http://tiny.cc/ift3355

IFT 3355: INFOGRAPHIE

REVUE 2

Mikhail Bessmeltsev

LES LIGNES ET LES COURBES

•Les fonction explicites: les coordonnées sont des fonctions des autres

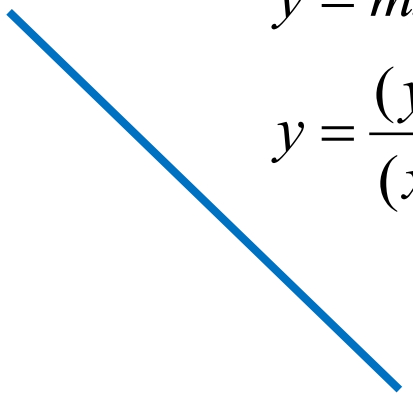
$$y = f(x)$$

$$z = f(x, y)$$

Line

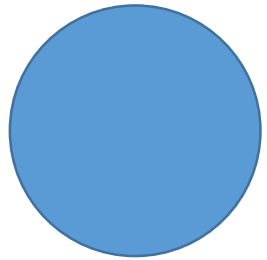
$$y = mx + b$$

$$y = \frac{(y_2 - y_1)}{(x_2 - x_1)}(x - x_1) + y_1$$



Circle

$$y = \pm\sqrt{r^2 - x^2}$$



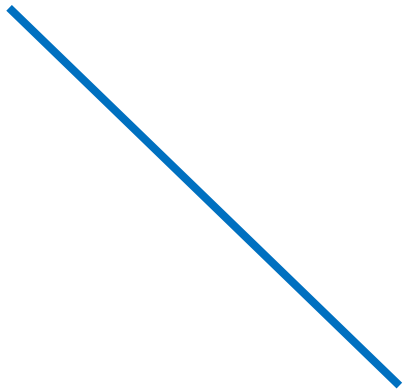
LES LIGNES ET LES COURBES

• Les fonctions paramétriques: toutes les coordonnées sont définies en fonction des valeurs de paramètres

$$(x, y) = (f_1(t), f_2(t))$$

$$(x, y, z) = (f_1(u, v), f_2(u, v), f_3(u, v))$$

Une ligne

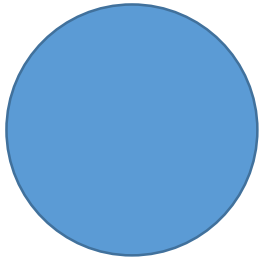


$$x(t) = x_1 + t(x_2 - x_1)$$

$$y(t) = y_1 + t(y_2 - y_1)$$

$$t \in [0,1]$$

Un Cercle



$$x(\theta) = r\cos(\theta)$$

$$y(\theta) = r\sin(\theta)$$

$$\theta \in [0,2\pi]$$

LES LIGNES ET LES COURBES

• Les fonctions implicites: l'objet est défini par les racines de la fonction

$$\{(x,y) : F(x,y) = 0\}$$

$$\{(x,y,z) : F(x,y,z) = 0\}$$

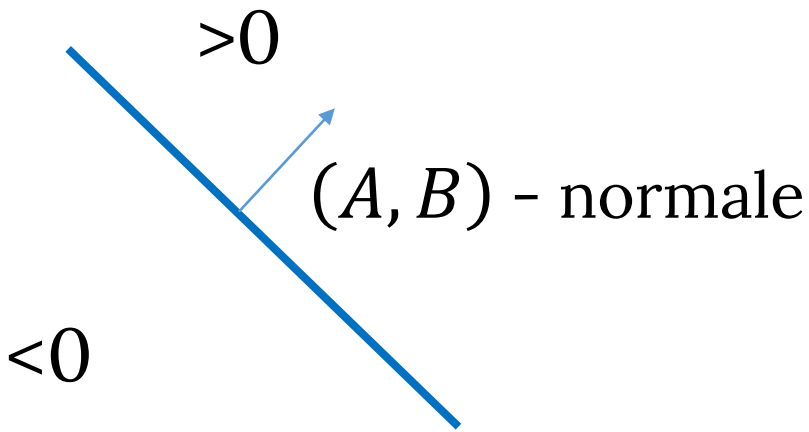
- Diviser l'espace

$$\{(x,y) : F(x,y) > 0\}, \{(x,y) : F(x,y) = 0\}, \{(x,y) : F(x,y) < 0\}$$

LES FONCTIONS IMPLICITES

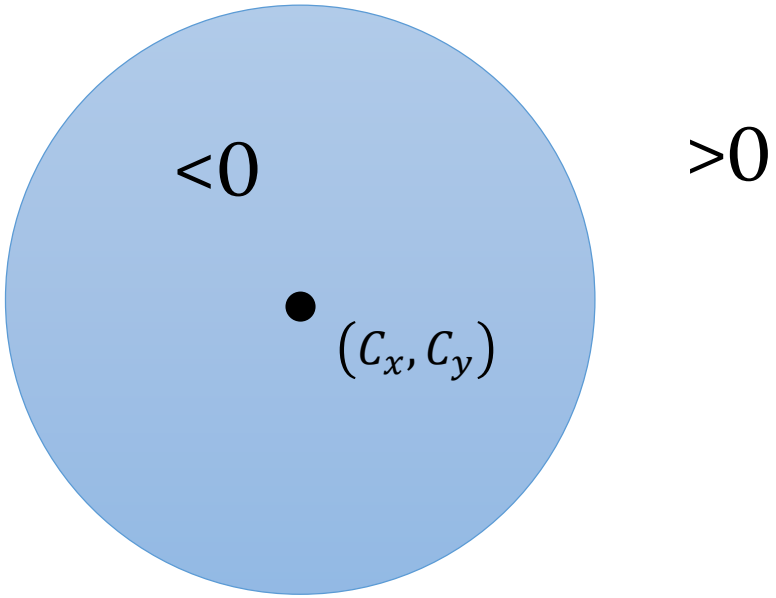
Une ligne

$$Ax + By + C = 0$$



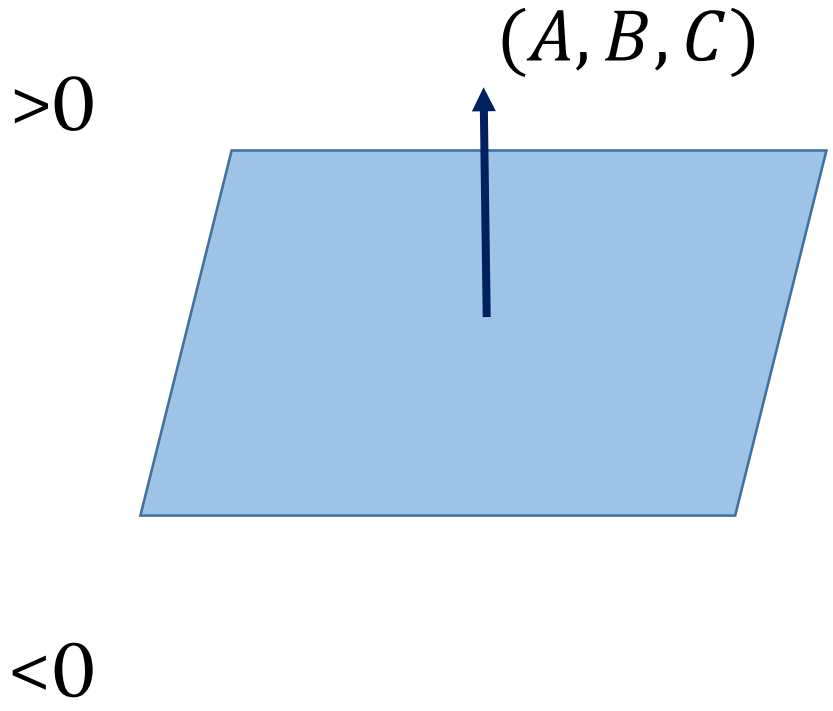
Un cercle

$$(x - C_x)^2 + \left(y - C_y\right)^2 - r^2 = 0$$



LE PLAN - IMPLICITE

$$Ax + By + Cz + D = 0$$

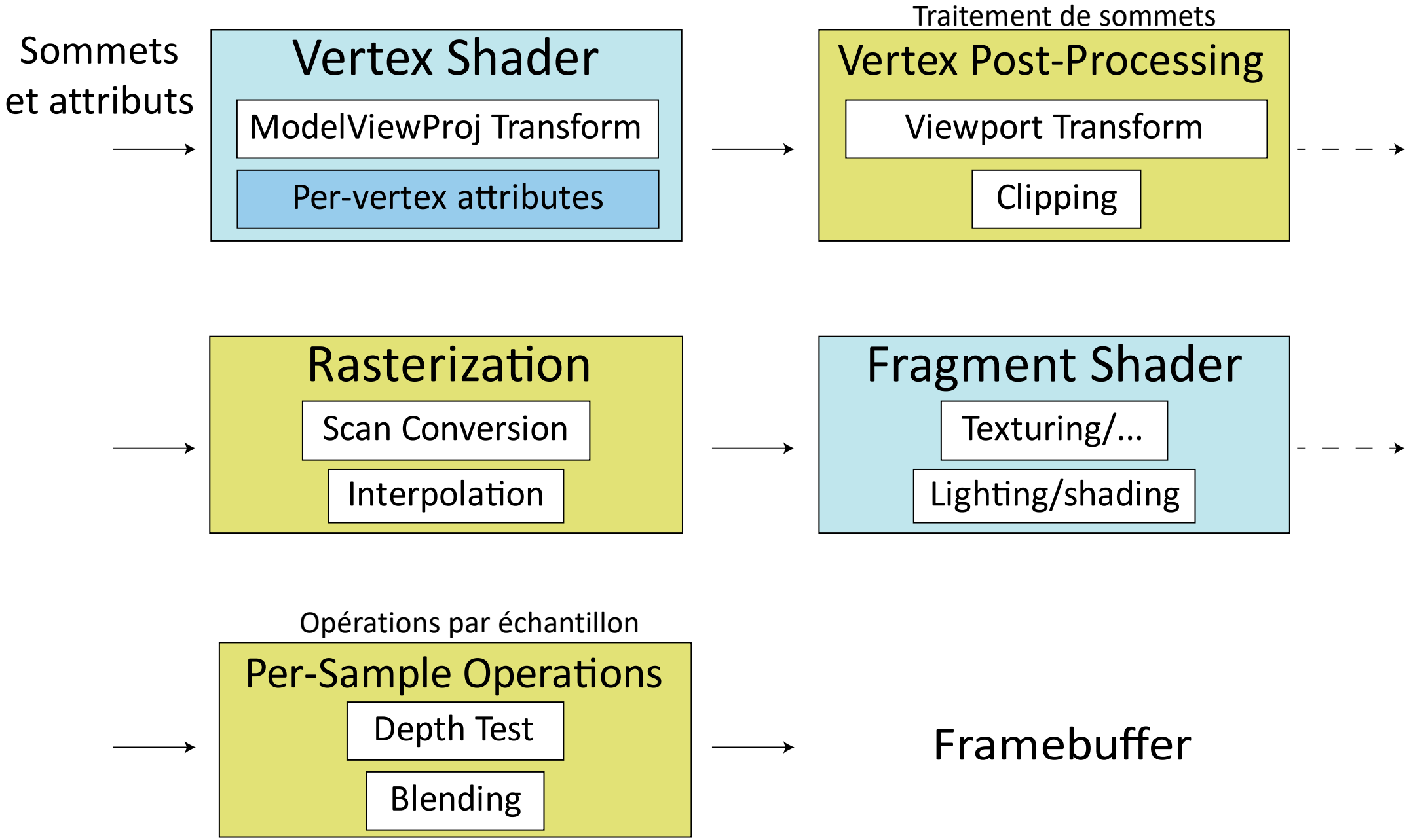


UNE FONCTION IMPLICITE ARBITRAIRE

$$F(x, y, z) = 0$$

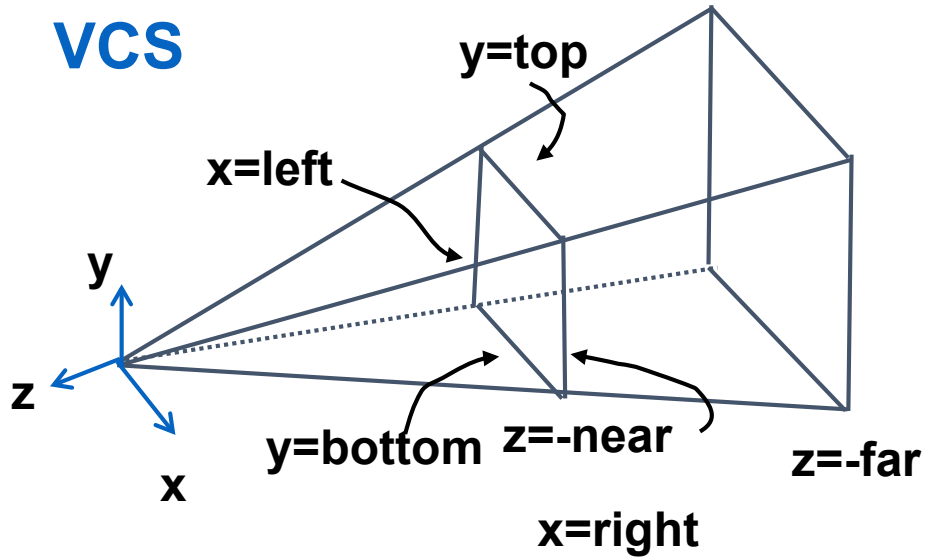
$$n(x, y, z) = \nabla F(x, y, z) = \begin{pmatrix} \partial F(x, y, z)/\partial x \\ \partial F(x, y, z)/\partial y \\ \partial F(x, y, z)/\partial z \end{pmatrix}$$

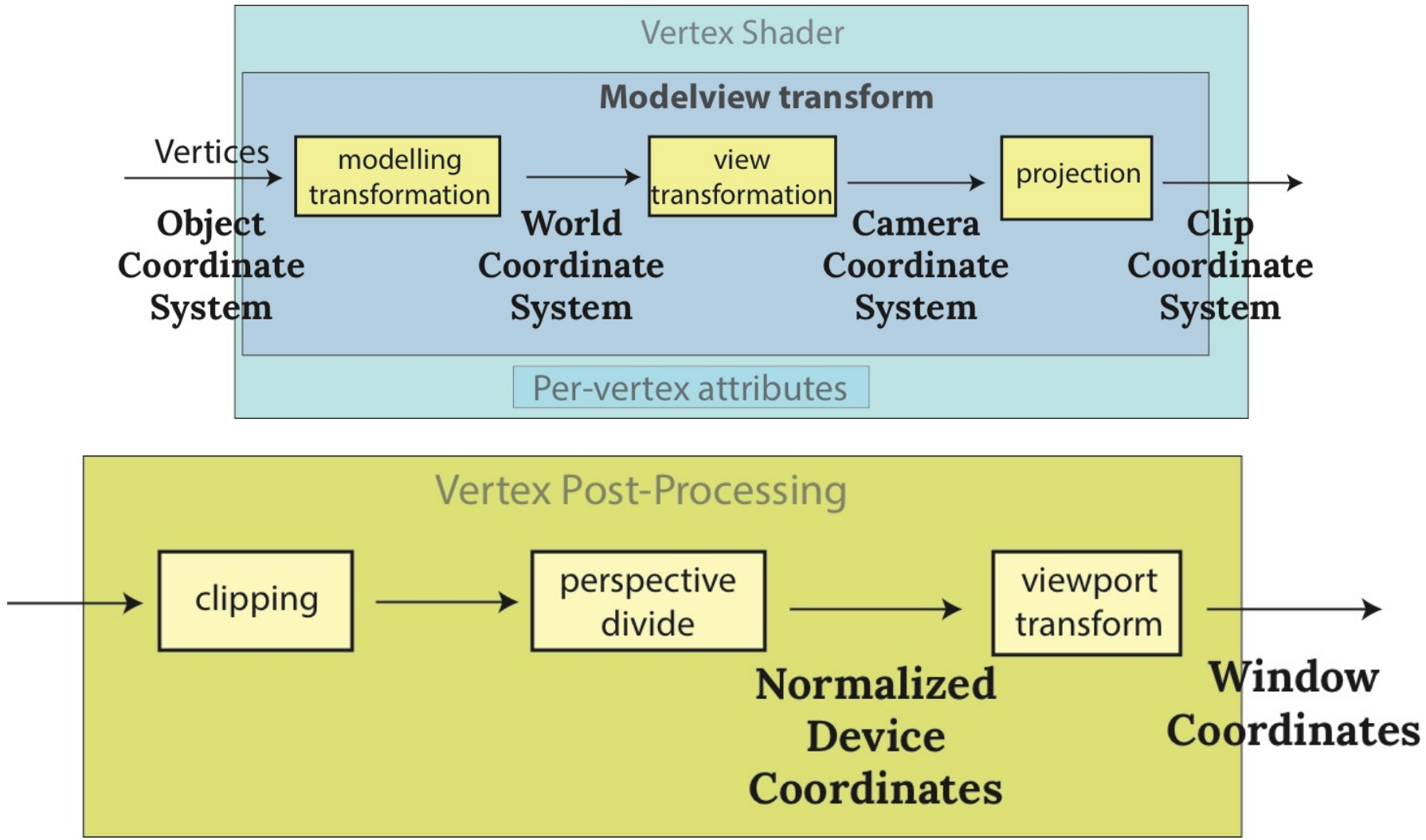
PIPELINE: PLUS DE DÉTAILS



CLIPPING

- Il faut découper tout ce qui est hors du volume de vue
- Hors du plan gauche/droit, supérieur/inférieur
- Et plus important, avant/arrière:




Vertex Shader
Modelview transform
Vertices
modelling transformation
view transformation
projection
Object Coordinate System
World Coordinate System
Camera Coordinate System
Clip Coordinate System
Per-vertex attributes
Vertex Post-Processing
clipping
perspective divide
viewport transform
Normalized Device Coordinates
Window Coordinates

LA MATRICE DE LA FENÊTRE (*VIEWPORT)*

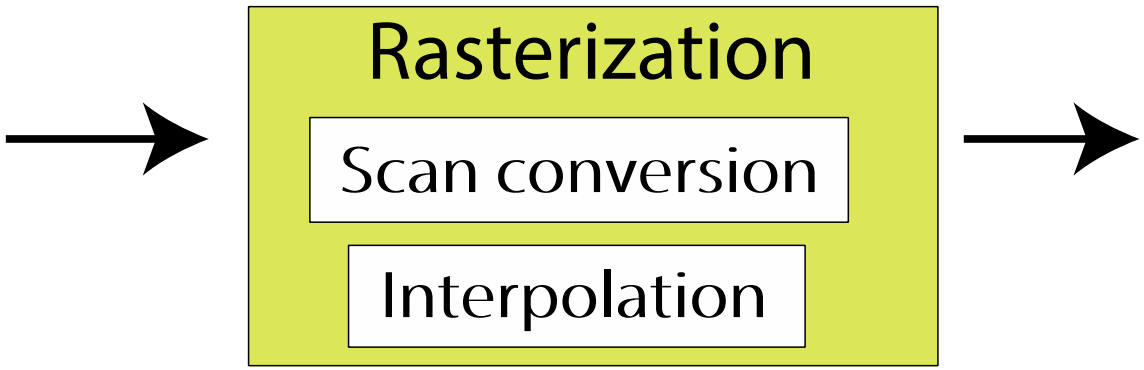
- Qu'est-ce qu'elle fait?

$$\begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{W}{2} & 0 & 0 & \frac{W-1}{2} \\ 0 & \frac{H}{2} & 0 & \frac{H-1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_n \\ y_n \\ z_n \\ 1 \end{bmatrix}$$

RASTERIZATION

- C'est une partie du pipeline fixe

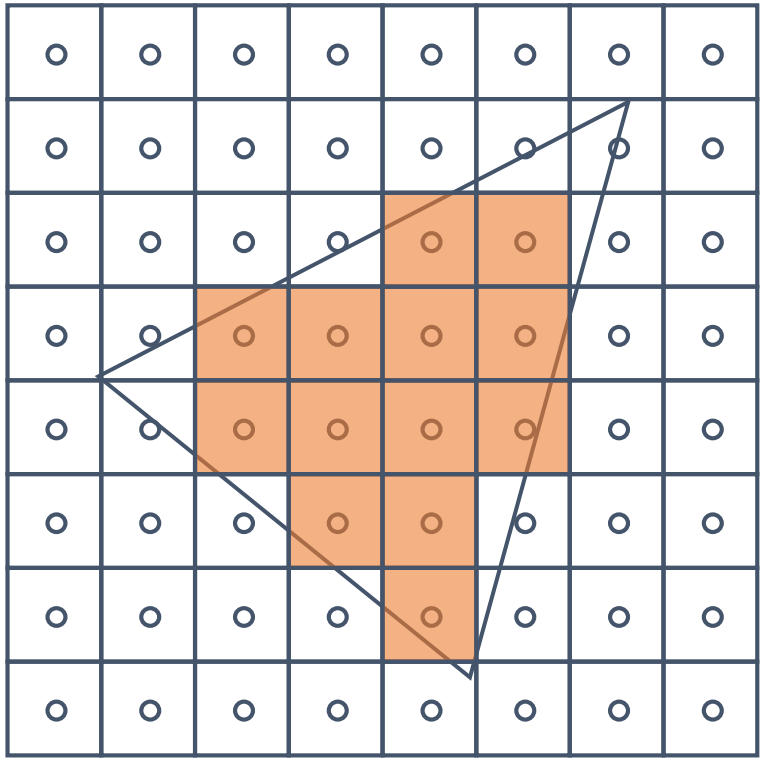
- Les données en entrée: tous les polygone clippés
- Les données en sortie: les fragments (avec les **varying** variables interpolées)

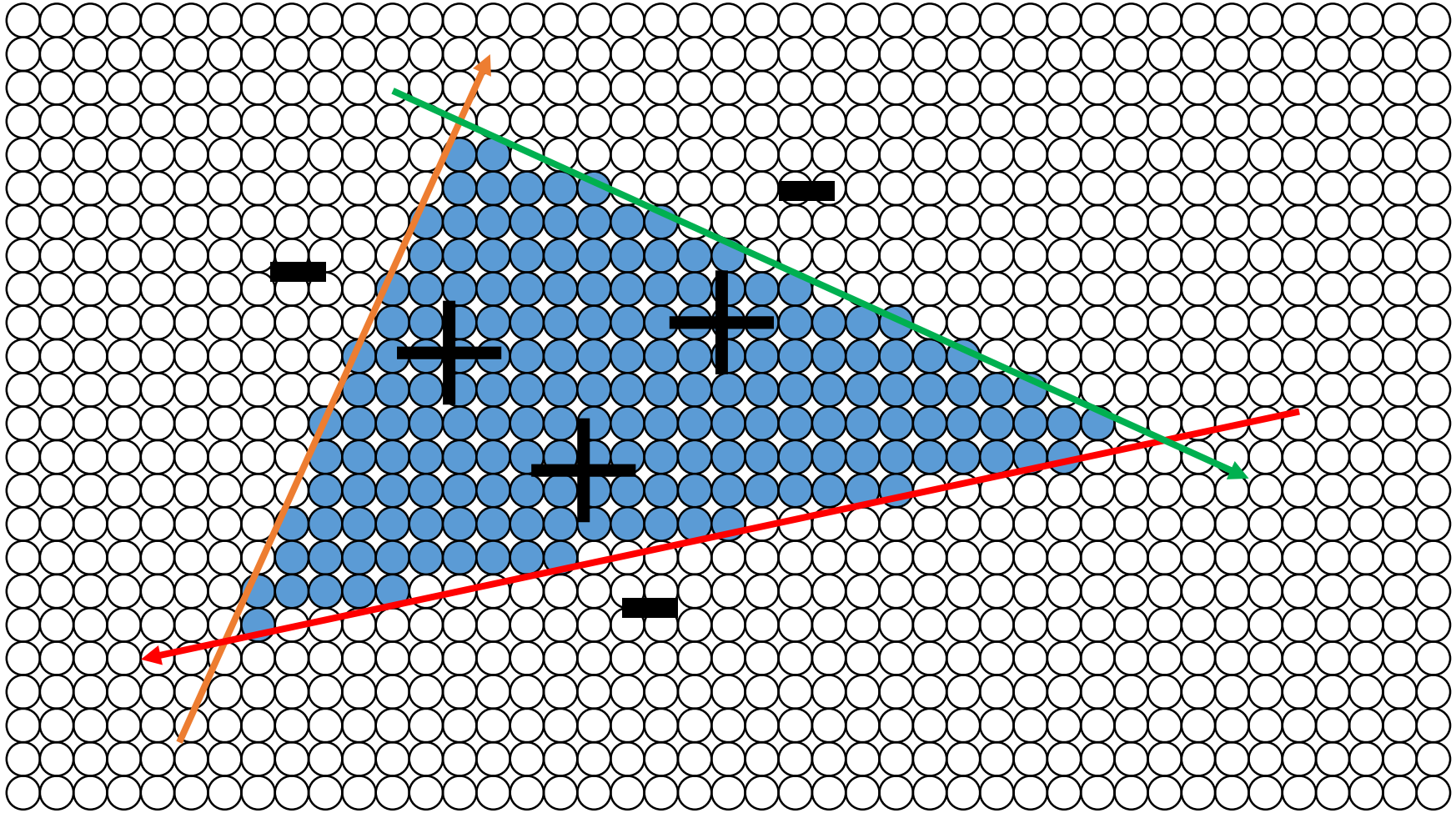


COMMENT VÉRIFIER SI UN PIXEL EST À L'INTÉRIEUR?

Un point est à l'intérieur ⇔

$$A_i x + B_i y + C > 0, i = 1, \dots, 3$$





SCANLINE: CODE

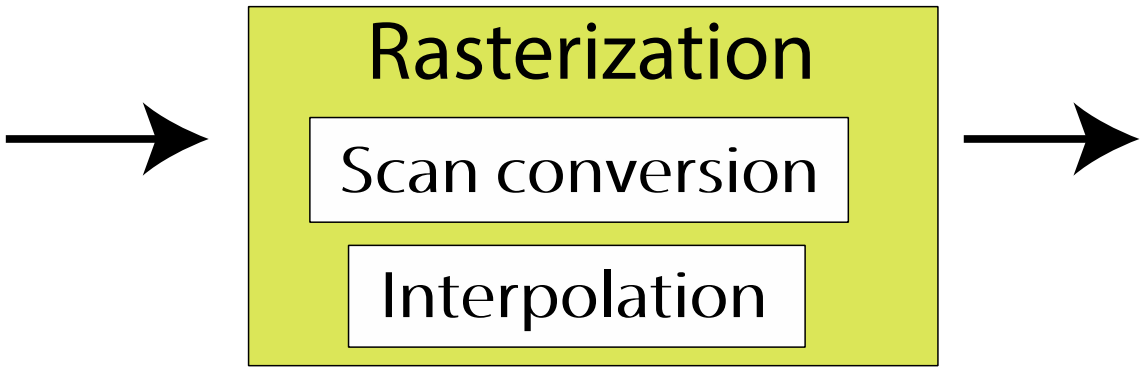
```
findBoundingBox(xmin, xmax, ymin, ymax);
setupEdges (a0,b0,c0,a1,b1,c1,a2,b2,c2);

for (int y = yMin; y <= yMax; y++) {
  for (int x = xMin; x <= xMax; x++) {
    float e0 = a0*x + b0*y + c0;
    float e1 = a1*x + b1*y + c1;
    float e2 = a2*x + b2*y + c2;
    if (e0 > 0 && e1 > 0 && e2 > 0)
        Image[x][y] = TriangleColor;
  }
}
```

SCAN CONVERSION

- What are problems of scan conversion?
- How to find a bounding box?
- How to scan-convert an arbitrary polygon?

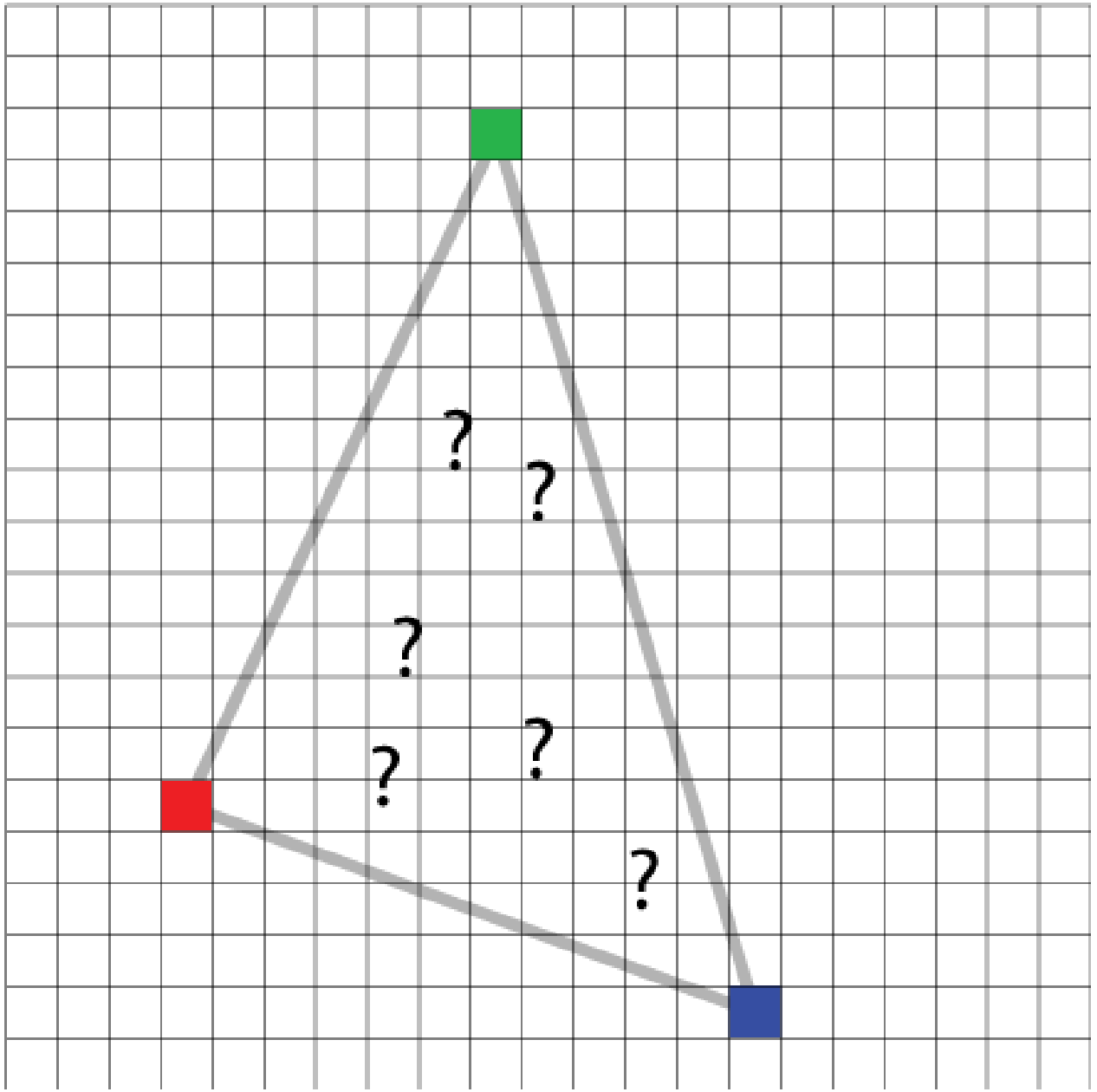
INTERPOLATION

- What does it do?



L'INTERPOLATION : CALCULER LES VALEURS ENTRE CELLES QU'ON CONNAÎT

- On interpole:
  - z
  - r, g, b – les composants de la couleur
  - u, v - les coordonnées de la texture
  - (n_x, n_y, n_x) – les composants de la normale
- Les méthodes suivantes sont équivalentes:
  - Les coordonnées barycentriques
  - L'interpolation bilinéaire
  - L'interpolation du plan

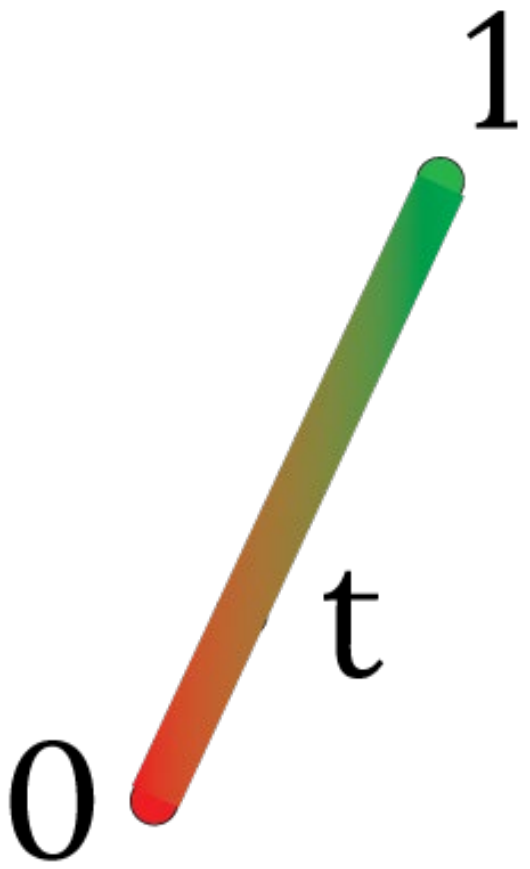


PLUS FACILE:

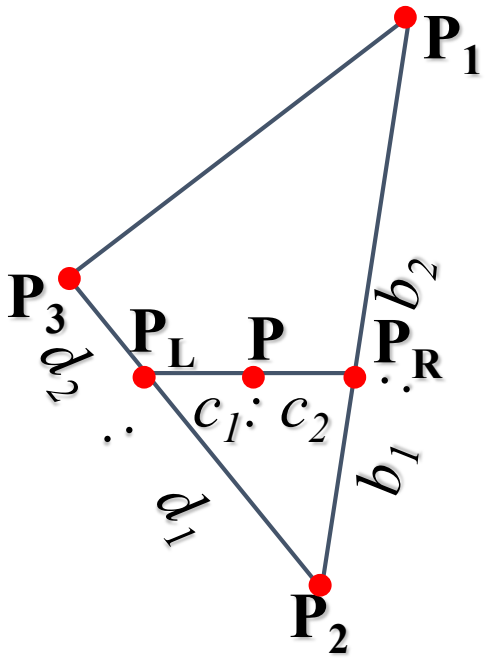
Comment interpoler la couleur entre les deux sommets?

$$c(t) = c(0) \cdot (1 - t) + c(1) \cdot t$$

Interpolation linéaire



L'INTERPOLATION BILINÉAIRE



$$P = \frac{c_2}{c_1 + c_2} \cdot P_L + \frac{c_1}{c_1 + c_2} \cdot P_R$$

$$P_L = \frac{d_2}{d_1 + d_2} P_2 + \frac{d_1}{d_1 + d_2} P_3$$

$$P_R = \frac{b_2}{b_1 + b_2} P_2 + \frac{b_1}{b_1 + b_2} P_1$$

$$P = \frac{c_2}{c_1 + c_2}\left(\frac{d_2}{d_1 + d_2} P_2 + \frac{d_1}{d_1 + d_2} P_3\right) + \frac{c_1}{c_1 + c_2}\left(\frac{b_2}{b_1 + b_2} P_2 + \frac{b_1}{b_1 + b_2} P_1\right)$$

LES COORDONNÉES BARYCENTRIQUES

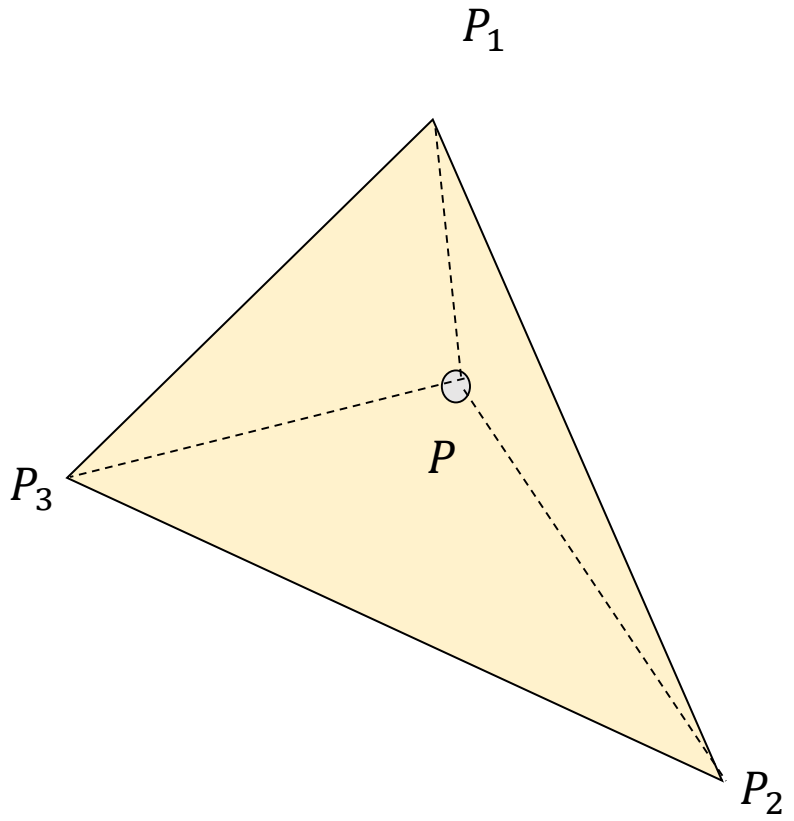
- L'aire

$$A = \frac{1}{2}\left\|\overrightarrow{P_1P}_2 \times \overrightarrow{P_1P_3}\right\|$$

- Les coordonnées barycentriques:

$$a_1 = A_{P_2P_3P}/A, a_2 = A_{P_3P_1P}/A,$$
$$a_3 = A_{P_1P_2P}/A,$$
$$P = a_1P_1 + a_2P_2 + a_3P_3$$

$$f(P) = a_1f(P_1) + a_2f(P_2) + a_3f(P_3)$$

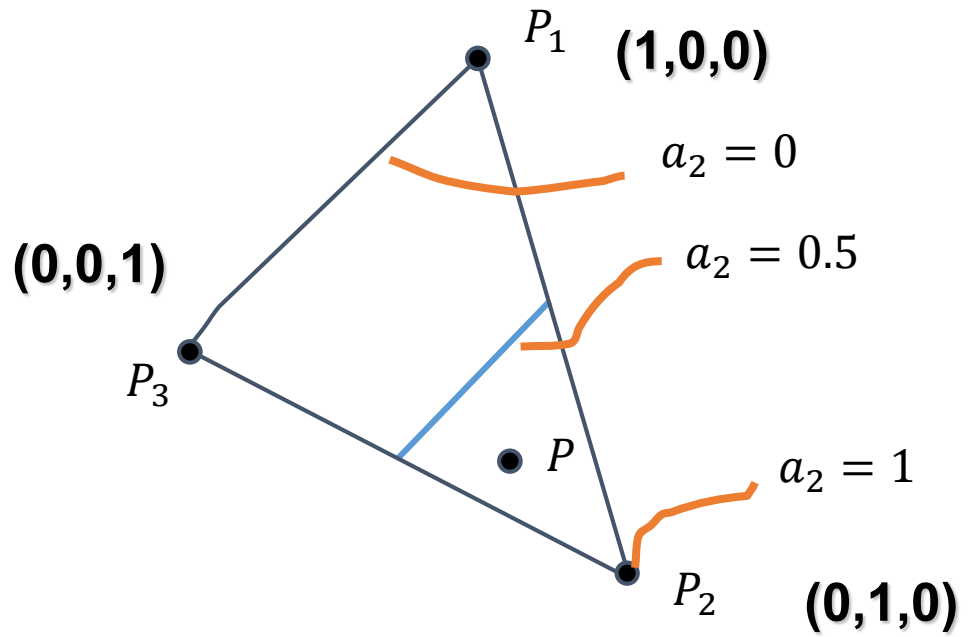


CHAQUE POINT À L'INTÉRIEUR DU TRIANGLE EST

- Une combinaison pondérée (affine) des sommets

$$P = a_1 \cdot P_1 + a_2 \cdot P_2 + a_3 \cdot P_3$$

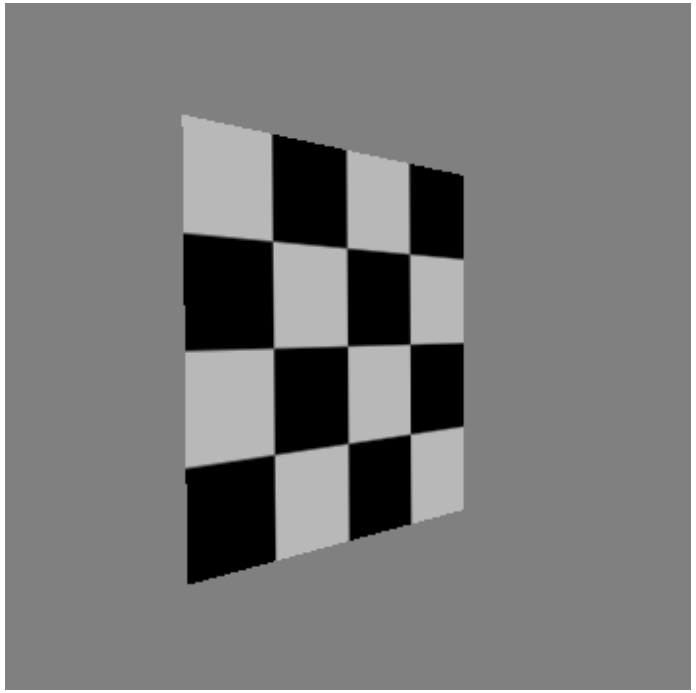
$$a_1 + a_2 + a_3 = 1$$
$$0 \leq a_1, a_2, a_3 \leq 1$$

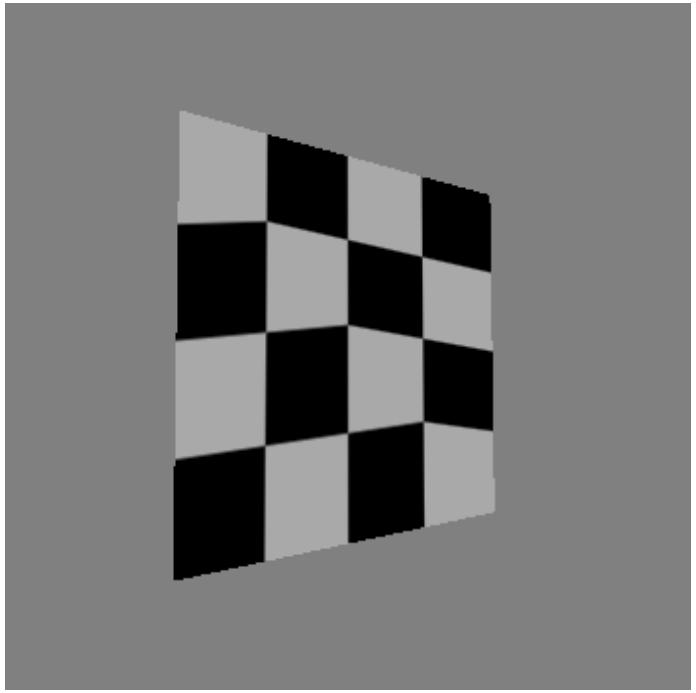


TEXTURE MAPPING

Après la projection perspective, l'interpolation linéaire dans l'espace de l'affichage est incorrecte

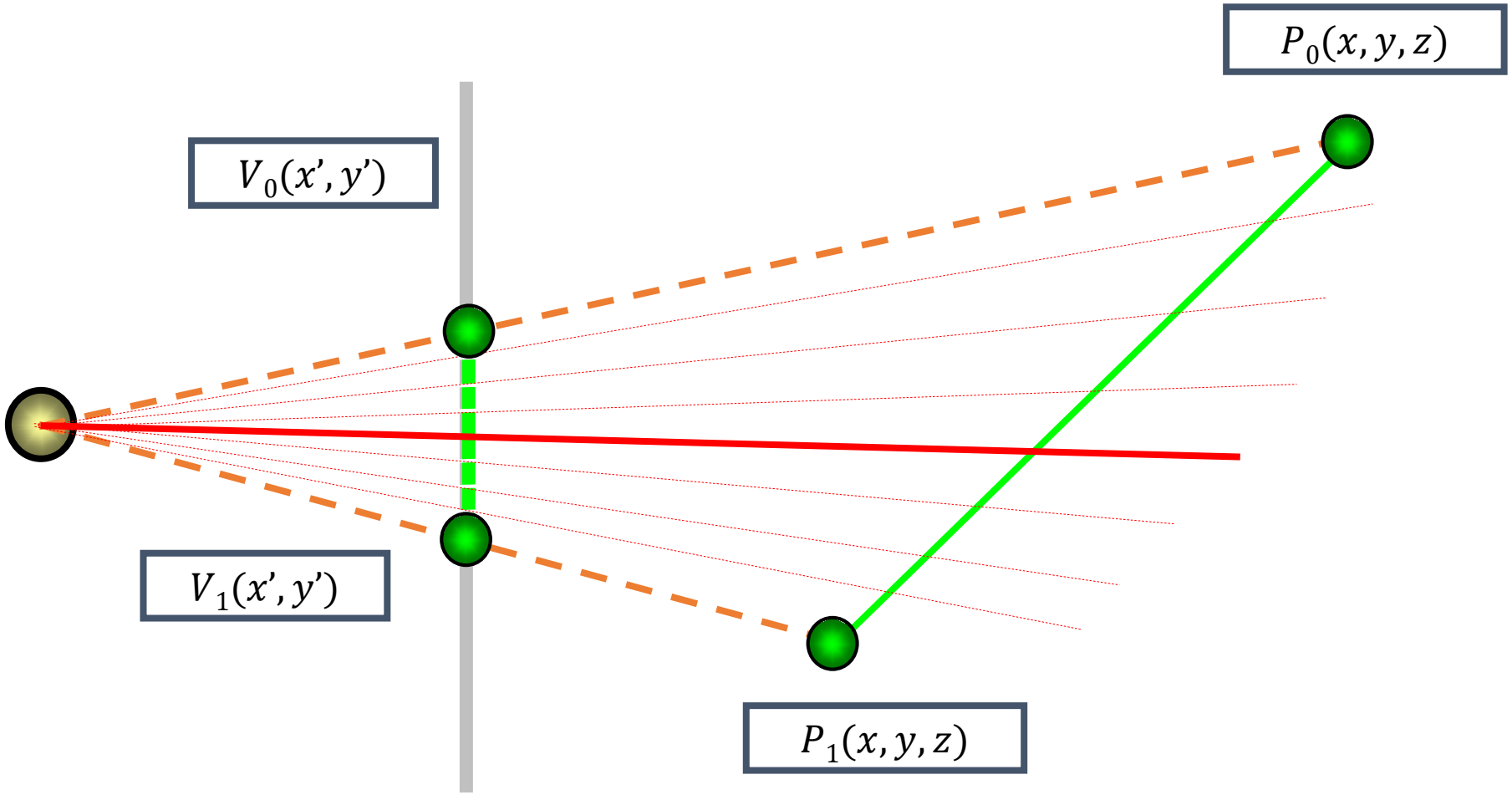
- Pour les textures, aussi bien que pour les couleurs, shading, etc.





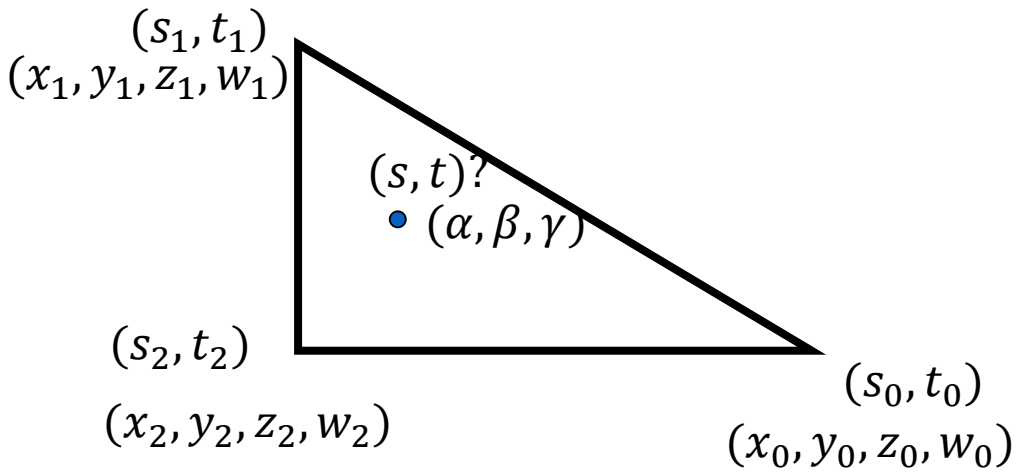
INTERPOLATION: L'ESPACE DE L'AFFICHAGE VS DU MONDE

- Normalement, on ignore le problème pour le *shading*, mais pour les textures les artefacts sont évidents



L'INTERPOLATION APRÈS LA PERSPECTIVE

- α, β, γ : les coordonnées barycentriques (2D) du point P
- s_0, s_1, s_2 : les coordonnées dans l'espace de la texture
- w_0, w_1, w_2 : les coordonnées homogènes des vecteurs



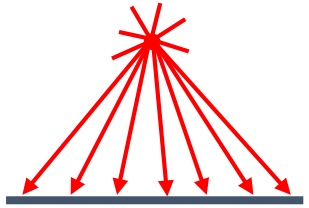
$$s = \frac{\alpha \cdot s_0/w_0 + \beta \cdot s_1/w_1 + \gamma \cdot s_2/w_2}{\alpha/w_0 + \beta/w_1 + \gamma/w_2}$$

- La même chose pour t

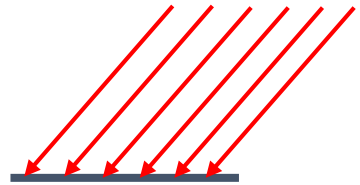
La dérivation (les triangles similaires):
https://www.comp.nus.edu.sg/~lowkl/publications/lowk_persp_interp_techrep.pdf

LES TYPES DE SOURCES DE LUMIIÈRE

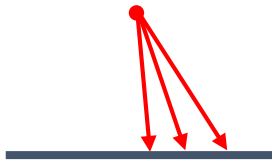
- Lumière ponctuelle
  - La lumière provient d'un point
  - définie par la position seulement



- Lumière directionnelle (=la lumière ponctuelle à l'infinité)
  - Les rayons sont parallèles
  - Les rayons frappent la surface au même angle
  - définie par la direction seulement



- Lumière ponctuelle de type *spotlight*
  - Une lumière ponctuelle mais avec les angles limités
  - définie par la position, la direction et l'intervalle des angles



LUMIÈRES

- La lumière a une couleur
- Elle interagit avec la couleur de la surface (r, g, b)

$$I = I_a k_a$$

$$I_a = (I_{ar}, I_{ag}, I_{ab})$$

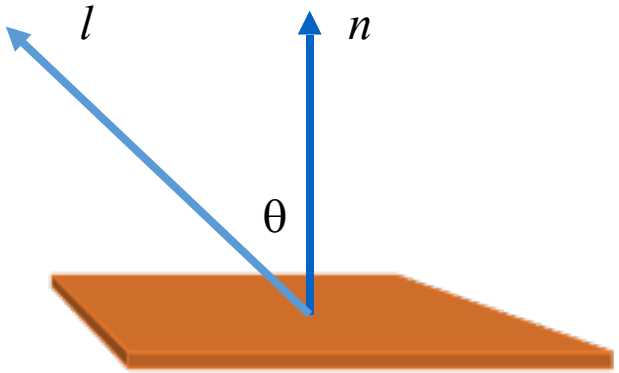
$$k_a = (k_{ar}, k_{ag}, k_{ab})$$

$$I = (I_r, I_g, I_b) = (I_{ar} k_{ar}, I_{ag} k_{ag}, I_{ab} k_{ab})$$

- La lumière bleue sur la surface blanche?
- La lumière bleue sur la surface rouge?

CALCULER LA RÉFLEXION DIFFUSE

Elle dépende de l'angle d'incidence: l'angle entre la normale est la direction de la lumière



$$I_{diffuse} = k_d I_{light} cos\theta = k_d I_{light}(n \cdot l)$$

Les variables suivantes sont de scalaires (pour les niveaux de gris) ou des triplés (la couleur)

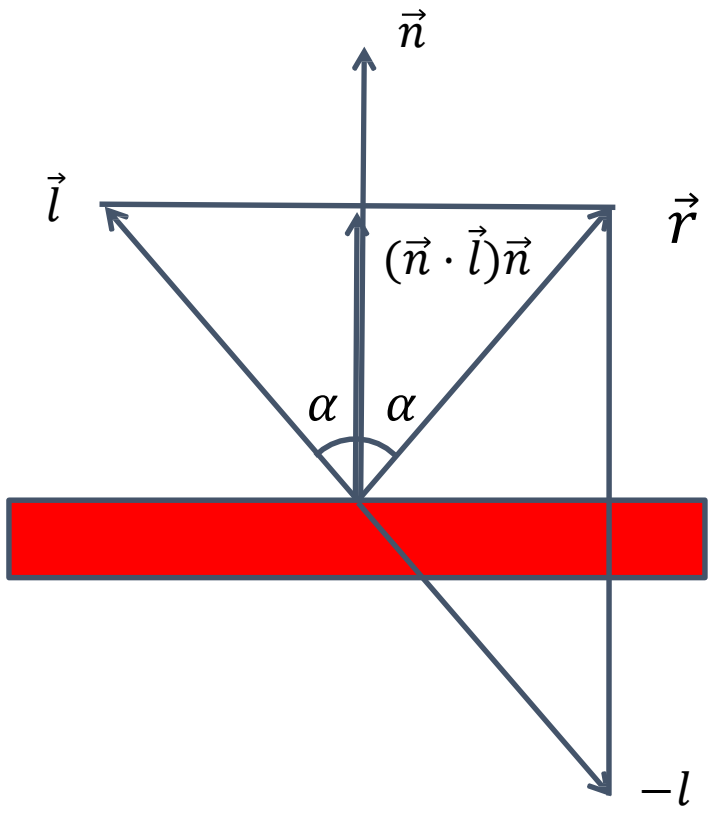
- k_d: un coefficient diffus, la couleur de surface
- I_{light}: l'intensité lumineuse entrante
- $I_{diffuse}$: l'intensité lumineuse sortante

NB: Il faut toujours normaliser les vecteurs en *shading*

- $\boldsymbol{n}, \boldsymbol{l}$ doivent avoir une longueur 1

LA PHYSIQUE DE LA RÉFLEXION SPÉCULAIRE

- La géométrie de la réflexion parfaite (spéculaire)
  - La loi de Snell-Descartes

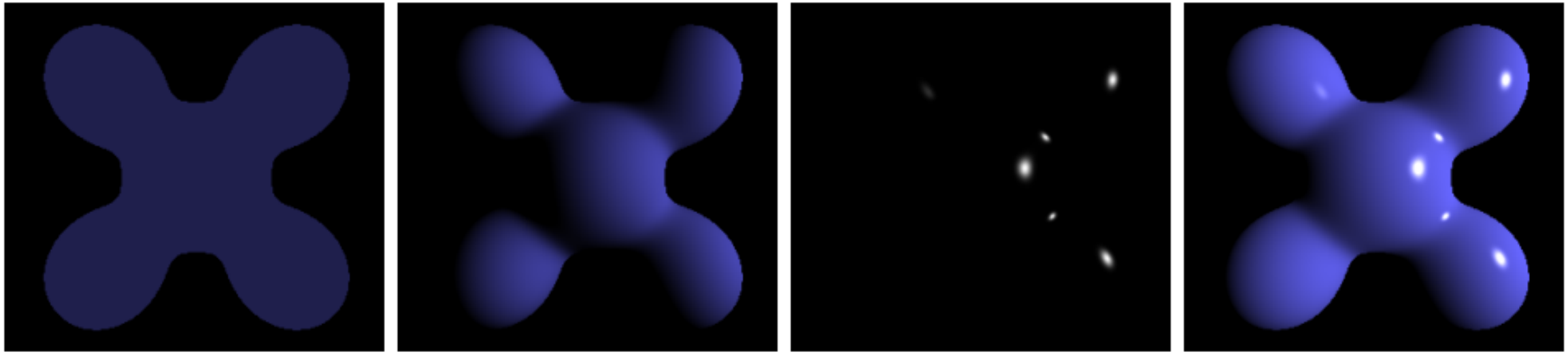


$$\vec{r} = -\vec{l} + 2(\vec{n} \cdot \vec{l})\vec{n}$$

L'ÉQUATION D'ÉCLAIRAGE (PHONG)

- Si on ajoute la composante de la lumière ambiante:

$$I_a k_a + \sum_p I_p (k_d (n \cdot l_p) + k_s (r_p \cdot v)^n)$$

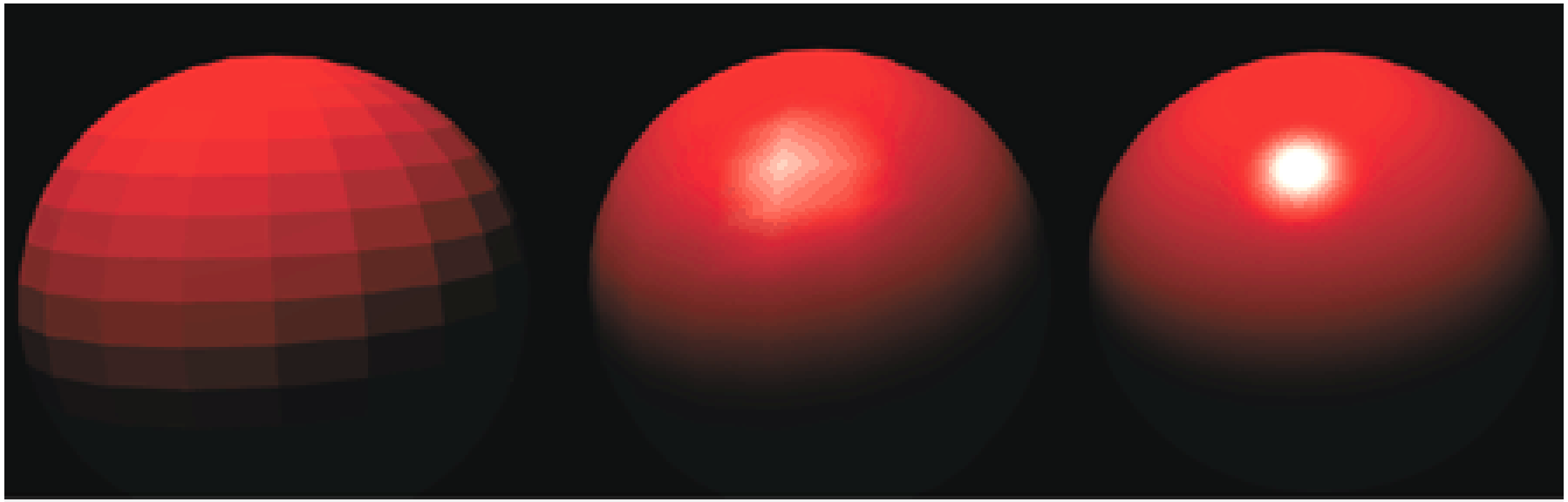


QUAND UTILISER UN MODÈLE D'ÉCLAIRAGE?

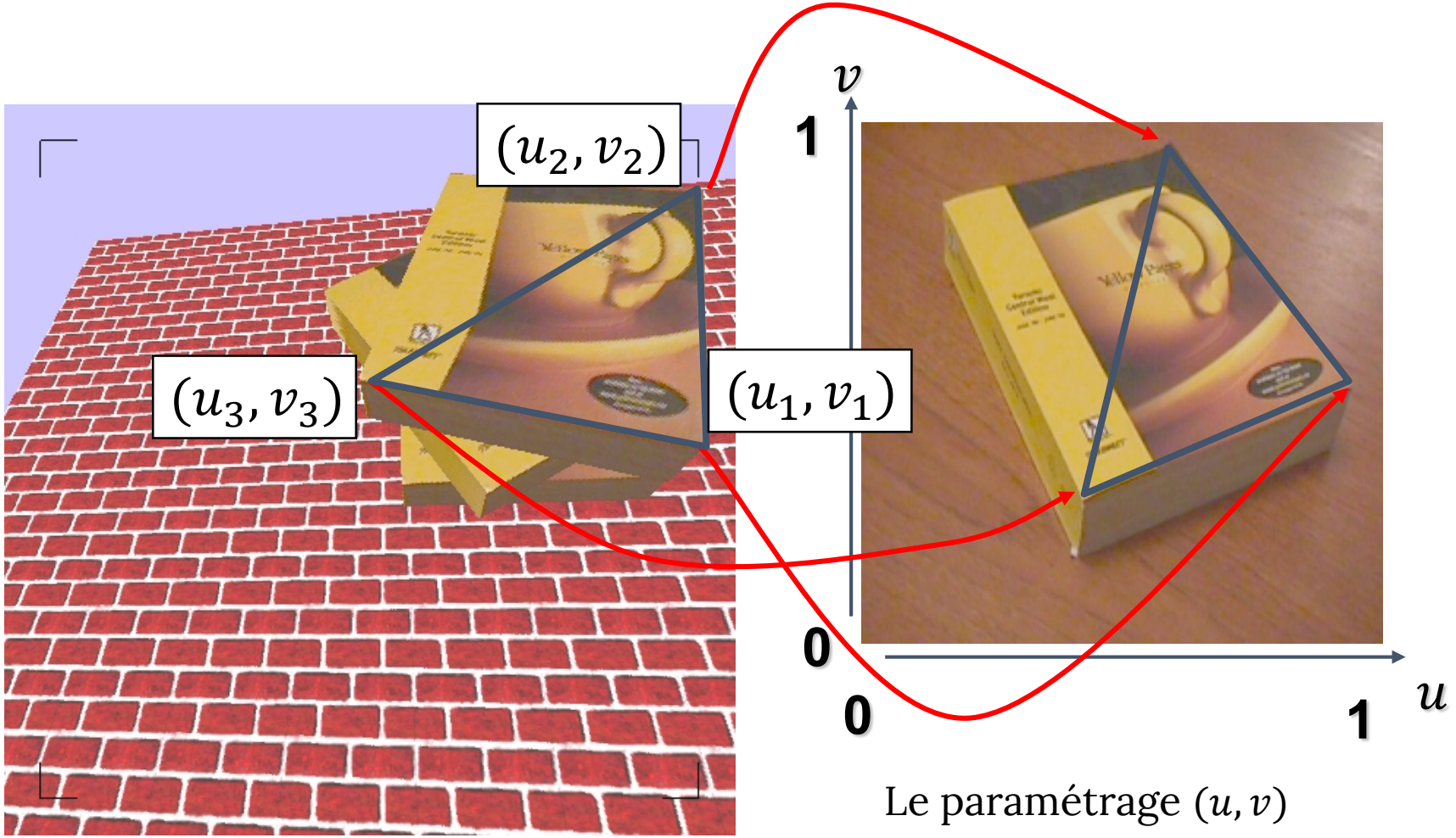
Par polygone
"l'ombrage plat"

Par sommet
"l'ombrage
de Gouraud"

Par pixel/fragment
"per pixel lighting"
"l'ombrage de
Phong"

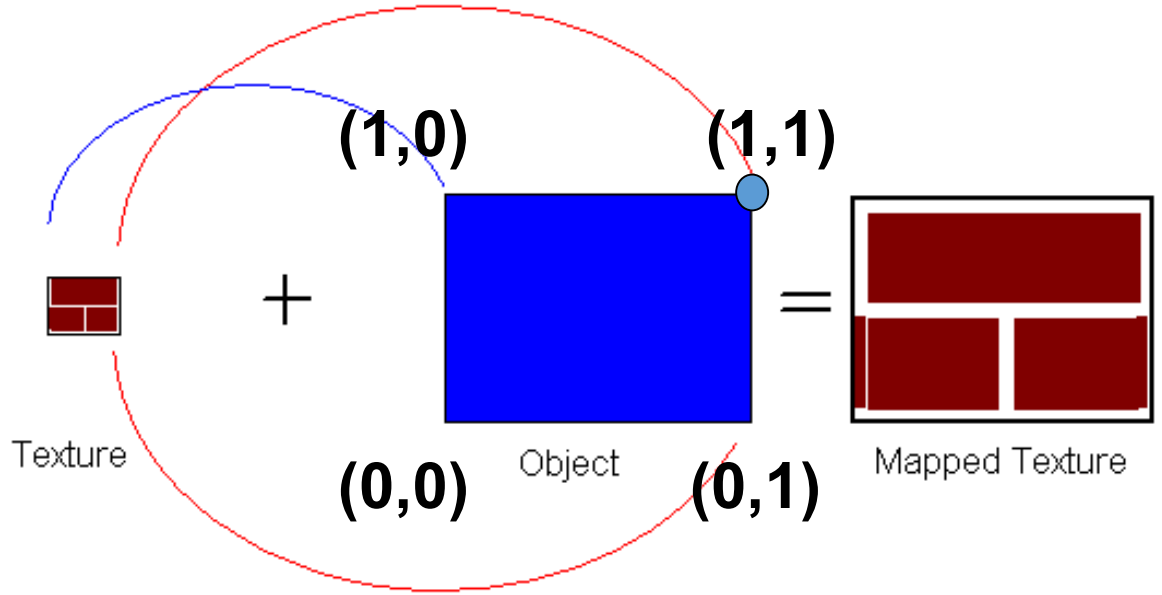


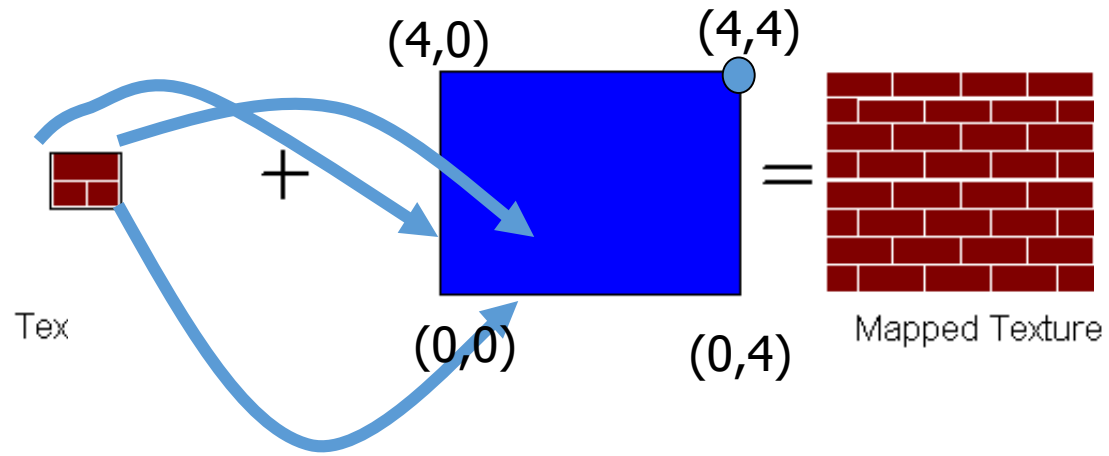
TEXTURE MAPPING



Le paramétrage (u, v)

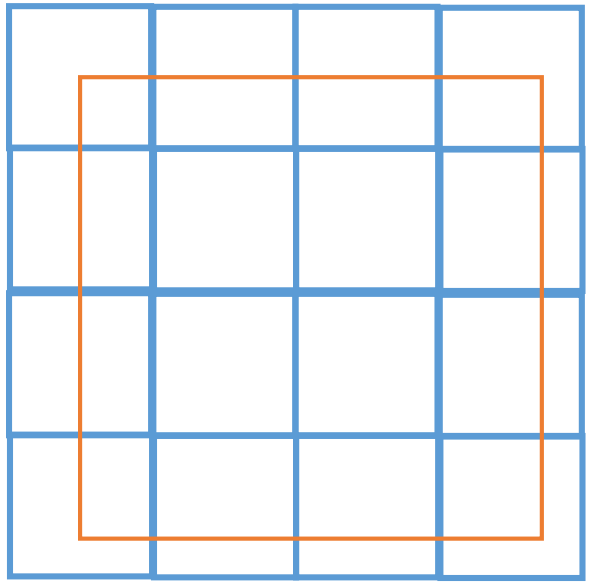
TILED TEXTURE MAP



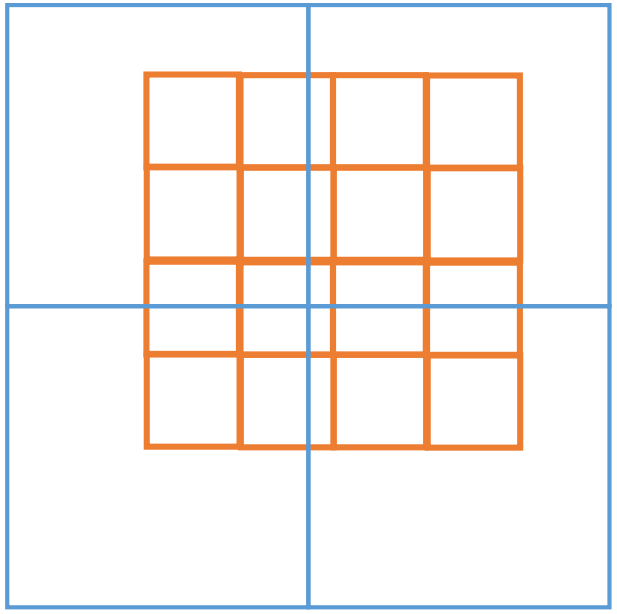


LA RECONSTRUCTION

- Comment faire avec:
  - Les pixels qui sont beaucoup plus grands que les texels?
    - La minification

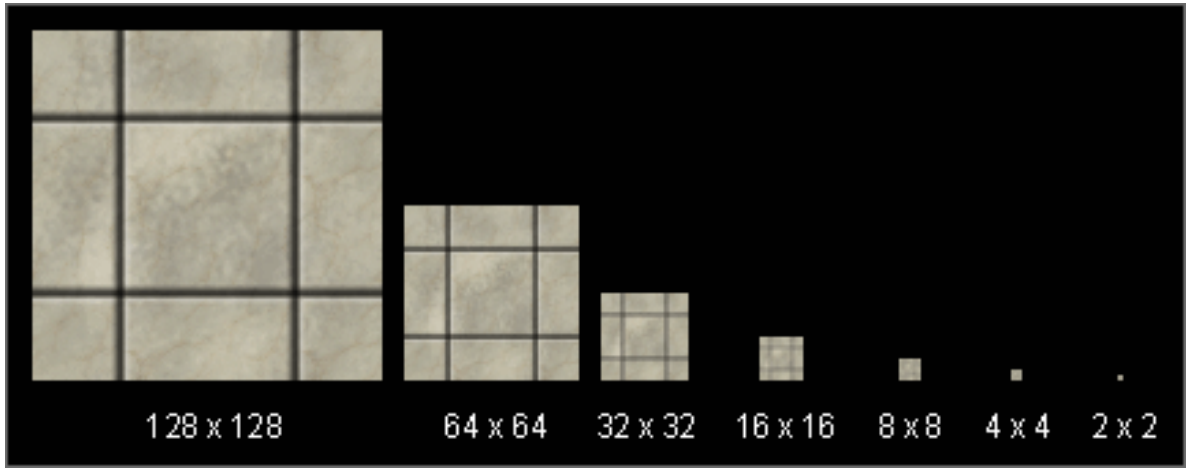


  - Les pixels qui sont beaucoup plus petits que les texels ?
    - La magnification

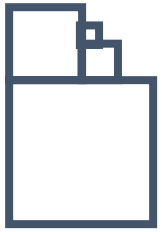


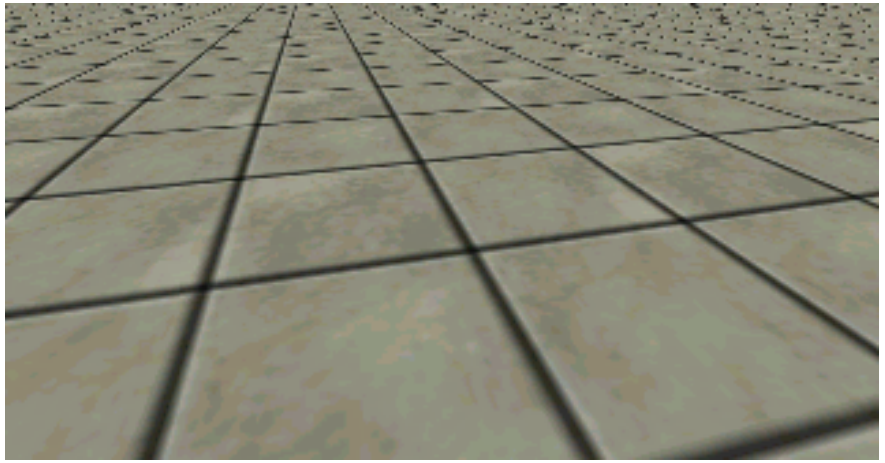
MIPMAPPING

Utiliser “une pyramide d’image” pour précalculer les versions moyennes d’image

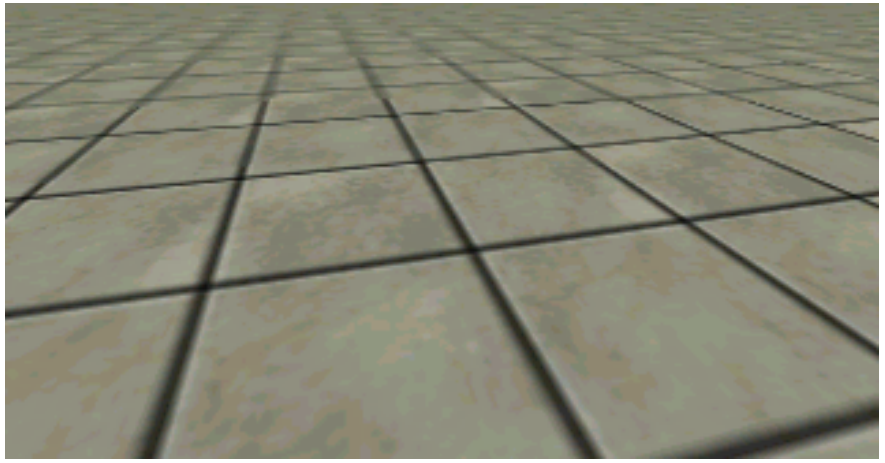


Stocker la pyramide entière comme un seul bloc de mémoire





Sans MIP-mapping

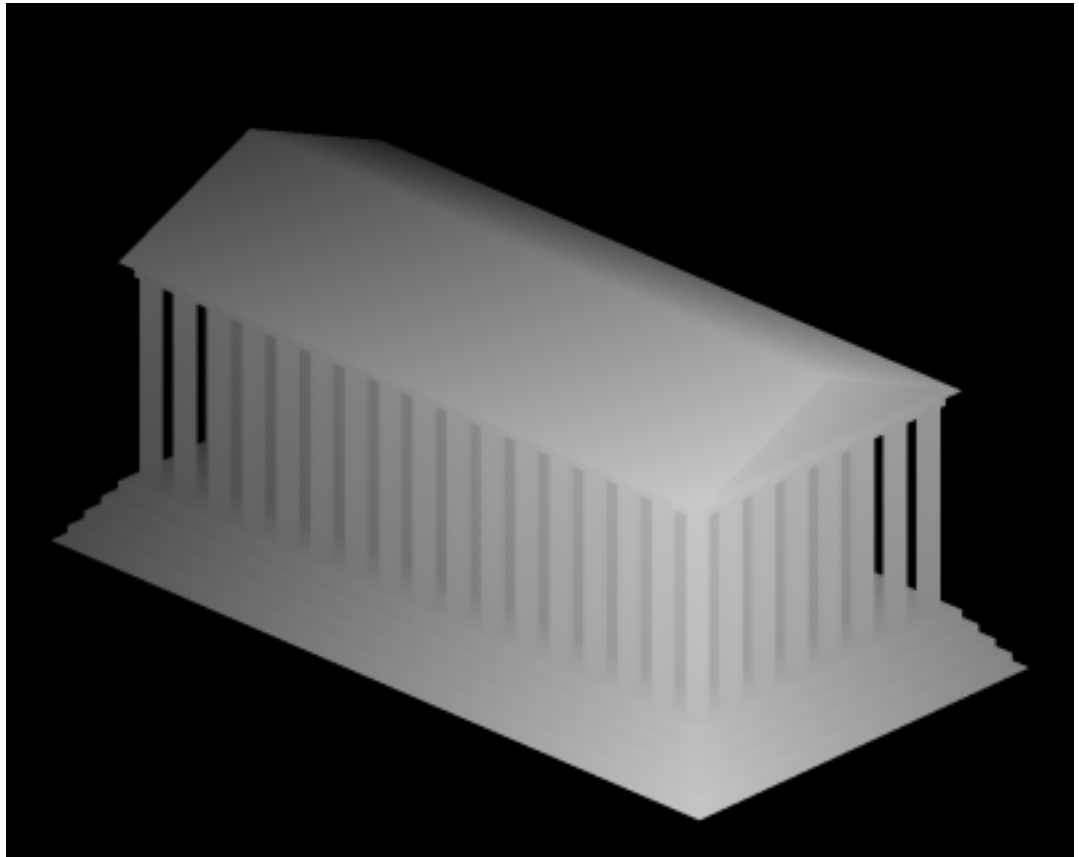


Avec MIP-mapping

LES OMBRES

On a besoin d'au moins 2 passes de *shaders* :

1. Rendre tout comme vu depuis **la lumière**
   Enregistrer la profondeur (tampon de profondeur, '*depth map*')



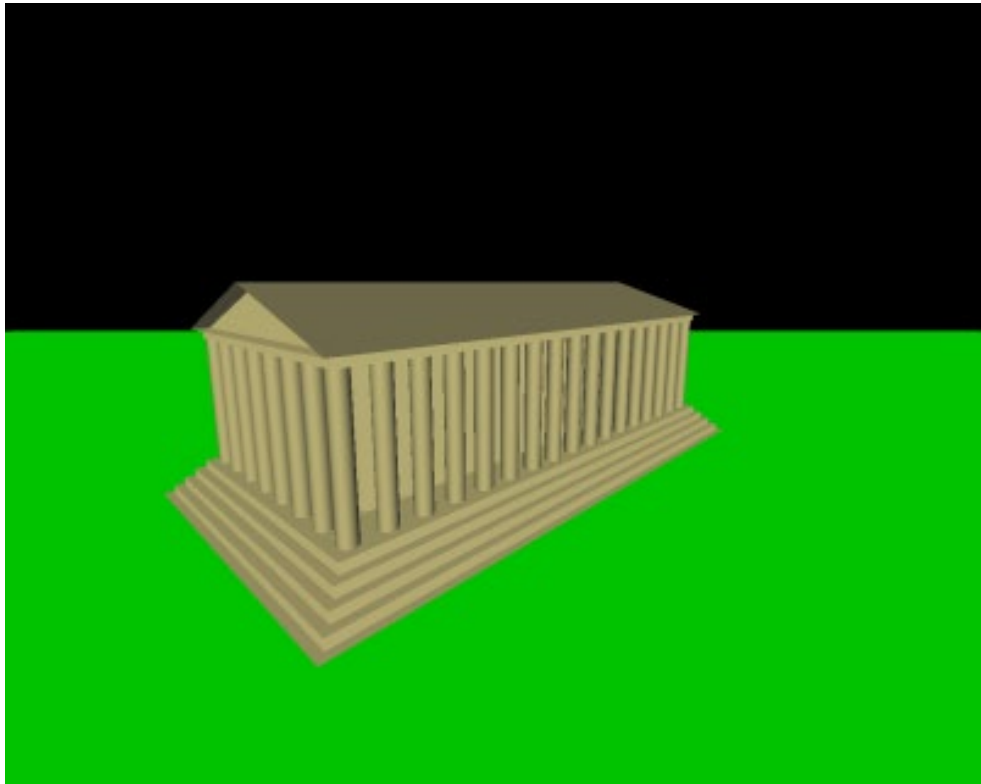
LES OMBRES (IDÉE)

On a besoin d'au moins 2 passes de *shaders* :

1. Rendre tout comme vu depuis **la lumière**
   Enregistrer la profondeur (tampon de profondeur/ombre, '*shadow map*')
2. Maintenant, rendre tout depuis la CAMÉRA
   Quand on calcule la couleur d'un fragment:

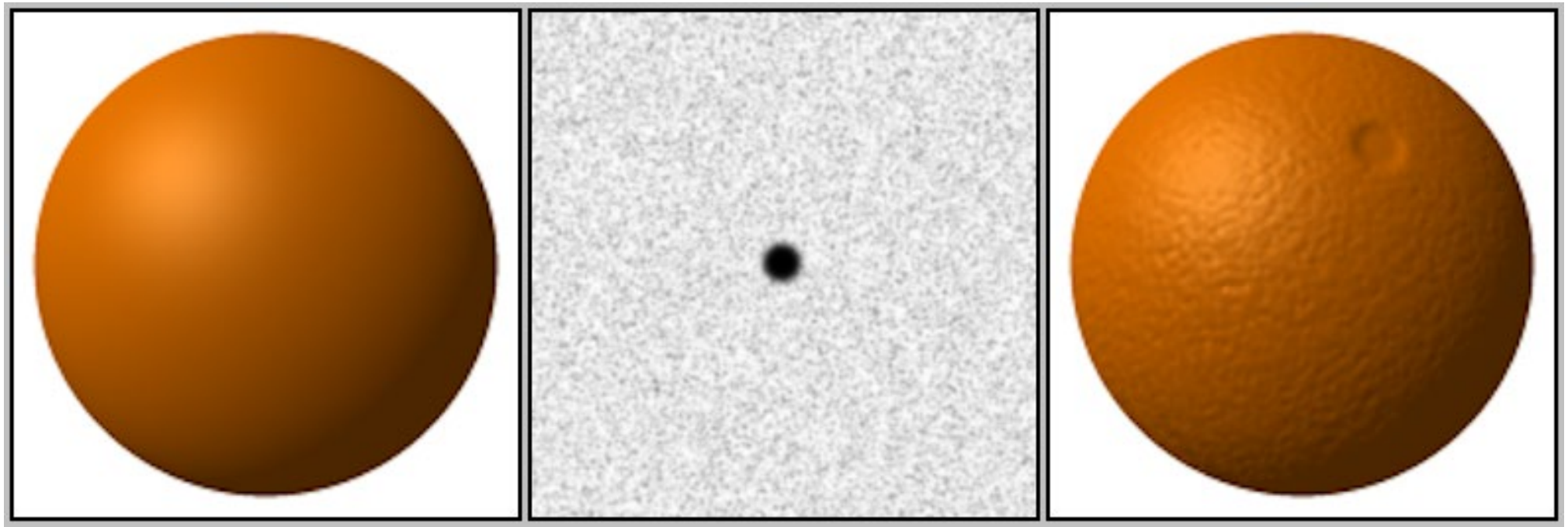
   • Convertir les coordonnées dans l'espace de la lumière (x_l, y_l, z_l)
      • Prendre la profondeur $D(x_l, y_l)$

   • Est-ce $z_l > D(x_l, y_l)$?
      • Oui: je suis dans l'ombre
      • Non: je suis éclairé

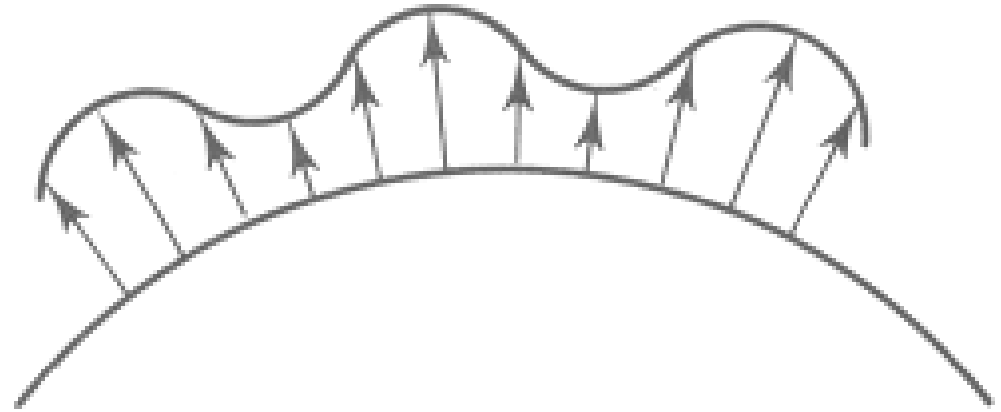


BUMP MAPPING ET NORMAL MAPPING

- La surface de l'objet est souvent rugueuse
  - On peut créer une géométrie plus complexe…
- Ou on peut la simuler en perturbant la normale seulement
  - Soit aléatoire
  - Soit spécifiée par une texture

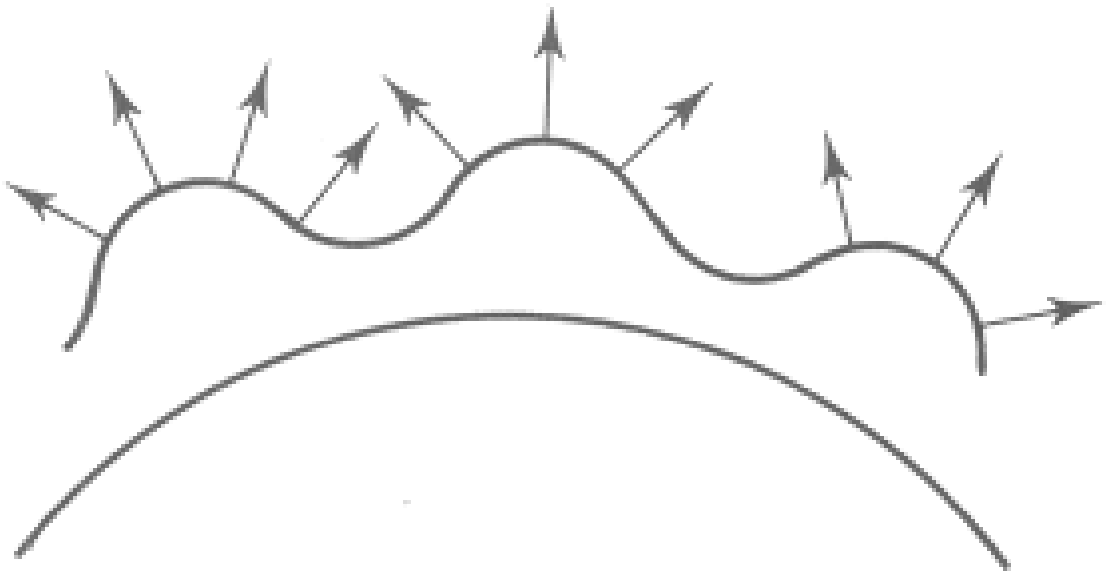


BUMP MAPPING



$O'(u)$

Lengthening or shortening $O(u)$ using $B(u)$

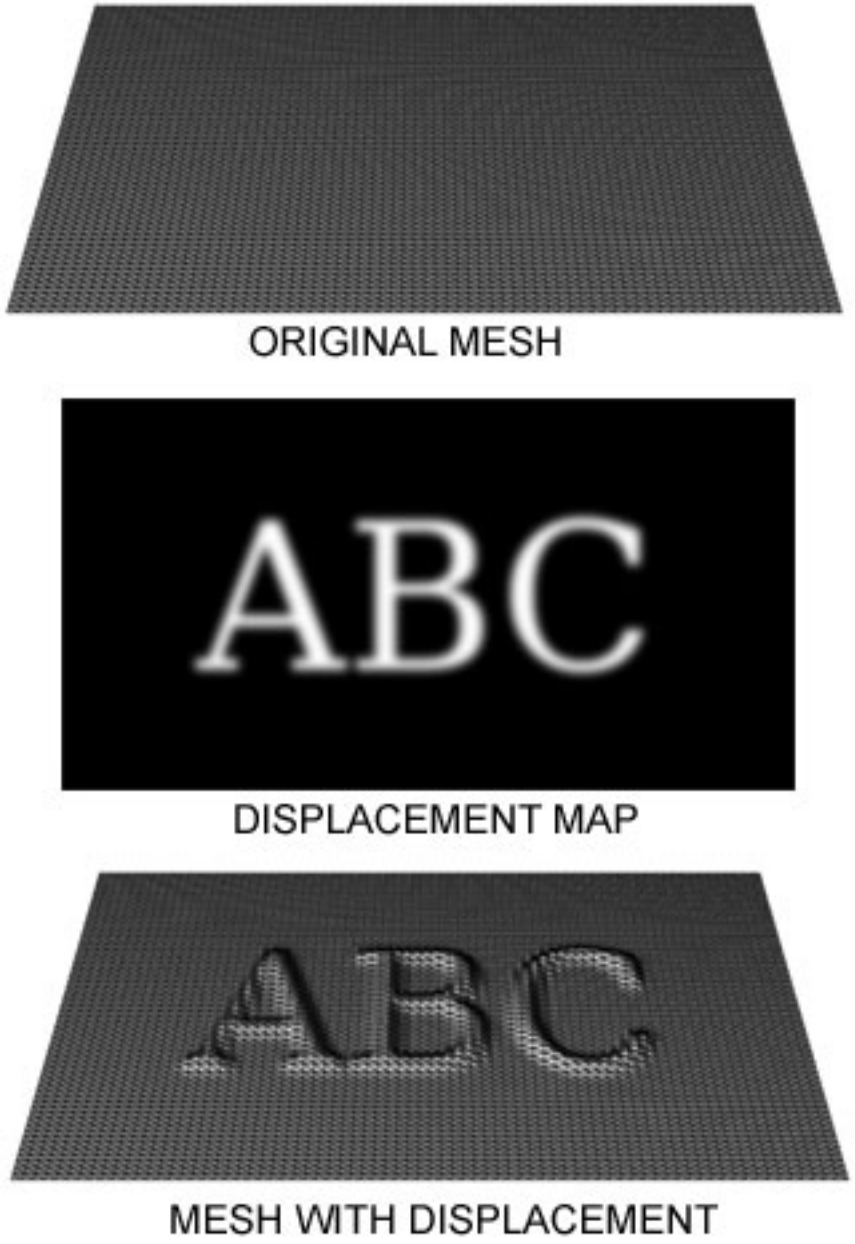


$N'(u)$

The vectors to the 'new' surface

DISPLACEMENT MAPPING

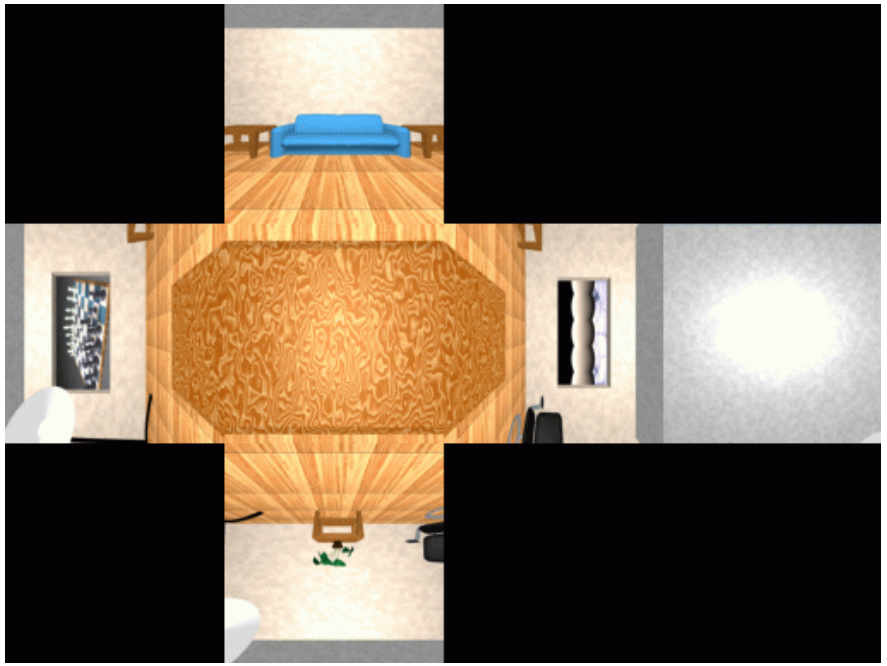
- Les silhouettes sont fausses après *le bump mapping*
  - Et les ombres aussi!
- Changer la géométrie en temps réel
  - On a besoin de subdiviser la surface



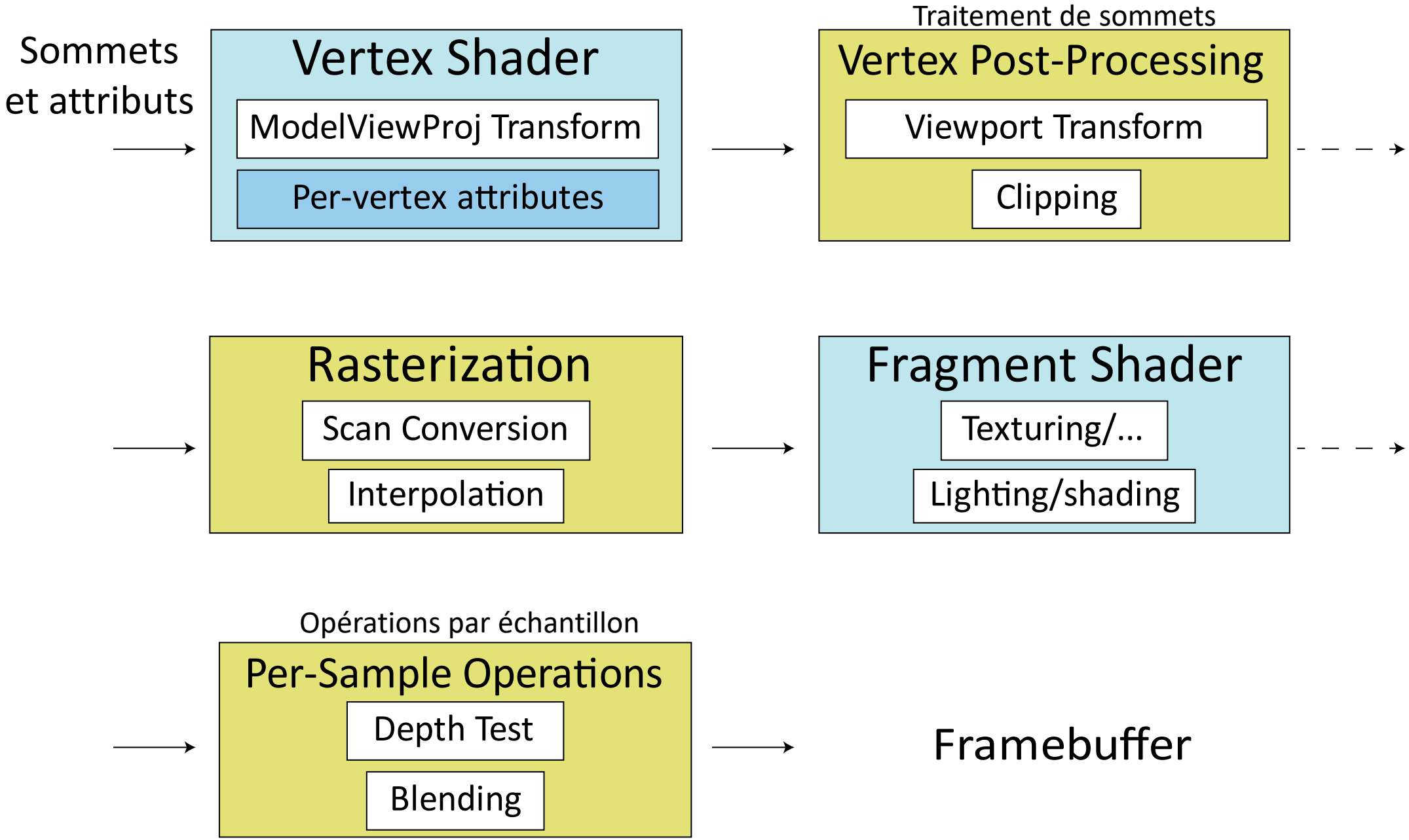
https://en.wikipedia.org/wiki/Displacement_mapping#/media/File:Displacement.jpg

CUBE MAPPING

- 6 textures planaires, les faces du cube
  - Pointe la caméra depuis l'origine dans 6 directions



PIPELINE: PLUS DE DÉTAILS



ALGORITHME DU PEINTRE: LES PROBLÈMES

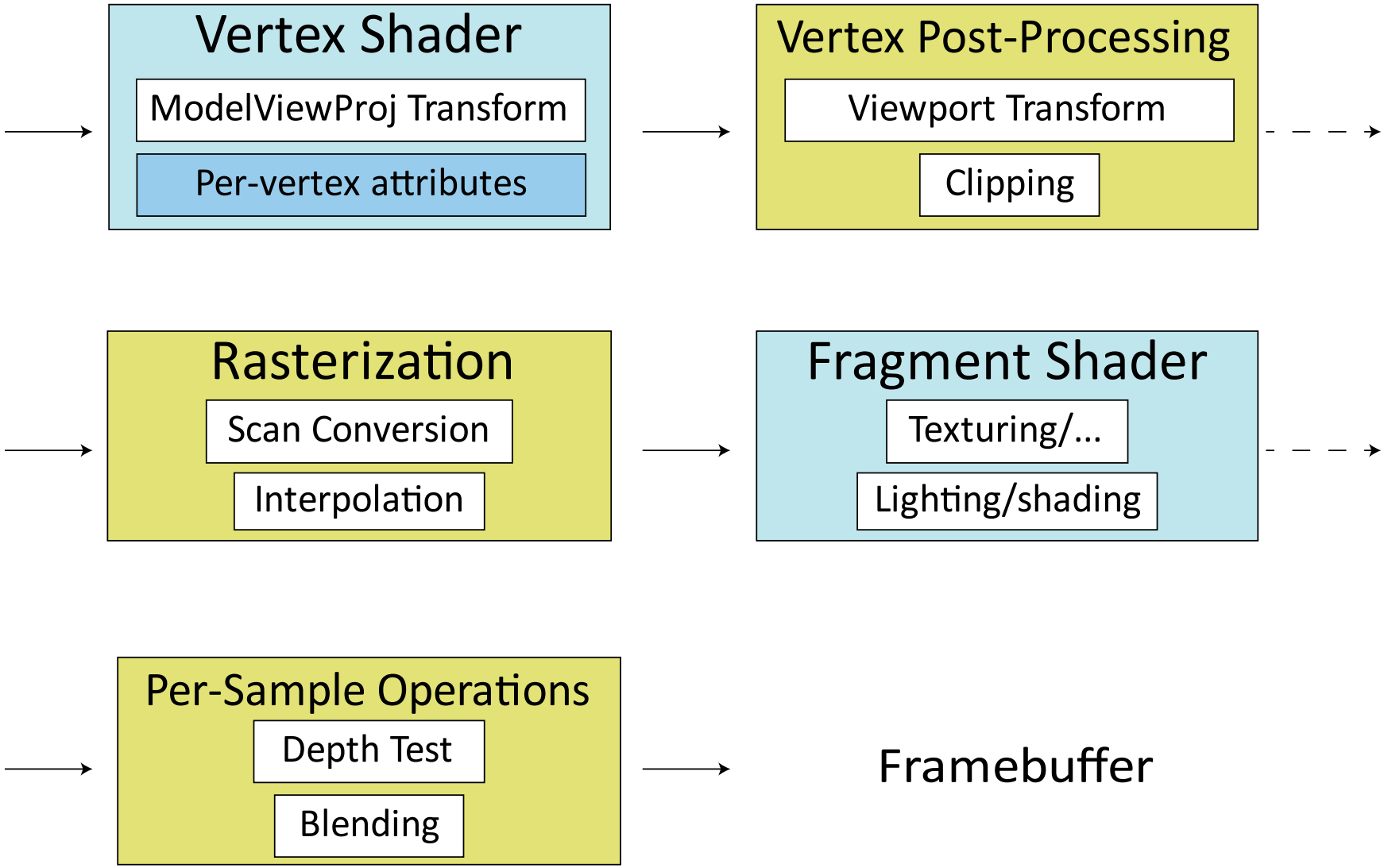
- Les polygones qui s'intersectent posent un problème
- Même les polygones sans intersections peuvent poser problème aussi:

TAMPON DE PROFONDEUR

- Stocker (r, g, b, z) pour chaque pixel
  - Normalement, 8+8+8+24 bits, peut-être plus

```
for all i,j {
 Depth[i,j] = MAX_DEPTH
 Image[i,j] = BACKGROUND_COLOUR
}
for all polygons P {
  for all pixels in P {
    if (Z_pixel < Depth[i,j]) {
      Image[i,j] = C_pixel
      Depth[i,j] = Z_pixel
    }
  }
}
```

POURQOUI APRÈS FRAGMENT SHADER??



LA PRÉCISION DU TEST DE PROFONDEUR

- Un rappel: une transformation projective transforme z du système de coordonnées de vue aux coordonnées normalisées (NDCS)
- L'exemple simple:

$$T\left(\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}\right) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

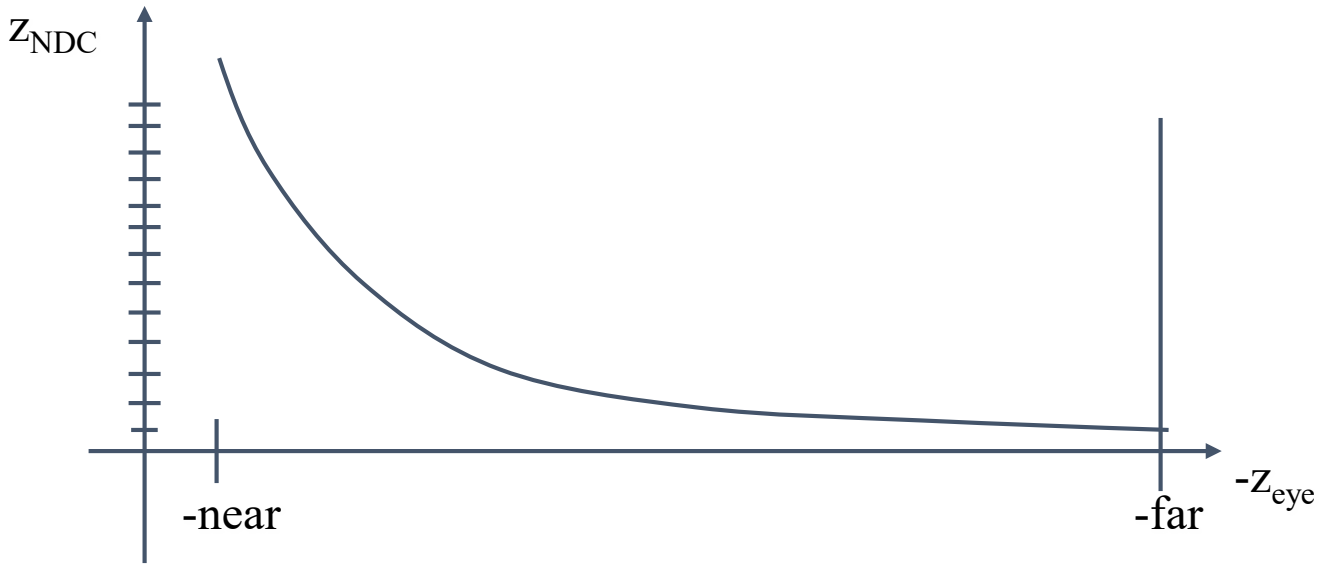
- Donc,

$$z_{NDC} = \frac{az_{eye} + b}{-z_{eye}} = -a - \frac{b}{z_{eye}}$$

LA PRÉCISION DU TEST DE PROFONDEUR

Donc, le tampon de profondeur stocke $1/z$ à la place de z!

- Les problèmes avec les tampons en entiers
  - Haute précision pour les objets proches
  - Faible précision pour les objets éloignés



LA PRÉCISION DU TEST DE PROFONDEUR

- La faible précision peut entraîner du Z-fighting pour les objets éloignés
  - Deux profondeurs différentes peuvent devenir la même valeur (quantization)
  - Quel objet gagne dépend de l'ordre de l'affichage

- Pire pour les grands ratios $\frac{f}{n}$
  - Règle générale: $\frac{f}{n} < 1000$ pour une profondeur sur 24 bits

- Avec 16 bits on ne voit pas les différences de 1cm à la distance de 1km