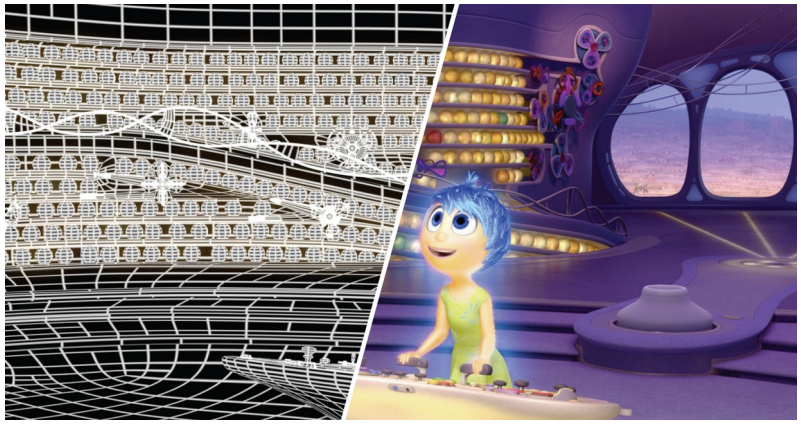
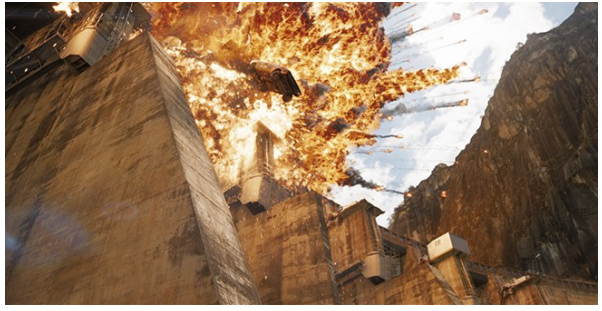


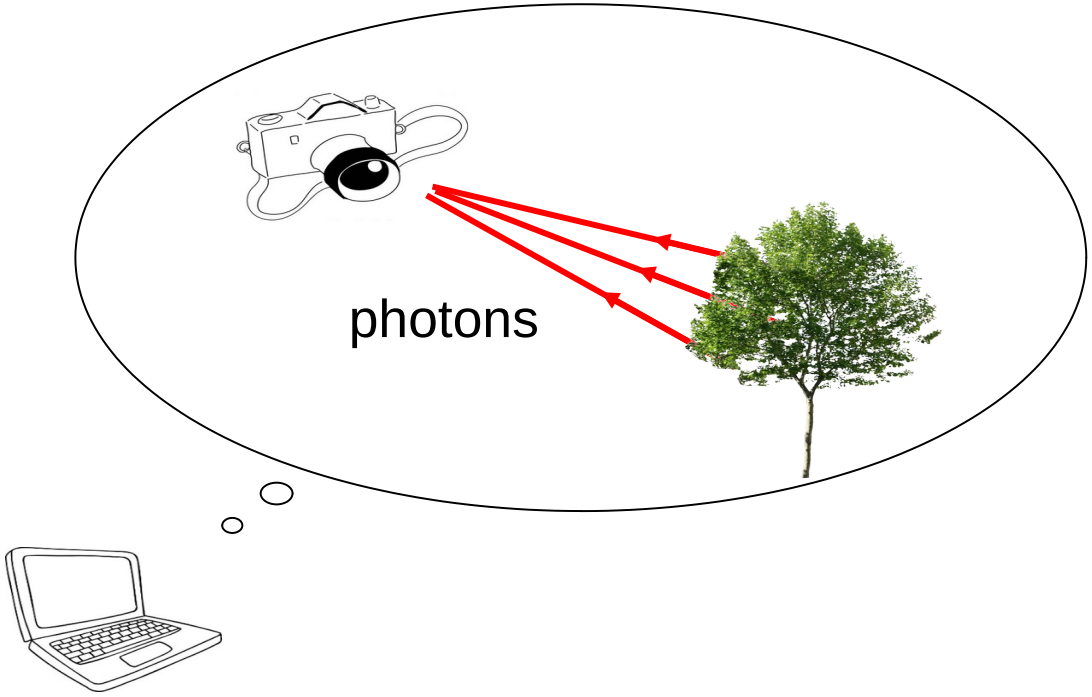
Rendu

Florence Zara (semestre automne)
LIRIS, équipe ORIGAMI
Université Lyon 1

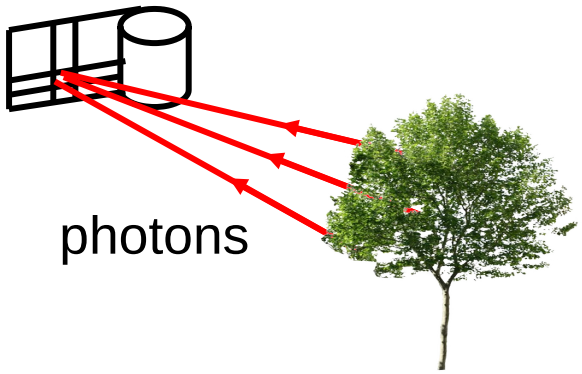


Synthèse d’images

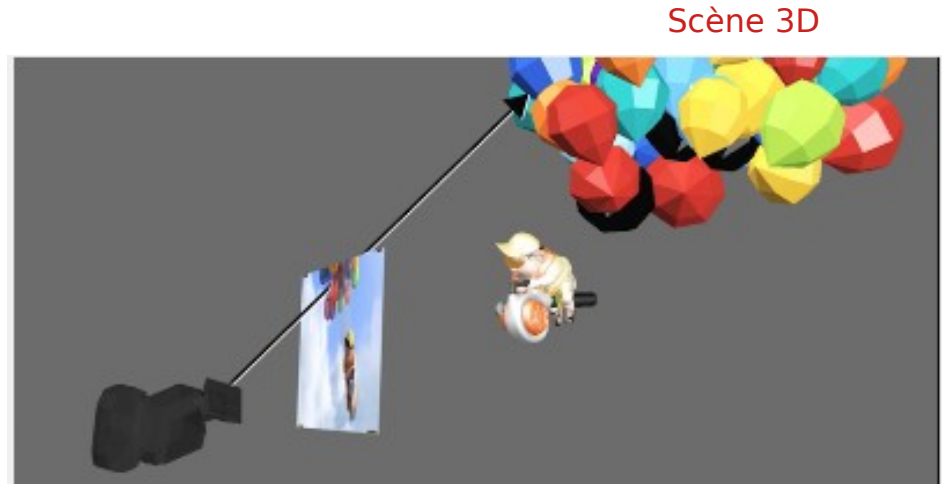
Comment dessiner une image ?



Analogie appareil photo et SI

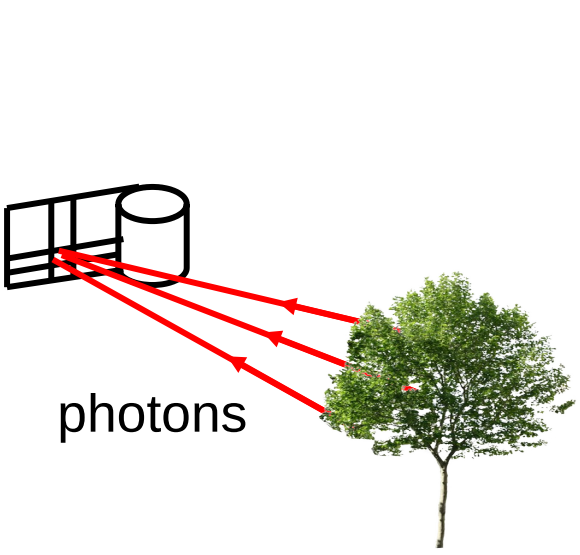


Appareil photos

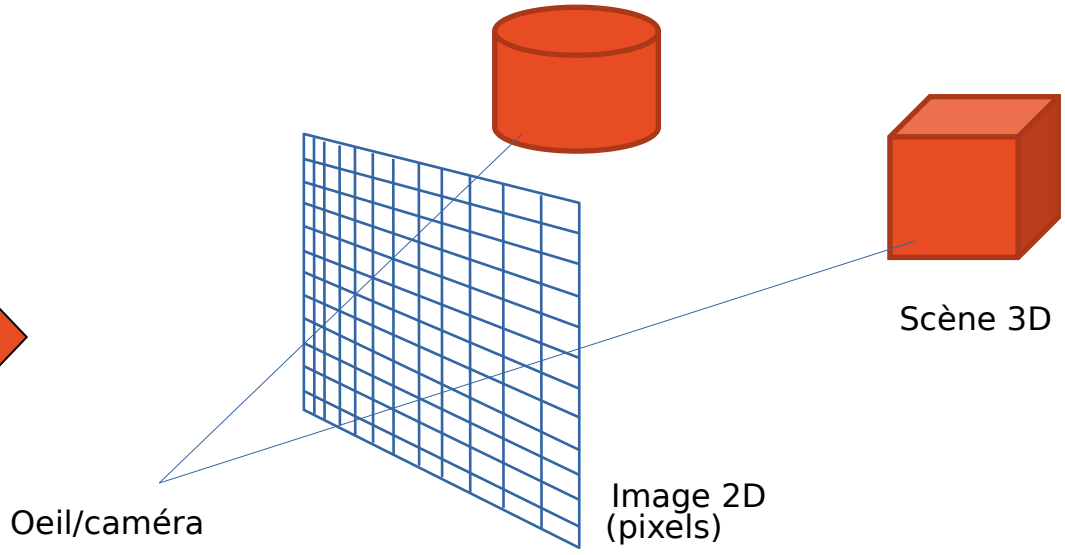


Synthèse d'Images

Analogie appareil photo et SI



Appareil photos



Synthèse d'Images

Pipeline graphique

Pipeline graphique = ensemble des traitements permettant de dessiner des objets

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

Exemple : dessin d'un point

- 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 qui 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**

Pipeline graphique

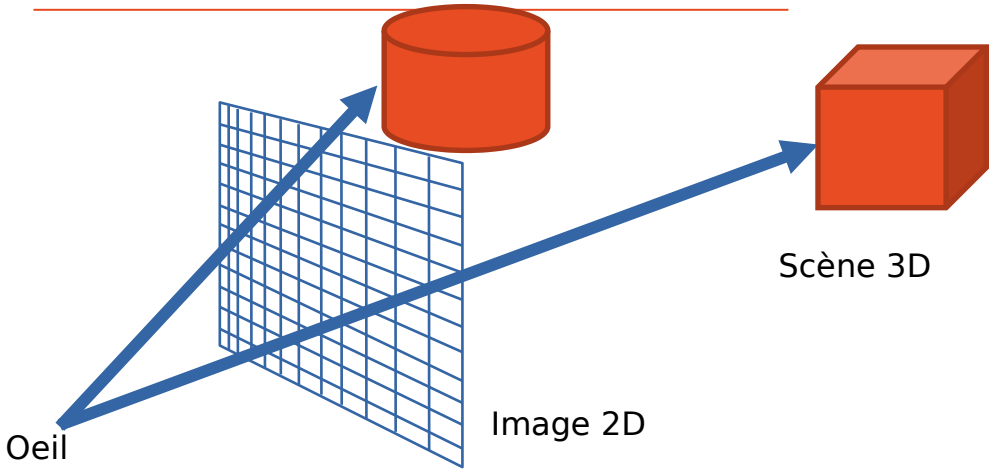
Pipeline Graphique OpenGL découpé en 2 parties :

- **Traitement de la géométrie, sommets des triangles :**
  - Passage des coordonnées des sommets selon un repère quelconque en positions 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 ?

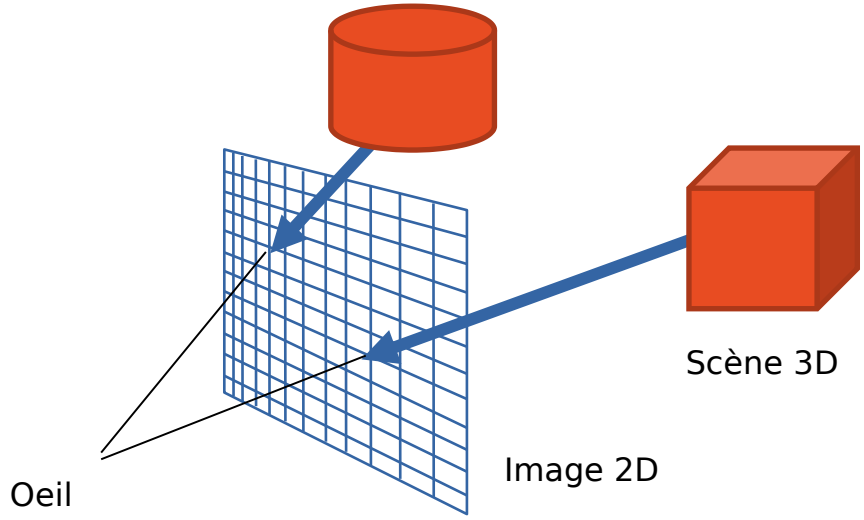
2 approches duales en SI



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

Ray-tracing

Donne une image réaliste mais lent

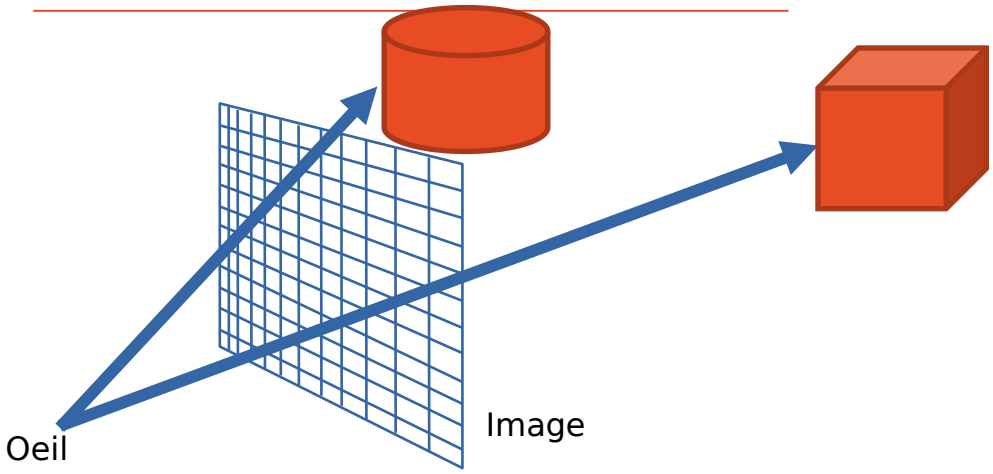


Les objets sont projetés sur
l'écran dans la direction de l'œil

Rendu projectif

Effectué sur GPU en temps réel

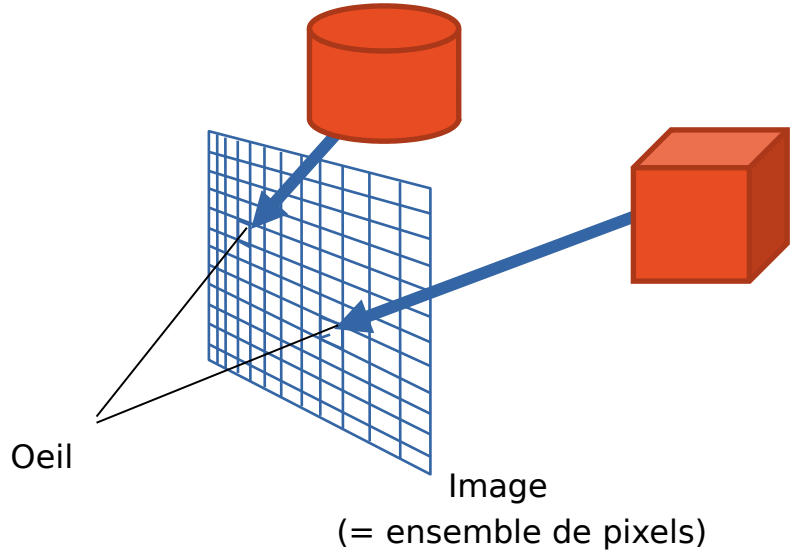
2e approche : rendu projectif



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

Ray-tracing

Donne une image réaliste mais lent

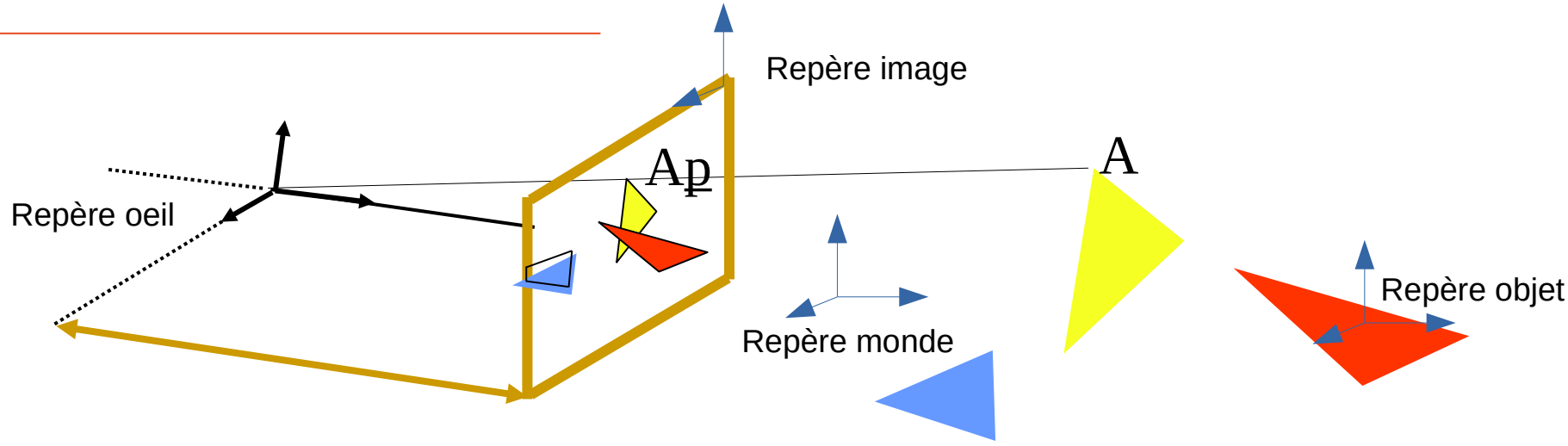


Les objets sont projetés sur
l'écran dans la direction de l'œil

Rendu projectif

Effectué sur GPU en temps réel

Rendu projectif : pipeline graphique



Les sommets existent dans plusieurs repères

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

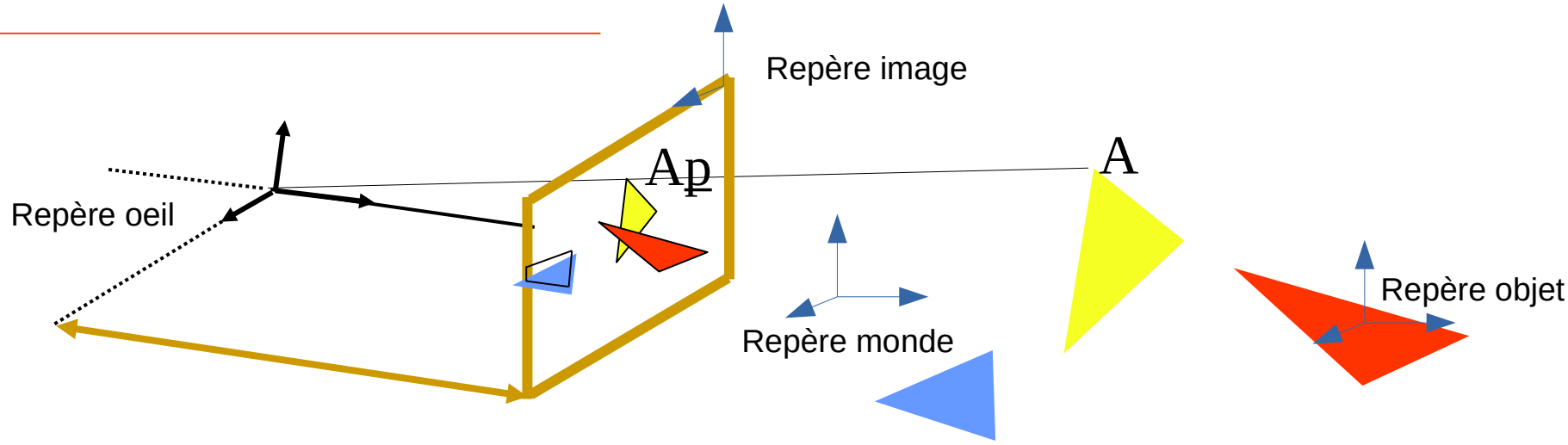
Repère monde/scène
Positionnement des objets, des lumières et de la caméra

Repère de la caméra/oeil
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

Rendu projectif : pipeline graphique

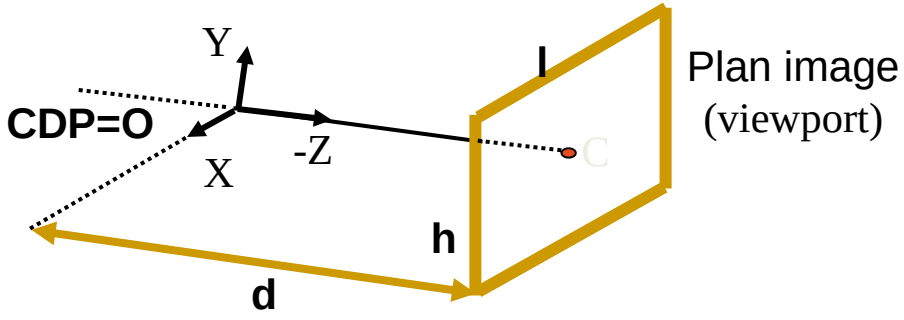


Pipeline graphique pour un rendu projectif :

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**

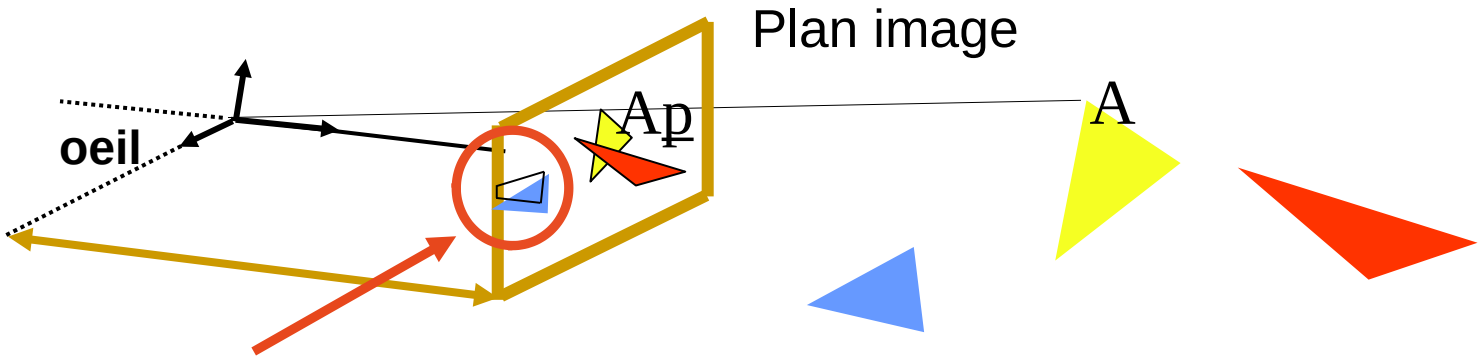
Définissons la caméra / repère de l'œil

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



Clipping selon la pyramide de vue

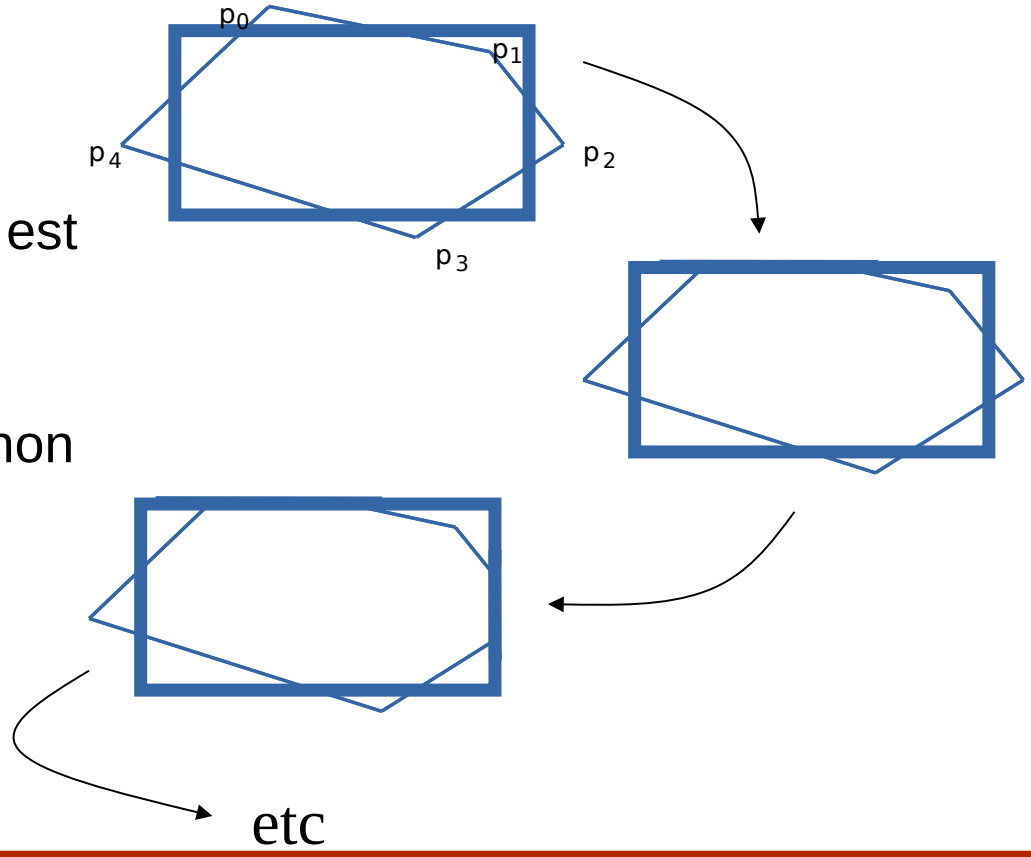
Trouver la partie exacte d'un polygone se trouvant dans la pyramide de vue



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

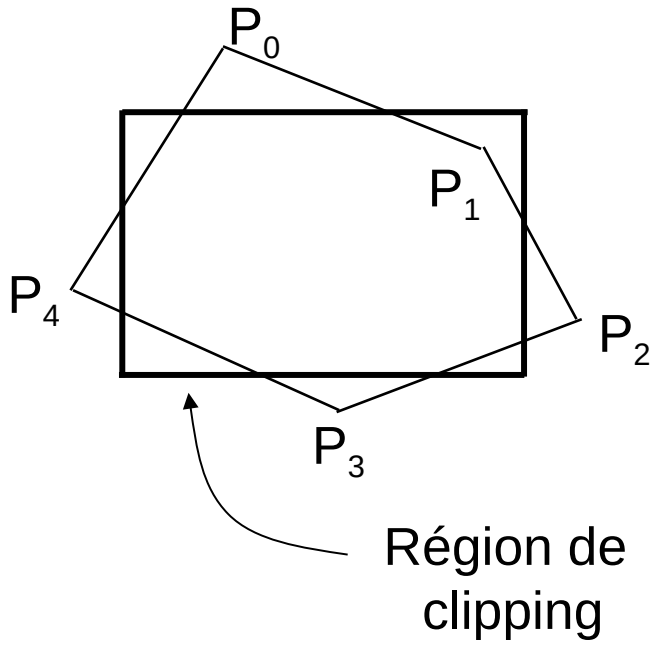
Algorithme de Sutherland-Hodgman

- 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



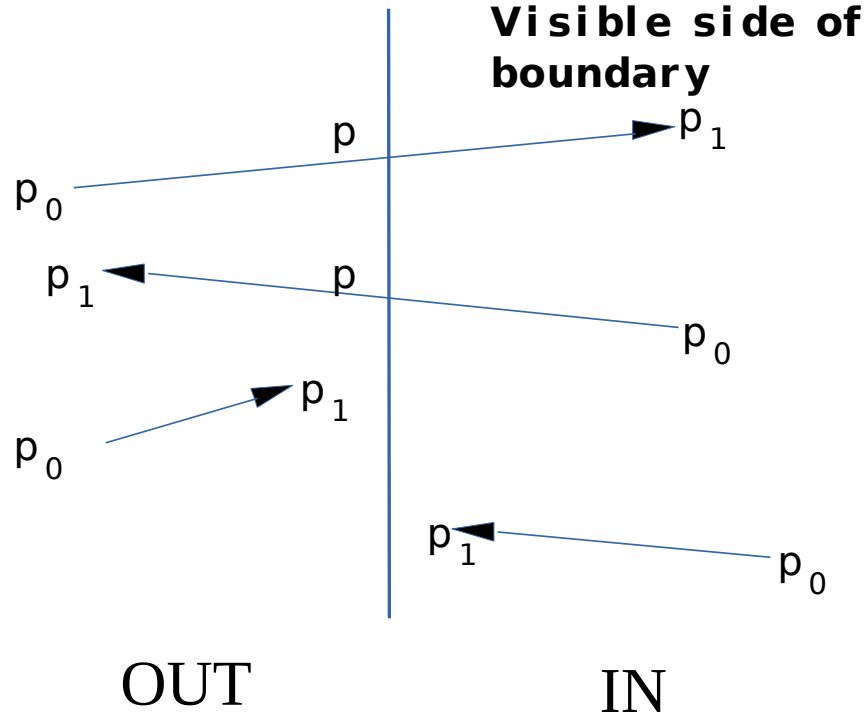
Clipping suivant une région

- 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

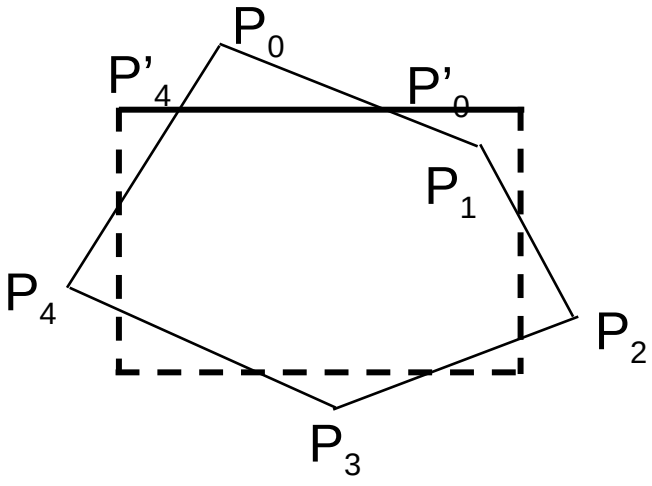


Clipping d'une arête

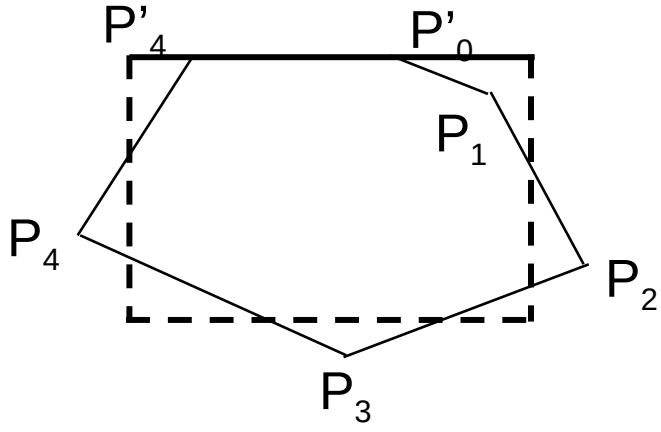
- P_0,P_1 une arête. 4 cas possibles :
  - Entrée dans la région de clipping
    - ajouter P et P_1
  - Sortie de la région de clipping
    - ajouter P
  - Entièrement en dehors
    - ne rien faire
  - Entièrement à l'intérieur
    - ajouter P1
- *P est le point d'intersection*



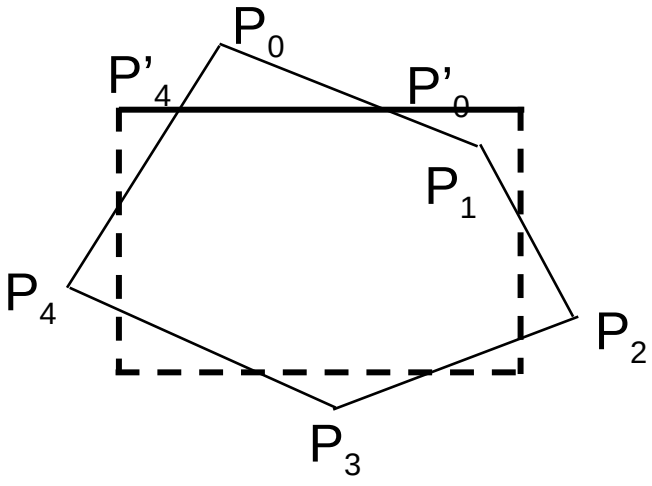
Sutherland-Hodgman – un exemple

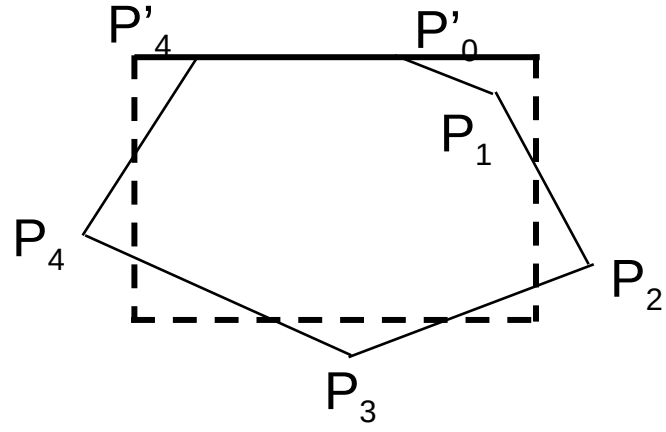


Etape 0 : vide
Etape 1 : ?
…



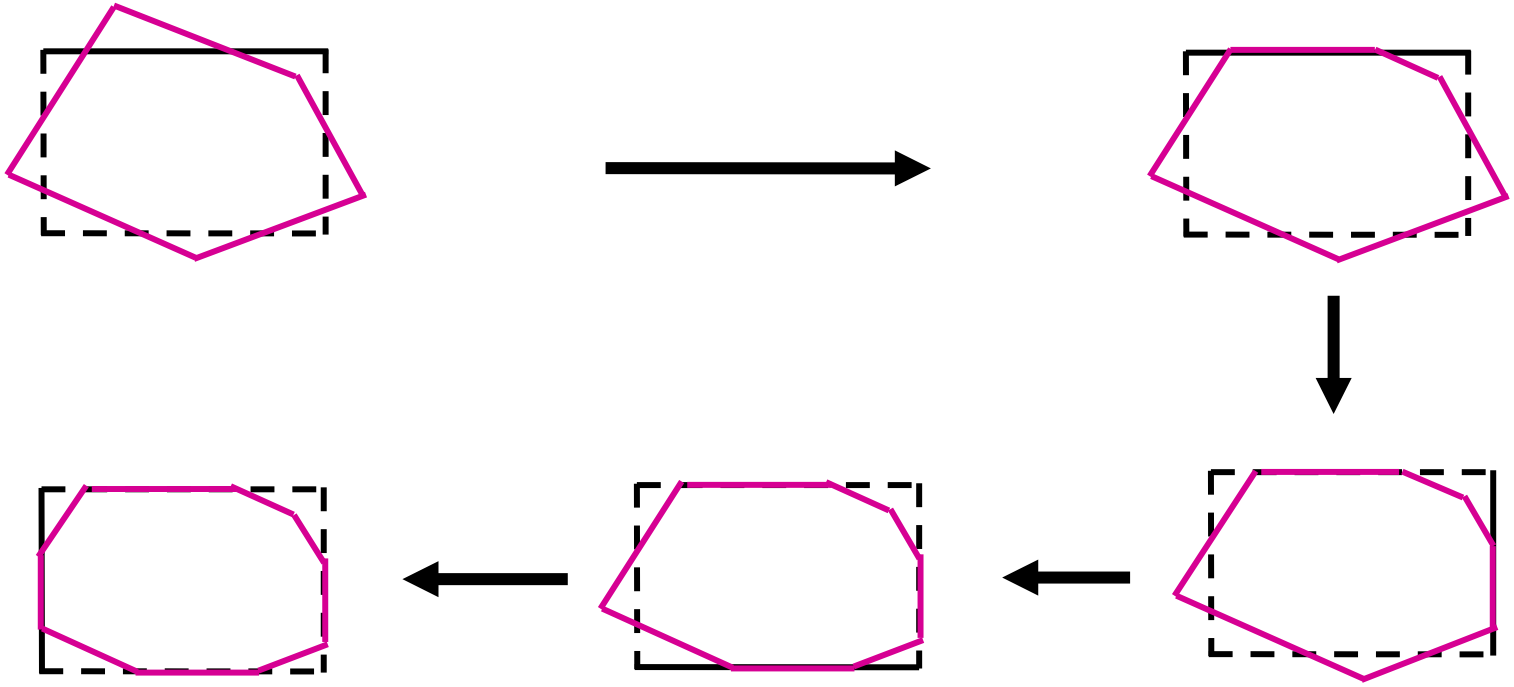
Sutherland-Hodgman – un exemple





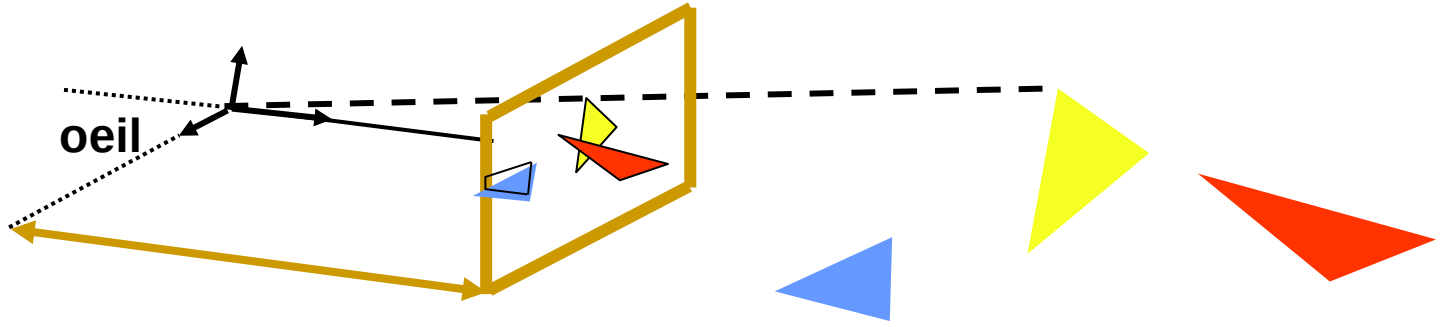
Etape 0 : vide
Etape 1 : $P'_0\ P_1$
Etape 2 : $P'_0\ P_1\ P_2$
Etape 3 : $P'_0\ P_1\ P_2\ P_3$
Etape 3 : $P'_0\ P_1\ P_2\ P_3\ P_4$
Etape 4 : $P'_0\ P_1\ P_2\ P_3\ P_4\ P'_4$

Sutherland-Hodgman – un exemple



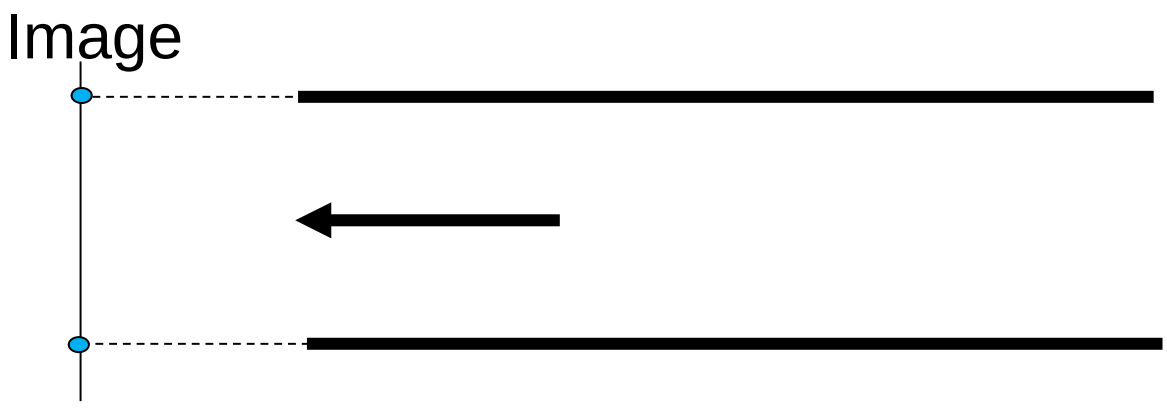
Pipeline graphique

- Pipeline dans le cadre du rendu projectif :
  - Clipping
  - Projection
  - Remplissage des polygônes
    - Suppression des parties cachées
    - Illumination



Projection parallèle

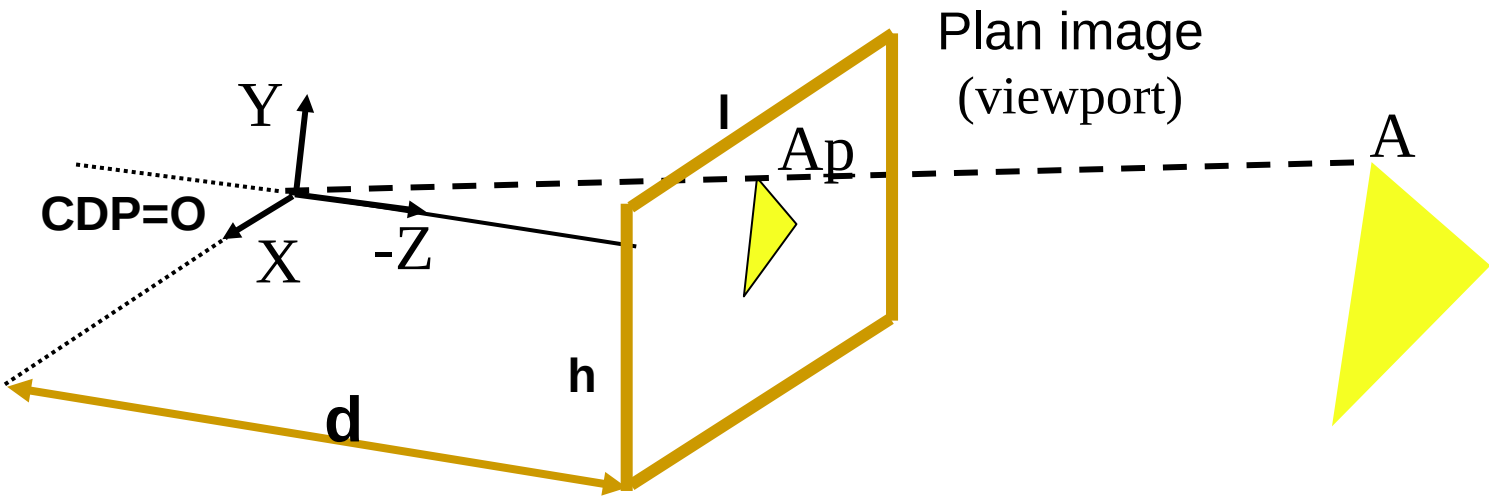
- Projection parallèle sur le plan z = 0
- Direction De Projection (DDP) est (0,0,1)
- P = (x,y,z) → P’ = (x, y, 0)



Projection parallèle

- Manque de réalisme

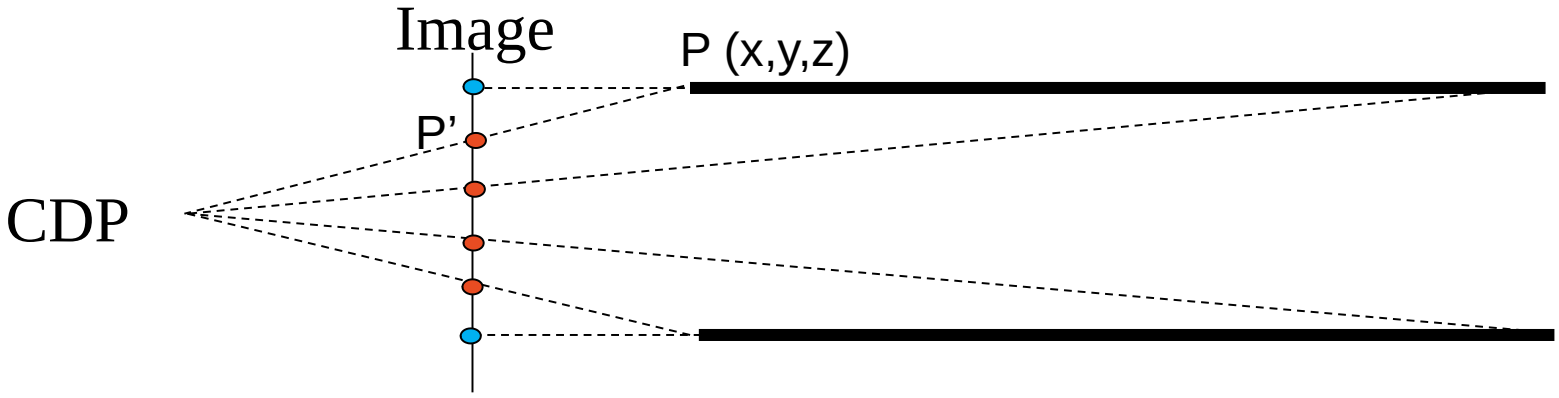
Projection perspective

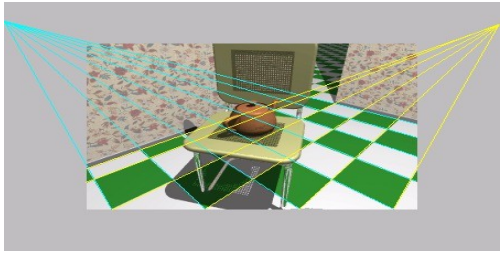


1. Projette les points du triangle sur le plan image par rapport au CDP

Projection perspective

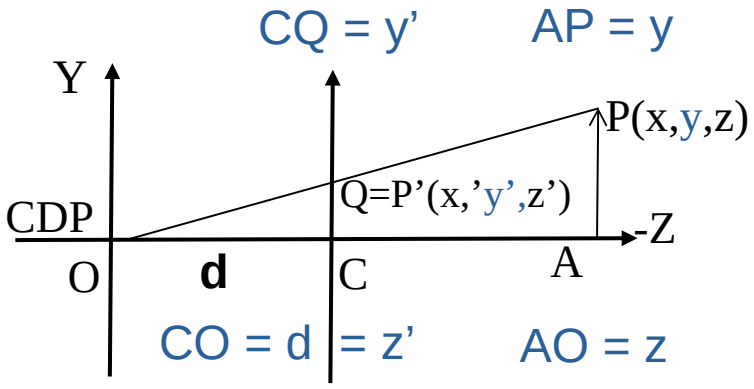
- Projette vers le centre de projection CDP
- P = (x,y,z) → P’ = ??





Projection perspective

Dans le plan (Y, -Z) :



Plan image

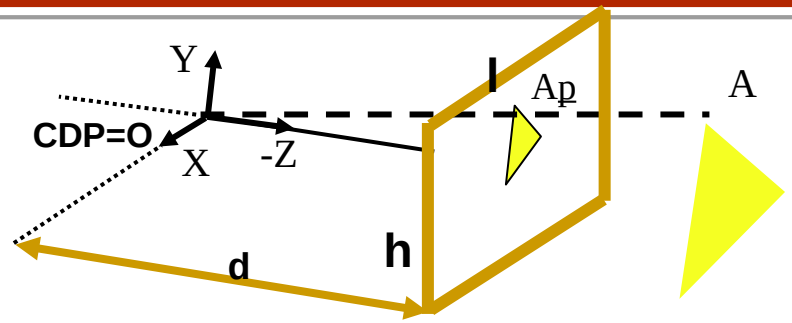
$$\frac{CQ}{AP} = \frac{CO}{AO} \Leftrightarrow CQ = y' = \frac{y.d}{z}$$

Par analogie, dans le plan (X, -Z) : $x' = \frac{x.d}{z}$

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

d=distance focale

Projection perspective



Au final, on a :

$$\begin{cases} x' = x\, d/z \\ y' = y\, d/z \\ z' = d \end{cases}$$

On souhaiterait définir la matrice de transformation correspondante :

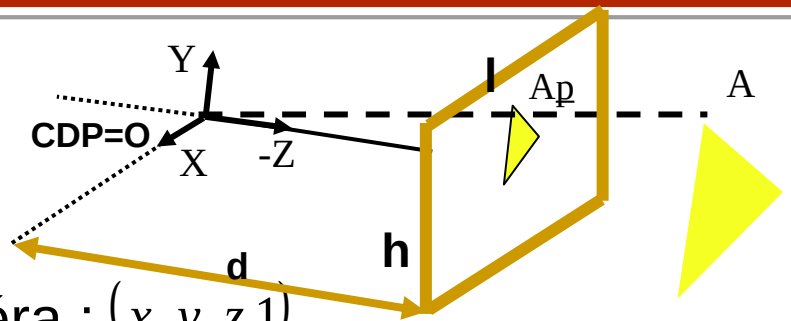
- A partir d'un point défini dans le repère de la caméra
- Obtenir ses coordonnées dans le repère de l'image

<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 : $M_{I \leftarrow C}$



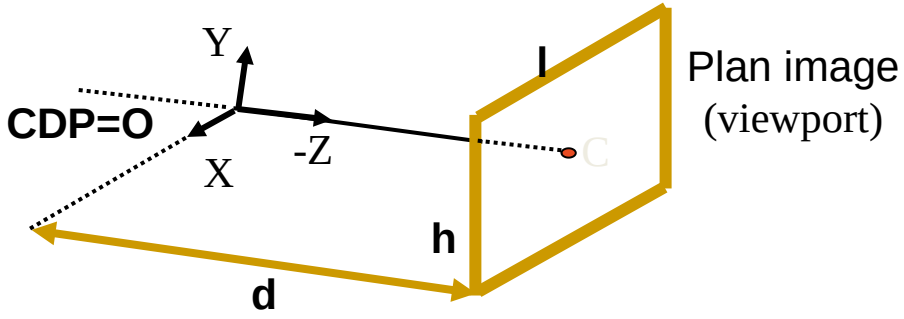
- **Entrée :** un point dans l'espace de la caméra : $(x, y, z, 1)$
- **Résultat :** un point dans l'espace Image : (wx', wy', wz', w)

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

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

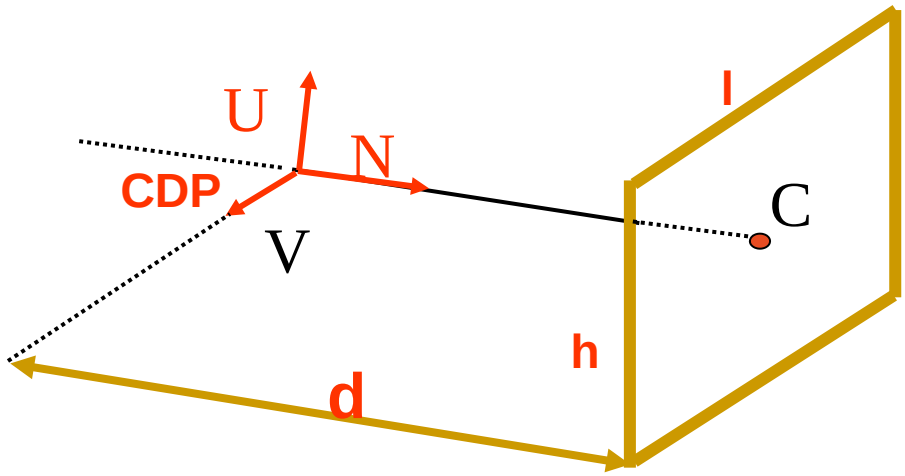
Repère caméra != repère monde

- Jusqu'à présent on avait :
  - CDP = Origine O
  - La caméra regarde vers –Z
  - Le haut de la caméra est Y

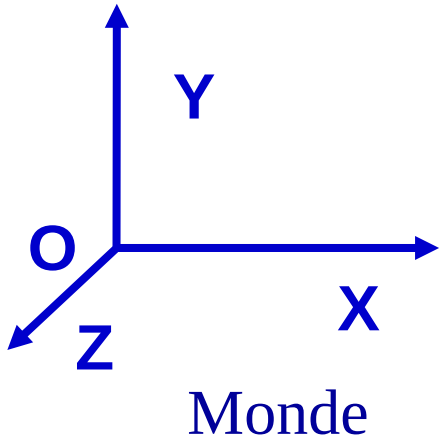


On aimerait pouvoir placer la caméra n'importe où !
On va définir un repère pour la caméra (différent du repère du monde)

Caméra plus générale



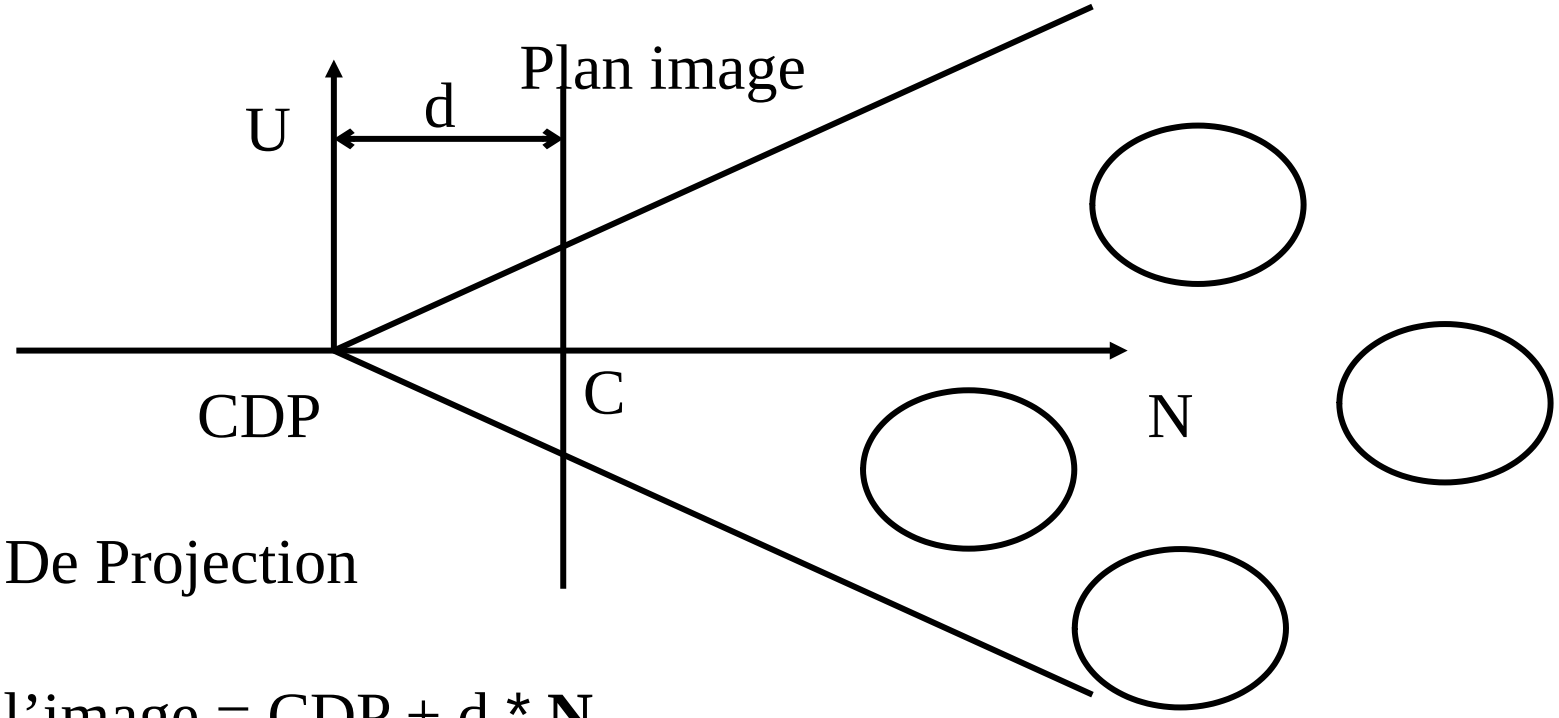
Plan image
(viewport)



CDP = Centre De Projection (!=O)
N = Direction de vue (N ⊥ au plan image), |N|=1
U = Direction du haut de la camera (Up), |U|=1
V = le 3e vecteur pour que UVN 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
mxn = taille en pixels de l'image

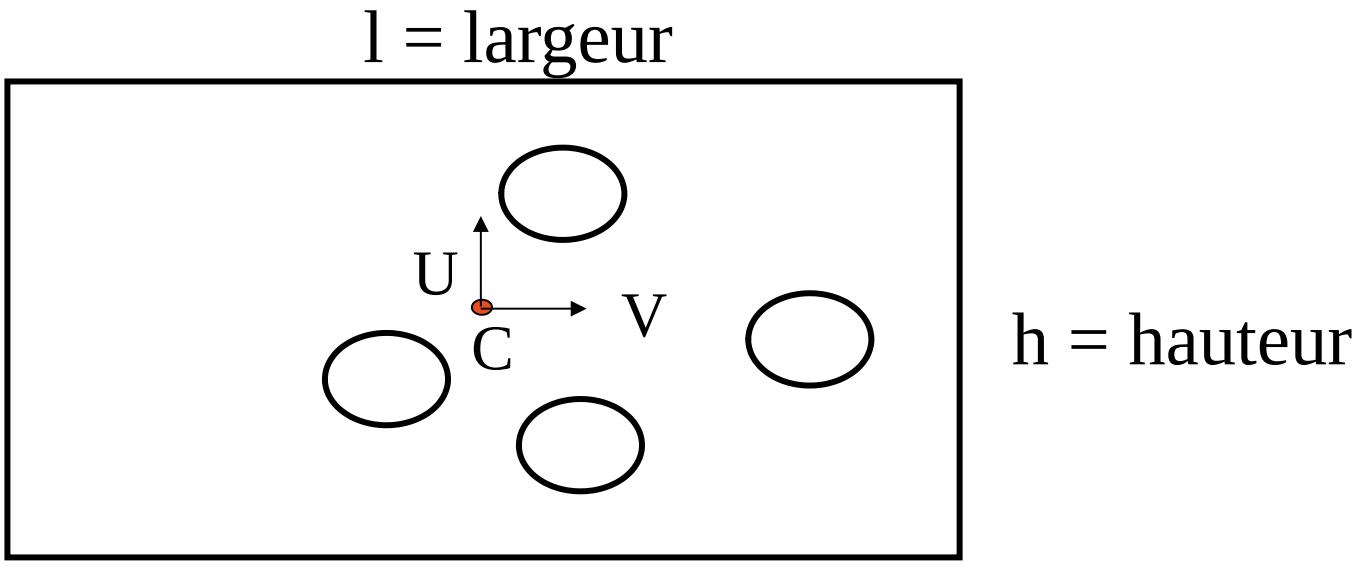
Caméra - section en coupe



CDP = Centre De Projection

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

Vue depuis la caméra

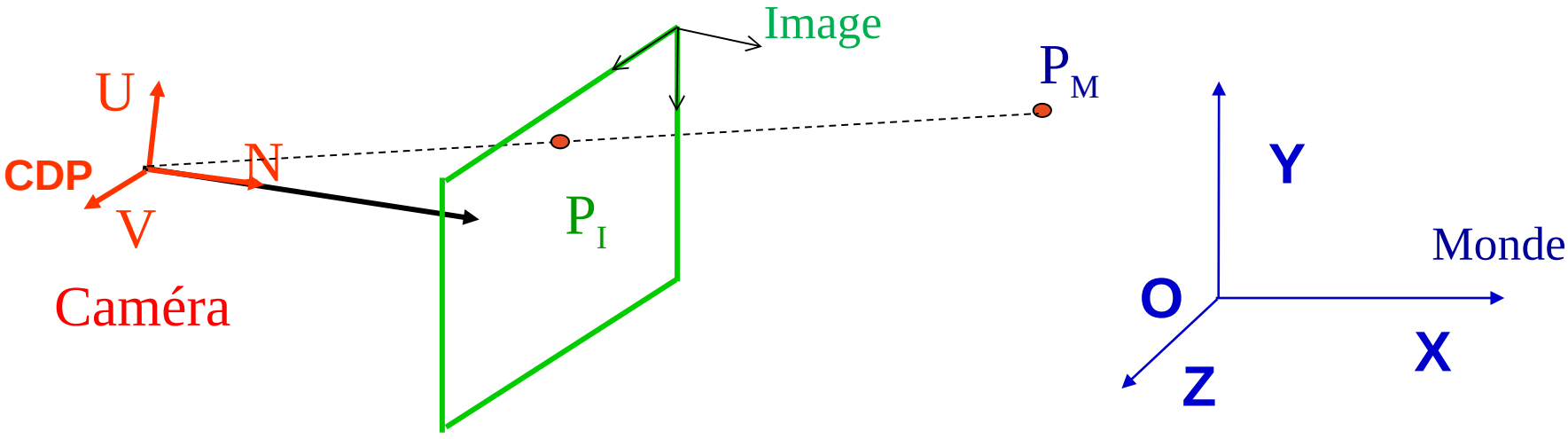


C = Centre de l'image = CDP + d * **N**

V = **N** x **U**

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



En résumé – Projection de la scène 3D vers l'Image

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

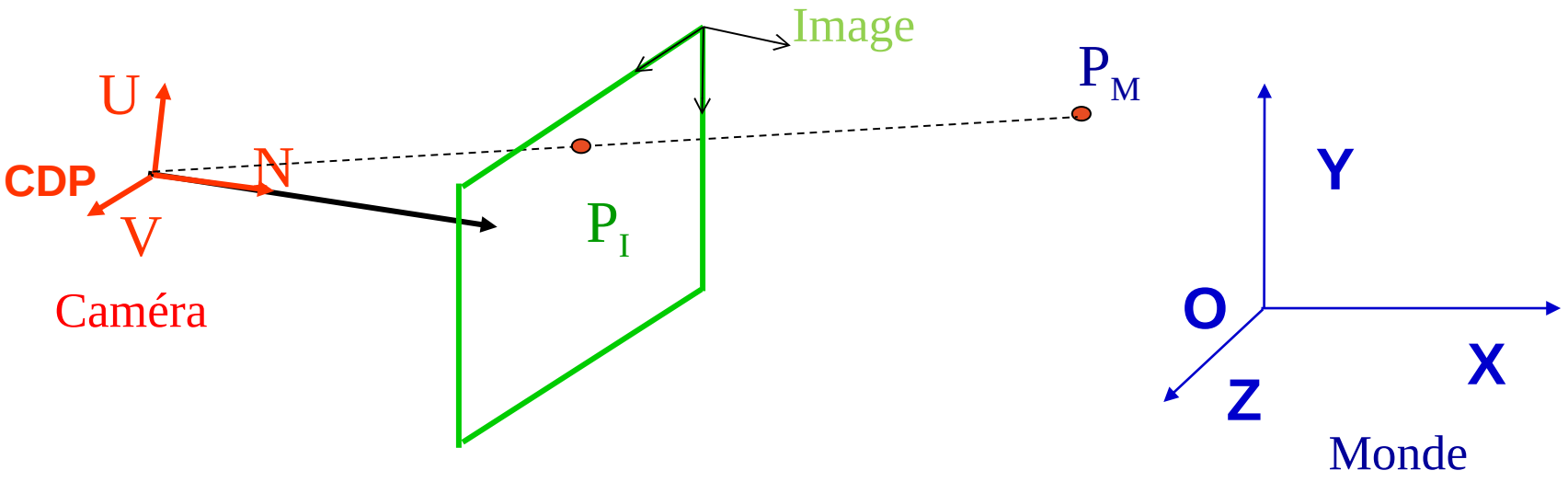
1. Tous les points de la scène 3D (repère M) sont ramenés dans le repère de la caméra (repère C)
2. Projection des points définis dans le repère de la caméra (repère C) vers le repère image (repère I)

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



En résumé – On se répète … 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

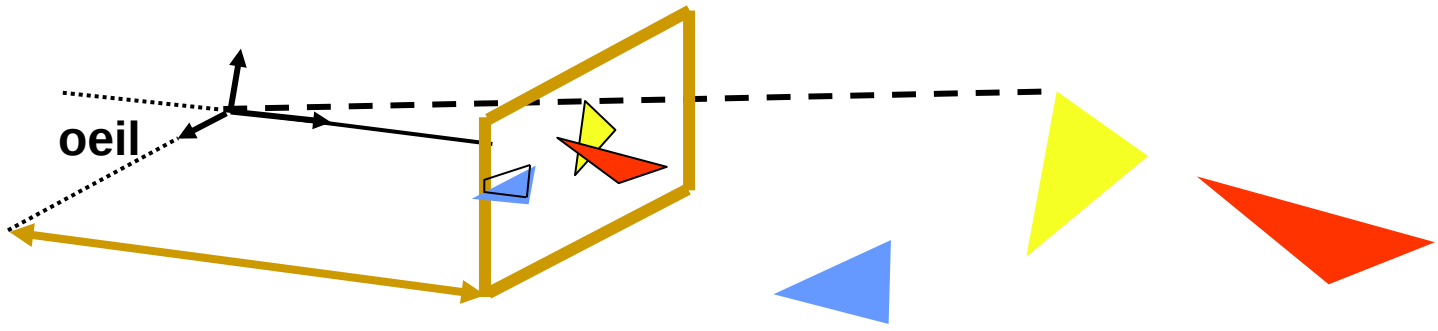
Matrices 4 x 4 pour passer d'un repère à l'autre :
- Repère création objet vers repère monde : matrice *model*
- Repère monde vers caméra : matrice *view*
- Repère caméra vers projectif : matrice *projection*
- Repère projectif vers repère image : matrice *viewport*

- $M_{C \leftarrow M}$ = GL_MODELVIEW
  - **gluLookAt**(Oeil, Direction, Haut)

- $M_{I \leftarrow C}$ = GL_PROJECTION
  - **gluPerspective**(*fovy*, *aspect*, *zNear*, *zFar*)

Pipeline graphique

- Pipeline graphique dans le cadre du rendu projectif
  - Clipping
  - Projection
  - Remplissage des polygônes
    - Suppression des parties cachées
    - Illumination

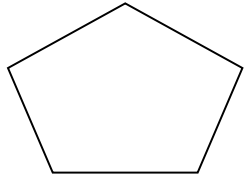


Quel type de polygone ?

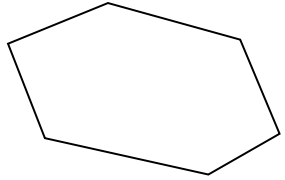
- Polygone simple
  - Aucune intersection d'arêtes

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

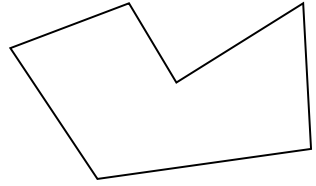
- Sinon polygone arbitraire ou complexe



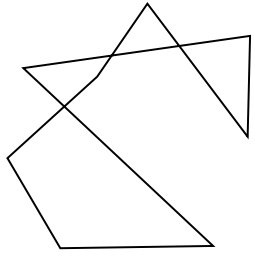
Simple



Convexe



Concave

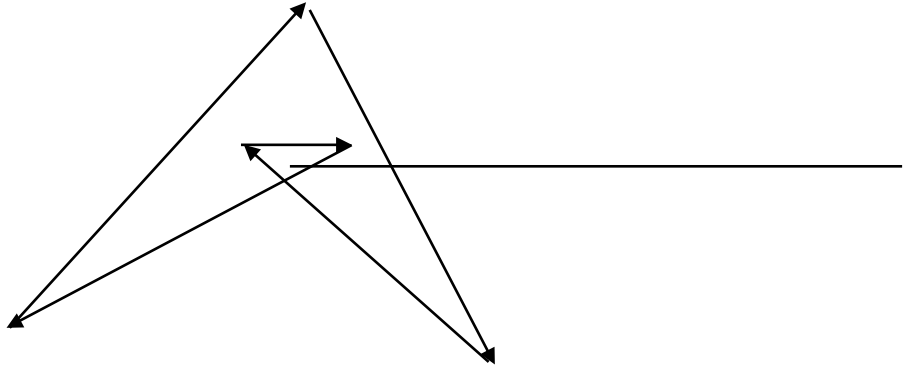


Arbitraire ou complexe

Intérieur d’un polygone ?

- Règle pair - impair
  - Pour chaque point, trace une ligne horizontal allant à l’infini
  - Compte le nombre d’intersection 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)



extérieur

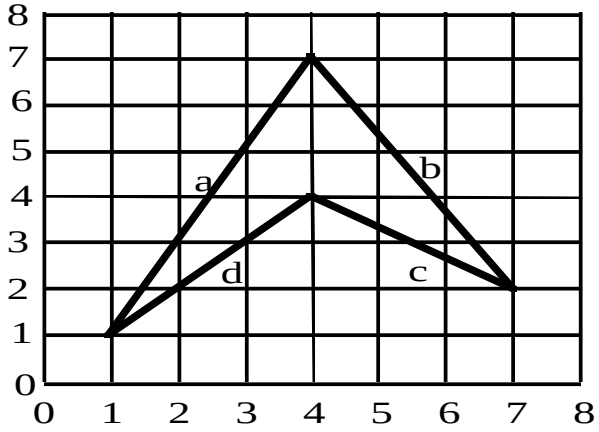
Remplir un polygone

- Comment remplir un polygone ?
  - Faire un test d’ "intérieur" pour chaque pixel de l’image
    - OK mais lent…
  - Pour chaque pixel de la boite englobante (bounding box) …
    - Toujours peu efficace

→ On utilise la cohérence

Remplir un polygone - 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	

Remplir un polygone - Algorithme de Scan-Line

Idée de l'algorithme :

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

Principe de l'algorithme :

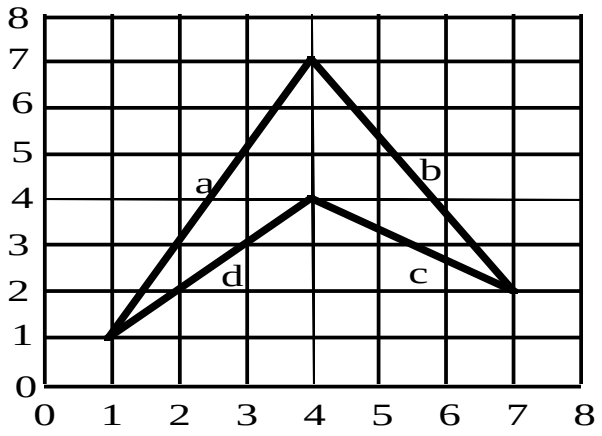
- Pour chaque ligne horizontale

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

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

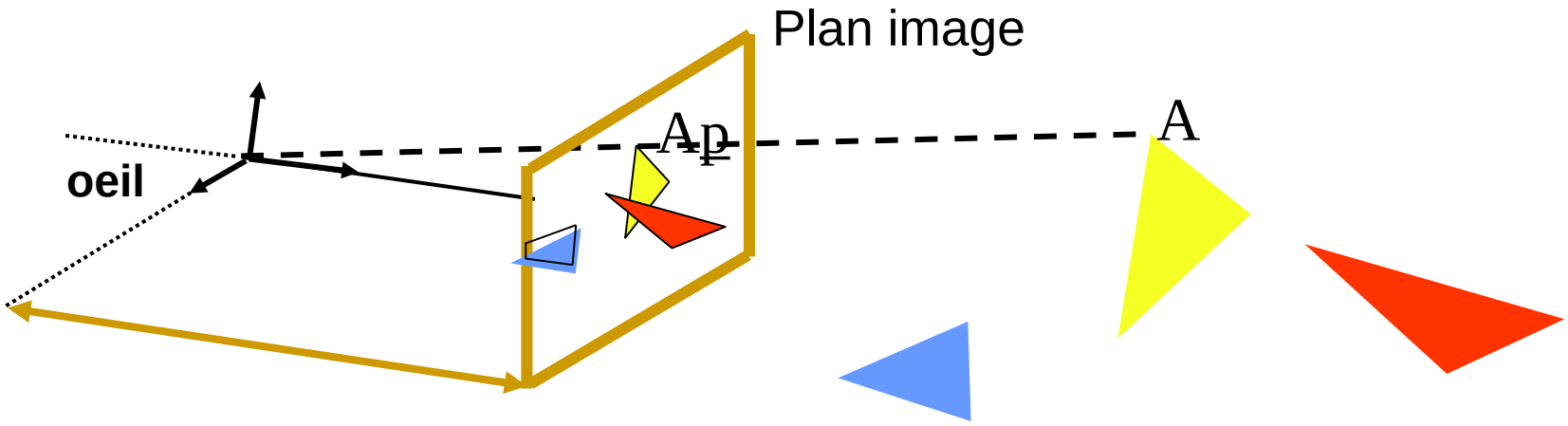
3.Affiche les pixels à l'intérieur du polygone entre les paires d'extrémités
   - Règle de la parité pour savoir si pixel à l'intérieur ou non

Remplir un polygone - Algorithme de Scan-Line



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	

Rendu projectif : pipeline graphique

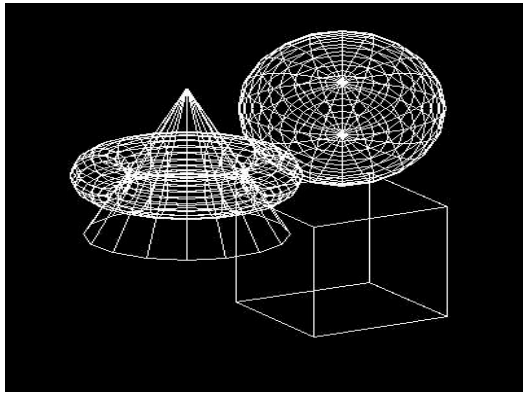


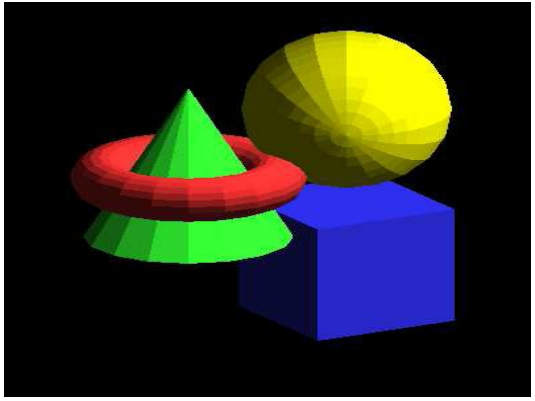
Pipeline graphique dans le cadre du rendu projectif :

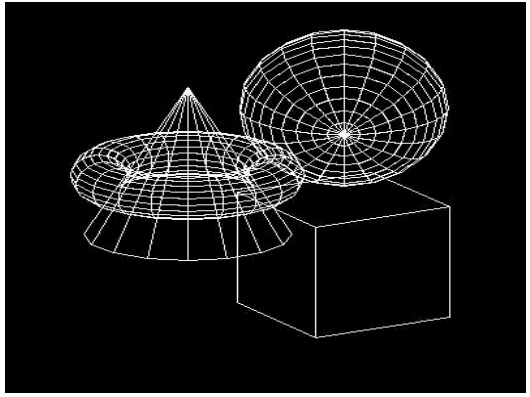
1. Clipping des polygones en 3D suivant la pyramide de vue
2. Projection des points sur le plan image
3. Remplissage des triangles (Rasterizing) dans l'image
   a. Suppression des parties cachées : Z-Buffer
   b. Calcul de la couleur : illumination

GPU

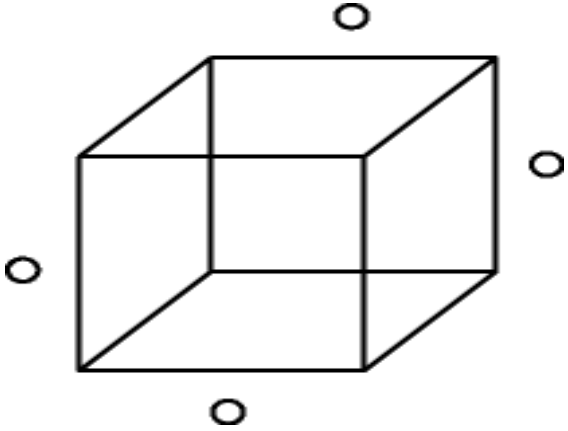
Élimination des parties cachées

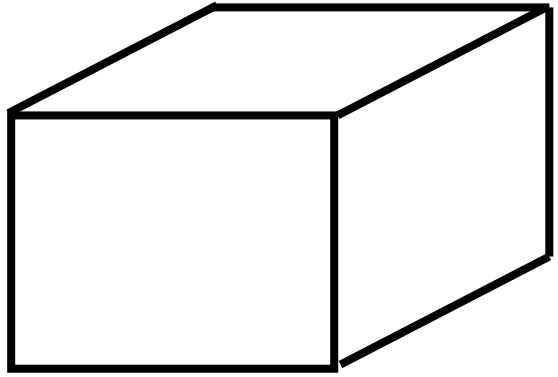


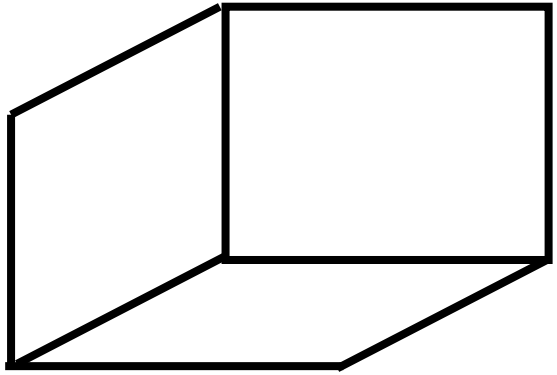




Suppression des parties cachées



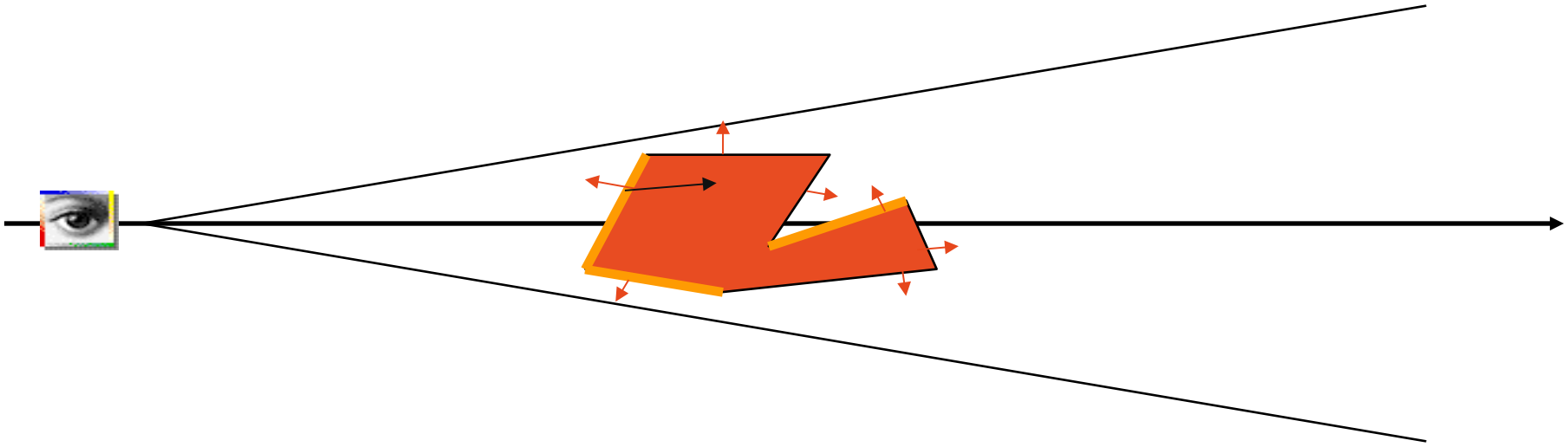




Problème : comment n’afficher que ce qui est visible ?

Back Face Culling

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



Back Face Culling : algorithme

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

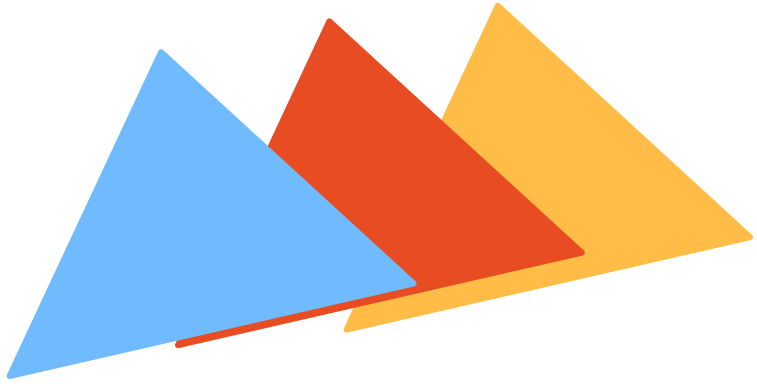
- Produit scalaire :
  - (Sommet - PointDeVue) . VecteurNormal
  - < 0 (angle obtus) : on garde le polygone
  - > 0 (angle aigu) : on l'élimine

Back Face Culling

- Économise 50 % du temps de calcul
  - En moyenne
- Faible coût par polygone
- Étape préliminaire pour les autres algorithmes
- Suffisant pour un seul objet convexe
- Pas suffisant pour plusieurs objets

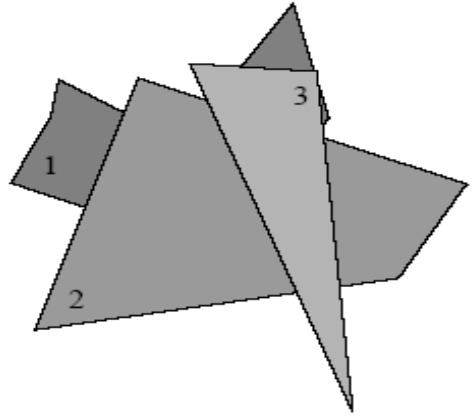
Suppression des parties cachées

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



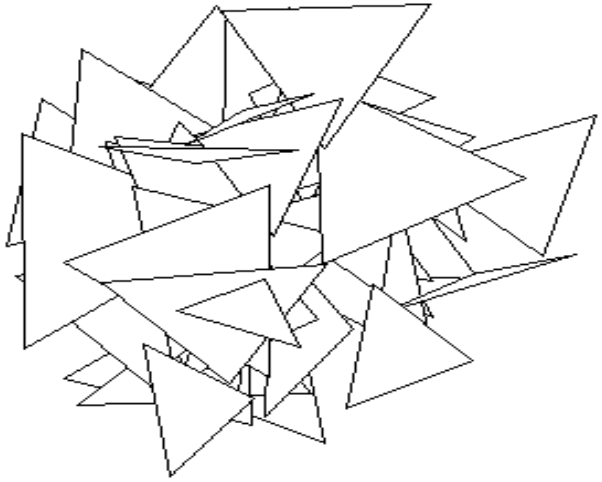
1ere idée : algorithme du peintre

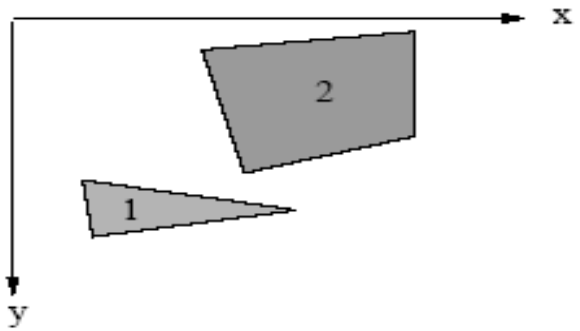
Peindre les facettes polygonales dans la mémoire vidéo suivant un ordre de distance décroissante au point d'observation.



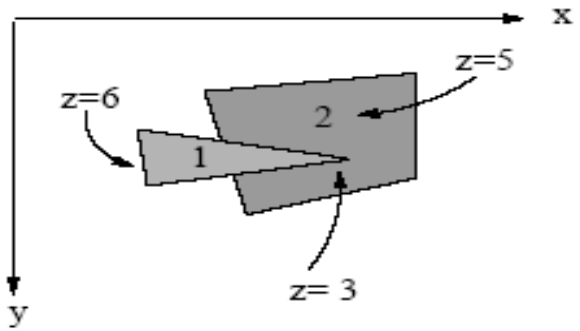
1ere idée : algorithme du peintre

Ambiguïtés

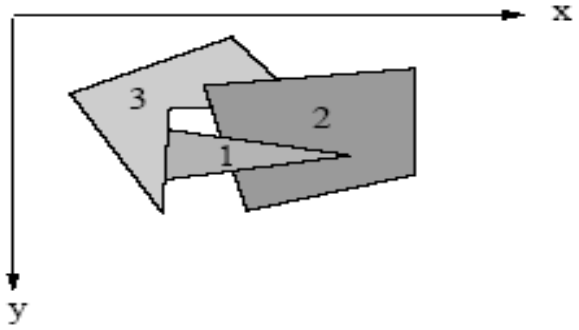




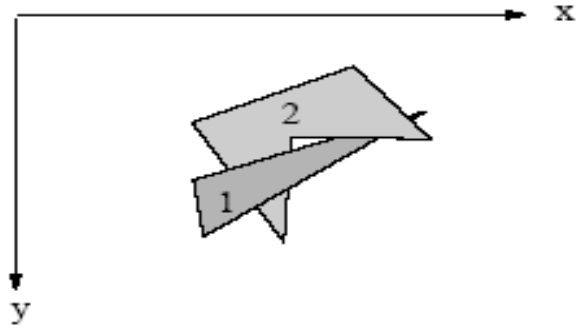
(a)



(b)



(c)



(d)

1ere idée : algorithme du peintre

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

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

3. Projeter les facettes et remplir les polygones suivant la liste.

Algorithme du peintre : pour ou contre

- Le plus intuitif des algorithmes
- Problème des ambiguïtés
- 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

Algorithme du Z-Buffer

- Un tableau, de la taille de l'écran
  - Stocke la valeur minimal de z pour chaque pixel
  - Initialisation : tous les pixels à l'infini
- Projection de tous les polygones
- On met à jour les pixels de la projection du polygone

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 c'est que le fragment du polygone est caché donc on ne fait RIEN

Z-Buffer

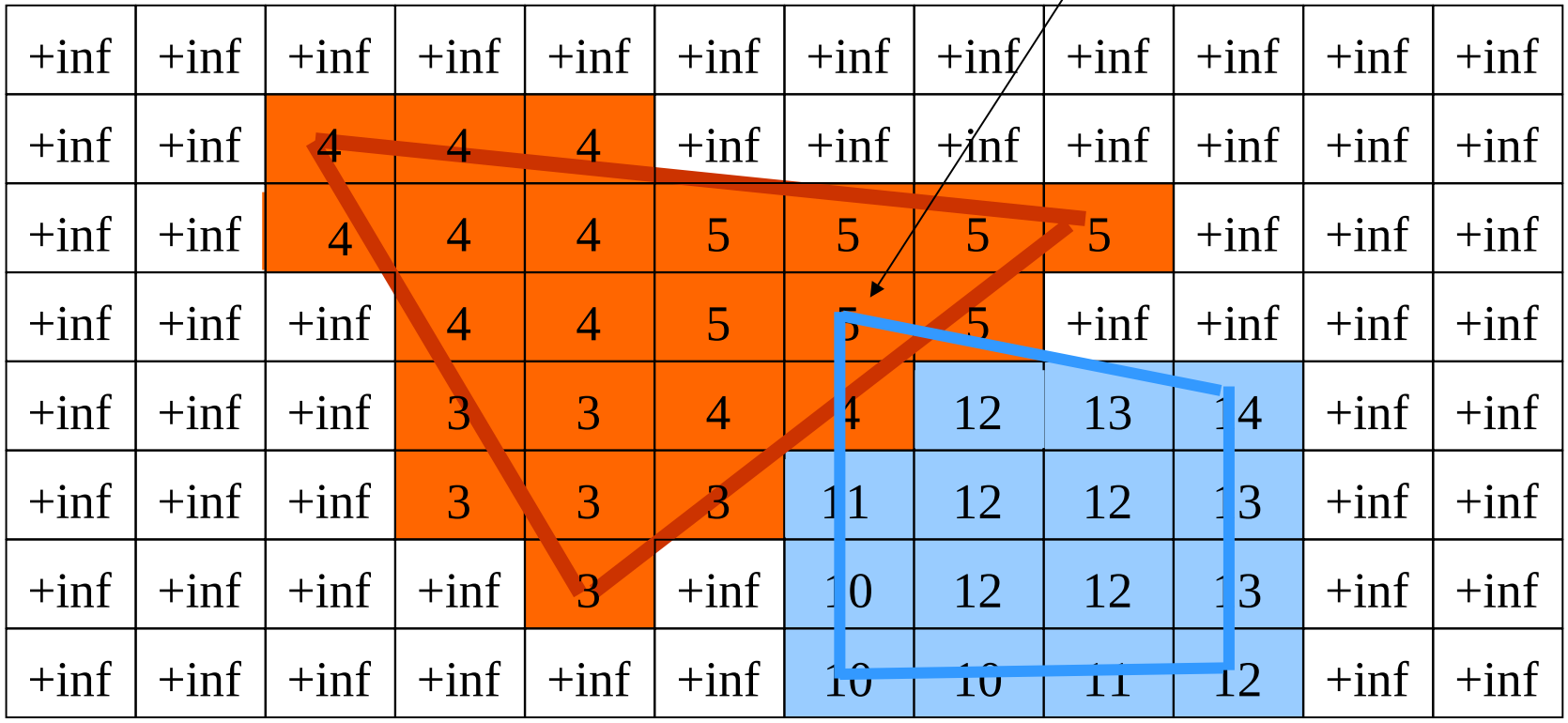
+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf
+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf
+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf
+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf
+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf
+inf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

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

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

Exemples

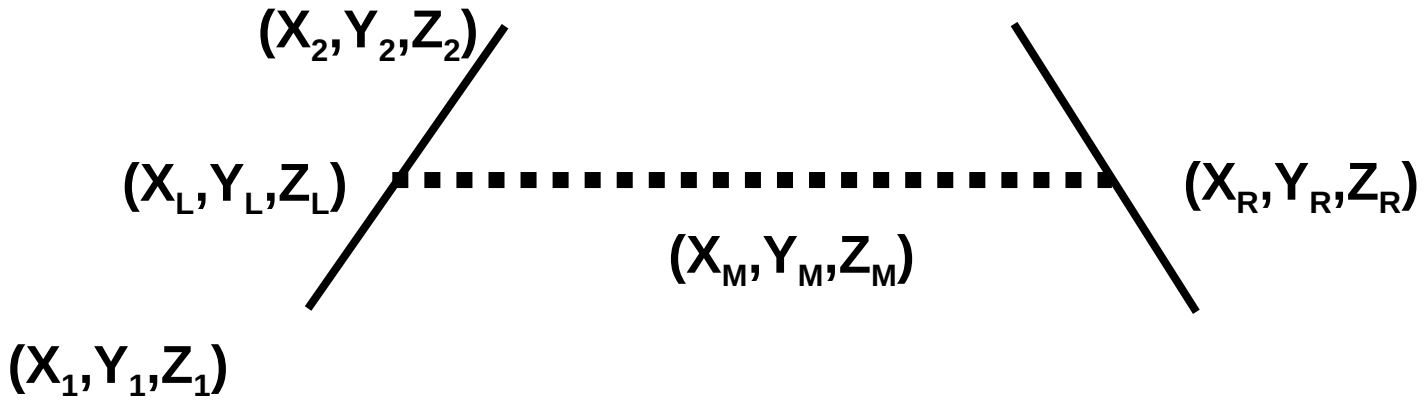
Calculer la valeur de *z* pour les pixels

- Comment calculer la valeur de Z pour chaque fragment du polygones en cours de remplissage ?
  - Depuis l'équation du plan
    ax+by+cz+d=0 => z=(d-ax-by)/c
  - Coûteux!!
    Il faut faire mieux …

    interpolation?

Interpoler la profondeur

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



Interpoler la profondeur

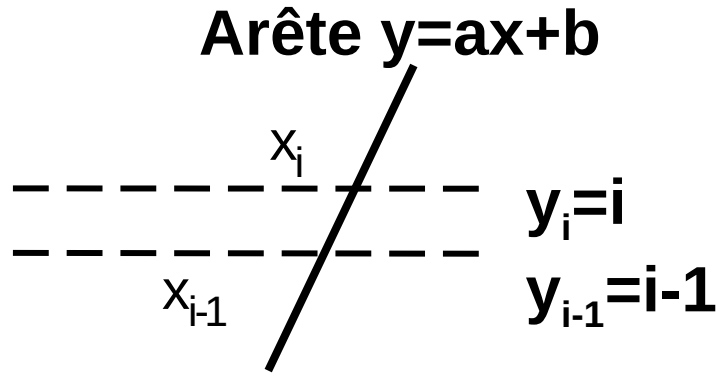
- Quelques rappels sur **l'interpolation linéaire**
  - Equation d'une droite : y =ax+b → donne équation pour une arête
  - Considère deux lignes horizontales qui intersectent l'arête : i et i-1

- 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) = (yi – yi-1) / (xi – xi-1) = (i – (i-1)) / (xi – xi-1)
  1/a = xi – xi-1

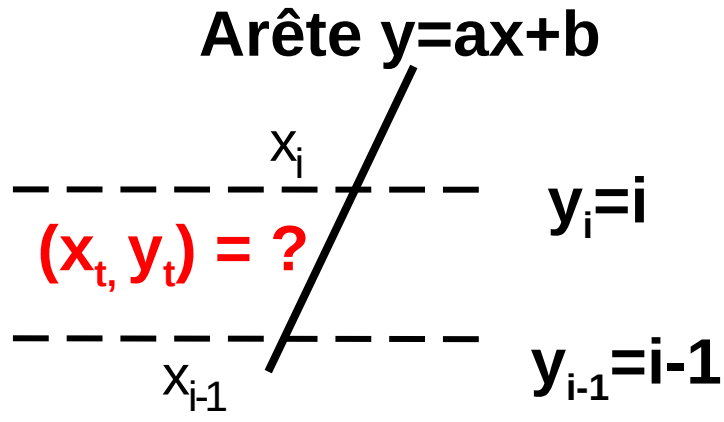
Interpoler la profondeur

- Pente de la droite aussi égale à :

  a = (variation de y) / (variation de x) = (yt – yi-1) / (xt – xi-1)

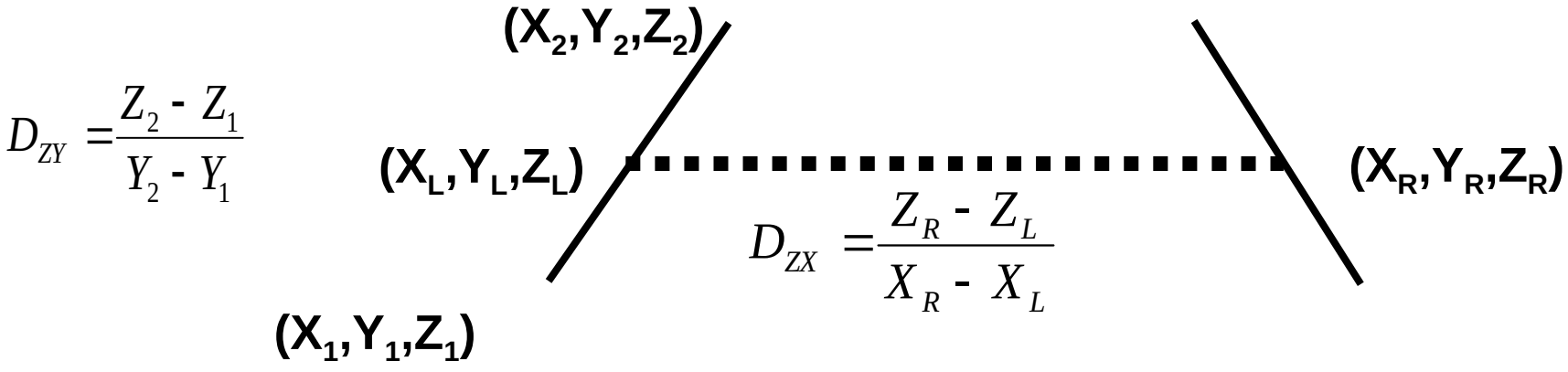
- Ainsi, valeur sur la droite donnée par :

  yt = a (xt – xi-1) + yi-1



Interpoler la profondeur

- Interpoler z le long de l'arête du polygone : ok !
- On fait cela sur les deux arêtes : deux points obtenus
- Puis interpoler z sur la ligne de remplissage en refaisant pareil interpolation bi-linéaire



Z-Buffer : pour ou contre

- Z-Buffer
  - Simple à implémenter
  - Coûteux en mémoire (plus un pb aujourd'hui)
  - Hardware

  - Problème de précision
    - Exemple:
    - 1 octet pour la profondeur
      → 256 niveaux de profondeur (seulement!!)
    - Near=1.0 mètre et Far=1000 mètres
    - 4 mètres entre 2 valeurs de profondeur

OpenGL : par exemple

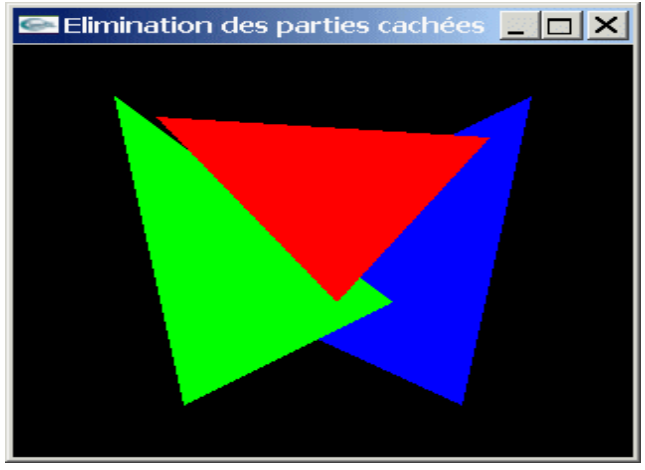
- Effacer le buffer et le Zbuffer entre chaque image

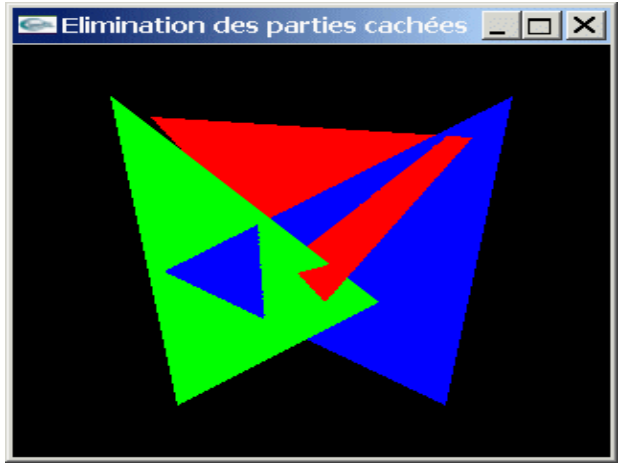
```
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
```

- Active le test des Z avec le Z-buffer

```
glEnable(GL_DEPTH_TEST) ;
```

non activé





activé

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 (Rasterizing) dans l'image
     a. Suppression des parties cachées : **Z-Buffer**
     b. Calcul de la couleur : **illumination**