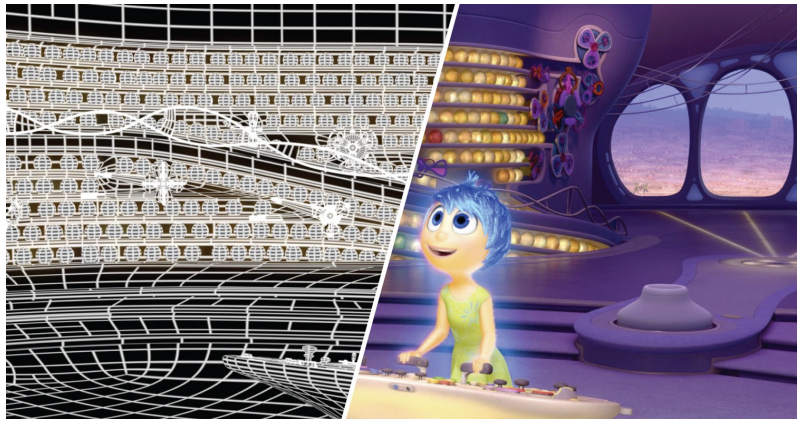
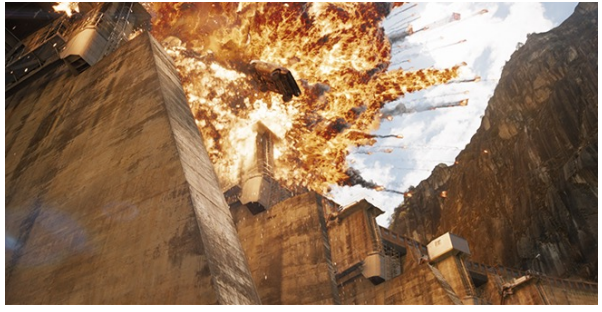


Rendu

Florence Zara (semestre automne)

LIRIS, équipe ORIGAMI

Université Lyon 1



Pipeline de génération en synthèse d'images

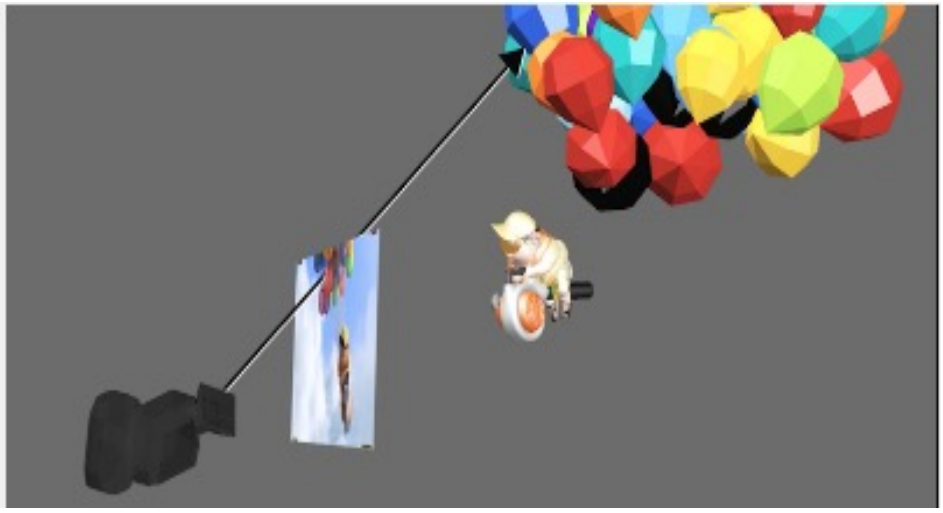
Etapes pour générer une image 2D :

- Modélisation : géométrie donnant la forme des objets
- Simulation : animation donnant du mouvement
- Rendu : affichage en fonction de la lumière, de la matière, de la caméra
- Post-traitement : correction des couleurs

Objectif du cours : comprendre comment se fait l'étape de rendu créant l'image 2D à partir de la scène 3D (comprenant les objets modélisés) et en fonction de la position de la caméra

Création de l'image 2D en synthèse d'images

Passage de la scène 3D vers l'image 2D (vue à l'écran) en fonction du point de vue de l'œil



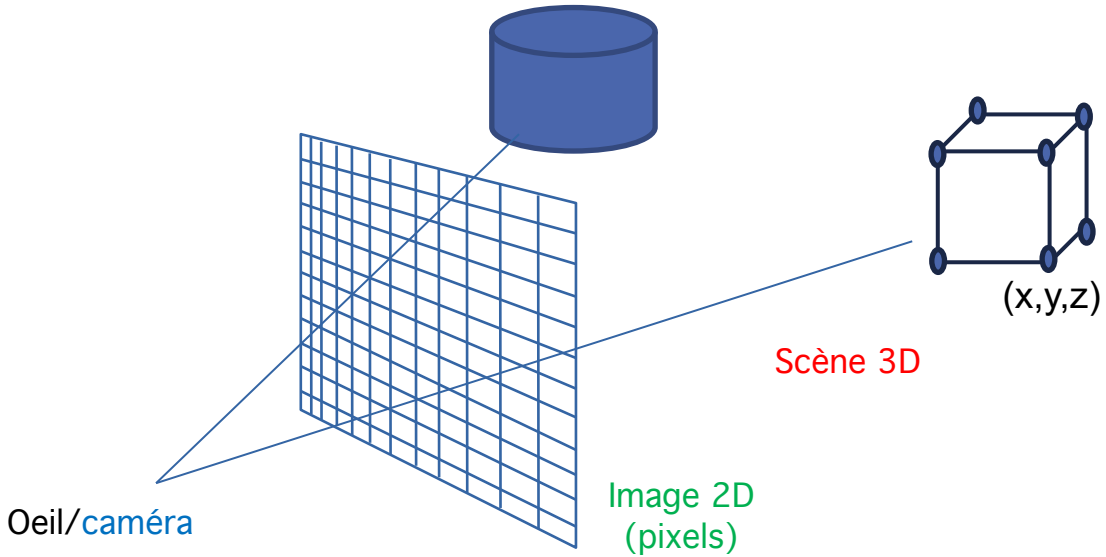
Scène 3D comportant les objets qui ont une représentation mathématique

Œil /caméra

Image 2D
(ensemble de pixels)

Création de l'image 2D en synthèse d'images

Passage de la scène 3D vers l'image 2D (vue à l'écran) en fonction du point de vue de l'œil



Objet défini par des sommets ayant coordonnées dans l'espace
=
modèle géométrique de l'objet

Pipeline graphique

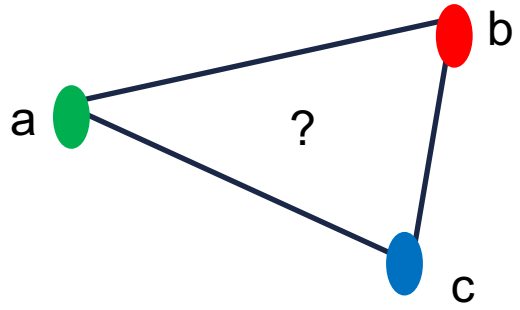
Pipeline graphique = ensemble des traitements permettant de dessiner des objets

Pour dessiner un point, il faut :

- Définir ses coordonnées x,y,z
- Convertir ses coordonnées en position (i,j) dans l'image
  - Besoin de connaître la dimension de l'image : largeur, hauteur
- Choisir une couleur pour le pixel (i,j)

- et ensuite il faut stocker le résultat qui est l'image

Pipeline graphique

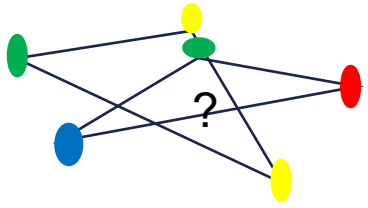


Pour dessiner un triangle :

- Besoin des informations sur les 3 sommets du triangle
- Choix à faire si les sommets ont des couleurs différentes
  - Interpolation des couleurs ?

Pour dessiner plusieurs triangles :

- Pipeline des informations des coordonnées et couleurs des sommets de chaque triangle
- Choix à faire si plusieurs triangles se dessinent sur le même pixel de l'image
  - Dessiner les triangles dans l'ordre
    chaque pixel conserve couleur du dernier triangle dessiné
  - Utiliser la profondeur en 3D pour garder couleur du triangle le plus proche ou plus loin



Pipeline graphique – Utilisation de l'API OpenGL

API 3D = permet de configurer et paramétrer les différentes étapes du pipeline graphique

OpenGL = API 3D permettant d'utiliser la carte graphique pour dessiner des objets

API OpenGL permet :

- de **fournir les informations (coordonnées, couleurs) sur les sommets** des triangles à dessiner
- de fournir une fonction **transformant les coordonnées des sommets en position dans l'image**
- de choisir **comment remplir l'intérieur des triangles**
  - couleur des sommets, interpolation, etc.
- de choisir **quel triangle garder pour chaque pixel de l'image**

OpenGL – Quelques infos

- OpenGL 1.x est une API 3D (Application Programmer ’s Interface)
  - Début en 1992 !
  - Initialement basé sur IRIS GL de Silicon Graphics
  - Langage de bas niveau pour avoir l’indépendance vis-à-vis
    - système de fenêtrage
    - plate-forme matérielle
    - système d’exploitation

- Architecture de type client-serveur :
  - le client émet les commandes
  - le serveur les exécute

OpenGL : librairies associées

Comme OpenGL est indépendant du système de fenêtrage, des librairies additionnelles sont utilisées pour intégrer OpenGL dans de tels systèmes :

- GLX pour Xwindow
- WGL pour Windows
- GLUT indépendant du système de fenêtrage
- SDL
- Qt
- MFC (windows)

OpenGL : machine à états

- OpenGL possède un état courant pour tout (repère courant, projection courante, couleur courante, matrice courante, …)

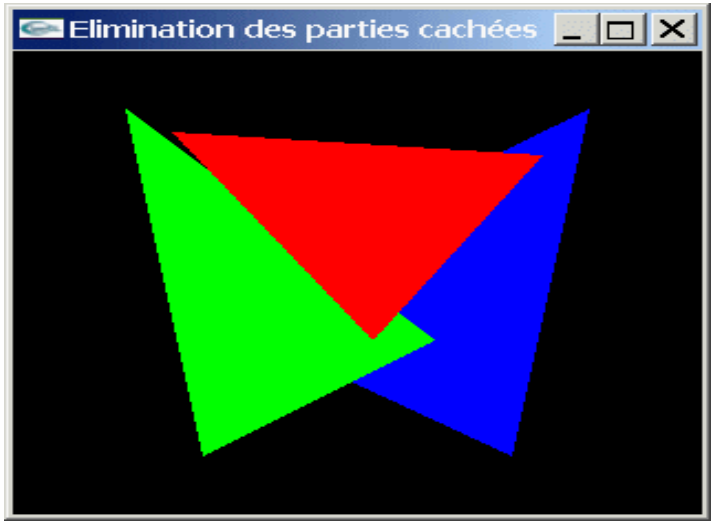
- On peut activer / désactiver les états avec
  - glEnable()
  - glDisable()

- Tous les états ont des valeurs par défaut (pas besoin de tous les définir en début de programme !)

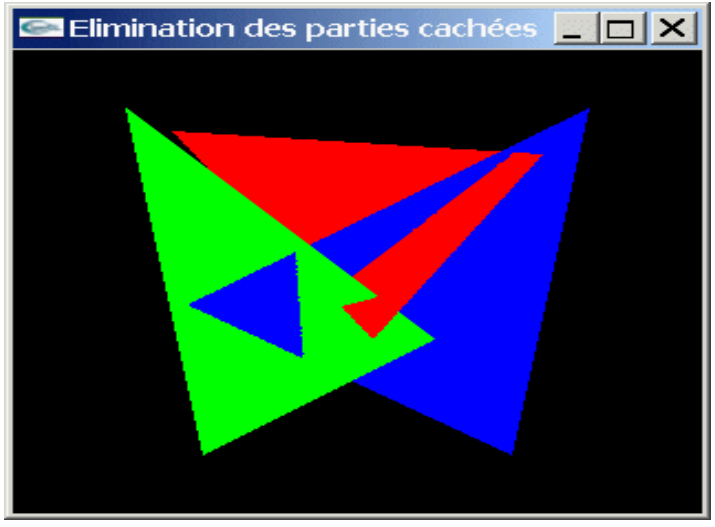
Exemple avec le Z-buffer

Pour activer le test des profondeurs utilisant le Z-buffer : `glEnable(GL_DEPTH_TEST) ;`

Pour la gestion des triangles les uns devant les autres



non activé



activé

On verra l'algorithme du Z-buffer à la fin du cours

OpenGL - Pipeline graphique

Pipeline Graphique OpenGL découpé en 2 parties :

- **Traitement de la géométrie, sommets des triangles :**
  - Passage des coordonnées des sommets (x,y,z) selon un repère quelconque en positions (i,j) dans l'image

- **Traitement des pixels :**
  - Quels pixels dans l'image impactés par les triangles ? (utilise les positions calculées précédemment)

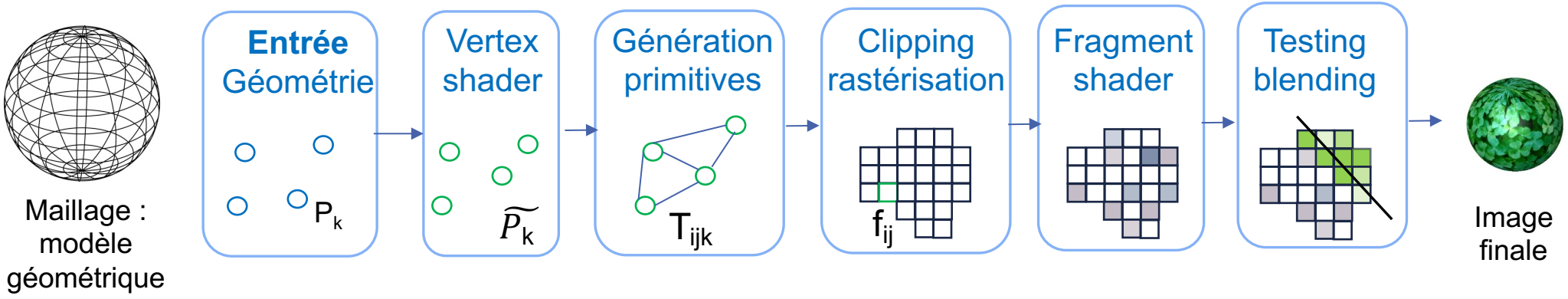
  - Quelles couleurs pour ces pixels ?

Pipeline graphique de rendu par rastérisation

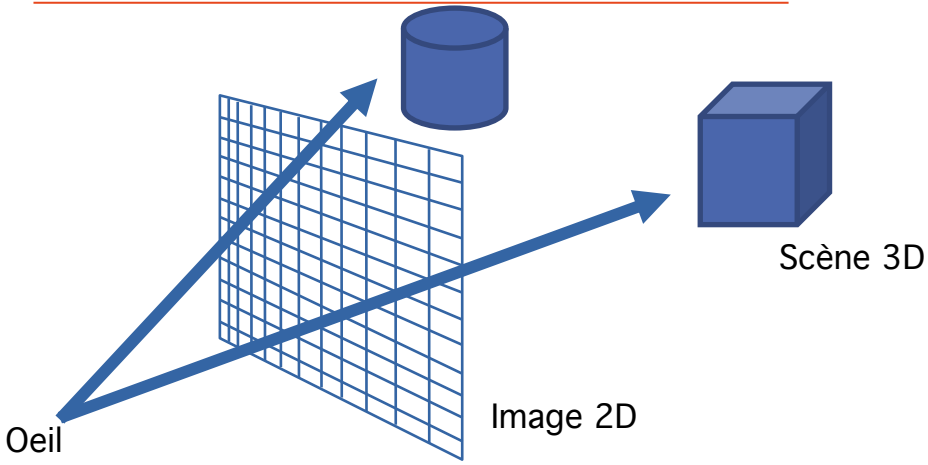
Rastérisation = processus pour convertir modèle continu en images matricielles (*rasters*) constituées de pixels d'une certaine couleur

Pipeline graphique de rendu par rastérisation

Rastérisation = processus pour convertir modèle continu en images matricielles (*rasters*) constituées de pixels d'une certaine couleur

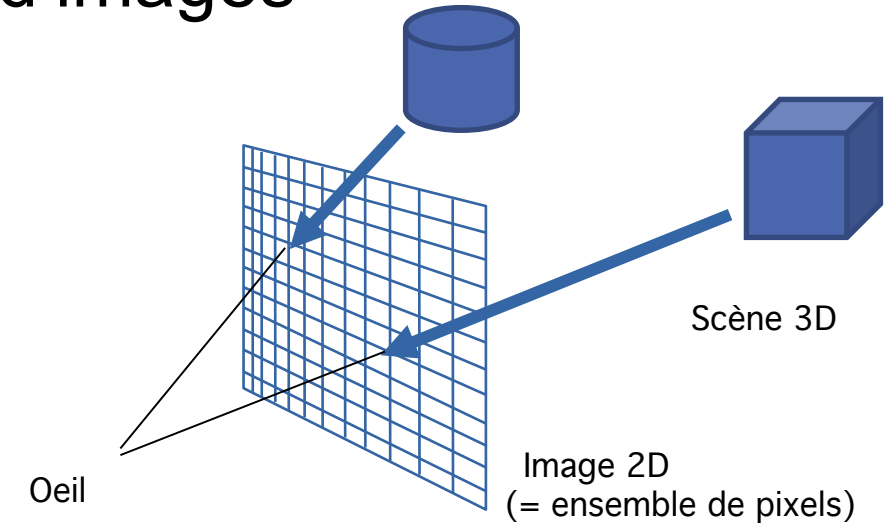


2 approches duales en synthèse d'images



Des rayons sont lancés
depuis l'œil vers la scène
en passant par un pixel

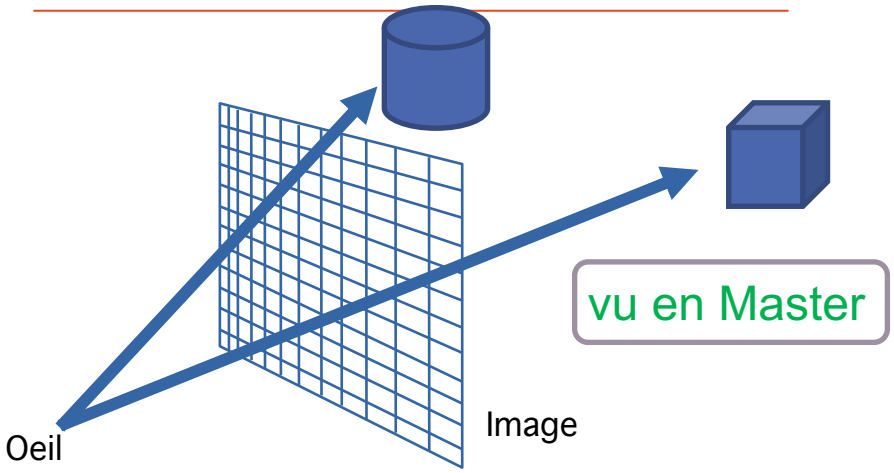
Ray-tracing
donne une image réaliste
mais couteux / lent



Les objets 3D sont projetés sur
l'écran 2D dans la direction de l'œil

Rendu projectif
effectué sur GPU
en temps réel

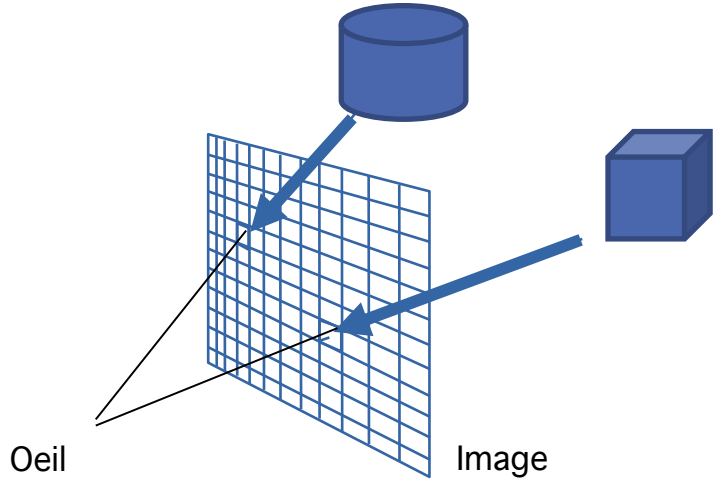
2e approche : rendu projectif



vu en Master

Des rayons sont lancés
depuis l'œil vers la scène
en passant par un pixel

Ray-tracing
donne une image réaliste
mais couteux / lent

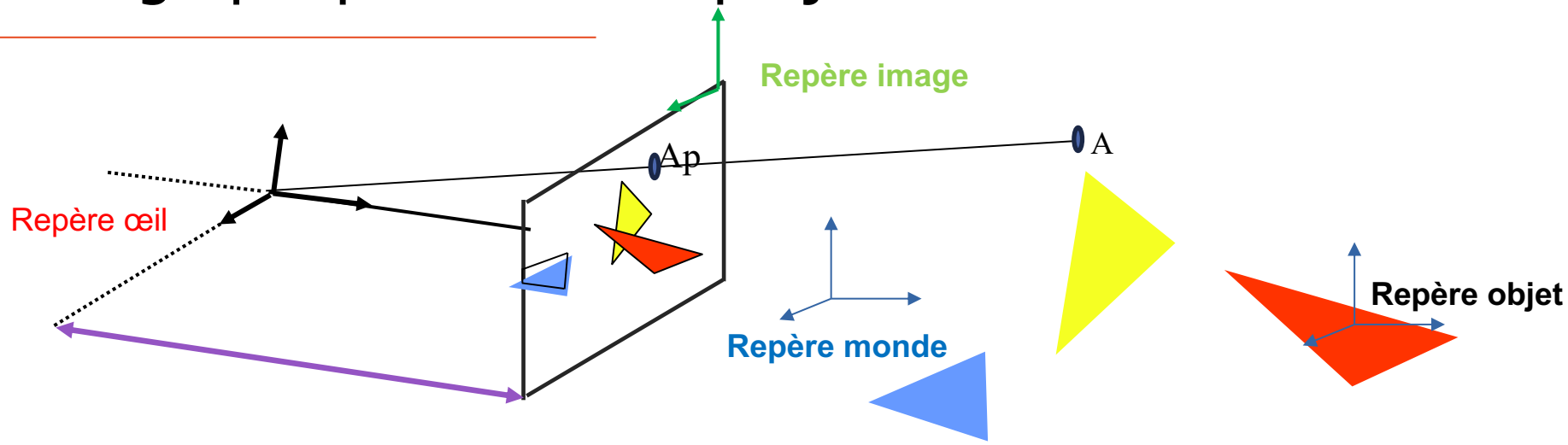


(= ensemble de pixels)

Les objets sont projetés sur
l'écran dans la direction de l'œil

Rendu projectif
effectué sur GPU en temps réel

Pipeline graphique du rendu projectif



Les sommets existent dans plusieurs repères :

Repère de création de l'objet
Espace du modèle 3D lui-même

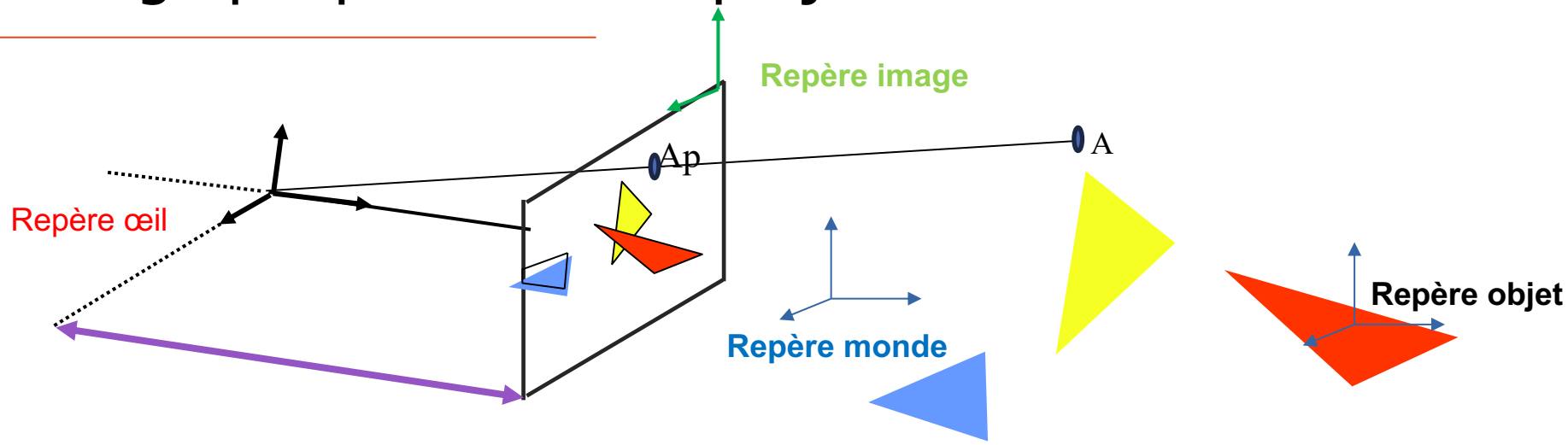
Repère monde / scène
Positionnement des objets, des lumières et de la caméra

Repère de la caméra / œil
Espace du point de vue de l'observateur

Repère image
Contient les points projetés (passage de la 3D au 2D de l'image)

Espace écran
Image finale en pixels

Pipeline graphique du rendu projectif

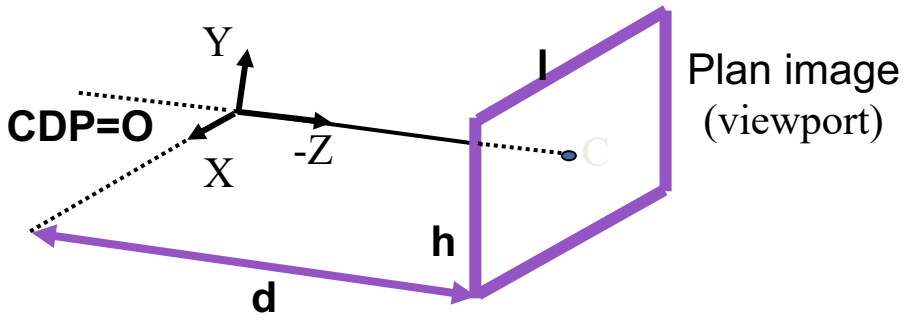


Détails du pipeline graphique pour un rendu projectif :

1. *Clipping* des polygones en 3D suivant la pyramide de vue
2. Projection des points sur le plan image
3. Remplissage des triangles dans l'image
   a. Suppression des parties cachées : Z-Buffer
   b. Calcul de la couleur : illumination

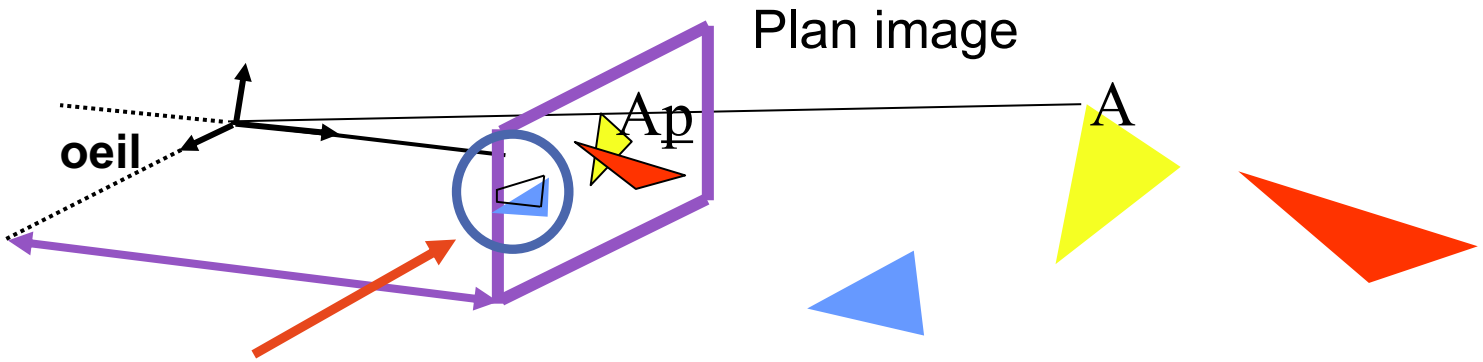
Définissons la caméra / repère de l'œil

- Une caméra dans l'espace
  - CDP = Origine O
  - La caméra regarde vers –Z
  - Le haut de la caméra est Y



Première étape du pipeline - Clipping selon la pyramide de vue

Trouver la partie exacte d'un polygone se trouvant dans la pyramide de vue



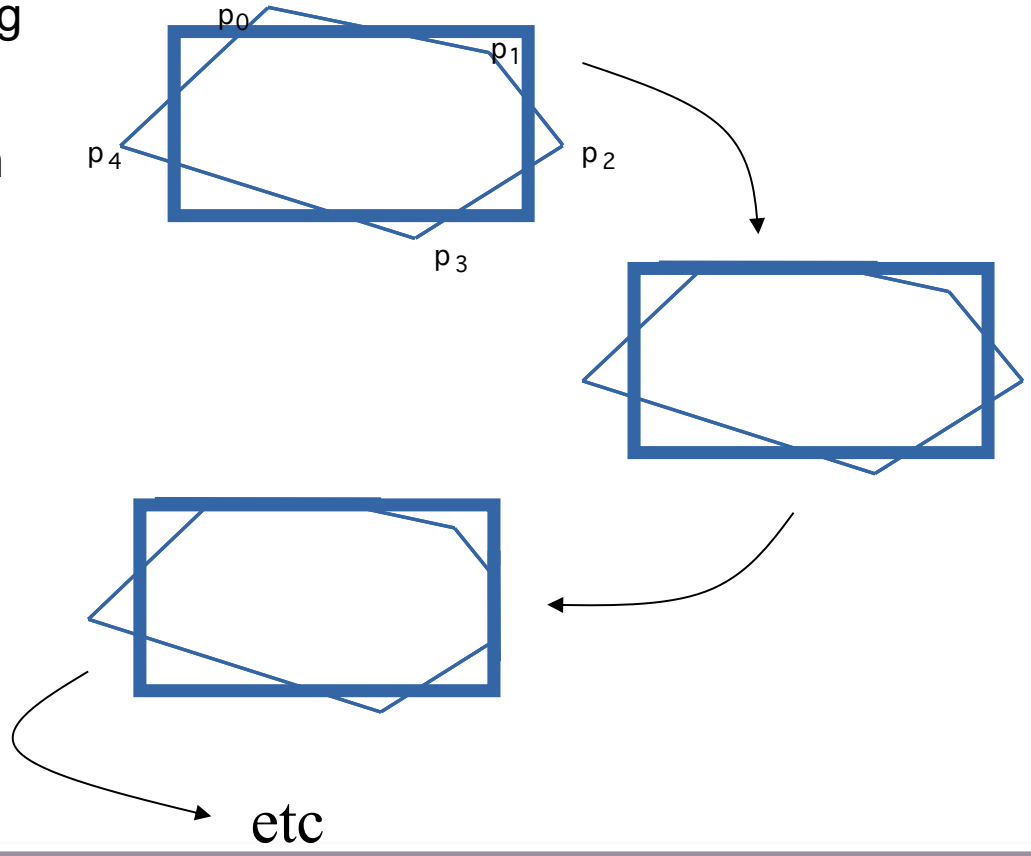
- Le clipping d'un polygone est un polygone
- Le clipping peut se faire
  - en 2D - après la projection
  - en 3D – avant la projection

Algorithme de Sutherland-Hodgman : 1974 !

Itération sur les arêtes de la région de clipping

Résultat peut être NULL si le polygone est en dehors

Généralisation possible à des régions non rectangulaires

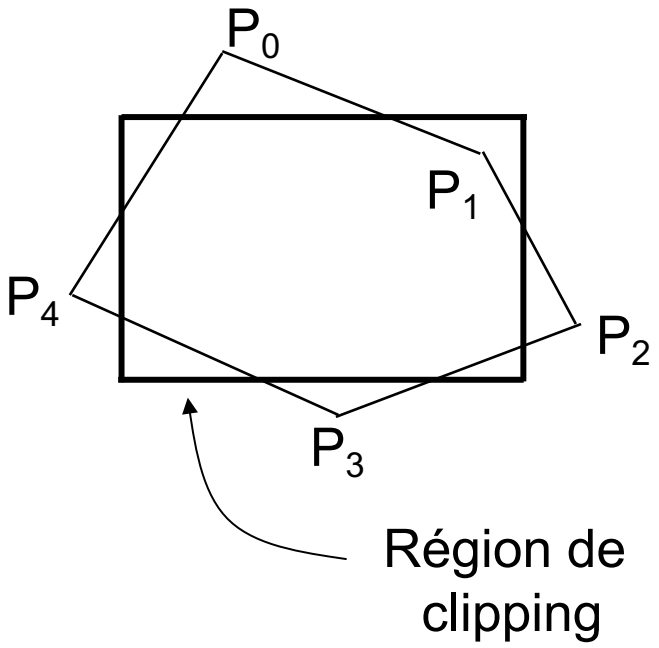


Sutherland-Hodgman - Clipping suivant une région

Pour construire le nouveau polygone :

- Itération sur les arêtes c
  (par exemple de P0 à P4 pour le polygone ci-contre)
- Construction d'une nouvelle séquence de points
- Initialisation avec une séquence vide

4 cas à considérer

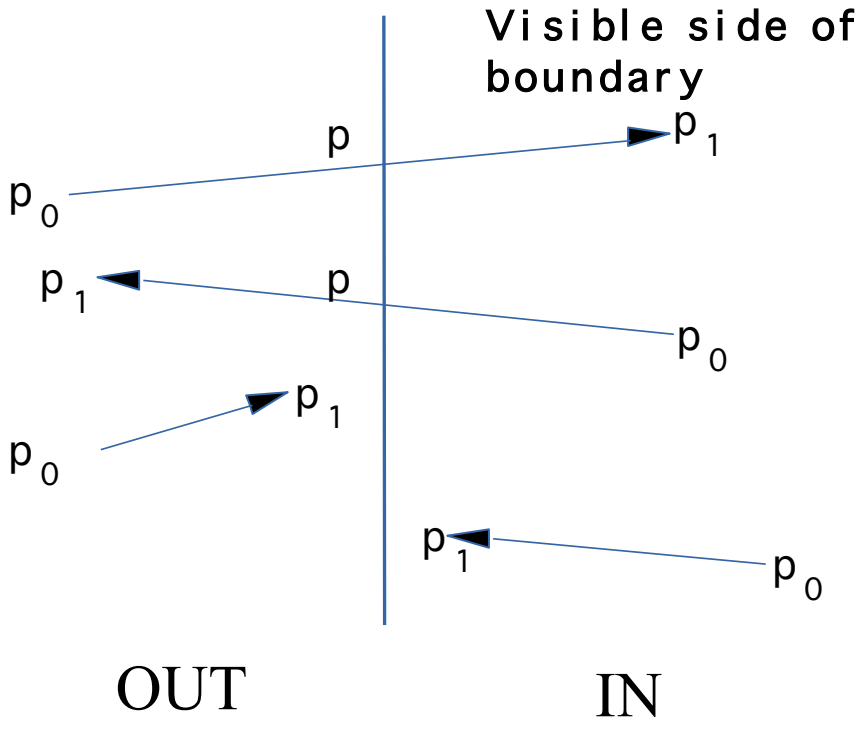


Clipping d'une arête

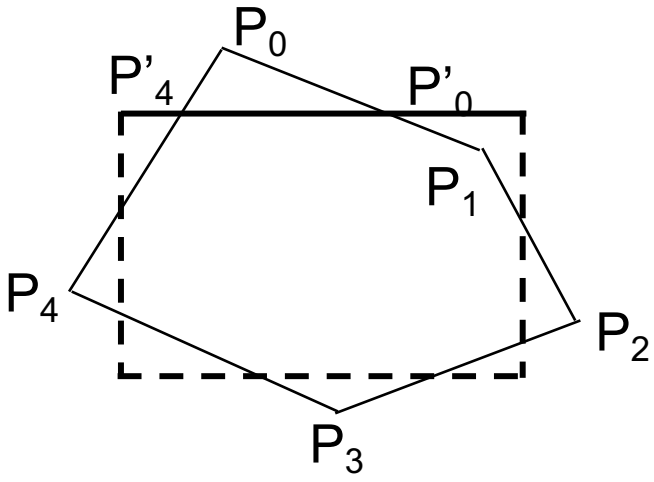
Soit arête P_0,P_1 - 4 cas possibles :

- Entrée dans la région de *clipping*
  - ajouter *P* et P_1
- Sortie de la région de *clipping*
  - ajouter *P*
- Entièrement en dehors
  - ne rien faire
- Entièrement à l'intérieur
  - ajouter P1

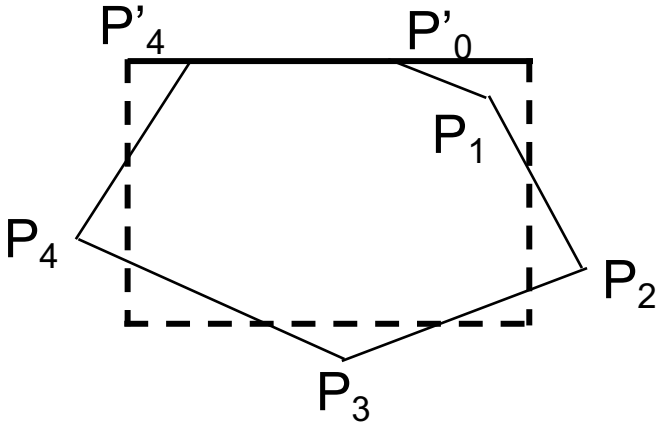
avec P le point d'intersection



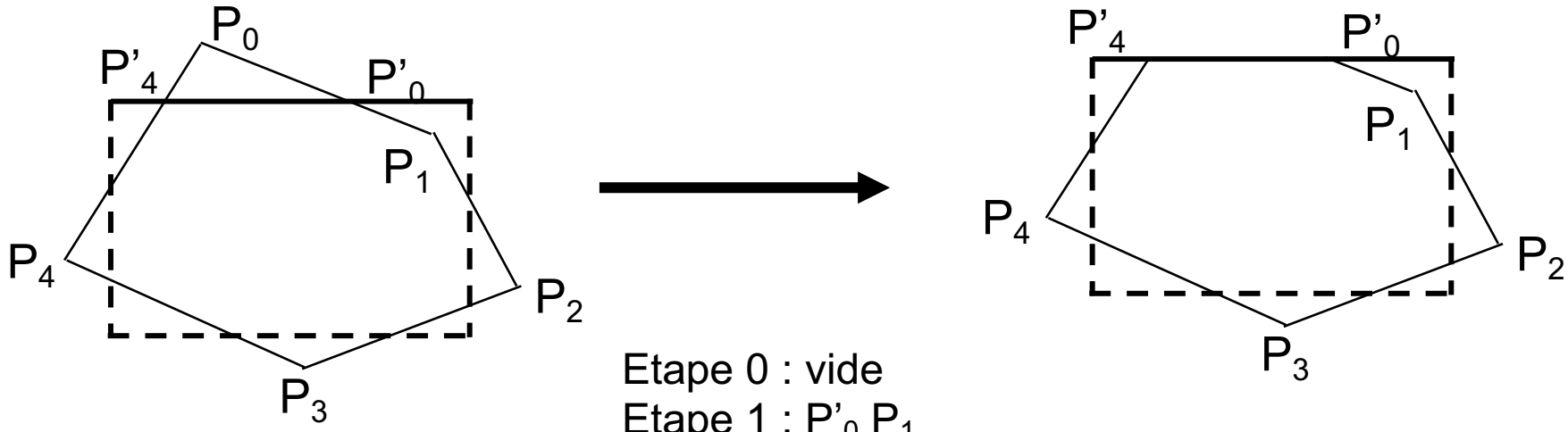
Algorithme de Sutherland-Hodgman - Exemple



Etape 0 : vide
Etape 1 : ?
…

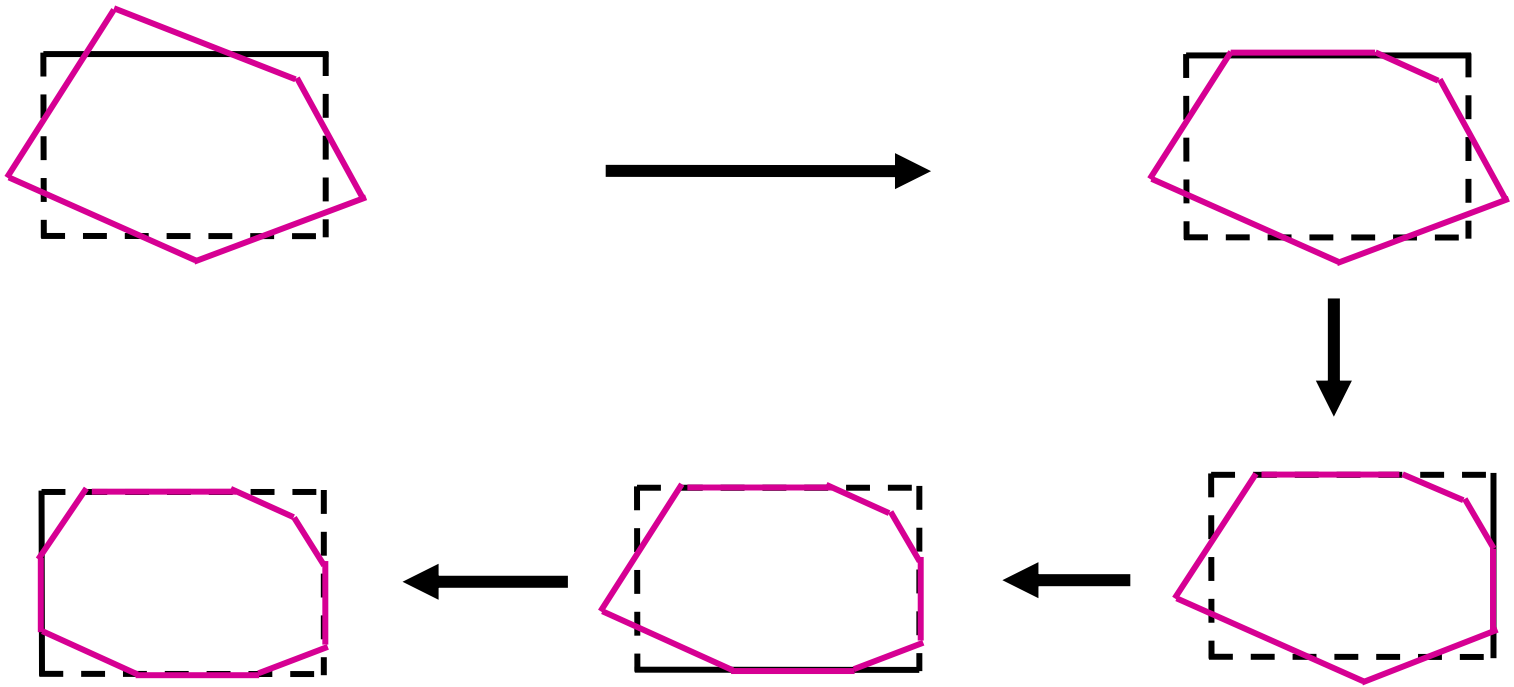


Algorithme de Sutherland-Hodgman - Exemple



Etape 0 : vide
Etape 1 : $P'_0\ P_1$
Etape 2 : $P'_0\ P_1\ P_2$
Etape 3 : $P'_0\ P_1\ P_2\ P_3$
Etape 3 : $P'_0\ P_1\ P_2\ P_3\ P_4$
Etape 4 : $P'_0\ P_1\ P_2\ P_3\ P_4\ P'_4$

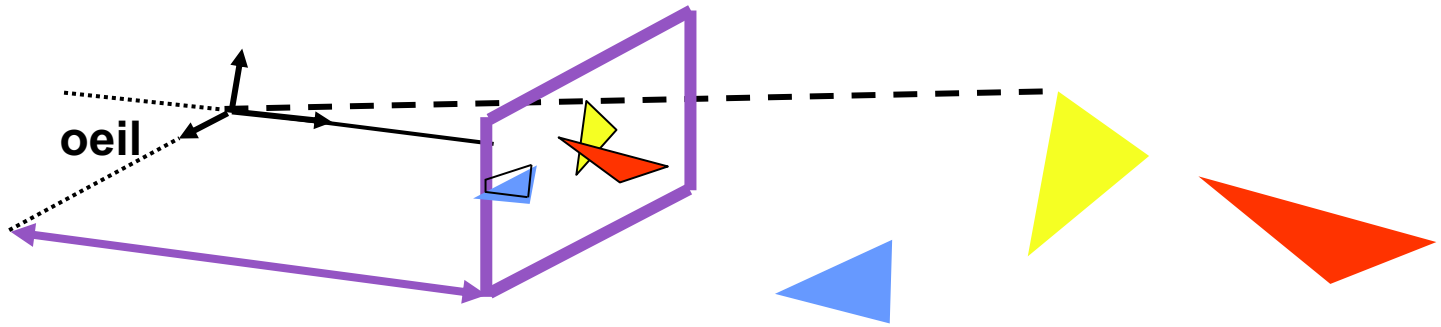
Algorithme de Sutherland-Hodgman - Exemple



Pipeline graphique du rendu projectif

Détails du pipeline graphique pour un rendu projectif :

- Clipping
- Projection
- Remplissage des polygônes
  - Suppression des parties cachées
  - Illumination

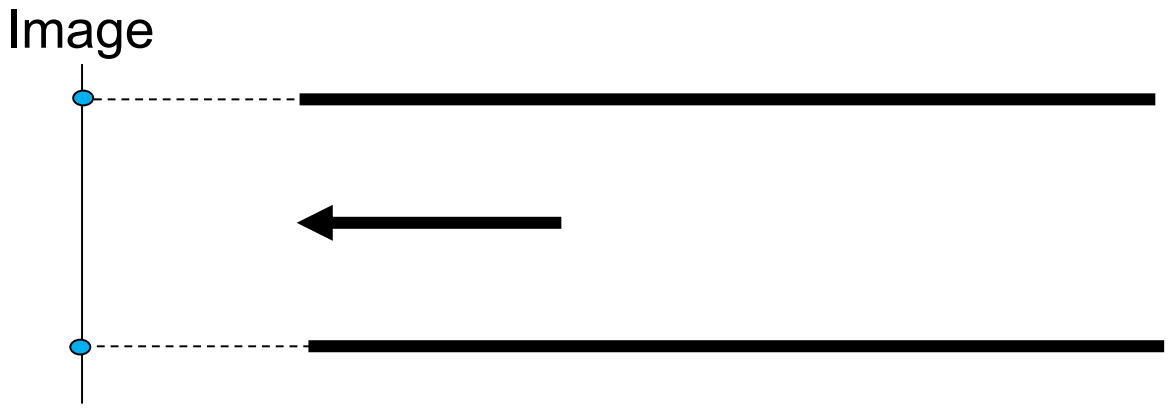


2e étape = projection

Projection - Passage de la scène 3D à l'image 2D

Cas de la projection parallèle

- Projection parallèle sur le plan z = 0
- Direction De Projection (DDP) est (0,0,1)
- P = (x,y,z) → P' = (x, y, 0)

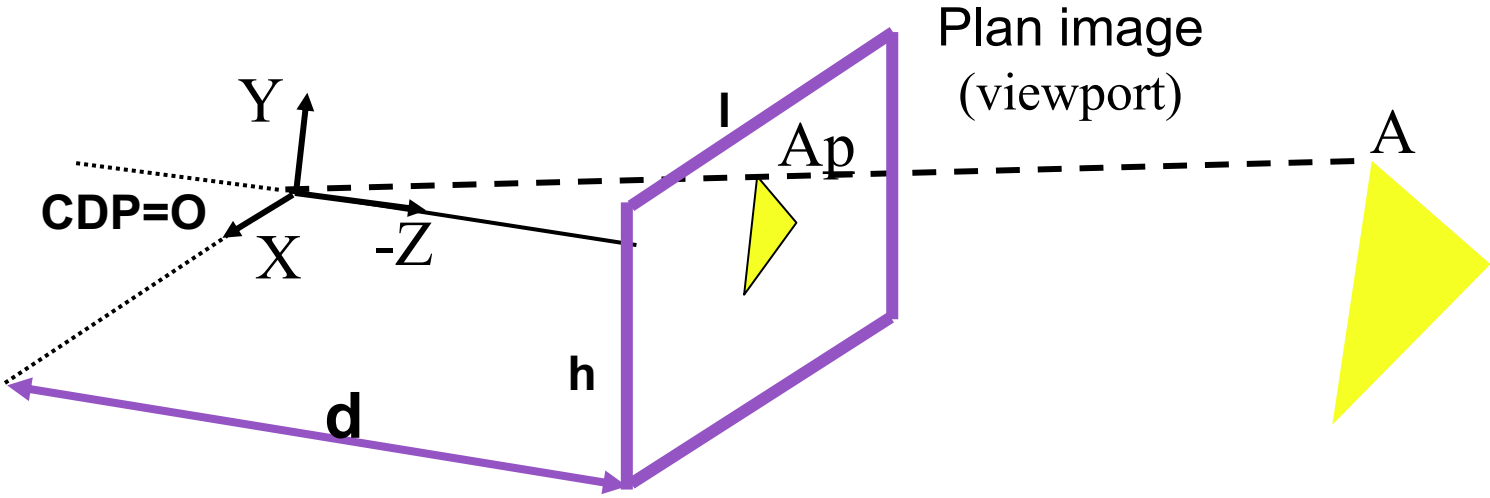


Projection - Passage de la scène 3D à l'image 2D

Projection parallèle = manque de réalisme

Projection - Passage de la scène 3D à l'image 2D

Cas de la projection perspective

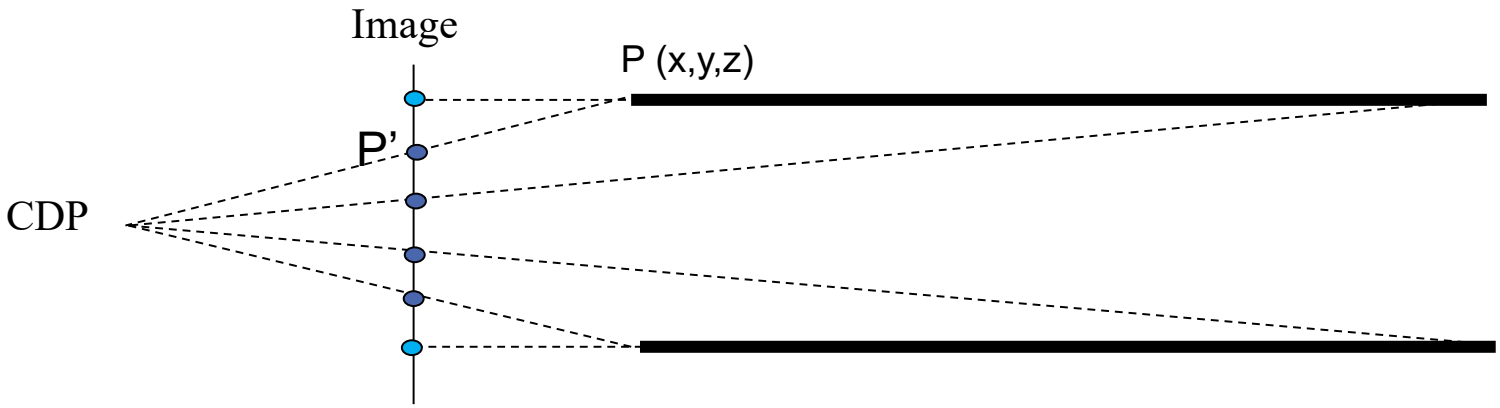


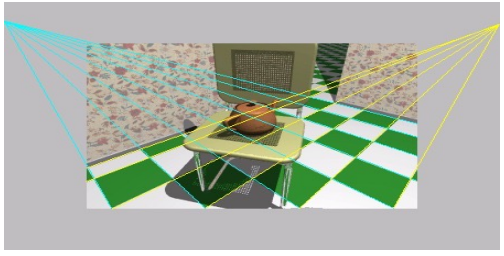
Projette les points du triangle sur le plan image par rapport au CDP

Projection perspective

Projection vers le centre de projection CDP

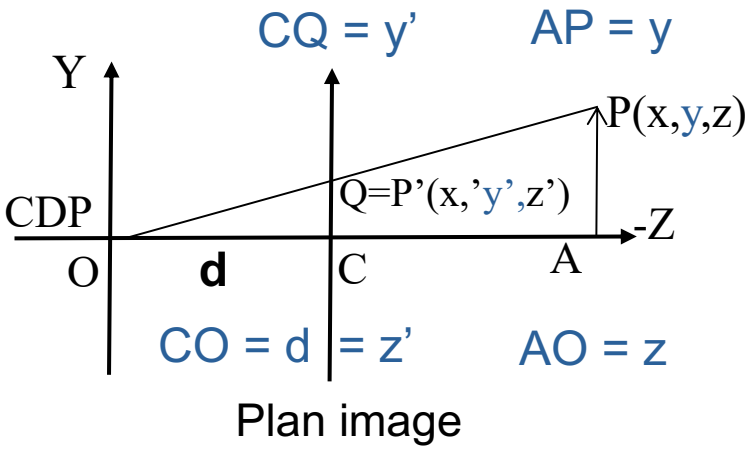
P = (x,y,z) → Trouver les coordonnées du point projeté P'





Projection perspective

Dans le plan (Y, -Z) :



d = distance focale

z' = d

$$\frac{CQ}{AP} = \frac{CO}{AO} \Leftrightarrow CQ = y' = \frac{y.d}{z}$$

Par analogie, dans le plan (X, -Z) :

$$x' = \frac{x.d}{z}$$

Si $d = 1 \Rightarrow y' = \frac{y}{z}$ et $x' = \frac{x}{z}$

Au final, on a :

- x' = x d/z
- y' = y d/z
- z' = d

Projection perspective

On souhaiterait définir la matrice de transformation correspondante :

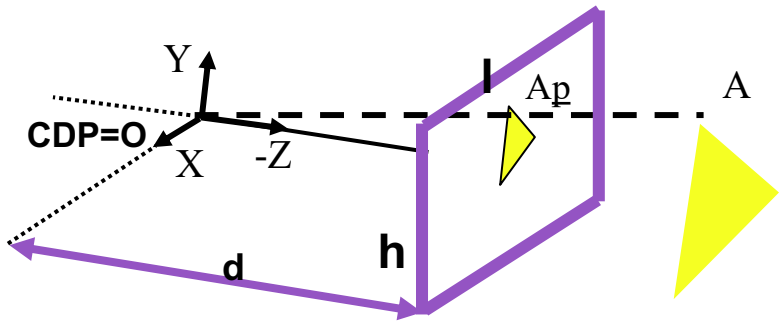
Entrée : Point défini dans le repère de la caméra

Résultat : Coordonnées du point dans le repère de l'image

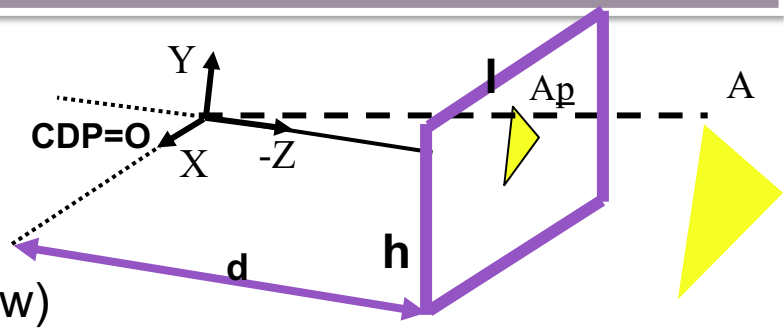
<u>On se replace en coordonnées homogénes :</u>

Entrée : un point dans l'espace de la caméra : $(x, y, z, 1)$

Résultat : un point dans l'espace Image : (x', y', z', 1) = (wx', wy', wz', w)



Matrice de projection : $M_{I \leftarrow C}$



Entrée : un point dans l'espace de la caméra : $(x, y, z, 1)$

Résultat : un point dans l'espace Image : (wx', wy', wz', w)

On a :

$$\begin{cases} x' = x\,d/z = x/w \\ y' = y\,d/z = y/w \\ z' = d \quad = z/w \\ w = z/d \end{cases} \Rightarrow \begin{cases} wx' = x \\ wy' = y \\ wz' = z \\ w = z/d \end{cases}$$

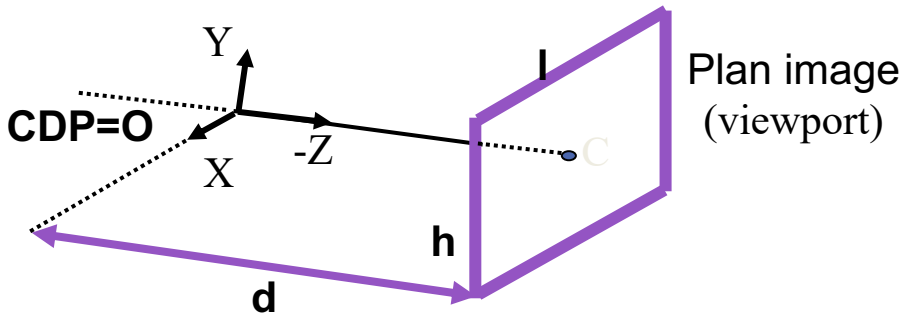
$$M_{I \leftarrow C} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix}$$

Matrice de transformation
d'un point (x y z 1) de l'espace caméra
vers le point (wx', wy', wz', w) de l'espace image

Repère caméra ≠ repère monde

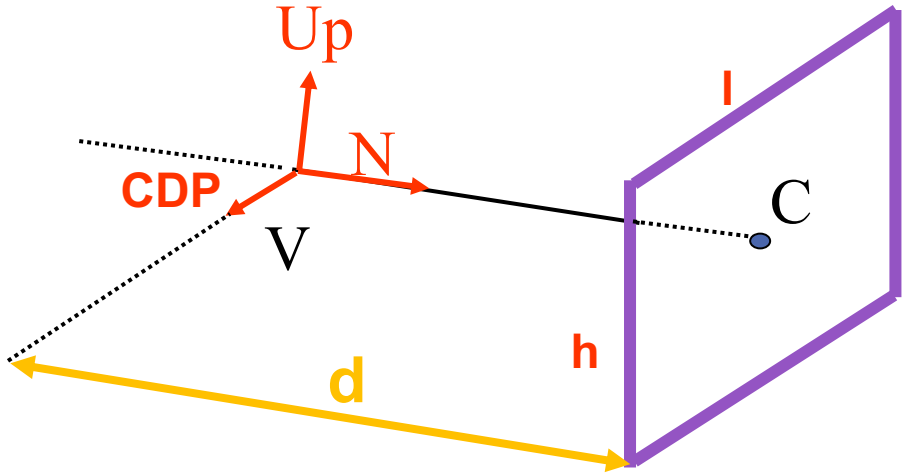
Jusqu'à présent on avait :

- CDP = Origine O
- La caméra regarde vers –Z
- Le haut de la caméra est Y

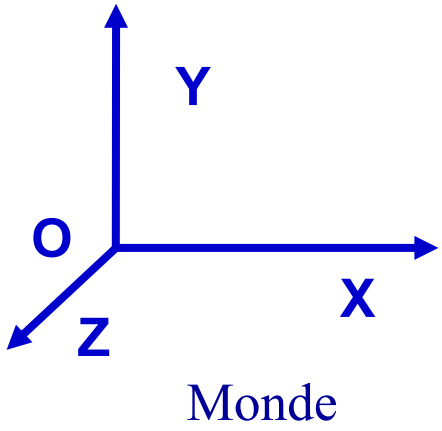


On aimerait pouvoir placer la caméra n'importe où !
On va définir un repère pour la caméra (différent du repère du monde)

Caméra plus générale



Plan image
(viewport)

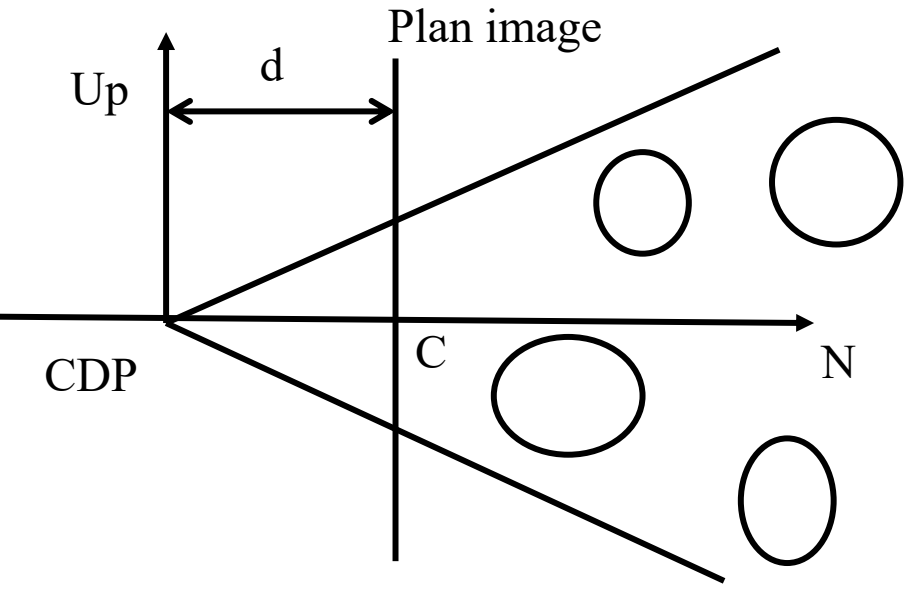


CDP = Centre De Projection (≠ O)
N = Direction de vue (N ⊥ au plan image), |N|=1
Up = Direction du haut de la camera (Up), |U|=1
V = le 3e vecteur pour que UpVN forment un repère, |V|=1

d = distance focale
(distance entre CDP et le plan image)
l = largeur de l'image dans l'espace caméra
h = hauteur de l'image dans l'espace caméra
m x n = taille en pixels de l'image

Caméra plus générale

C = centre de l'image = CDP + d * **N**

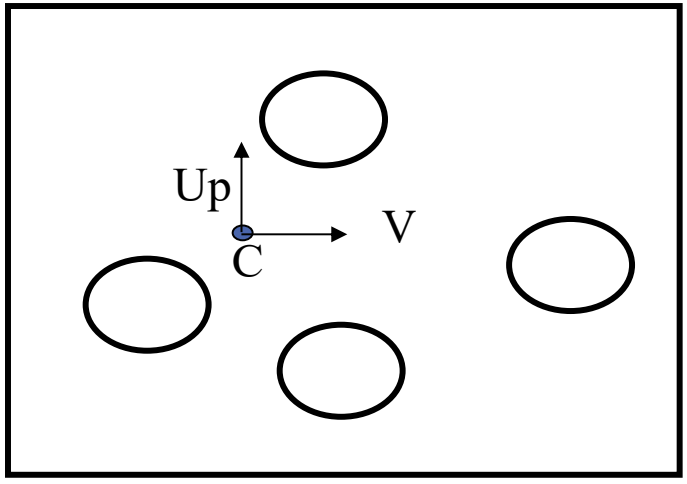


CDP = Centre De Projection

Section en coupe

l = largeur

h = hauteur



V = **N** x **Up**

Vue depuis la caméra

En résumé – 3 repères déjà définis

Repère Monde (M)

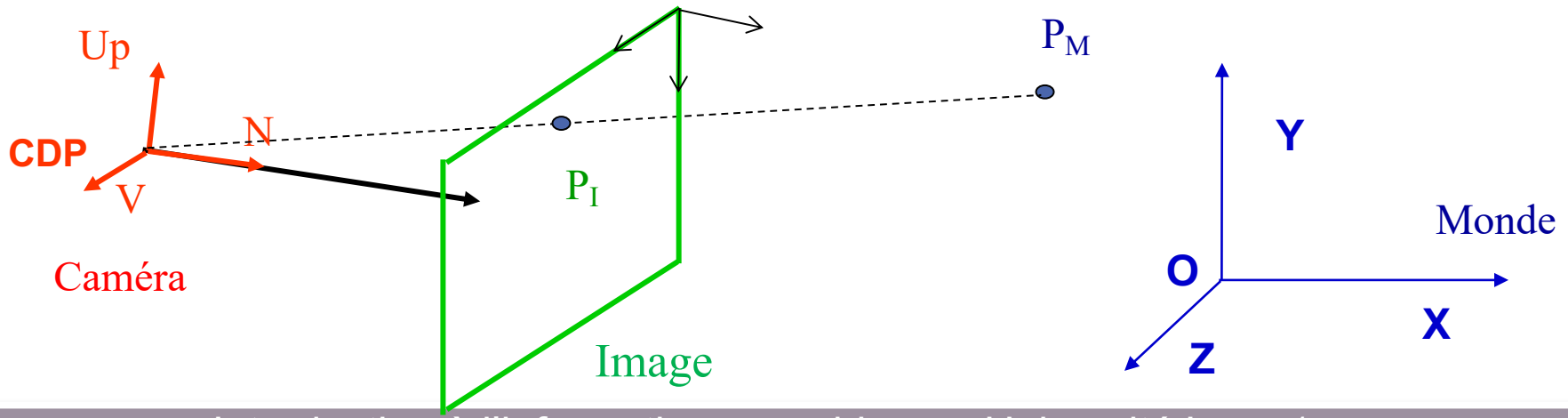
- Les points sont décrits dans l'espace monde
- Positionnement de la caméra, des lumières

Repère Caméra (C)

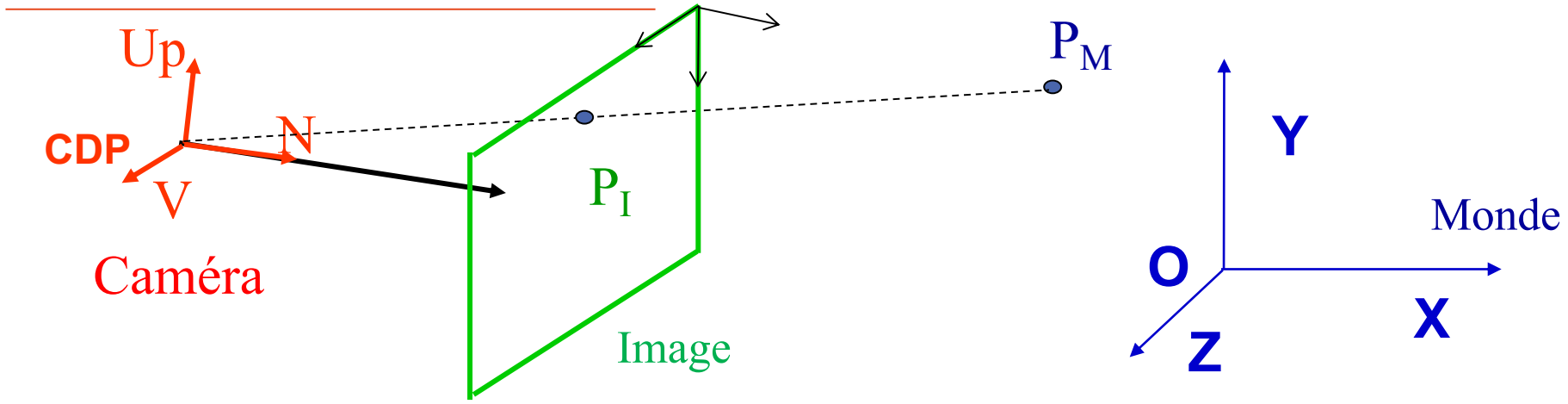
- Dans cet espace, CDP=O, N=-Z, U=Y, V=X

Repère Image (I)

- Dans cet espace, le pt (i,j,z) correspond au pixel (i,j) de l'image



Projection de la scène 3D vers l'Image en résumé



La projection de la scène 3D vers Image se fait en 2 étapes :

1. Tous les points de la scène 3D (repère M) sont ramenés dans le repère de la caméra (repère C)
2. Projection des points définis dans le repère de la caméra (repère C) vers le repère image (repère I)

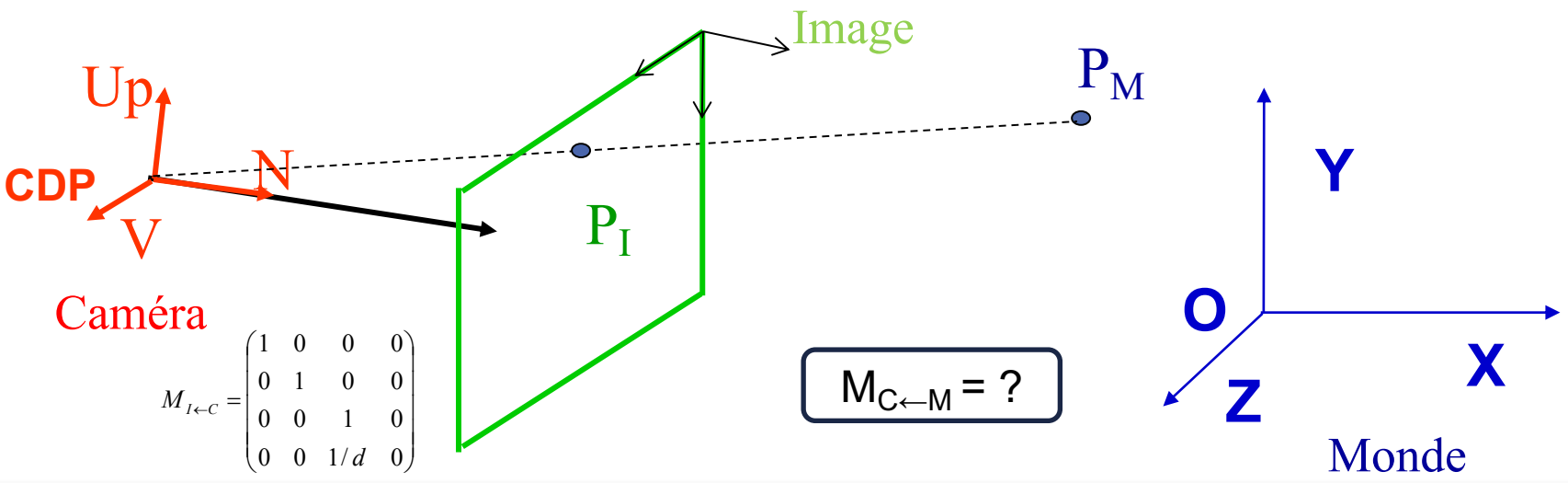
Utilisation de matrices de passage pour passer d'un repère à l'autre

Passage d'un repère à l'autre

Matrices de passage :

- $M_{C \leftarrow M}$ faisant passer un point du repère M au repère C
- $M_{I \leftarrow C}$ faisant passer un point du repère C au repère I

Scène 3D $\xrightarrow[M_{C \leftarrow M}]{}$ repère Caméra $\xrightarrow[M_{I \leftarrow C}]{}$ repère Image

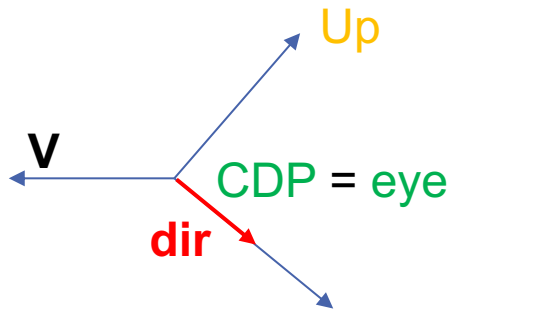


Passage d'un repère à l'autre

Reste à définir $M_{C \leftarrow M}$ faisant passer un point du repère M au repère C

Définition du repère de la caméra :

Position de la caméra (CDP) notée eye = (E_x, E_y, E_z)
Point de visée (N) noté Target = (T_x, T_y, T_z)
Vecteur Up = (U_x, U_y, U_z)



Calcul du vecteur de direction de la caméra : **dir** = $\frac{\text{target} - \text{eye}}{|| \text{ target} - \text{eye} ||}$

Calcul du vecteur V pour former repère orthonormée : **V** = **dir** x **Up**

Passage d'un repère à l'autre – Monde vers Caméra

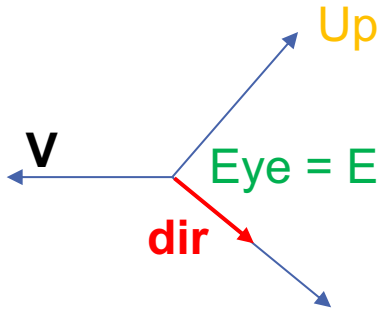
Matrice de translation : $T = \begin{pmatrix} 1 & 0 & 0 & -E_x \\ 0 & 1 & 0 & -E_y \\ 0 & 0 & 0 & -E_z \\ 0 & 0 & 1 & 1 \end{pmatrix}$

Pour « changer » d'origine

Matrice de rotation : $R = \begin{pmatrix} dir_x & dir_y & dir_z & 0 \\ Up_x & Up_y & Up_z & 0 \\ -V_x & -V_y & -V_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Pour prendre en compte rotation différente entre les 2 repères

Combinaison des matrices : $M_{C \leftarrow M} = R \ T$



Passage d'un repère à l'autre

$M_{C \leftarrow M}$ faisant passer un point du repère M au repère C

$$M = \begin{pmatrix} dir_x & dir_y & dir_z & -dir.Eye \\ Up_x & Up_y & Up_z & -Up.Eye \\ -V_x & -V_y & -V_z & -V.Eye \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

En résumé … différents repères / espaces

Espace objet

- Espace du modèle 3D lui-même

Espace monde

- Positionnement des objets, des lumières et de la caméra

Espace caméra

- Espace du point de vue de l'observateur

Espace image

- Contient les points projetés

Espace écran

- Image finale en pixels

En OpenGL

Matrices 4 x 4 pour passer d'un repère à l'autre :

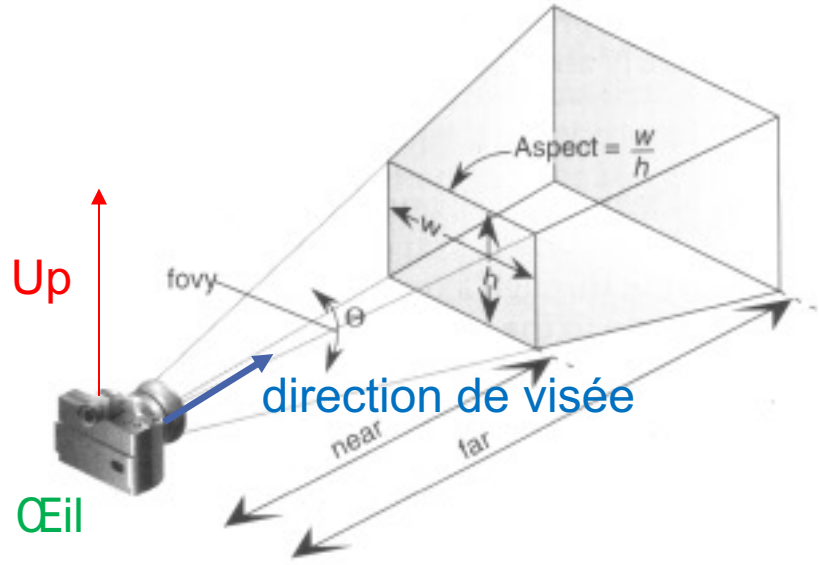
- Repère création objet vers repère monde : matrice *model*
- Repère monde vers caméra : matrice *view*
- Repère caméra vers projectif : matrice *projection*
- Repère projectif vers repère image : matrice *viewport*

$M_{C \leftarrow M}$ = GL_MODELVIEW

gluLookAt(Oeil, Direction, Up)

$M_{I \leftarrow C}$ = GL_PROJECTION

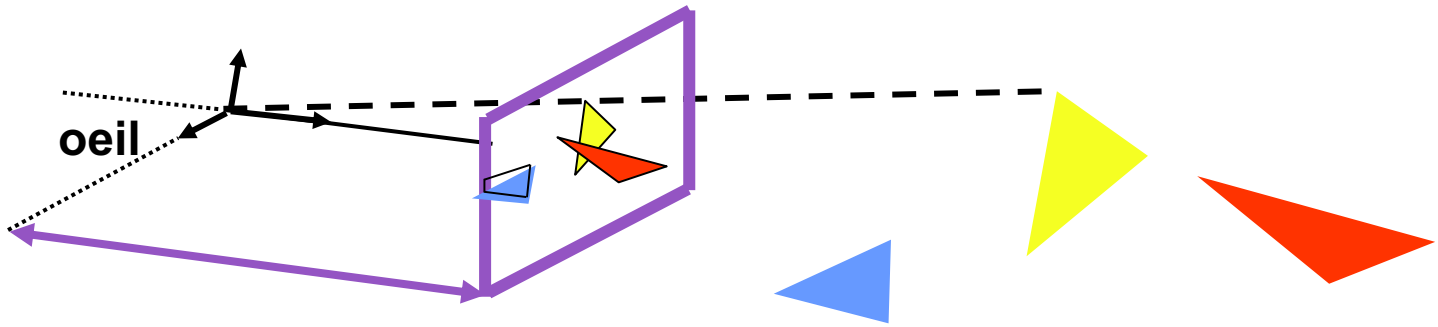
gluPerspective(*fovy*, *aspect*, *zNear*, *zFar*)



Pipeline graphique du rendu projectif

Détails du pipeline graphique pour un rendu projectif :

- Clipping
- Projection
- Remplissage des polygônes
  - Suppression des parties cachées
  - Illumination



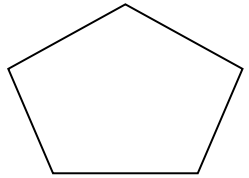
3e étape = remplissage des polygônes

Quel type de polygone ?

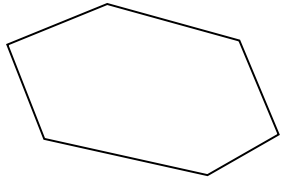
- Polygone simple
  - Aucune intersection d'arêtes

- Polygone convexe
  - Chaque angle intérieur $P_{i-1}P_iP_{i+1}$<Pi

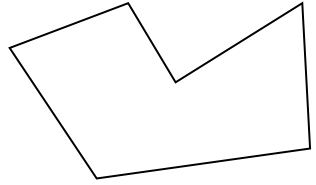
- Sinon polygone arbitraire ou complexe



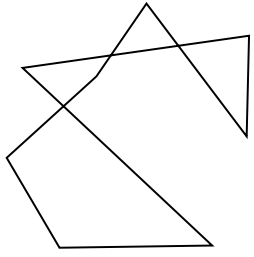
Simple



Convexe



Concave



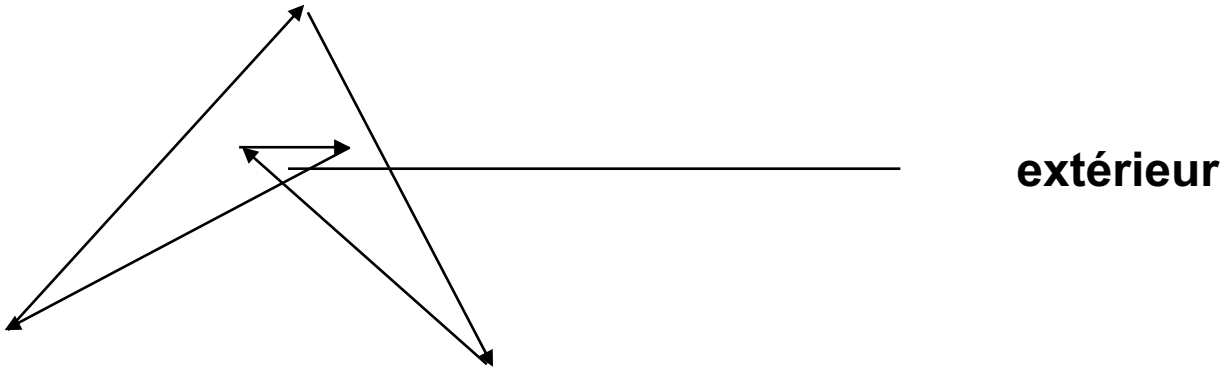
Arbitraire ou complexe

Remplissage des polygones

Il faut déterminer si on est à l'intérieur d'un polygone

- Règle pair - impair
  - Pour chaque point, trace une ligne horizontal allant à l'infini
  - Compte le nombre d'intersections avec des arêtes
    - Impaire veut dire à l'intérieur, pair à l'extérieur

    (compte les sommets Max et Min comme 0; les autres comme 1)



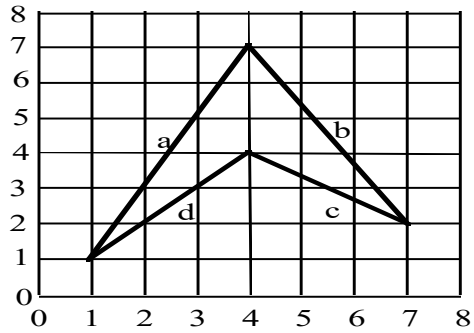
OK mais lent si on le fait pour chaque pixel de l'image…

Remplissage des polygones - Algorithme de Scan-Line

Consiste à calculer des segments horizontaux inclus dans le polygone

Idée de l'algorithme :

- Pour chaque ligne horizontale de l'image
  - Détermine les pixels à l'intérieur du polygone
  - Pixels marqués / affichés avec la bonne valeur / couleur



Principe de l'algorithme :

- Pour chaque ligne horizontale de l'image

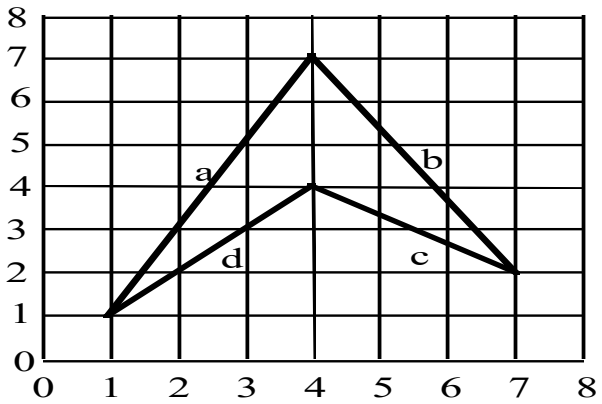
  Détermine les extrémités des segments horizontaux inclus dans le polygone
  - Extrémités = intersection entre ligne horizontale avec les arêtes du polygone

  Trie les extrémités dans l'ordre croissant des coordonnées en x

  Affiche les pixels à l'intérieur du polygone entre les paires d'extrémités
  - Règle de la parité pour savoir si pixel à l'intérieur ou non

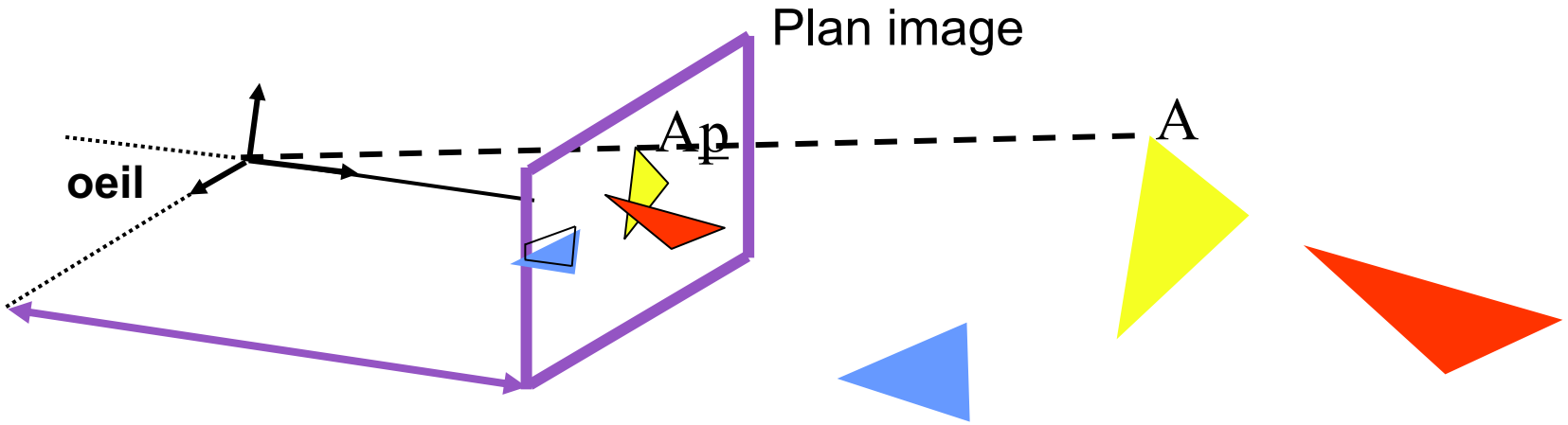
Remplissage des polygones - Algorithme de Scan-Line

Consiste à calculer des segments horizontaux inclus dans le polygone



Ligne	Intervalles de dessin
0	
1	1 à 1
2	1.5 à 2, 7 à 7
3	2.0 à 3, 5.5 à 6.4
4	2.5 à 5.8
5	3.0 à 5.2
6	3.5 à 4.6
7	
8	

Pipeline graphique du rendu projectif

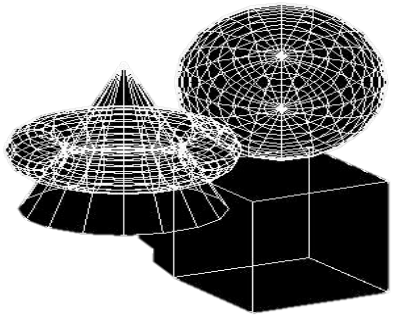


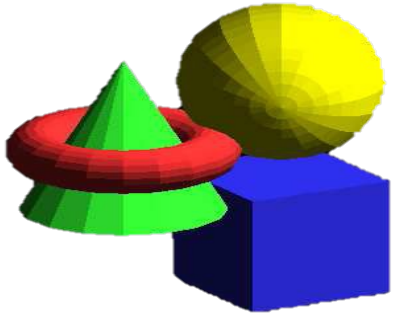
Détails du pipeline graphique pour un rendu projectif :

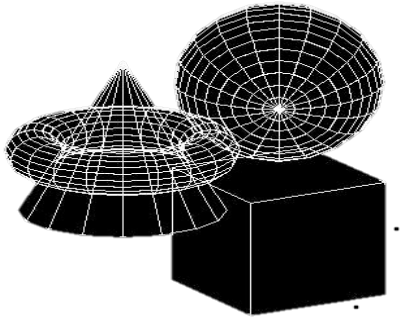
1. Clipping des polygones en 3D suivant la pyramide de vue
2. Projection des points sur le plan image
3. Remplissage des triangles (Rasterizing) dans l'image
   a. Suppression des parties cachées : Z-Buffer
   b. Calcul de la couleur : illumination

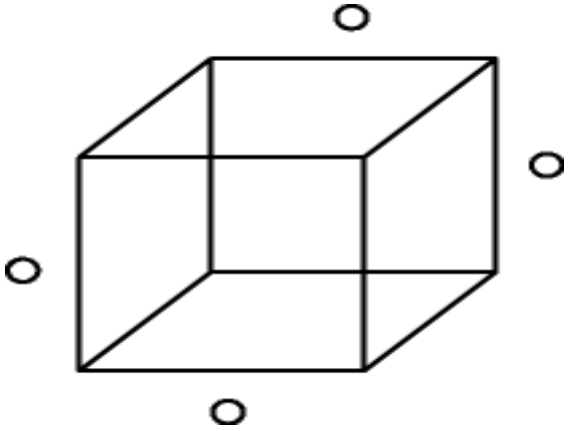
GPU

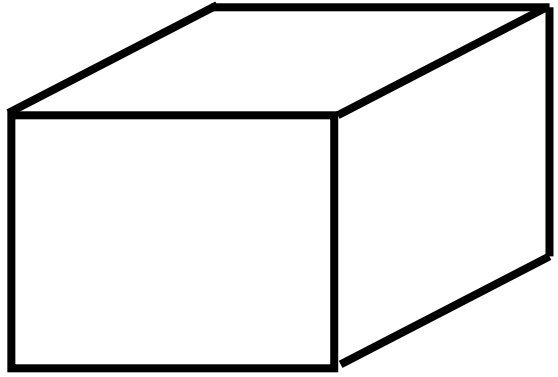
Élimination des parties cachées

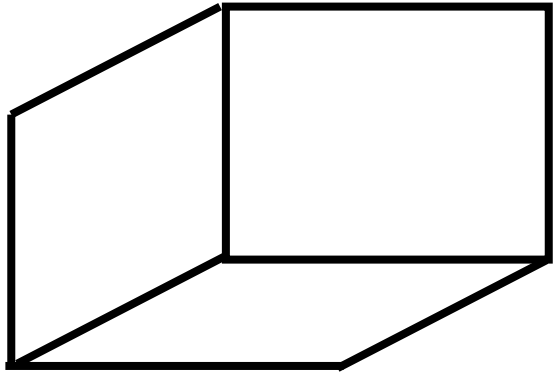








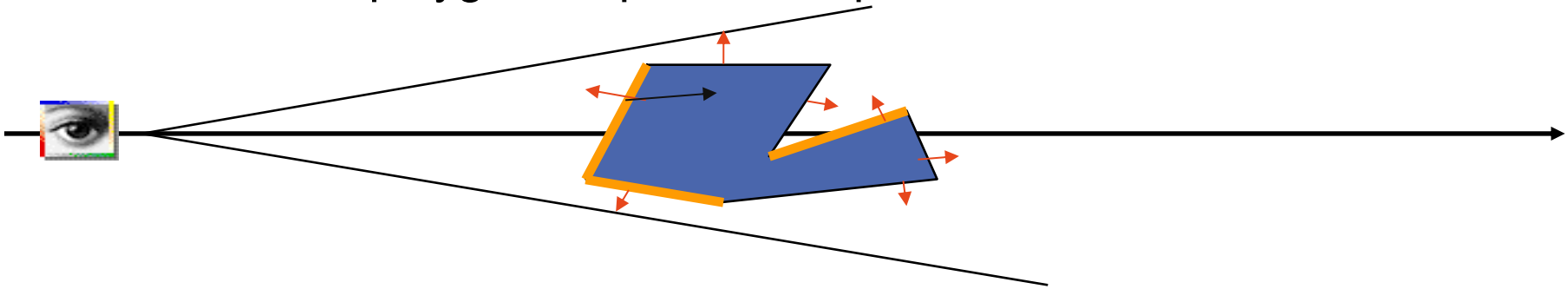




Problème : comment n'afficher que ce qui est visible ?

Algorithme du Back Face Culling

Éliminer tous les polygones qui ne sont pas tournés vers la caméra



Si le point de vue n'est pas devant le polygone on n'affiche pas le polygone

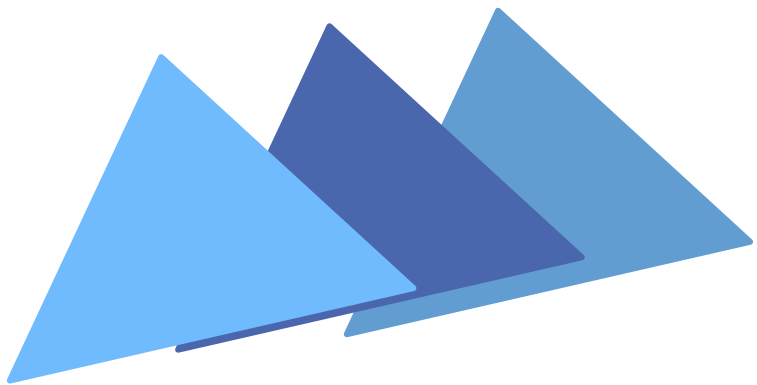
Simple produit scalaire à calculer :

- (Sommet de la face – Position de la caméra) . Vecteur normal à la face
- < 0 (angle obtus) : on garde le polygone
- > 0 (angle aigu) : on l'élimine

Pas suffisant pour plusieurs objets

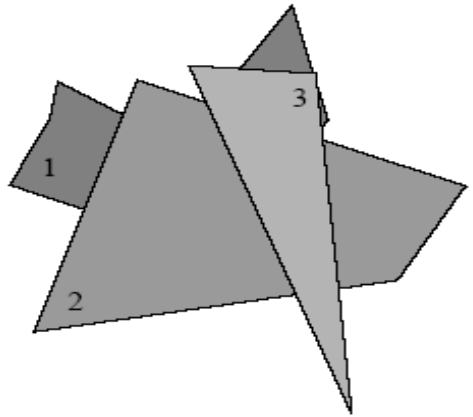
Suppression des parties cachées

Problème : comment n'afficher que ce qui est visible ?
cas où il y a plusieurs objets



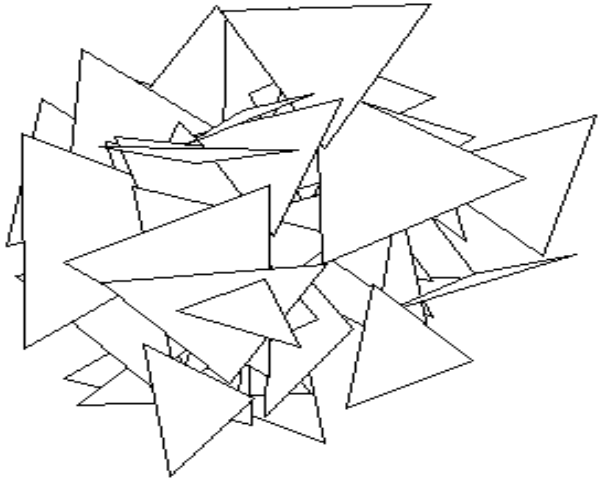
1ere idée : algorithme du peintre

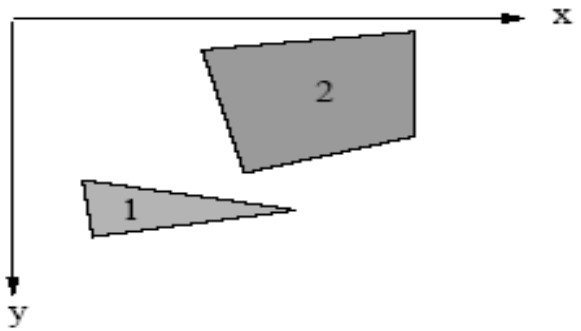
Peindre les facettes polygonales dans la mémoire vidéo suivant un ordre de distance décroissante au point d'observation.



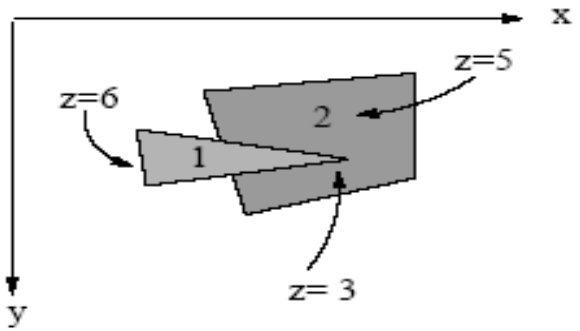
1ere idée : algorithme du peintre

Ambiguïtés

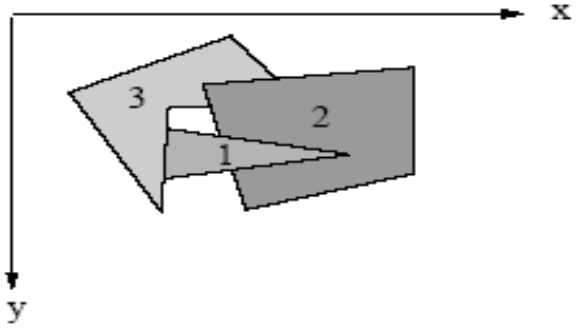




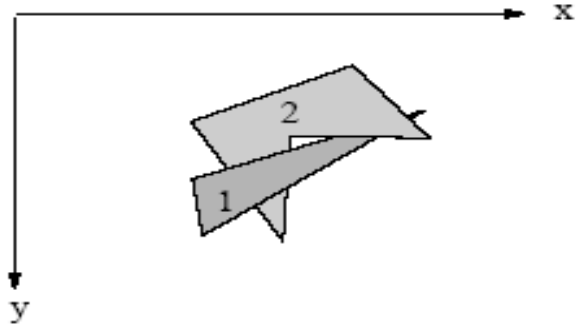
(a)



(b)



(c)



(d)

1ere idée : algorithme du peintre

Trier les facettes suivant les z décroissants dans le repère de la caméra

Résoudre les ambiguïtés dans la liste lorsque les facettes se recouvrent
découper les polygones ambigus

Projeter les facettes et remplir les polygones suivant la liste

- Le plus intuitif des algorithmes
- Mais problème des ambiguïtés à résoudre
- Coût en mémoire et temps de calcul :
  - Affichage direct à l'écran : O(p)
  - Il faut trier les polygones : O(n log$n$)
- Efficace surtout sur des petites scènes

2e idée - Algorithme du Z-Buffer

Structure de données : un tableau de la taille de l'écran

- Objectif : stocker la valeur minimal de z pour chaque pixel
- Initialisation : toutes les valeurs des pixels initialisées à l'infini

+inf	+inf	+inf
+inf	+inf	+inf
+inf	+inf	+inf
+inf	+inf	+inf

Principe de l'algorithme :

- Projection de tous les polygones
- Mise à jour des pixels suite à la projection du polygone

2e idée - Algorithme du Z-Buffer

Pour chaque polygone :

- Projeter le polygone sur le plan image

- Pour chaque pixel (i,j) de la projection du polygone
  - Calculer la valeur de *z* pour ce pixel

  - Si z < zbuffer[i][j] alors
    zbuffer[i][j] = z
    Afficher le pixel à l'écran de la couleur du polygone

  - Sinon dans le cas où le fragment du polygone est caché donc on ne fait RIEN

2e idée - Algorithme du Z-Buffer

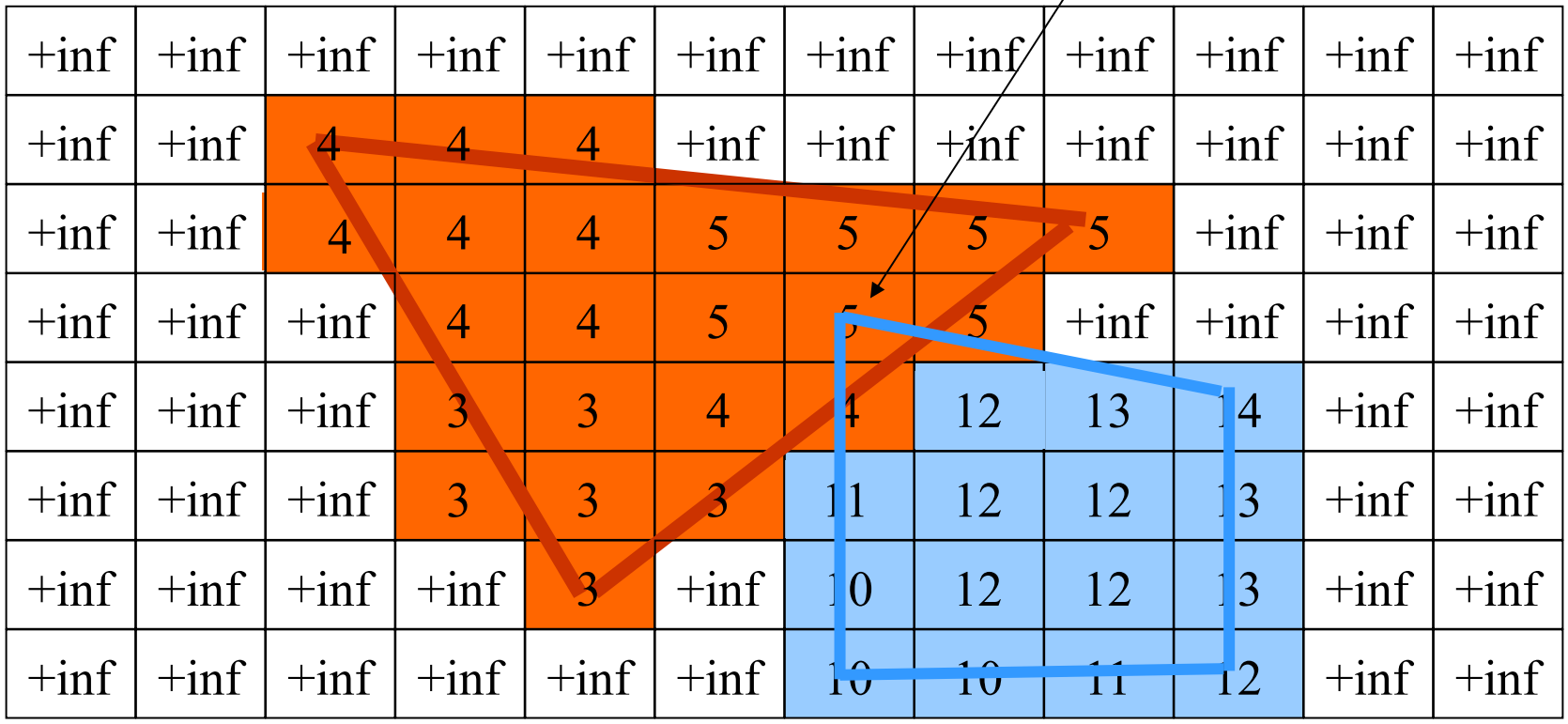
+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf
+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf
+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf
+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf
+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf
+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf
+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf
+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf

2e idée - Algorithme du Z-Buffer

+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf
+inf	+inf	4	4	4	+inf	+inf	+inf	+inf	+inf	+inf	+inf
+inf	+inf	4	4	4	5	5	5	5	+inf	+inf	+inf
+inf	+inf	+inf	4	4	5	5	5	+inf	+inf	+inf	+inf
+inf	+inf	+inf	3	3	4	4	+inf	+inf	+inf	+inf	+inf
+inf	+inf	+inf	3	3	3	+inf	+inf	+inf	+inf	+inf	+inf
+inf	+inf	+inf	+inf	3	+inf	+inf	+inf	+inf	+inf	+inf	+inf
+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf

2e idée - Algorithme du Z-Buffer

Z = 11>5 donc caché



+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf
+inf	+inf	4	4	4	+inf	+inf	+inf	+inf	+inf	+inf	+inf
+inf	+inf	4	4	4	5	5	5	5	+inf	+inf	+inf
+inf	+inf	+inf	4	4	5	5	5	+inf	+inf	+inf	+inf
+inf	+inf	+inf	3	3	4	4	12	13	14	+inf	+inf
+inf	+inf	+inf	3	3	3	11	12	12	13	+inf	+inf
+inf	+inf	+inf	+inf	3	+inf	10	12	12	13	+inf	+inf
+inf	+inf	+inf	+inf	+inf	+inf	10	10	11	12	+inf	+inf

2e idée - Algorithme du Z-Buffer

Comment calculer la valeur de *z* pour les pixels ?

Enjeu : calculer la valeur de Z pour chaque fragment du polygone en cours de remplissage

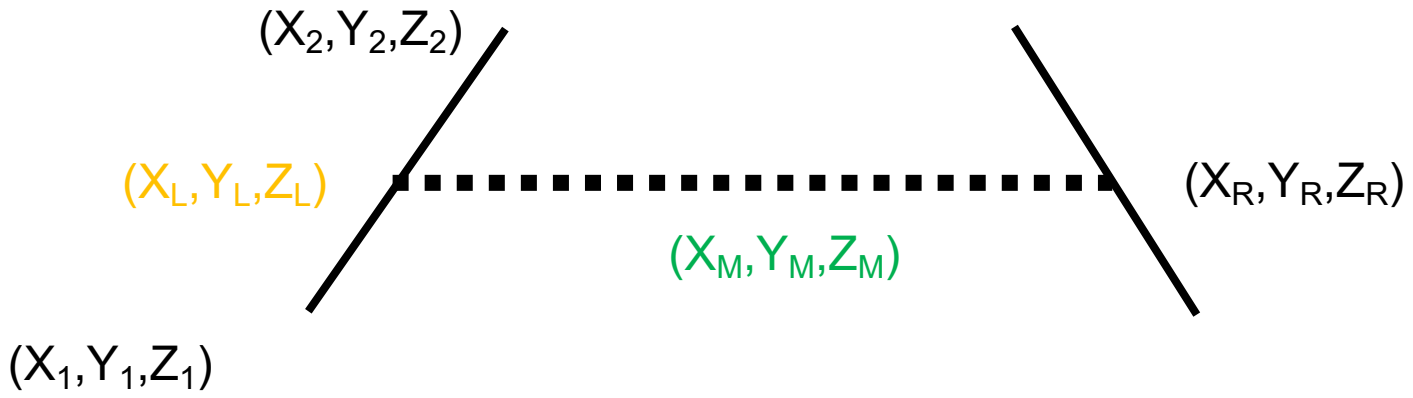
- Une idée : en utilisant l'équation du plan

  ax + by + cz + d = 0 => z = (d – ax - by) / c Coûteux!! Il faut faire mieux …

- Autre idée : en faisant une interpolation

Interpolation de la profondeur

- Interpoler z le long de l'arête du polygone — interpolation linéaire
- Puis interpoler z sur la ligne de remplissage — interpolation bi-linéaire



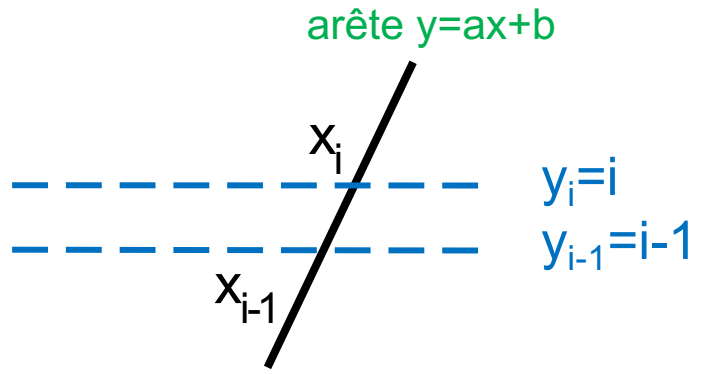
Interpolation de la profondeur

Quelques rappels sur **l'interpolation linéaire**

- Equation d'une droite : y = a x + b donne équation pour une arête
- Considère deux lignes horizontales de l'image qui intersectent l'arête : i et i-1

- On a ainsi comme relations :

$$y_i = i = ax_i + b$$
$$y_{i-1} = i - 1 = ax_{i-1} + b$$
$$x_i = x_{i-1} + \frac{1}{a}$$



Avec pente de la droite :
a = (variation de y) / (variation de x) = $(y_i - y_{i-1}) / (x_i - x_{i-1})$ = $(i - (i-1)) / (x_i - x_{i-1})$
=> $1/a = x_i - x_{i-1}$

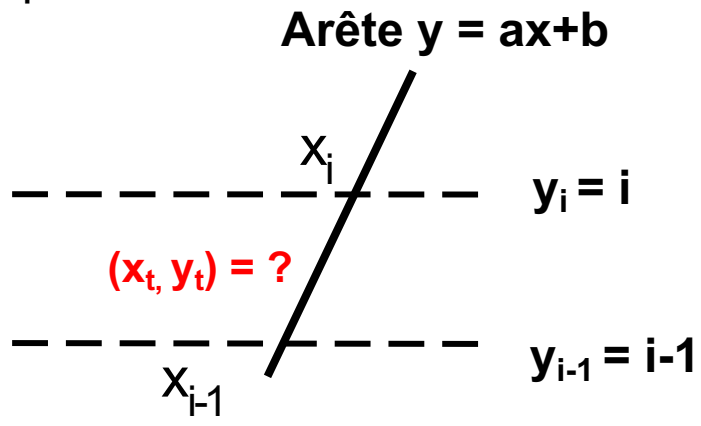
Interpolation de la profondeur

Pente de la droite aussi égale à :

a = (variation de y) / (variation de x) = $(y_t - y_{i-1}) / (x_t - x_{i-1})$

Ainsi, valeur sur la droite correspondant à x_t donnée par :

$y_t = a\,(x_t - x_{i-1}) + y_{i-1}$

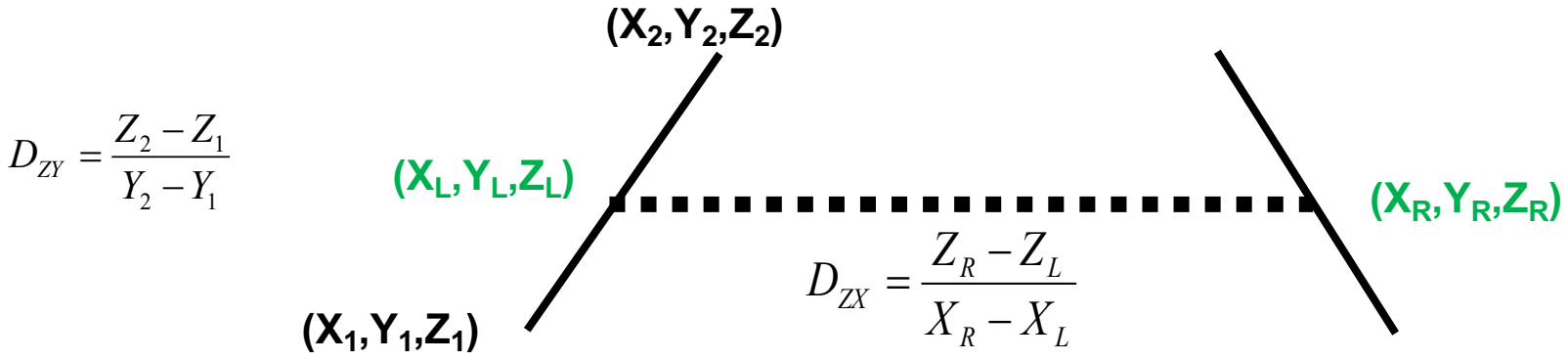


Interpoler z le long de l'arête du polygone : ok !

Interpolation de la profondeur

On fait interpolation linéaire sur les deux arêtes : deux points obtenus

Puis interpoler z sur la ligne de remplissage en refaisant pareil : interpolation bi-linéaire



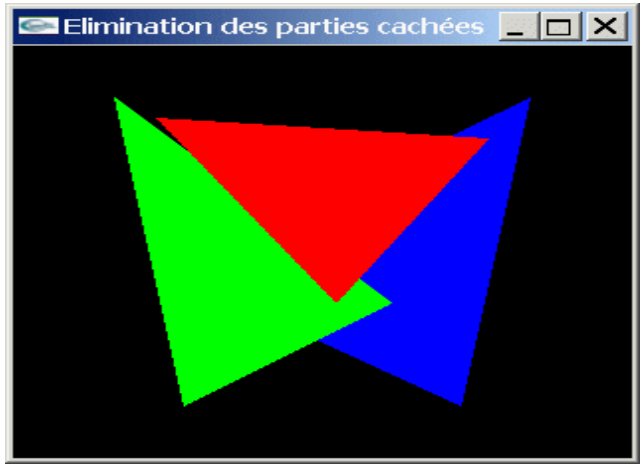
Z-buffer en OpenGL

Effacer le buffer et le Zbuffer entre chaque image

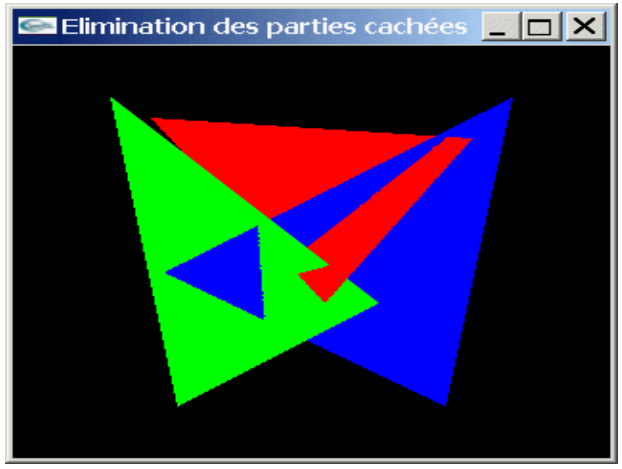
```
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
```

Activer le test des Z avec le Z-buffer

```
glEnable(GL_DEPTH_TEST) ;
```



non activé



activé

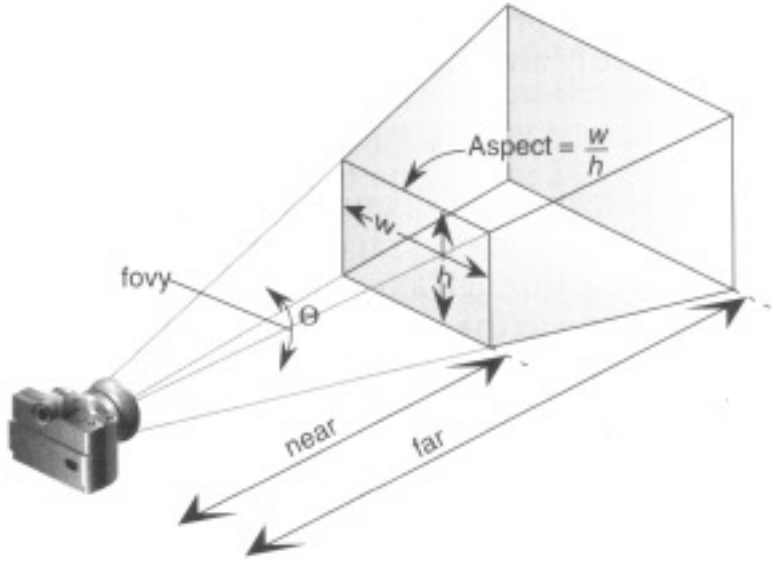
Z-buffer en OpenGL

Régler la précision du Zbuffer avec NEAR et FAR dans la matrice de projection

```
gluPerspective(fovy, aspect, zNear, zFar)
```

Ne jamais mettre zNear à 0 (problème de division par 0 dans la matrice générale de projection ; intervalle des z infini)

En général ~ zNear = 1.0 et zFar = 1000.0



Ce qu'il faut retenir / comprendre de ce cours

Notions de plusieurs repères :

- Repère Monde
- Repère Caméra
- Repère Image

Les étapes du pipeline graphique (cas rendu projectif) :

1. **Clipping** des polygones en 3D suivant la pyramide de vue
2. **Projection** des points sur le plan image
3. **Remplissage** des triangles dans l'image
   a. Suppression des parties cachées : **Z-Buffer**
   b. Calcul de la couleur : **illumination**