Le pipeline graphique (suite)

Ressource : www.scratchapixel.com
http://www.songho.ca/opengl/gl_projectionmatrix.html

Le pipeline graphique

Modèle géométrique : objets, surfaces, sources de lumière…

Modèle d'illumination : calcul des interactions lumineuses

Caméra : point de vue et ouverture (*frustum*)

Fenêtre (*viewport*) : grille de pixel

Modeling
Transformations

Viewing Transformation
(Perspective / Orthographic)

Clipping

Viewport transform

Scan Conversion
(Rasterization)

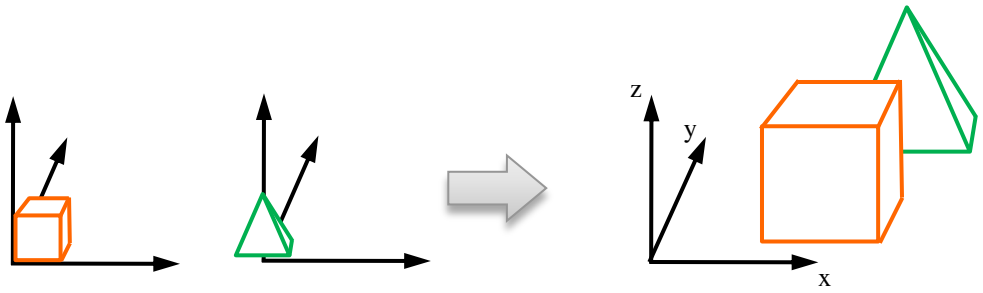
Illumination
(Shading)

Per fragment operations
(visibility, blending)

Couleurs, intensités convenant à l'afficheur (ex : 32 bits, RGBA)

Transformations objet

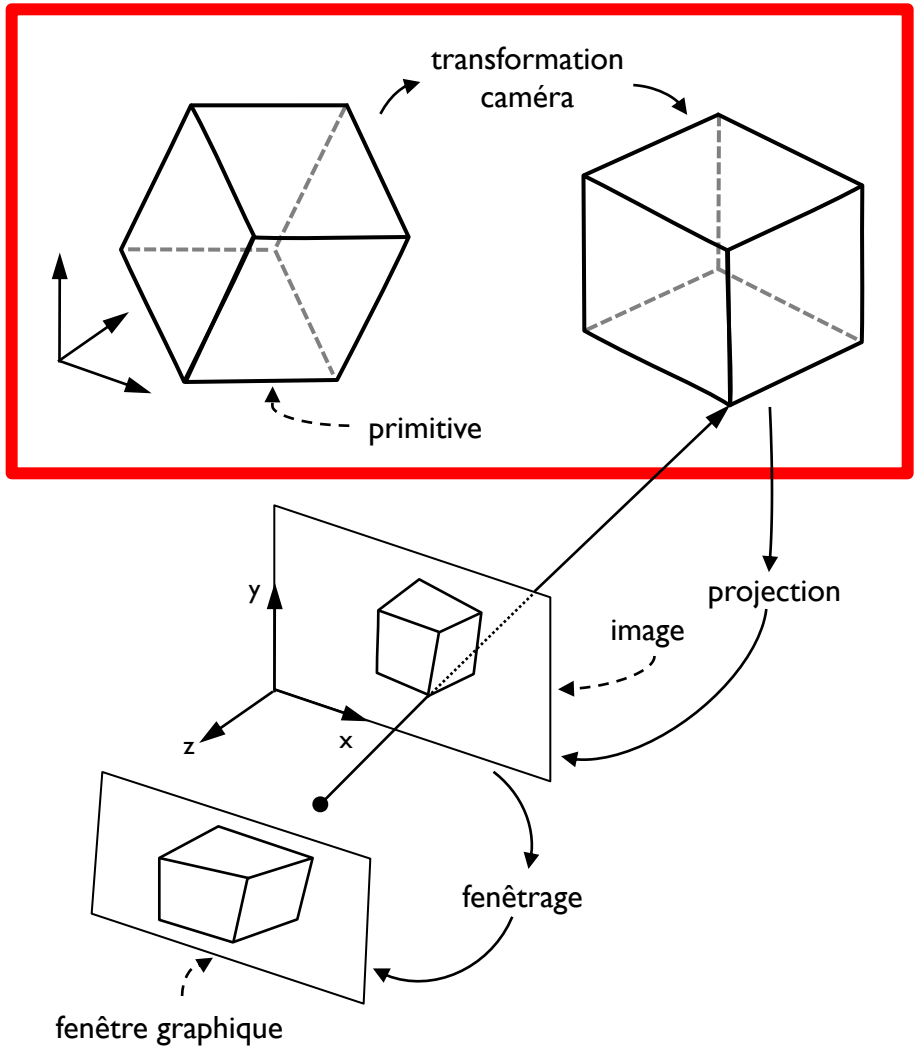
Passage du système de **coordonnées locales** de chaque objet 3D (*object space*) vers un **repère global** (*world space*)



Object space *World space*

- Modeling Transformations
- Viewing Transformation (Perspective / Orthographic)
- Clipping
- Viewport transform
- Scan Conversion (Rasterization)
- Illumination (Shading)
- Per fragment operations (visibility, blending)

Transformation caméra



- Modeling Transformations
- Viewing Transformation (Perspective / Orthographic)
- Clipping
- Viewport transform
- Scan Conversion (Rasterization)
- Illumination (Shading)
- Per fragment operations (visibility, blending)

« Look at »

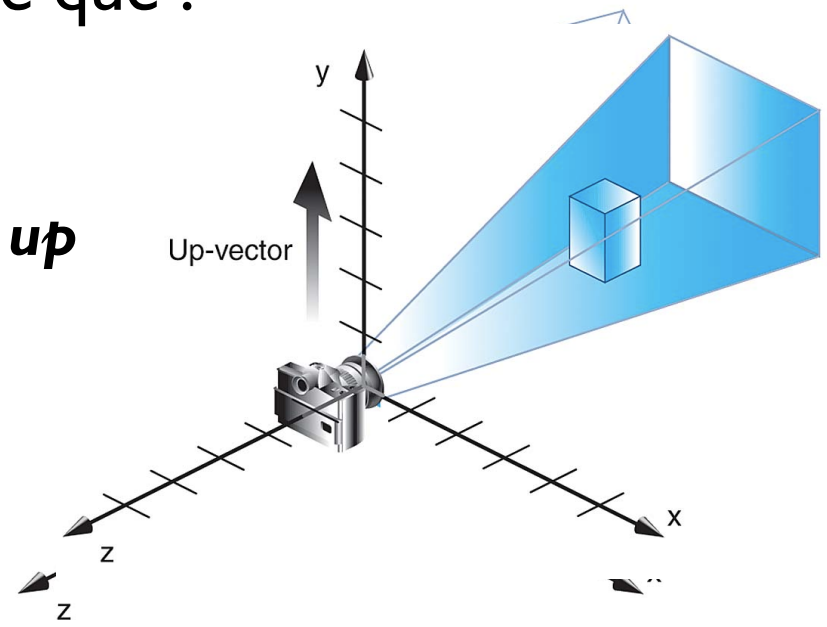
Entrées (en espace monde)

- ***eye*** : position de la caméra
- ***at*** : position du point de référence visé
- ***up*** : direction indiquant le « haut »

Sortie

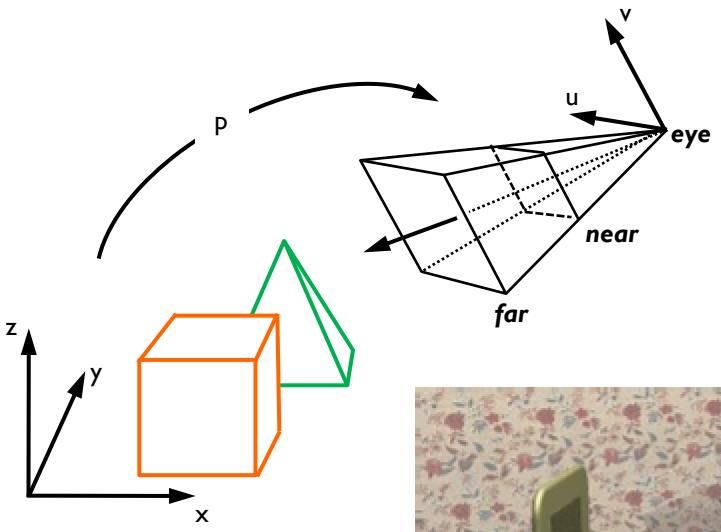
matrice de transformation telle que :

- *eye* soit à l'origine du repère
- l'axe -**z** du repère pointe vers ***at***
- l'axe *y* du repère soit aligné avec ***up***



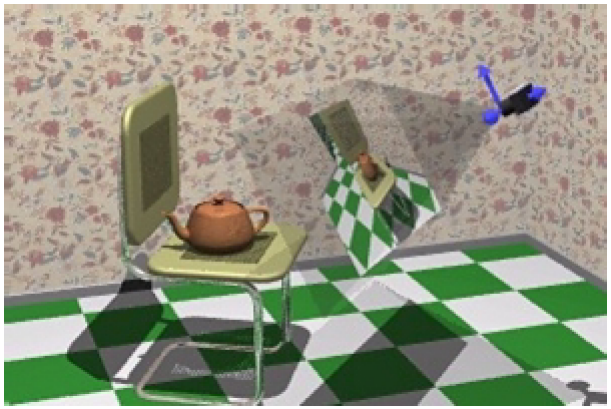
Transformation caméra

Passage des **coordonnées du monde** à celles de **la caméra** (*eye space*).



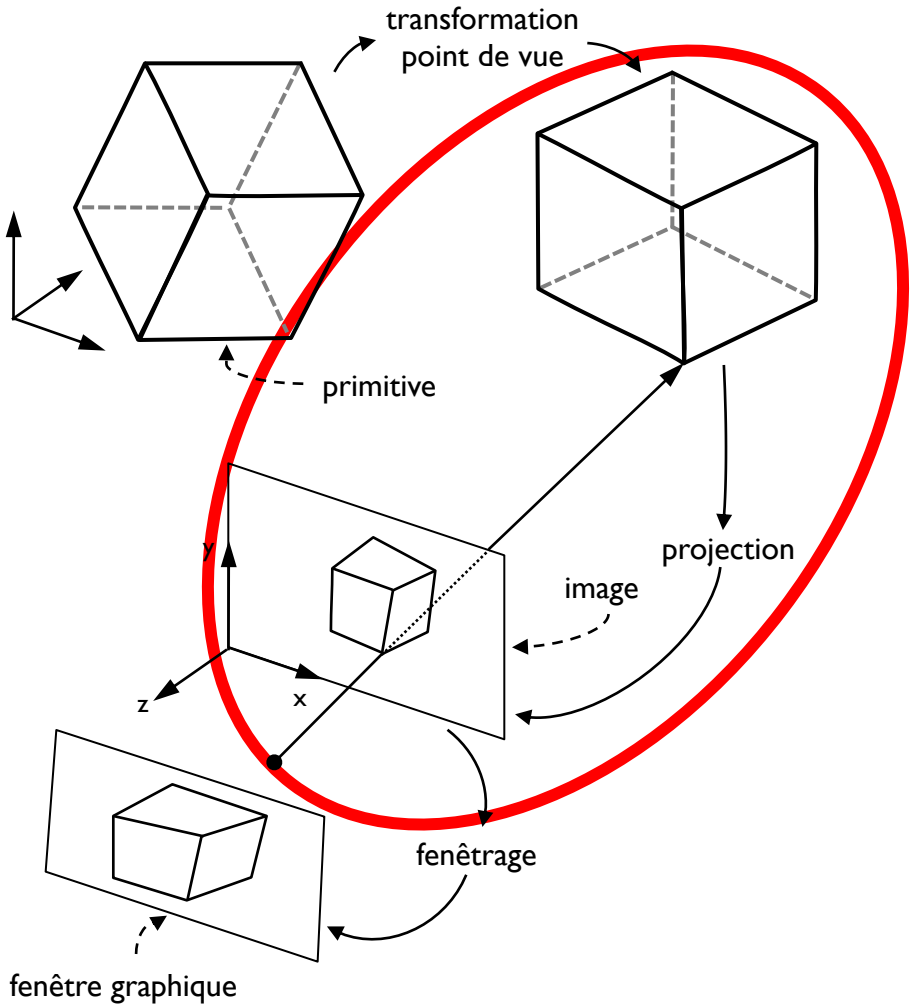
Eye space

World space



- Modeling Transformations
- **Viewing Transformation (Perspective / Orthographic)**
- Clipping
- Viewport transform
- Scan Conversion (Rasterization)
- Illumination (Shading)
- Per fragment operations (visibility, blending)

Projections

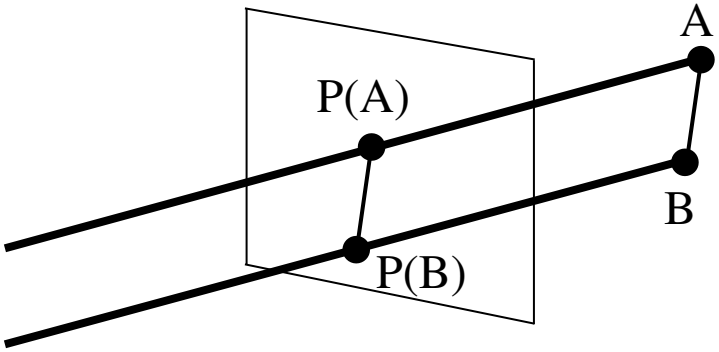


- Modeling Transformations
- Viewing Transformation (Perspective / Orthographic)
- Clipping
- Viewport transform
- Scan Conversion (Rasterization)
- Illumination (Shading)
- Per fragment operations (visibility, blending)

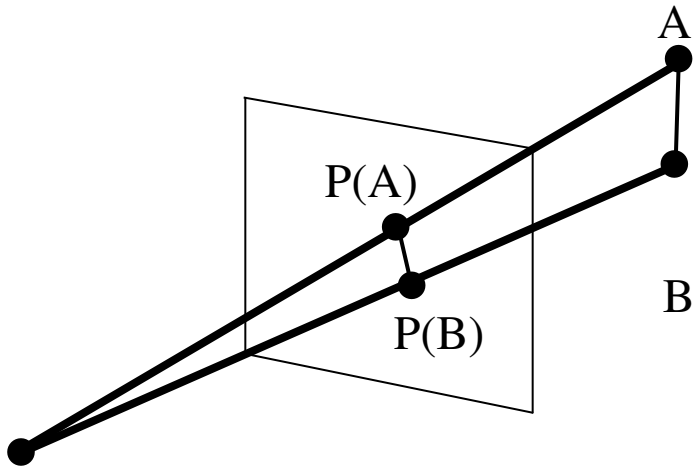
Projections

Utilisées en synthèse d'image et en vision par ordinateur (modèles de caméras)

Deux grandes familles :



Projection **parallèle**



Projection **perspective**

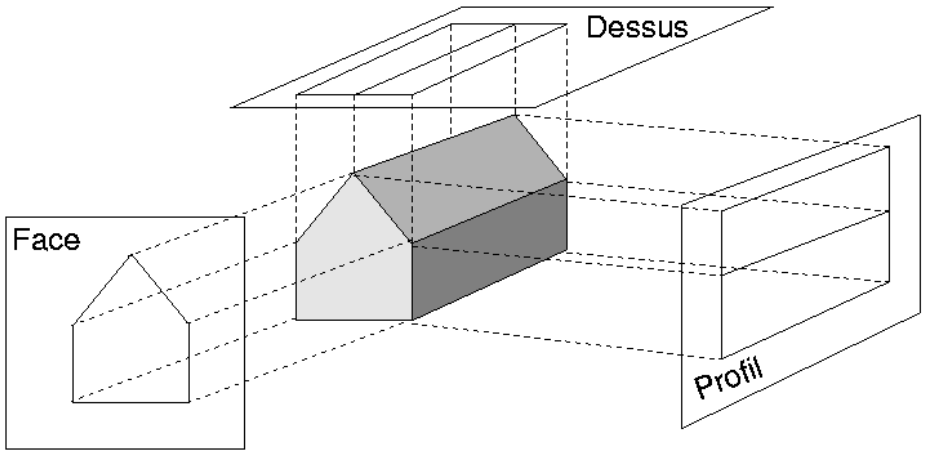
Projection parallèle

Projection **orthographique** lorsque la direction est **perpendiculaire** au plan de projection.

Projection **oblique** sinon.

Propriétés géométriques des projections parallèles

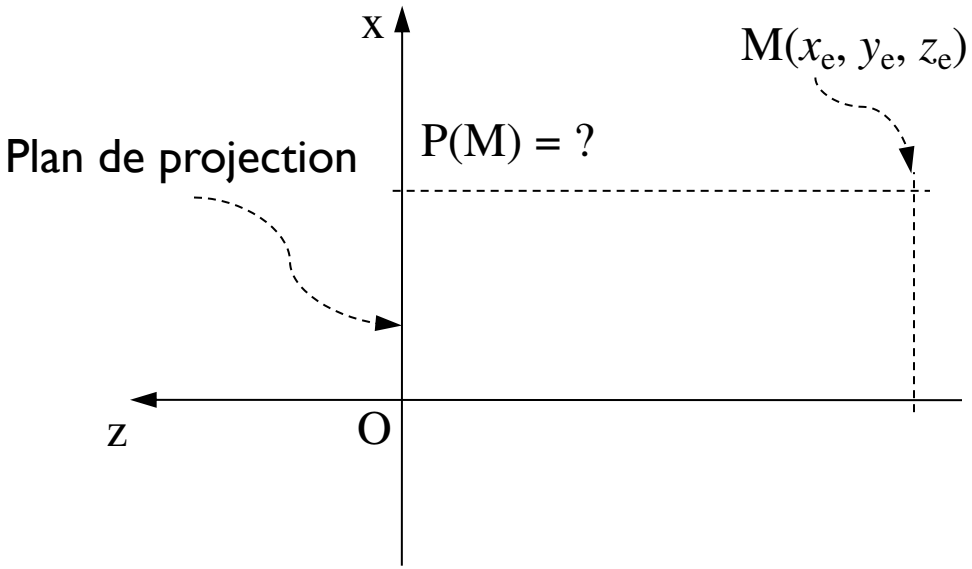
- Conservent le **parallélisme des droites**
- Conservent les **rapports des distances** selon une direction donnée



Projection parallèle

Matrice en coordonnées homogènes de la projection orthographique canonique :

$$P(x_e, y_e, z_e, w_e) = ?$$



Matrice de la projection orthographique sur xOy

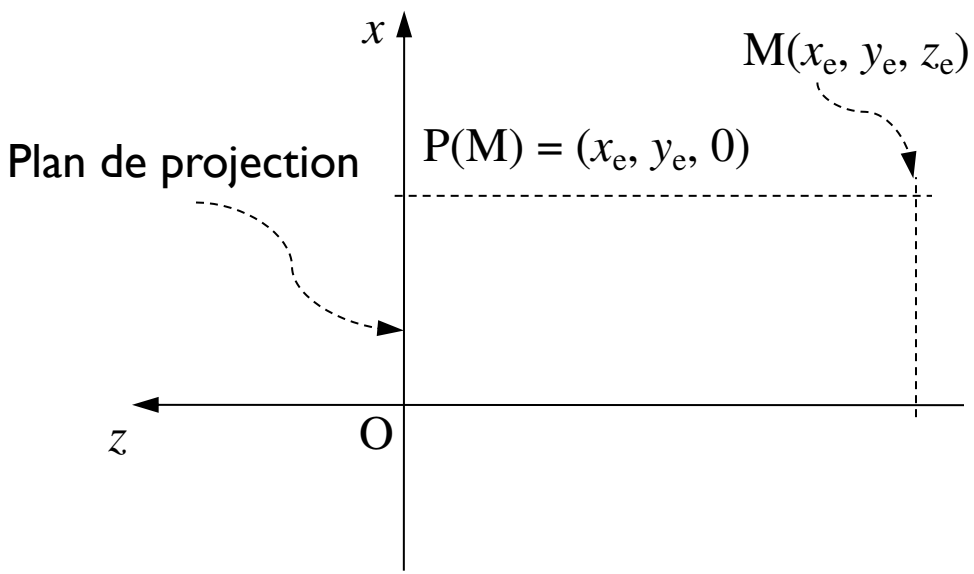
$$\begin{pmatrix} x_p \\ y_p \\ z_p \\ w_p \end{pmatrix} = \begin{pmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{pmatrix} \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix}$$

$$P(x_e, y_e) = (x_p/w_p,\ y_p/w_p,\ z_p/w_p)$$

Projection parallèle

Matrice en coordonnées homogènes de la projection orthographique canonique :

$$P(x_e, y_e, z_e, w_e) = (x_e, y_e, 0, 1)$$



Matrice de la projection orthographique sur xOy

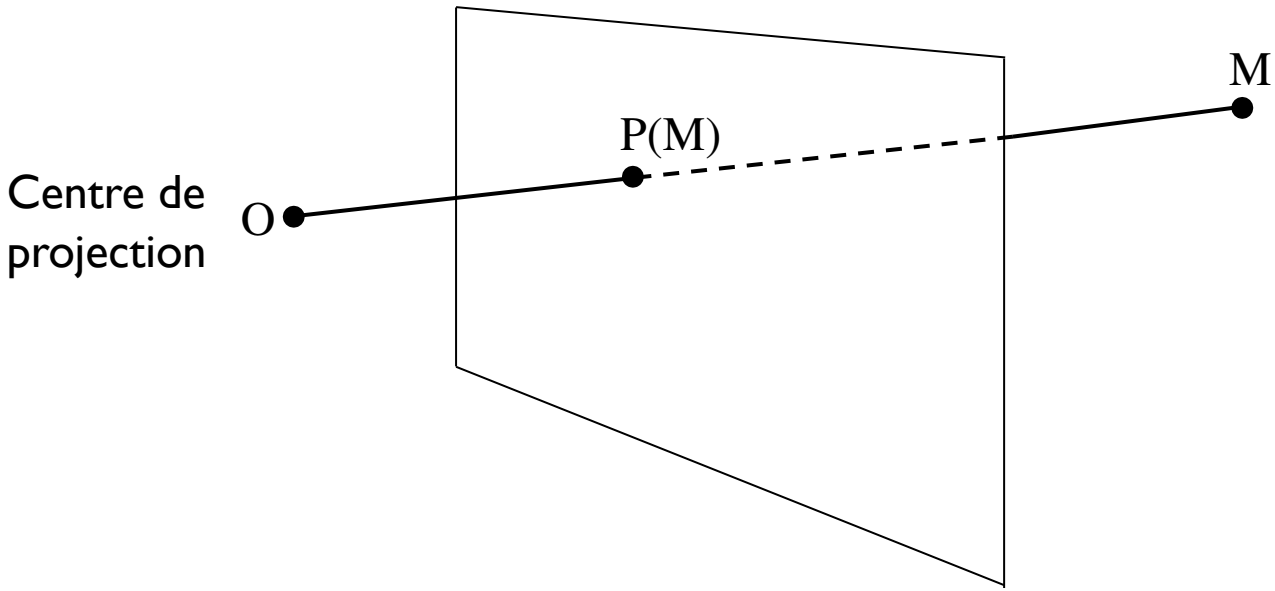
$$\begin{pmatrix} x_p \\ y_p \\ z_p \\ w_p \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix}$$

$$P(x_e, y_e) = (x_p/w_p,\ y_p/w_p,\ z_p/w_p)$$

Projection perspective

L'image de M sur le plan P par rapport au centre O est l'**intersection de la droite OM avec le plan P**

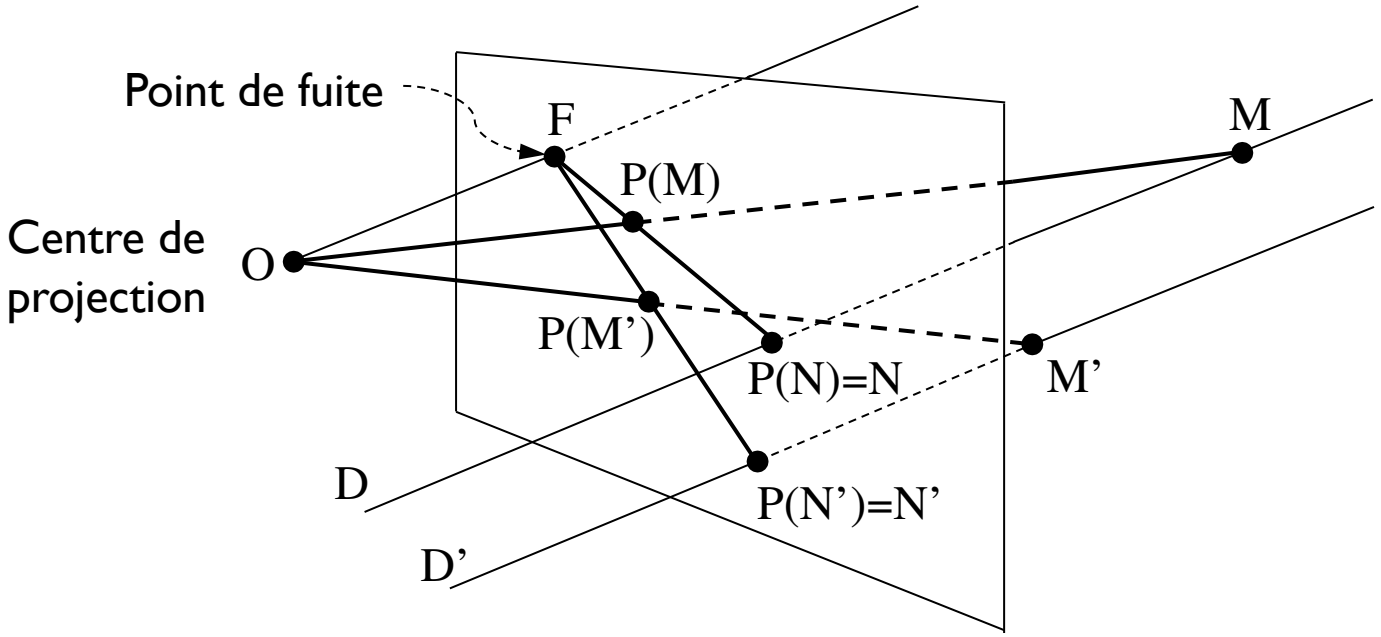
Remarque : Une projection en perspective dont le centre de projection est à l'infini est une projection parallèle



Projection perspective

Propriétés géométriques

- Ne **conservent pas le parallélisme** des droites non parallèles au plan de projection

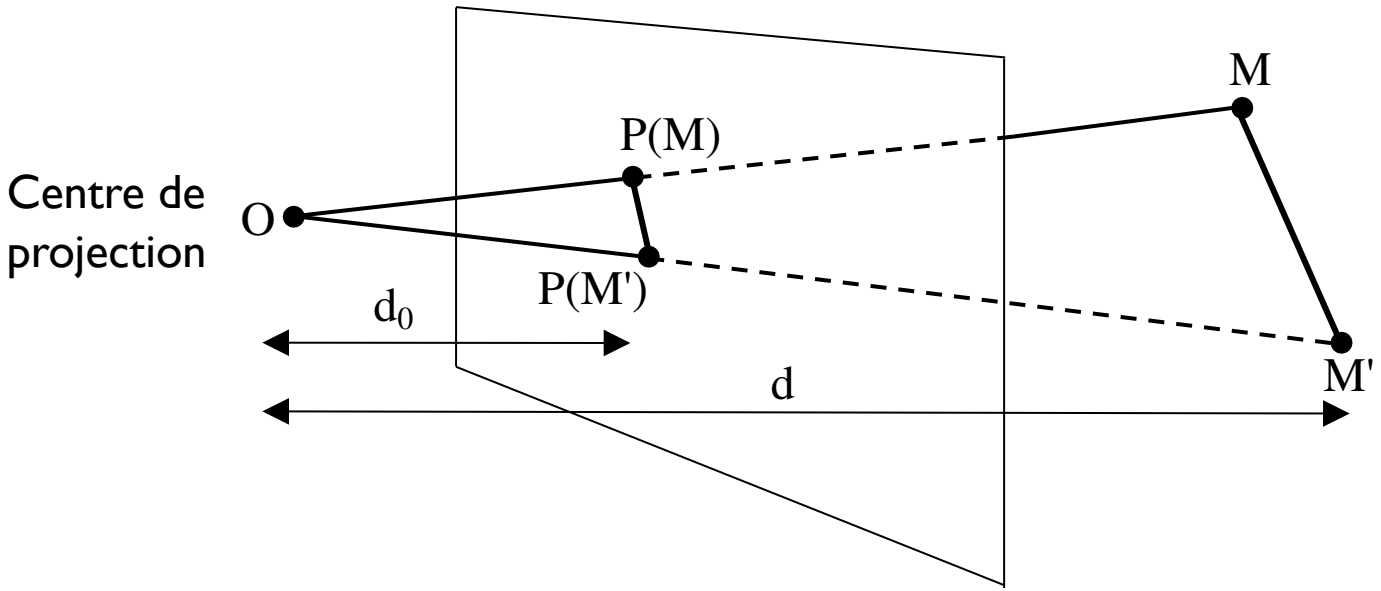


Projection perspective

Propriétés géométriques

- Ne conservent pas le parallélisme des droites non parallèles au plan de projection
- La **taille** d'un objet est **inversement proportionnelle** à sa distance au point de projection :

$$|P(M)\,P(M')| = |MM'|\,d_0/d$$

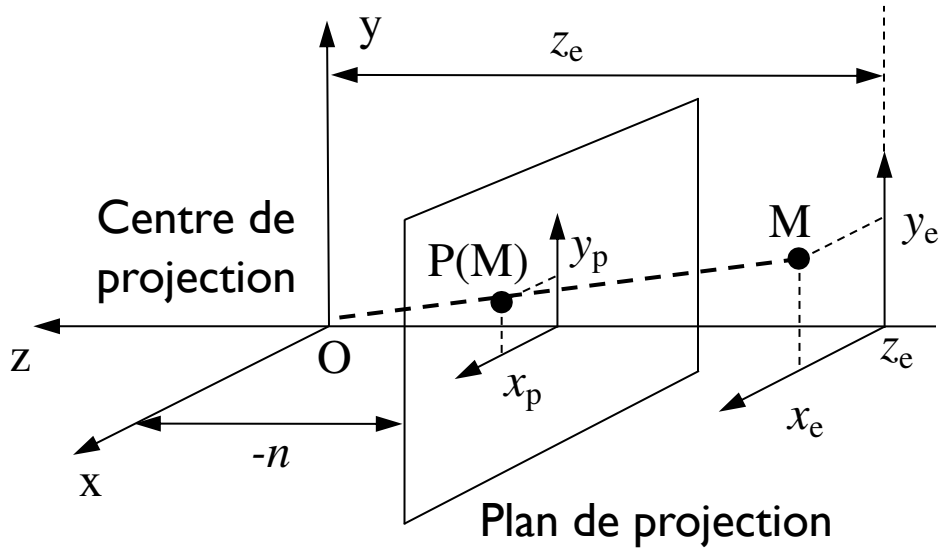


Projection perspective

Calcul des coordonnées projetées en perspective

- On se place dans le cas d'une projection canonique : centre O(0,0,0) et plan de projection parallèle à xOy, z=-*n*
- **Coordonnées dans le plan de projection**

$P(x_e, y_e) = ?$



Matrice de la projection de centre O sur le plan z=-*n*

$$\begin{pmatrix} x_p \\ y_p \\ z_p \\ w_p \end{pmatrix} = \begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & \end{pmatrix} \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix}$$

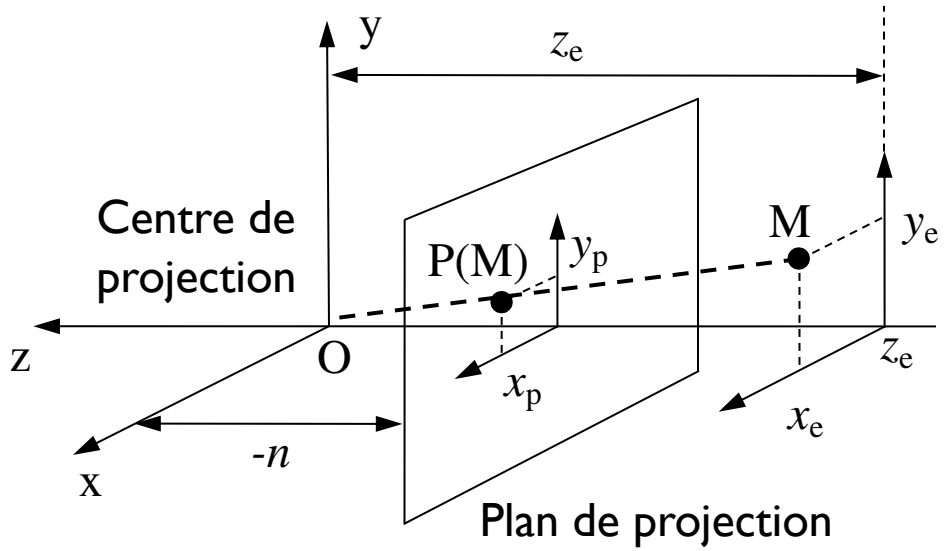
$P(x_e,y_e) = (x_p/w_p,\ y_p/w_p,\ z_p/w_p)$

Projections perspectives

Calcul des coordonnées projetées en perspective

- On se place dans le cas d'une projection canonique : centre O(0,0,0) et plan de projection parallèle à xOy, z=-*n*
- **Coordonnées dans le plan de projection**

$$P(x_e, y_e) = (x_p, y_p) = ((n\,x_e) / -z_e, (n\,y_e) / -z_e)$$



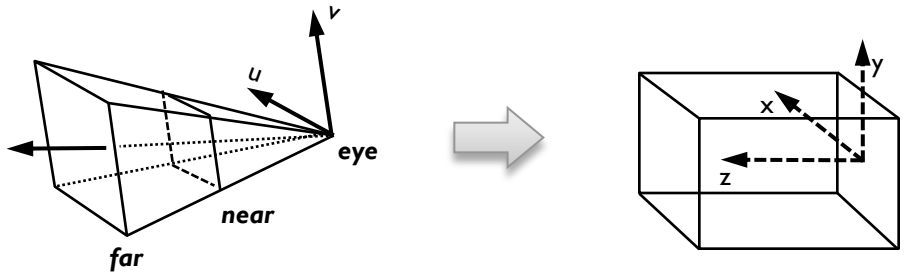
Matrice de la projection de centre O sur le plan z=-*n*

$$\begin{pmatrix} x_p \\ y_p \\ z_p \\ w_p \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/-n & 0 \end{pmatrix} \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix}$$

$$P(x_e,y_e) = (x_p/w_p,\ y_p/w_p,\ z_p/w_p)$$

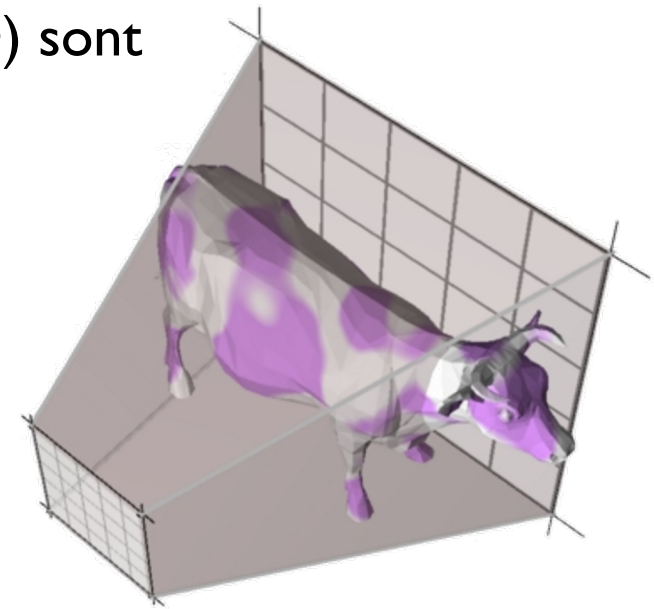
Clipping

Passage en **coordonnées normalisées**



Eye space *NDC*

Les portions en dehors du volume de vue (*frustum*) sont coupées.



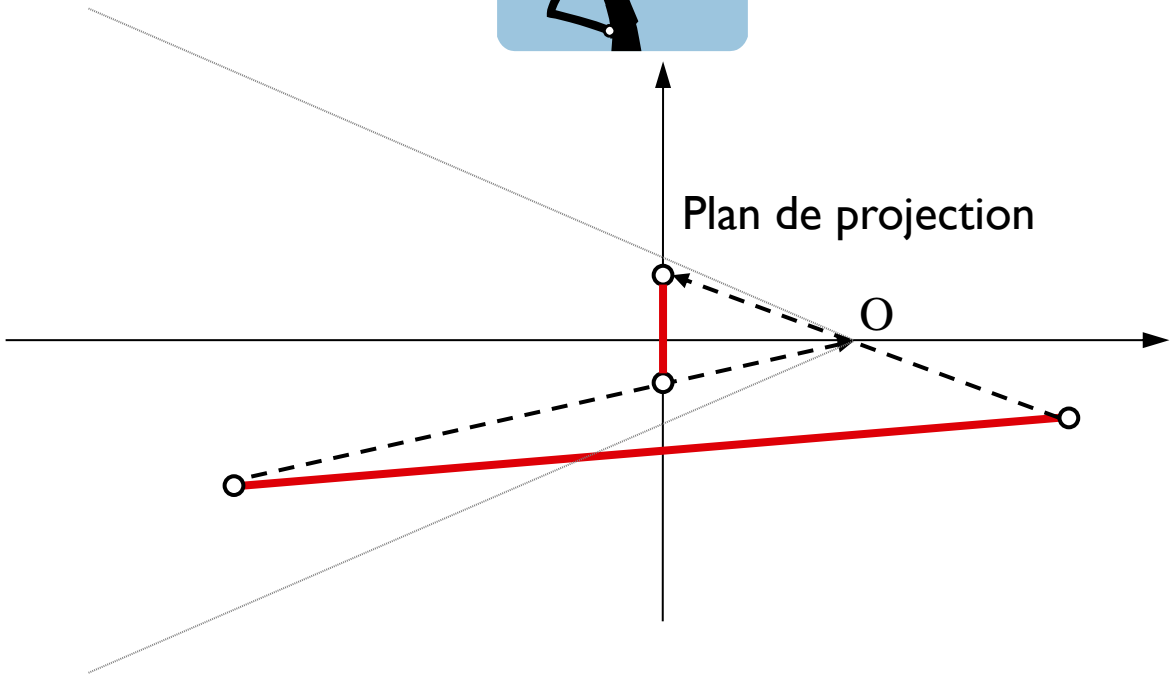
- Modeling Transformations
- Viewing Transformation (Perspective / Orthographic)
- **Clipping**
- Viewport transform
- Scan Conversion (Rasterization)
- Illumination (Shading)
- Per fragment operations (visibility, blending)

Clipping

Projection perspective = division par z

Que se passe-t-il quand un point est derrière la caméra ?

Sur le plan de projection ?

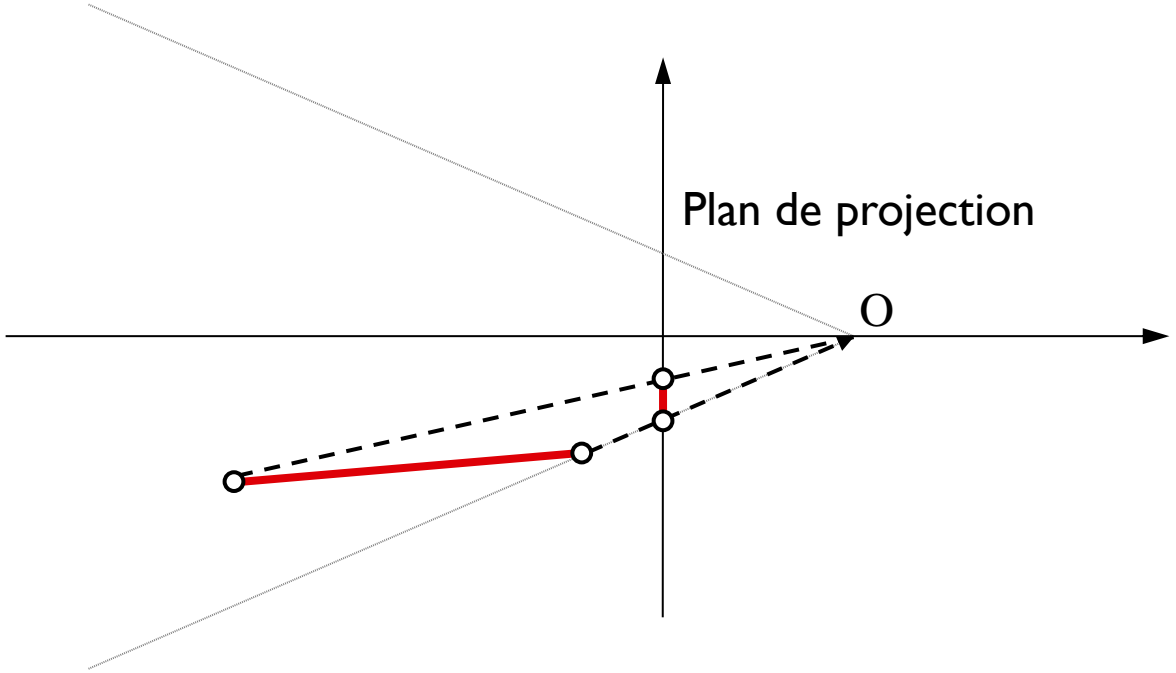


Clipping

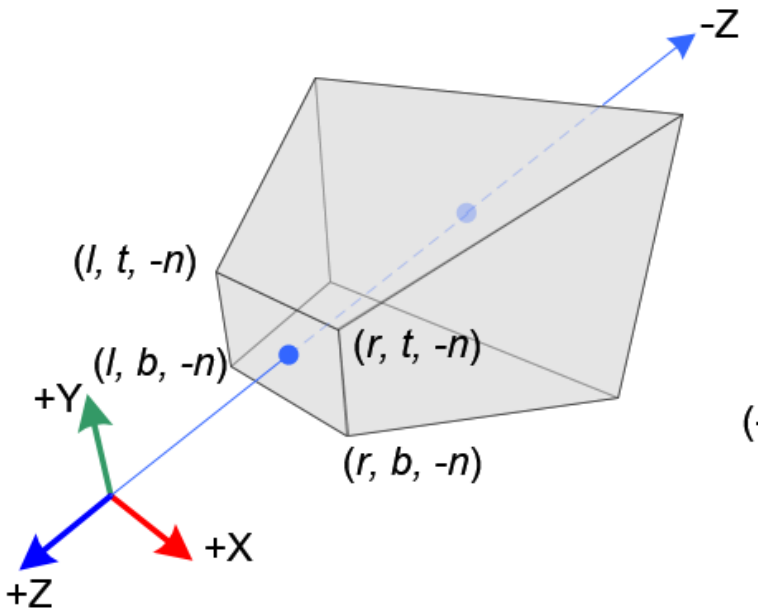
Projection perspective = division par z

Que se passe-t-il quand un point est derrière la caméra ?
Sur le plan de projection ?

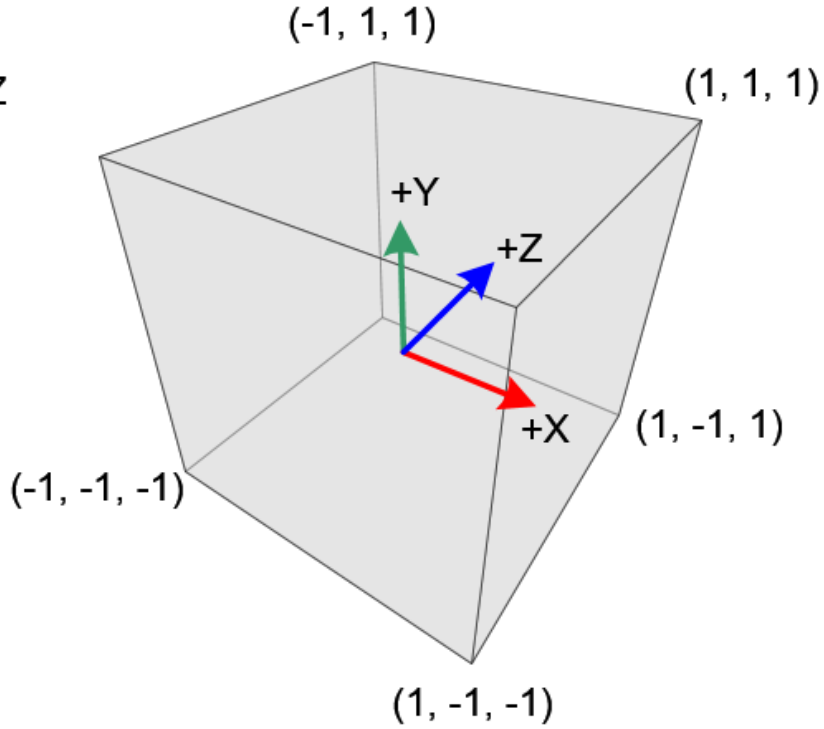
$\Longrightarrow$ **Solution : *clipping***



Coordonnées normalisées



Frustum perspective

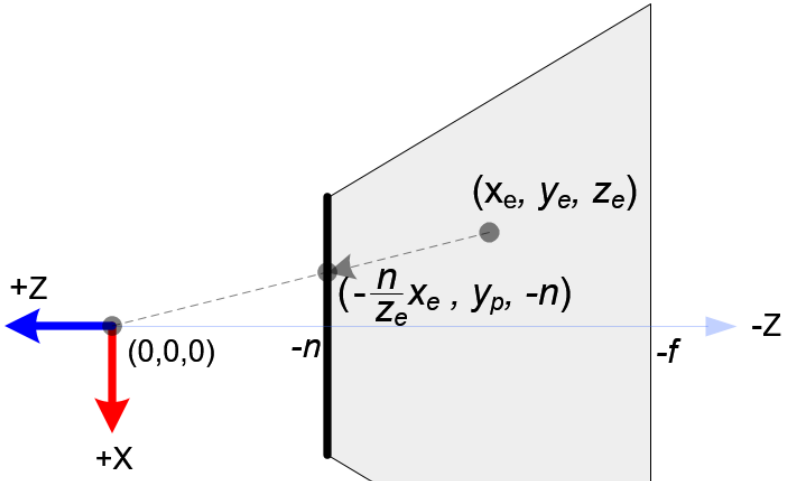


Normalized Device Coordinates (NDC)

http://www.songho.ca/opengl/gl_projectionmatrix.html

Coordonnées normalisées

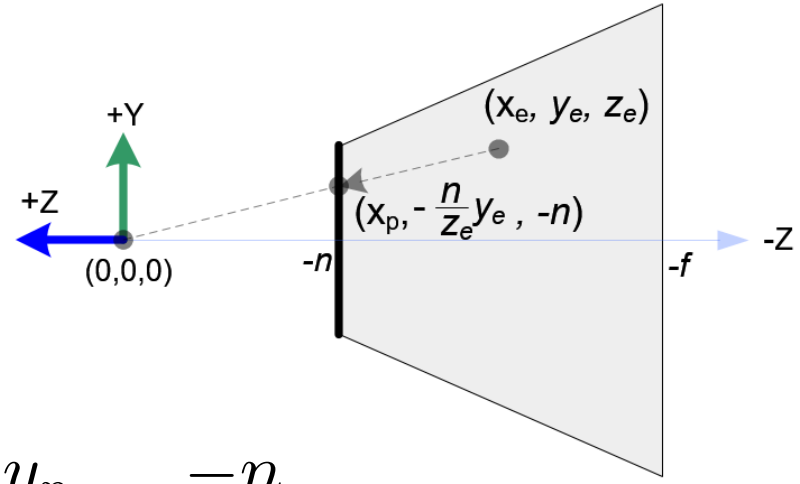
Vue du dessus



$$\frac{x_p}{x_e} = \frac{-n}{z_e}$$

$$x_p = \frac{n\,x_e}{-z_e}$$

Vue de côté



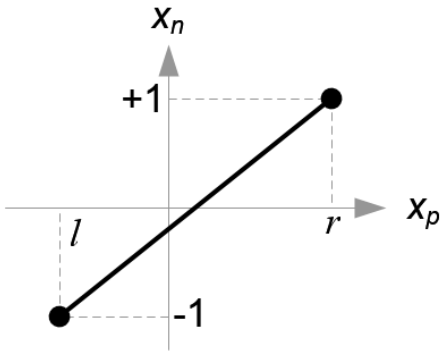
$$\frac{y_p}{y_e} = \frac{-n}{z_e}$$

$$y_p = \frac{n\,y_e}{-z_e}$$

http://www.songho.ca/opengl/gl_projectionmatrix.html

Coordonnées normalisées

$[l, r] \Rightarrow [-1, 1]$



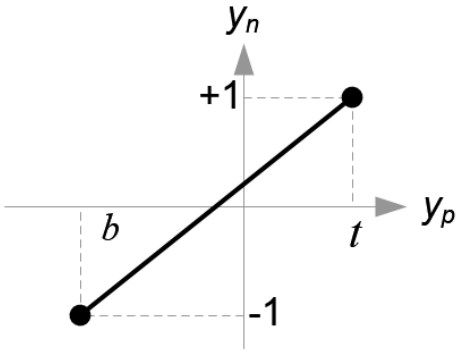
$$x_n = \frac{1-(-1)}{r-l}x_p + \beta$$

$$1 = \frac{2r}{r-l}\beta \qquad (x_p = r, x_n = 1)$$

$$\Rightarrow \beta = 1 - \frac{2r}{r-l} = -\frac{r+l}{r-l}$$

$$\Rightarrow x_n = \frac{2x_p - r - l}{r-l}$$

$[b, t] \Rightarrow [-1, 1]$



Similairement :

$$y_n = \frac{2y_p - t - b}{t-b}$$

http://www.songho.ca/opengl/gl_projectionmatrix.html

Coordonnées normalisées

$$x_n = \frac{2x_p - r - l}{r - l} \quad \text{or } x_p = \frac{nx_e}{-z_e}$$

$$= \frac{2\dfrac{nx_e}{-z_e} - r - l}{r - l}$$

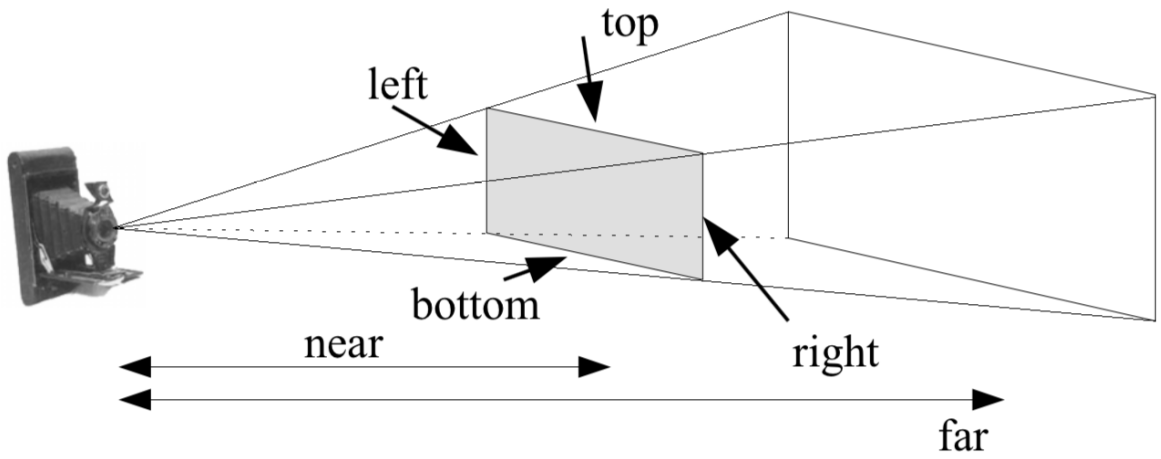
$$= \frac{\dfrac{2n}{r-l}x_e + \dfrac{r+l}{r-l}z_e}{-z_e} = x_c$$

Similairement :

$$y_n = \frac{\dfrac{2n}{t-b}y_e + \dfrac{t+b}{t-b}z_e}{-z_e} = y_c$$

http://www.songho.ca/opengl/gl_projectionmatrix.html

Coordonnées normalisées



$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & A & B \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix}$$

$$z_n = \frac{z_c}{w_c} = \frac{Az_e + B}{-z_e}$$

http://www.songho.ca/opengl/gl_projectionmatrix.html

Coordonnées normalisées

$[-n, -f] \Rightarrow [-1, 1]$

$$z_n = \frac{Az_e + B}{-z_e} \Rightarrow \begin{cases} \dfrac{-An + B}{n} = -1 \\ \dfrac{-Af + B}{f} = 1 \end{cases} \Leftrightarrow \begin{cases} -An + B = -n & (1) \\ -Af + B = f & (2) \end{cases}$$

$$(1) \Rightarrow B = An - n \quad (3)$$

$$(2)+(3) \Rightarrow -Af + (An - n) = f$$

$$\Leftrightarrow -(f - n)A = f + n$$

$$\Leftrightarrow A = -\frac{f + n}{f - n} \quad (4)$$

http://www.songho.ca/opengl/gl_projectionmatrix.html

Coordonnées normalisées

(4) $A = -\dfrac{f+n}{f-n}$ et (1) $-An + B = -n$

$$\Rightarrow \left(\frac{f+n}{f-n}\right) n + B = -n$$

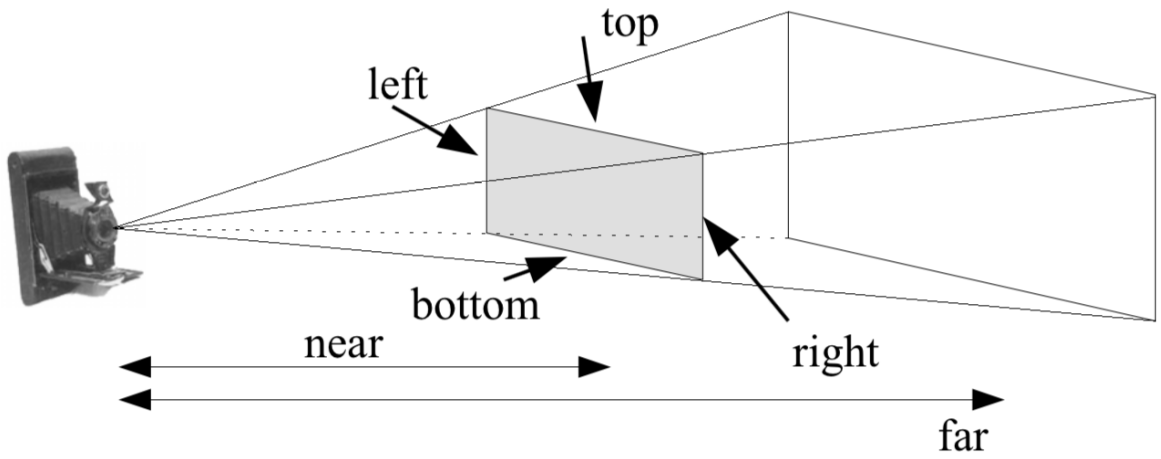
$$\Rightarrow B = -n - \left(\frac{f+n}{f-n}\right) n = -\left(1 + \frac{f+n}{f-n}\right) n = -\left(1 + \frac{f-n+f+n}{f-n}\right) n$$

$$= -\frac{2fn}{f-n}$$

D'où : $z_n = \dfrac{Az_e + B}{-z_e} \Rightarrow z_n = \dfrac{-\dfrac{f+n}{f-n} z_e + -\dfrac{2fn}{f-n}}{-z_e}$

http://www.songho.ca/opengl/gl_projectionmatrix.html

Coordonnées normalisées



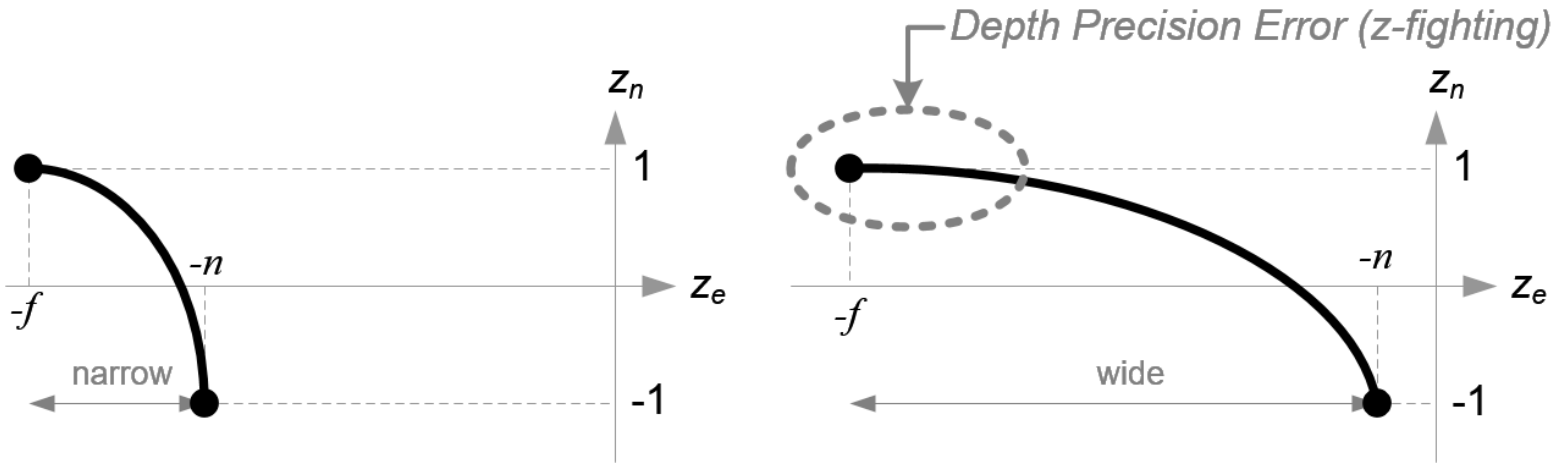
$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

http://www.songho.ca/opengl/gl_projectionmatrix.html

Projection perspective

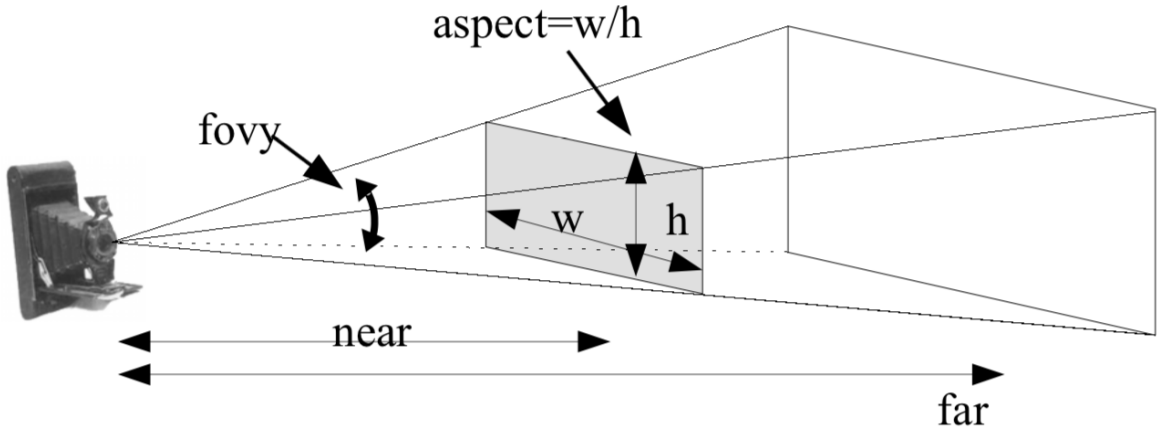
Relation non-linéaire en z

$\Longrightarrow$ grande précision près du plan ***near***
mais faible précision au niveau du plan ***far***

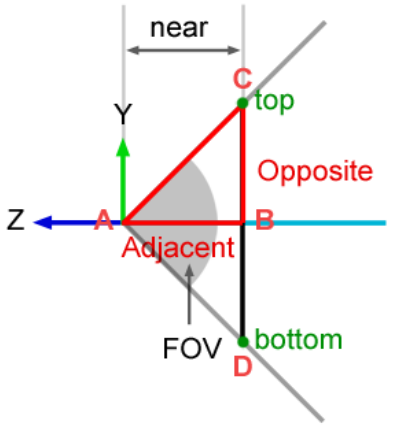


http://www.songho.ca/opengl/gl_projectionmatrix.html

Projection perspective



$$\tan\left(\frac{\text{fovy}}{2}\right) = \frac{\text{opposite}}{\text{adjacent}} = \frac{t}{n}$$

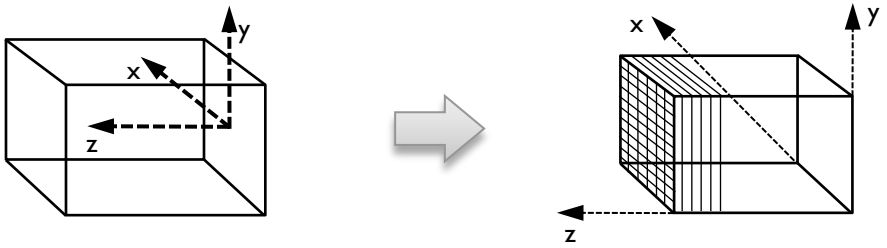


$$\Rightarrow \begin{cases} t = \tan\left(\frac{\text{fovy}}{2}\right) \times n \\ b = -t \\ r = -t \times \text{aspect} \\ l = -r \end{cases}$$

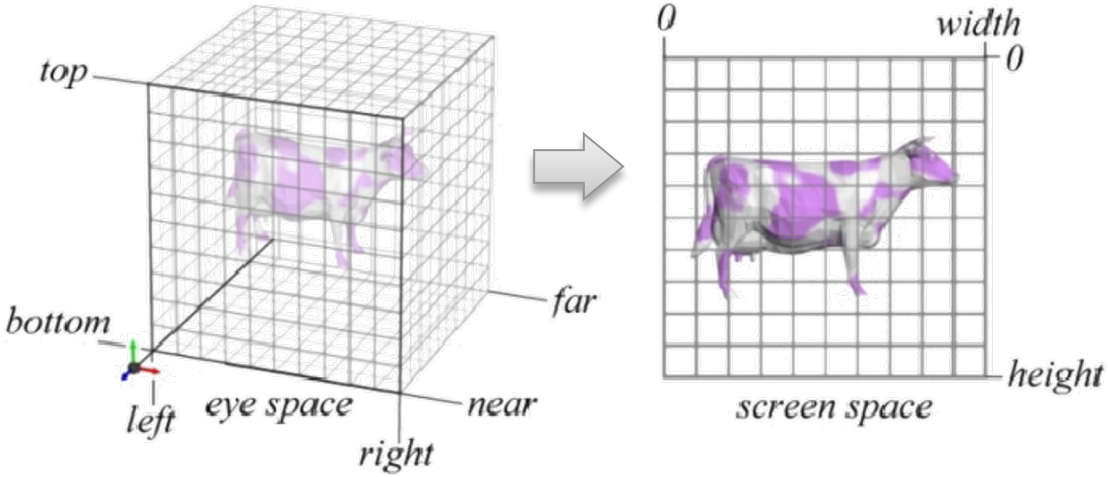
http://www.songho.ca/opengl/gl_projectionmatrix.html

Viewport

Les primitives 3D sont projetées sur l'image 2D *(screen space)*



NDC *Screen Space*

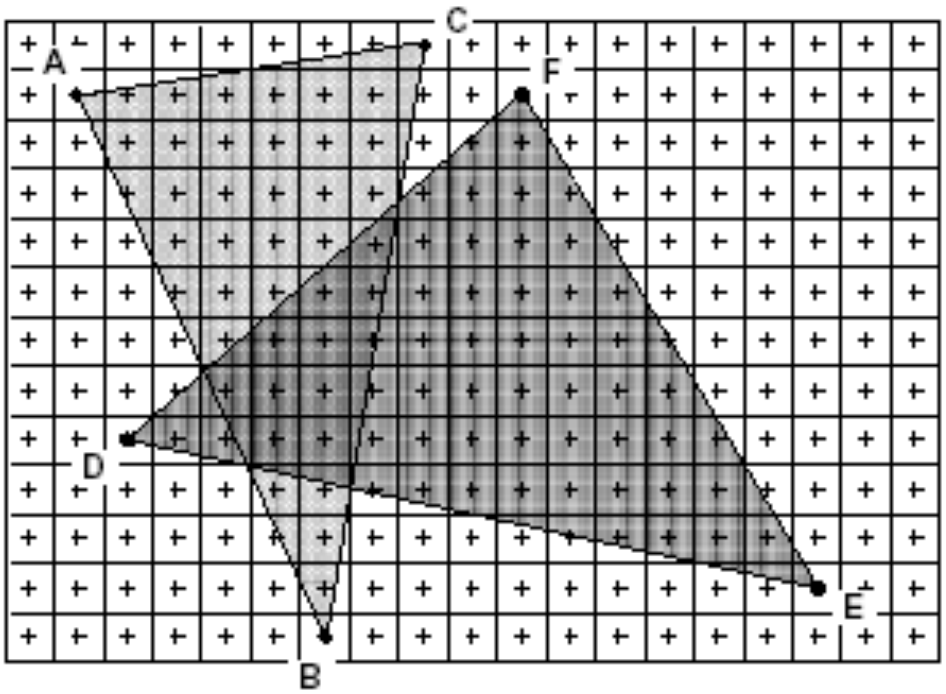


- Modeling Transformations
- Viewing Transformation (Perspective / Orthographic)
- Clipping
- **Viewport transform**
- Scan Conversion (Rasterization)
- Illumination (Shading)
- Per fragment operations (visibility, blending)

Rastérisation

Découpe de la primitive 2D **en pixels**

Interpolation des valeurs connues aux sommets : couleur, profondeur,…



Modeling Transformations

Viewing Transformation (Perspective / Orthographic)

Clipping

Viewport transform

Scan Conversion (Rasterization)

Illumination (Shading)

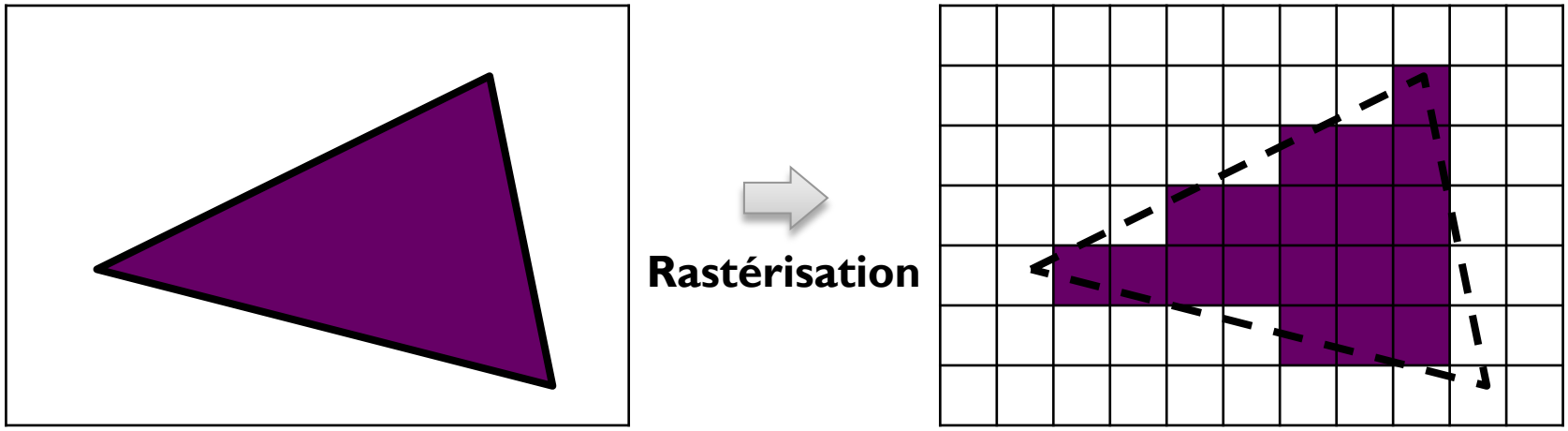
Per fragment operations (visibility, blending)

Rastérisation

Un **fragment** = un **pixel** dans l'image finale

Les attributs (couleur, coordonnées de textures, profondeur) sont assignés à chaque fragment

Ces informations sont **interpolées** à partir des valeurs aux sommets



Représentation **vectorielle**

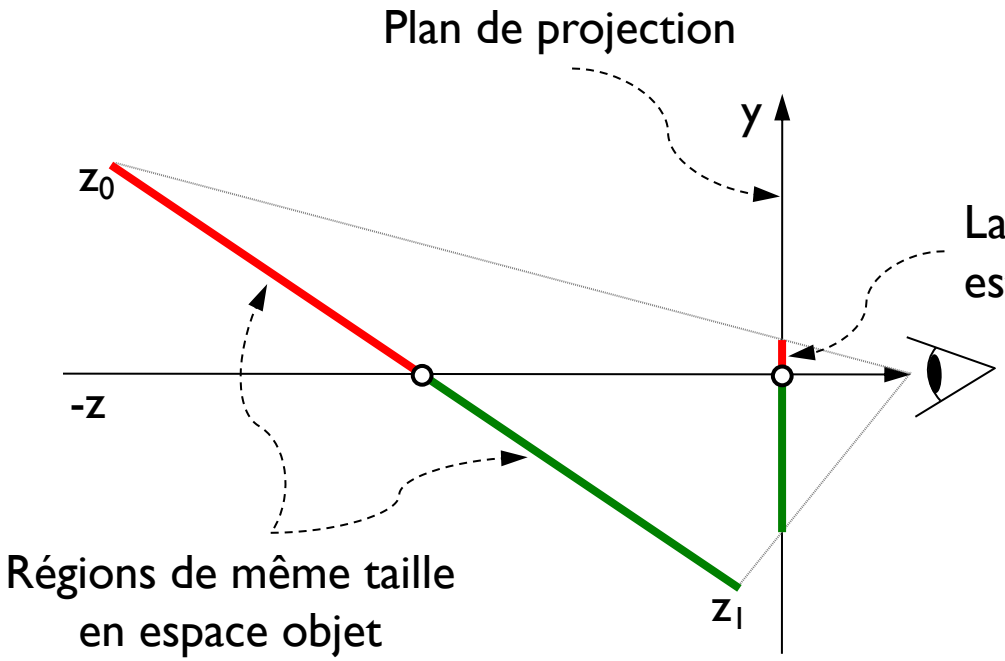
Représentation **matricielle**

Interpolation des attributs

On doit interpoler les attribut définis **aux sommets**
(couleurs, normales, coordonnées de textures…)

Interpolation linéaire en espace objet

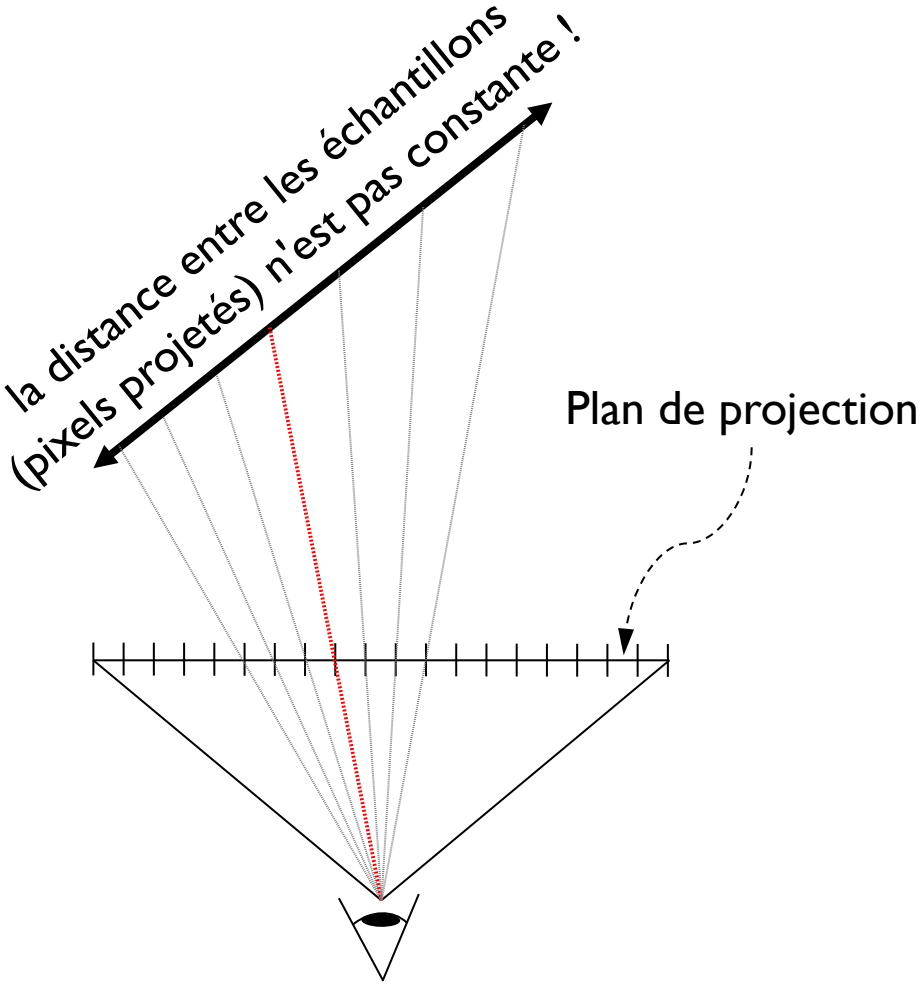
⟹ **Est-elle linéaire en espace écran ?**



Attention à la perspective

La projection perspective n'est pas linéaire !

$\Longrightarrow$ On ne peut pas interpoler linéairement dans le plan image 2D



Attention à la perspective !

Solution : interpoler en considérant la profondeur,

$\Longrightarrow$ **$1/z$ et u/z sont linéaires en espace écran**

$\Longrightarrow$ calculs en **coordonnées homogènes**

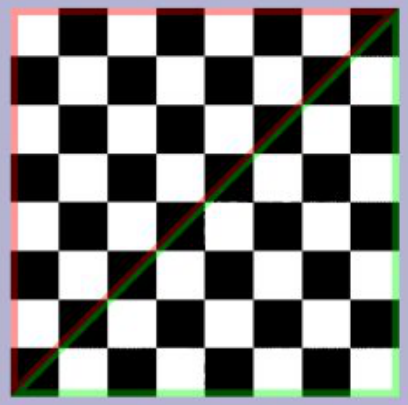
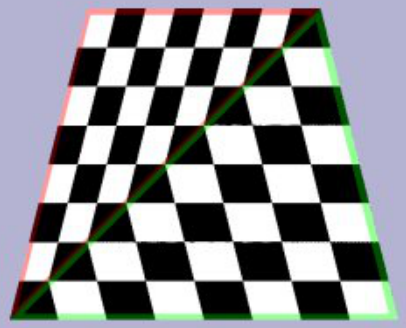
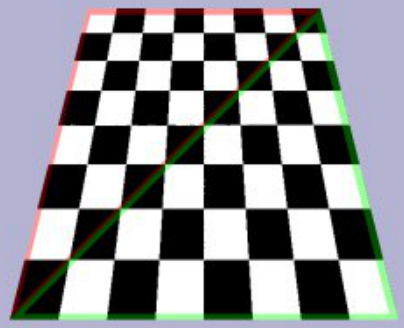
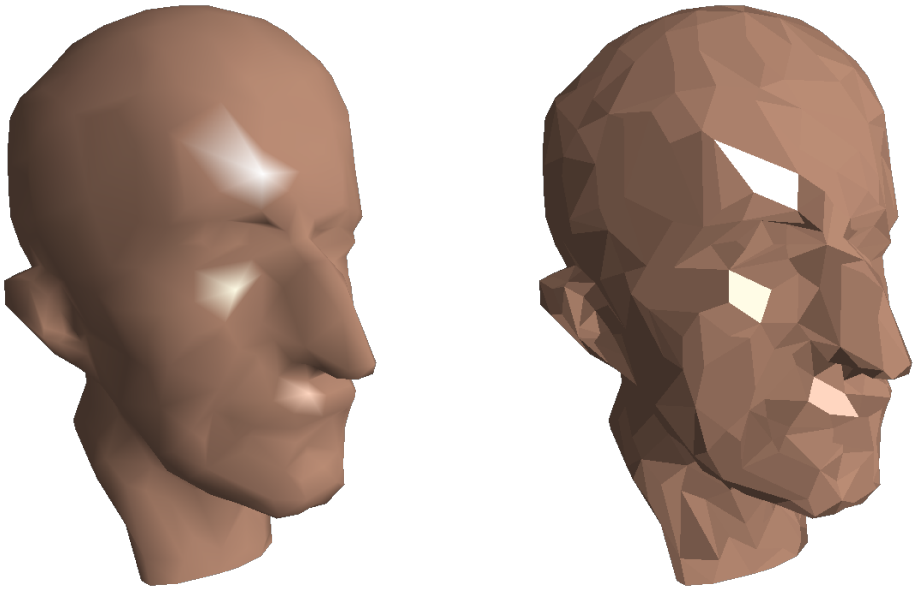
Interpolation à z constant	**Interpolation linéaire en espace image**	**Prise en compte de la perspective**
		
	$u_\alpha = (1-\alpha)u_0 + \alpha u_1$ $\alpha \in [0,1]$	$u_\alpha = \dfrac{(1-\alpha)\frac{u_0}{z_0} + \alpha\frac{u_1}{z_1}}{(1-\alpha)\frac{1}{z_0} + \alpha\frac{1}{z_1}}$

Image : Wikipedia

Interpolation des attributs

Contrôle de l'interpolation en GLSL

```
                in vec3 color;   // interpolation lisse par défaut
smooth          in vec3 color;
flat            in vec3 color;   // copie la valeur du 1er sommet
noperspective   in vec3 color;   // interpolation linéaire
```

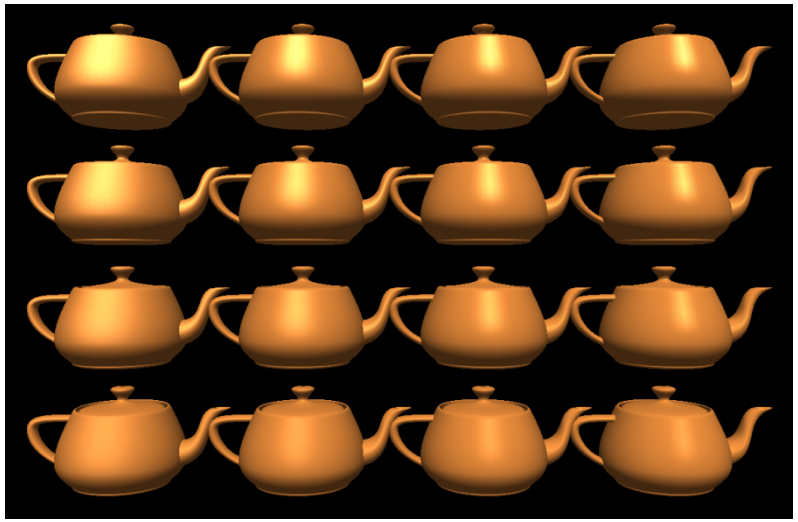


smooth flat

Illumination

Les primitives sont éclairées selon leur **matériau** et les **sources de lumière**.

Les modèles d'éclairage sont **locaux**
⟹ le calcul est effectué **par primitive**.



Modeling
Transformations

Viewing Transformation
(Perspective / Orthographic)

Clipping

Viewport transform

Scan Conversion
(Rasterization)

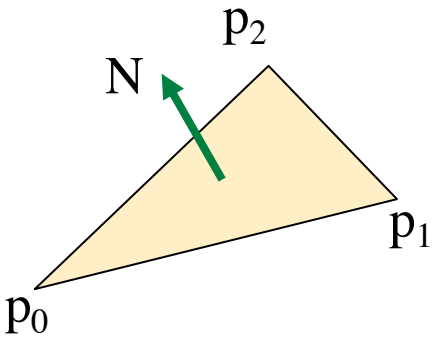
Illumination
(Shading)

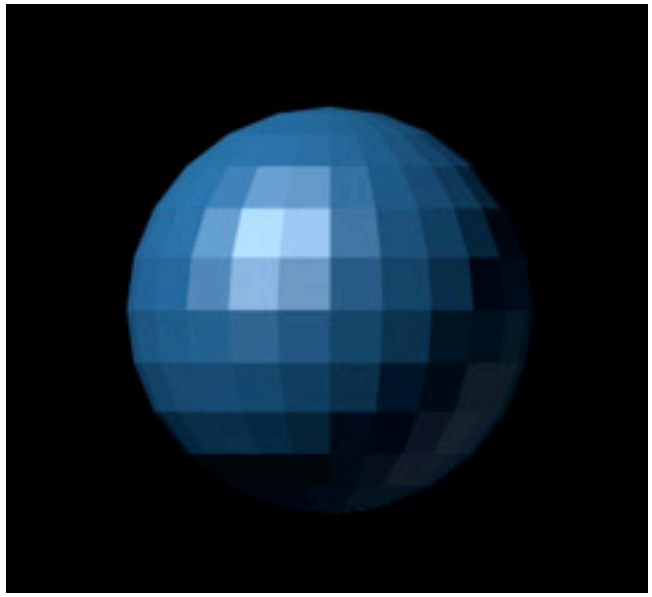
Per fragment operations
(visibility, blending)

Flat shading

Calcul des valeurs d'**intensité par face**

- Calculer une seule valeur d'illumination pour la face
- Ex. : milieu de la face, normale est celle du plan contenant la face N =



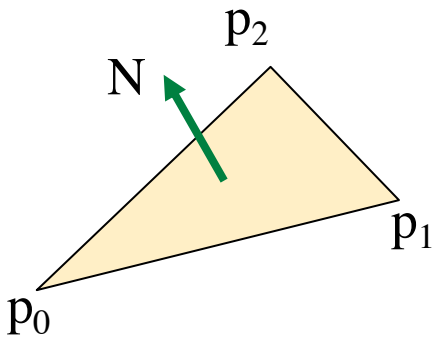


http://unit20alistair2.blogspot.fr/

Flat shading

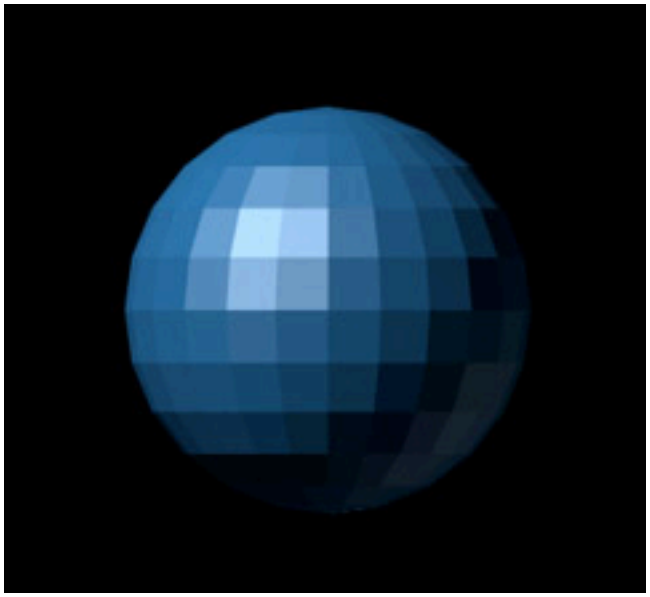
Calcul des valeurs d'**intensité par face**

- Calculer une seule valeur d'illumination pour la face
- Ex. : milieu de la face, normale est celle du plan contenant la face $N = (\mathbf{p}_1\text{-}\mathbf{p}_0) \times (\mathbf{p}_2\text{-}\mathbf{p}_0)$



Approche **valide quand** :

- la source lumineuse est à l'infini
- la projection est orthographique
- la surface est composée de faces polygonales uniquement



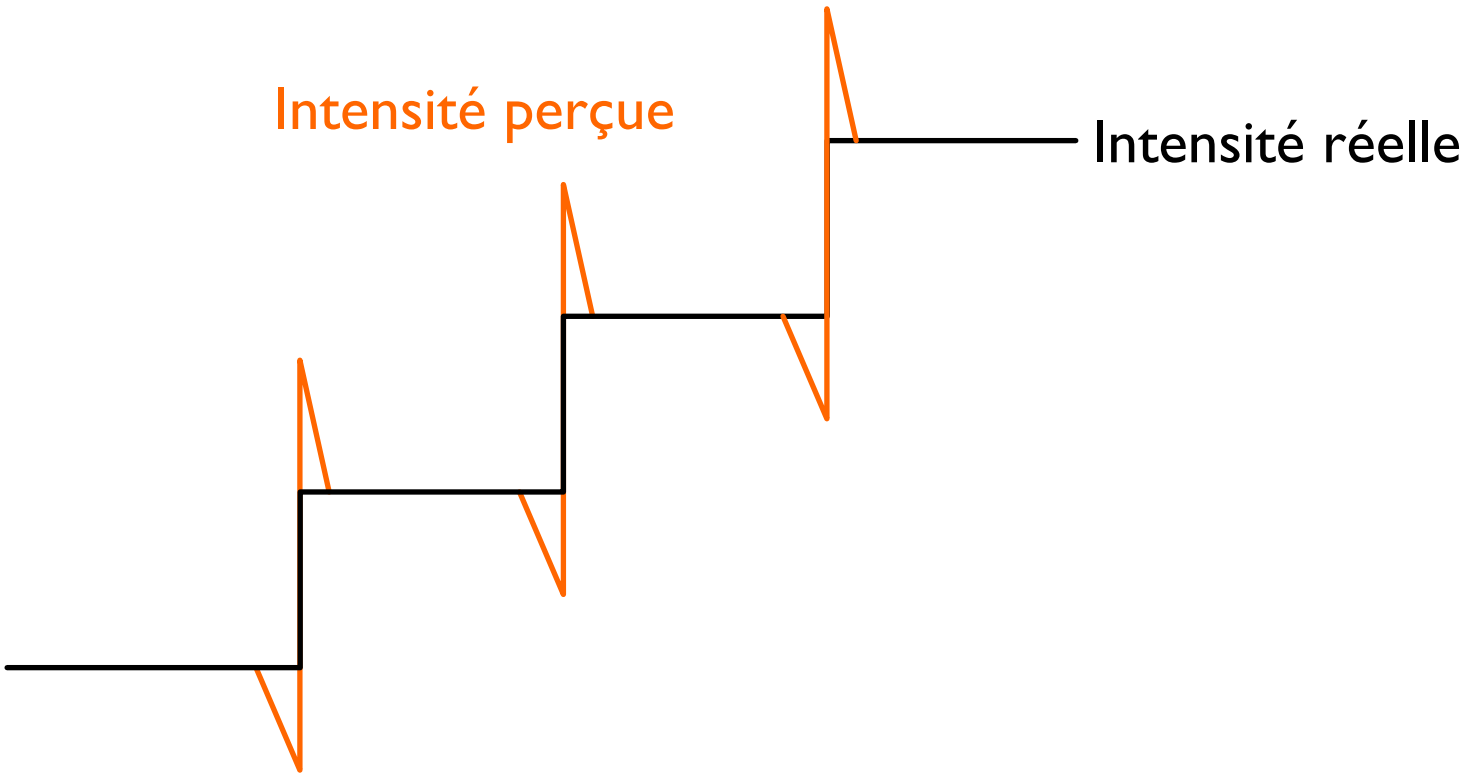
http://unit20alistair2.blogspot.fr/

Flat shading

Problème : discontinuités entre les faces

L'œil les perçoit très bien

$\Longrightarrow$ effet de « **Mach Banding** »

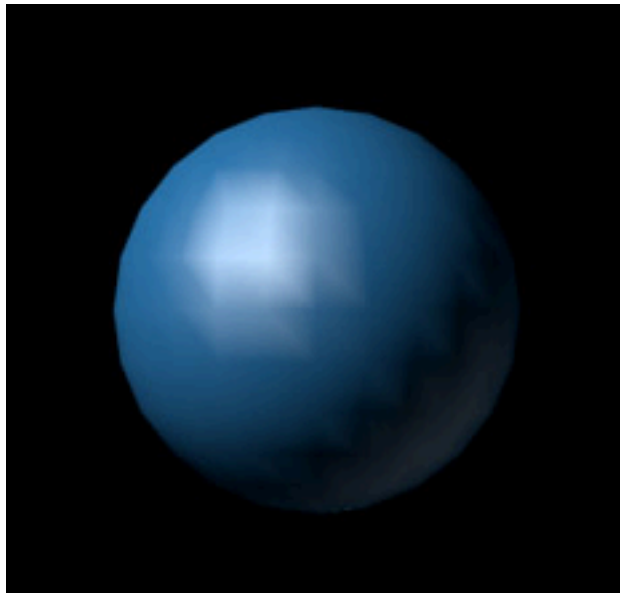


« Mach Banding »

Gouraud shading

Interpolation des valeurs d'intensité calculées **aux sommets** de la face

1. Calcul des normales aux sommets
2. Calcul des intensités (couleurs) aux sommets
3. Interpolation des intensité avec les coordonnées barycentriques

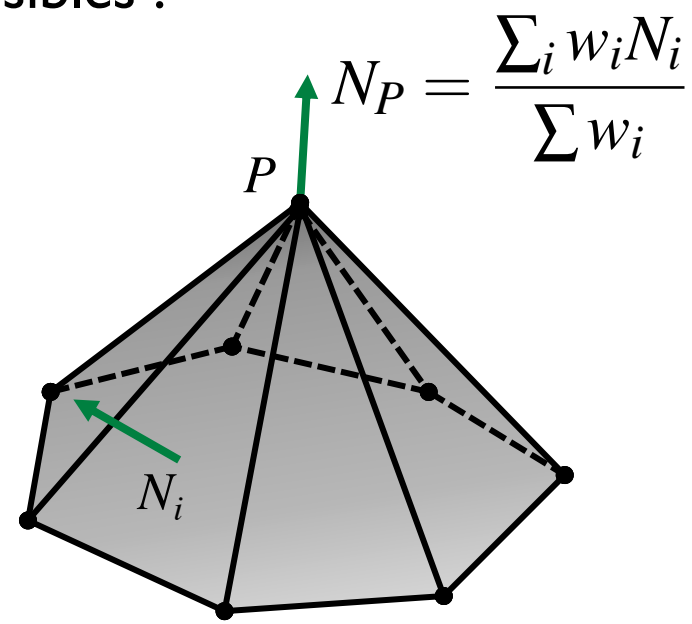


http://unit20alistair2.blogspot.fr/

Normales aux sommets

Moyenne (pondérée) des **normales des faces** adjacentes à ce sommet

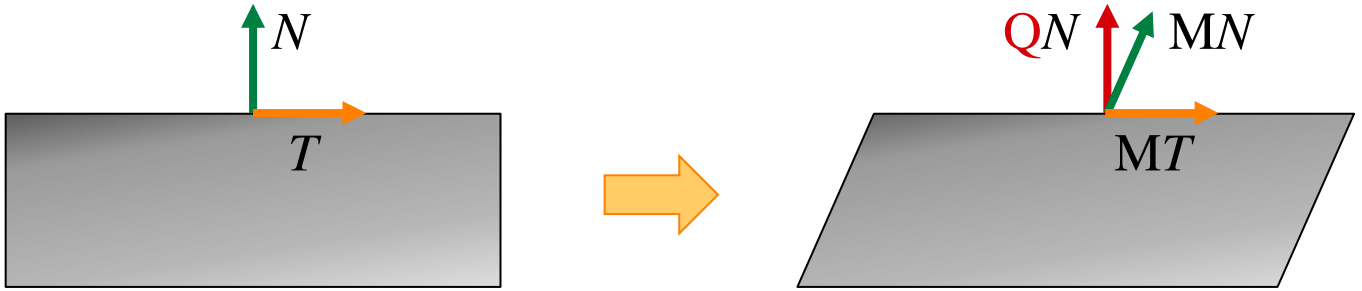
- Normales unitaires
- Différentes pondérations w_i possibles :
  - Unité
  - Aire de la face
  - Angle entre faces adjacentes



c.f. http://www.bytehazard.com/articles/vertnorm.html

Transformation des normales

Ne se transforme pas comme un point



Quelle est la bonne transformation $\mathbf{Q}$?

- Avant transformation : $N \cdot T = 0 \Leftrightarrow N^\top T = 0$
- Après transformation :

$$(\mathbf{Q}N)^\top(\mathbf{M}T) = N^\top \mathbf{Q}^\top \mathbf{M}T = 0$$

$$\Rightarrow \mathbf{Q}^\top \mathbf{M} = \mathbf{I}$$

$$\Rightarrow \mathbf{Q} = (\mathbf{M}^{-1})^\top$$

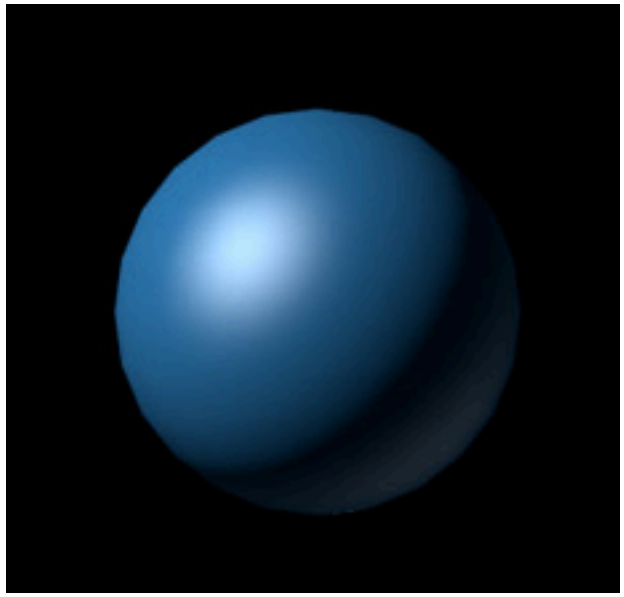
…pas en coordonnées homogène (seulement la matrice 3x3 supérieure)

Phong shading

…à ne pas confondre avec la BRDF de Phong !

Interpolation des normales calculées **aux sommets** de la face

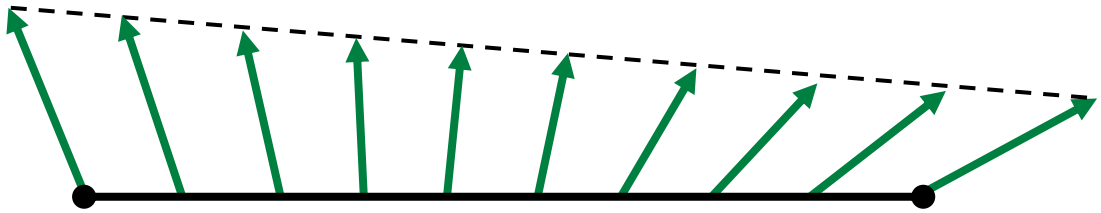
1. Calcul des normales aux sommets
2. Interpolation des normales avec les coordonnées barycentriques
3. Calcul des intensités par fragment



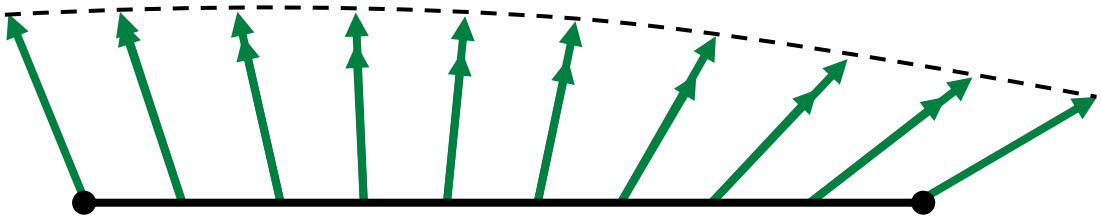
http://unit20alistair2.blogspot.fr/

Interpolation des normales

Interpolation naïve



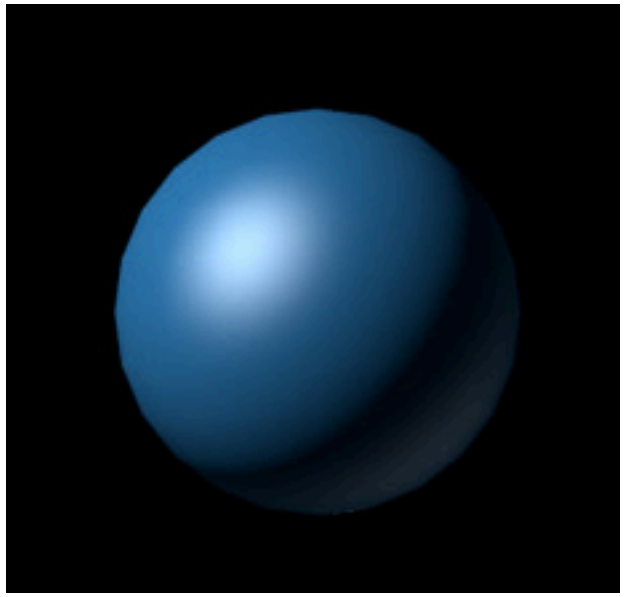
Interpolation avec normalisation



Phong shading

Interpolation des normales calculées **aux sommets** de la face

1. Calcul des normales aux sommets
2. Interpolation des normales lors de la rastérisation
3. **Normalisation des normales**
4. Calcul des intensités par fragment



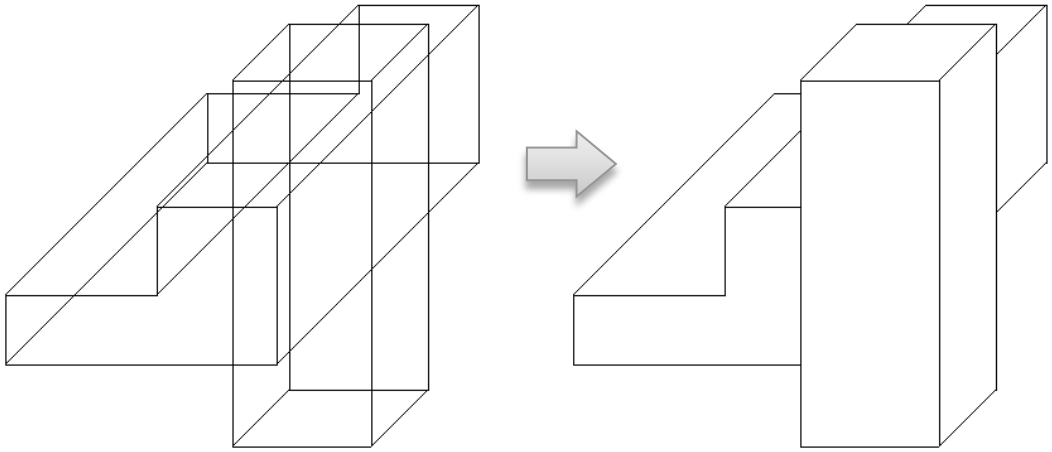
http://unit20alistair2.blogspot.fr/

Visibilité, affichage

Calcule des primitives visibles :

$\Longrightarrow$ ***depth buffer***

Remplissage du ***frame buffer*** avec le bon format de couleur



Modeling Transformations

Viewing Transformation (Perspective / Orthographic)

Clipping

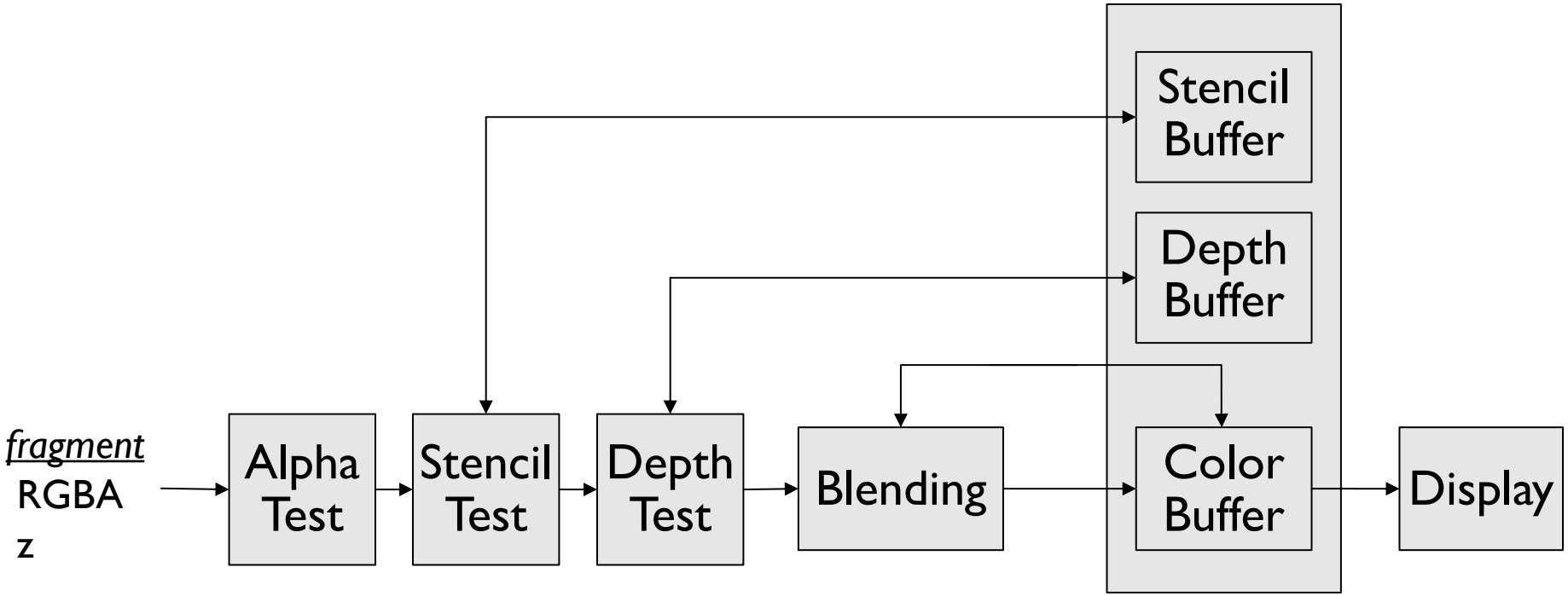
Viewport transform

Scan Conversion (Rasterization)

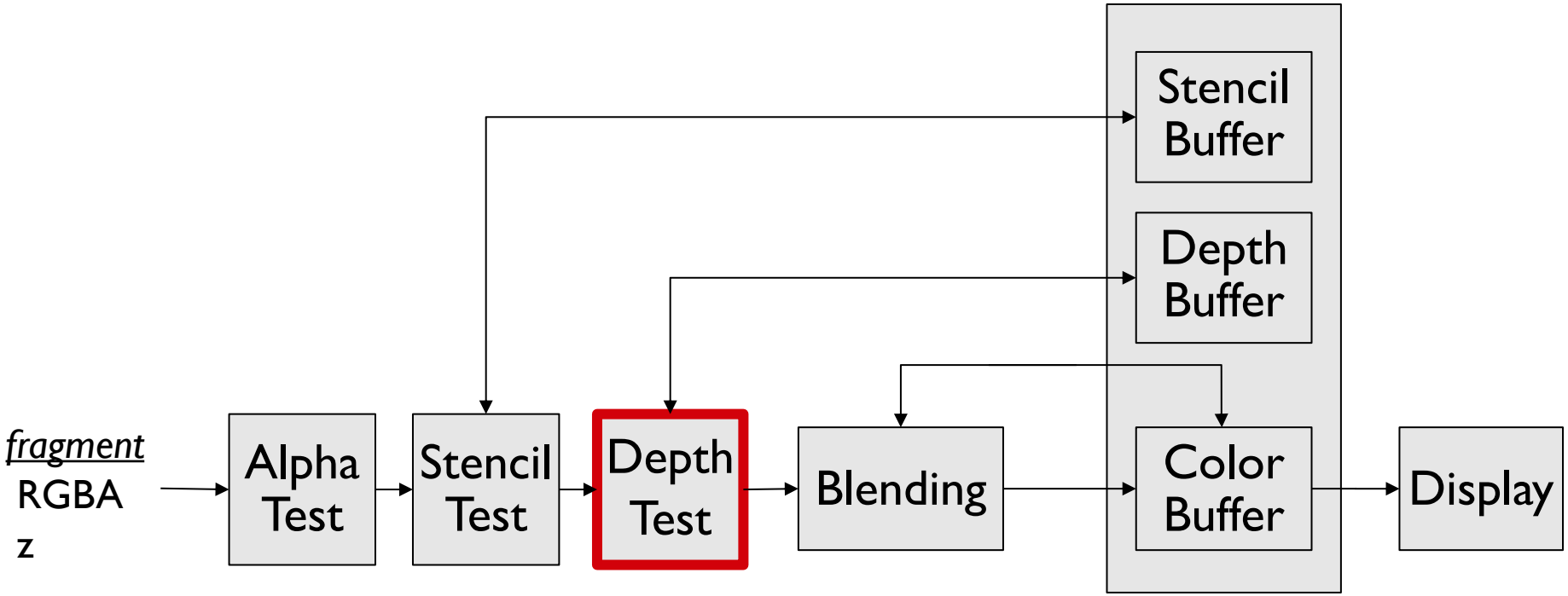
Illumination (Shading)

Per fragment operations (visibility, blending)

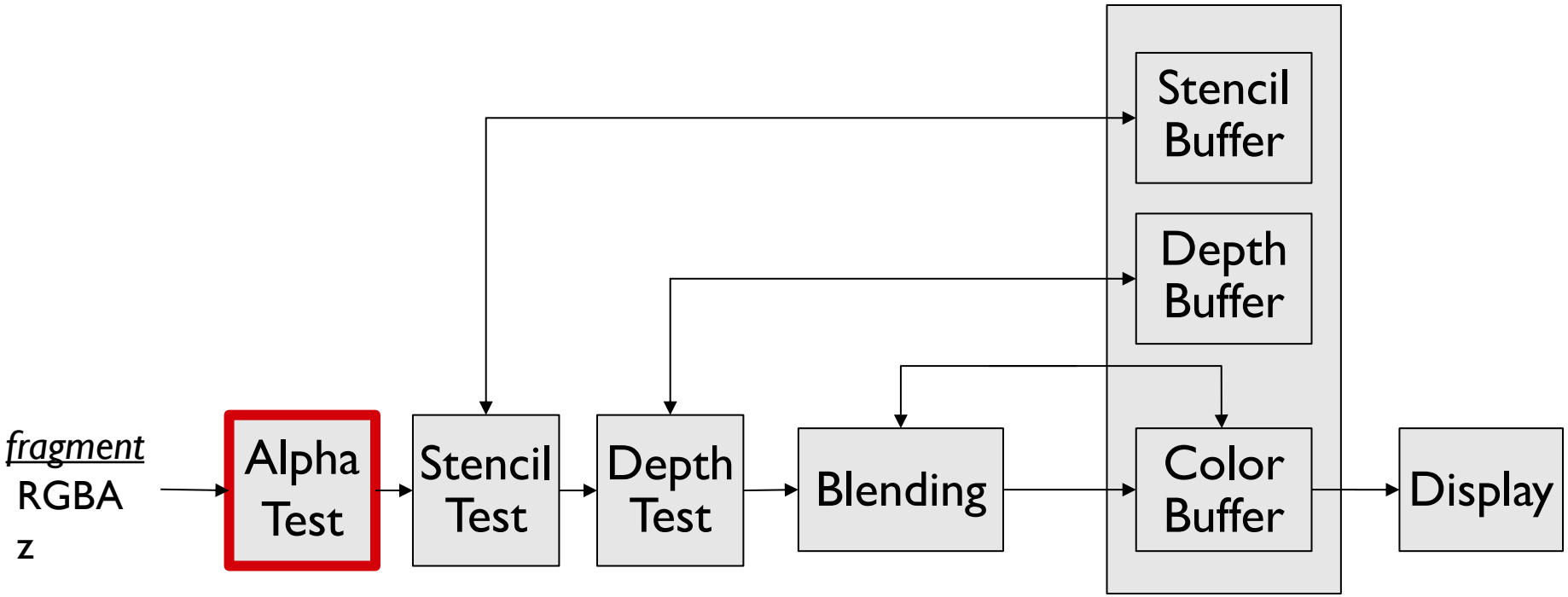
Per fragment operations



Depth test


fragment
RGBA
z
Alpha Test
Stencil Test
Depth Test
Blending
Stencil Buffer
Depth Buffer
Color Buffer
Display

Alpha test


fragment
RGBA
z
Alpha Test
Stencil Test
Depth Test
Blending
Color Buffer
Stencil Buffer
Depth Buffer
Display

Alpha test

Supprime le fragment si la **composante transparence** (alpha) ne satisfait pas un certain critère

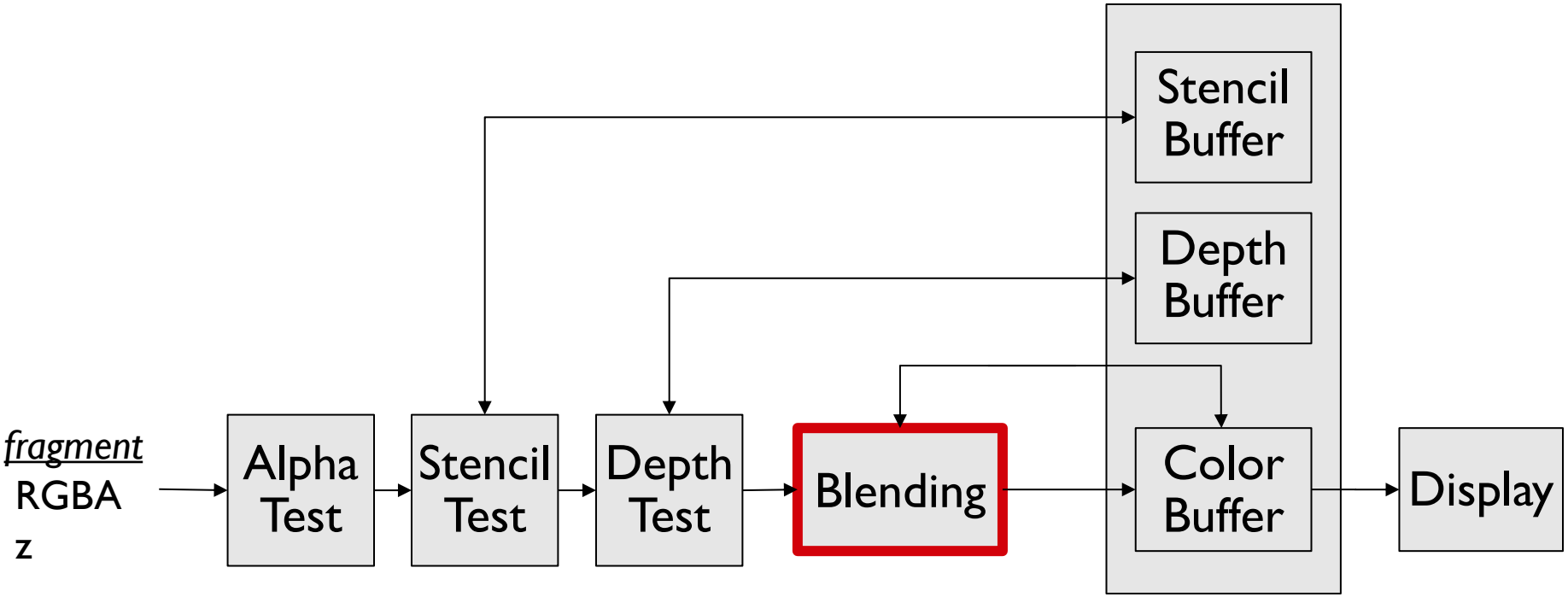
- Critère = **comparaison à une valeur de référence**
- Ex : grillage, végétation (herbe, feuilles d'arbres)…

En OpenGL :

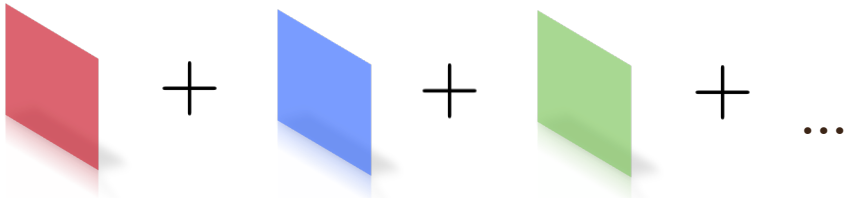
- `glAlphaFunc(GLenum func, GLclampf ref)`
- func = `GL_NEVER, GL_ALWAYS, GL_LESS, GL_GREATER, GL_EQUAL, GL_NOTEQUAL, GL_LEQUAL, GL_GEQUAL`
- Activé par : `glEnable/Disable(GL_ALPHA_TEST)`

http://www.tomshardware.fr/

Compositing / Blending



Compositing / Blending

couleur du pixel =  + ...

Compositing / Blending

Les fragments **sont mélangés** avec les pixels du **color buffer :**

$$C_{pixel} = C_{frag} \times factor_{src} + C_{pixel} \times factor_{dst}$$

Permet de simuler la transparence

En OpenGL :

- `glBlendFunc(GLenum source, GLenum destination)`
- Activé par : `glEnable/Disable(GL_BLEND)`

Contrôle du mélange

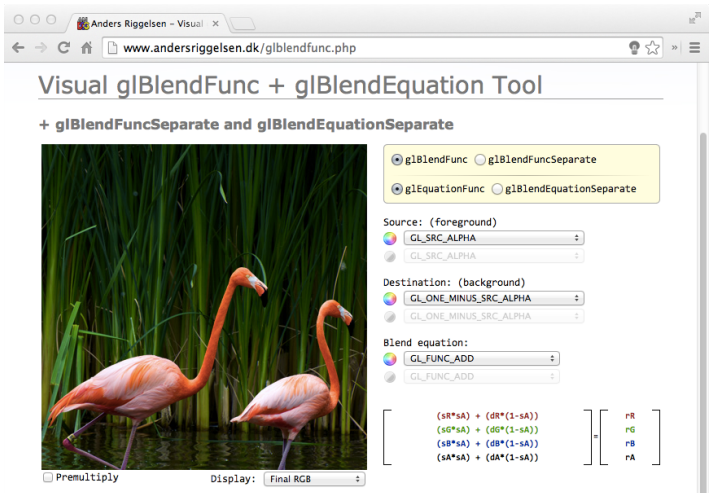
$$RGB_{pixel} = func(RGB_{frag} \times fRGB_{src}, RGB_{pixel} \times fRGB_{dst})$$

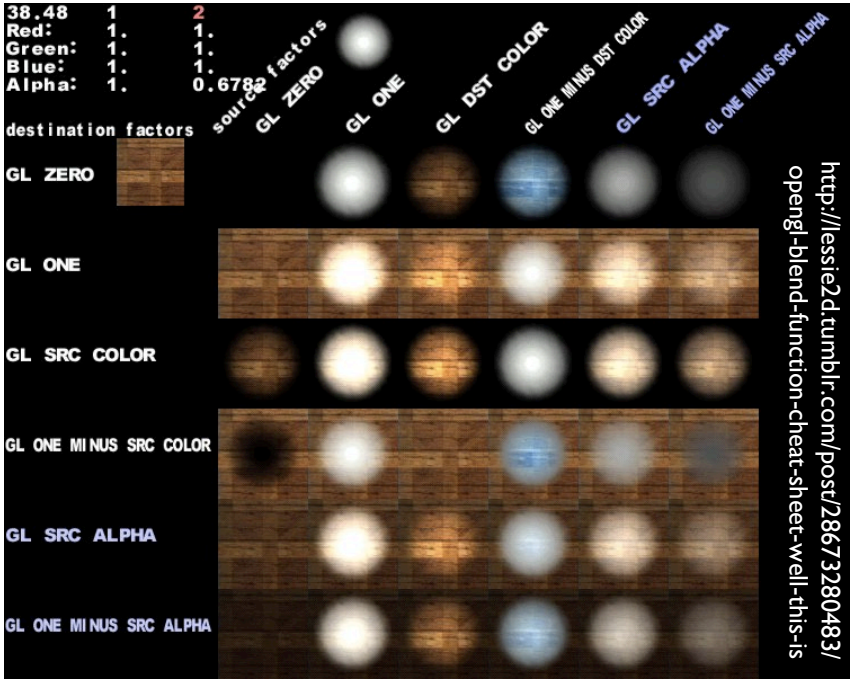
$$A_{pixel} = func(A_{frag} \times fA_{src}, A_{pixel} \times fA_{dst})$$

Avec :

func = *MIN,MAX,FUNC_ADD,FUNC_SUBTRACT,FUNC_REVERSE_SUBTRACT*

f* = *GL_{SRC, DST, CONSTANT}_{ALPHA, COLOR}*
GL_ONE_MINUS_{SRC, DST, CONSTANT}_{ALPHA, COLOR}





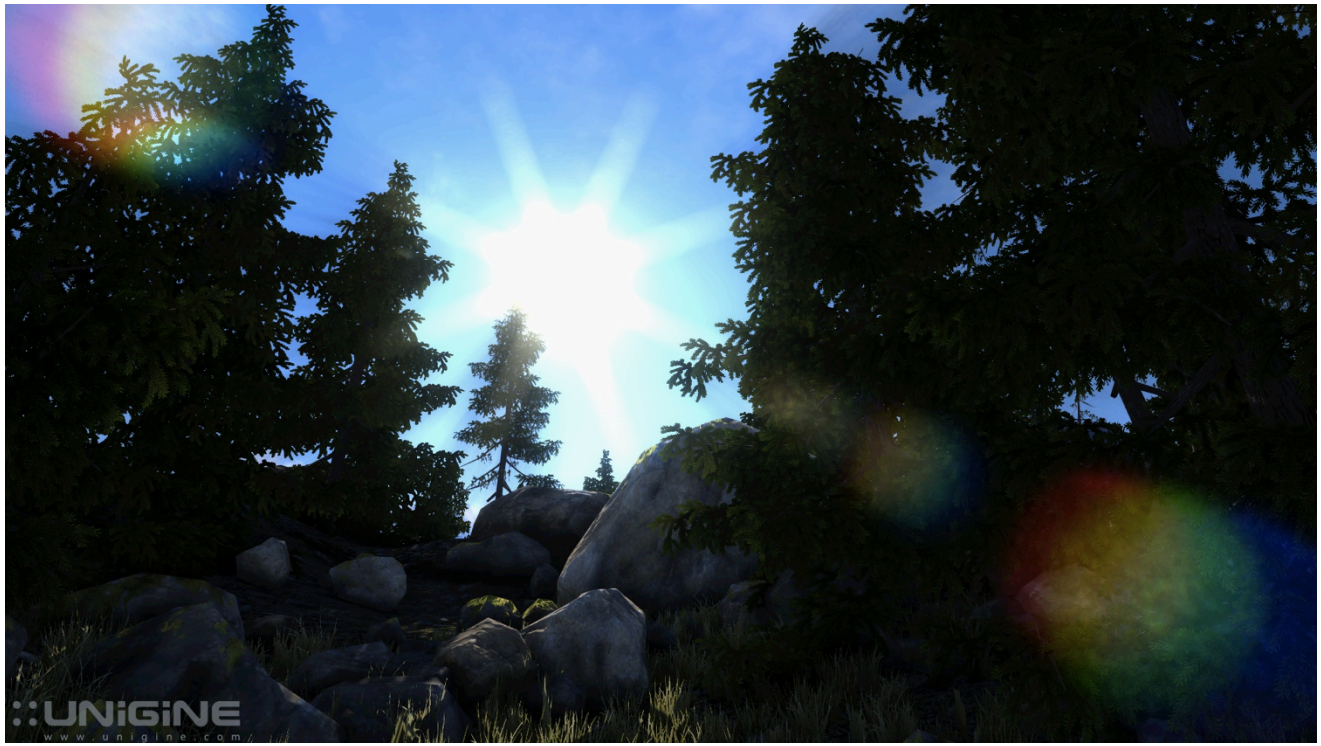
Cf. http://www.andersriggelsen.dk/glblendfunc.php

Rendu des objets transparents

Addition

$$C_{pixel} = C_{pixel} + C_{frag}$$

Utilisé pour les phénomènes lumineux (halos, lumière, fantômes,...)



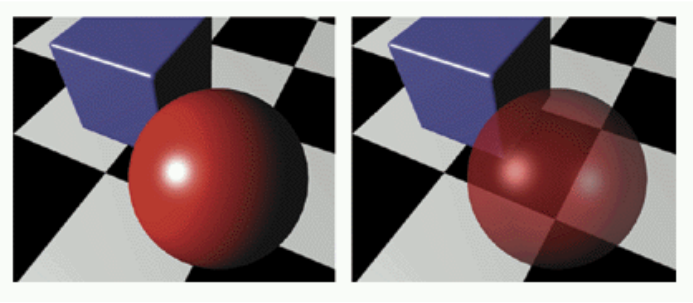
Rendu des objets transparents

Alpha-blending [Porter and Duff 1984]

$$\begin{cases} C_{pixel\_a} = C_{frag\_\alpha} + C_{pixel\_\alpha}(1 - C_{frag\_\alpha}) \\ C_{pixel\_rgb} = (C_{frag\_rgb}\, C_{frag\_\alpha} + C_{pixel\_rgb}\, C_{pixel\_\alpha}\, (1 - C_{frag\_\alpha}))/C_{pixel\_a} \end{cases}$$

En OpenGL :

- `glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)`
- Utilise l'alpha du fragment



Rendu des objets transparents

Alpha-blending [Porter and Duff 1984]

$$\begin{cases} C_{pixel\_a} = 1 \\ C_{pixel\_rgb} = C_{frag\_rgb}\, C_{frag\_\alpha} + C_{pixel\_rgb}\,(1 - C_{frag\_\alpha}) \end{cases}$$

En OpenGL :

- `glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)`
- Utilise l'alpha du fragment

Alpha-blending

Alpha-blending

$$C_0 = \alpha_0 c_0$$

$$C_n = \alpha_n c_n + (1 - \alpha_n) C_{n-1}$$

Rendu des objets transparents

Alpha-blending [Porter and Duff 1984]

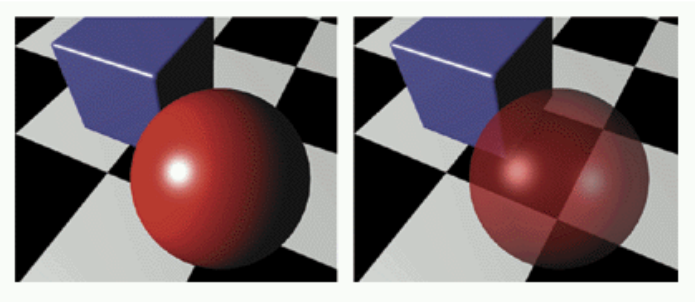
$$C_{pixel\_rgb} = C_{frag\_rgb}\, C_{frag\_\alpha} + C_{pixel\_rgb}\,(1 - C_{frag\_\alpha})$$

En OpenGL :

- `glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)`
- Utilise l'alpha du fragment

Méthode de rendu :

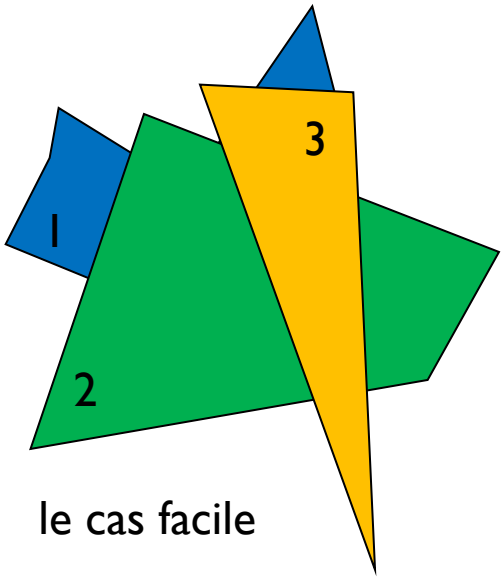
1. tracer les objets opaques
2. activer le blending
3. tracer les objets transparents
   du plus profond au moins profond
   (algorithme du peintre)



Algorithme du peintre

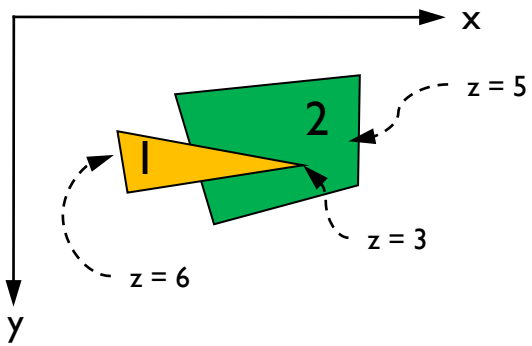
« Peindre » les polygones suivant un **ordre de distance décroissante** au point de vue

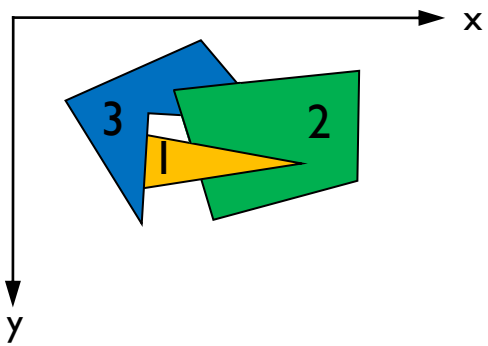
- **Trier** suivant les z décroissants dans le repère de la camera
  ⟹ meilleur coût : O(n log(n))
- Résoudre les **ambiguïtés** dues aux recouvrements

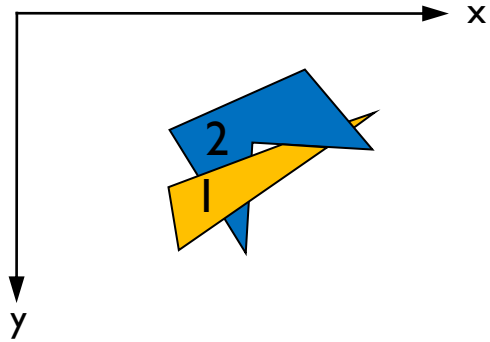


le cas facile

cas ambigus







Rendu des objets transparents

Alpha-Blending sans tri

Order-Independent Transparency

Order-Independent Transparency

Objectif :

ne pas avoir à trier les objets (triangles)

Solutions :

Order-Independent Transparency

Objectif :

ne pas avoir à trier les objets (triangles)

Solutions :

$\Longrightarrow$ trier les **fragments** (visibilité implicite)

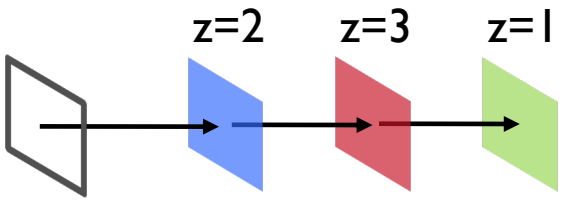
ou

$\Longrightarrow$ calculer explicitement la **fonction de visibilité**

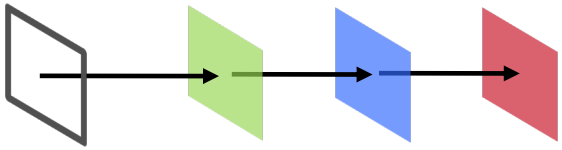
couleur du pixel = $\sum_{i=0}^{n} c_i \, \alpha_i \, vis(z_i)$

A-buffer [Carpenter 1984, Yang et al. 2009]

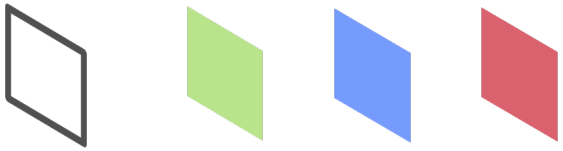
1. **Pour chaque pixel, liste** des fragments (couleur et z)

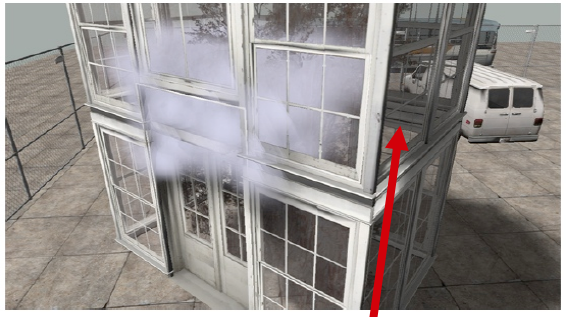


1. **Trie** de la liste sur la **profondeur**



2. **Alpha-blending** des fragments





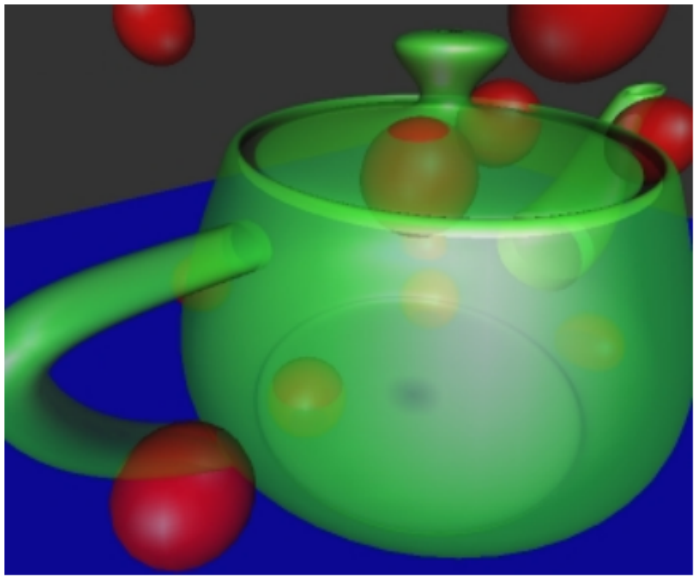
⟹ Pas de limite sur la quantité de mémoire nécessaire

Depth Peeling [Everitt 2001]

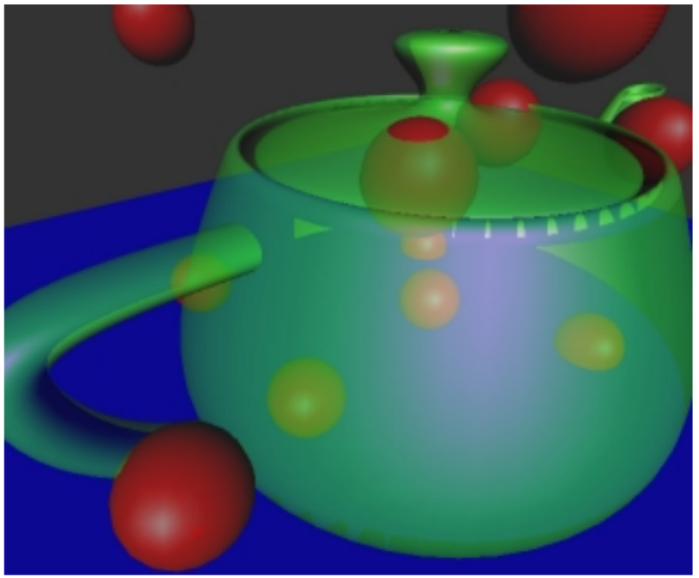
Rendu de la scène en **plusieurs passes** : **n *layers***

1. Rendu normal ⟹ garder le *depth buffer*
2. Nouveau rendu :
   - ignorer le fragment si son z ≤ *depth buffer*
   - générer un nouveau *depth buffer*

   ↩ itérer
3. Alpha-blending des layers

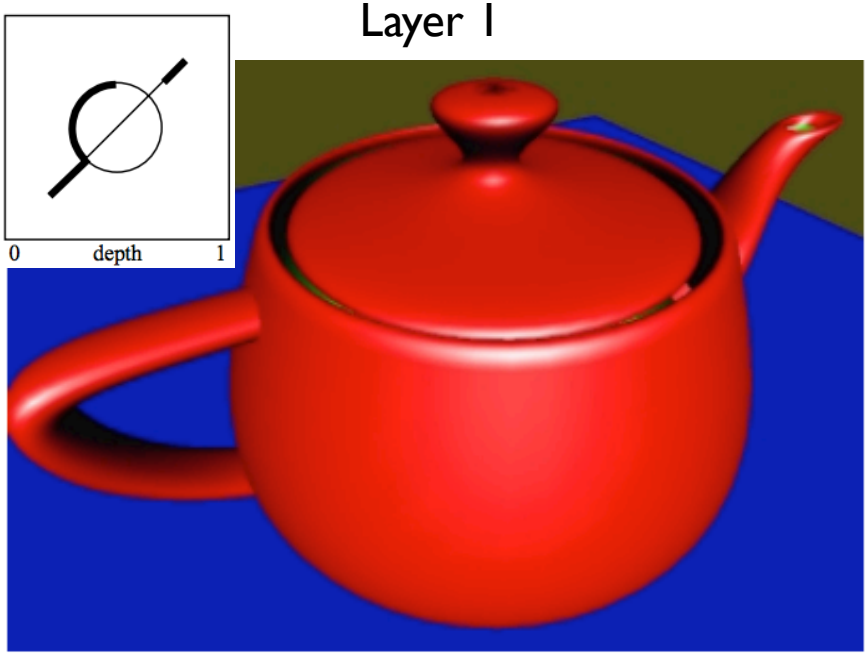


Transparence correcte

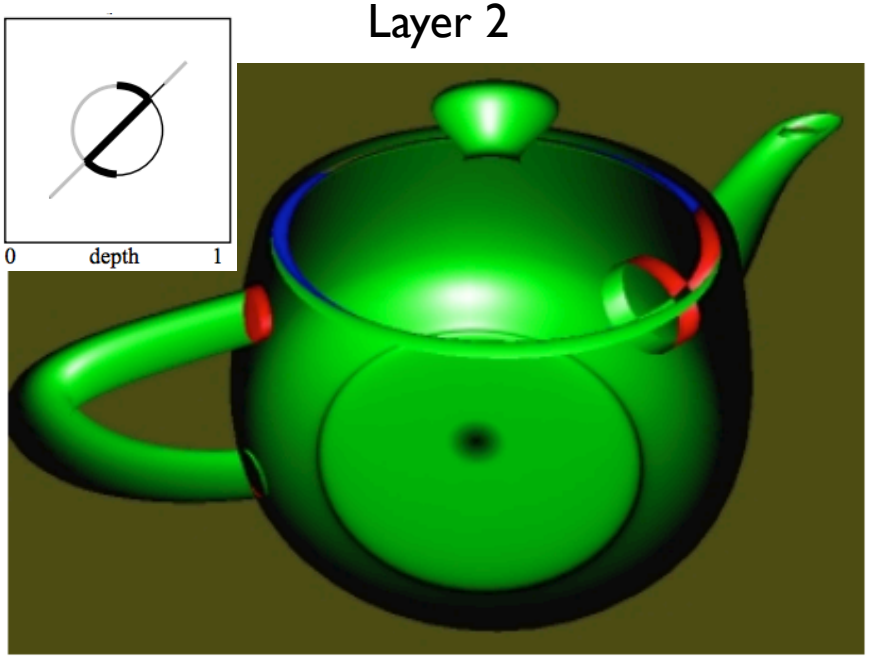


Transparence incorrecte

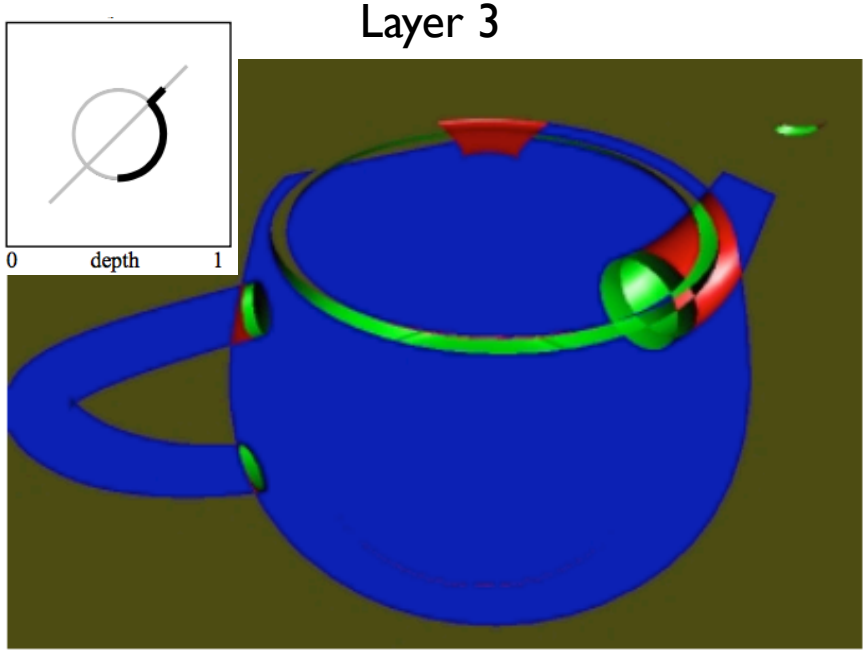
Layer 1



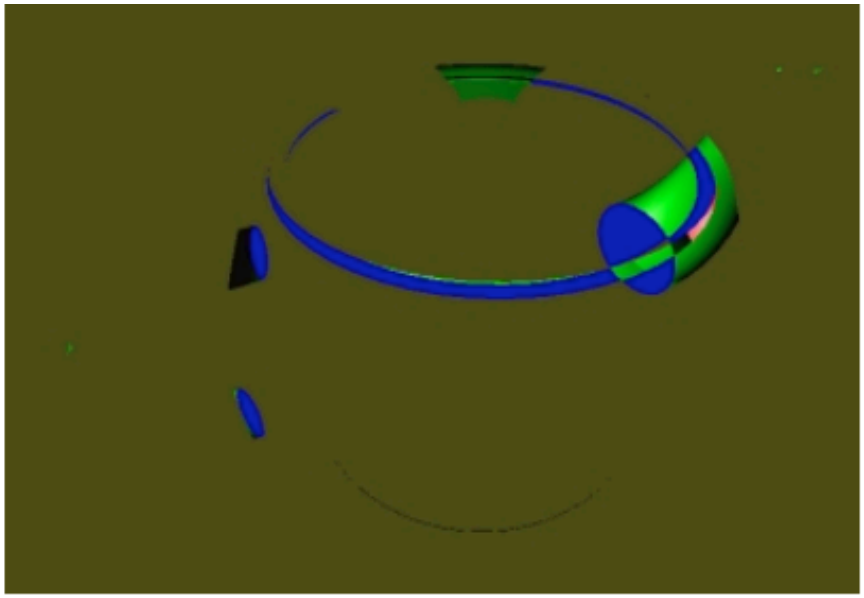
Layer 2



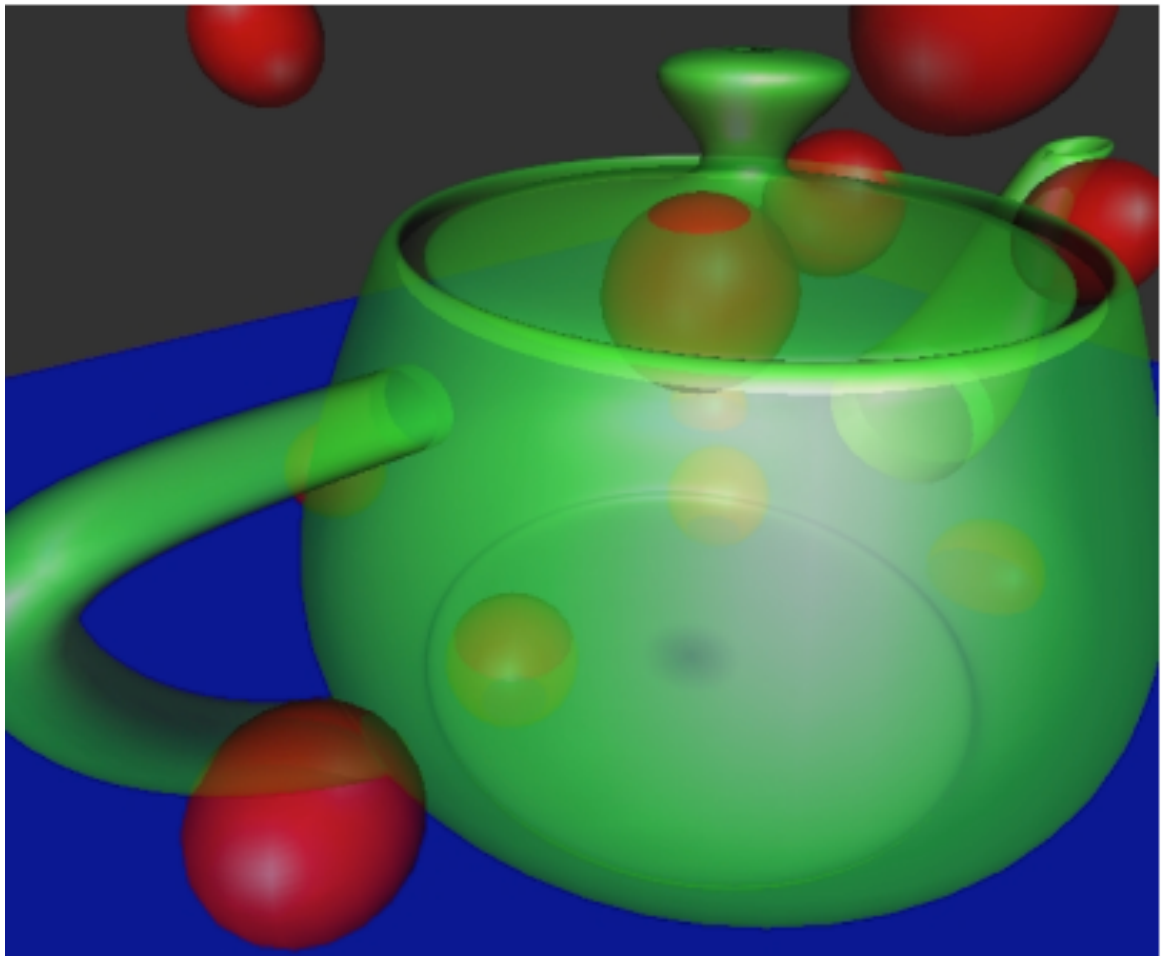
Layer 3



Layer 4



Depth Peeling [Everitt 2001]



4 layers

Depth Peeling [Everitt 2001]

Rendu de la scène en **plusieurs passes** : **n *layers***

1. Rendu normal ⟹ garder le *depth buffer*
2. Nouveau rendu :
   - ignorer le fragment si son z ≤ *depth buffer*
   - générer un nouveau *depth buffer*

   ↩ itérer
3. Alpha-blending des layers

⟹ **2 *depth buffers* nécessaires (FBO)**

⟹ Rendu correcte quand n ⟶ ∞