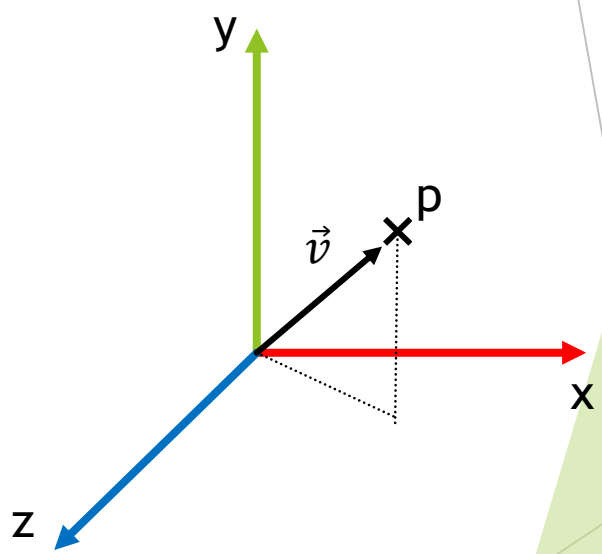
Rendu 3D –Les transformations

David Murray

Source: Gael Guennebaud, Pierre Bénard

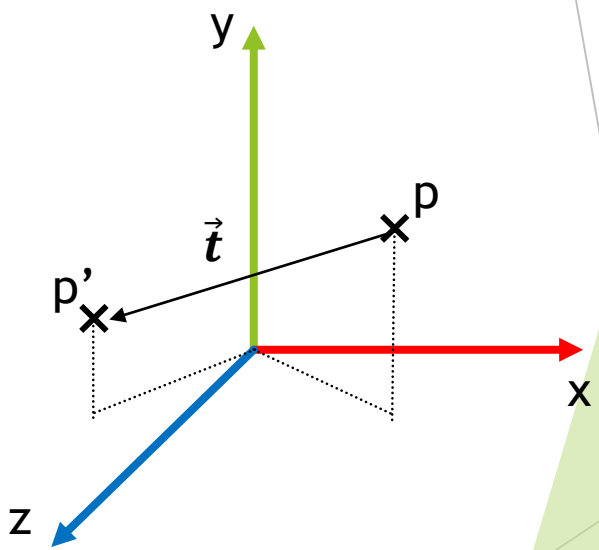
Rappels de Mathématiques

- Point euclidien/Coordonnées homogènes:
  - p=(x, y, z, w) ~ p'=(x/w, y/w, z/w), si w ≠ 0
  - En général en 3D, on définit un point par: (x, y, z, 1)
- Vecteur/Direction:
  - $\vec{v}$=(x, y, z, 0)
- Transformation:
  - Matrice de changement de base ! ☺



Rappels de Mathématiques

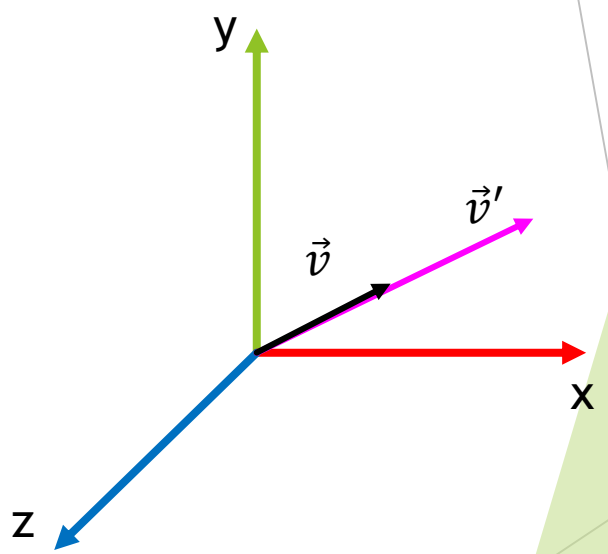
- Translation d'un vecteur $\vec{t}$
  - p' = p + $\vec{t}$
  - Equivalent à p' = T * p
  - T = $\begin{pmatrix} 1 & 0 & 0 & \boldsymbol{tx} \\ 0 & 1 & 0 & \boldsymbol{ty} \\ 0 & 0 & 1 & \boldsymbol{tz} \\ 0 & 0 & 0 & 1 \end{pmatrix}$ -> T * p = $\begin{pmatrix} x + \boldsymbol{tx} \\ y + \boldsymbol{ty} \\ z + \boldsymbol{tz} \\ 1 \end{pmatrix}$



  - Attention: Pour un vecteur, aucun impact: $\overrightarrow{v'}$ = T * $\vec{v}$ = $\begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{z} \\ \boldsymbol{0} \end{pmatrix}$

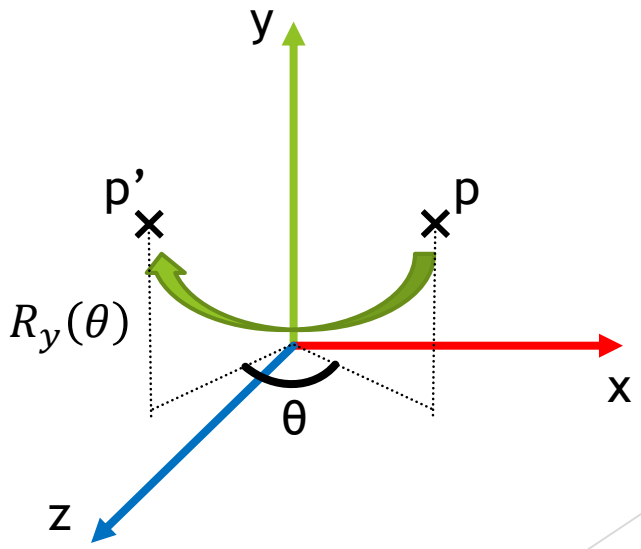
Rappels de Mathématiques

- Mise à l'échelle d'un facteur s(sx, sy, sz)
  - p' = p * s
  - Equivalent à p' = S * p
  - S = $\begin{pmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ -> S * p = $\begin{pmatrix} x * sx \\ y * sy \\ z * sz \\ 1 \end{pmatrix}$
  - Si sx = sy = sz = s, on peut utiliser:
  - S = $\begin{pmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{1}{s} \end{pmatrix}$



Rappels de Mathématiques

- Rotation d'un angle θ autour d'un axe $\vec{u}(ux, uy, uz)$
  - p' = R(θ, u) * p
  - $R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
  - $R_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
  - $R_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$



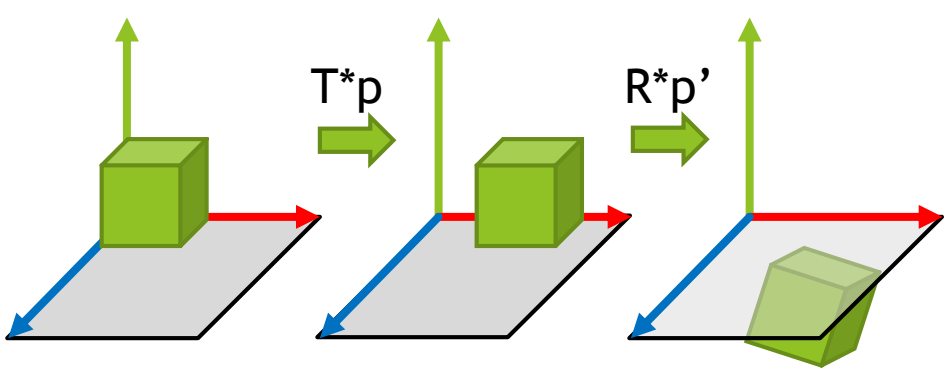
Rappels de Mathématiques

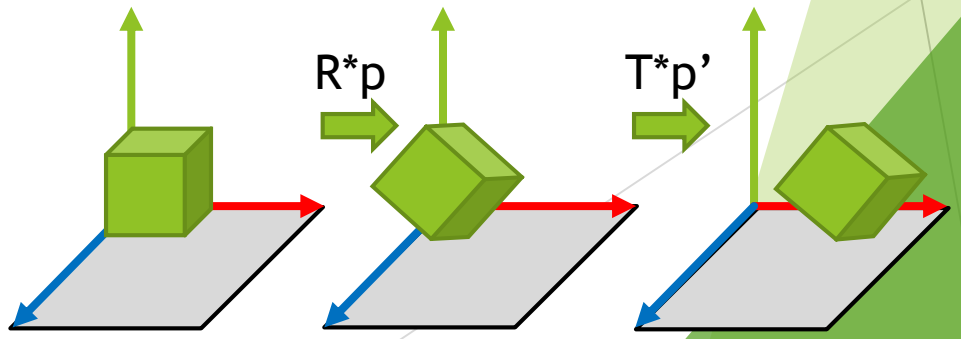
- Rotation d'un angle θ autour d'un axe $\vec{u}(u_x, u_y, u_z)$
  - Plusieurs solutions:
    - Décomposer $R(\theta, \vec{u}) = R_x(\alpha)\ R_y(\beta)\ R_z(\gamma)$ -> besoin de trouver α, β, γ -> très compliqué à automatiser
    - Simplifier: $r(\theta, \vec{u}) = P(\vec{u}) + \cos\theta\ (I\ - P(\vec{u})) + \sin\theta\ Q(\vec{u})$
      - $P(\vec{u}) = \vec{u}\vec{u}^t = \begin{pmatrix} u_x^2 & u_x u_y & u_x u_z \\ u_x u_y & u_y^2 & u_y u_z \\ u_x u_z & u_y u_z & u_z^2 \end{pmatrix}; I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}; Q(\vec{u}) = \begin{pmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{pmatrix}; P(\vec{u}) = I + Q(\vec{u})^{\,2}$
    - Utiliser un quaternion -> la solution la plus efficace mais la plus compliqué à apréhender

Rappels de Mathématiques

- Que faire si on souhaite appliquer une translation puis une rotation ?
  - Produit matriciel !
  - p'= R*T*p

- ATTENTION: le produit matriciel n'est pas commutatif !
- Les matrices de transformation doivent être dans le bon ordre:
  - R*T*p ≠ T*R*p



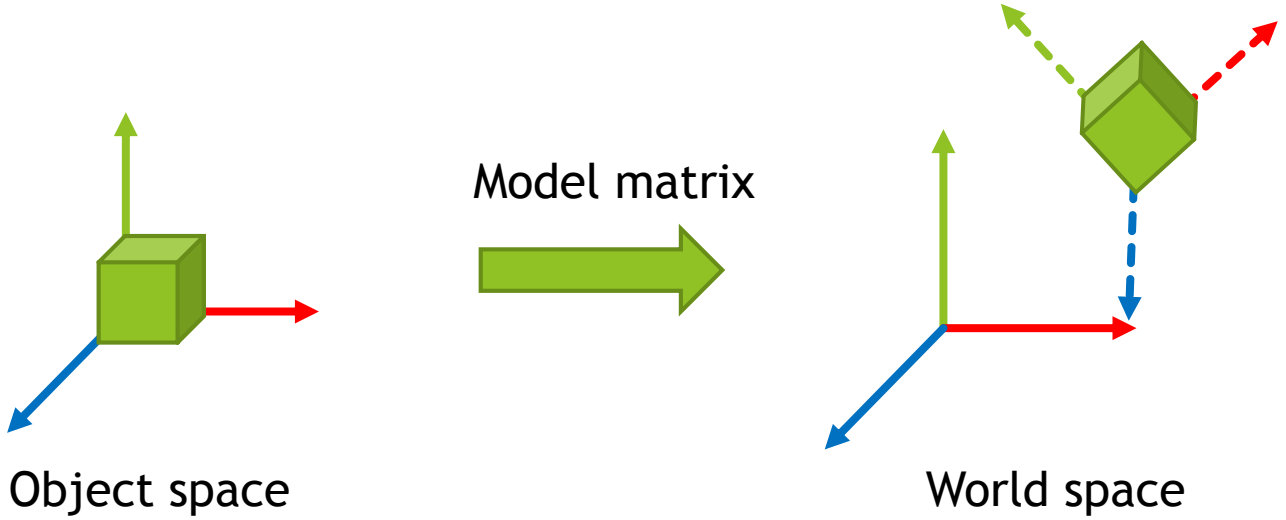


Les transformations: pourquoi ?

- Pourquoi appliquer des transformations ?
- Permet de créer chaque objet indépendamment des autres
  - Chacun dans son propre repère: « Object space »
- Il suffit alors d'appliquer la même transformation à tous les sommets de l'objet pour le placer dans la scène
  - Dans le repère monde: « World space »
- L'ensemble des transformations: « Model matrix »

Les transformations: résumé

- On sait maintenant placer les objets dans la scène
  - Passer du repère objet au repère monde: « Object space » vers« World space »
- A retenir:

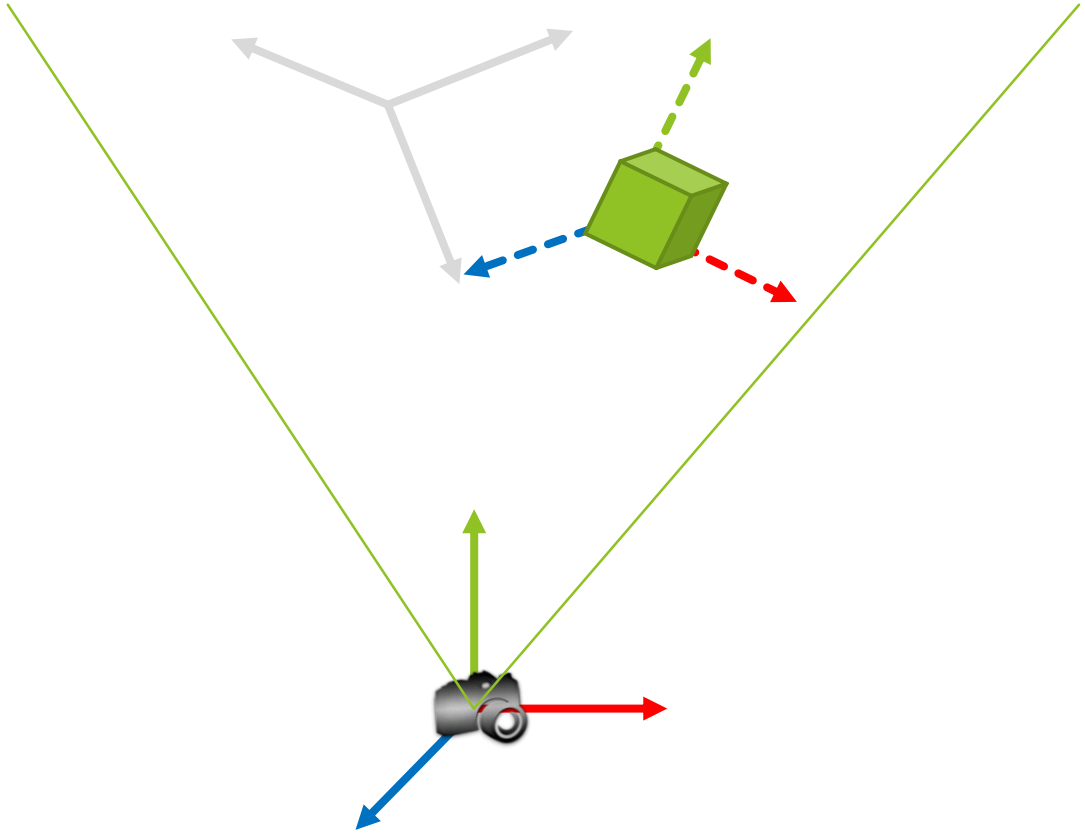


La caméra

- Et maintenant ? Ça sert à quoi ?
- Tous les objets sont dans le même repère ☺
- Il ne reste plus qu'à les exprimer dans l'espace caméra puis les projeter dans le plan de celle-ci !
- Comment ?
  - View matrix -> espace caméra
  - Projection matrix -> projection sur la matrice de pixel

La caméra

- View matrix: « world space » vers « camera space »



La caméra

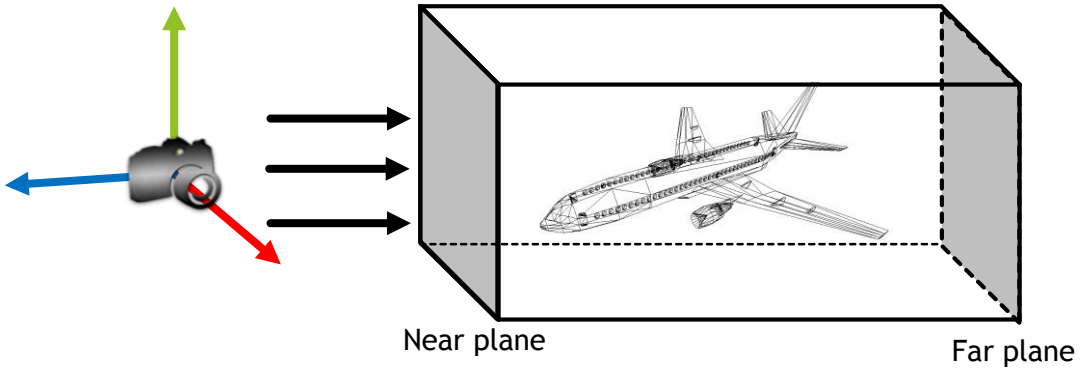
- View matrix: comment la calculer ?
- La caméra est un objet !

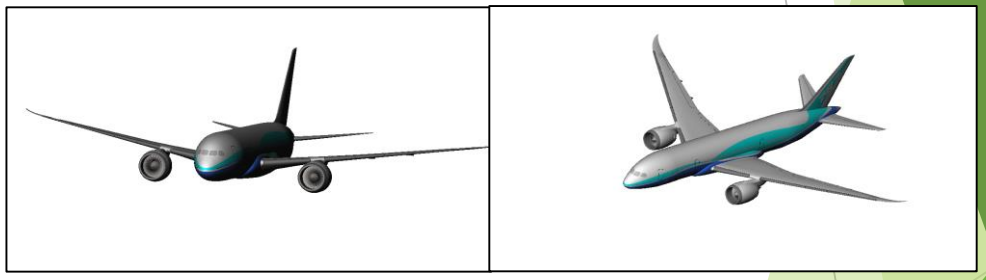
- On calcule la « model matrix » de la caméra en fonction de sa position dans le repère monde

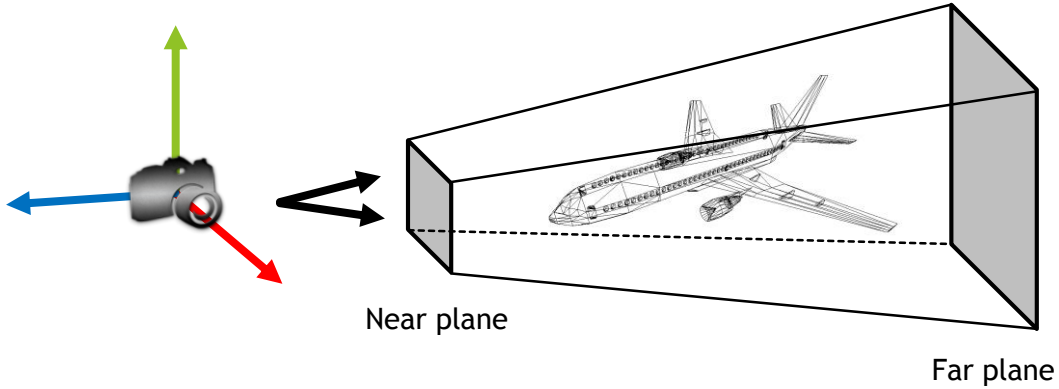
- On inverse cette matrix -> View matrix

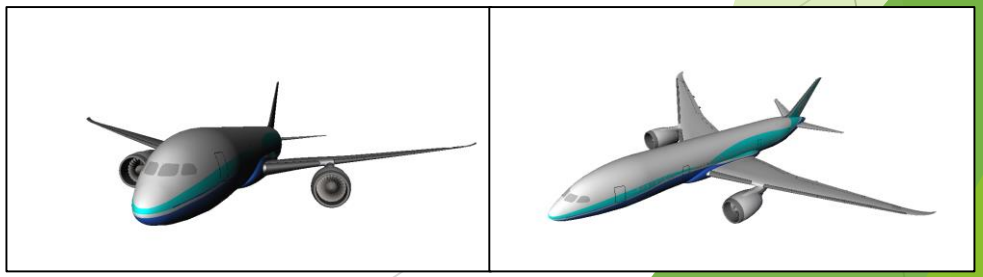
La caméra

- Il ne reste plus qu'à projeter les objets sur le plan de la caméra
- 2 possibilités: orthographique et perspective









La caméra

- Projection matrix:
- 4 paramètres: Field Of View (FOV), Near plane depth, Far plane depth, Aspect

  - n = near plane
  - f = far plane
  - t = $n * \tan\left(\frac{\pi}{180} * \frac{FOV}{2}\right)$
  - b = -t
  - r = t*aspect
  - l = -r

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

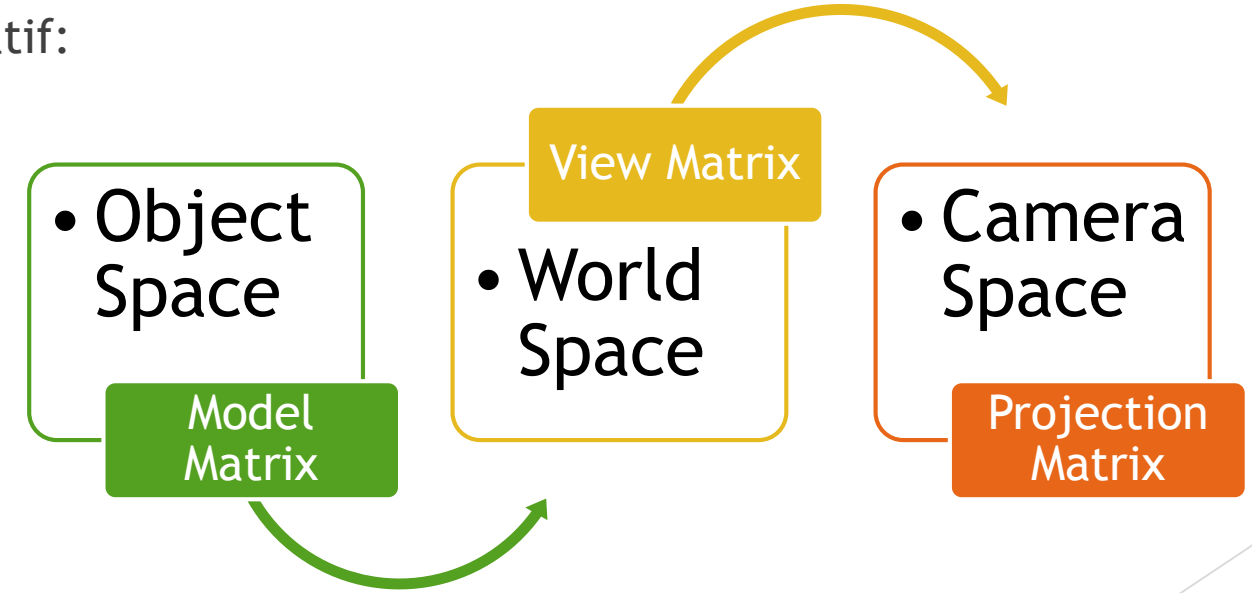
Orthographic projection

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

Perspective projection

Résumé

- 1 objet = 1 model matrix
- Chaque vertex est transformé par la model matrix correspondante
- On utilise la View pour passer en Camera Space
- On utilise la Projection matrix pour projet sur les pixel
- Récapitulatif:



Shader: premier pas

- Rappel: Shader = char*
- Défini directement dans le programme C++ ou dans un fichier séparé
- A activer AVANT un draw call, a désactiver après
- Ecrit en GLSL -> proche du C

Shader: passer des données

- GPU -> pas de partage direct entre le C++ et le shader
- Variables uniformes:
  - Permet de transmettre des données au shader
  - A créer en C++: glUniformXXX
  - A récupérer dans le shader: uniform float/int/...
- Pour passer des tableaux, images… -> Textures !
  - Dans une prochaine séance ☺

A vous !

- Téléchargez la nouvelle version…