes (Bresenham)
8. extrapolation de la couleur et profondeur de chacun de ces pixels



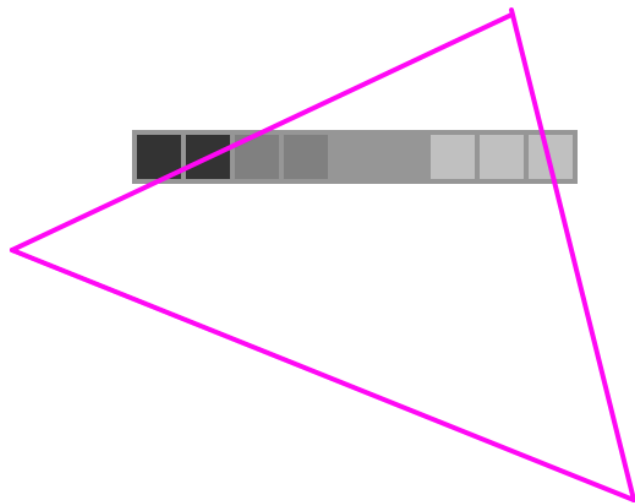
Etapes 5 et 6



Etapes 7 et 8



9. **pour** chaque segment horizontal reliant deux arêtes
- pour** chaque pixel de ce segment
- extrapoler sa profondeur
 - **si** sa profondeur $<$ son équivalent dans le z-buffer
 - affecter sa profondeur dans le z-buffer
 - extrapoler sa couleur et l'afficher



Remplissage de ligne

