

IFT 3355: INFOGRAPHIE

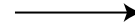
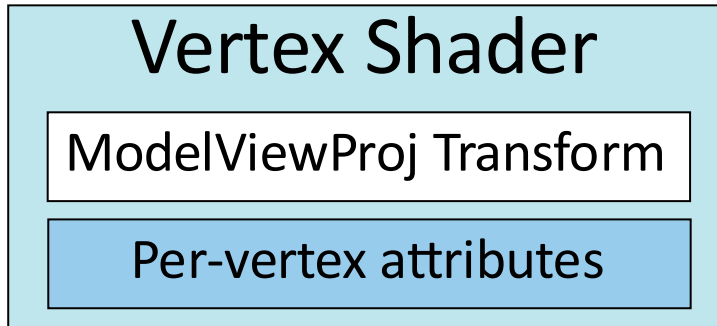
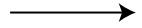
TEST DE PROFONDEUR



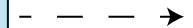
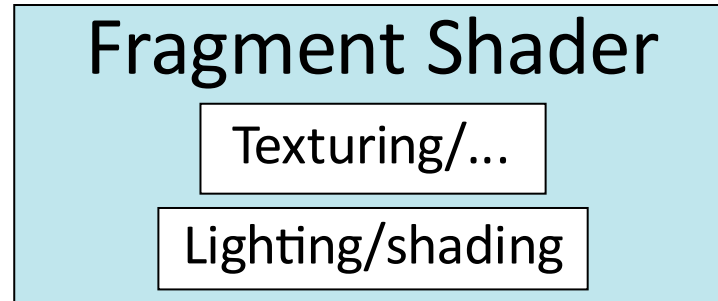
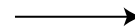
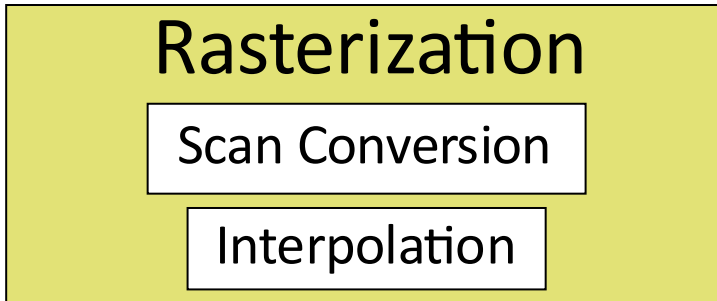
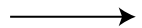
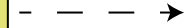
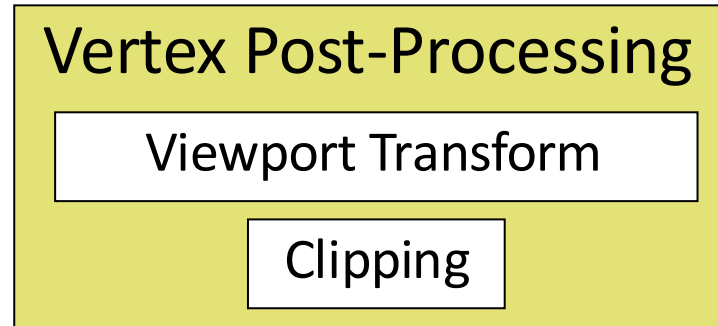
Livre de référence: G:11.1; S: 8

Mikhail Bessmeltsev

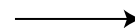
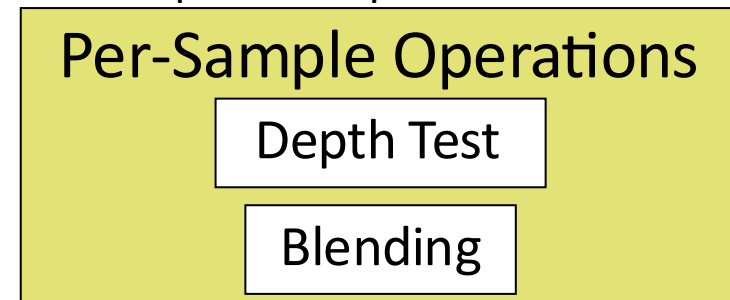
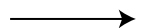
Sommets
et attributs



Traitement de sommets



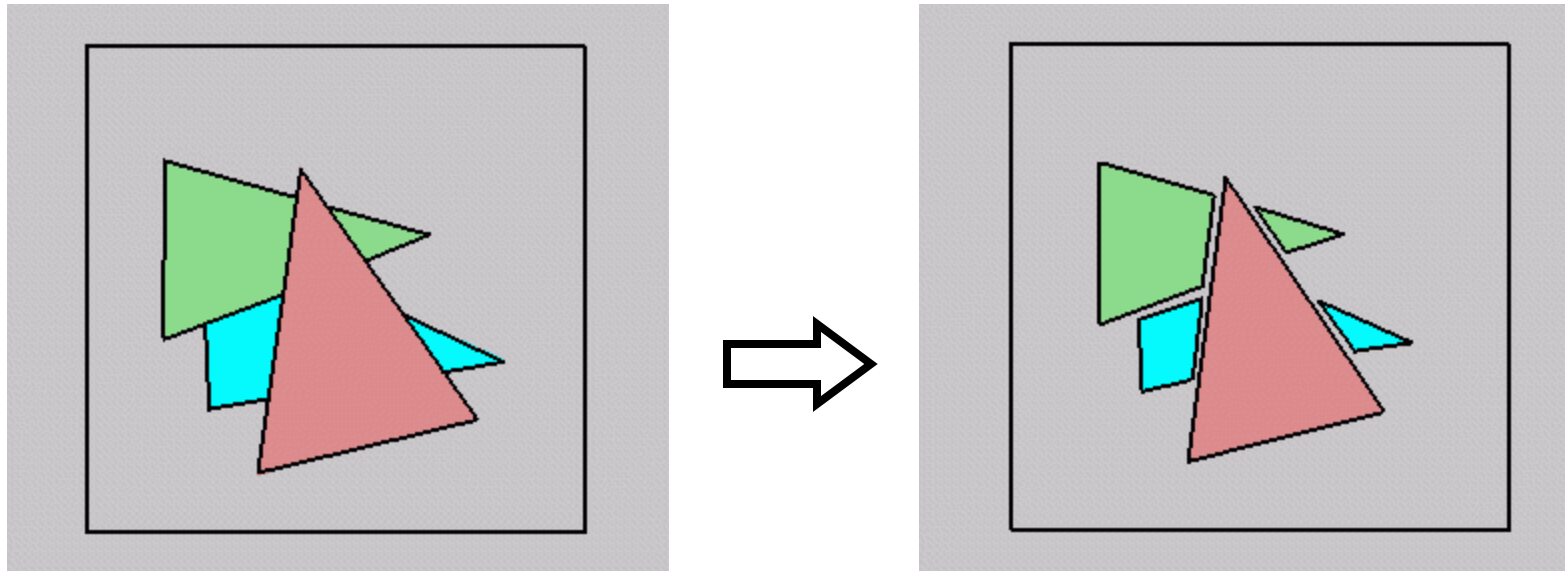
Opérations par échantillon



Framebuffer

VISIBILITÉ

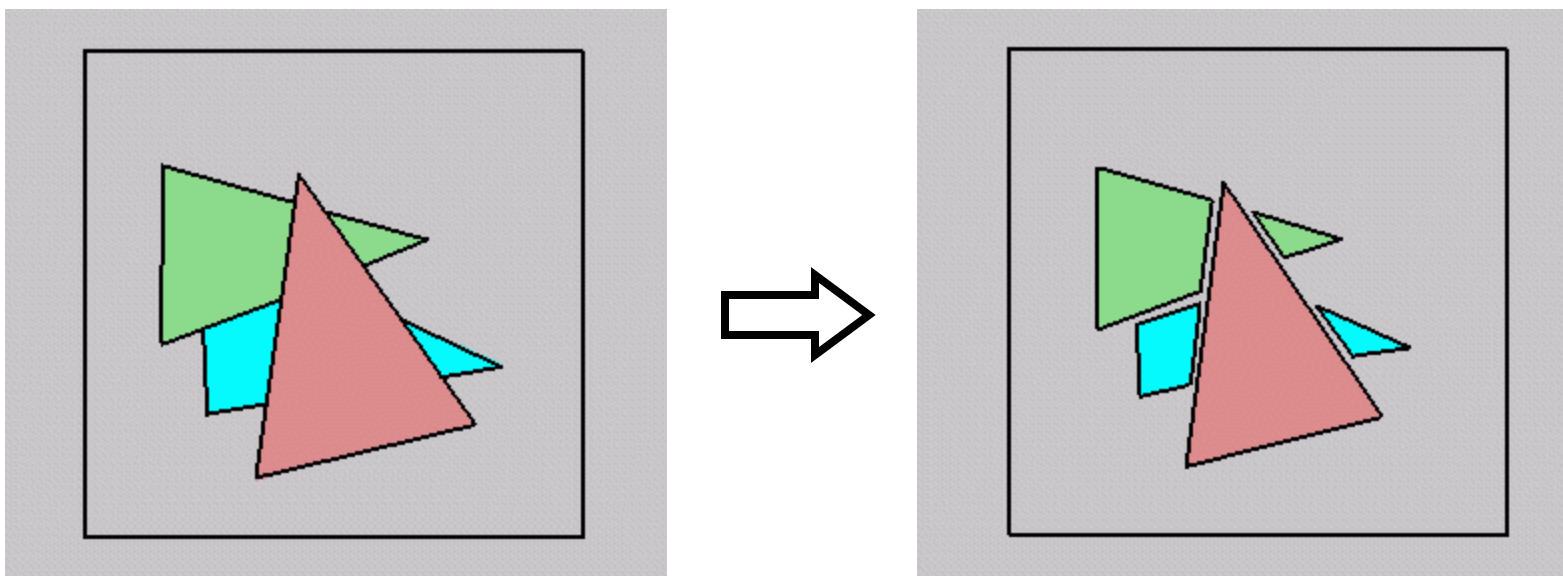
- Pour plusieurs scènes intéressantes, les polygones se chevauchent



- Il faut déterminer quels polygones sont en avant pour rendre l'image correcte

ALGORITHME DU PEINTRE

- Simple: rendre les polygones de l'arrière vers l'avant, en peignant par-dessus les précédents



- Rendre le triangle cyan, puis le bleu, et finalement le rouge
- Est-ce que ça fonctionne en général?

ALGORITHME DU PEINTRE: LES PROBLÈMES

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

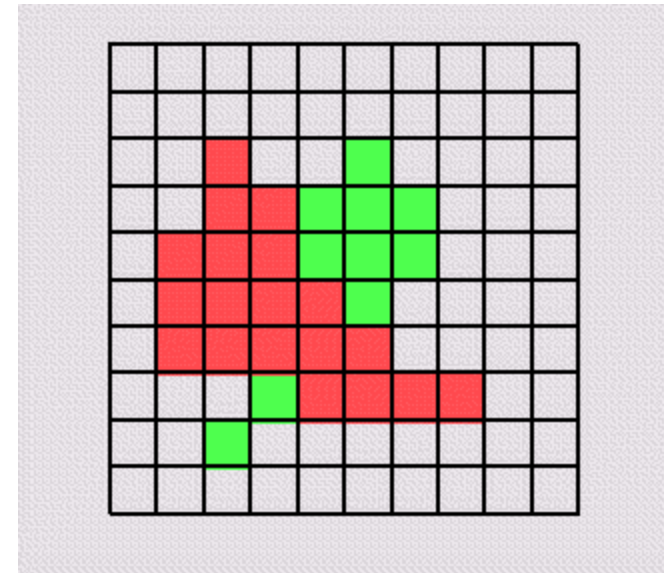
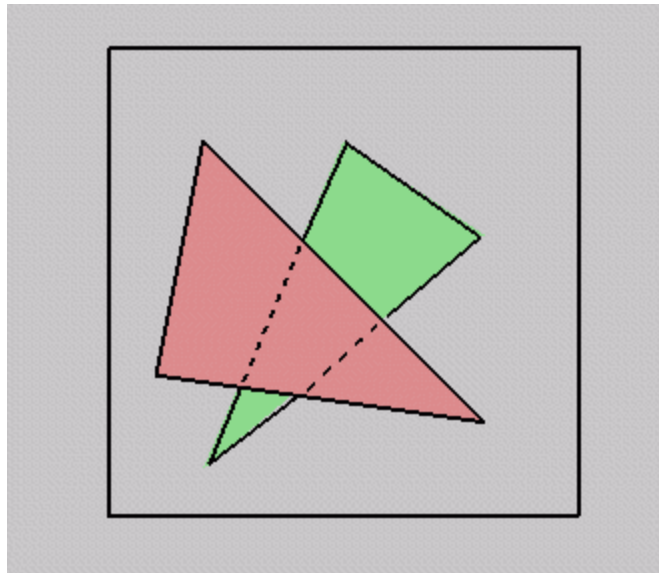


ÉLIMINER LES SURFACES CACHÉES

- Algorithmes en espace objet:
 - En 3D avant la rasterisation
 - E.g. l'algorithme du peintre
 - Souvent sont indépendants de la résolution/la position de la fenêtre
- Algorithmes en espace image:
 - Après la rasterisation, déterminer pour chaque élément d'image, ce qui est le plus près
 - Tampon de profondeur (*Z-Buffer/Depth Buffer*)
 - Peuvent être plus rapides, mais ils dépendent de la résolution de l'image

TAMPON DE PROFONDEUR

- Que se passe-t-il s'il y a plusieurs polygones qui occupent le même pixel?
- Quel objet doit être peint dans le pixel?



TAMPON DE PROFONDEUR

- L'idée: garder la profondeur après la projection
 - Chaque sommet garde sa coordonnée z
 - Par rapport à la caméra
 - Utiliser les coordonnées barycentriques pour calculer z par fragment
 - N'oubliez pas la correction de perspective
- Ou peut-être le fragment shader modifie z

TAMPON DE PROFONDEUR

- En plus du tampon de couleur, stocker le tampon de profondeur
 - *Z-Buffer/Depth buffer*
 - Tout d'abord, initialiser toutes les valeurs à ∞ (profondeur = loin)
- Pendant la rasterisation: interpoler z à l'intérieur du polygone
- Vérifier la valeur dans le tampon de profondeur avant d'écrire la couleur ou la profondeur
 - Ne pas écrire si sa coordonnée z est plus loin que la valeur déjà stockée

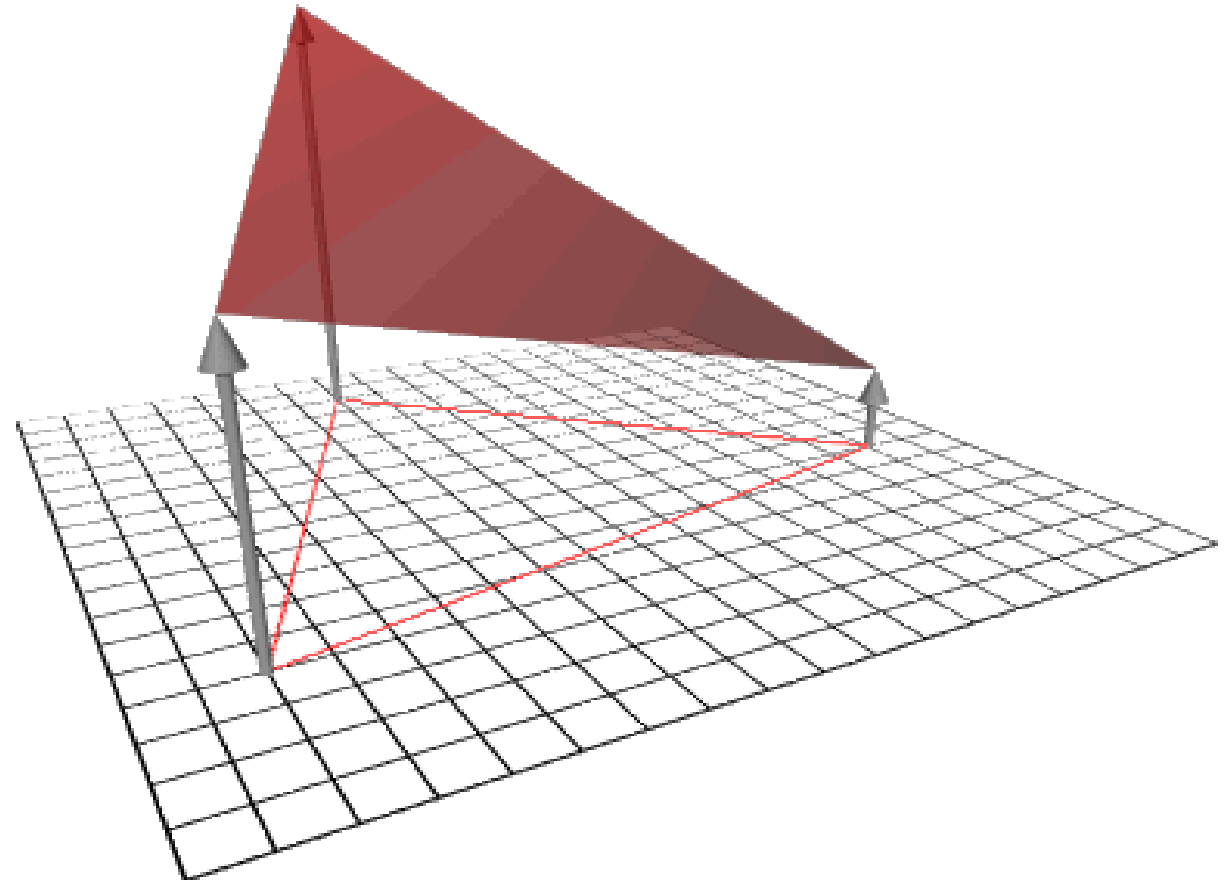
TAMPON DE PROFONDEUR

- Stocker (r, g, b, z) pour chaque pixel
 - Normalement, 8+8+8+24 bits, peut-être plus
 - Souvent **virgule fixe** (équivalent aux nombres entiers)

```
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
        }
    }
}
```

INTERPOLER Z

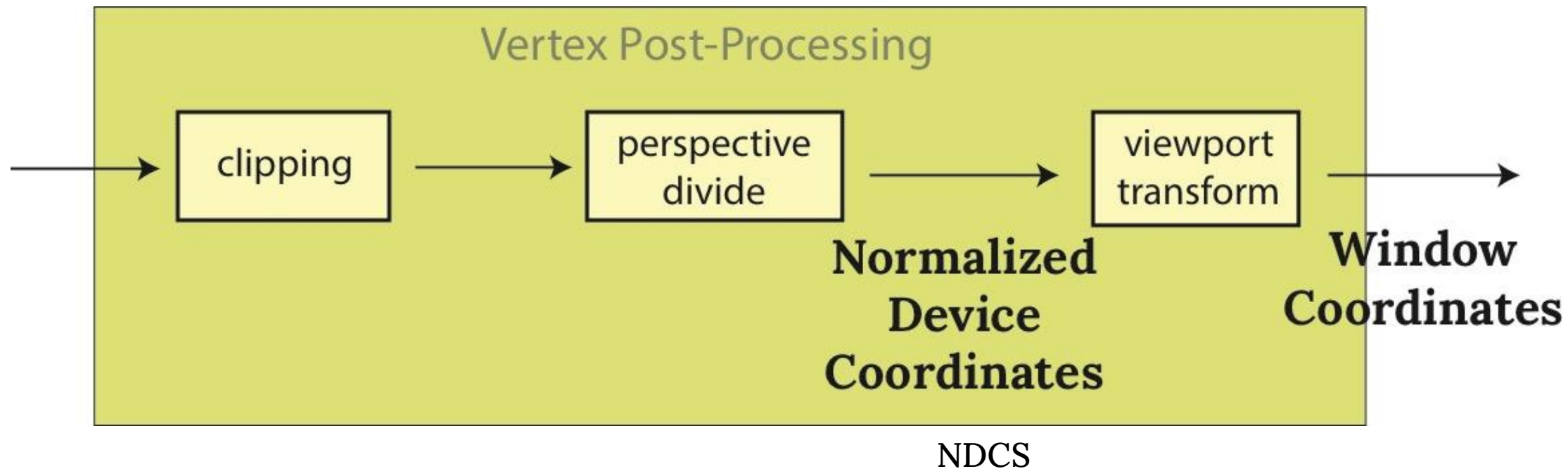
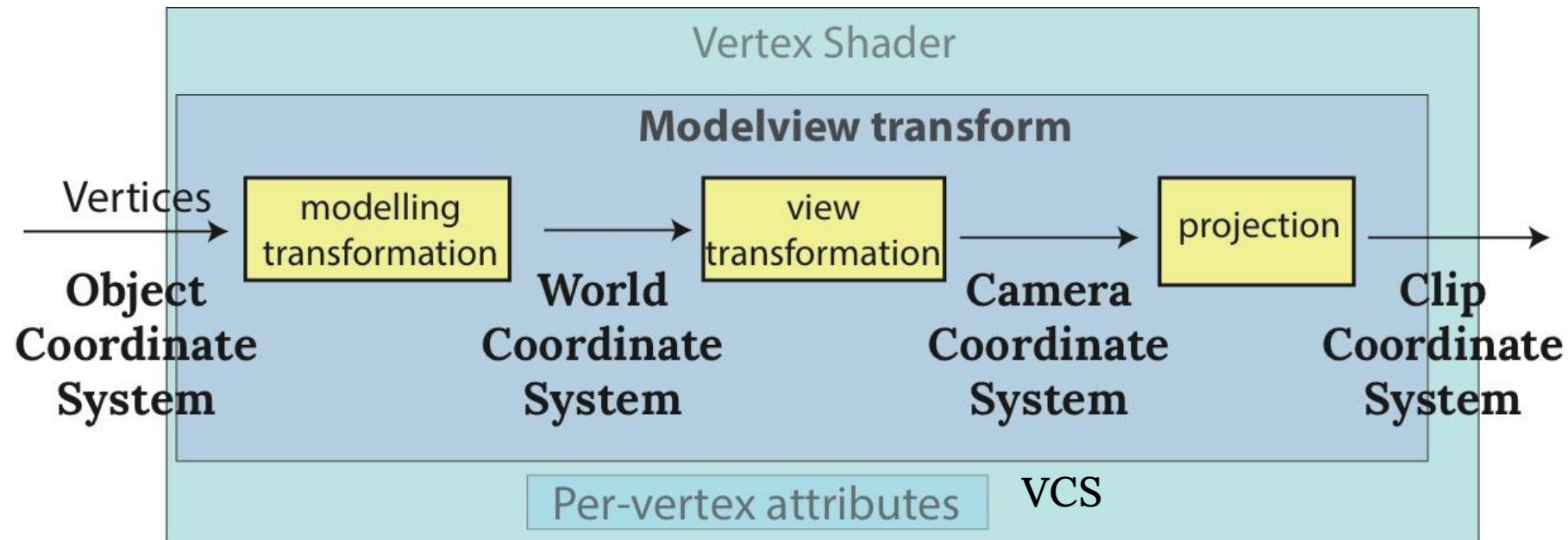
- Utiliser les coordonnées barycentriques
 - Comme tous les autres paramètres
 - E.g. la couleur



TAMPON DE PROFONDEUR (70S)

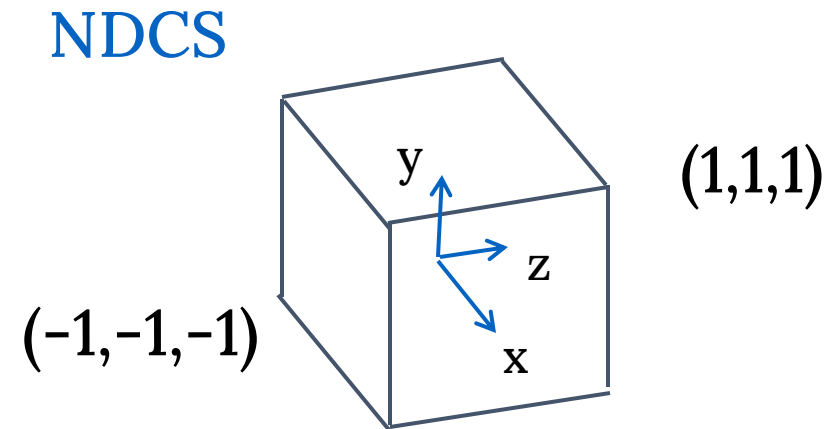
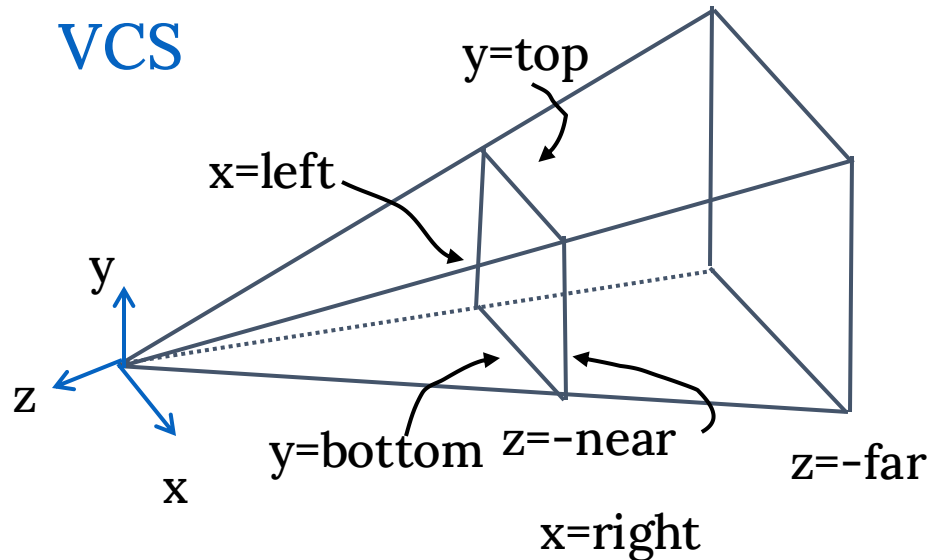
- L'histoire:
 - La mémoire était coûteuse, donc les algorithmes en espace objet ont été proposés
 - Le premier framebuffer 512x512 était >\$50,000!
- Une nouvelle approche radicale à l'époque
 - L'idée générale: déterminer la visibilité par pixel

Rappel



UN SYSTÈME LINÉAIRE

$$\begin{aligned} x = \text{left} &\rightarrow \frac{x'}{w'} = -1; x = \text{right} \rightarrow \frac{x'}{w'} = 1 \\ y = \text{top} &\rightarrow \frac{y'}{w'} = 1; y = \text{bottom} \rightarrow \frac{y'}{w'} = -1 \\ z = -\text{near} &\rightarrow \frac{z'}{w'} = 1; z = -\text{far} \rightarrow \frac{z'}{w'} = -1 \end{aligned}$$



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:

$$\begin{pmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} x_{NDC} \\ y_{NDC} \\ z_{NDC} \end{pmatrix} = \begin{pmatrix} x_{clip}/w_{clip} \\ y_{clip}/w_{clip} \\ z_{clip}/w_{clip} \end{pmatrix}$$

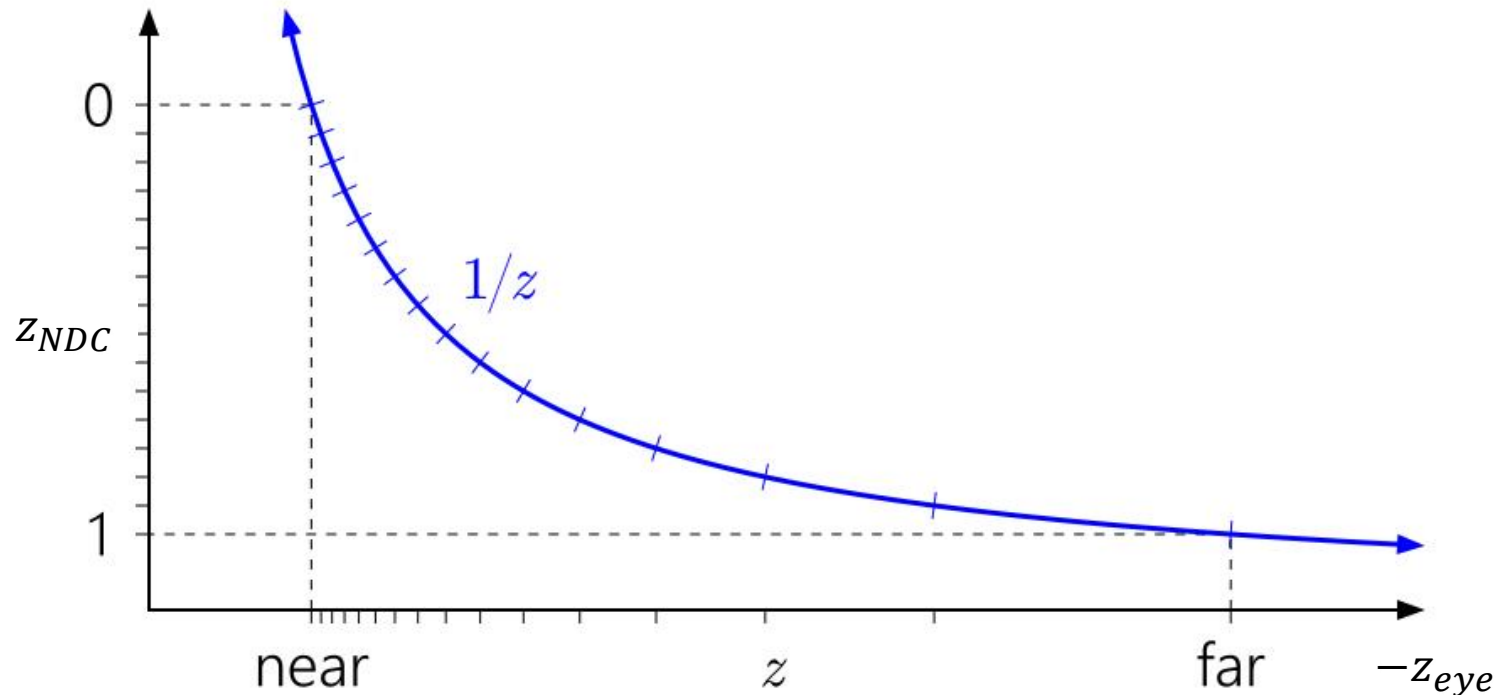
- Donc,

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

LA PRÉCISION DU TEST DE PROFONDEUR

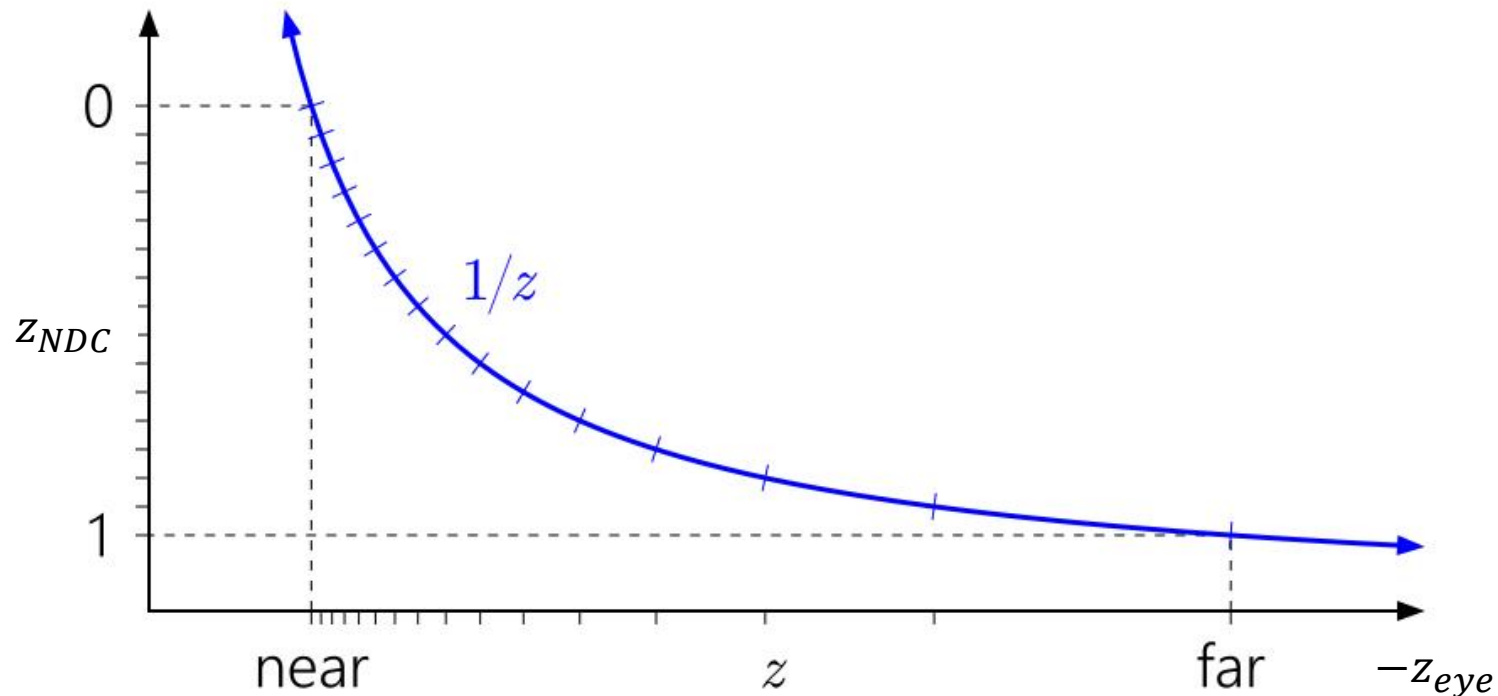
Donc, le tampon de profondeur stocke $1/z$ à la place de z !

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



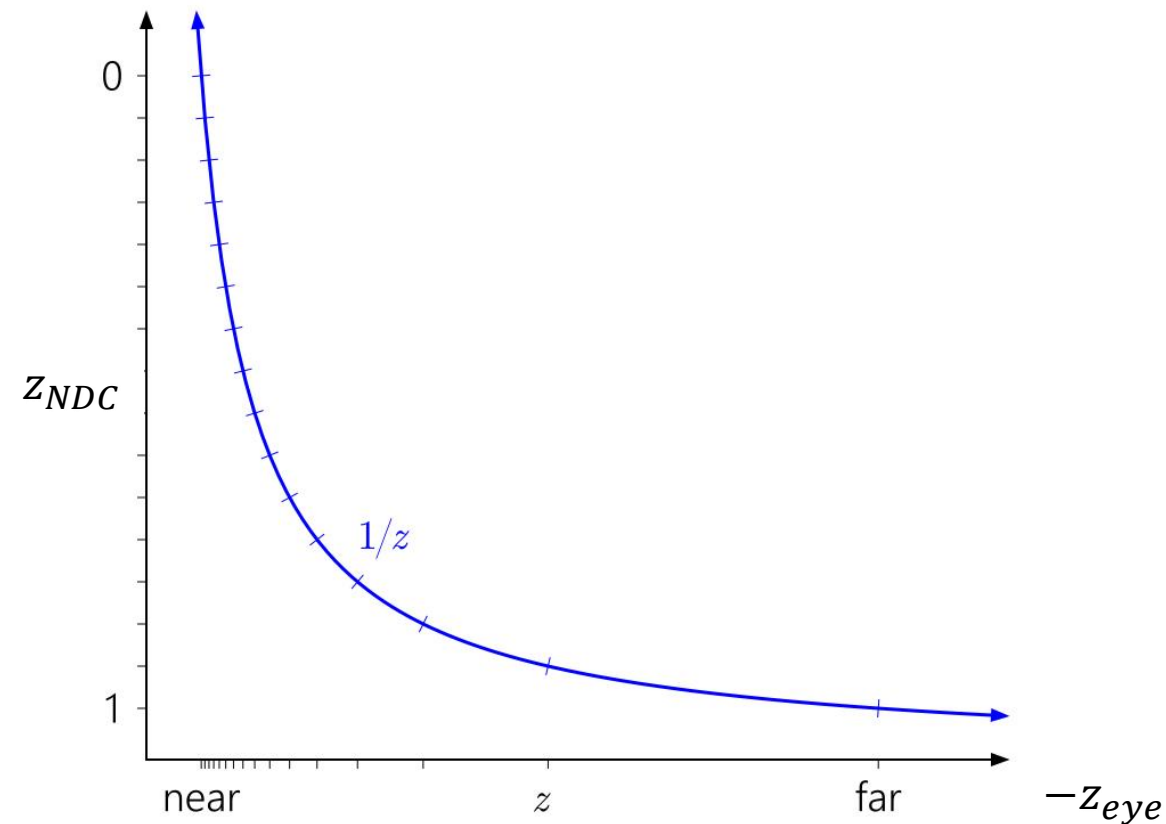
LA PRÉCISION DU TEST DE PROFONDEUR

Si on déplace l'avant-plan plus proche de la caméra:



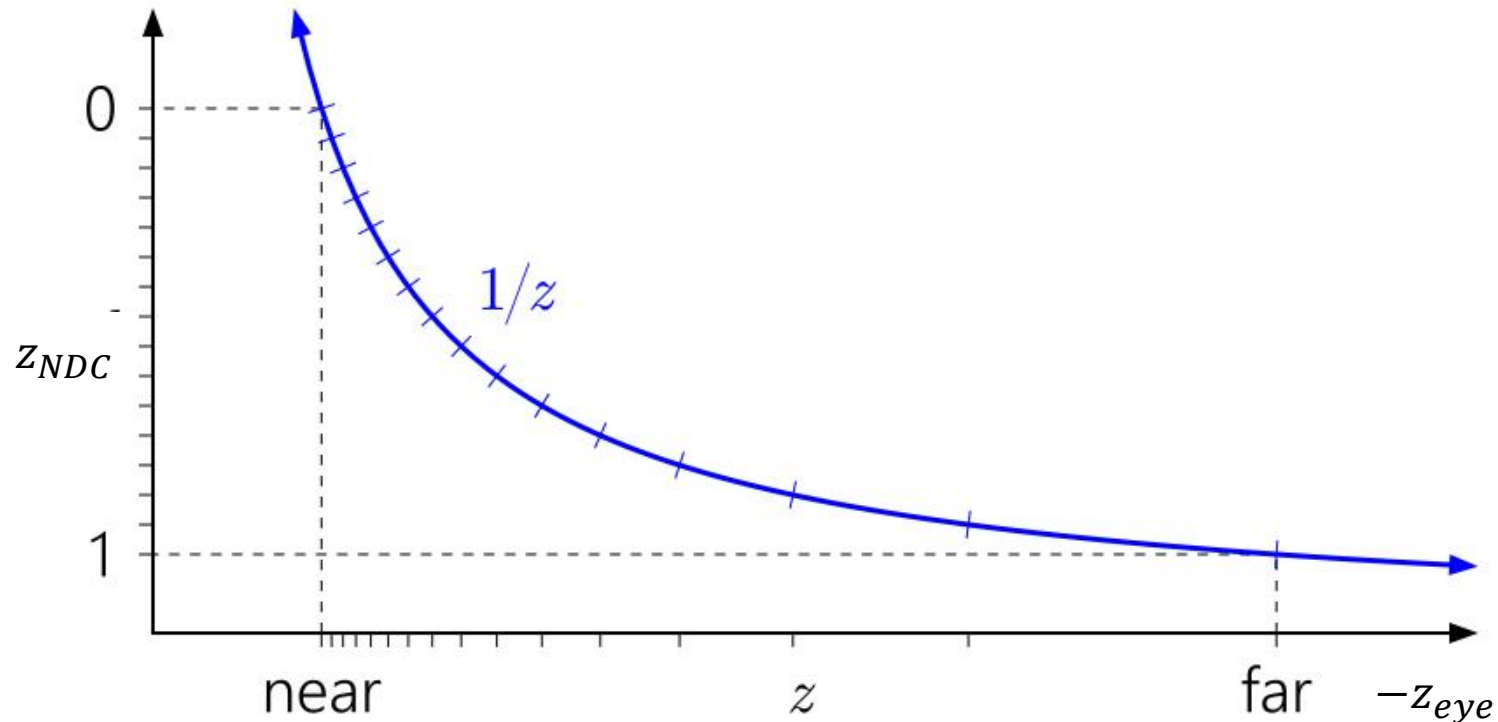
LA PRÉCISION DU TEST DE PROFONDEUR

Si on déplace l'avant-plan plus proche de la caméra:



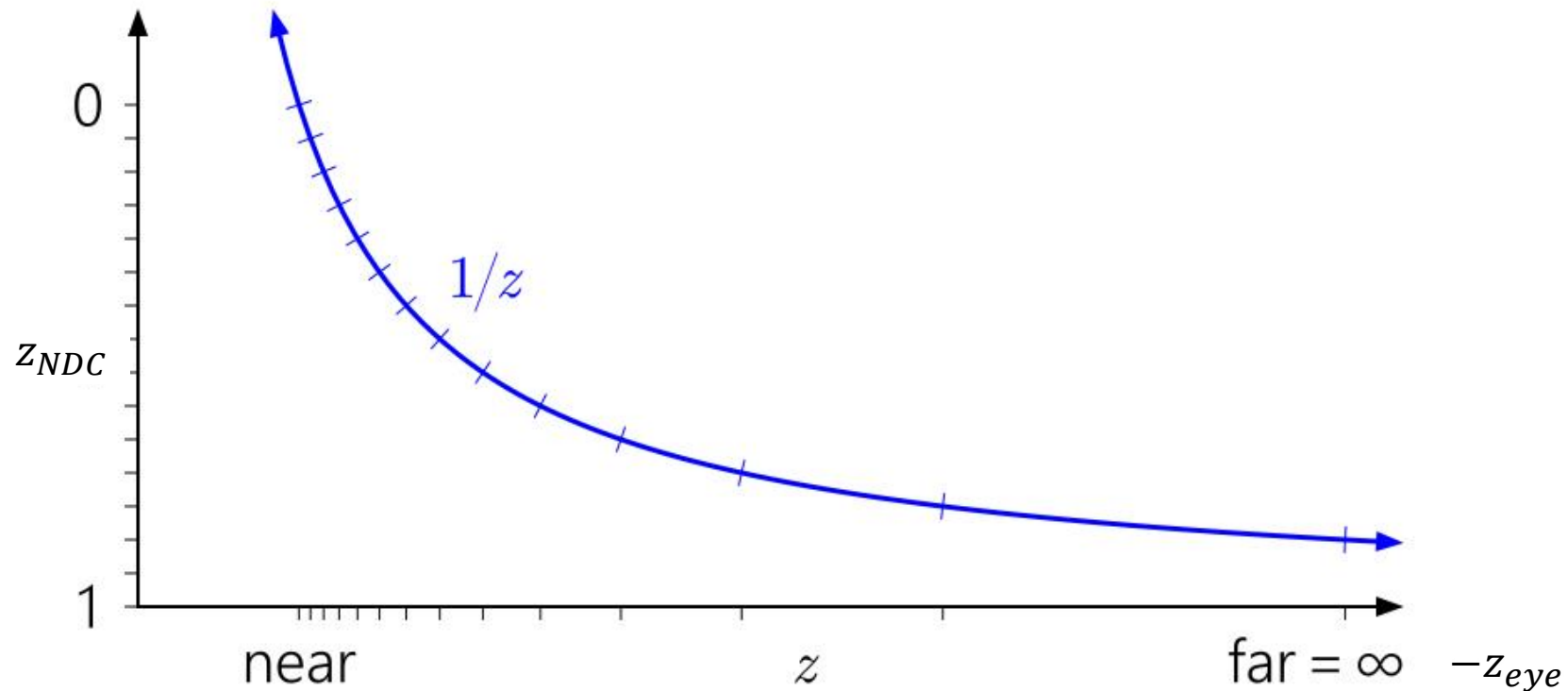
LA PRÉCISION DU TEST DE PROFONDEUR

Si on déplace l'**arrière**-plan plus **loin** de la caméra:

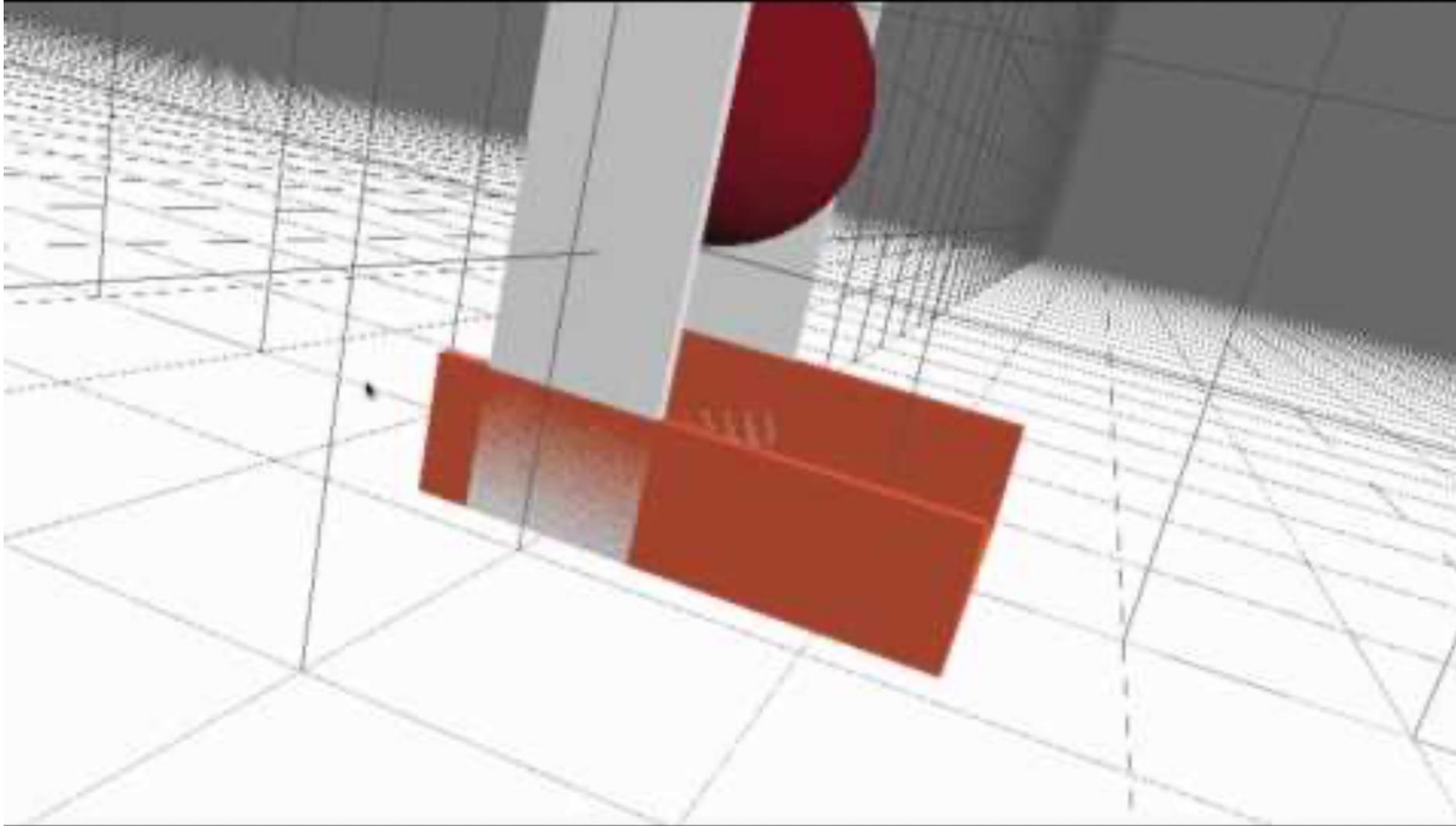


LA PRÉCISION DU TEST DE PROFONDEUR

Si on déplace l'**arrière**-plan plus **loin** de la caméra:



Z-FIGHTING



<https://www.youtube.com/watch?v=CjckWVwd2ek>

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 (quantification)
 - 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

COMMENT LES ARRIÈRE-PLAN ET AVANT-PLAN AFFECTENT LA PRÉCISION

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

$$z_{NDC} = \frac{f + n}{f - n} + \frac{2fn}{(f - n)z_{eye}}$$

$$\frac{dz_{NDC}}{dz_{eye}} = \frac{-2fn}{(f - n)z_{eye}^2} = -\frac{2f}{\left(\frac{f}{n} - 1\right)z_{eye}^2}$$

LES QUESTIONS

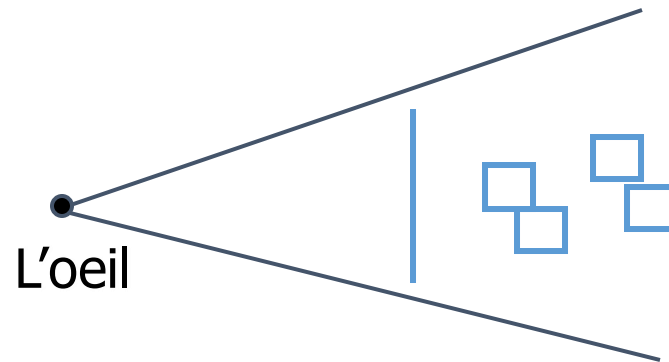
- Combien de mémoire est utilisée par le test de profondeur?
- Est-ce que l'image dépend de l'ordre d'affichage?
- Est-ce que le temps de rendu dépend de l'ordre d'affichage?
- Comment la complexité change en variant de la résolution ou le nombre de polygones?

AVANTAGES DU TEST DE DE PROFONDEUR

- Simple!
- *Hardware parallelism* (parallélisme matériel)
 - Presque tous les GPU prennent en charge
- Les polygones peuvent être traités dans un ordre arbitraire
- Gère facilement l'interpénétration des polygones

DÉSAVANTAGES

- Mauvais pour les scènes avec une complexité élevée
 - Il faut rendre tous les polygones, même si ils sont cachés



- Les segments partagés sont affichés de façon incohérente
 - *Dépend de l'ordre*

DÉSAVANTAGES

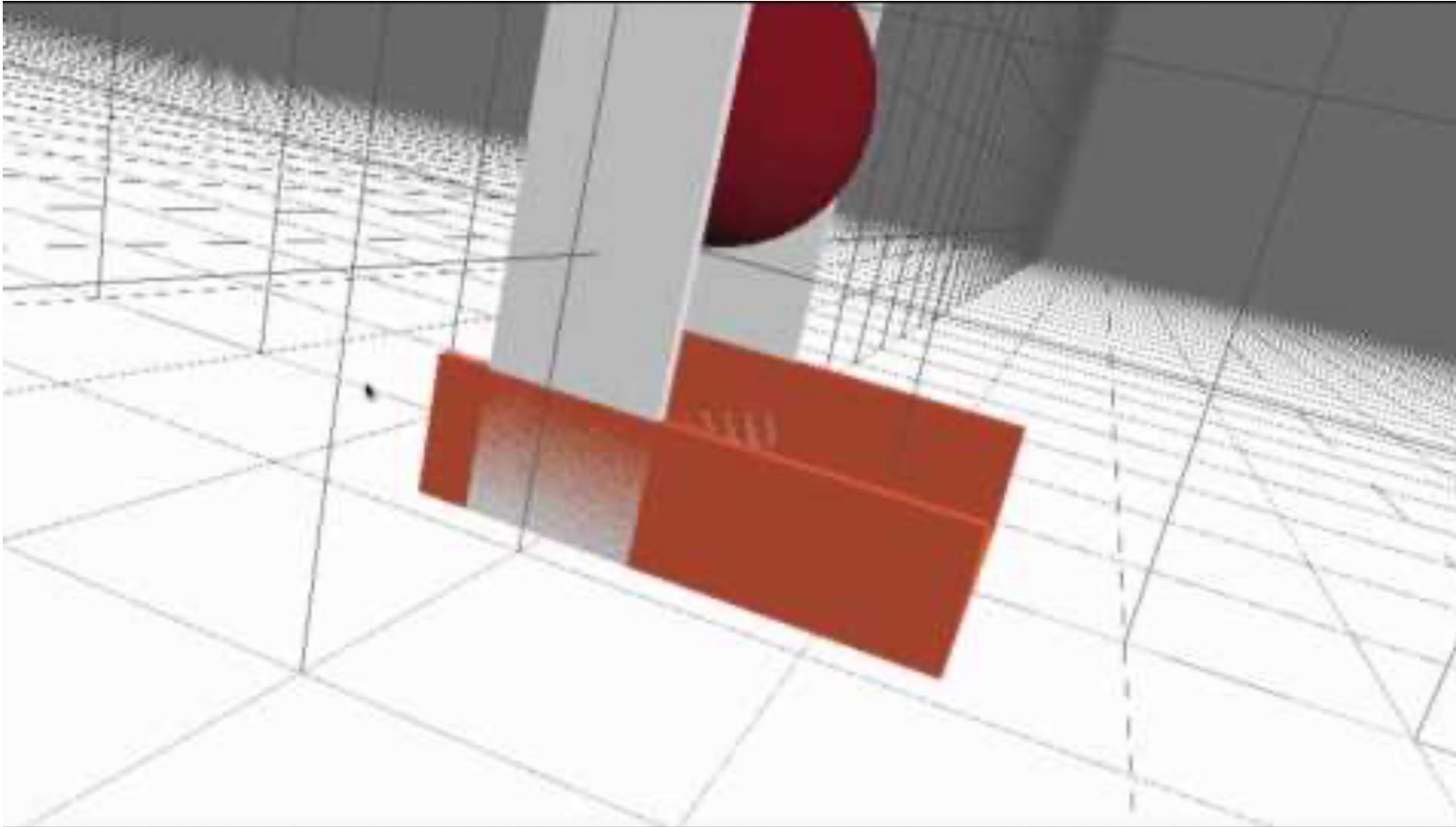
- Mémoire additionnelle pour les tampons
 - (e.g. 1280x1024x32 bits, dépend de la mise en œuvre)
- Il faut avoir de la mémoire rapide
- Il est difficile de simuler des polygones transparents
 - On jette la couleur des polygones derrière le polygone le plus proche
 - OK si les polygones sont triés d'arrière en avant
 - Mais ça prend du temps

ALGORITHMES EN ESPACE OBJET

- Par chaque objet ou polygone
- Indépendant de la résolution
 - Calculer explicitement les parties visibles des polygones
- Tôt dans le pipeline
 - Après *clipping*
- Nécessite un tri en profondeur
 - L'algorithme du peintre
 - Les arbres BSP

Z-FIGHTING – QUOI FAIRE?

- 3 stratégies



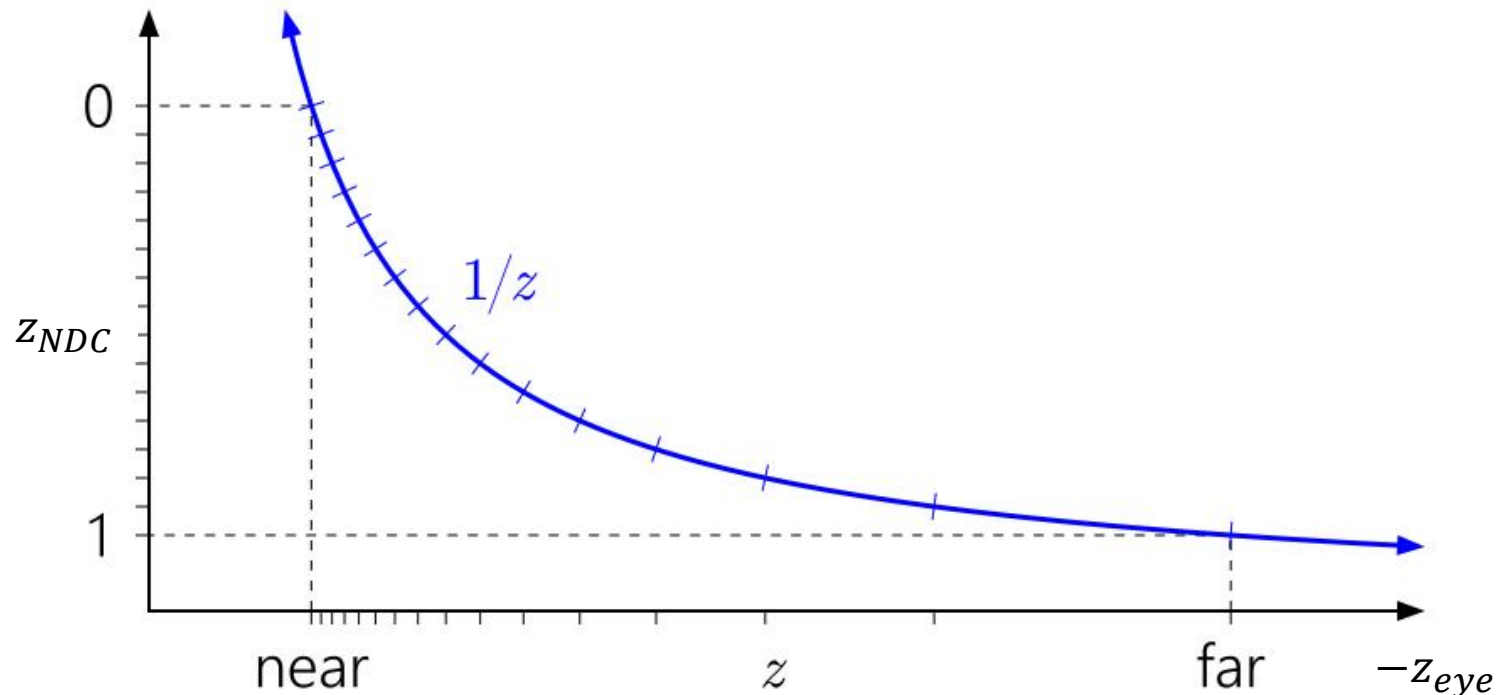
Z-FIGHTING – QUOI FAIRE?

- 3 stratégies:
 - Déplacer les objets
 - Déplacer l'avant-plan plus loin
 - Utiliser plus de bits pour le tampon de profondeur

LA PRÉCISION DU TEST DE PROFONDEUR

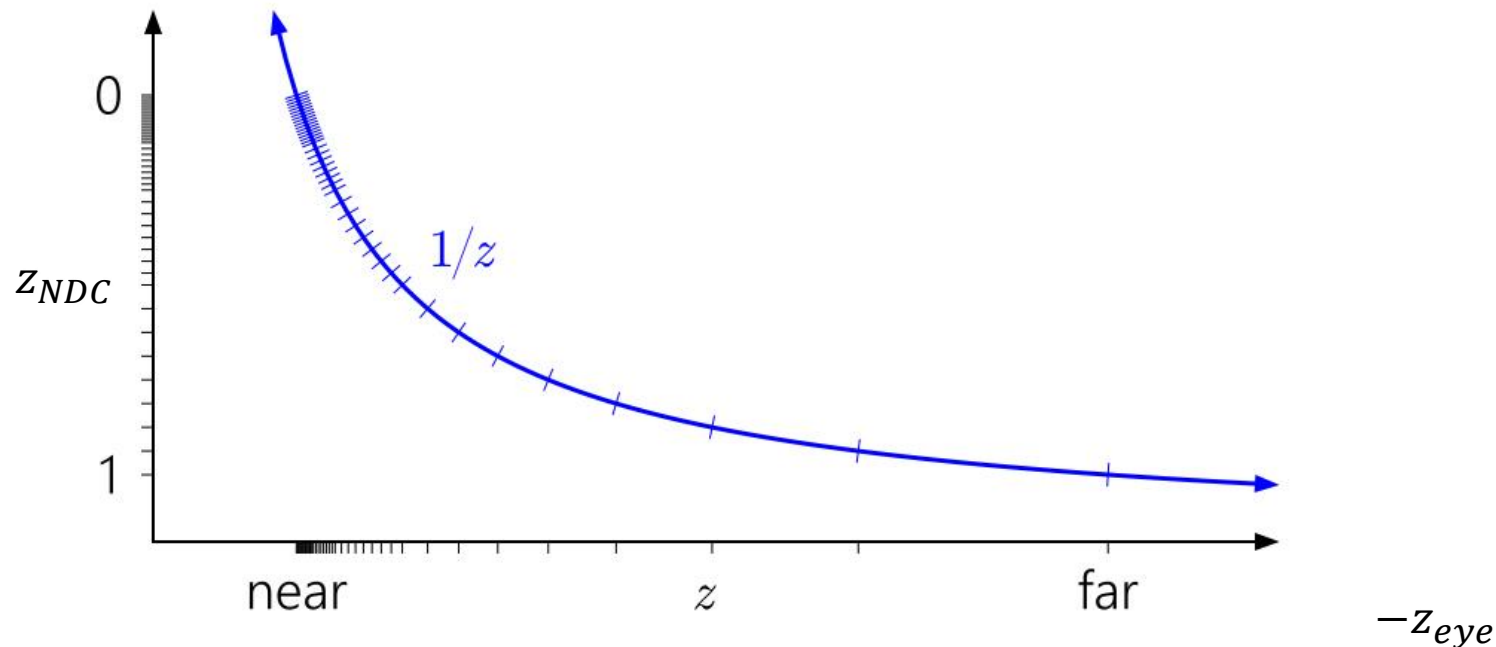
Donc, le tampon de profondeur stocke $1/z$ à la place de z !

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



TAMPON EN NOMBRES À VIRGULE FLOTTANTE

Plus de précision, mais exactement là où on ne le veut pas

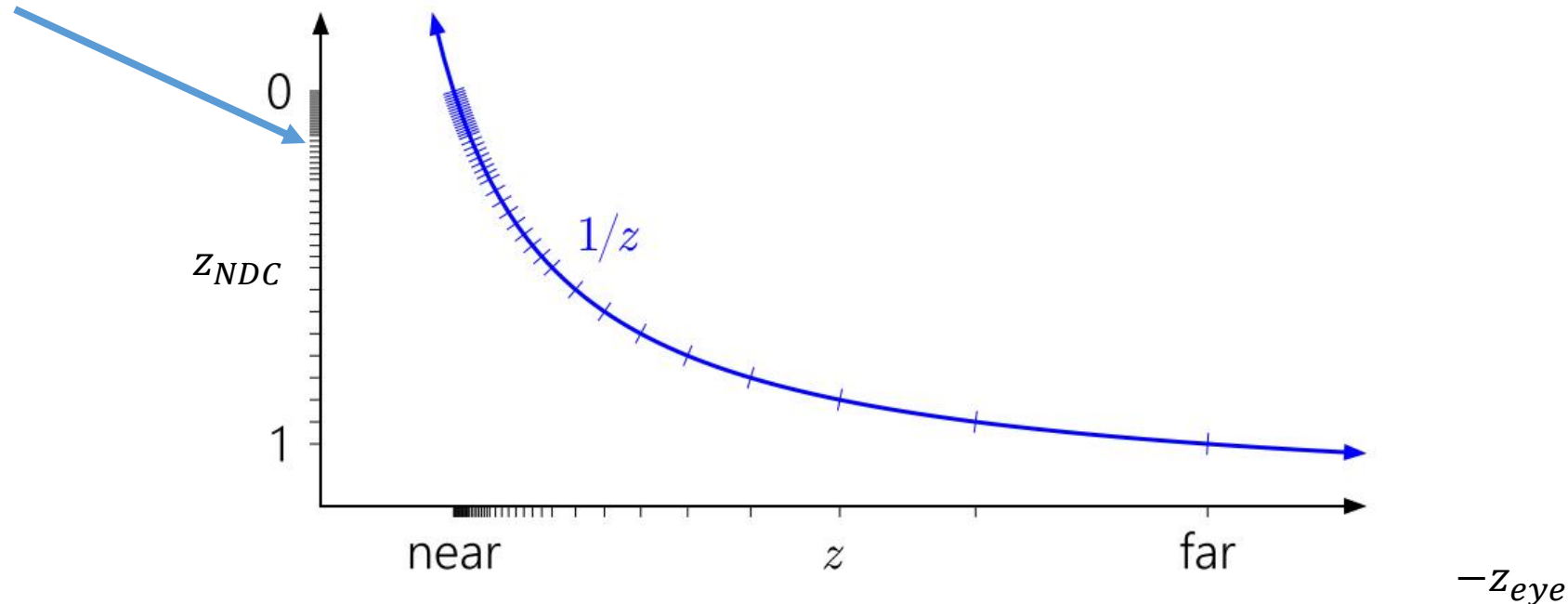


TAMPON EN NOMBRES À VIRGULE FLOTTANTE

La précision est là où on ne le veut pas

$$(-1)^S \times M \times 2^{(E-127)}$$

Densité quasi-logarithmique



TAMPON EN NOMBRES À VIRGULE FLOTTANTE

Solution: switcher 0 et 1 (par une fonction linéaire):

Densité quasi-logarithmique

