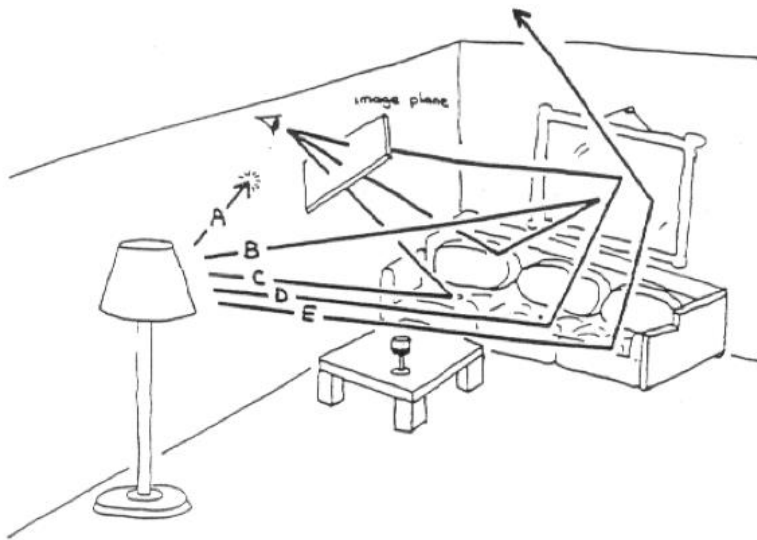


Computer graphics

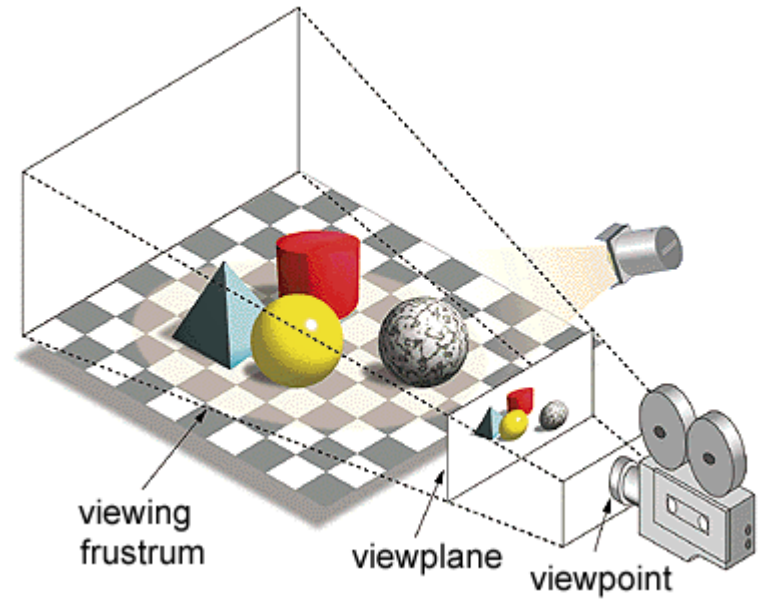
计算机图像学

Nicolas Stoiber / Renaud Séguier
Dynamixyz / CentraleSupélec
(www.dynamixyz.com)

Image Synthesis



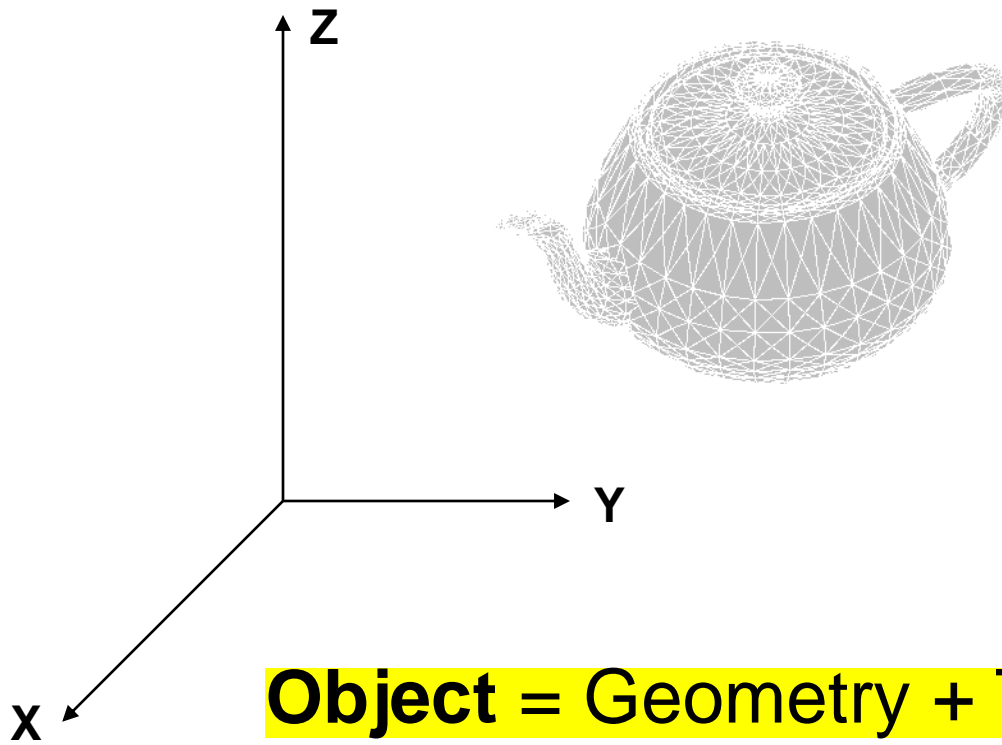
Real life



Computer graphics

Image Synthesis

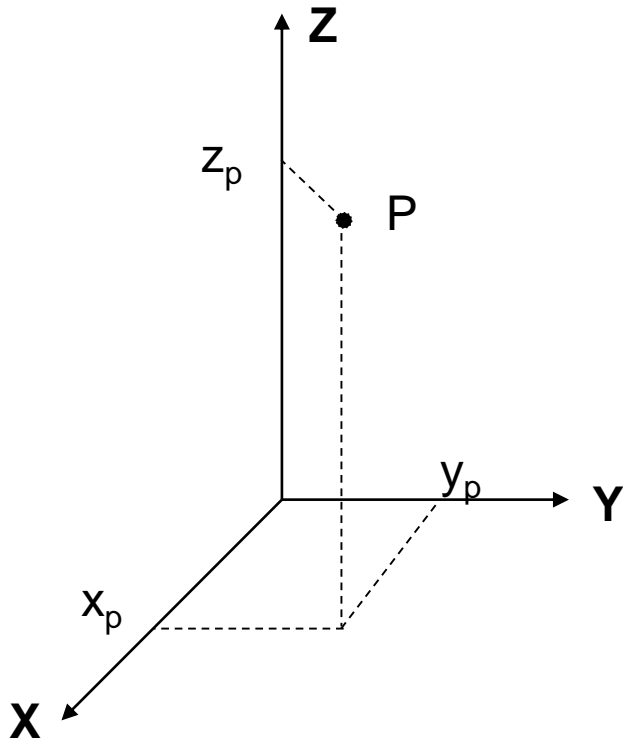
- Building a scene (modeling) = creating and placing objects



Object = Geometry + Transformation

Geometry

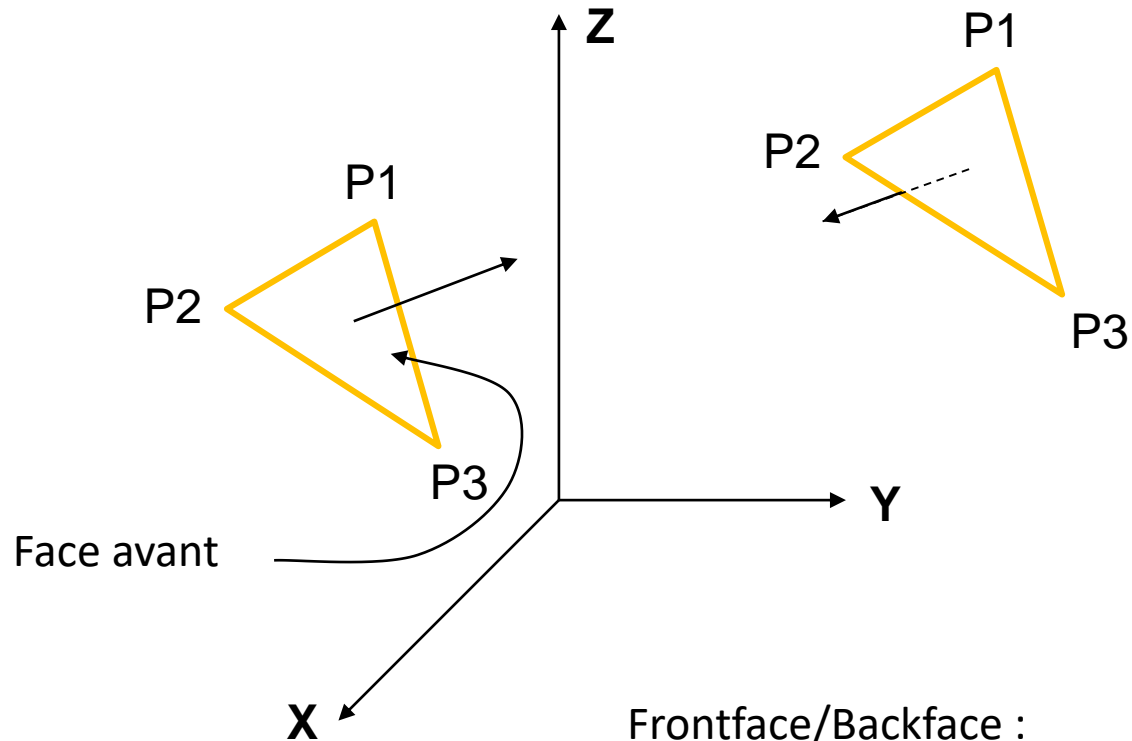
- Primitives (point)



Vertex : $P \rightarrow (x_p, y_p, z_p)$

Geometry

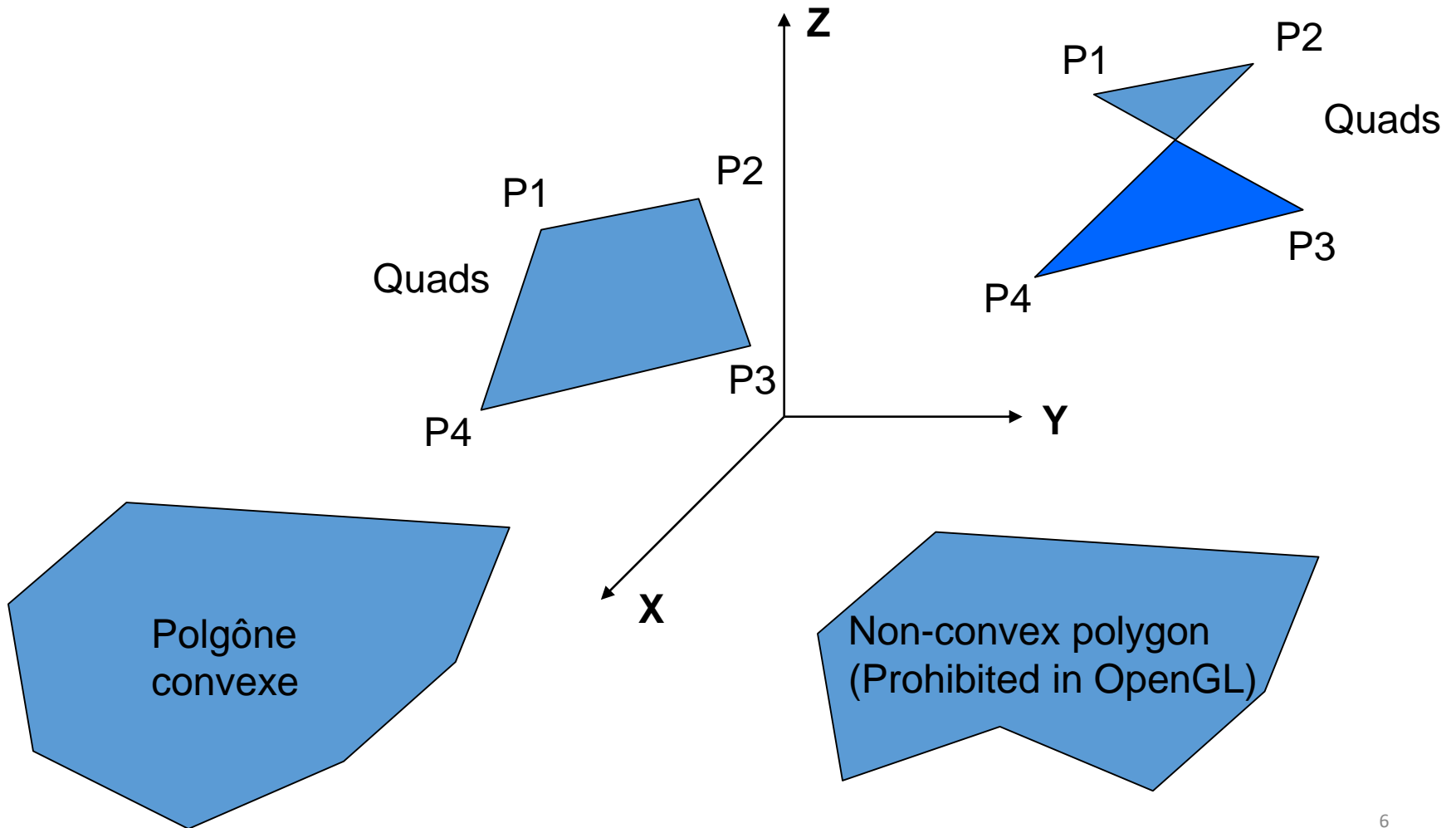
- Primitives (triangles)



Frontface/Backface :
The points order defines the orientation
of a surface.

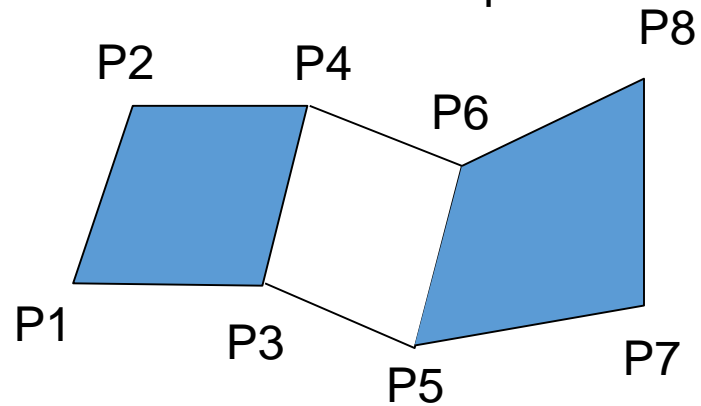
Geometry

- Primitives (polygones) 多边形

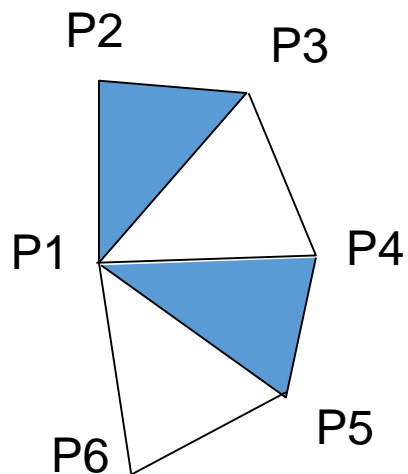


Geometry

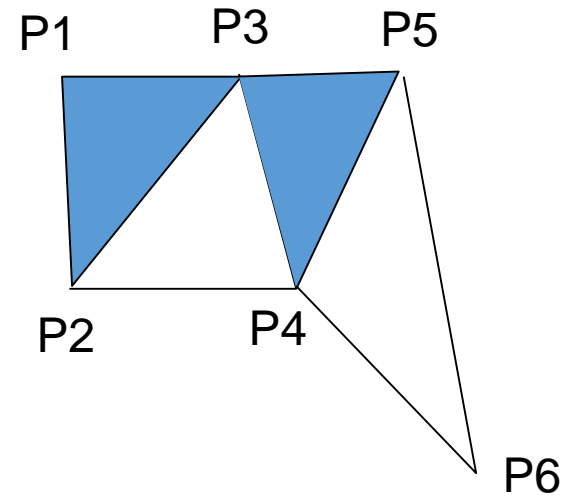
Quads Strip



Triangle Fan



Triangle Strip



Transformations

几何+变换

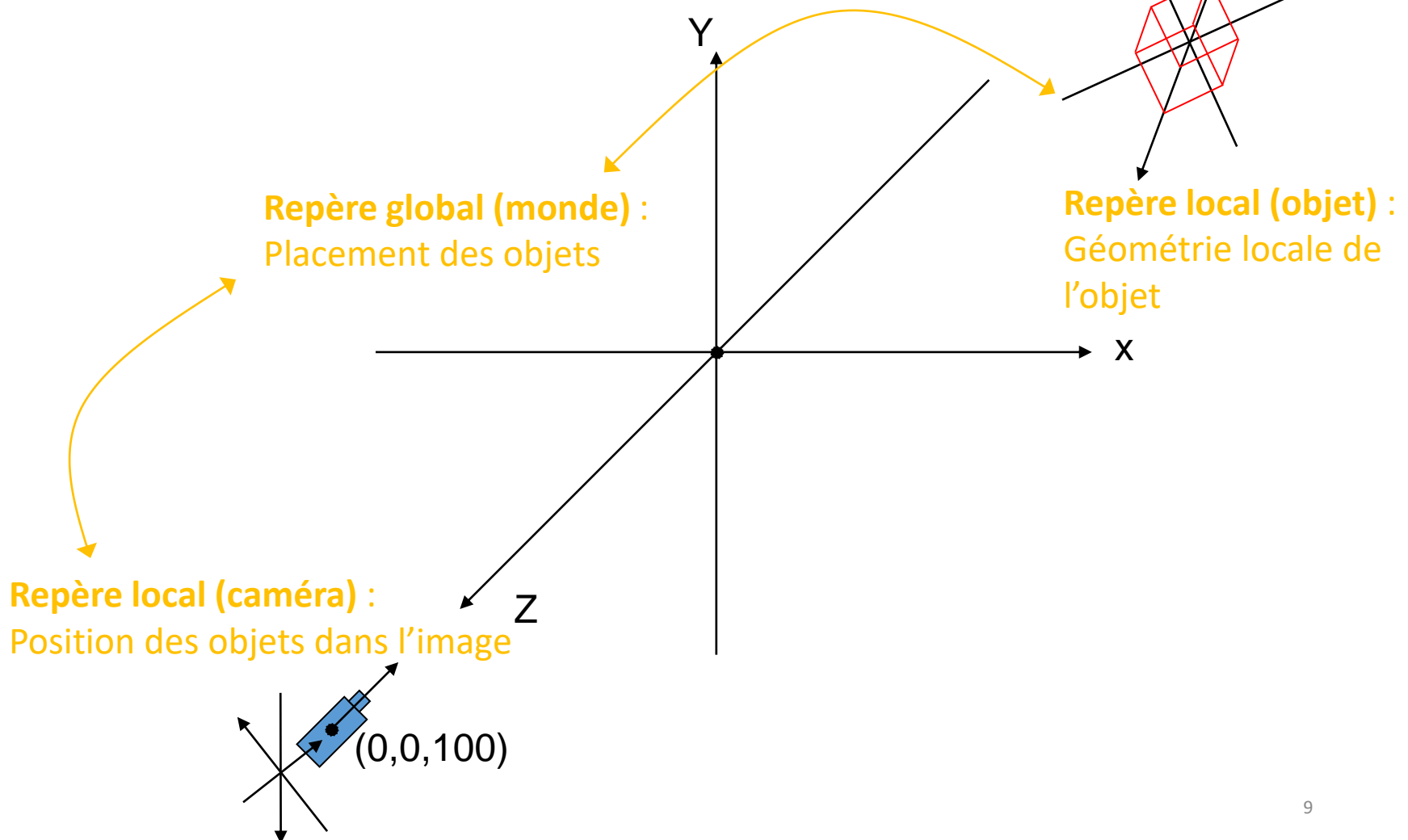
Objet = Géométrie + Transformation

- Rôle des transformations :
 - Changements de repères
 - Placer les objets dans la scène (modélisation)
 - Modifier la forme des objets (redimensionnement, ...)
 - Simuler la transformation projective de la caméra virtuelle (transformation projective)
 - Animation

Transformations

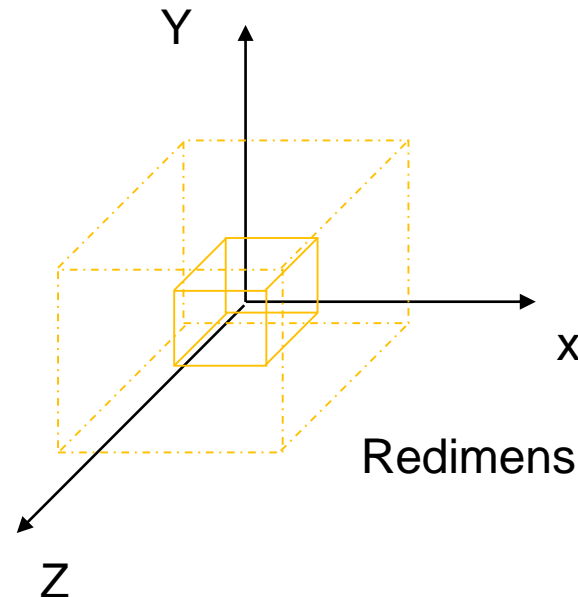
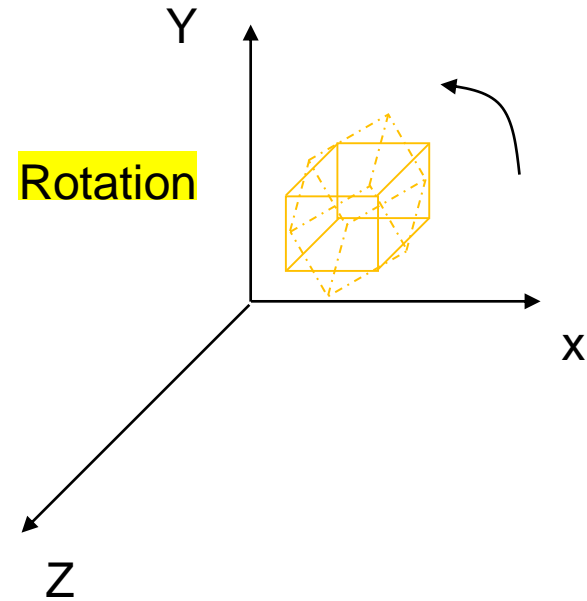
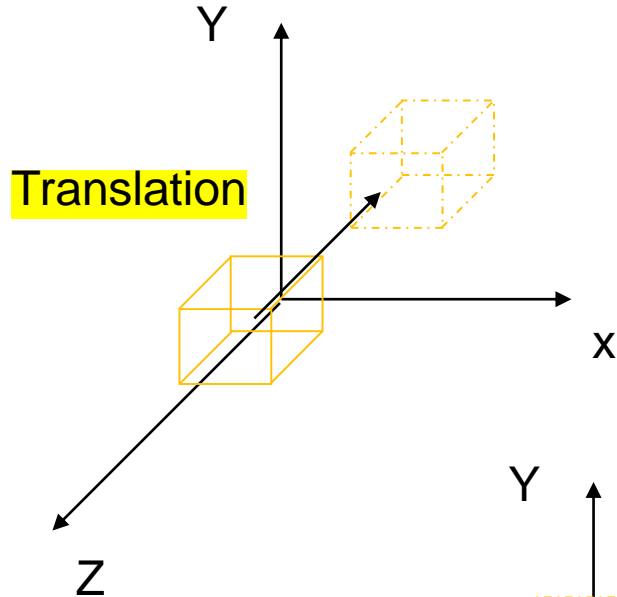
参考系

- Transformations : Changements de repères



Transformations

- Transformations simples (illustration)



Redimensionnement (*scaling*) en x, y et z

Transformations

- Transformations simples (modélisation)

Translation

$$\mathbf{p}' = \mathbf{p} + \mathbf{t}$$

$$\begin{matrix} x' \\ y' \\ z' \end{matrix} = \begin{matrix} x \\ y \\ z \end{matrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

Rotation

$$\mathbf{p}' = \mathbf{R}\mathbf{p}$$

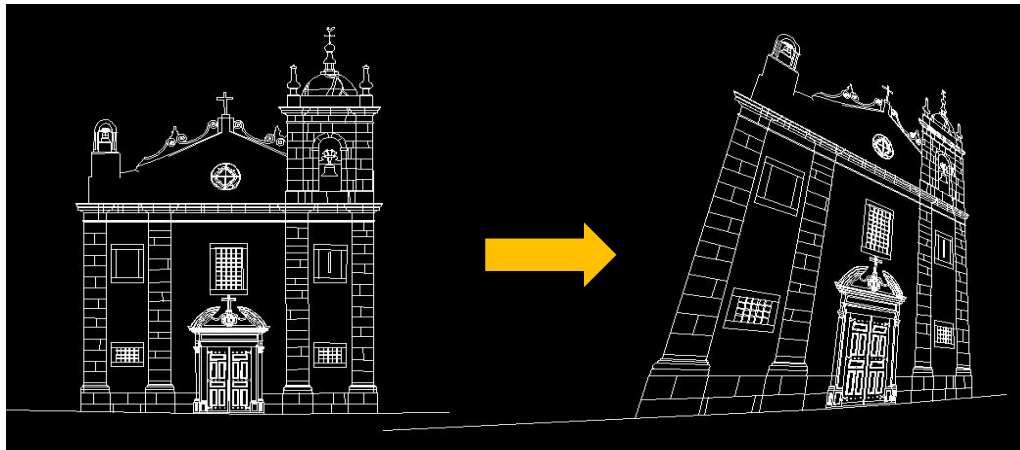
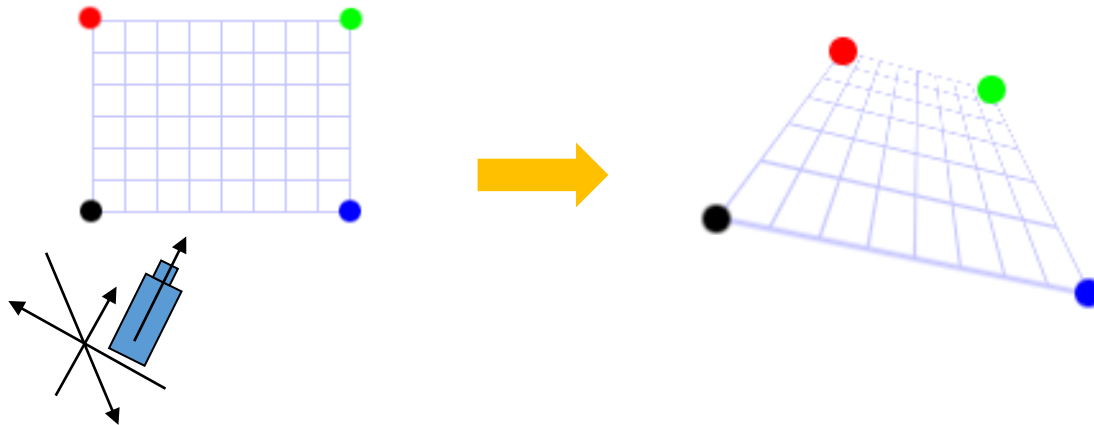
$$\begin{matrix} x' \\ y' \\ z' \end{matrix} = \begin{bmatrix} a & b & c \\ e & f & g \\ i & j & k \end{bmatrix} \begin{matrix} x \\ y \\ z \end{matrix}$$

R est une matrice orthogonale
(groupe des matrices de rotation)

$$\mathbf{R}^T \mathbf{R} = \mathbf{R} \mathbf{R}^T = \mathbf{I}$$

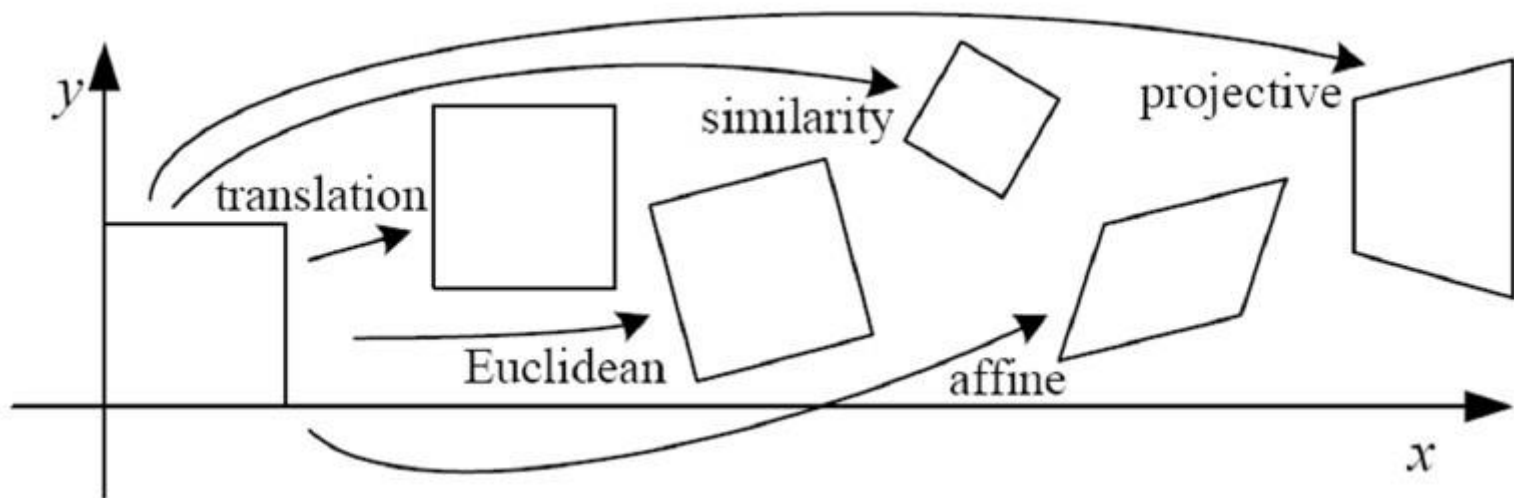
Transformations

- Transformations affines et projectives
→ Calcul de la projection en perspective



Transformations

- Transformations générales



Représentation unifiée grâce aux **coordonnées homogènes**

Transformations

- Coordonnées homogènes

En coordonnées euclidiennes : translation \neq multiplication matricielle

→ représentation des transformations non homogène 非同构变换

$$x' = a.x + b.y + c.z + d$$

$$y' = e.x + f.y + g.z + h$$

$$z' = i.x + j.y + k.z + l$$

$$\begin{matrix} x' \\ y' \\ z' \end{matrix} = \begin{bmatrix} a & b & c \\ e & f & g \\ i & j & k \end{bmatrix} \begin{matrix} x \\ y \\ z \end{matrix} + \begin{bmatrix} d \\ h \\ l \end{bmatrix}$$

$$p' = M p + t$$

Transformations

- Coordonnées homogènes

Principe : Ajout d'une coordonnée !

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$
$$p' = M p$$

Pour les transformations linéaires et affines $w = w' = p = 1$, $m=n=o=d=h=l=0$

线性和仿射变换

Transformations

Translation :

平移

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling :

$$\mathbf{S} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotations :

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta_x & -\sin \theta_x & 0 \\ 0 & \sin \theta_x & \cos \theta_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

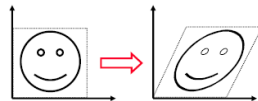
$$\mathbf{R}_y = \begin{bmatrix} \cos \theta_y & 0 & \sin \theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta_y & 0 & \cos \theta_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R}_z = \begin{bmatrix} \cos \theta_z & -\sin \theta_z & 0 & 0 \\ \sin \theta_z & \cos \theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Affine:

$$\mathbf{M} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\left(\begin{array}{l} ex: \mathbf{S}_h = \begin{bmatrix} 1 & z & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \right)$$



Projective:

$$\mathbf{M} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}$$

$$\left(\begin{array}{l} ex: \mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{f} & 0 \end{bmatrix} \end{array} \right)$$

Transformations

- Coordonnées homogènes : composition
 - Simple produit, opération homogène pour tout type de transformation.

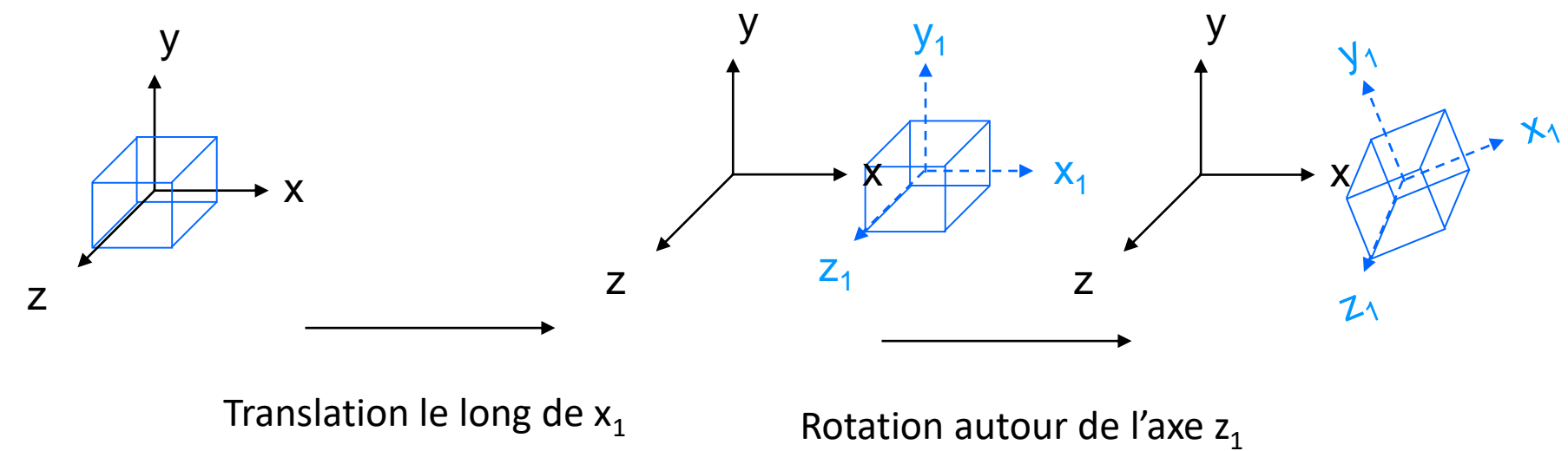
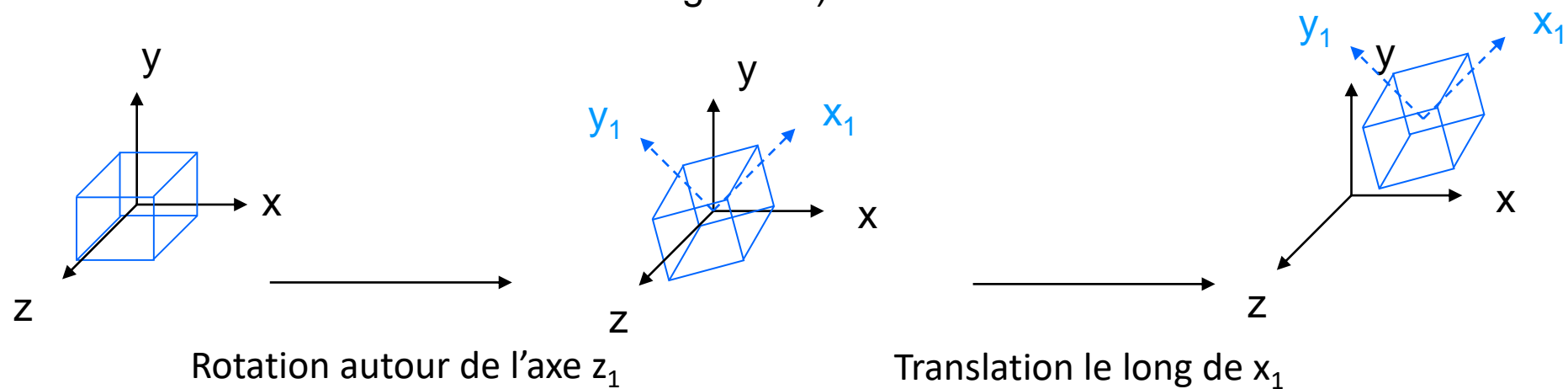
$$\mathbf{v}' = S\mathbf{v}$$

$$\mathbf{v}'' = R\mathbf{v}' = RS\mathbf{v}$$

$$\mathbf{v}''' = T\mathbf{v}'' = TR\mathbf{v}' = TRS\mathbf{v}$$

$$\mathbf{v}''' = M\mathbf{v} \qquad M = TRS$$

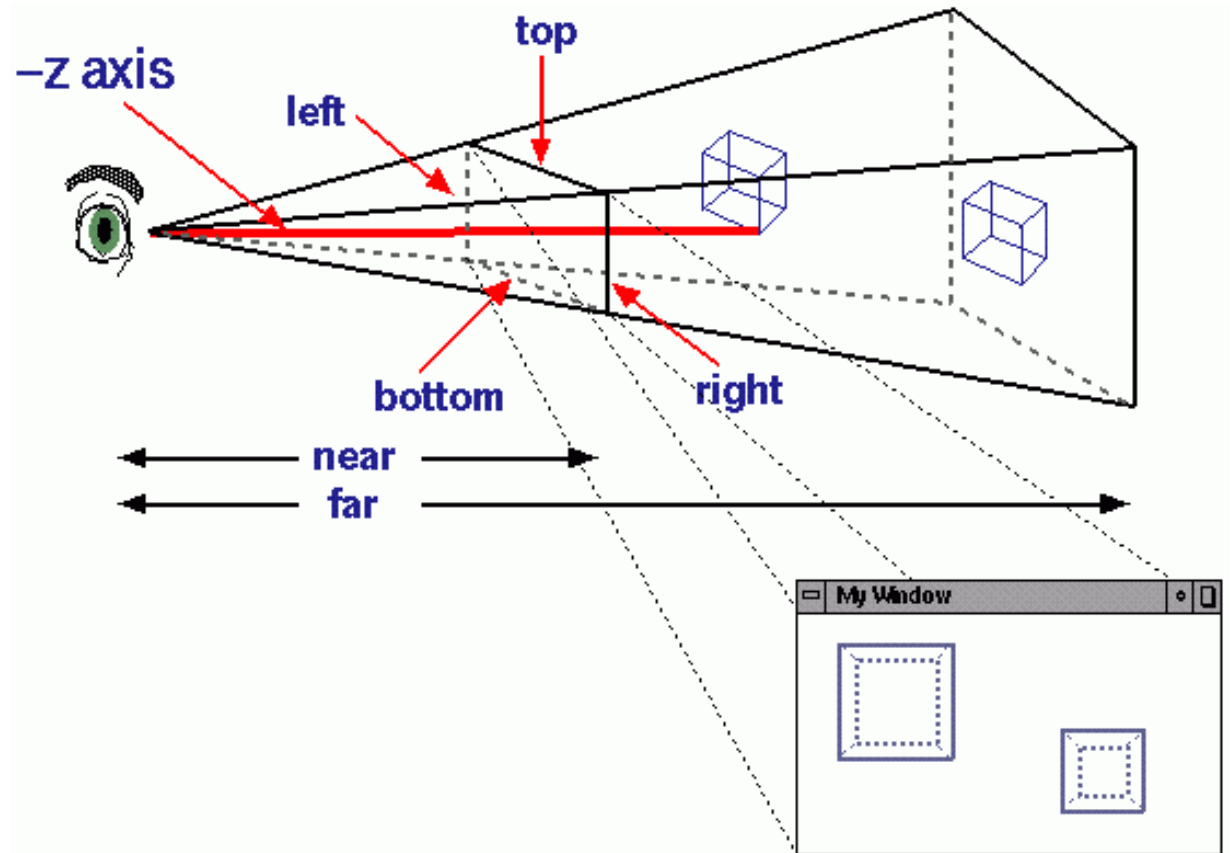
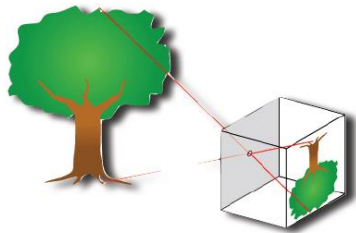
Attention : les transformations se réfèrent aux axes locaux des objets (transformations non commutatives en général).



Transformations

- Camera virtuelle

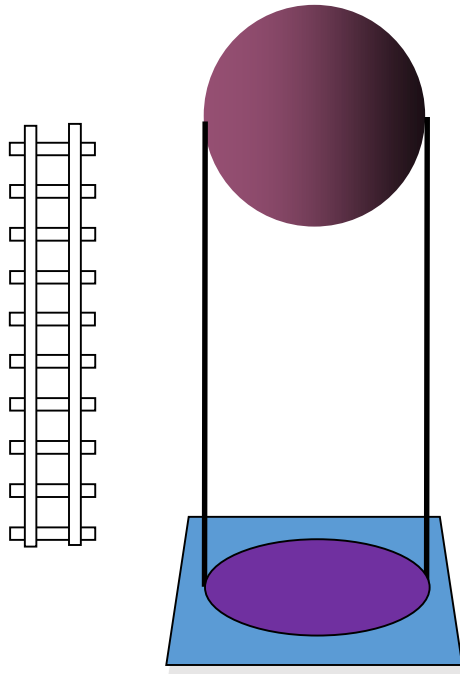
Caméra « pinhole »



Transformations

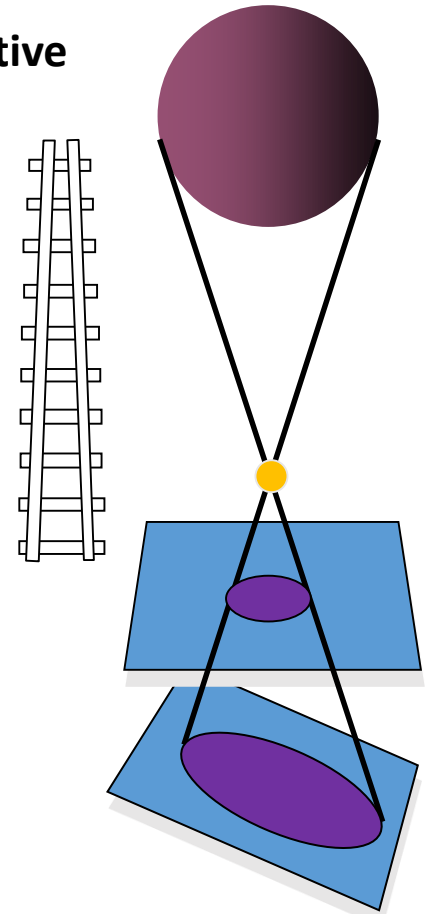
- Projection

Orthographique



Applications : CAO, Architecture
Pour travailler les objets : *modeling*

Perspective

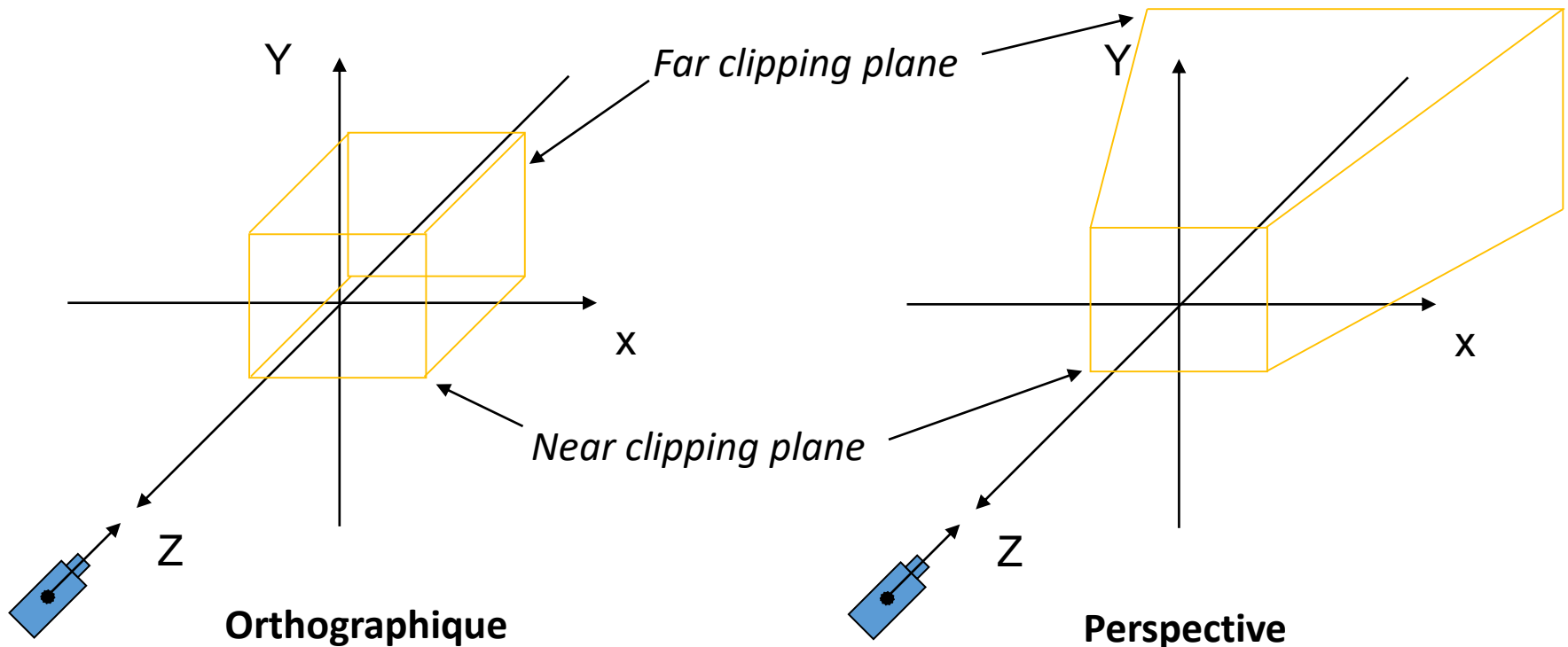


Applications : Vue naturelle. Réalité virtuelle.

Transformations

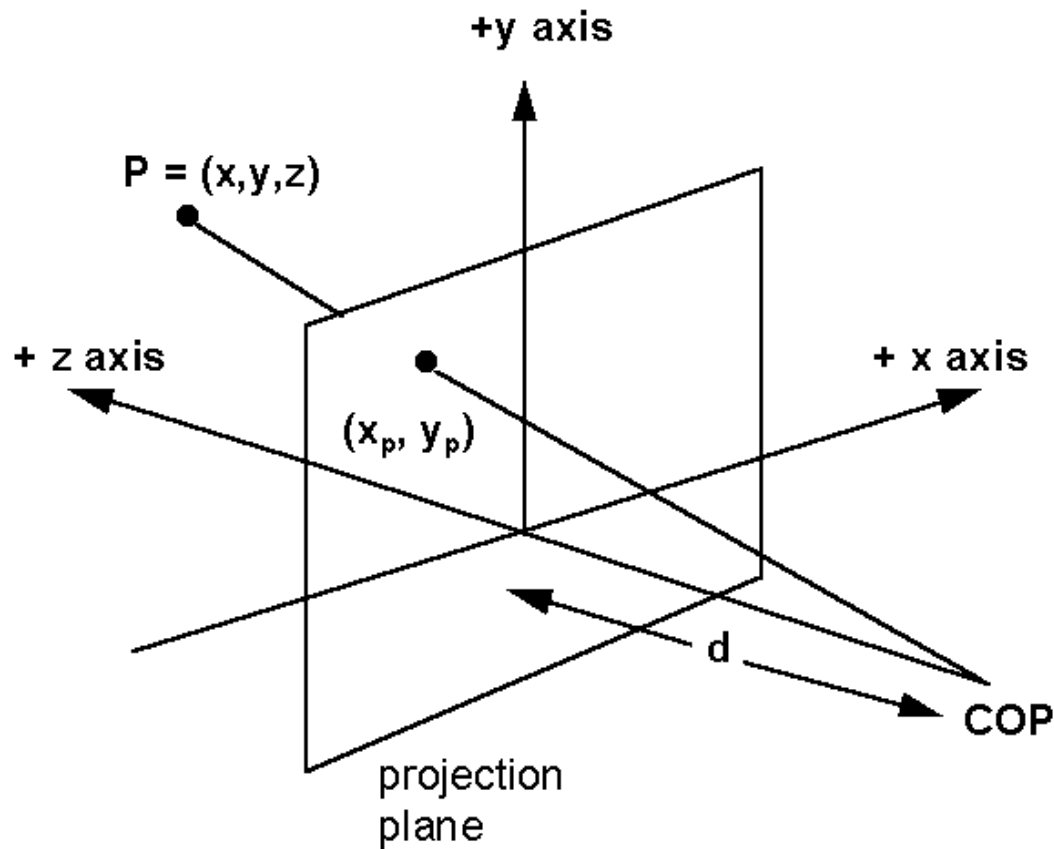
- Projection : calcul des coordonnées des objets **dans l'image**

« *Clipping volume* » : volume visualisé par la caméra

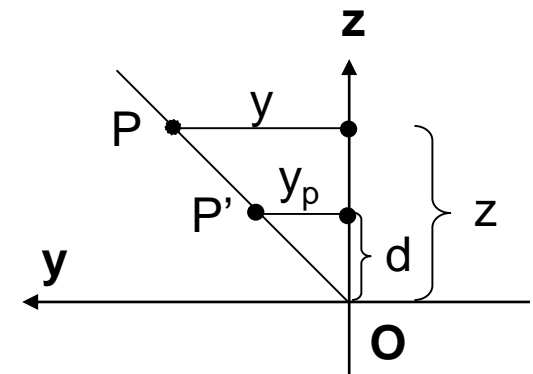


Transformations

- Projection



O : centre optique
 d : distance focale



Thalès :
$$\frac{y}{z} = \frac{y_p}{d}$$

$$y_p = \frac{y \cdot d}{z}$$

Transformations

- Projection

$$x_p = \frac{x \cdot d}{z}$$

$$y_p = \frac{y \cdot d}{z}$$

$$z_p = d$$



Matrice de projection
(coordonnées homogènes)

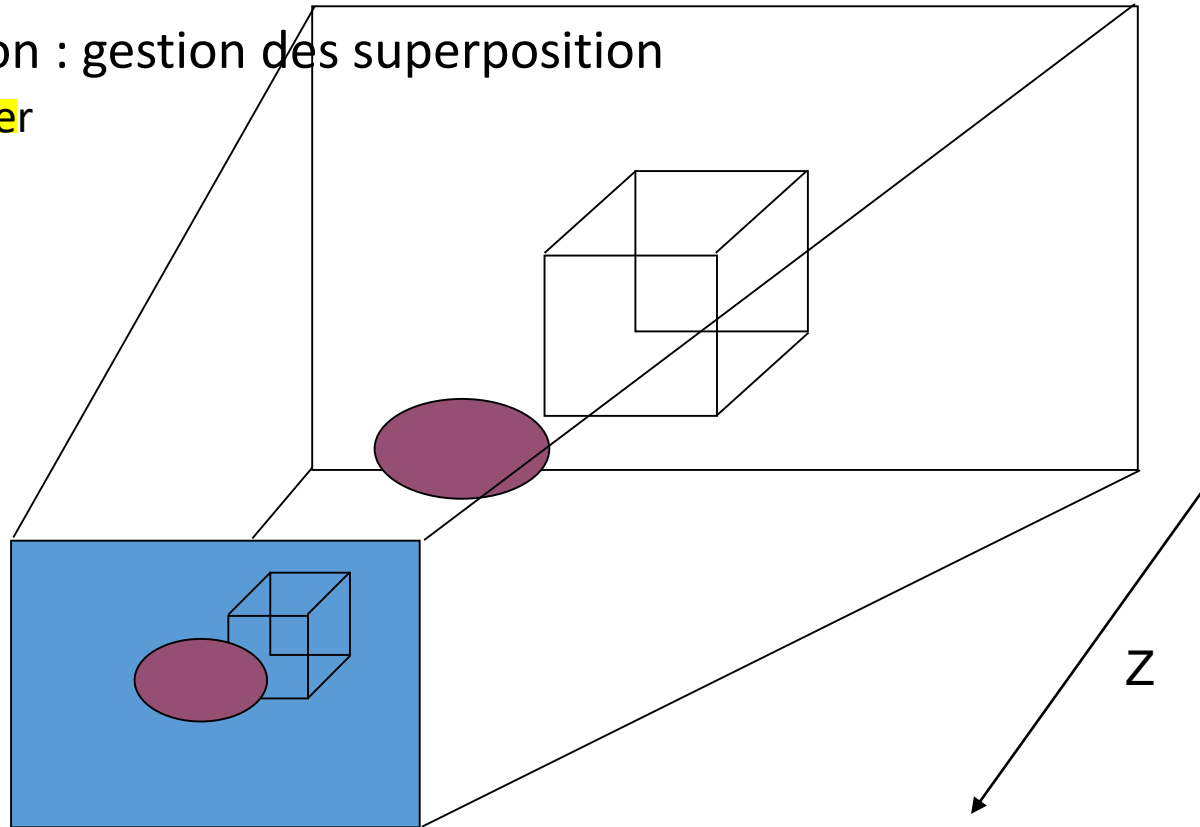
$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \end{bmatrix}$$

Point projeté :

$$\mathbf{P}' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ \frac{z}{d} \end{bmatrix} = \begin{bmatrix} \frac{x \cdot d}{z} \\ \frac{y \cdot d}{z} \\ z \\ 1 \end{bmatrix}$$

Transformations

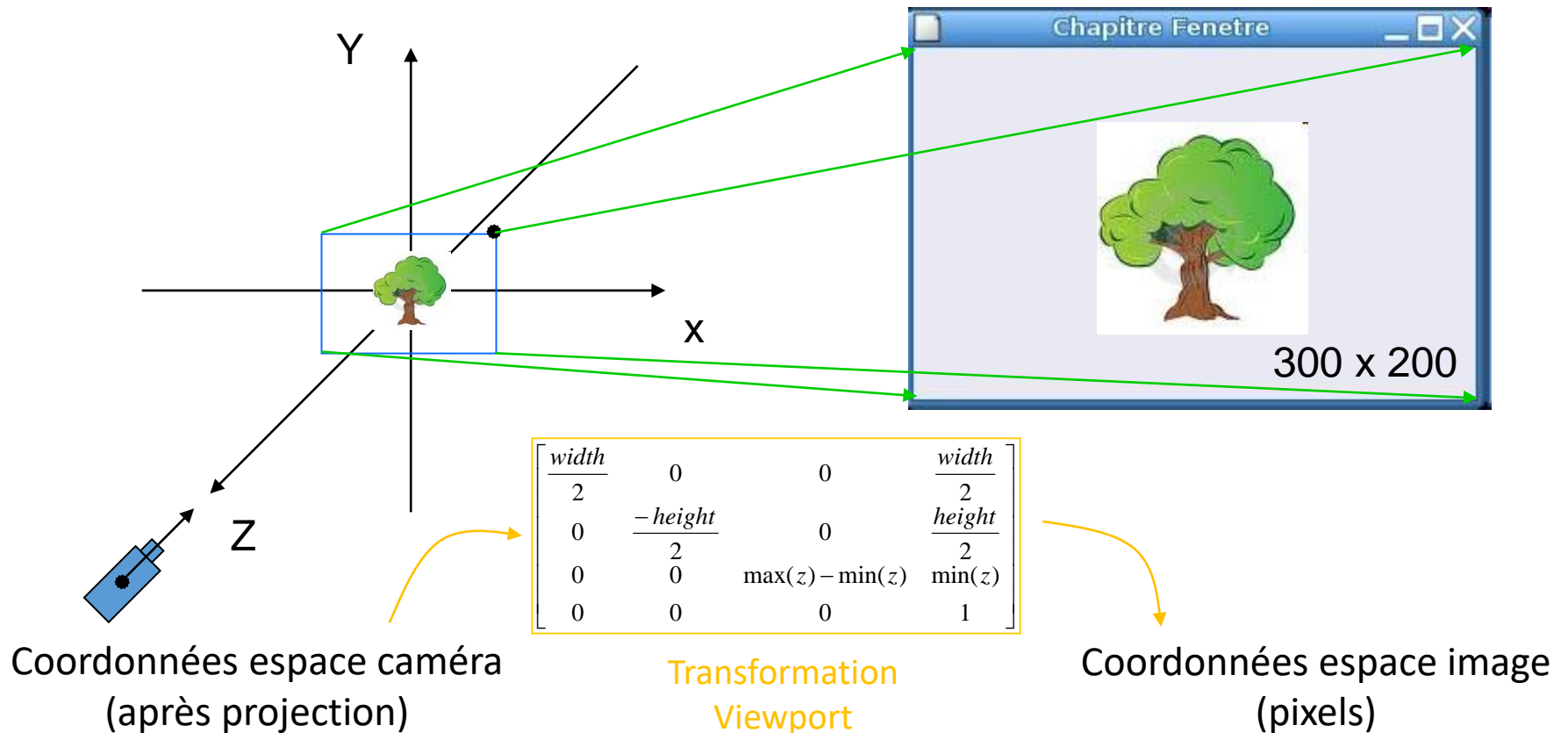
- Projection : gestion des superposition
 - Z Buffer



Z Buffer = Distance de chaque pixel par rapport au premier plan
→ comparaison avant affichage de la distance de chaque pixel

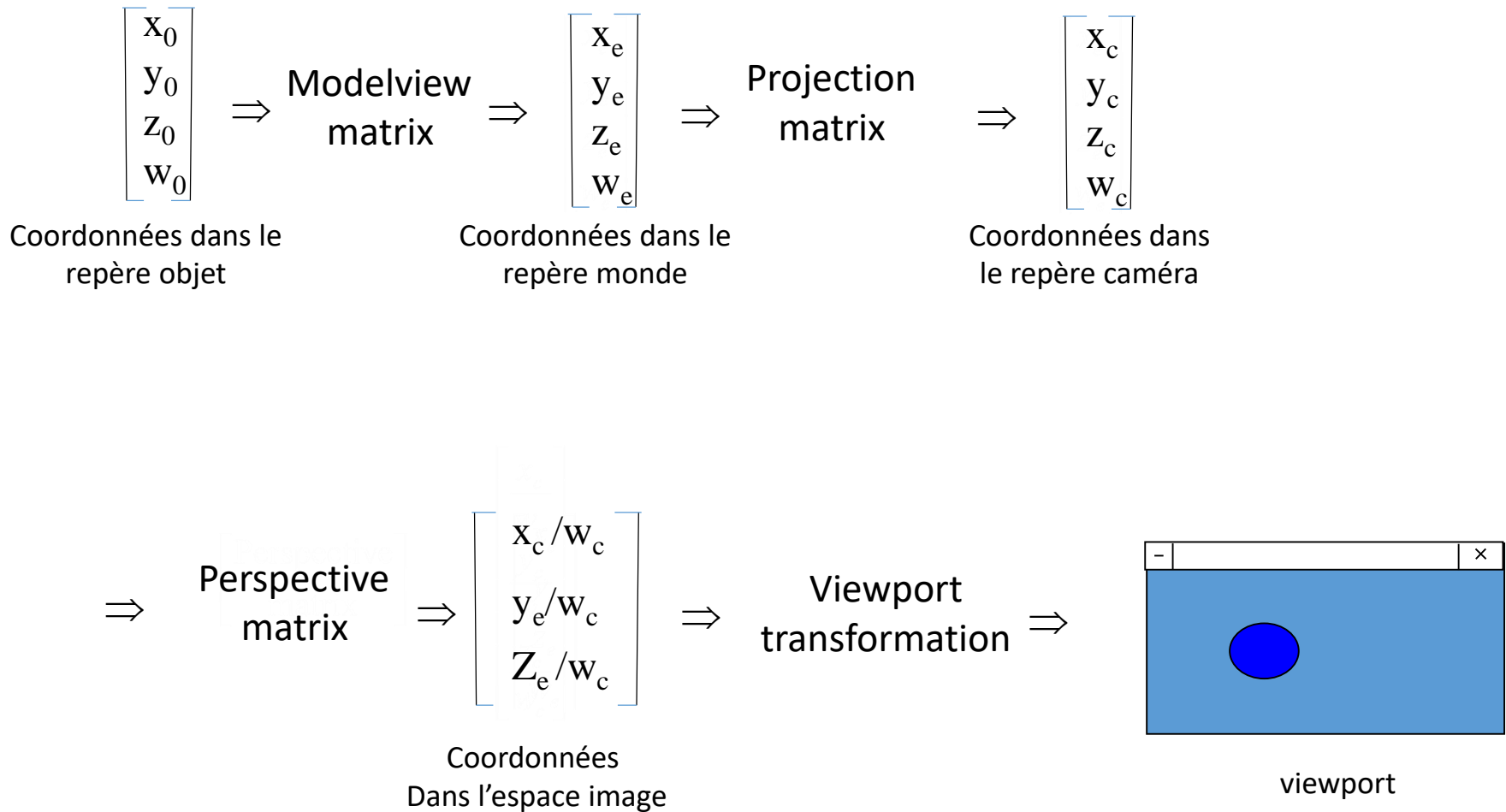
Transformations

- Viewport : de la projection à l'espace image



Transformations

- Pipeline des transformations géométriques**



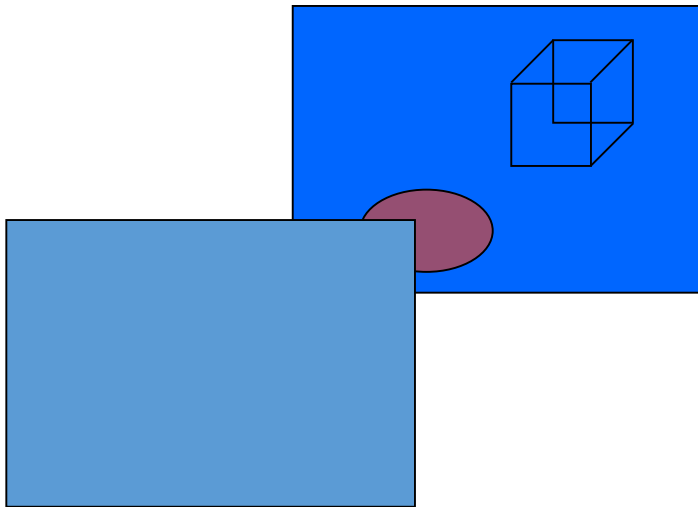
Pipeline graphique et GPU

buffer:缓冲区

- Rendu interactif : astuce du double frame-buffer.
Permet d'éviter qu'on dessine directement sur l'image visible → rendu plus stable.

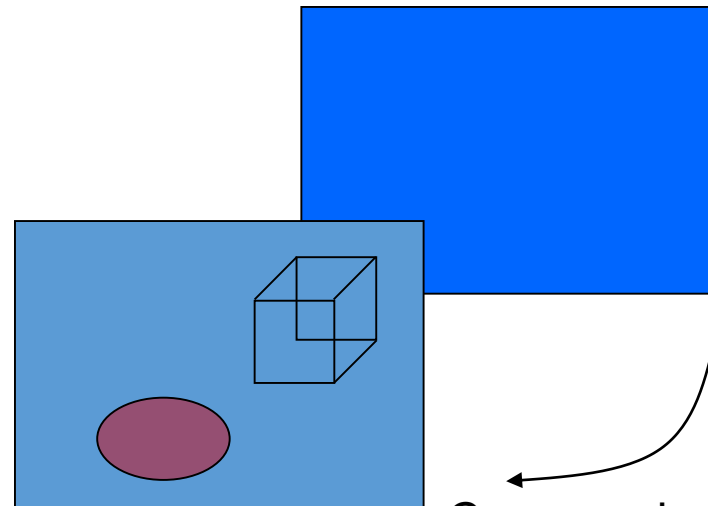
Instant t

Back buffer (rendu en cours)



Front buffer (image courante)

Instant t+1



Commande d'échange
« swap »

1) Inclure la bibliothèque :

```
#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <windows.h>
#include <time.h>
```

2) Initialiser une fenêtre de rendu

```
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    // caracterisation de la fenetre d'affichage
    glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
    // Taille de la fenetre
    glutInitWindowSize (GLB_large, GLB_haut);
    // position de la fenetre
    glutInitWindowPosition (100, 100);
    // creation de la fenetre
    glutCreateWindow ("TD OpenGL Supelec");
}
```

```

void init(void)
{
    // on nettoie le buffer de sortie
    glClearColor (0.0, 0.0, 0.0, 0.0);
    // on autorise l'interpolation de couleur
    glShadeModel (GL_SMOOTH);
    // on autorise le test en profondeur
    glEnable(GL_DEPTH_TEST);
    // initialisation de la position de la particule
    GLB_X=(2*frand()-1)*10;
    GLB_Y=(2*frand()-1)*10;
    glutTimerFunc(40,trigger,20);
    glEnable(GL_TEXTURE_2D);
    tex = LoadTexture("im.bmp");
}

```

*La fonction « trigger » va être exécutée dans 40ms
Avec la valeur « 20 » passée en paramètre*

```

void trigger(int toto)
{
    // on force la fct dispaly a etre execute'e
    GLB_angle_rotation+=0.1;
    // calcul sur la position de la pyramide
    GLB_Z-= 1;
    glutTimerFunc(40,trigger,20);
}

```

```
/* gestion de la projection des objets */  
void reshape (int w, int h)
```

```
{  
  GLfloat fAspect;  
  glViewport (0, 0, (GLsizei) w, (GLsizei) h);  
  glMatrixMode (GL_PROJECTION);  
  glLoadIdentity();  
  if (0){  
    // projection orthographique  
    glOrtho (-1.5, 1.5, -1.5*(GLfloat)h/(GLfloat)w, 1.5*(GLfloat)h/(GLfloat)w, -30.0, 30.0);  
  }  
}
```

Coordonnées des plans de clip verticaux droite et gauche

Coordonnées des plans de clip horizontaux haut et bas

Distances aux plans de clip proche et lointain

```
else{  
  // projection perspective  
  fAspect=(GLfloat)w/(GLfloat)h;  
  // deux derniers para : near and far clipping plane a  
  // partir de l'oeil de la camera : valeur toujours positives !  
  // attention near jamais nul sinon ratio far/near infini : pb d'implementation  
  // (voir p582 ref.manual OpenGL 1.2)  
  gluPerspective(45.0f, fAspect, 10.0, 130.0);  
}
```

Angle de vue dans la direction des y

Ratio (largeur sur hauteur) de l'angle de vue dans la direction des x

Distances aux plans de clip proche et lointain valeurs positives

// point de vue de la camera : sa position, celle de la coordonnée fixe'e,

```
// definition de son axe vertical  
gluLookAt (0.0, 0.0, -30.0, 0.0, 0.0, 0.0, 1.0, 0.0);
```

« up vector » vecteur qui pointe vers le haut de la caméra

Position de la caméra

Position du point filmé

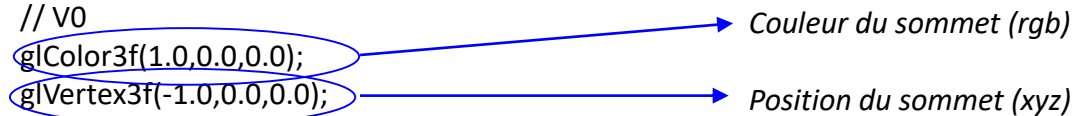
```

/* fct d'affichage */
void display(void)
{
    // nettoyage du buffer d'affichage
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glBegin(GL_TRIANGLE_STRIP);
        // V0
        glColor3f(1.0,0.0,0.0);
        glVertex3f(-1.0,0.0,0.0);
        // V1
        glColor3f(0.0,1.0,0.0);
        glVertex3f(0.0,1.0,0.0);
        // V2
        glColor3f(0.0,0.0,1.0);
        glVertex3f(1.0,0.0,0.0);
        // V3
        glColor3f(0.0,1.0,1.0);
        glVertex3f(0.0,0.0,-1.0);
        // V4
        glColor3f(1.0,0.0,0.0);
        glVertex3f(-1.0,0.0,0.0);
    glEnd();

    // on force OpenGL a executer toutes les commandes
    glFlush ();
    // on swap les buffer de sorties et de travail
    glutSwapBuffers ( );
    // on force la fct display a etre execute'e
    glutPostRedisplay();
}

```


 The diagram consists of two blue arrows. The first arrow originates from the circled arguments (1.0, 0.0, 0.0) of the first `glVertex3f` call and points to the text "Couleur du sommet (rgb)". The second arrow originates from the circled arguments (-1.0, 0.0, 0.0) of the second `glVertex3f` call and points to the text "Position du sommet (xyz)".

```

/* Gestion du clavier */
void keyboard(unsigned char key, int x, int y)
{
    switch (key) {
        case 27:
            exit(0);
            break;
        case 50: // bas
            GLB_Y_curseur-=1;
            break;
        case 56: // haut
            GLB_Y_curseur+=1;
            break;
        case 52: // gauche
            GLB_X_curseur+=1;
            break;
        case 54: // droite
            GLB_X_curseur-=1;
            break;
        case 112: // p pour pause
            getchar();
            break;
    }
    if (GLB_X_curseur>10) GLB_X_curseur=10;
    if (GLB_X_curseur<-10) GLB_X_curseur=-10;
    if (GLB_Y_curseur>10) GLB_Y_curseur=10;
    if (GLB_Y_curseur<-10) GLB_Y_curseur=-10;
}

```



```
GLuint LoadTexture(char *filename)
{
    unsigned char* pMatrice;

    if ((pMatrice= (unsigned char*) malloc(GLB_haut_text*GLB_large_text*3 * sizeof(unsigned char))) == NULL) {
        printf("Impossible d'alouer la memoire d'image");
        exit(-1);}
    }
```

Lecture d'un fichier image

```
BmpReadRVB ("im.bmp", pMatrice, GLB_haut_text, GLB_large_text) ;
```

```
GLuint texID;
glGenTextures(1, &texID);
glBindTexture(GL_TEXTURE_2D, texID);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, GLB_large_text, GLB_haut_text, 0, GL_RGB, GL_UNSIGNED_BYTE, pMatrice);
free(pMatrice);
return texID;
}
```

Dimensions de la texture

Pointeur sur l'image (CPU)

Format de la texture dans le GPU

Format et type des pixels rendus

Chargement de la texture en mémoire GPU

Coordonnées dans la texture
Coordonnées du vertex

```
/* fct d'affichage */
void display(void)
{
    // nettoyage du buffer d'affichage
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    // on va agir sur le referentiel de l'objet
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    // une petite translation
    glTranslatef(GLB_X,GLB_Y,GLB_Z); // peut aller jusqu'a 10 10 -5
    // on applique une rotation sur l'objet
    glRotatef(GLB_angle_rotation,1,1,1);

```

```
glBegin(GL_TRIANGLE_STRIP);
    ...
glEnd();
```

```
// on force Opengl a executer toutes les commandes
glFlush ();
// on swap les buffer de sorties et de travail
glutSwapBuffers ( );
// on force la fct dispaly a etre execute'e
glutPostRedisplay();
```

```
glBegin(GL_TRIANGLE_STRIP);
    // V0
    Utext=78.0f/GLB_large_text;
    Vtext=101.0f/GLB_haut_text;
    glTexCoord2f(Utext,Vtext);
    glVertex3f(-1.0,0.0,0.0);

    // V1
    Utext=96.0/GLB_large_text;
    Vtext=100.0f/GLB_haut_text;
    glTexCoord2f(Utext,Vtext);
    glVertex3f(0.0,1.0,0.0);

    // V2
    Utext=88.0/GLB_large_text;
    Vtext=87.0f/GLB_haut_text;
    glTexCoord2f(Utext,Vtext);
    glVertex3f(1.0,0.0,0.0);

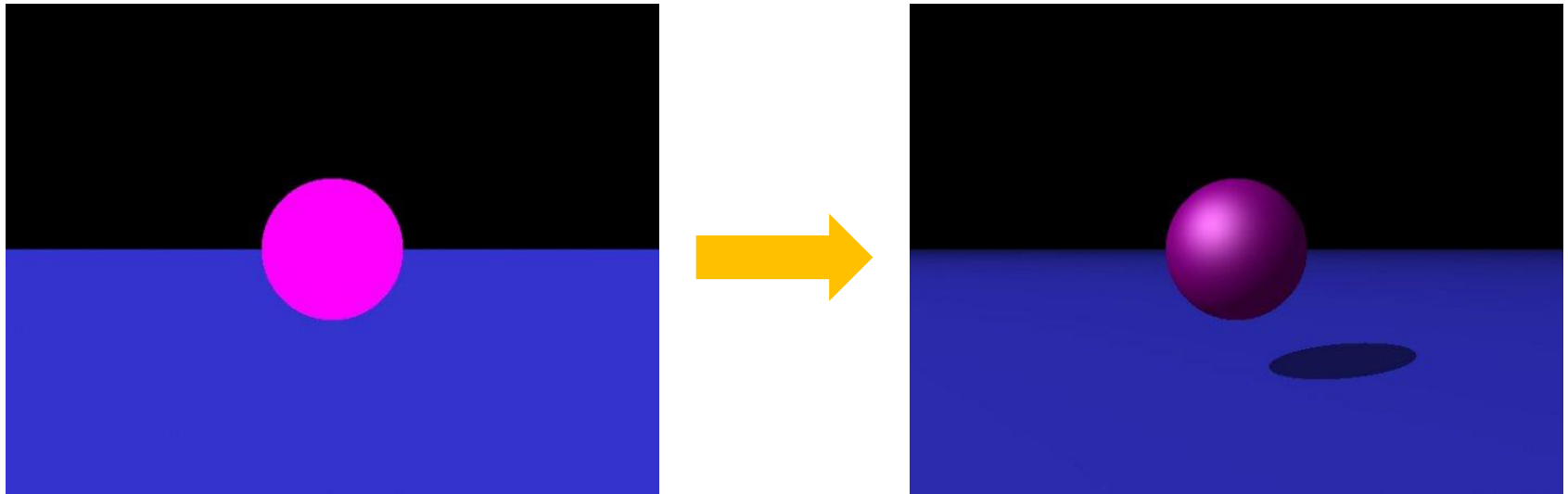
    // V3
    Utext=78.0/GLB_large_text;
    Vtext=101.0f/GLB_haut_text;
    glTexCoord2f(Utext,Vtext);
    glVertex3f(0.0,0.0,-1.0);

    // V4
    Utext=96.0/GLB_large_text;
    Vtext=100.0f/GLB_haut_text;
    glTexCoord2f(Utext,Vtext);
    glVertex3f(-1.0,0.0,0.0);

glEnd();
```

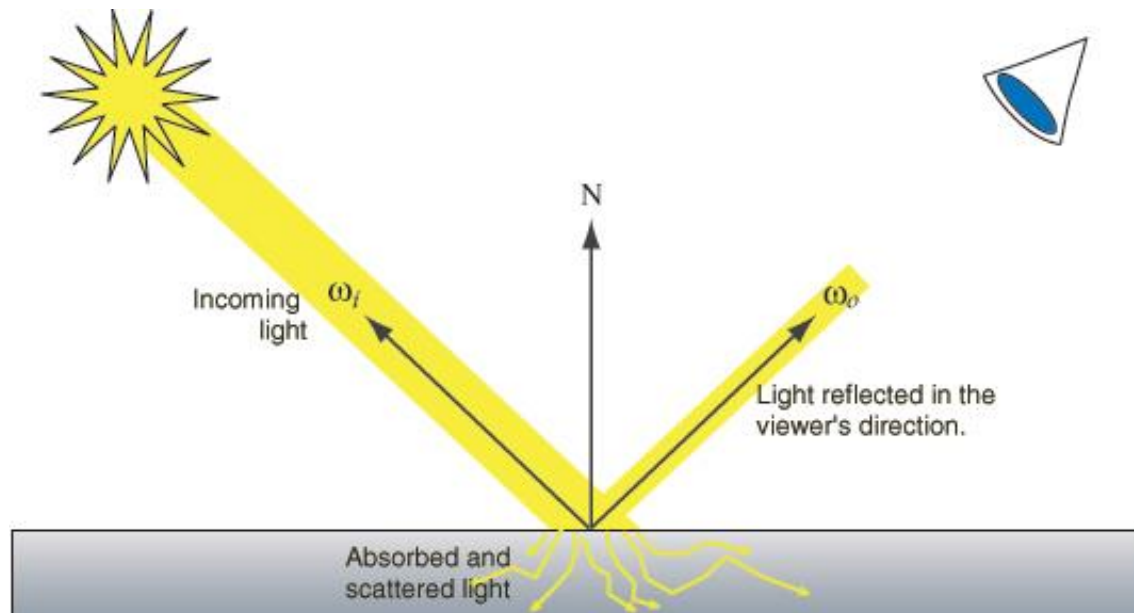
Synthèse d'image

- Shading (« ombrage »)



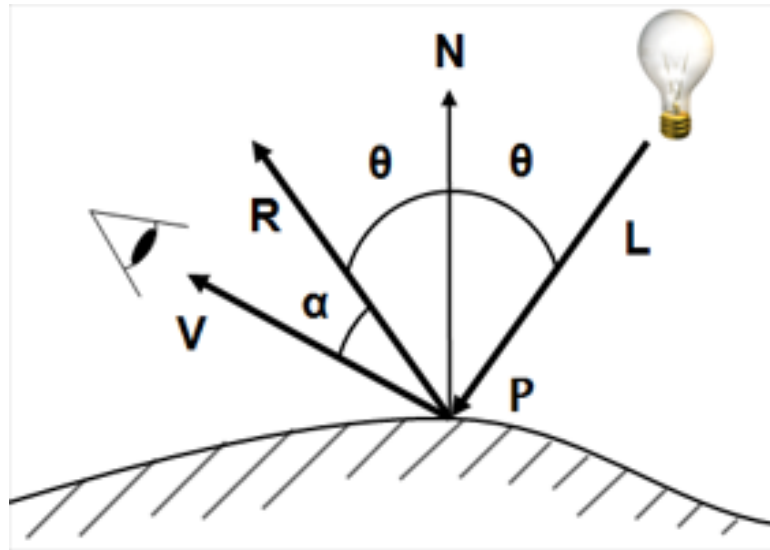
Shading

- L'apparence d'un objet est défini par la quantité de lumière qu'il émet et reflète.
- **Luminosité** : quantité d'énergie lumineuse
- **Couleur** : longueur d'onde



Shading

- Composantes du shading



Principalement :

Lumière **ambiante** L_A

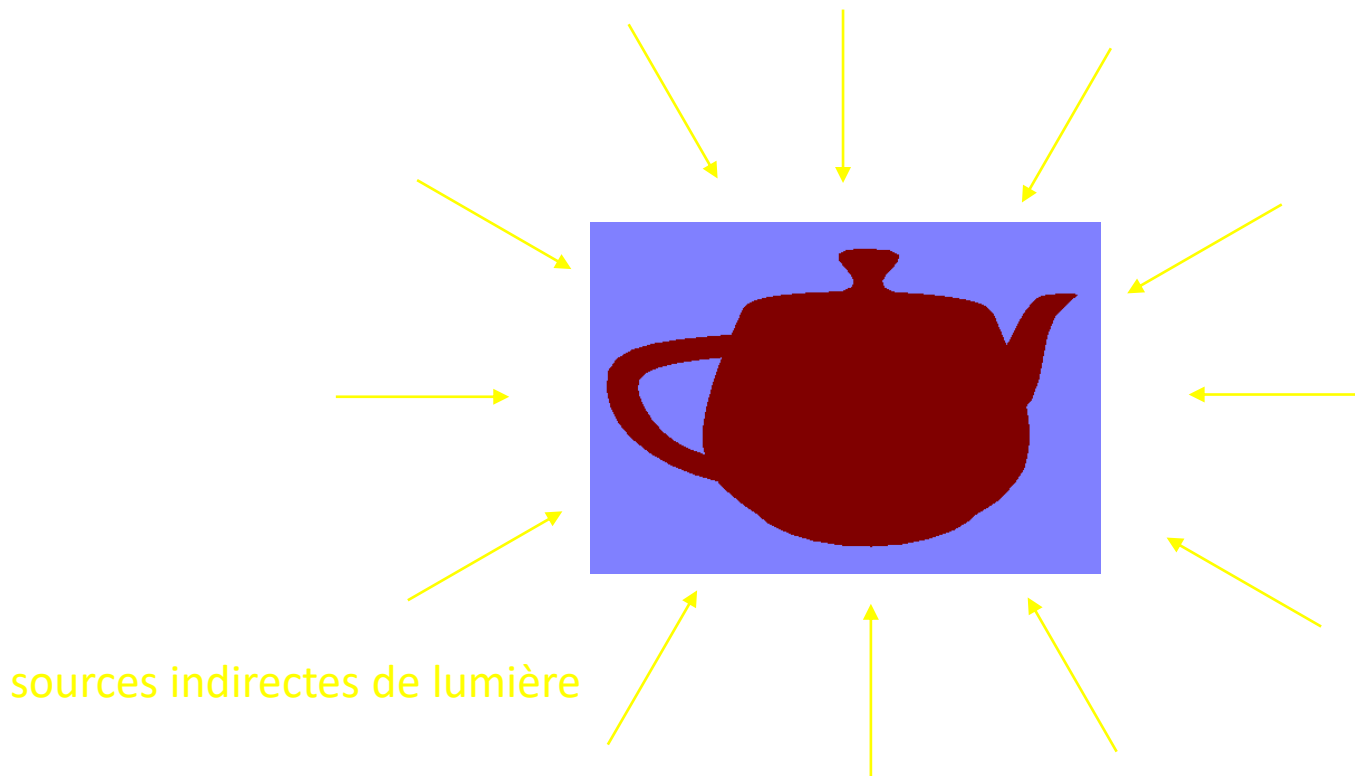
Réflexion **diffuse** L_D

Réflexion **spéculaire** L_S

Résultat visible sur l'image : $L_A + L_D + L_S$

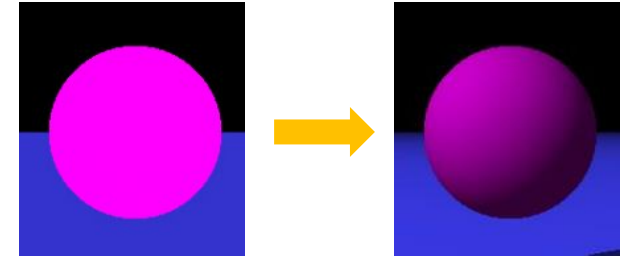
Shading

- Shading ambient : simulation de la lumière indirecte atteignant les objets.

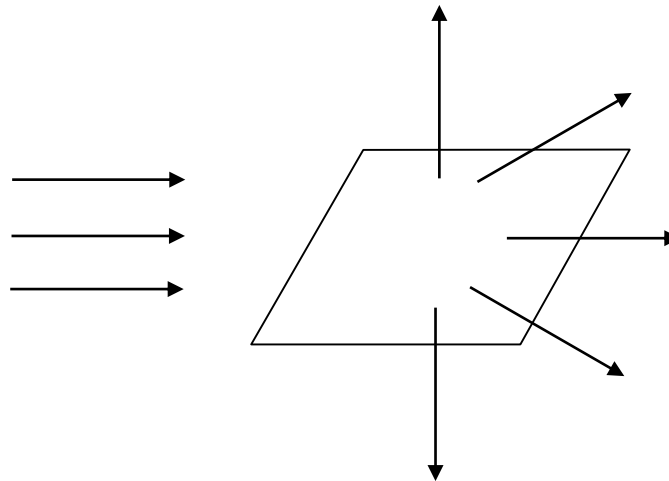


Shading

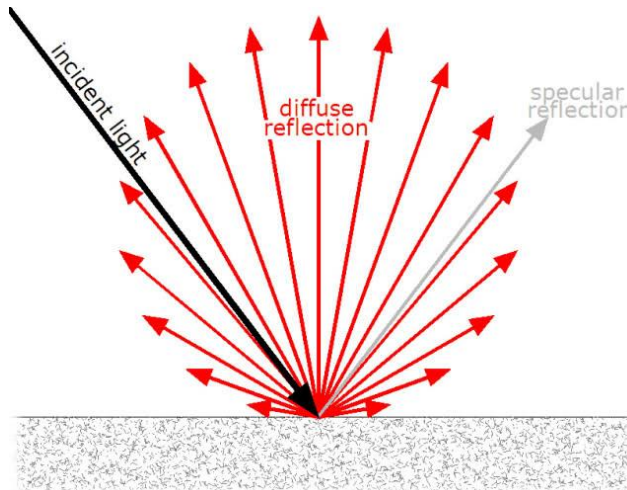
- Shading lumière diffuse



Une source lumineuse



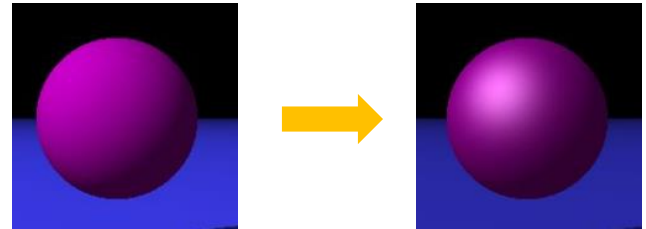
Diffusion dans toutes
Les directions



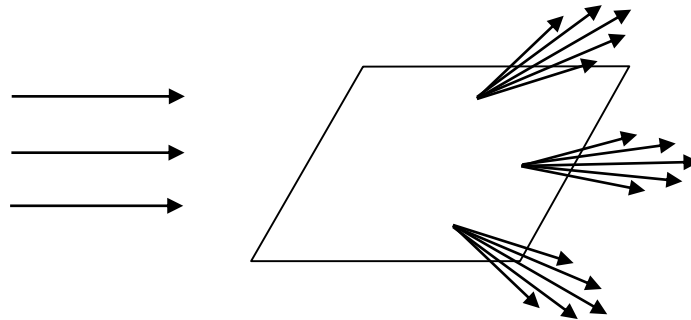
Si lumière rasante : peu de lumière réfléchi

Shading

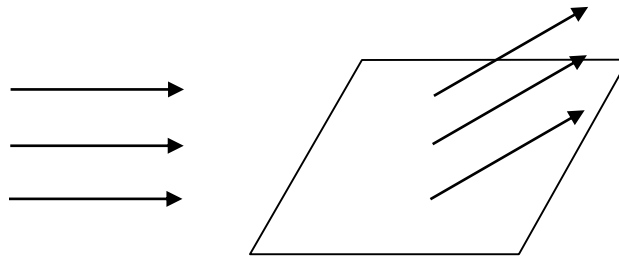
- Shading lumière spéculaire



Une source lumineuse



Diffusion dans une direction précise



Paramétrage : effet miroir

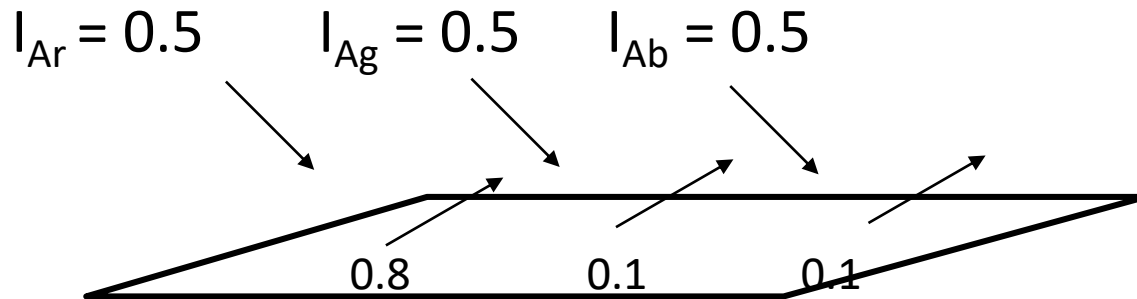
Shading

- Composante diffuse

$$L_A = C_{Argb} \times I_A$$

↑
Coefficient de réflexion du matériau pour la lumière ambiante

↖
Intensité lumineuse provenant de la source

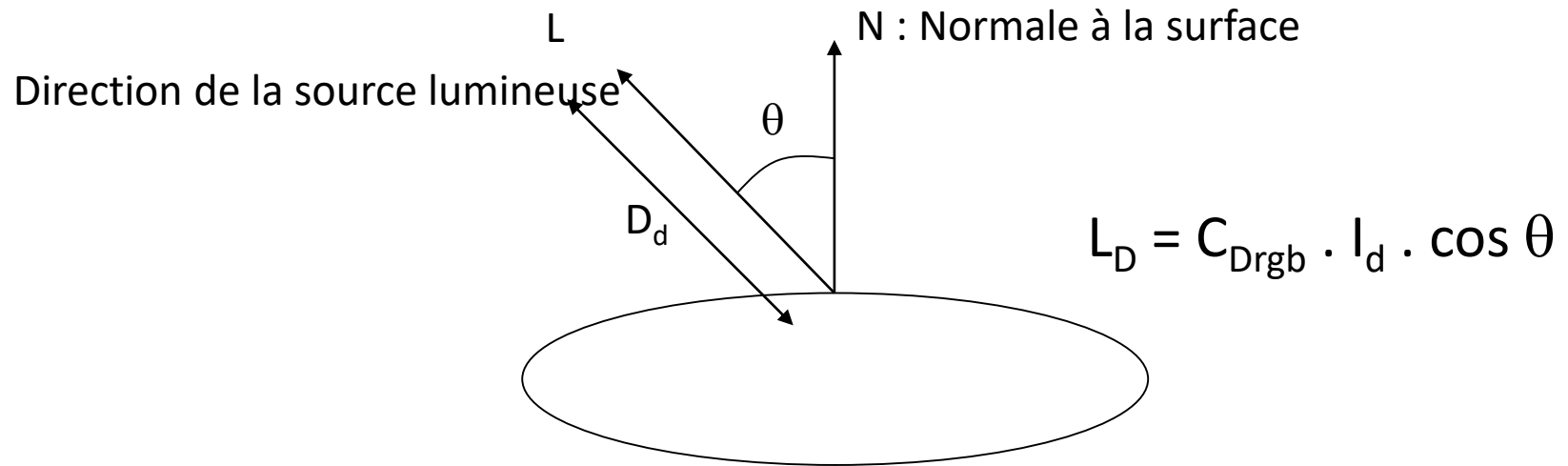


Couleur apparente réfléchiée par le polygone : RGB= (0.45, 0.05, 0.05)

Shading

- Loi de Lambert

La lumière réfléchie est proportionnelle au cosinus de l'angle d'incidence

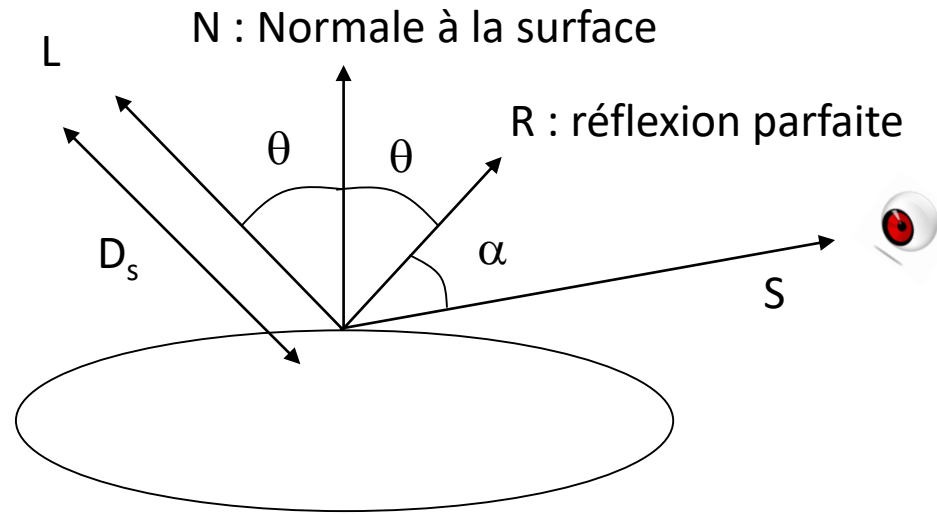


On module l'intensité par la distance de la source lumineuse (D_d) :

$$\begin{aligned} L_D &= (C_{Drgb} \cdot I_d \cdot \cos \theta) / (D_d + d_0) \\ &= (C_{Drgb} \cdot I_d \cdot N \cdot L) / (D_d + d_0) \end{aligned}$$

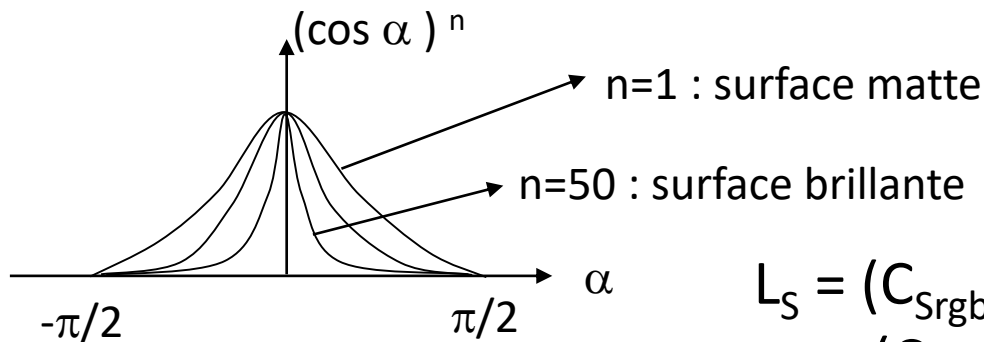
Shading

- Loi de Phong



Loi empirique

Pour des surface brillante, l'intensité réfléchie décroît rapidement avec α
 Pour des surfaces mates, elle décroît plus lentement

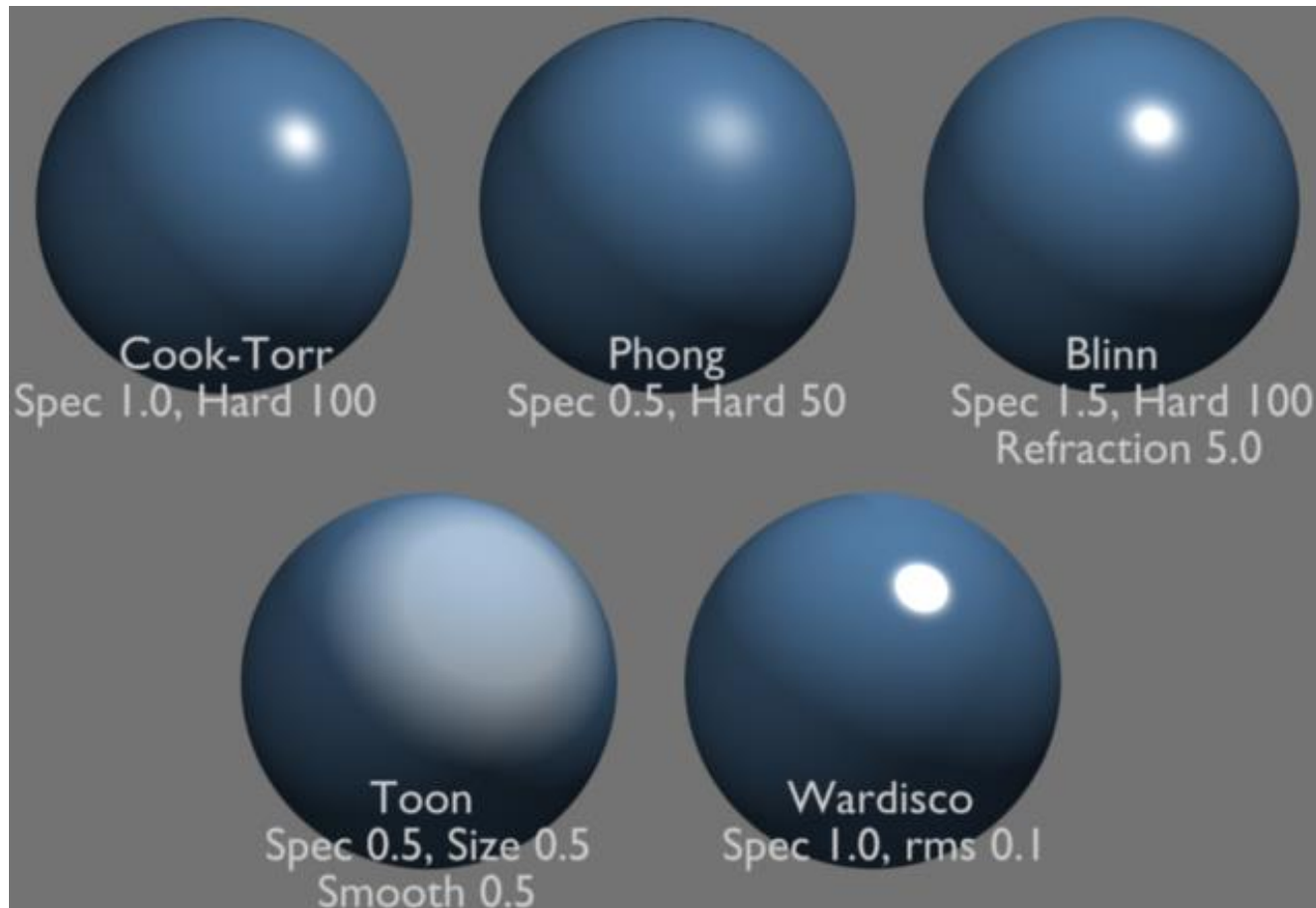


$$L_s = (C_{Srgb} \cdot I_s \cdot (\cos \alpha)^n) / (D_s + d_0)$$

$$= (C_{Srgb} \cdot I_s \cdot (S \cdot R)^n) / (D_s + d_0)$$

Shading

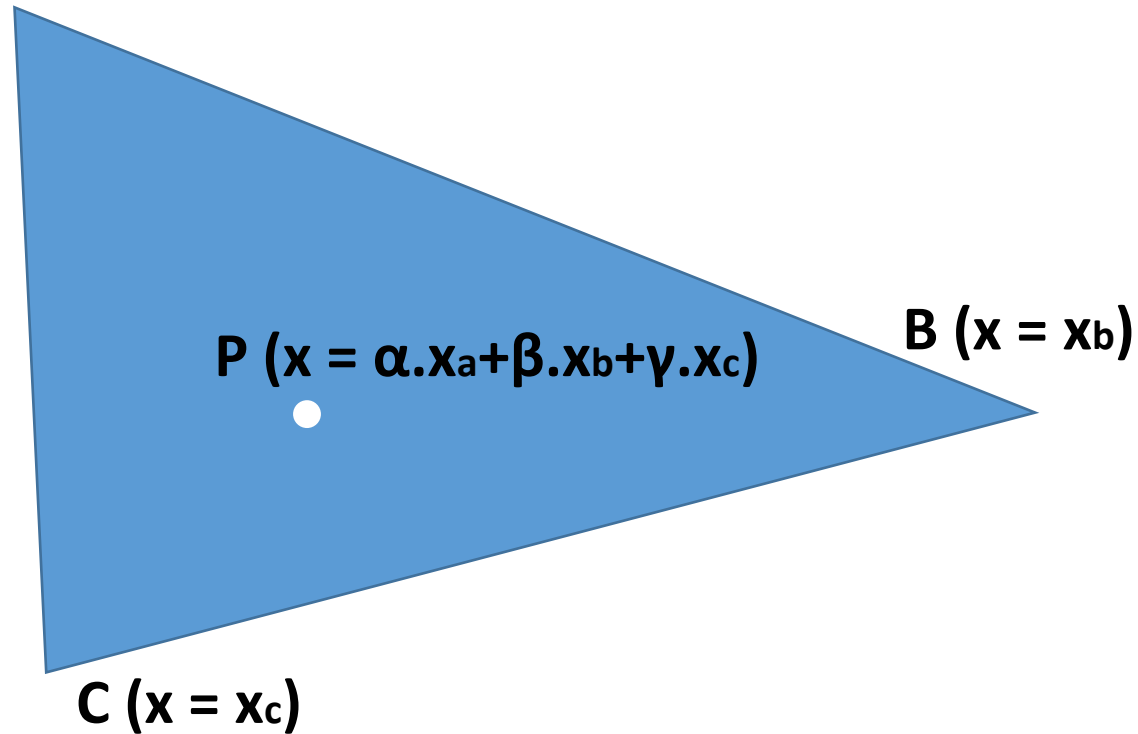
En synthèse d'image, on appelle « **material** » (matériau) la spécification pour un objet des coefficients ambiant, diffuse, et specular.



Shading

- Propriétés de shading pour la surface d'un polygone :
Interpolation des propriétés des sommets !

A ($x = x_a$)



Plusieurs possibilités :

- Interpolation des résultats
- Interpolation des coefficients
- Interpolation de la géométrie (normales)

Shading

- Propriétés de shading pour la surface d'un polygone :
Interpolation des propriété des sommets.
Coordonnées barycentriques ! (barycentric/area coordinates)

(α, β, γ) coord. barycentriques de P

Interprétations :

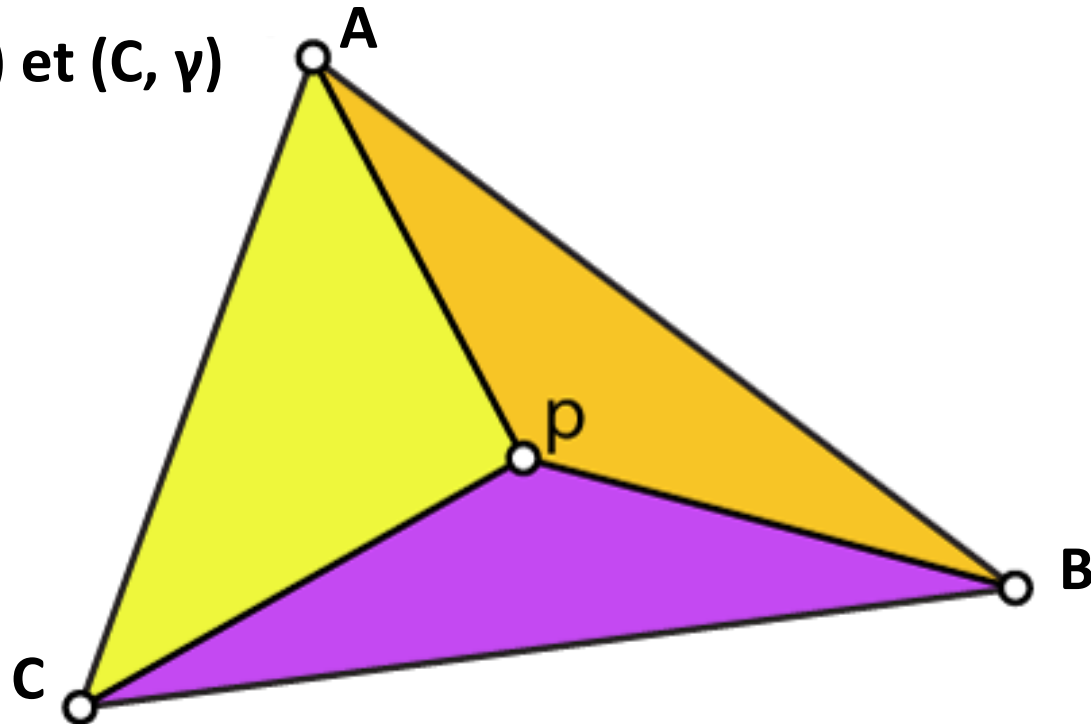
- P barycentre de (A, α) (B, β) et (C, γ)

- $\alpha = \text{aire}(\text{PBC}) / \text{aire}(\text{ABC})$

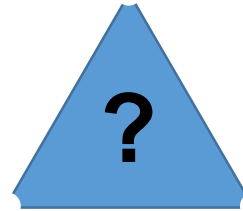
$$\beta = \text{aire}(\text{PAC}) / \text{aire}(\text{ABC})$$

$$\gamma = \text{aire}(\text{PAB}) / \text{aire}(\text{ABC})$$

Avec toujours : $\alpha + \beta + \gamma = 1$



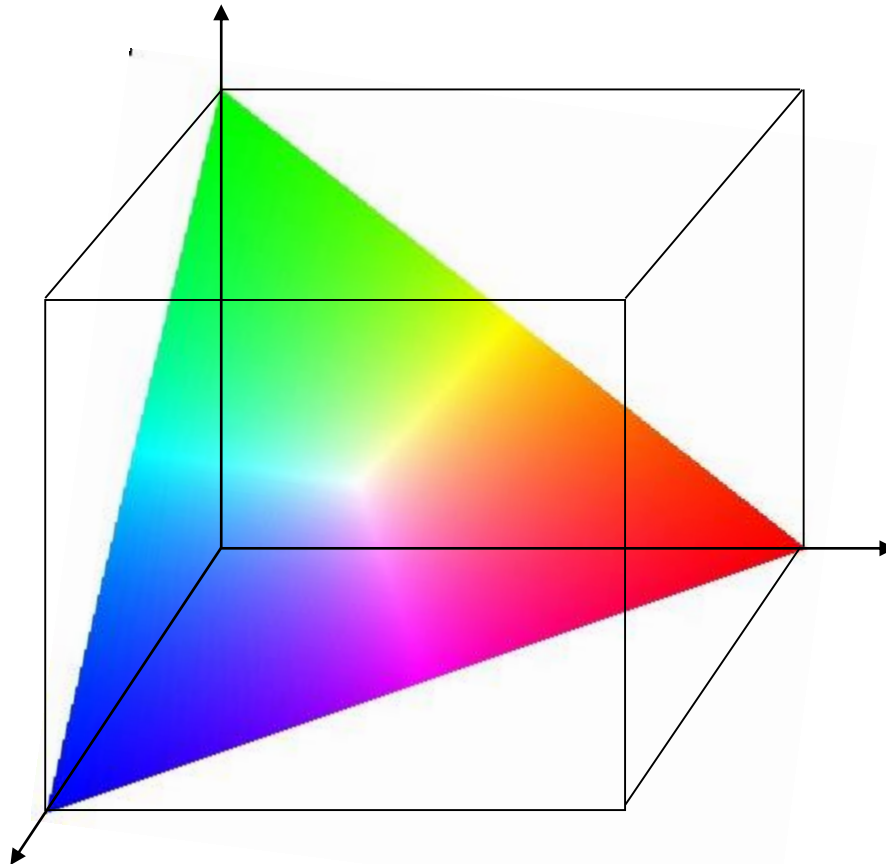
A (0,255,0)



B (0,0,255)

C (255,0,0)

Exemple de shading
(couleur seulement) :



Shading

- Codage couleur (par pixel)

Mode 24 bits (RGB = 8+8+8) : 16 millions de couleurs

Mode RGBA : 32 bits (RGBA = 8+8+8+8) : plus rapide car câblé sur les cartes graphiques.

Canal A (*alpha*) pour l'interpolation entre deux surfaces superposées :

$$R = \alpha R_1 + (1 - \alpha) R_2$$

$$G = \dots$$

$$B = \dots$$

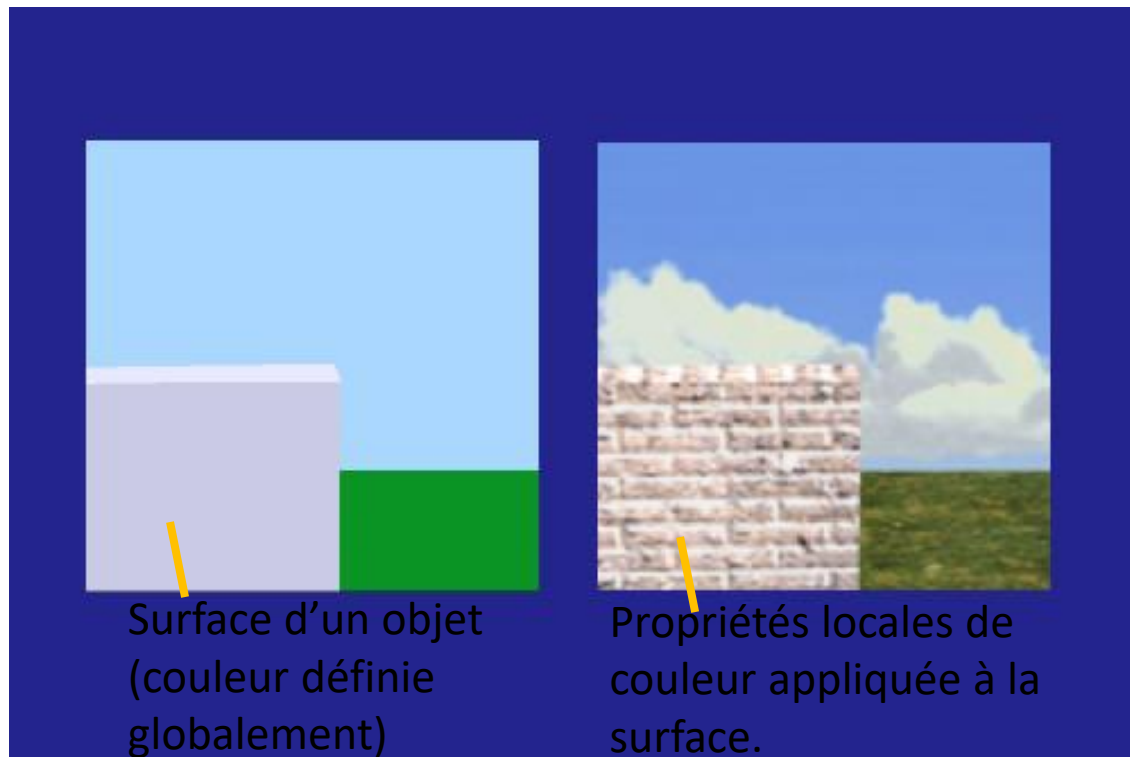
Permet des effets de transparence/opacité.

Shading

- Textures :

Texturing = Définition locales des propriétés de surface (couleur, réflexion, ...) d'un objet.

On associe donc à la surface d'un objet une fonction 2D (souvent une image) qui définit la variation locale du paramètre considéré.

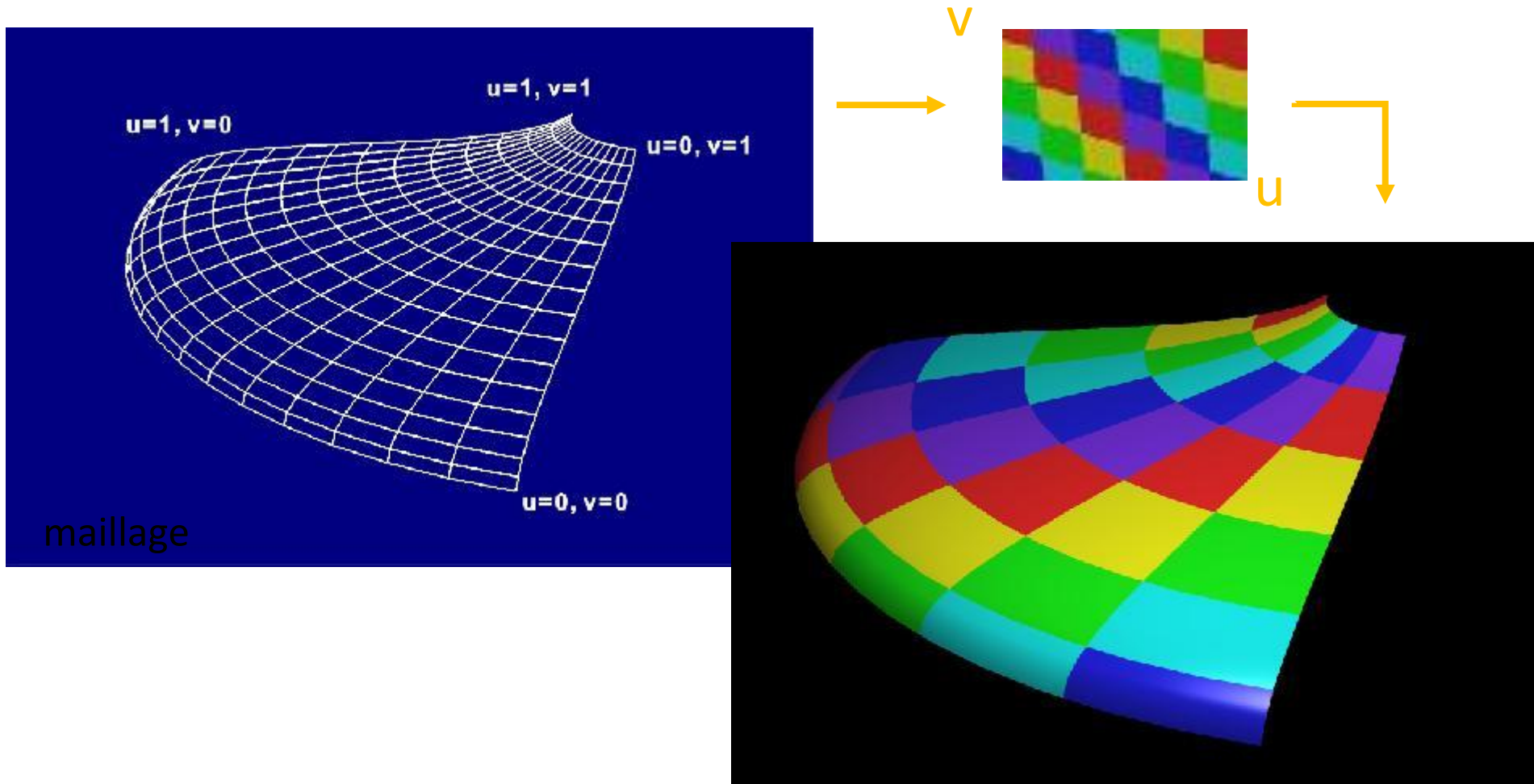


Textures

- Mapping UV : Spécification « sommet par sommet » de la correspondance maillage-texture.
- Fonctionnement : spécifier pour chaque sommet quelle zone de l'image doit lui être « épinglée ».
- **U, V** : coordonnées en pixel dans une image 2D (la texture)

Textures

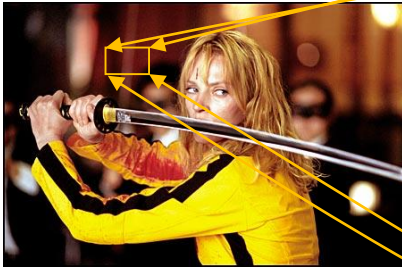
- Mapping UV : Spécification « sommet par sommet de la correspondance maillage-texture.



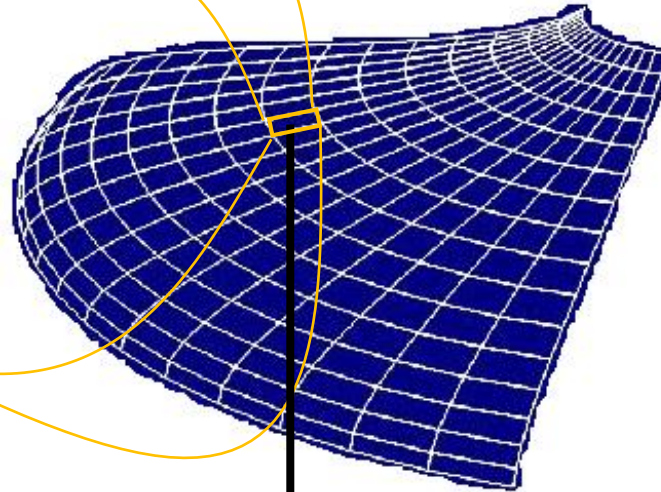
Textures

- Mapping UV

v



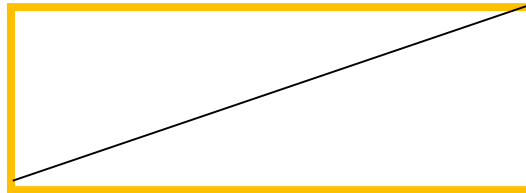
u



$V2 (X_2, Y_2, Z_2, U_2, V_2)$

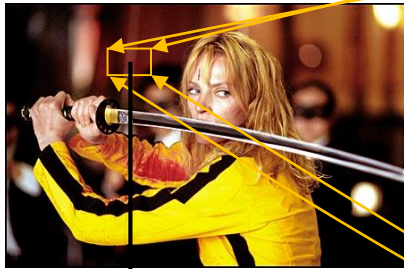
$V3 (X_3, Y_3, Z_3, U_3, V_3)$

$V1 (X_1, Y_1, Z_1, U_1, V_1)$



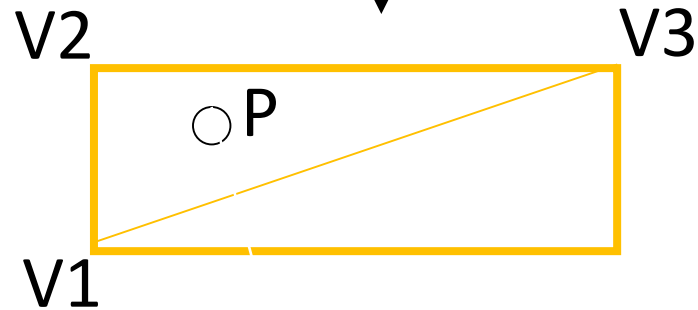
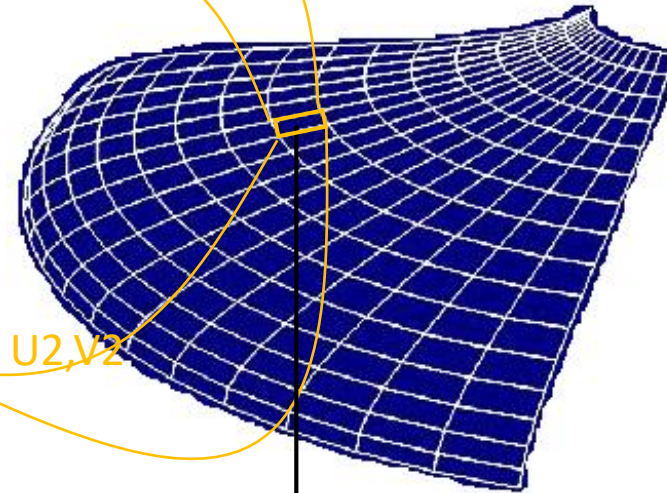
Textures

- Mapping UV



$$U_p = \alpha.U_1 + \beta.U_2 + \gamma.U_3$$

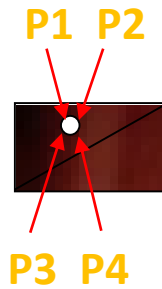
$$V_p = \alpha.V_1 + \beta.V_2 + \gamma.V_3$$



$$P = \alpha.V_1 + \beta.V_2 + \gamma.V_3$$

Textures

- **Mapping UV** : en général, les coordonnées U, V interpolées ne tombent pas sur des coordonnées pixel entières.
- On interpole les valeurs des voisins.

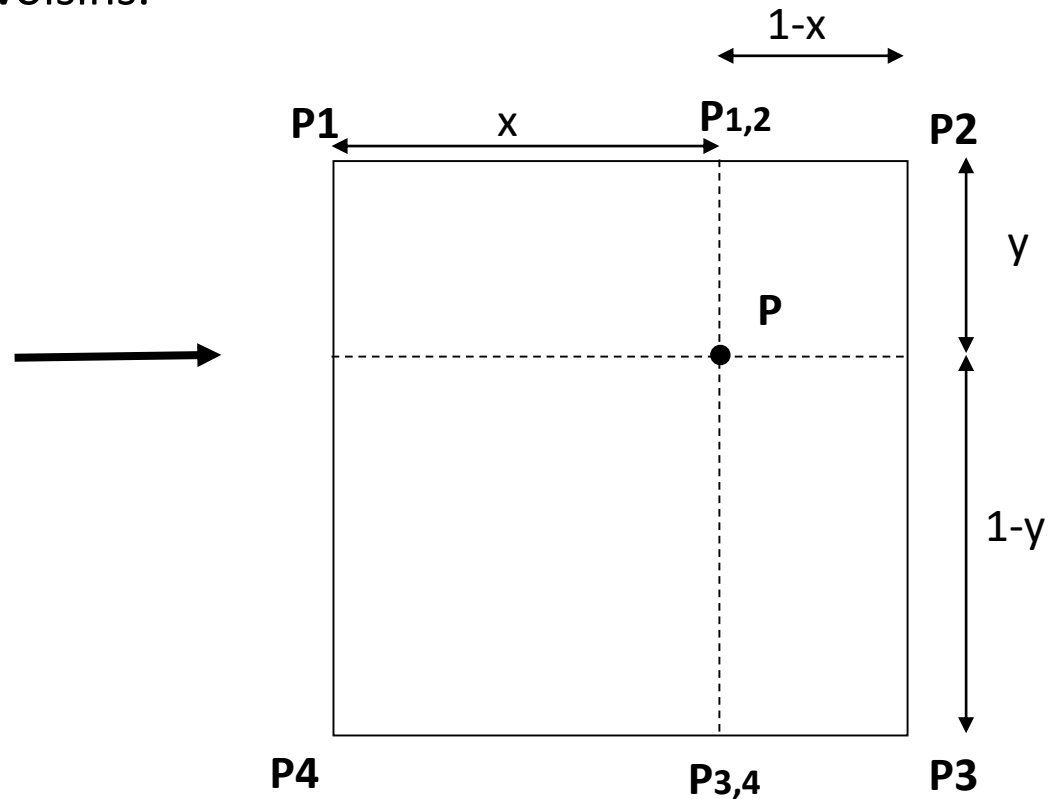


$$P_{1,2} = (1-x) \cdot P1 + x \cdot P2$$

$$P_{3,4} = (1-x) \cdot P4 + x \cdot P3$$

$$P = (1-y) \cdot P_{1,2} + y \cdot P_{3,4}$$

Interpolation bilinéaire !



Textures

- Résultats

