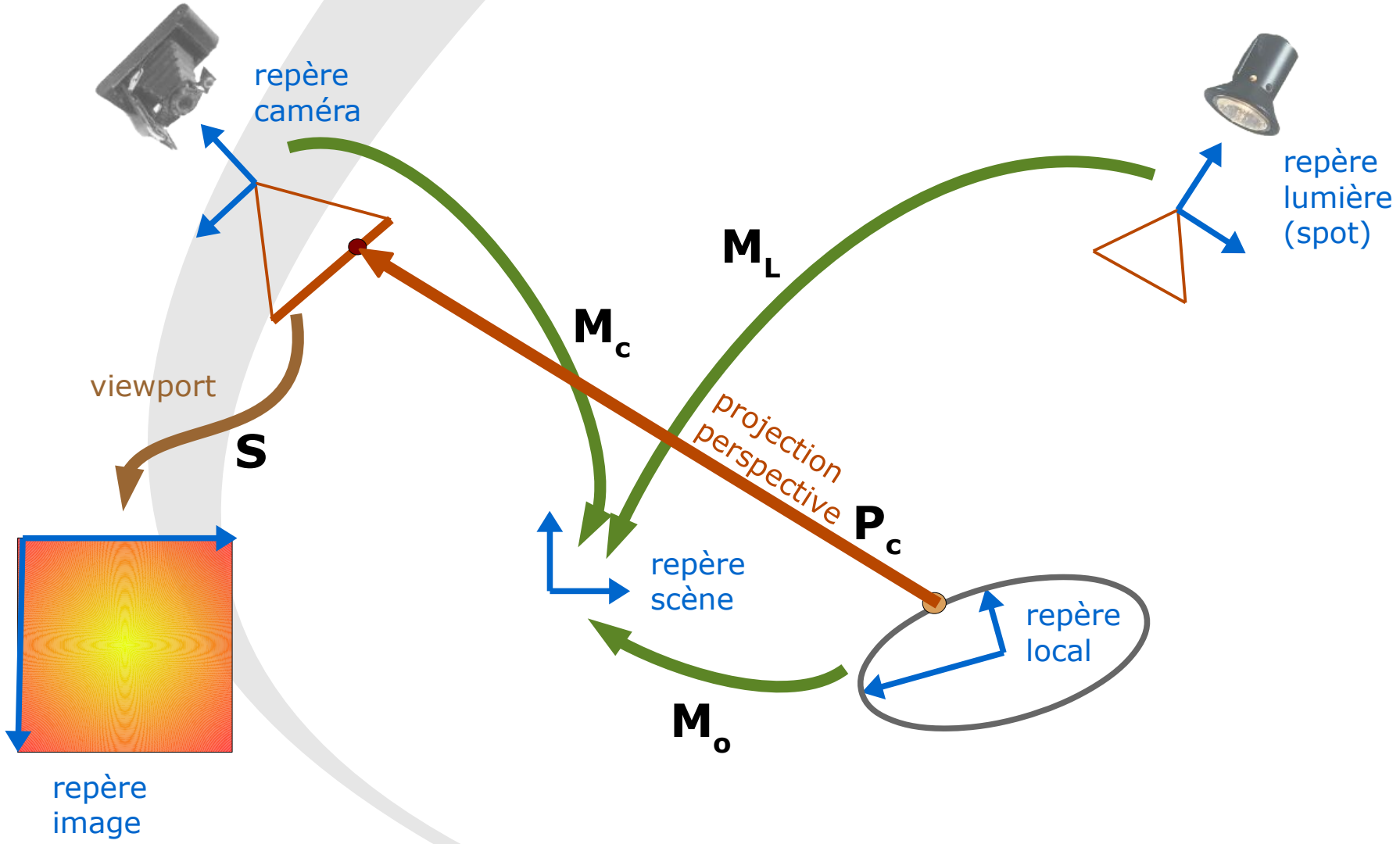
Transformations

gael.guennebaud@inria.fr

http://www.labri.fr/perso/pbenard/teaching/mondes3d

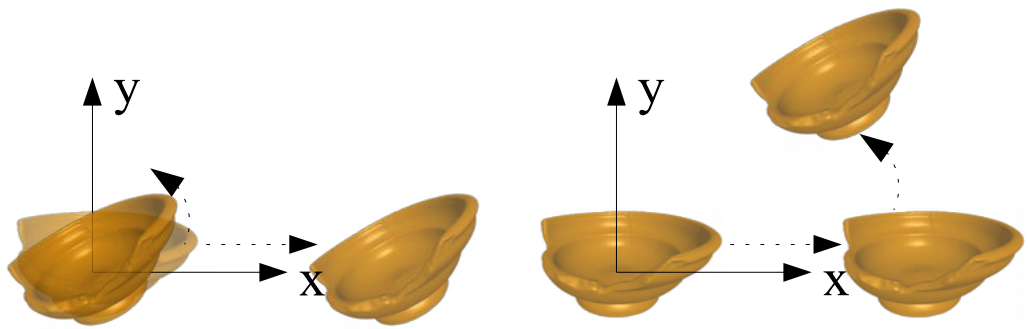
Les transformations mises en jeu



Rappels sur les Transformations

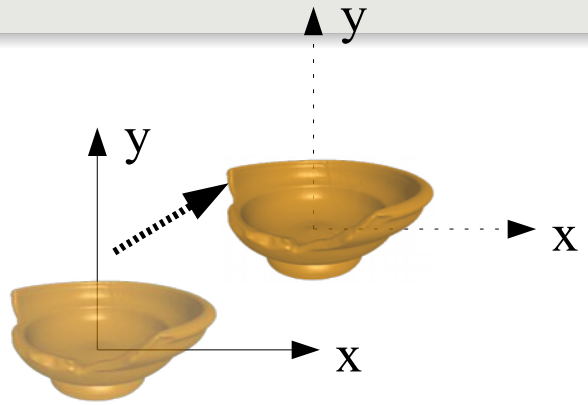
- **Utilisation des coordonnées homogènes**
  - point euclidien :
    - $v=(x,y,z,w) \sim v'=(x',y',z') = (x/w, y/w, z/w)$ *si* $w \neq 0$
  - vecteur/direction : $w = 0$ (point à l'infini !)
  - transformations affines représentées par produit matriciel
    - $v' = M * v \qquad v = M\text{-}1 * v'$
  - enchaînement de transformations = multiplication matricielle
    - $v' = T * R * v$
  - ATTENTION : non commutatif !

$$T*R*v \neq R*T*v$$

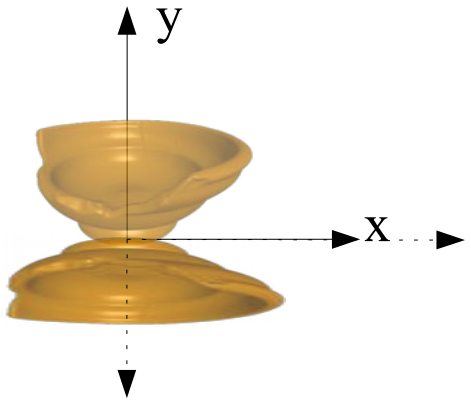


Transformations élémentaires

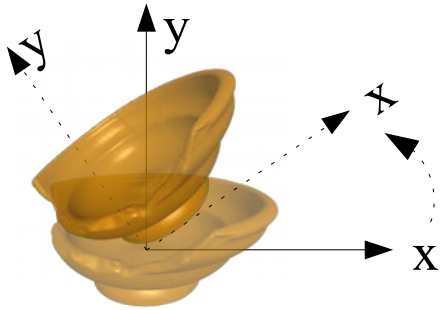
- **Translation :**
  - Eigen :
    - **v'** = **v** + *t;*
    - **v'** = *Translation3f(***t***)* * **v***;*
    - **v'** = *Translation3f(tx,ty,tz)* * **v***;*



- **Mise à l'echelle :**
  - Eigen :
    - **v'** = *s* * **v** *;*
    - **v'** = *Scaling3f(sx,sy,sz)* * **v**
    - **v'** = *Vector3f(sx,sy,sz).asDiagonal()* * **v**



- **Rotation :**
  - Eigen :
    - **v'** = *AngleAxis3f(angle,axis)* * **v***;*
    - *Matrix3f R = Matrix3f(AngleAxis3f(angle,axis));*
    - *Quaternion q = AngleAxis3f(angle,axis) ;*



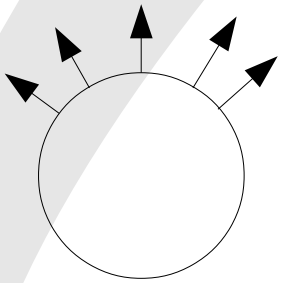
Combinaison

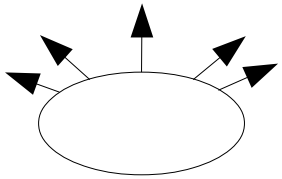
- **Combination with Eigen :**

  - *Affine3f M = AngleAxisf(angle1,axis1)*
    ** Translation3f(t1)*
    ** Scaling3f(s1)*
    ** AngleAxisf(angle2,axis2) ;*

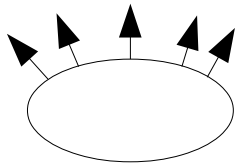
Transformation des normales

- **Problème d'une mise à l'échelle non uniforme M :**





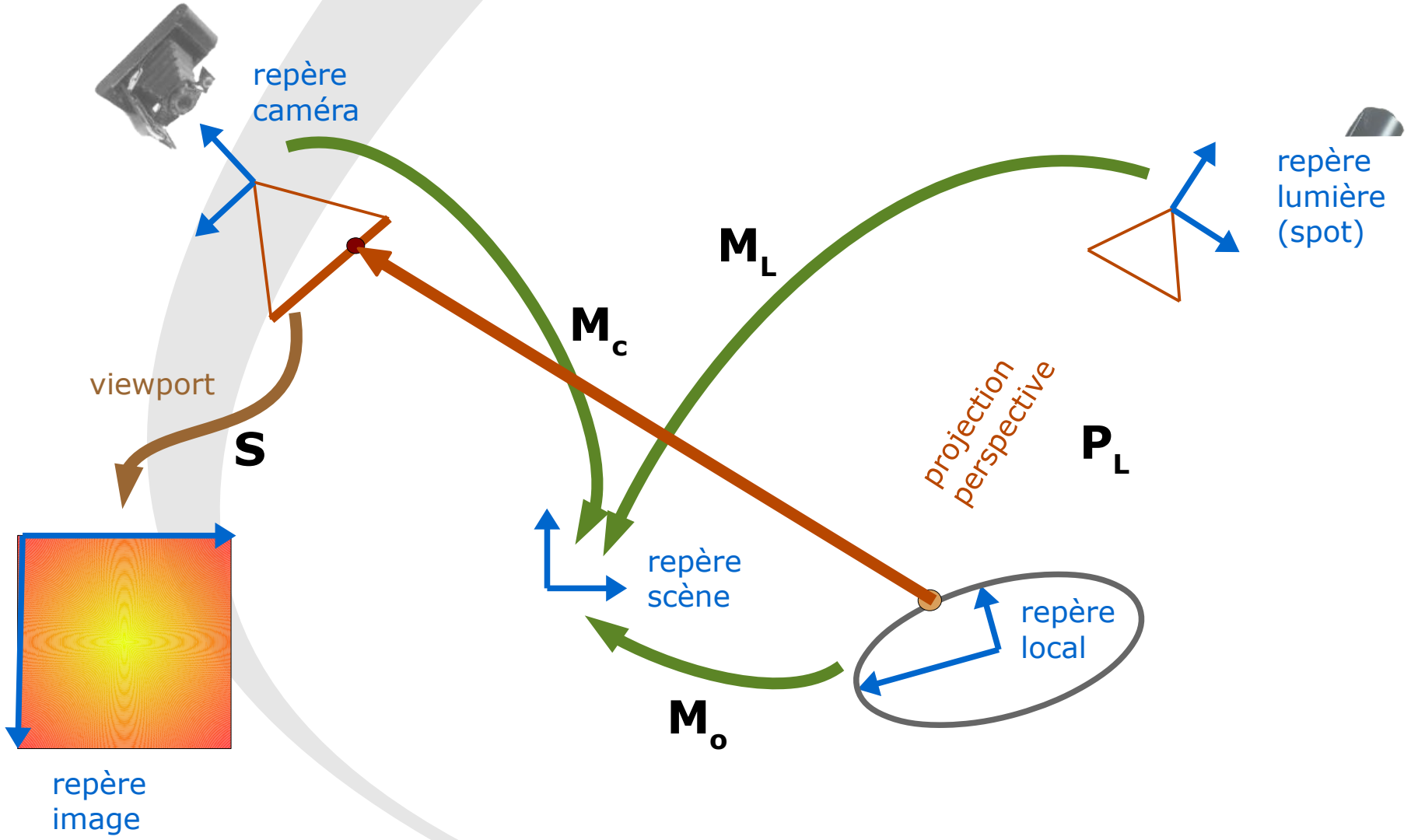
n' = M * n
=> Faux !



correct

- **Règle: utiliser la transposée de l'inverse:**
  - Soit M la matrice 3x3 de transformation courante:
    $v' = M * v + T$
  - Alors :
    $n' = (M^{-1})^t * n$

Les transformations mises en jeu

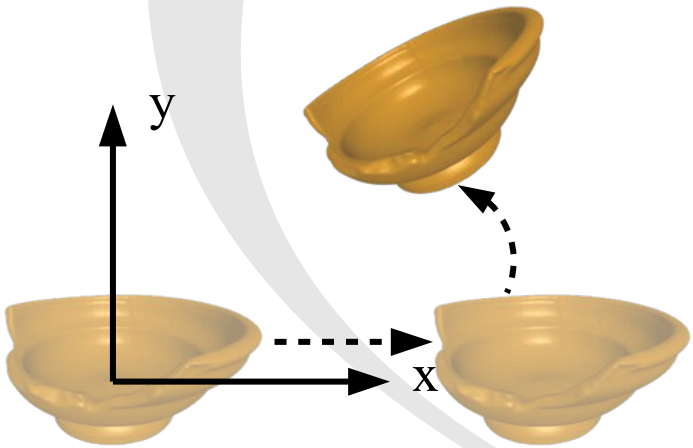


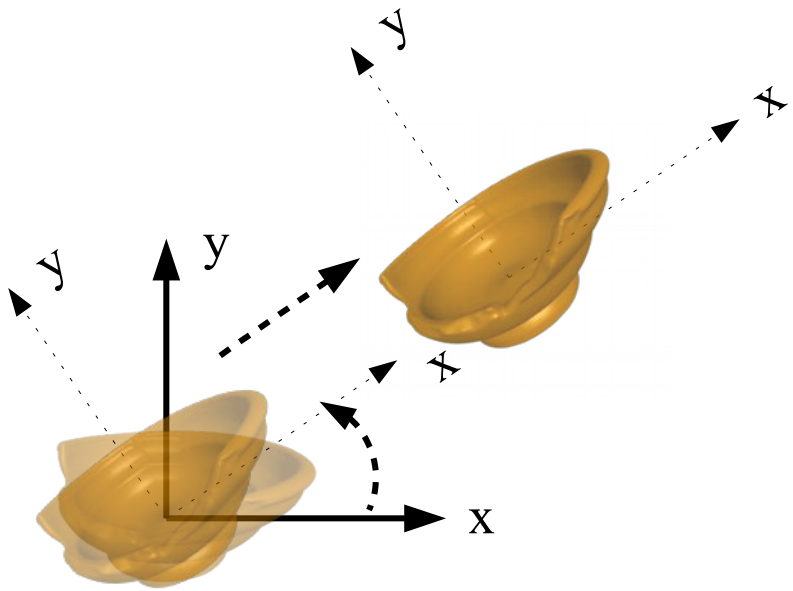
“Penser” les transformations

- **Approche “repère global”**
  - le repère reste fixe
  - définir les transformations de droite à gauche

- **Approche “repère local”**
  - chaque transformations affecte le repère courant
  - “amener” le repère dans la position/orientation voulue
  - composer de gauche à droite

R * T * v

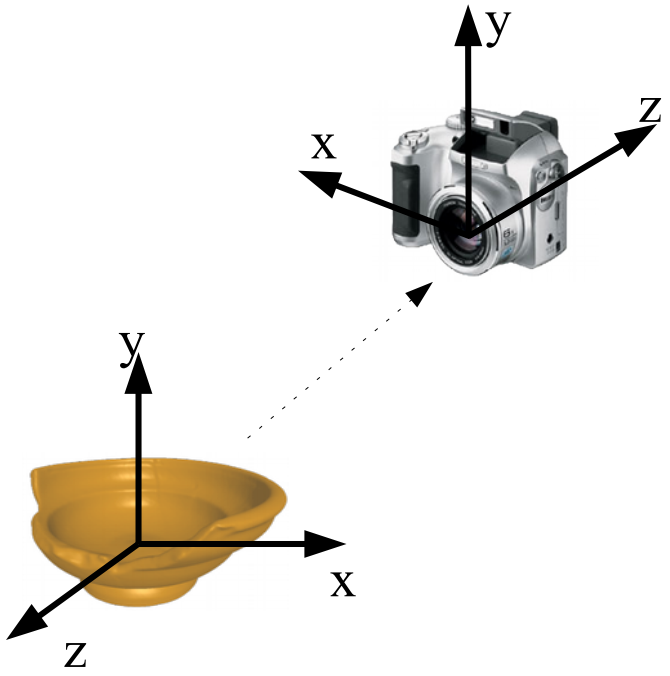




Transformation de visualisation

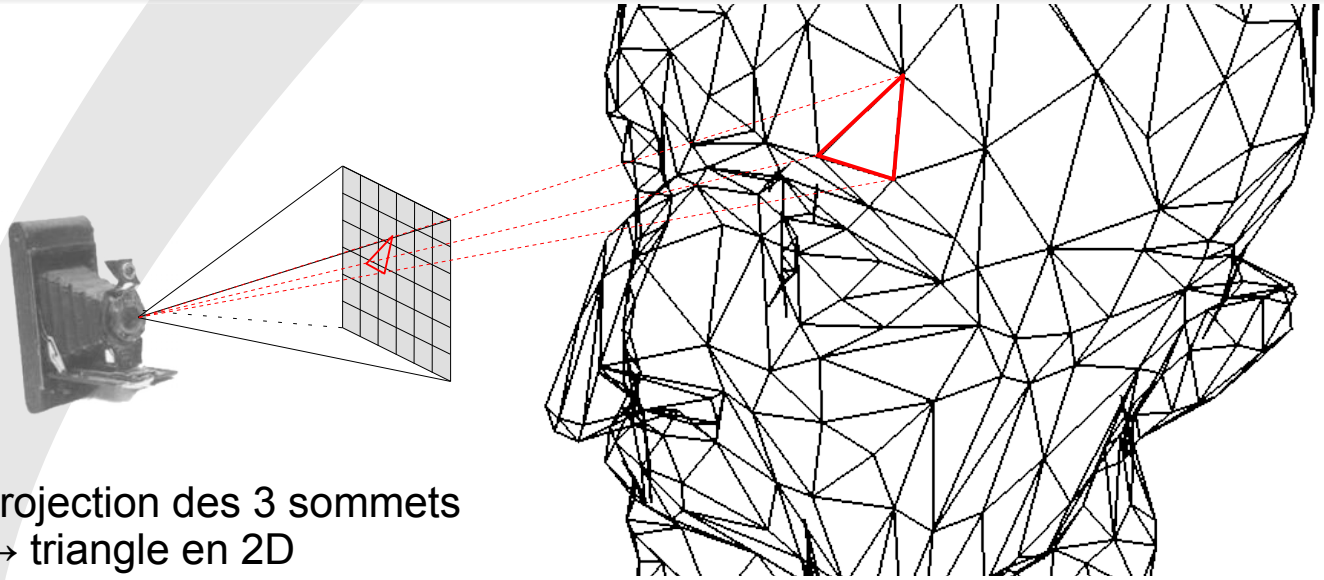
- **Permet de positionner la caméra**

  transformation de visualisation
  ~ inverse du repère local



- **Exercice : positionner et orienter la caméra :**
  - *Matrix4f lookAt(const Vector3f& p,* *// position de la caméra*
    const Vector3f& t, *// point de visé (target)*
    const Vector3f& u) *// vecteur “hauteur” (up)*

Rastérisation

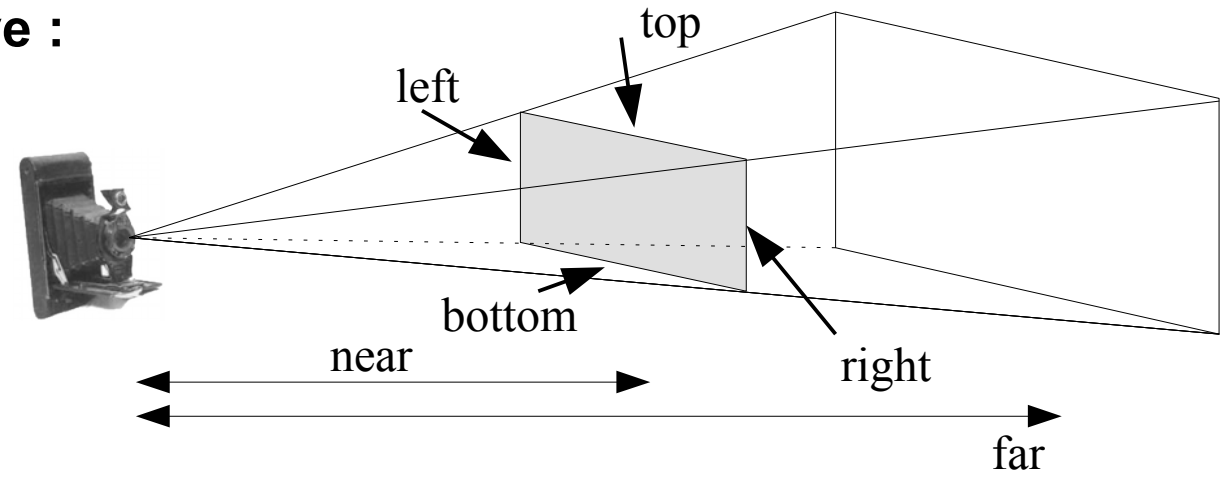


projection des 3 sommets
→ triangle en 2D

- **Rastérisation**
  - Pour chaque primitive P_i, trouver les rayons intersectant P_i
  - Rendu en deux étapes
    - projection des primitives sur l'écran (*forward projection*)
    - discrétisation (conversion des primitives 2D en pixels)

Transformation de projection

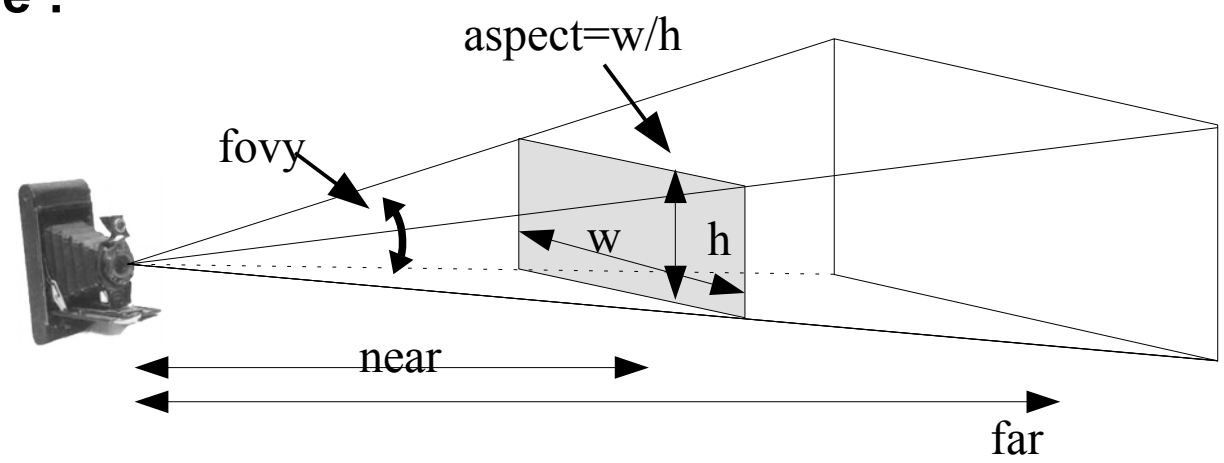
- **Projection perspective :**



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

Transformation de projection

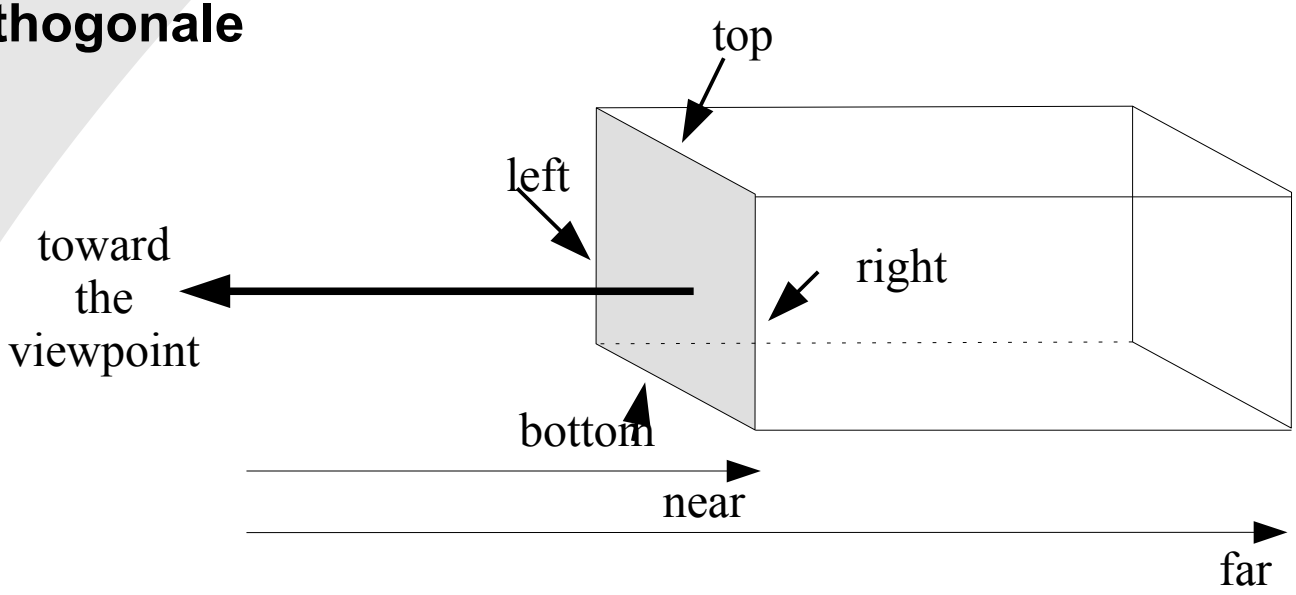
- **Projection perspective :**



- **Exercice : écrire une fonction :**
  Matrix4f makePerspective(float fov, float aspect, float n, float f)

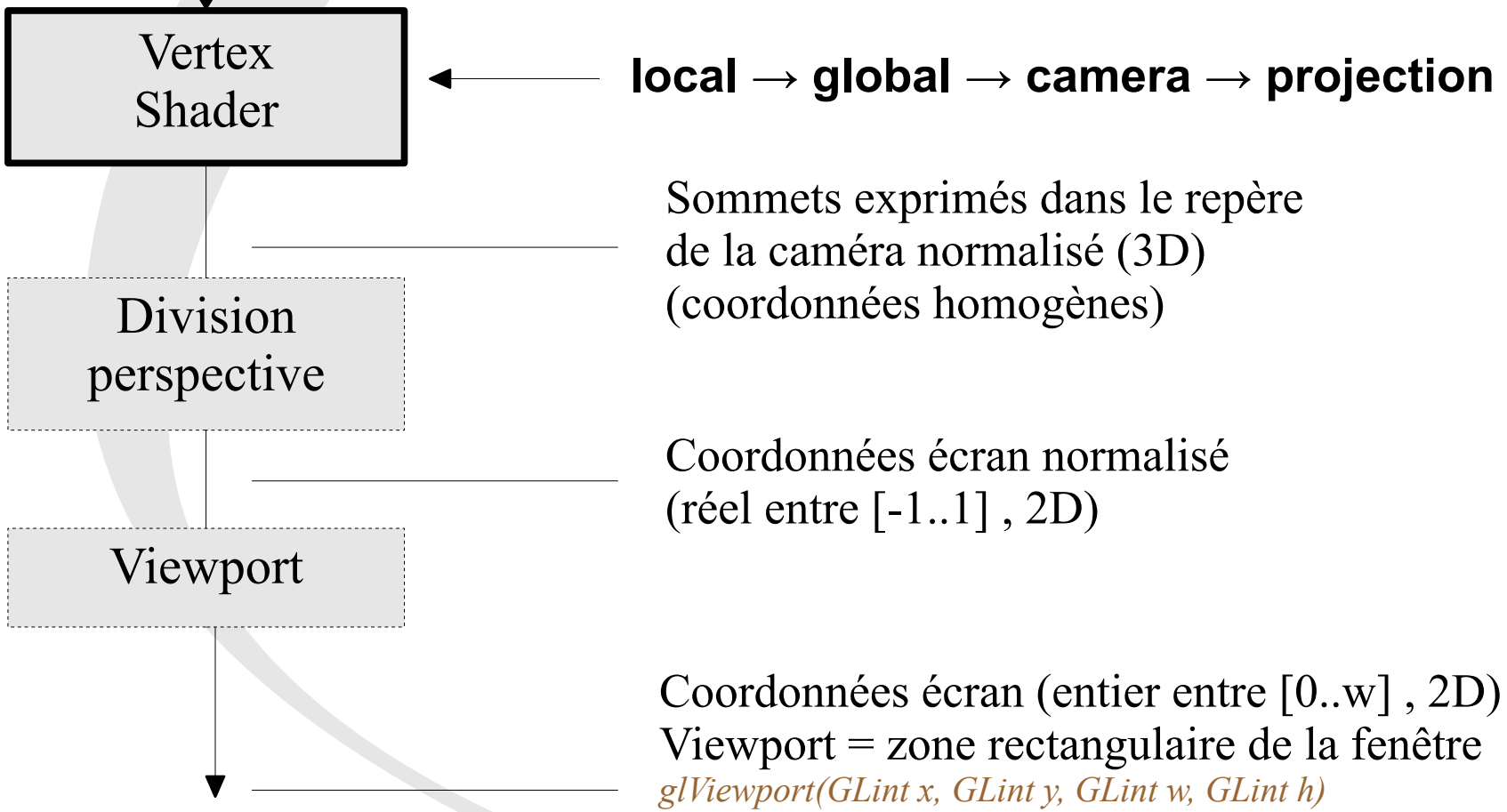
Transformation de projection

- **Projection orthogonale**



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

Vertex pipeline : résumé



API

	GLSL	**C++ (Eigen)**
vecteur 2D	vec2	Vector2f
vecteur 3D	vec3	Vector3f
vecteur 4D	vec4	Vector4f
matrice 2x2	mat2,mat2x2	Matrix2f
matrice 3x3	mat3,mat3x3	Matrix3f
matrice 4x4	mat4,mat4x4	Matrix4f
matrice 3x4	mat3x4	Matrix<float,3,4>
accès	v.x	v.x(), v[0], v(0)
	v.y	v.y(), v[1], v(1)
	v.z	v.z(), v[2], v(2)
	v.w	v.w(), v[3], v(3)
(i=ligne,j=colone)	M[j][i]	M(i,j)
colone d'une matrice	M[j]	M.col(j)
ligne d'une matrice		M.row(i)
sous-matrice		M.block(i,j,rows,cols)

API

	GLSL	**C++ (Eigen)**
produits	`M * v` `M1 * M2`	`M * v` `M1 * M2`
transpose	`transpose(M)`	`M.transpose()`
combinaison linéaire	`a*v1 + b*v2`	`a*v1 + b*v2`
produit scalaire	`dot(v1,v2)`	`v1.dot(v2)` `v1.transpose()*v2`
longueur/norme	`length(v)`	`v.norm()`
produit vectoriel	`v3 = cross(v1,v2)`	`v3 = v1.cross(v2)`

Eigen

Initialisation

```
Matrix3 M ;
M <<    1, 2, 3,
        4, 5, 6,
        7, 8, 9;
```

Translation par un vecteur v	`Translation3f(v)`
Rotation d'un angle theta autour d'un axe v	`AngleAxisf(theta,v)` `M = Matrix3f(AngleAxisf(theta,v))`
Mise à l'échelle	`Scaling3f(v)` `Scaling3f(s0, s1, s2)`
Transformation affine	`Affine3f A ;`
partie linéaire (3x3)	`A.linear()`
translation	`A.translation()`
matrice 4x4	`A.matrix()`

Exemple :

```
Affine3f A ;
A = Translation3f(p) * AngleAxisf(theta,v) * Translation3f(-p) ;
v2 = A * v1 ;
```