

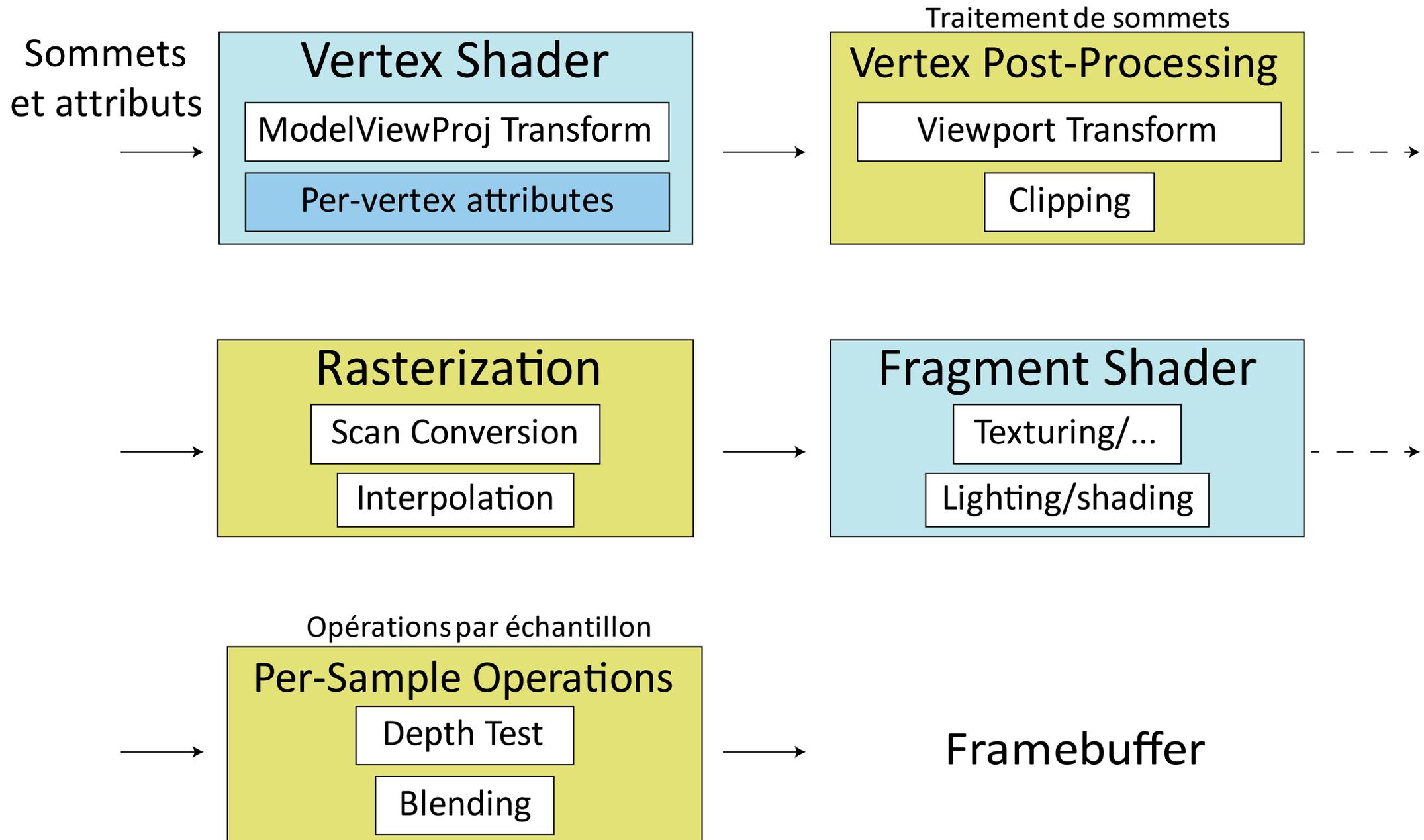
<http://tiny.cc/ift3355>

# IFT 3355: INFOGRAPHIE CLIPPING ET RASTERIZATION

Livre de référence: G:12.4; S: 3

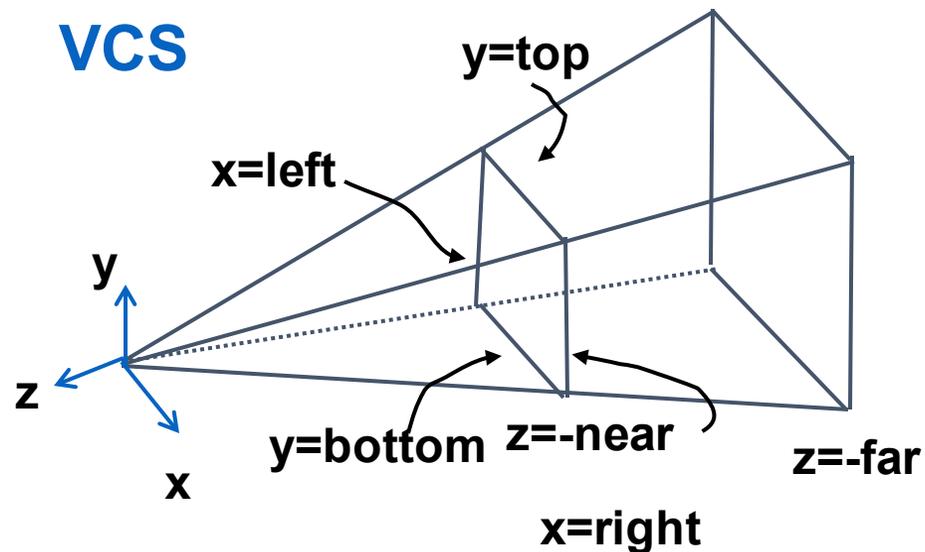
Mikhail Bessmeltsev

# PIPELINE: PLUS DE DÉTAILS



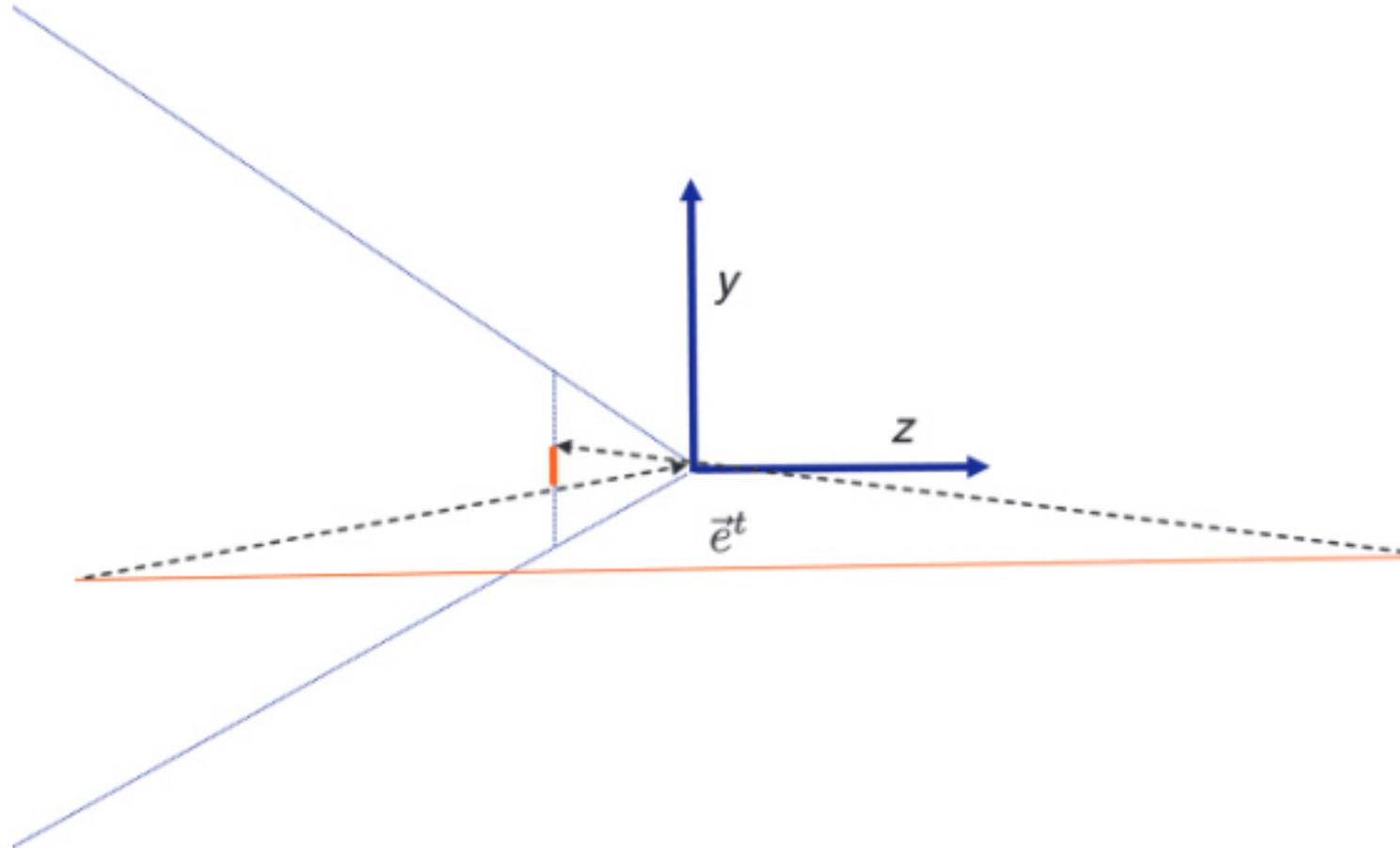
# CLIPPING

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



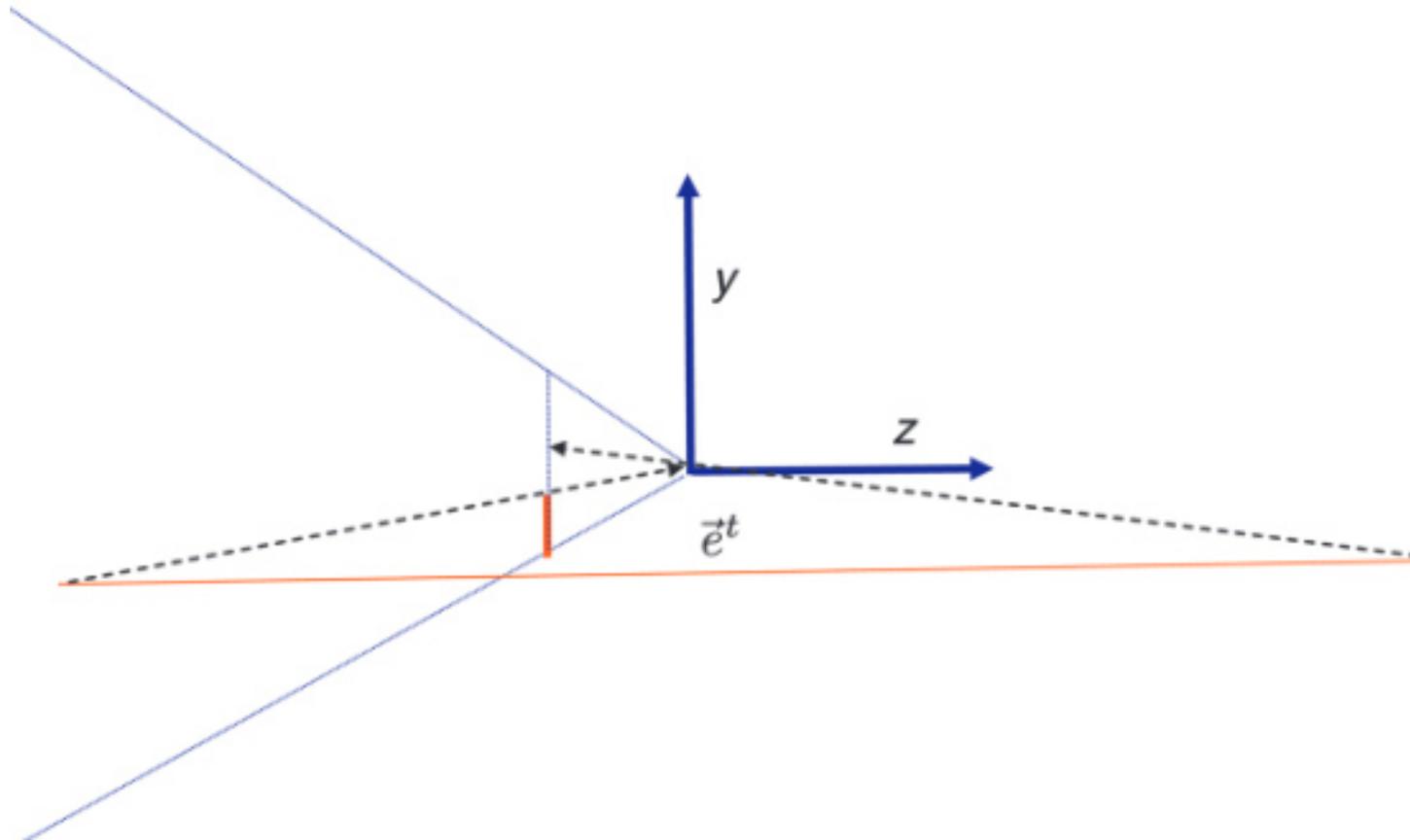
# CLIPPING

- Plus important, avant/arrière:



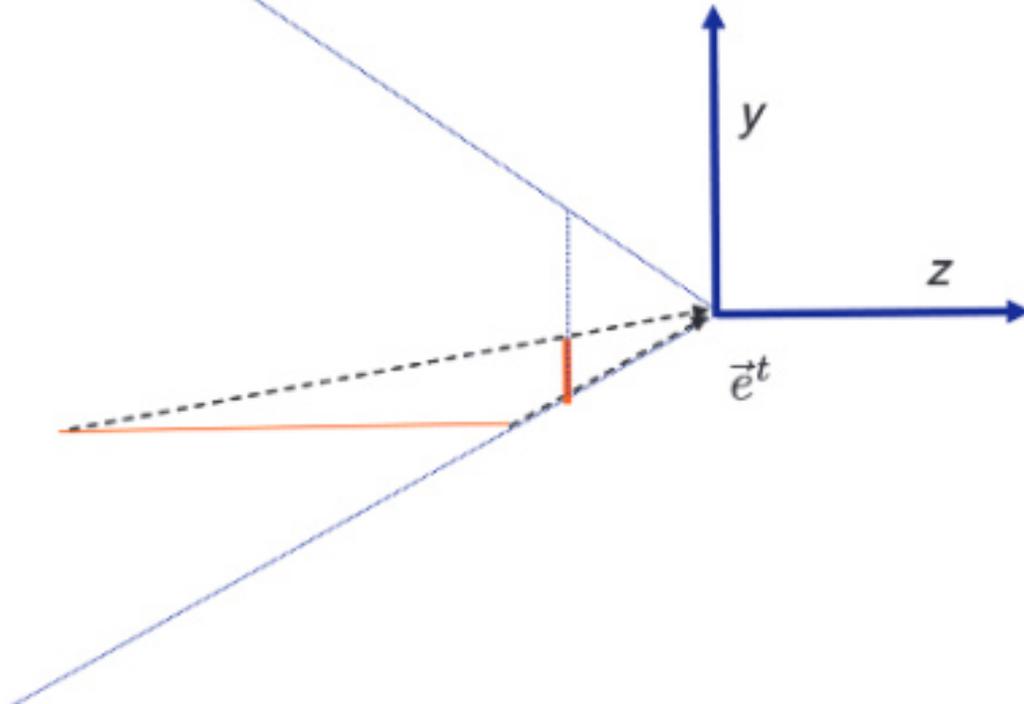
# CLIPPING

- Plus important, avant/arrière:



# CLIPPING

- Plus important, avant/arrière:



# CLIPPING

Où le faire dans le pipeline?

# CLIPPING

Où le faire dans le pipeline?

Le plus tôt le mieux

# CLIPPING

Où le faire dans le pipeline?

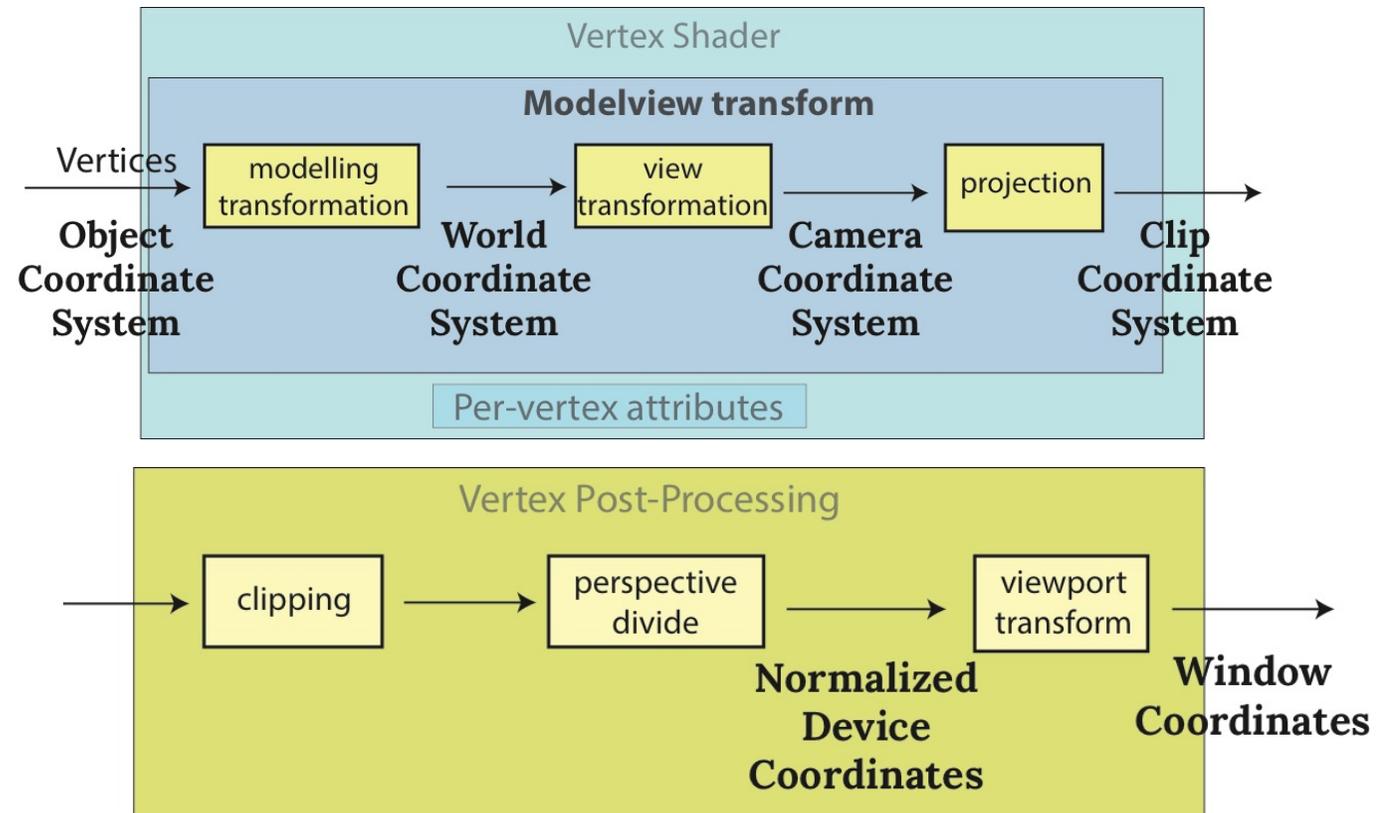
Le plus tôt le mieux

Avant la projection?

Les plans du volume de vue sont plus compliqués

# CLIPPING

- Immédiatement après la projection, dans le cadre de coordonnées du *clip*!
  - Avant la division perspective



# CLIPPING

- Faire *clipping* dans l'espace de *clip*
  - Après la projection, mais avant la division par  $w$

$$-w_c < x_c < w_c$$

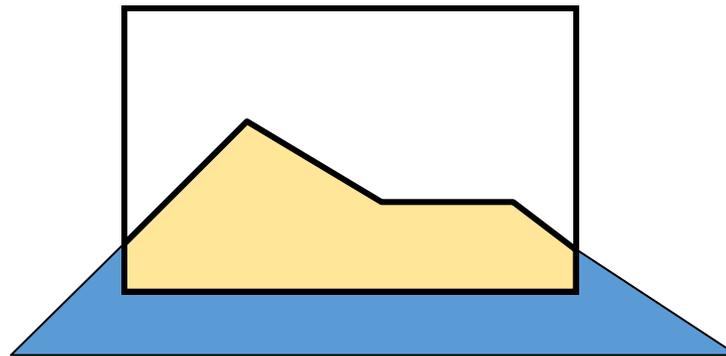
$$-w_c < y_c < w_c$$

$$-w_c < z_c < w_c$$

Pas de division  $\Rightarrow$  Efficacité

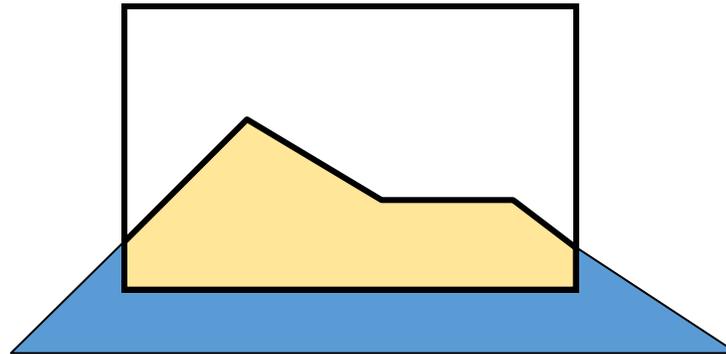
# CLIPPING: SOUS LE CAPOT

- On crée des nouveaux sommets
- C'est fait automatiquement, nous n'apprendrons pas l'algorithme



# CLIPPING: SOUS LE CAPOT

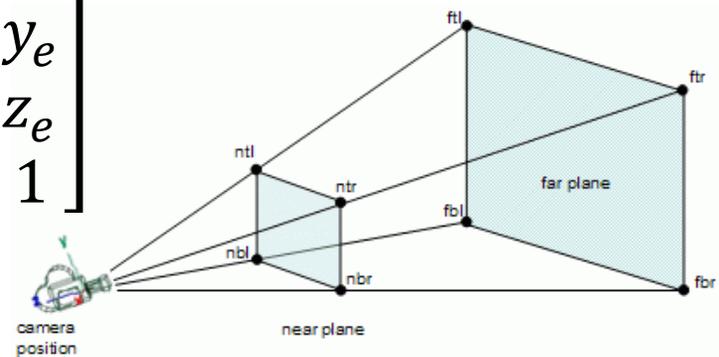
- On crée des nouveaux sommets
- C'est fait automatiquement, nous n'apprendrons pas l'algorithme
  - Rejeter les sommets
  - Découper les triangles



# LES COORDONNÉES DE *CLIPPING*

- Les coordonnées de vue  $\rightarrow$  les coordonnées de *clipping*  $\rightarrow$  les coordonnées de l'appareil normalisées (NDC)
- Après, diviser les coordonnées homogènes  $(x_c, y_c, z_c, w_c)$  par  $w_c$ . La dernière coordonnée,  $w_c$ , est passée sans changement

$$\begin{bmatrix} x_n w_n \\ y_n w_n \\ z_n w_n \\ w_n \end{bmatrix} = \begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} = \begin{bmatrix} s_x & 0 & -c_x & 0 \\ 0 & s_y & -c_y & 0 \\ 0 & 0 & \frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix}$$

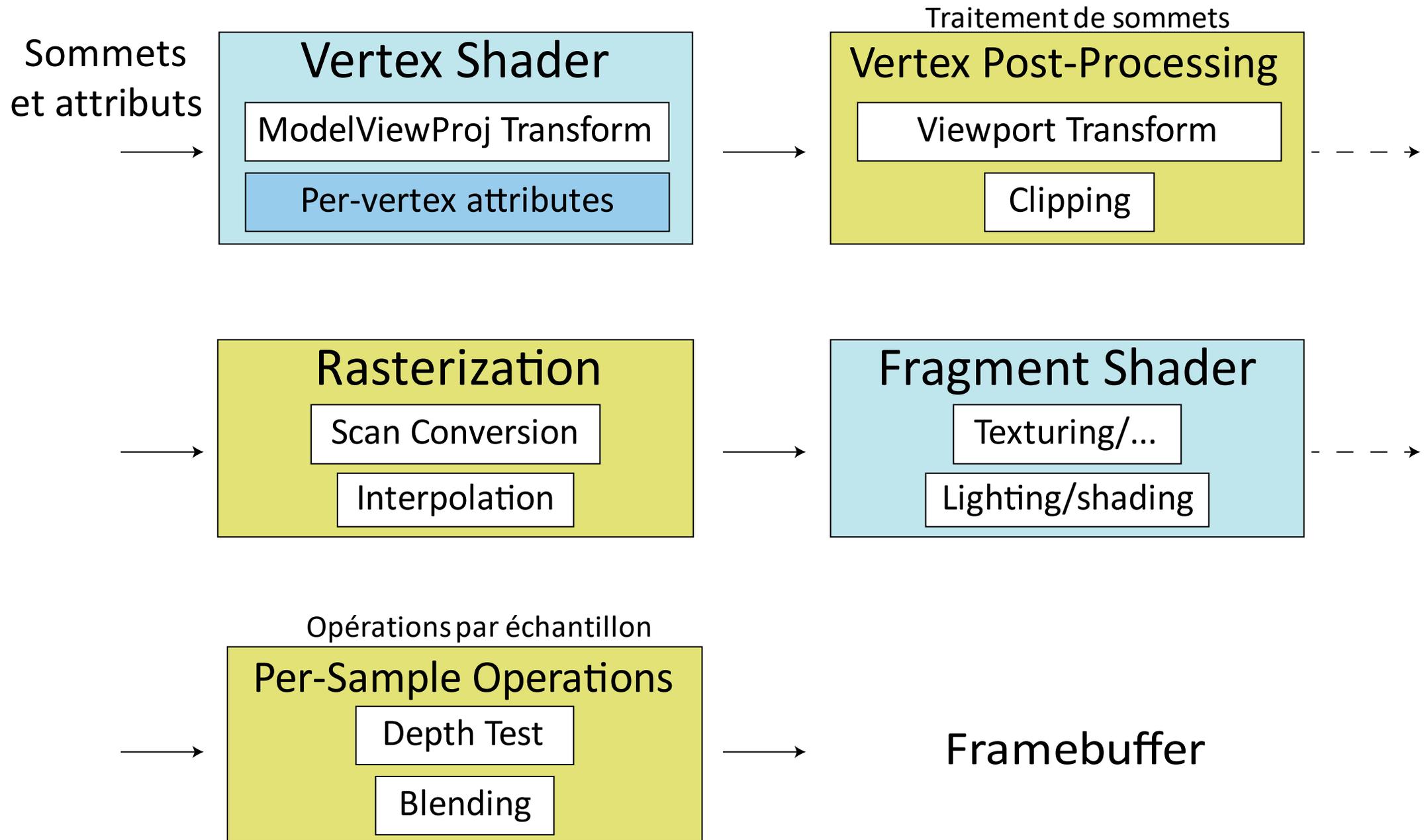


# LA MATRICE DE LA FENÊTRE (*VIEWPORT*)

- On a besoin de la transformation qui convertit le coin inférieur gauche en  $(-0.5, -0.5)^T$  et le coin supérieur droit en  $(W - 0.5, H - 0.5)^T$
- C'est n'est qu'un changement d'échelle + une translation

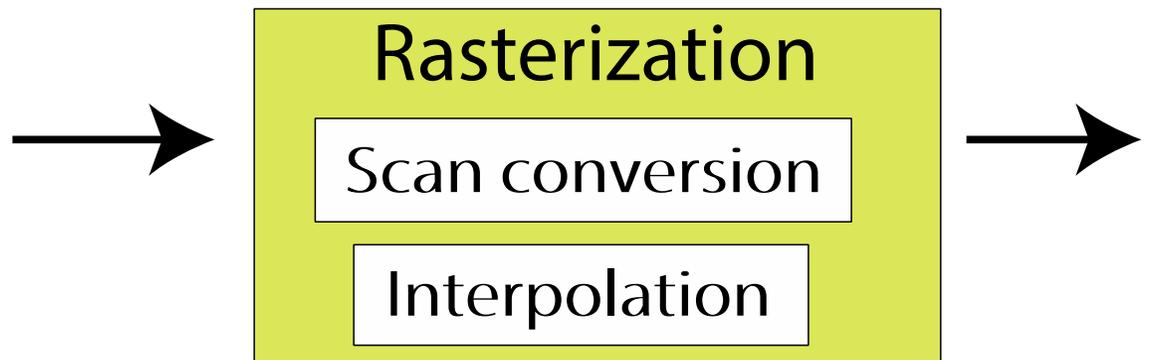
$$\begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{W}{2} & 0 & 0 & \frac{W-1}{2} \\ 0 & \frac{H}{2} & 0 & \frac{H-1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_n \\ y_n \\ z_n \\ 1 \end{bmatrix}$$

# PIPELINE: PLUS DE DÉTAILS

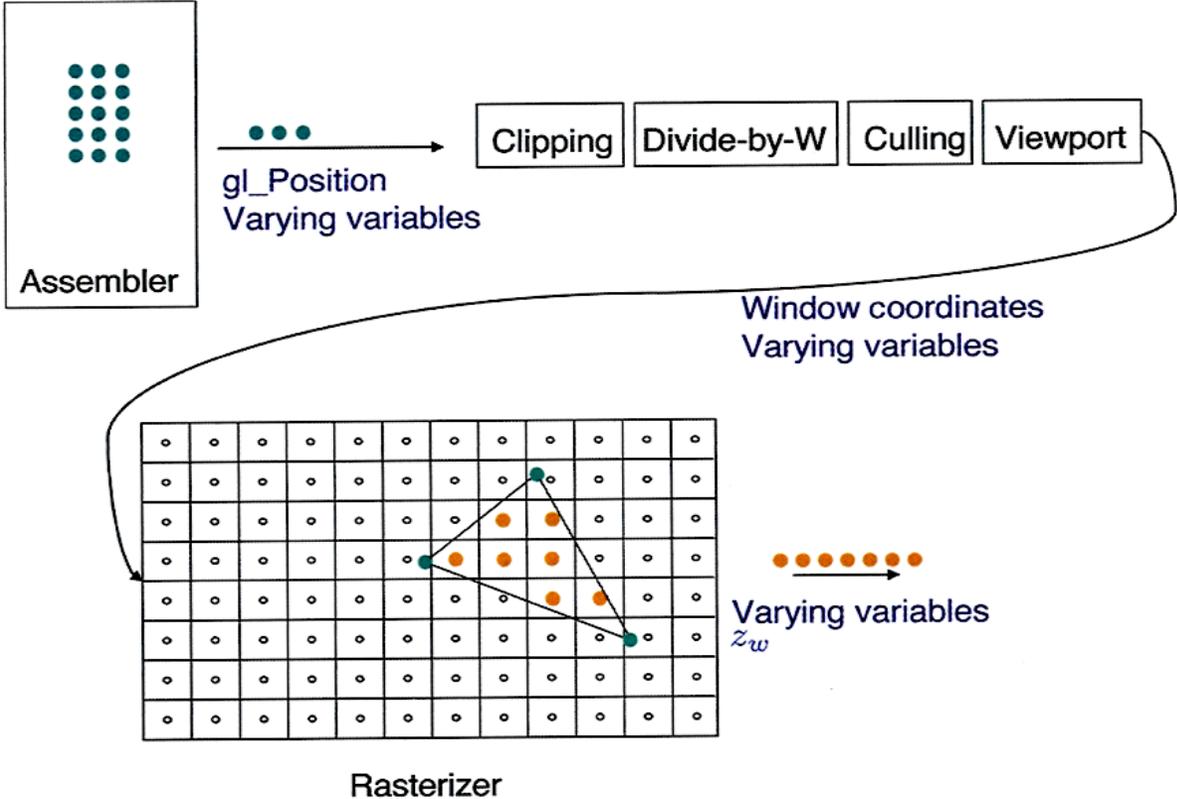


# RASTERIZATION

- 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)



# UN CHEMIN DU SOMMET AU PIXEL



Avant:

# LES FORMES – LES COURBES/SURFACES

- Les représentations mathématiques:
  - Les fonctions explicites
  - Les fonctions paramétriques
  - Les fonctions implicites

Avant:

# LES FORMES: LES FONCTIONS EXPLICITES

- Les courbes:

- $y$  est une fonction de  $x$ :
- Pas toutes le courbes

$$y := \sin(x)$$

- Les surfaces:

- $z$  est une fonction de  $x$  et  $y$ :
- Pas toutes les surfaces

$$z := \sin(x) + \cos(y)$$

Avant:

# LES FORMES: LES FONCTIONS PARAMÉTRIQUES

- Les courbes:
  - 2D:  $x$  et  $y$  sont en fonction de la valeur du paramètre  $t$
  - 3D:  $x$ ,  $y$  et  $z$  sont en fonction de la valeur du paramètre  $t$

$$C(t) := \begin{pmatrix} \cos(t) \\ \sin(t) \\ t \end{pmatrix}$$

Avant:

# LES FORMES: LES FONCTIONS PARAMÉTRIQUES

- Les surfaces:
  - Une surface  $S$  est définie en fonction des valeurs des paramètres  $s, t$
  - Les noms des paramètres peuvent être différents:

$$S(\phi, \theta) := \begin{pmatrix} \cos(\phi) \cos(\theta) \\ \sin(\phi) \cos(\theta) \\ \sin(\theta) \end{pmatrix}$$

Avant:

# LES FORMES: LES FONCTIONS IMPLICITES

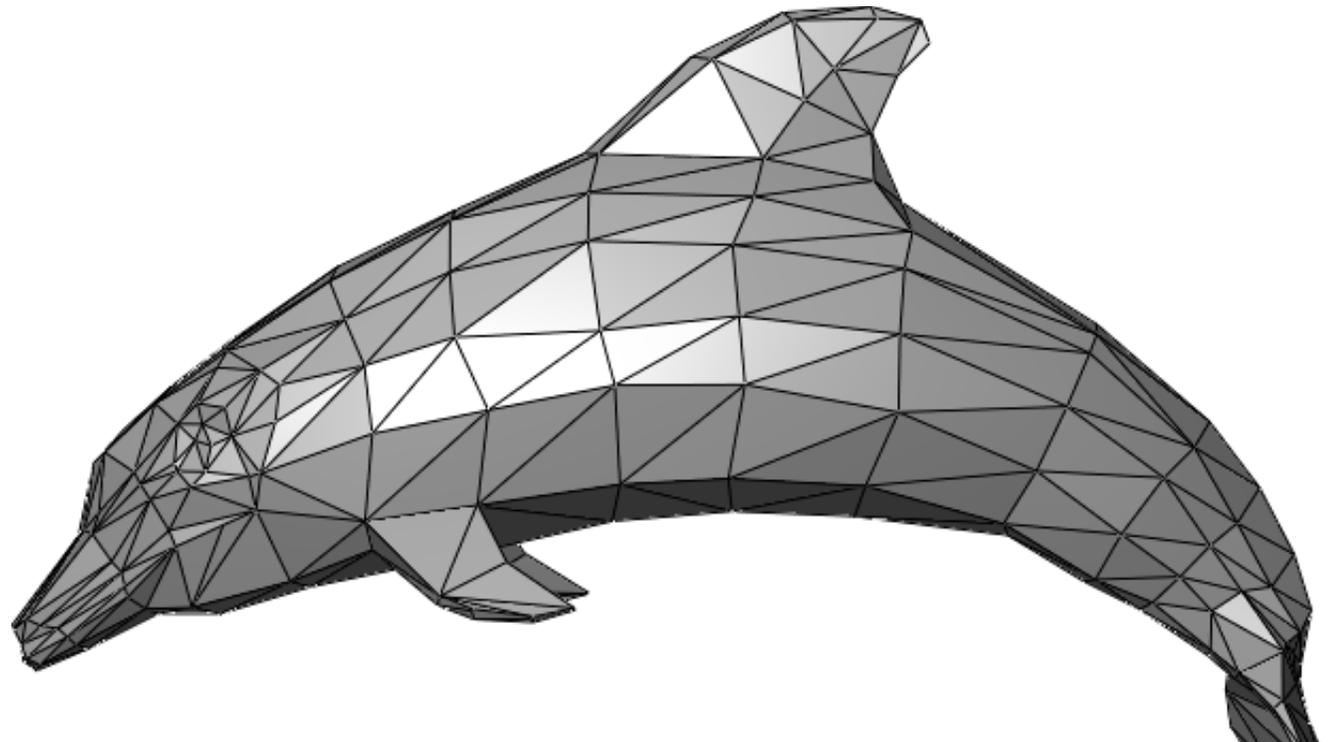
- La surface (3D) ou la courbe (2D) est définie par les racines de la fonction
  - E.g.:

$$S(x, y, z): x^2 + y^2 + z^2 - 1 = 0$$

Avant:

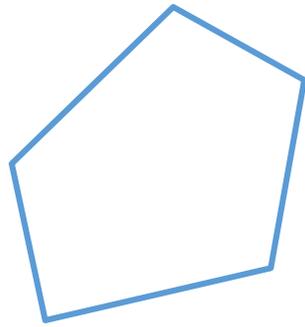
# LES FORMES: LES MAILLAGES DE TRIANGLES

- La liste de sommets
- Les triangles sont définis comme  $\{\text{vertex\_index1}, \text{vertex\_index2}, \text{vertex\_index3}\}$

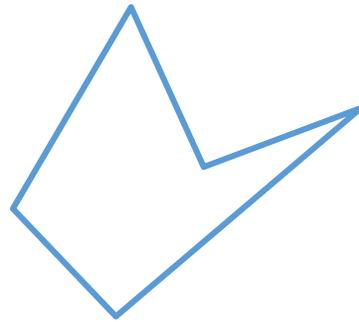


# LES POLYGONES

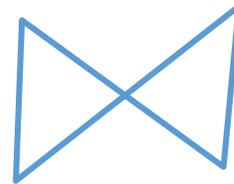
- Les types de base



Convexe  
simple



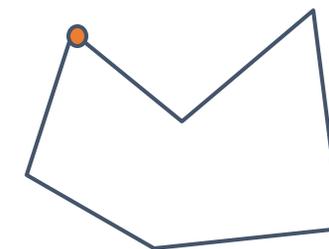
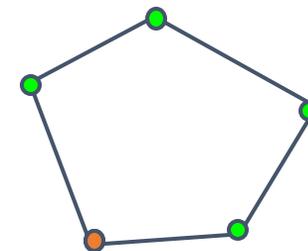
Non convexe  
simple



Non-simple  
(contient une auto-intersection)

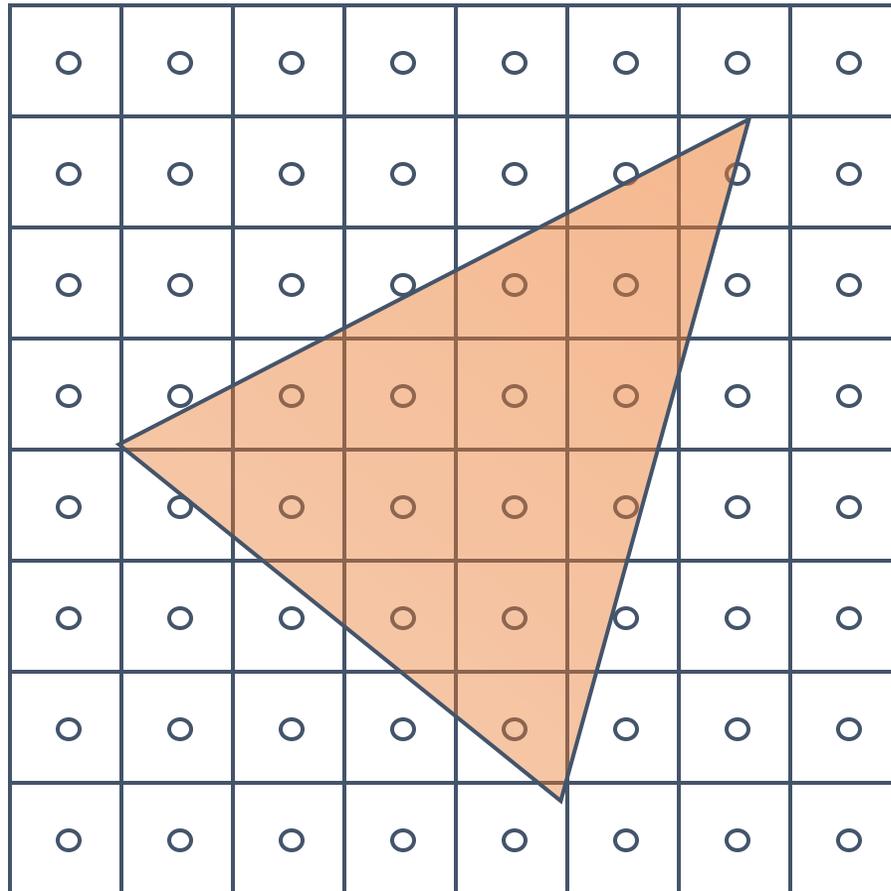
# DES POLYGONES AUX TRIANGLES

- Pourquoi? Les triangles sont toujours convexes
- Les polygones convexes simples sont faciles à convertir en triangles
- Pour les autres
  - C'est plus compliqué: penser à trouver une diagonal récursivement

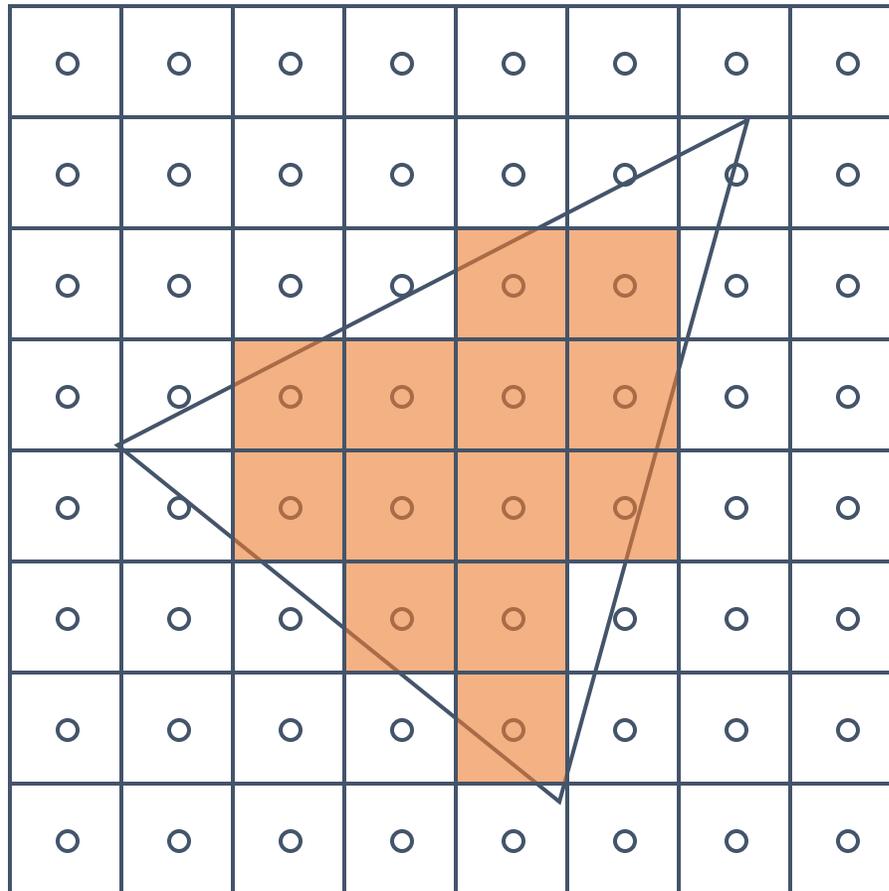


# C'EST QUOI, SCAN CONVERSION? (ALIAS RASTERIZATION)

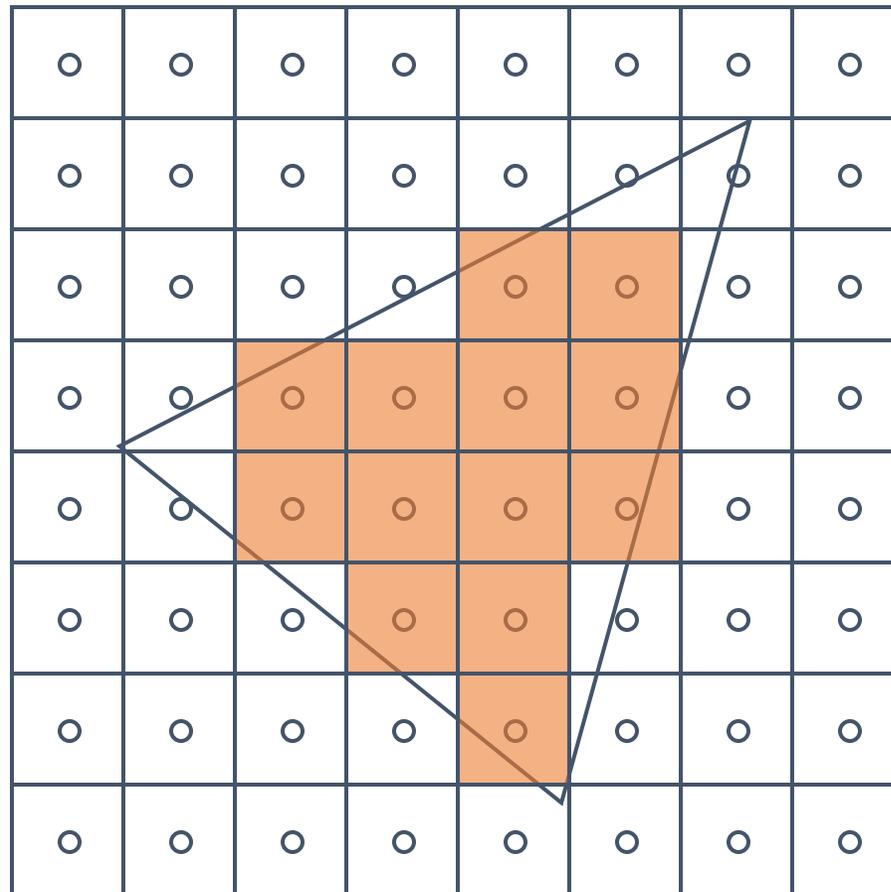
- L'affichage est discret



- une possibilité

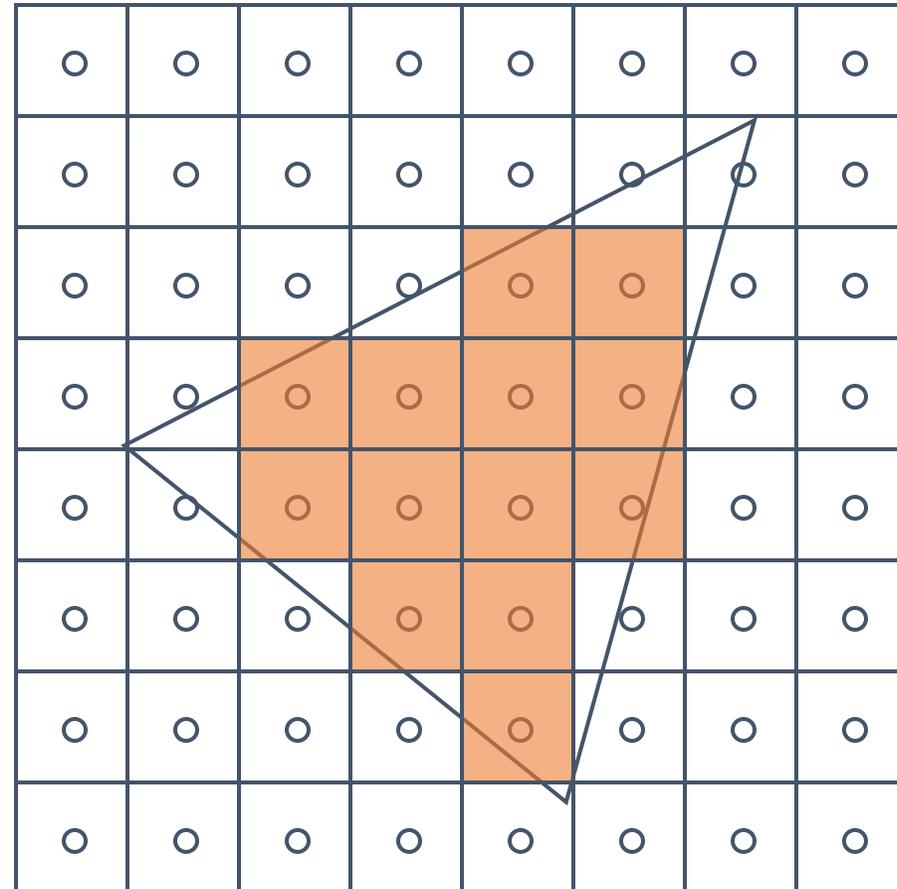


# COMMENT SAVOIR QUELS PIXELS SE TROUVENT À L'INTÉRIEUR



# COMMENT VÉRIFIER SI UN PIXEL EST À L'INTÉRIEUR?

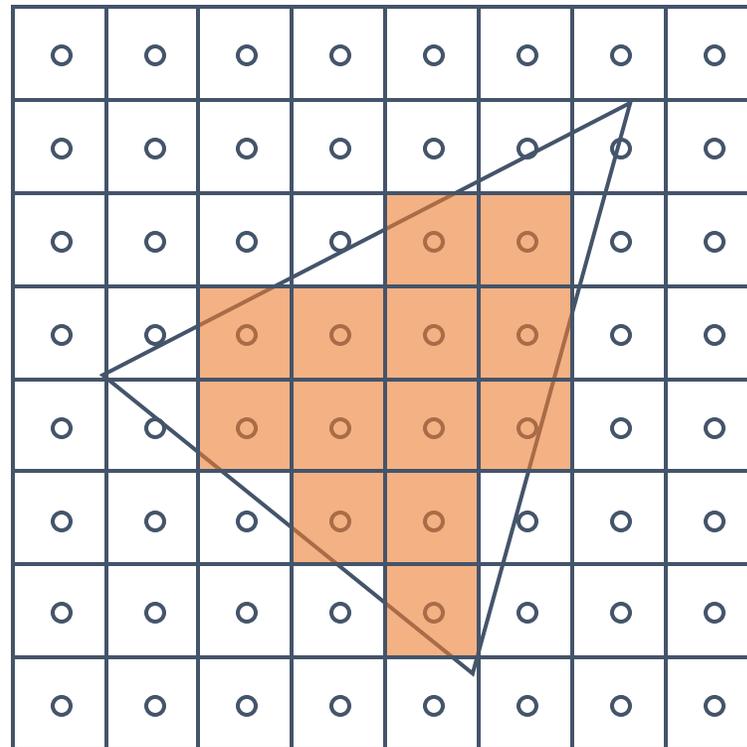
- Utiliser l'équation implicite:
  - $Ax + By + C = 0$
- Comment trouver  $A, B, C$ ?
- L'orientation?



# COMMENT VÉRIFIER SI UN PIXEL EST À L'INTÉRIEUR?

Un point est à l'intérieur  $\Leftrightarrow$

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



# COMMENT TRAITER LA FRONTIÈRE

# COMMENT TRAITER LA FRONTIÈRE

- S'il y a deux triangles qui partagent un segment, la rasterization doit être cohérente
  - Aucun pixel affiché deux fois
  - Pas de trou
- Des idées pour la stratégie?

# BONUS 2

- Quel est le nombre minimal de pixels qui seront affichés pour le triangle suivant:

$$\begin{aligned}P_1 &= (0.5, 0.5) \\P_2 &= (99.5, 100.5) \\P_3 &= (-98.5, 100.5)\end{aligned}$$

- Prouvez-le!
- Une seule solution correcte sera acceptée
- +5% à la note finale

# SCAN CONVERSION NAÏVE

- Tester chaque pixel coûte cher
- De meilleures idées?

# SCANLINE

- La structure de code de base:
  - L'initialisation: calculer les équations des segments et la boîte englobante
  - (La boucle extérieure) Pour chaque ligne de scan dans la boîte...
  - (La boucle intérieure) ...vérifier chaque pixel sur la ligne, évaluer les équations des segments, afficher si toutes sont positives

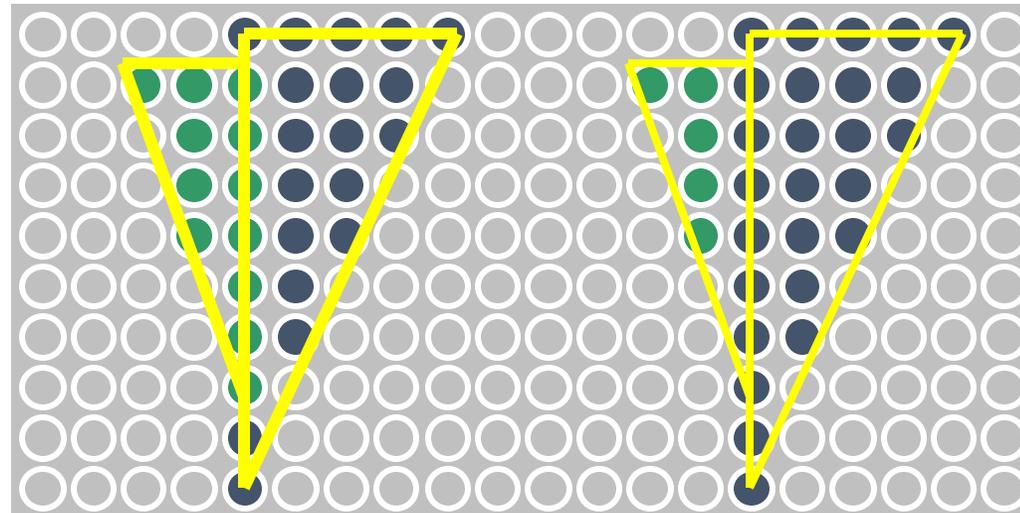
# SCANLINE: CODE

```
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;  
    }  
}
```

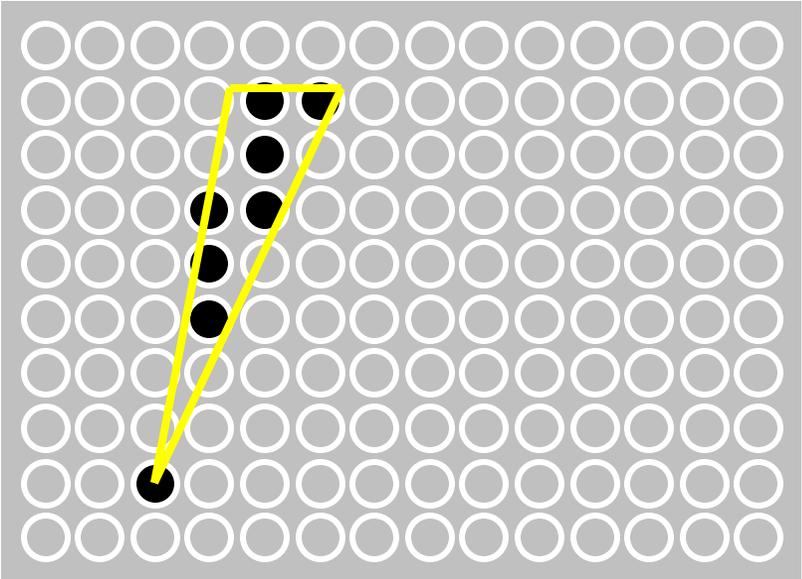
# LES PROBLÈMES DE RASTERISATION

- On sait que les pixels à l'intérieur du triangle doivent être affichés
- Et sur la frontière?

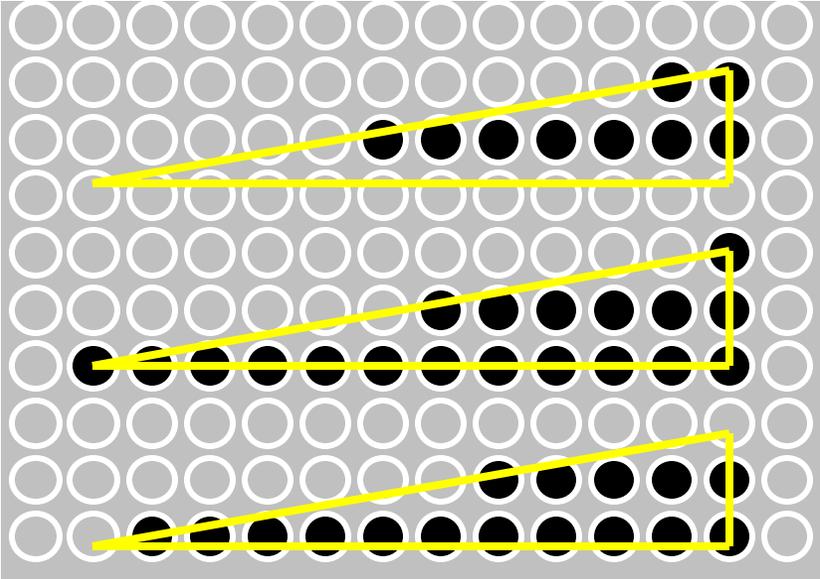


# LES PROBLÈMES DE RASTERISATION

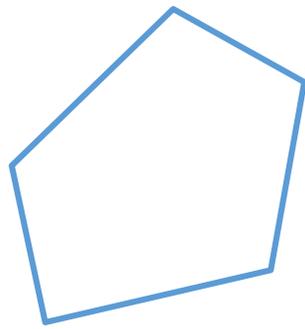
Les petits morceaux  
(*slivers*)



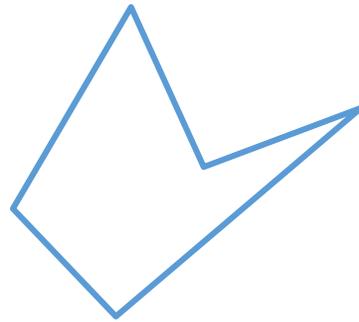
Les morceaux qui bougent



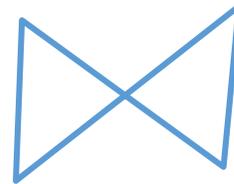
# COMMENT VÉRIFIER SI LE POINT EST À L'INTÉRIEUR?



Convexe  
simple



Non convexe  
simple



Non simple  
(contient une auto-intersection)