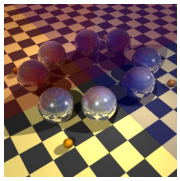



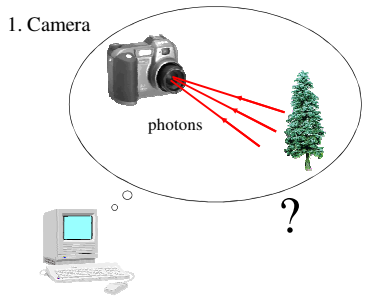

Rendu

1. Lancer de rayons

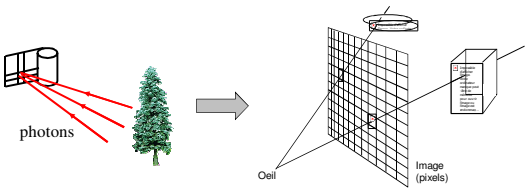
2. Pipeline du rendu projectif

Alexandre Meyer
http://licence-info.univ-lyon1.fr/LIO41

Synthèse d'images



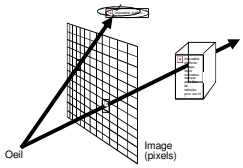
Analogie appareil photo et SI



Appareil photos

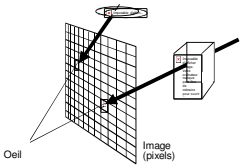
Synthèse d'images

2 approches duales en SI



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

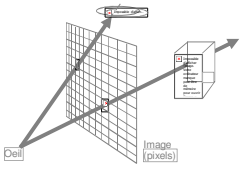
Ray-tracing
- Image réaliste
- Lent



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

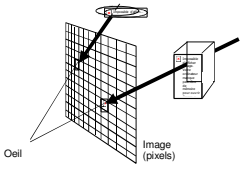
Rendu projectif (cablé sur les cartes graphiques modernes -> temps réel)

2e approches : rendu projectif



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

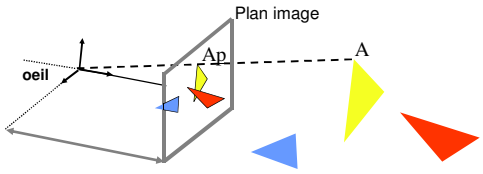
Ray-tracing
- Image réaliste
- Lent



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

Rendu projectif (cablé sur les cartes graphiques modernes -> temps réel)

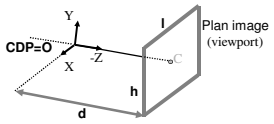
Rendu projectif : PIPELINE



Pipeline
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

Une caméra de base

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

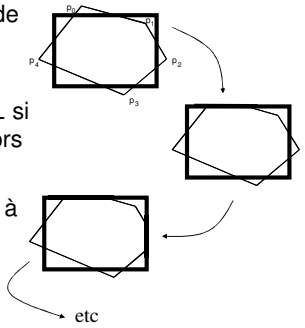


Clipping

- 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 ou en 3D
  - Clipping avant (3D) ou après la projection (2D)?

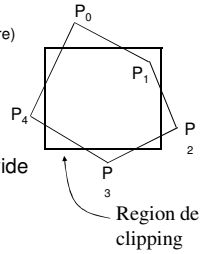
Algorithme de Sutherland-Hodgman

- 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



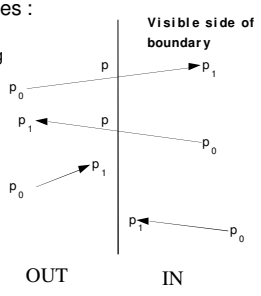
Clipping suivant une région

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



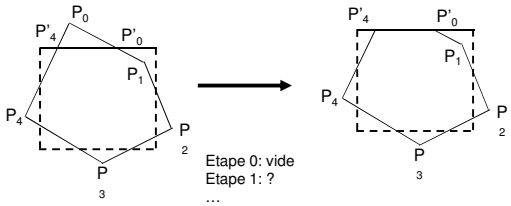
Clipping d'une arête

- 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:

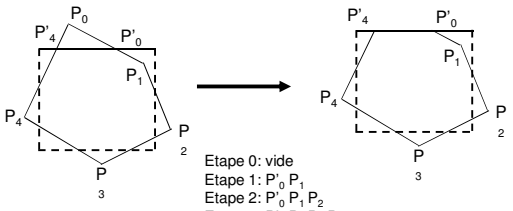


Etape 0: vide
Etape 1: ?
…

Exercice

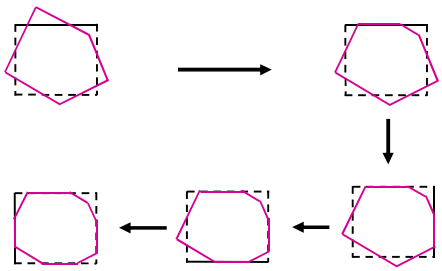
Sutherland-Hodgman

- Un exemple:



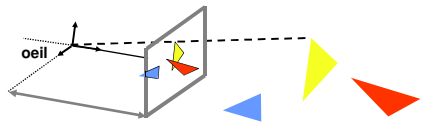
Etape 0: vide
Etape 1: $P'_0 P_1$
Etape 2: $P'_0 P_1 P_2$
Etape 3: $P'_0 P_1 P_2 P_3$
Etape 3: $P'_0 P_1 P_2 P_3 P_4$
Etape 4: $P'_0 P_1 P_2 P_3 P_4 P'_4$

Sutherland-Hodgman



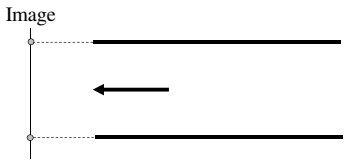
Plan

- Rendu par lancé de rayons (ray-Tracing)
- Rendu projectif
  - Clipping
  - Projection
  - Remplissage des polygones
    - Suppression des parties cachées
    - Illumination

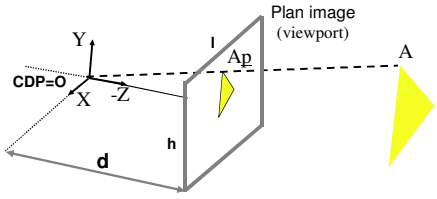


Projection parallèle

- Direction De Projection (DDP) est (0,0,1)
- P=(x,y,z) -> P' = (x, y, 0)
- Manque de réalisme



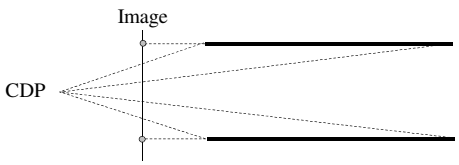
Projection perspective



1. Projette les points du triangle sur le plan image

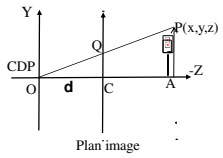
Projection perspective

- Projete vers le centre de projection CDP
- P=(x,y,z) -> P' = ??



Projection perspective

- Besoin de perspective
- Configuration simple :



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

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

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

d=distance focale

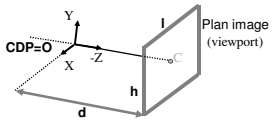
Matrice de projection : $M_{I\leftarrow C}$

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

- Soit un point dans l'espace de la camera
  $(x,y,z,1)$
- Résultat : un point dans l'espace Image
  $$(x,y,z,z/d)=\left(\frac{xd}{z},\frac{yd}{z},d,1\right)$$

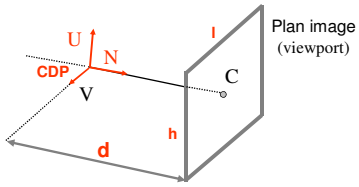
Espace Camera != Espace Monde

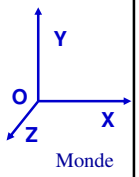
- Jusqu'à présent on avait
  - CDP = Origine O
  - La caméra regarde vers –Z
  - Le haut de la caméra est Y



On aimerait pouvoir placer la camera n'importe où!

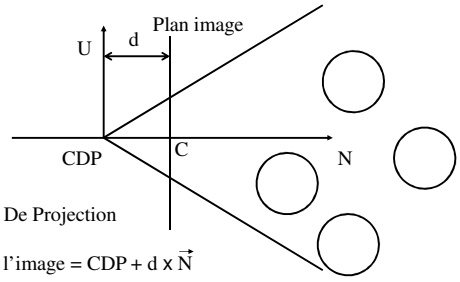
Caméra plus générale





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 camera
h = hauteur de l'image dans l'espace camera
mxn = taille en pixel de l'image

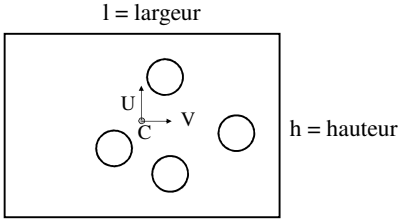
Caméra (Section en coupe)



CDP = Centre De Projection

C = centre de l'image = CDP + d x $\vec{N}$

Vue depuis la camera

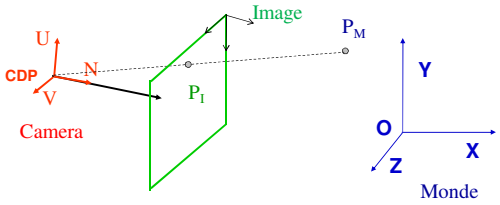


C = Centre de l'image = CDP + d x $\vec{N}$

$\vec{V}=\vec{N}\times\vec{U}$

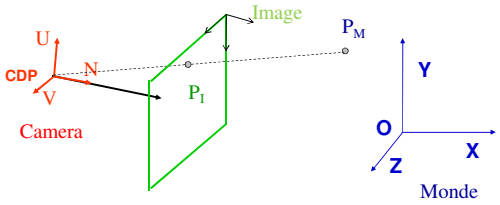
Différent espaces

- Espace Monde (M)
  - Les points sont décrits dans l'espace monde
- Espace Camera (C)
  - Dans cette espace, CDP=O, N=-Z, U=Y, V=X
- Espace Image (I)
  - Dans cette espace, le pt (i,j,z) correspond au pixel (i,j) de l'image.



Différent espaces

- Espaces : Monde (M), Camera (C), Image (I)
- Besoin des matrices
  - $M_{C \leftarrow M}$ faisant passer un point du repère M au repère C
  - $M_{I \leftarrow C}$ faisant passer un point du repère C au repère I (voir transparent 10 pour avoir $M_{C \leftarrow I}$ qu'on inverse)

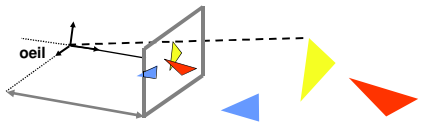


OpenGL

- $M_{C \leftarrow M}$ = GL_MODELVIEW
  - **gluLookAt**(Oeil, Direction, Haut)
- $M_{I \leftarrow C}$ = GL_PROJECTION
  - **gluPerspective**(*fovy*, *aspect*, *zNear*, *zFar*)

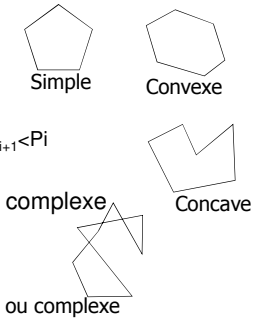
Plan

- Rendu par lancé de rayons (ray-Tracing)
- Rendu projectif
  - Clipping
  - Projection
  - Remplissage des polygones
    - Suppression des parties cachées
    - Illumination



Simple, Convexe et complexe

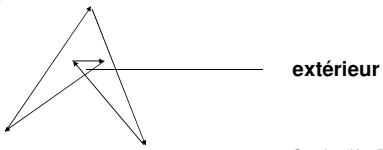
- Simple
  - Aucune intersection d'arêtes
- Convexe
  - Chaque angle intérieur $P_{i-1}P_iP_{i+1}<Pi$
- Sinon polygone arbitraire ou complexe



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)

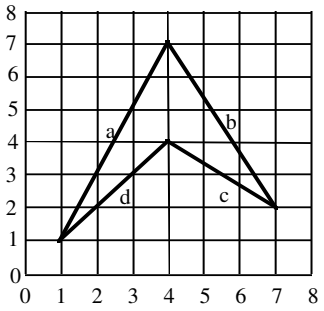


Remplir un polygone

- 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

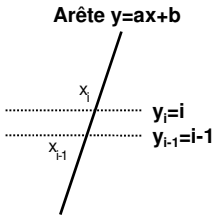
Algorithme de Scan-Line



Algorithme de Scan-Line

- L'intersection entre l'arête d'un polygone et deux lignes horizontale change peu et est calculable facilement

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



Active Edge Table (AET)

- Pour chaque ligne horizontale d'un polygone on a besoin de considérer que certaines arêtes
- Garde une table des arêtes actives = Active Edge Table
  - Mise à jour de cette table à chaque changement de ligne
- On extrait de cette table les intervalles à colorier

Initialisation : Edge Table (ET)

- Ordre de description d'une arête
  - Être sur que y1<y2 pour chaque arête (x1,y1) (x2,y2)
- Construire une table des arêtes Edge Table (ET)
  - Tri des arêtes suivant leur Y minimum
  - Des arêtes commençant avec le même Y sont rangées dans la même "boite"
  - Chaque élément contient
    - (Ymax, Xmin, incrément des X)

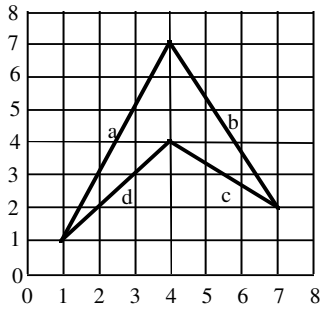
Maintenir l'AET

- Passage à la ligne horizontale suivante
  - Enlève chaque arête dont y2 = ligne courante
  - Mettre à jour la valeur de x pour chaque arêtes restantes
  - Ajoute chaque arête dont y1 est égal à la ligne courante

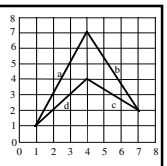
Dessiner à partir de l'AET

- Trier les arêtes actives selon x
- Les pairs d'arêtes donnent les segments à dessiner
- Attention
  - Les sommets Maximum ne sont pas dessinés
  - Cas particulier quand un polygones partages des arêtes avec un polygone voisin

Exemple



Initialisation



- Arêtes (Edges)

(Ymax, Xmin, incrément des X)

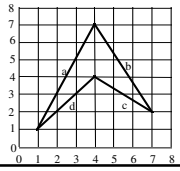
Label	Coordonnées	y1	Structure
a	(1,1) to (4,7)	1	(7,1,0.5)
b	(7,2) to (4,7)	2	(7,7,-0.6)
c	(7,2) to (4,4)	2	(4,7,-1.5)
d	(1,1) to (4,4)	1	(4,1,1)

- Table des arêtes (Edge Table) contient

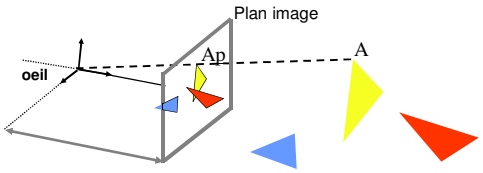
y1	Sequence d'arêtes
1	(7,1,0.5), (4, 1, 1)
2	(7,7,-0.6), (4, 7,-1.5)

Pour chaque ligne de l'AET

Ligne	Active Edge Table	Interval de dessin
0	vide	
1	(7, 1, 0.5), (4, 1, 1)	1 à 1
2	(7, 1.5, 0.5), (4, 2, 1), (7, 7, -0.6), (4, 7, -1.5)	1.5 à 2, 7 à 7
3	(7, 2.0, 0.5), (4, 3, 1), (4, 5.5, -1.5), (7, 6.4, -0.6)	2.0 à 3, 5.5 à 6.4
4	(7, 2.5, 0.5), (7, 5.8, -0.6)	2.5 à 5.8
5	(7,3. 0,0. 5), (7, 5.2, -0.6)	3.0 à 5.2
6	(7, 3.5, 0.5), (7, 4.6, -0.6)	3.5 à 4.6
7	vide	
8	vide	

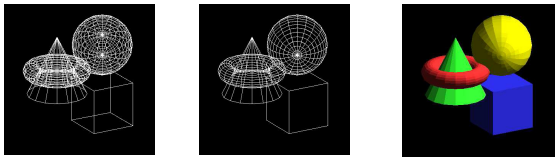


Rendu projectif : PIPELINE



Pipeline
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



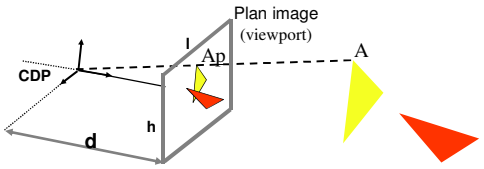
Élimination des parties cachées

Synthèse d'images

©Alexandre Meyer 2002-2005 Anthony Steed 1999

42

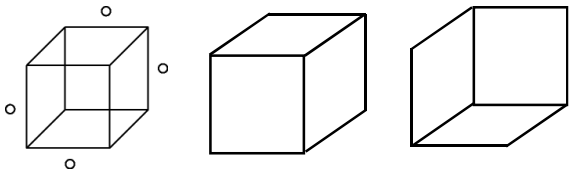
Rendu projectif



1. Projection des points sur le plan image
2. Clipping
3. Remplissage des triangles (rasterisation) dans l'image
4. Suppression des parties cachées
5. Calcul de la couleur : illumination

Problème des parties cachées

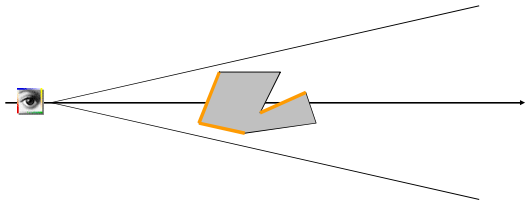
- Problème



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)*normale
  - < 0 : on garde le polygone
  - > 0 : 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

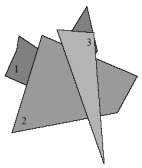
Problème des parties cachées

- Problème

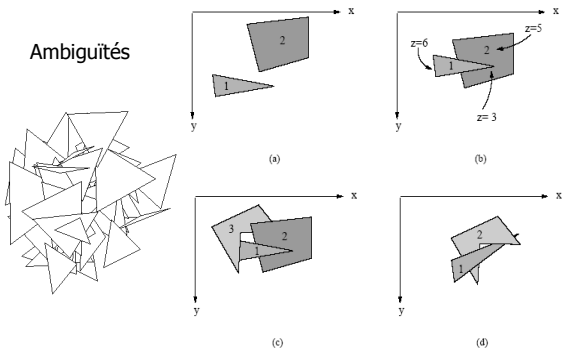
Comment n'afficher que ce qui est visible?

1ere idée : algo du peintre

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



1ere idée : algo du peintre



1ere idée : algo du peintre

- Algo
  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.

Algo 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

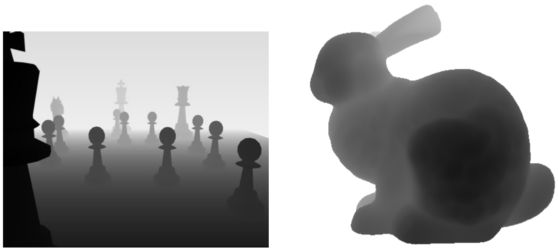
+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf	+inf
+inf	+inf	4	4	4	+inf	+inf	+inf	+inf	+inf	+inf	+inf
+inf	+inf	4	4	4	5	5	5	5	+inf	+inf	+inf
+inf	+inf	+inf	4	4	5	5	5	+inf	+inf	+inf	+inf
+inf	+inf	+inf	3	3	4	4	+inf	+inf	+inf	+inf	+inf
+inf	+inf	+inf	3	3	3	+inf	+inf	+inf	+inf	+inf	+inf
+inf	+inf	+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
- Puis interpoler z sur la ligne de remplissage

=> interpolation bi-linéaire

$$D_{ZY} = \frac{Z_2 - Z_1}{Y_2 - Y_1}$$

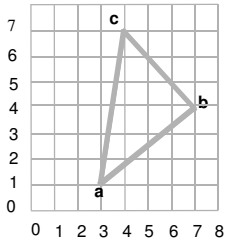
(X_2,Y_2,Z_2)

(X_L,Y_L,Z_L) .... (X_R,Y_R,Z_R)

$$D_{ZX} = \frac{Z_R - Z_L}{X_R - X_L}$$

(X_1,Y_1,Z_1)

Zbuffer : exemple de remplissage



- Table des arêtes (Edge Table=ET)
  - (y2, x1, dx/dy, z1, dz/dy)
- ET[1] =
  - ac (7, 3, 1/6, 1, 3/6)
  - ab (4, 3, 4/3, 1, 1/3)
- ET[4] =
  - cb (7, 7, -1, 2, 2/3)

a=(3,1,1) b=(7,4,2) c=(4,7,4)

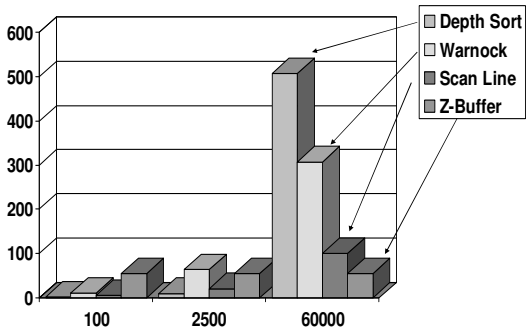
Contenu de la table des arêtes actives (AET)

- Ligne y=1
  - ac (7, 3, 1/6, 1, 3/6)
  - ab (4, 3, 4/3, 1, 1/3) pas des z : de 1 en 1
- y=2
  - ac (7, 3.166, 1/6, 1.5, 3/6)
  - ab (4, 4.333, 4/3, 1.333, 1/3) pas des z : de1.5 en 1.333
- y3
  - ac (7, 3.333, 1/6, 2.0, 3/6)
  - ab (4, 5.666, 4/3, 1.666, 1/3) pas des z : de 2 en 1.666

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

Coûts comparés

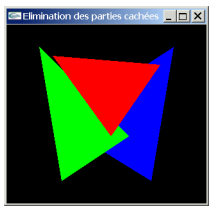


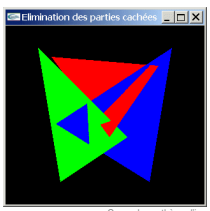
OpenGL : par exemple

- 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 z-buffer 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

Pipeline et carte graphique

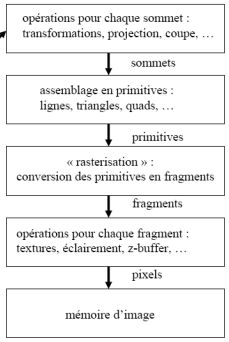
Cablé sur les cartes graphiques modernes

GPU pour Graphic Process Unit
→ Temps réel

Pipeline et shader

pipeline classique :

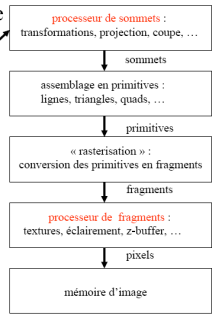
ensemble de sommets fournis par l'application



Pipeline et shader

pipeline programmable

ensemble de sommets fournis par l'application

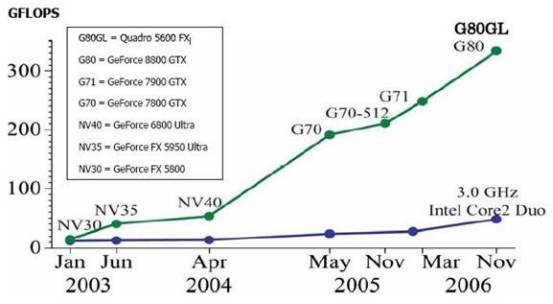


Vertex program

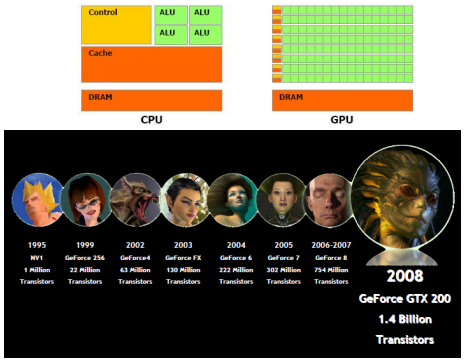
Fragment program

Shader = programme utilisateur exécuté dans le pipeline graphique lors d'étapes spécifiques

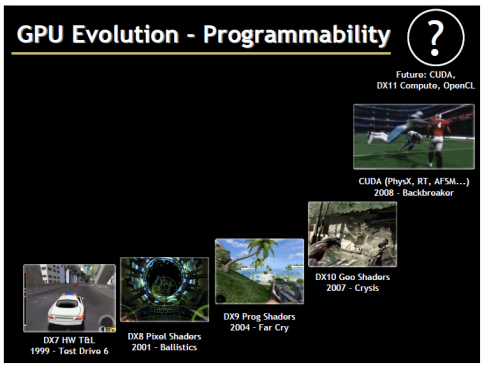
Evolutions des GPU



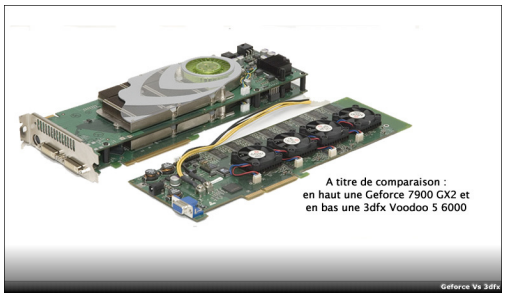
Evolutions des GPU



Evolutions des GPU



Evolutions des GPU



Fin

Evolutions du GPU (cf. pdf)

PC Graphics growth (225%/yr)

Season	tech	#trans	Gflop*	Mpix	Mpoly	Mvector
...						
spring/00	0.18	25M	35	800	30M	50M
fall/00	0.15	50M	150	1.2G	75M	100M
spring/01	0.15	55M	180	2.0G	100M	200M
fall/01	0.13	100M	280	4.0G	200M	400M
...						
spring/03 (NV30)	0.13	140M	500	8.0G**	300M	300M
spring/04 (NV40)	0.13	220M	1000	25.6G**	600M	600M

* Special purpose math, not all general purpose programmable math
** Samples (multiple color values within a pixel, for smooth edges)

Evolution_GPU.pdf pour les images

Evolutions du GPU