

Images de synthèse

Igor Stéphan

UFR Sciences Angers

2018-2019

Plan

1 Introduction

2 Eléments de mathématiques

3 Les transformations en action

4 Les algorithmes du rendu

La synthèse d'images

- Objectif : produire des images synthétiques.
- Simulateurs, cinéma, jeux vidéo, publicité... .
- Deux approches différentes :
 - le rendu temps-réel (simulateurs, jeux vidéo),
 - le rendu temps différé (images photo-réalistes, animation).
- Limites techniques :
 - technologie de restitution,
 - complexité calculatoire des algorithmes,
 - un espace mémoire borné.

Cadre du cours de synthèse d'images

- Le rendu temps-réel . . .
- par discréttisation d'objets «réels» 3D continus vers 2D discret . . .
- pour remplir un tampon de *pixels* (picture elements) . . .
- tandis qu'un autre tampon est présenté à l'écran . . .
- pour tromper l'œil et donner l'illusion du mouvement.
- Outils théoriques :
 - Maillages (polygones convexes).
 - Transformations géométriques (translation, rotation, homothétie, cisaillement).
 - Point de vue et projection 2D de visualisation.
 - Couleur, éclairage, transparence, matière, texture, ombrage, etc . . .
- Outil technique : OpenGL

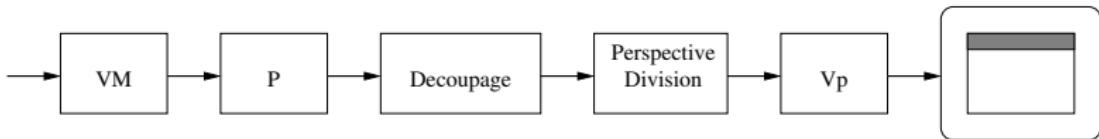
Qu'est-ce qu'OpenGL

- Une interface logicielle avec la carte graphique.
- Une librairie graphique portable *open source*.
 - Une machine à états.
 - Un ensemble de tampons (*buffers*).
 - Un pipeline de transformation.
- Une mise-en-œuvre d'un processus de restitution :
 - Construction de formes géométriques primitives (des «objets») ;
 - Organisation des objets dans un espace tridimensionnel ;
 - Sélection du point de vue ;
 - Elimination des objets et faces cachées ;
 - Calcul des couleurs des objets (couleur, éclairage, transparence, matière, texture, etc...) ;
 - Conversion en mode point («rastérisation»).

Ce que n'est ou ne fait pas OpenGL

- Un ensemble de primitives graphiques de haut niveau (GLU) ;
- Une programmation dirigée par les événements (GLUT) :
 - une fonction d'initialisation de la fenêtre de vue,
 - une fonction d'initialisation de la machine OpenGL,
 - une fonction de rafraîchissement de la scène (modélisation des objets et du point de vue),
 - une fonction par type d'événements (clavier, clics souris, redimensionnement de la fenêtre),
 - une boucle perpétuelle.
- Un accès aux extensions (GLEW)
- OpenGL ne traite pas de la cohérence de l'univers.

Le pipeline d'OpenGL



- VM transformations de modélisation M et de visualisation V (*modelview matrix*),
- P transformation de projection (*projection matrix*),
- V_p transformation de cadrage (*viewport matrix*).
- En entrée : des sommets (pouvant former points, lignes et polygones).
- En sortie : des pixels.
- Une architecture Client/Serveur.

Les primitives graphiques d'OpenGL

- Une liste de sommets entre deux commandes :
`glBegin(<Mode>)` et `glEnd()`
- Avec pour `<Mode>` possibles :
 - `GL_POINTS` pour des points
 - `GL_LINES` 2 par 2 pour des segments
 - `GL_LINE_STRIP` pour une ligne brisée
 - `GL_LINE_LOOP` pour une ligne brisée fermée
 - `GL_TRIANGLES` 3 par 3 pour des triangles
 - `GL_QUADS` 4 par 4 pour des quadrilatères convexes
 - `GL_POLYGON` pour un polygone convexe
- Des états pour la taille des points, l'épaisseur des lignes, la continuité des lignes, les bordures des polygones, ...

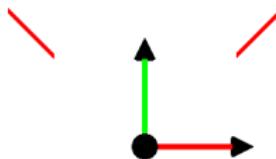
Les primitives graphiques d'OpenGL : GL_POINTS pour des points

```
glBegin(GL_POINTS);
glVertex3f(1.0f, 1.0f, 1.0f);
glVertex3f(1.0f, 1.0f, -1.0f);
glVertex3f(-1.0f, 1.0f, -1.0f);
glVertex3f(-1.0f, 1.0f, 1.0f);
glEnd();
```



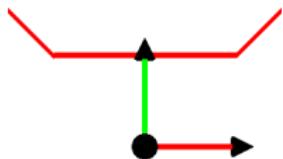
Les primitives graphiques d'OpenGL : GL_LINES 2 par 2 pour des segments

```
glBegin(GL_LINES);
glVertex3f(1.0f, 1.0f, 1.0f);
glVertex3f(1.0f, 1.0f, -1.0f);
glVertex3f(-1.0f, 1.0f, -1.0f);
glVertex3f(-1.0f, 1.0f, 1.0f);
glEnd();
```



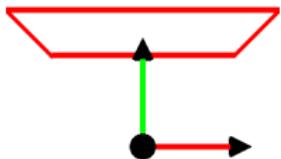
Les primitives graphiques d'OpenGL : GL_LINE_STRIP pour une ligne brisée

```
glBegin(GL_LINE_STRIP);
glVertex3f(1.0f, 1.0f, 1.0f);
glVertex3f(1.0f, 1.0f, -1.0f);
glVertex3f(-1.0f, 1.0f, -1.0f);
glVertex3f(-1.0f, 1.0f, 1.0f);
glEnd();
```



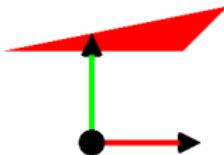
Les primitives graphiques d'OpenGL : GL_LINE_LOOP pour une ligne brisée fermée

```
glBegin(GL_LINE_LOOP);  
glVertex3f(1.0f, 1.0f, 1.0f);  
glVertex3f(1.0f, 1.0f, -1.0f);  
glVertex3f(-1.0f, 1.0f, -1.0f);  
glVertex3f(-1.0f, 1.0f, 1.0f);  
glEnd();
```



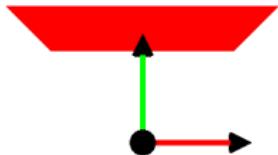
Les primitives graphiques d'OpenGL : GL_TRIANGLES 3 par 3 pour des triangles

```
glBegin(GL_TRIANGLES);
glVertex3f(1.0f, 1.0f, 1.0f);
glVertex3f(1.0f, 1.0f, -1.0f);
glVertex3f(-1.0f, 1.0f, -1.0f);
glVertex3f(-1.0f, 1.0f, 1.0f); /* ignoré */
glEnd();
```



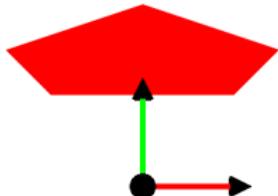
Les primitives graphiques d'OpenGL : GL_QUADS 4 par 4 pour des quadrilatères convexes

```
glBegin(GL_QUADS);  
glVertex3f(1.0f, 1.0f, 1.0f);  
glVertex3f(1.0f, 1.0f, -1.0f);  
glVertex3f(-1.0f, 1.0f, -1.0f);  
glVertex3f(-1.0f, 1.0f, 1.0f);  
glEnd();
```



Les primitives graphiques d'OpenGL : GL_POLYGON pour un polygone convexe

```
glBegin(GL_POLYGON);
glVertex3f(1.0f, 1.0f, 1.0f);
glVertex3f(1.0f, 1.0f, -1.0f);
glVertex3f(-1.0f, 1.0f, -1.0f);
glVertex3f(-1.0f, 1.0f, 1.0f);
glVertex3f(0.0f, 1.0f, 2.0f);
glEnd();
```



Plan

- 1 Introduction
- 2 Eléments de mathématiques
- 3 Les transformations en action
- 4 Les algorithmes du rendu

Lois élémentaires dans \mathbb{R}^n

- Somme dans \mathbb{R}^n : si $a = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}$, $b = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix} \in \mathbb{R}^n$

$$\text{alors } a + b = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} + \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix} = \begin{pmatrix} a_1 + b_1 \\ \vdots \\ a_n + b_n \end{pmatrix}.$$

- Multiplication par un *scalaire* dans \mathbb{R}^n :

$$a = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} \in \mathbb{R}^n, \alpha \in \mathbb{R}$$

$$\text{alors } \alpha a = \alpha \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} \alpha a_1 \\ \vdots \\ \alpha a_n \end{pmatrix}.$$

Vecteur de \mathbb{R}^3 et norme d'un vecteur de \mathbb{R}^3

- Un vecteur de \mathbb{R}^3 est un élément de \mathbb{R}^3
- La norme d'un vecteur est la longueur du vecteur
- $\left\| \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \right\| = \sqrt{v_x^2 + v_y^2 + v_z^2}$
- \mathbf{u} vecteur unitaire si $\|\mathbf{u}\| = 1$
- Le vecteur $\frac{1}{\|\mathbf{v}\|} \mathbf{v}$ est le vecteur normalisé du vecteur \mathbf{v} .

Système « main-droite »

- Système de coordonnées tridimensionnelles :
 - l'axe des x : le pouce,
 - l'axe des y : l'indexe,
 - l'axe des z : le majeur.
- Système « main-droite » en OpenGL :
 - l'axe des x horizontal (négatif à gauche et positif à droite),
 - l'axe des y vertical (positif vers le haut et négatif vers le bas),
 - l'axe des z perpendiculairement à l'écran (négatif vers l'intérieur et positif vers l'extérieur).
- Un repère orthonormé pour système « main-droite » ($\mathbf{i}, \mathbf{j}, \mathbf{k}$) :
 - vecteur unitaire \mathbf{i} selon l'axe des x ;
 - vecteur unitaire \mathbf{j} selon l'axe des y ;
 - vecteur unitaire \mathbf{k} selon l'axe des z .

Produit scalaire dans \mathbb{R}^3

- Produit scalaire de deux vecteurs $\mathbf{u} = \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix}$ et $\mathbf{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$: $\mathbf{u} \cdot \mathbf{v} = u_x v_x + u_y v_y + u_z v_z$
- Dans un repère orthonormé, \mathbf{u} et \mathbf{v} formant un angle θ :
$$\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\| \|\mathbf{v}\| \cos \theta$$

Produit vectoriel dans \mathbb{R}^3

- Produit vectoriel de deux vecteurs $\mathbf{u} = \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix}$ et $\mathbf{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$ dans un repère orthonormé :
$$\mathbf{u} \times \mathbf{v} = \begin{pmatrix} u_y v_z - u_z v_y \\ u_z v_x - u_x v_z \\ u_x v_y - u_y v_x \end{pmatrix}$$

Produit vectoriel dans \mathbb{R}^3

- Si θ est l'angle de \mathbf{u} à \mathbf{v} alors
$$\begin{pmatrix} u_x \\ u_y \\ 0 \end{pmatrix} \times \begin{pmatrix} v_x \\ v_y \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ u_x v_y - u_y v_x \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ ||\mathbf{u}|| ||\mathbf{v}|| \sin \theta \end{pmatrix}$$

Produit de matrices 4×4

Produit NM de deux matrices 4×4 :

$$\begin{pmatrix}
 n_{11} & n_{12} & n_{13} & n_{14} \\
 n_{21} & n_{22} & n_{23} & n_{24} \\
 n_{31} & n_{32} & n_{33} & n_{34} \\
 n_{41} & n_{42} & n_{43} & n_{44}
 \end{pmatrix}
 \begin{pmatrix}
 m_{11} & m_{12} & m_{13} & m_{14} \\
 m_{21} & m_{22} & m_{23} & m_{24} \\
 m_{31} & m_{32} & m_{33} & m_{34} \\
 m_{41} & m_{42} & m_{43} & m_{44}
 \end{pmatrix} =$$

$$\begin{pmatrix}
 \sum_{i=1}^4 n_{1i} m_{i1} & \sum_{i=1}^4 n_{1i} m_{i2} & \sum_{i=1}^4 n_{1i} m_{i3} & \sum_{i=1}^4 n_{1i} m_{i4} \\
 \sum_{i=1}^4 n_{2i} m_{i1} & \sum_{i=1}^4 n_{2i} m_{i2} & \sum_{i=1}^4 n_{2i} m_{i3} & \sum_{i=1}^4 n_{2i} m_{i4} \\
 \sum_{i=1}^4 n_{3i} m_{i1} & \sum_{i=1}^4 n_{3i} m_{i2} & \sum_{i=1}^4 n_{3i} m_{i3} & \sum_{i=1}^4 n_{3i} m_{i4} \\
 \sum_{i=1}^4 n_{4i} m_{i1} & \sum_{i=1}^4 n_{4i} m_{i2} & \sum_{i=1}^4 n_{4i} m_{i3} & \sum_{i=1}^4 n_{4i} m_{i4}
 \end{pmatrix}$$

Produit d'une matrice 4×4 et d'un élément de \mathbb{R}^4

Produit Na d'un matrice 4×4 par un élément a de \mathbb{R}^4 :

$$\begin{pmatrix} n_{xx} & n_{xy} & n_{xz} & n_{xw} \\ n_{yx} & n_{yy} & n_{yz} & n_{yw} \\ n_{zx} & n_{zy} & n_{zz} & n_{zw} \\ n_{wx} & n_{wy} & n_{wz} & n_{ww} \end{pmatrix} \begin{pmatrix} a_x \\ a_y \\ a_z \\ a_w \end{pmatrix} =$$

$$\begin{pmatrix} n_{xx}a_x + n_{xy}a_y + n_{xz}a_z + n_{xw}a_w \\ n_{yx}a_x + n_{yy}a_y + n_{yz}a_z + n_{yw}a_w \\ n_{zx}a_x + n_{zy}a_y + n_{zz}a_z + n_{zw}a_w \\ n_{wx}a_x + n_{wy}a_y + n_{wz}a_z + n_{ww}a_w \end{pmatrix}$$

Transformations dans \mathbb{R}^n

- Une *transformation* est une fonction de \mathbb{R}^n dans lui-même.
- Une transformation A est linéaire si :
 - $a \in \mathbb{R}^n, \alpha \in \mathbb{R}$ alors $A(\alpha a) = \alpha A(a)$
 - $a, b \in \mathbb{R}^n$ alors $A(a + b) = A(a) + A(b)$
- Une transformation A est une *translation* s'il existe $v \in \mathbb{R}^n$ tel que pour tout $a \in \mathbb{R}^n$, $A(a) = a + v$.
- La composition de deux transformations A et B est notée $A \circ B$ et définie $(A \circ B)(a) = A(B(a))$.
- Une transformation A est affine si elle est la composition d'une translation T_v et d'une transformation linéaire B :
$$A = T_v \circ B.$$

Représentation par matrice des transformations linéaires de \mathbb{R}^3

- $\mathbf{i} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \mathbf{j} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \mathbf{k} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$
- un vecteur $\mathbf{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$ se décompose (linéairement) de manière unique en $\mathbf{v} = v_x\mathbf{i} + v_y\mathbf{j} + v_z\mathbf{k}$

- $\mathbf{v} = v_x \mathbf{i} + v_y \mathbf{j} + v_z \mathbf{k}$

- A une transformation linéaire alors

$$A(v_x \mathbf{i} + v_y \mathbf{j} + v_z \mathbf{k}) = v_x A(\mathbf{i}) + v_y A(\mathbf{j}) + v_z A(\mathbf{k})$$

- $\mathbf{i}^A = A(\mathbf{i}) = \begin{pmatrix} i_x^A \\ i_y^A \\ i_z^A \end{pmatrix}, \mathbf{j}^A = A(\mathbf{j}) = \begin{pmatrix} j_x^A \\ j_y^A \\ j_z^A \end{pmatrix},$

$$\mathbf{k}^A = A(\mathbf{k}) = \begin{pmatrix} k_x^A \\ k_y^A \\ k_z^A \end{pmatrix} \text{ alors } A(\mathbf{v}) =$$

$$\begin{pmatrix} v_x i_x^A + v_y j_x^A + v_z k_x^A \\ v_x i_y^A + v_y j_y^A + v_z k_y^A \\ v_x i_z^A + v_y j_z^A + v_z k_z^A \end{pmatrix} = \begin{pmatrix} i_x^A & j_x^A & k_x^A \\ i_y^A & j_y^A & k_y^A \\ i_z^A & j_z^A & k_z^A \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$$

Coordonnées homogènes

- Les points $\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$ et $\begin{pmatrix} W * x \\ W * y \\ W * z \\ W \end{pmatrix}$ pour $W \neq 0$ représentent le même point de l'espace 3D.

- Homogénéisation d'un point homogène $\begin{pmatrix} x \\ y \\ z \\ W \end{pmatrix}$ en $\begin{pmatrix} \frac{x}{W} \\ \frac{y}{W} \\ \frac{z}{W} \\ 1 \end{pmatrix}$.
- Le vecteur en coordonnées homogènes : $\begin{pmatrix} x \\ y \\ z \\ 0 \end{pmatrix}$.

Transformation affine

Definition (Transformation affine)

- Une fonction de l'ensemble des points dans l'ensemble des points
- Une représentation matricielle

$$M = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Les transformations affines préservent les lignes droites.
- Les transformations affines préservent le parallélisme.

Composition de transformations

- Composition (ou concaténation) de transformations par application du produit matriciel.
- Le produit $M_n \dots M_1$ applique les transformations dans l'ordre M_1, \dots, M_n .
- Le produit est non-commutatif.
- Transformation inverse M^{-1} telle que

$$MM^{-1} = M^{-1}M = I = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{aligned}
 M & \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix} \\
 = & \begin{pmatrix} (m_{11} * P_x) + (m_{12} * P_y) + (m_{13} * P_z) + (m_{14} * 1) \\ (m_{21} * P_x) + (m_{22} * P_y) + (m_{23} * P_z) + (m_{24} * 1) \\ (m_{31} * P_x) + (m_{32} * P_y) + (m_{33} * P_z) + (m_{34} * 1) \\ (0 * P_x) + (0 * P_y) + (0 * P_z) + (1 * 1) \end{pmatrix} \\
 = & \begin{pmatrix} (m_{11} * P_x) + (m_{12} * P_y) + (m_{13} * P_z) + m_{14} \\ (m_{21} * P_x) + (m_{22} * P_y) + (m_{23} * P_z) + m_{24} \\ (m_{31} * P_x) + (m_{32} * P_y) + (m_{33} * P_z) + m_{34} \\ 1 \end{pmatrix}
 \end{aligned}$$

Translation

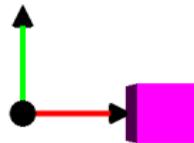
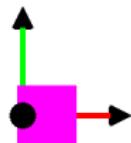
Definition (Translation)

Une translation est une transformation affine de la forme

$$T = \begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Translation selon un vecteur $\begin{pmatrix} d_x \\ d_y \\ d_z \\ 0 \end{pmatrix}$.
- Composition additive des translations.

Translation



$$\begin{aligned} T \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix} &= \begin{pmatrix} (1 * P_x) + (0 * P_y) + (0 * P_z) + (d_x * 1) \\ (0 * P_x) + (1 * P_y) + (0 * P_z) + (d_y * 1) \\ (0 * P_x) + (0 * P_y) + (1 * P_z) + (d_z * 1) \\ (0 * P_x) + (0 * P_y) + (0 * P_z) + (1 * 1) \end{pmatrix} \\ &= \begin{pmatrix} P_x + d_x \\ P_y + d_y \\ P_z + d_z \\ 1 \end{pmatrix} = \begin{pmatrix} d_x \\ d_y \\ d_z \\ 0 \end{pmatrix} + \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix} \end{aligned}$$

Translation inverse

L'inverse d'une translation $T = \begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$ est la

translation $T^{-1} = \begin{pmatrix} 1 & 0 & 0 & -d_x \\ 0 & 1 & 0 & -d_y \\ 0 & 0 & 1 & -d_z \\ 0 & 0 & 0 & 1 \end{pmatrix}.$

Homothétie

Definition (Homothétie)

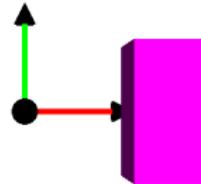
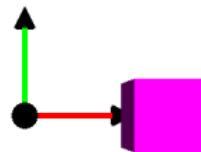
Une homothétie (ou mise à l'échelle) est une transformation affine

de la forme $H = \begin{pmatrix} h_x & 0 & 0 & 0 \\ 0 & h_y & 0 & 0 \\ 0 & 0 & h_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

- Homothérie par rapport à l'origine.
- Homothétie uniforme si $h_x = h_y = h_z$ sinon homothétie différentielle.
- Le point se rapproche de l'origine selon l'axe α si $h_\alpha < 1$ et sinon s'éloigne.
- Réflexion par rapport à l'axe α si $h_\alpha < 0$.
- Composition multiplicative des homothéties.

$$\begin{aligned} H \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix} &= \begin{pmatrix} (h_x * P_x) + (0 * P_y) + (0 * P_z) + (0 * 1) \\ (0 * P_x) + (h_y * P_y) + (0 * P_z) + (0 * 1) \\ (0 * P_x) + (0 * P_y) + (h_z * P_z) + (0 * 1) \\ (0 * P_x) + (0 * P_y) + (0 * P_z) + (1 * 1) \end{pmatrix} \\ &= \begin{pmatrix} h_x * P_x \\ h_y * P_y \\ h_z * P_z \\ 1 \end{pmatrix} \end{aligned}$$

Homothétie



Homothétie par rapport à un point

- Homothétie de rapports h_x, h_y, h_z par rapport à $\begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix}$:
- Translation du repère selon le vecteur $\begin{pmatrix} -P_x \\ -P_y \\ -P_z \\ 0 \end{pmatrix}$ par T ;
- Homothétie de rapports h_x, h_y, h_z par la matrice H ;
- Translation du repère selon le vecteur $\begin{pmatrix} P_x \\ P_y \\ P_z \\ 0 \end{pmatrix}$ par T^{-1} .

$$T^{-1}HT$$

$$= \begin{pmatrix} 1 & 0 & 0 & P_x \\ 0 & 1 & 0 & P_y \\ 0 & 0 & 1 & P_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} h_x & 0 & 0 & 0 \\ 0 & h_y & 0 & 0 \\ 0 & 0 & h_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} h_x & 0 & 0 & P_x(1 - h_x) \\ 0 & h_y & 0 & P_y(1 - h_y) \\ 0 & 0 & h_z & P_z(1 - h_z) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} h_x & 0 & 0 & P_x(1 - h_x) \\ 0 & h_y & 0 & P_y(1 - h_y) \\ 0 & 0 & h_z & P_z(1 - h_z) \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix} = \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix}$$

Cisaillement

Definition (Cisaillement selon un plan)

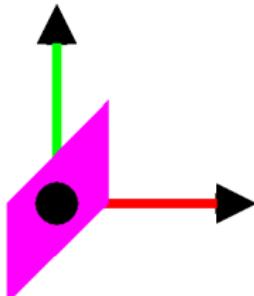
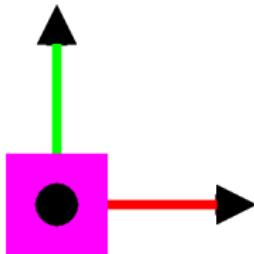
Une cisaillement est une transformation affine de la forme

$$C = \begin{pmatrix} 1 & c_x^y & c_x^z & 0 \\ c_y^x & 1 & c_y^z & 0 \\ c_z^x & c_z^y & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \text{ avec}$$

- Cisaillement selon le plan $(x, y) : c_z^x = c_y^x = c_z^y = c_x^y = 0$;
- Cisaillement selon le plan $(x, z) : c_y^x = c_z^x = c_y^z = c_x^z = 0$;
- Cisaillement selon le plan $(y, z) : c_x^y = c_z^y = c_x^z = c_y^z = 0$.

$$\begin{aligned}
 & C \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix} \\
 = & \begin{pmatrix} (1 * P_x) + (c_x^y * P_y) + (c_x^z * P_z) + (0 * 1) \\ (c_y^x * P_x) + (1 * P_y) + (c_y^z * P_z) + (0 * 1) \\ (c_z^x * P_x) + (c_z^y * P_y) + (1 * P_z) + (0 * 1) \\ (0 * P_x) + (0 * P_y) + (0 * P_z) + (1 * 1) \end{pmatrix} \\
 = & \begin{pmatrix} P_x + (c_x^y * P_y) + (c_x^z * P_z) \\ (c_y^x * P_x) + P_y + (c_y^z * P_z) \\ (c_z^x * P_x) + (c_z^y * P_y) + P_z \\ 1 \end{pmatrix}
 \end{aligned}$$

Cisaillement zy



Rotation

Definition (Rotation en 2D par rapport à l'origine)

Une rotation en 2D par rapport à l'origine selon un angle θ est une transformation affine de la forme $R_{2D}^o = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$

- Une valeur de θ négative opère une rotation dans le sens des aiguilles d'une montre.

Definition (Rotation par rapport à l'axe des z)

Une rotation d'axe z selon un angle θ est une transformation affine de la forme $R_z = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Definition (Rotation par rapport à l'axe des x)

Une rotation d'axe x selon un angle θ est une transformation affine

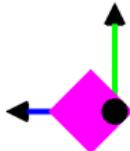
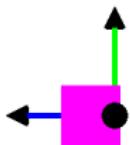
de la forme $R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Definition (Rotation par rapport à l'axe des y)

Une rotation d'axe y selon un angle θ est une transformation affine

de la forme $R_y = \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Rotation



Rotation autour d'un axe passant par l'origine selon un angle

Theorem (Euler)

Toute rotation ou combinaison de rotations autour d'un point est équivalente à une seule rotation selon un axe passant par ce point.

Definition (Rotation selon un vecteur normalisé et un angle)

vecteur normalisé $\begin{pmatrix} u_x \\ u_y \\ u_z \\ 0 \end{pmatrix}$ et angle θ ($c = \cos(\theta)$, $s = \sin(\theta)$).

$$\begin{pmatrix} (1 - c)u_x^2 + c & (1 - c)u_xu_y - su_z & (1 - c)u_xu_z + su_y & 0 \\ (1 - c)u_xu_y + su_z & (1 - c)u_y^2 + c & (1 - c)u_yu_z - su_x & 0 \\ (1 - c)u_xu_z - su_y & (1 - c)u_yu_z + su_x & (1 - c)u_z^2 + c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

En OpenGL

- identité : `glLoadIdentity()`
- translation : `glTranslate{fd}(d_x, d_y, d_z)`
- rotation : `glRotate{fd}(angle, u_x, u_y, u_z)`
- homothétie : `glScale{fd}(h_x, h_y, h_z)`
- multiplication par matrice 4x4 quelconque :
`glMultMatrix{fd}(* m)`
- La pile de matrice : `glPushMatrix()` et `glPopMatrix()`.

Changement de repère par transformation

- P_i un point dans un repère \mathcal{R}_i , $1 \leq i \leq n$;
- $M_{i \leftarrow j}$ matrice de changement du repère \mathcal{R}_j dans le repère \mathcal{R}_i ;
- $P_i = M_{i \leftarrow j} P_j$ et $P_j = M_{j \leftarrow k} P_k$;
- donc $P_i = M_{i \leftarrow j} \cdot M_{j \leftarrow k} P_k = M_{i \leftarrow k} P_k$;
- $M_{1 \leftarrow n} = M_{1 \leftarrow 2} \dots M_{(n-1) \leftarrow n}$.
- Inversion de l'ordre par rapport aux applications des transformations sur un point.

Transformations et repères

- M_2 une transformation dans un repère \mathcal{R}_2 et $M_{1 \leftarrow 2}$ un changement de repère de \mathcal{R}_2 vers \mathcal{R}_1 ;
- P_1 (resp. P_2) un point dans un repère \mathcal{R}_1 (resp. \mathcal{R}_2) ;
- Que vaut M_1 dans le repère \mathcal{R}_1 telle que $M_1 P_1 = M_{1 \leftarrow 2} \cdot M_2 P_2$?
- Comme $P_1 = M_{1 \leftarrow 2} P_2$ donc $M_1 \cdot M_{1 \leftarrow 2} P_2 = M_{1 \leftarrow 2} \cdot M_2 P_2$;
- donc $M_1 \cdot M_{1 \leftarrow 2} = M_{1 \leftarrow 2} \cdot M_2$
- donc $M_1 \cdot M_{1 \leftarrow 2} \cdot M_{1 \leftarrow 2}^{-1} = M_{1 \leftarrow 2} \cdot M_2 \cdot M_{1 \leftarrow 2}^{-1}$
- donc $M_1 = M_{1 \leftarrow 2} \cdot M_2 \cdot M_{1 \leftarrow 2}^{-1}$

Plan

- 1 Introduction
- 2 Eléments de mathématiques
- 3 Les transformations en action
- 4 Les algorithmes du rendu

Modélisation en OpenGL

- Transformations affines «appliquées aux sommets».
- Activation de la matrice de modélisation/visualisation :
`glMatrixMode(GL_MODELVIEW)`
- Rotation selon l'axe des x suivie d'une translation selon l'axe des y ?
 - `vrx = 1.0f; vty = 1.0f; angle = 45;`
`glRotatef(angle, vrx, 0.0f, 0.0f);`
`glTranslatef(0.0f, vty, 0.0f);`
 - Non! Rotation selon l'axe des x suivie d'une translation selon le nouvel axe des y .
- Changements de repère : translation selon l'axe des y suivie d'une rotation selon l'axe des x .
 - `glTranslatef(0.0f, vty, 0.0f);`
`glRotatef(angle, vrx, 0.0f, 0.0f);`
 - ...le résultat attendu.

24 points, 6 faces, 1 cube

Cube 2x2x2 centré en (0,0,0).

```
glBegin(GL_QUADS);  
/* face avant ccw */  
glVertex3f(-1.0f, -1.0f, 1.0f);  
glVertex3f( 1.0f, -1.0f, 1.0f);  
glVertex3f( 1.0f, 1.0f, 1.0f);  
glVertex3f(-1.0f, 1.0f, 1.0f);  
  
/* face arrière ccw */  
glVertex3f(-1.0f, -1.0f, -1.0f);  
glVertex3f(-1.0f, 1.0f, -1.0f);  
glVertex3f( 1.0f, 1.0f, -1.0f);  
glVertex3f( 1.0f, -1.0f, -1.0f);
```

```
/* face dessus ccw */  
glVertex3f(-1.0f, 1.0f, -1.0f);  
glVertex3f(-1.0f, 1.0f, 1.0f);  
glVertex3f( 1.0f, 1.0f, 1.0f);  
glVertex3f( 1.0f, 1.0f, -1.0f);  
  
/* face dessous ccw */  
glVertex3f(-1.0f, -1.0f, -1.0f);  
glVertex3f( 1.0f, -1.0f, -1.0f);  
glVertex3f( 1.0f, -1.0f, 1.0f);  
glVertex3f(-1.0f, -1.0f, 1.0f);
```

```
/* face de droite ccw */  
glVertex3f( 1.0f, -1.0f, -1.0f);  
glVertex3f( 1.0f, 1.0f, -1.0f);  
glVertex3f( 1.0f, 1.0f, 1.0f);  
glVertex3f( 1.0f, -1.0f, 1.0f);  
  
/* face de gauche ccw */  
glVertex3f(-1.0f, -1.0f, -1.0f);  
glVertex3f(-1.0f, -1.0f, 1.0f);  
glVertex3f(-1.0f, 1.0f, 1.0f);  
glVertex3f(-1.0f, 1.0f, -1.0f);  
  
glEnd();
```

Visusalisation

- Placer un point de vue sur une scène modélisée :
 - un «œil» : un point,
 - un «look» : un point,
 - une orientation : un vecteur non parallèle à la droite œil/look



- Transformations affines «appliquées aux repères» : une translation + une rotation.

Visualisation en OpenGL

- Activation de la matrice de modélisation/visualisation :
`glMatrixMode(GL_MODELVIEW)`
- Initialement : œil en $(0, 0, 0)$, look en $(0, 0, -\infty)$, axe des y .
- Deux possibilités :
 - «A la main» avec `glRotate*` et `glTranslate*`
 - La fonction GLU `gluLookAt`
- `gluLookAt(o_x, o_y, o_z, l_x, l_y, l_z, up_x, up_y, up_z)`
 - $oeil = (o_x, o_y, o_z)$: position de l'œil,
 - $look = (l_x, l_y, l_z)$: position du look,
 - $up = (up_x, up_y, up_z)$: orientation non parallèle à œil/look.
- Définition d'un nouveau repère : le «repère de l'œil».

Le repère de l'œil

- repère de l'œil (**u**, **v**, **n**)
- vecteur **n** selon la direction «look» vers «œil» ;
- vecteur **u** perpendiculaire aux vecteurs **n** et **up** ;
- vecteur **v** perpendiculaire aux vecteurs **n** et **u**.
- $\mathbf{n} = \text{oeil} - \text{look} = (n_x, n_y, n_z)$
- $\mathbf{u} = \mathbf{up} \times \mathbf{n} = (u_x, u_y, u_z)$
- $\mathbf{v} = \mathbf{n} \times \mathbf{u} = (v_x, v_y, v_z)$

La matrice de visualisation V

Pour \mathbf{u} , \mathbf{v} , \mathbf{n} normés,

$$V = \begin{pmatrix} u_x & u_y & u_z & d_x \\ v_x & v_y & v_z & d_y \\ n_x & n_y & n_z & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

avec $\mathbf{o} = ((0, 0, 0) - \text{oeil})$ et $(d_x, d_y, d_z) = (\mathbf{o} \cdot \mathbf{u}, \mathbf{o} \cdot \mathbf{v}, \mathbf{o} \cdot \mathbf{n})$

- $V \begin{pmatrix} o_x \\ o_y \\ o_z \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$

- $V \begin{pmatrix} u_x \\ u_y \\ u_z \\ 0 \end{pmatrix} = \mathbf{i}$, $V \begin{pmatrix} v_x \\ v_y \\ v_z \\ 0 \end{pmatrix} = \mathbf{j}$ et $V \begin{pmatrix} n_x \\ n_y \\ n_z \\ 0 \end{pmatrix} = \mathbf{k}$.

Exemple : une simple translation

- `glTranslatef(-5.0f,-2.0f,-10.0f);` avec `gluLookAt` ?
- `gluLookAt(5.0f,2.0f,10.0f, /* oeil */
0.0f,0.0f,0.0f, /* look */
0.0f,1.0f,0.0f); /* orientation */`
- L'œil ne regarde plus dans la bonne direction.
- `gluLookAt(5.0f,2.0f,10.0f, /* oeil */
5.0f,2.0f,0.0f, /* look */
0.0f,1.0f,0.0f); /* orientation */`

- $\mathbf{n} = \text{oeil} - \text{look}$ normalisé en $(\frac{5-5}{\|\mathbf{n}\|}, \frac{2-2}{\|\mathbf{n}\|}, \frac{10-0}{\|\mathbf{n}\|}) = (0, 0, 1)$;
- $\mathbf{u} = \mathbf{up} \times \mathbf{n}$ normalisé en
 $(\frac{1*10-0*0}{\|\mathbf{u}\|}, \frac{0*0-0*10}{\|\mathbf{u}\|}, \frac{0*0-1*0}{\|\mathbf{u}\|}) = (1, 0, 0)$;
- $\mathbf{v} = \mathbf{n} \times \mathbf{u}$ normalisé en
 $(\frac{0*0-10*0}{\|\mathbf{v}\|}, \frac{10*10-0*0}{\|\mathbf{v}\|}, \frac{0*0-0*0}{\|\mathbf{v}\|}) = (0, 1, 0)$
- et avec $\mathbf{o} = (-5, -2, -10)$ et
 $(d_x, d_y, d_z) = (\mathbf{o} \cdot \mathbf{u}, \mathbf{o} \cdot \mathbf{v}, \mathbf{o} \cdot \mathbf{n}) = (-5, -2, -10)\dots$
-

$$\begin{pmatrix} 1 & 0 & 0 & -5 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & -10 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

i.e. la matrice de la translation selon $(-5, -2, -10)$.

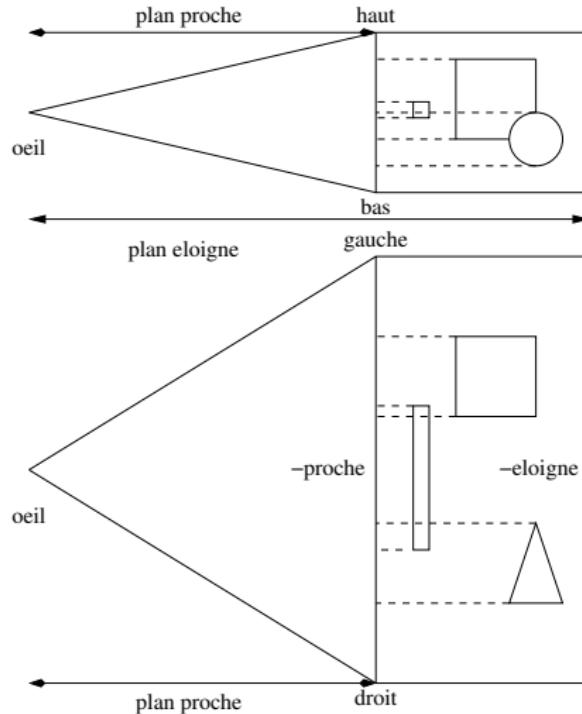
Volume visionné

- 6 plans déterminent le volume visionné :
 - les plans proche et éloigné,
 - les plans haut et bas,
 - les plans gauche et droit.
- Découpage des objets traversés par ces plans.
- Elimination des objets en dehors du volume visionné...
 - y compris ceux entre l'œil et le plan proche.
- Normalisation du volume visionné en un volume $2 \times 2 \times 2$ centré sur l'origine.

Projection

- 2 types de projection :
 - la projection orthogonale,
 - la projection conique.
- Activation en OpenGL de la matrice de projection :
`glMatrixMode(GL_PROJECTION);`

Volume visionné pour projection orthogonale



Projection orthogonale

- Projection «cavalière»
- Préservation des angles, des droites et des parallèles.
- Pas de diminution de la taille avec l'éloignement.
- Œil à l'infini.
- `glOrtho(gauche,droit,bas,haut,eloigne,proche)`
- $gauche \leq x \leq droit, bas \leq y \leq haut, proche \leq -z \leq eloigne$

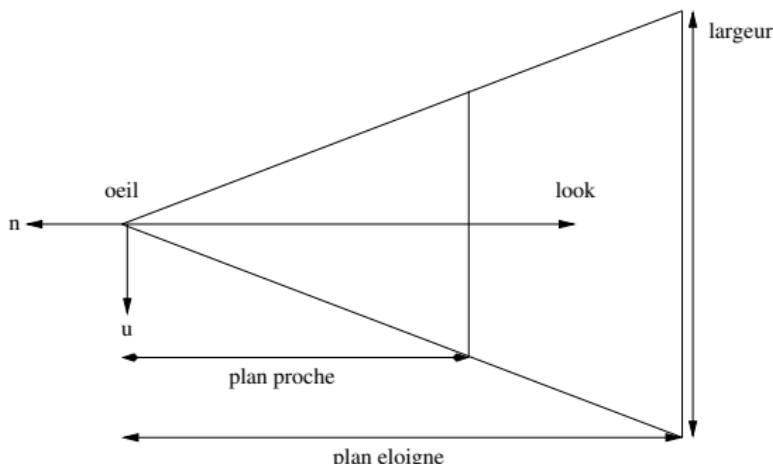
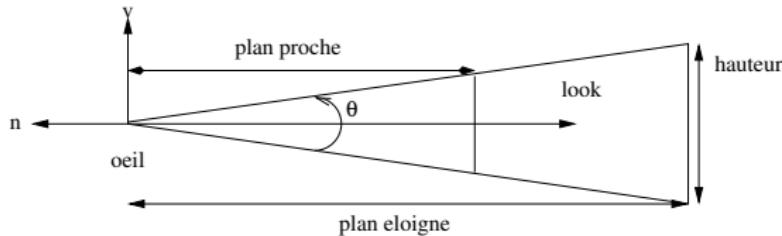
Transformation pour projection orthogonale

- Normalisation en un volume $2 \times 2 \times 2$ centré par une homothétie et une translation.
- Hypothèse : projection selon l'axe des z.

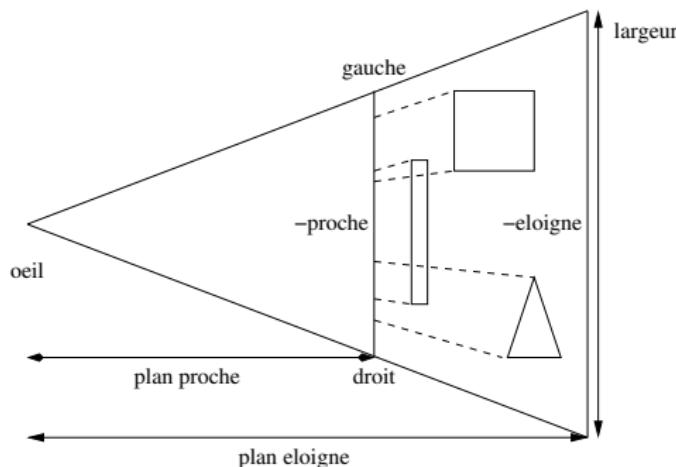
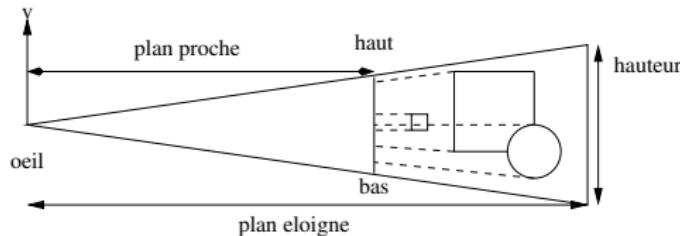
- $O = \begin{pmatrix} \frac{2}{\text{droit}-\text{gauche}} & 0 & 0 & -\frac{\text{droit}+\text{gauche}}{\text{droit}-\text{gauche}} \\ 0 & \frac{2}{\text{haut}-\text{bas}} & 0 & -\frac{\text{haut}+\text{bas}}{\text{haut}-\text{bas}} \\ 0 & 0 & -\frac{2}{\text{proche}-\text{eloigne}} & -\frac{\text{proche}+\text{eloigne}}{\text{proche}-\text{eloigne}} \\ 0 & 0 & 0 & 1 \end{pmatrix}$

- $O \begin{pmatrix} \text{droit} \\ \text{haut} \\ -\text{proche} \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, O \begin{pmatrix} \text{gauche} \\ \text{bas} \\ -\text{eloigne} \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ -1 \\ -1 \\ 1 \end{pmatrix}$

Volume visionné pour projection conique



Projection conique



Projection conique

- Effet de perspective.
- Préservation des droites.
- Diminution de la taille avec l'éloignement.
- $gauche \leq \frac{proche*x}{-z} \leq droit, bas \leq \frac{proche*y}{-z} \leq haut,$
 $proche \leq -z \leq eloigne$
- Perspective = Transformation pour projection conique + Projection orthogonale.

Transformation pour projection conique

- $P = \begin{pmatrix} \frac{2*\text{proche}}{\text{droit}-\text{gauche}} & 0 & \frac{\text{droit}+\text{gauche}}{\text{droit}-\text{gauche}} & 0 \\ 0 & \frac{2*\text{proche}}{\text{haut}-\text{bas}} & \frac{\text{haut}+\text{bas}}{\text{haut}-\text{bas}} & 0 \\ 0 & 0 & -\frac{\text{proche}+\text{eloigne}}{\text{proche}-\text{eloigne}} & -\frac{2*\text{proche}*\text{eloigne}}{\text{proche}-\text{eloigne}} \\ 0 & 0 & -1 & 0 \end{pmatrix}$

- $P \begin{pmatrix} \text{droit} \\ \text{haut} \\ -\text{proche} \\ 1 \end{pmatrix} = \begin{pmatrix} \text{proche} \\ \text{proche} \\ \text{proche} \\ \text{proche} \end{pmatrix}$

- `glFrustum(gauche, droit, bas, haut, proche, eloigne)`
- `gluPerspective(θ , largeur/hauteur, proche, eloigne)`
- $\text{haut} = \text{proche} * \tan(\theta/2)$, $\text{bas} = -\text{proche} * \tan(\theta/2)$,
 $\text{droit} = (\text{largeur}/\text{hauteur}) * \text{haut}$ et
 $\text{gauche} = (\text{largeur}/\text{hauteur}) * \text{bas}$.

Projection et faces cachées

- Perspective division : $\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$ homogénéisé en $\begin{pmatrix} \frac{x}{w} \\ \frac{y}{w} \\ \frac{z}{w} \\ \frac{w}{w} \end{pmatrix}$.
- Projection par élimination de la coordonnée en z
- Coordonnée en z : pseudo-distance pour élimination des faces cachées.
- z-buffer : tampon de profondeur

Elimination des faces arrières des polygones

- Face avant : polygone tracé dans le sens contraire des aiguilles d'une montre (ccw).
- Le «culling» : élimination des faces arrières (cachées)
- Activation : `glEnable(GL_CULL_FACE)`
- Déactivation : `glDisable(GL_CULL_FACE)`
- Problème avec les objects non clos : une face arrière peut être visible !

Transformation de cadrage

- Du résultat de la projection à un rectangle d'affichage dans la fenêtre.
- Par défaut : le rectangle d'affichage est toute la fenêtre avec $(0,0)$ en bas à gauche.
- `glViewport(GLint ori_x, GLint ori_y, GLsizei pixels_x, GLsizei pixels_y)`
- (ori_x, ori_y) est l'origine du cadrage en pixels
- $pixels_x$ est la largeur et $pixels_y$ est la hauteur du rectangle d'affichage.
- Proportions conservées si $hauteur/largeur = pixels_y/pixels_x$.

La gestion de la fenêtre lors de l'initialisation

- Initialisation de la librairie GLUT :

```
void glutInit(int *argc, char **argv);
```

- Sélectionne les buffers

```
void glutInitDisplayMode(unsigned int mode);
```

- GLUT_RGBA/GLUT_INDEX : masque pour le mode RGB/index des couleurs
- GLUT_SINGLE/GLUT_DOUBLE : masque pour le mode simple/double buffer(s)
- GLUT_DEPTH : masque pour le mode avec z-buffer
- GLUT_ALPHA : masque pour le mode transparence

La gestion de la fenêtre lors de l'initialisation (suite)

- Initialisation de la taille de la fenêtre :

```
void glutInitWindowSize(int largeur, int  
hauteur);  
largeur et hauteur en pixels
```

- Initialisation de la position de la fenêtre :

```
void glutInitWindowPosition(int x, int y);  
abscisse (x, de haut en bas) et ordonnée (y) en pixels écran  
du coin supérieur gauche de la fenêtre système.
```

- Création de la fenêtre :

```
int glutCreateWindow(char *name);
```

- Enregistrement de la fonction de visualisation/modélisation :

```
void glutDisplayFunc(void (*func)());
```

La gestion de la fenêtre lors de l'initialisation (suite)

- Enregistrement de la fonction de redimensionnement :

```
void glutReshapeFunc(void (*func)(int largeur,  
int hauteur));
```

largeur et hauteur en pixels

Appelé lors du premier affichage de la fenêtre système.

- Enregistrement de la fonction de comportement en l'absence d'événement : `void glutIdleFunc(void (*func)());`

- Enregistrement de la fonction de capture du clavier :

```
void glutKeyboardFunc(void (*func)(unsigned char  
touche, int x, int y));
```

touche pressée, x et y coordonnées souris en pixels fenetre

La gestion de la fenêtre lors de l'initialisation (fin)

- Enregistrement de la fonction de capture du clavier (touches spéciales) :

```
void glutSpecialFunc(void (*func)(int touche, int  
x, int y));
```

touche pressée, x et y coordonnées souris en pixels fenêtre.

- Initialisation de la matrice de projection et du cadrage.
- Définition la couleur de fond : `void glClearColor(GLfloat red, GLfloat green, GLfloat blue, GLfloat alpha);`
- La boucle GLUT prend la main : `void glutMainLoop();`

La gestion de l'écran lors de la visualisation/modelisation

- Effaçage du buffer écran

```
void glClear(unsigned int mode);
```

- GL_COLOR_BUFFER_BIT : masque pour le buffer écran
- GL_DEPTH_BUFFER_BIT : masque pour le z-buffer

- Echange des deux buffers écrans glutSwapBuffers();

La gestion de l'écran lors du redimensionnement

- Modifier le cadrage avec `glViewport`
- Modifier le ratio de perspective dans `gluPerspective` pour conserver les formes.

Plan

- 1 Introduction
- 2 Eléments de mathématiques
- 3 Les transformations en action
- 4 **Les algorithmes du rendu**

Les couleurs : le système RVB

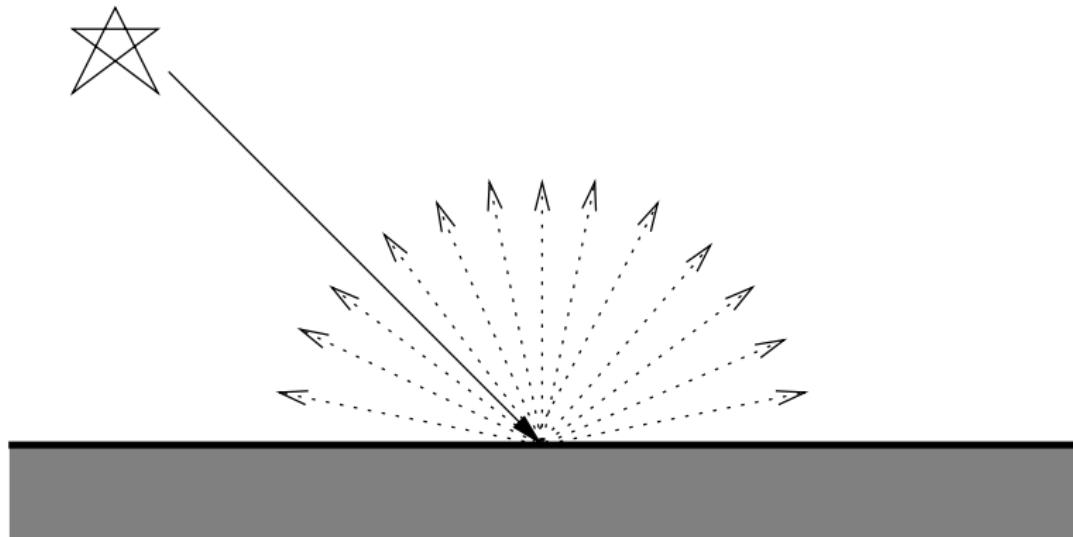
- 4 composantes : Rouge, Vert, Bleu + Alpha (α) ;
- Alpha : transparence (ou mélange des couleurs) par défaut à 1.0 (i.e. opacité) ;

couleur	R	V	B
blanc	1.0	1.0	1.0
noir	0.0	0.0	0.0
rouge	1.0	0.0	0.0
vert	0.0	1.0	0.0
bleu	0.0	0.0	1.0
cyan	0.0	1.0	1.0
jaune	1.0	1.0	0.0
magenta	1.0	0.0	1.0

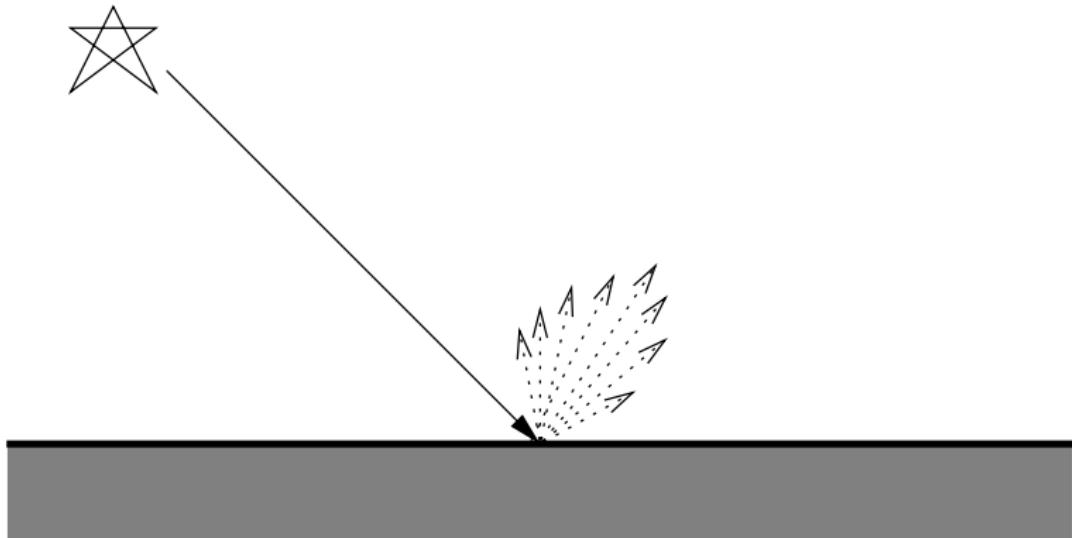
Modèle de Phong de la lumière

- Modèle de réflexion
- Source de lumière ponctuelle
- 3 composantes à la lumière : rouge, vert, bleu
- principe de superposition
- 3 types de lumières :
 - Lumière ambiante
 - Lumière diffuse
 - Lumière spéculaire (réflexion miroir)
- Propriété émissive de la surface

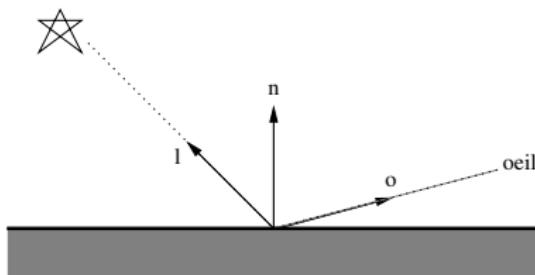
Lumière diffuse



Lumière spéculaire

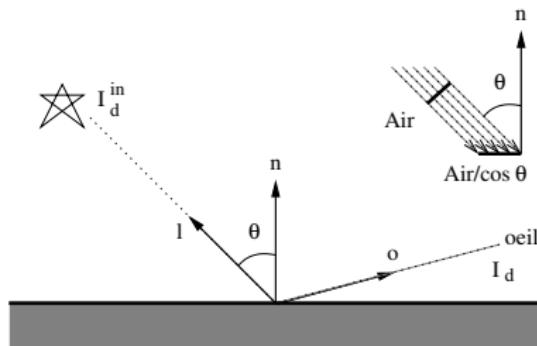


Les vecteurs pour le modèle de Phong de la lumière



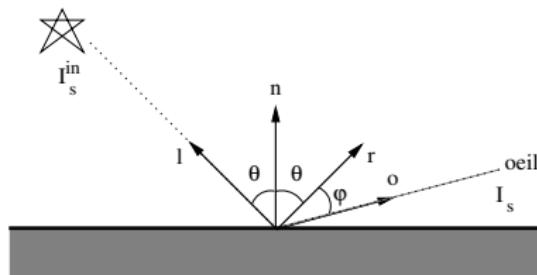
- Trois vecteurs unitaires au point d'impact du faisceau lumineux :
 - vecteur **I** dirigé vers la source lumineuse
 - vecteur **n** normal à la surface
 - vecteur **o** vers l'œil
- Lumière directionnelle (à l'infini, **I** constant) ou positionnelle
- Œil non local (à l'infini, **o** constant) ou local

Lumière diffuse dans le modèle de Phong



- I_d^{in} intensité de la source diffuse
- ρ_d coefficient de réflexivité diffuse
- θ angle entre \mathbf{l} et \mathbf{n}
- I_d intensité de la lumière diffuse perçue
- $I_d = \rho_d * I_d^{in} * \cos \theta = \rho_d * I_d^{in} * (\mathbf{l} \cdot \mathbf{n})$

Lumière spéculaire dans le modèle de Phong



- I_s^{in} intensité de la source spéculaire
- ρ_s coefficient de réflexivité spéculaire
- θ angle entre \mathbf{l} et \mathbf{n} et entre \mathbf{n} et $\mathbf{r} = 2 * ((\mathbf{l} \cdot \mathbf{n}) \cdot \mathbf{n}) - 1$
- φ angle entre \mathbf{r} et \mathbf{o} et f un facteur arbitraire de renforcement
- I_s intensité de la lumière spéculaire perçue
- $I_s = \rho_s * I_s^{in} * (\cos \varphi)^f = \rho_s * I_s^{in} * (\mathbf{r} \cdot \mathbf{o})^f$

Lumière et couleur

- I_a^{in} intensité de la lumière ambiente
- ρ_a coefficient de réflexivité de la lumière ambiente
- I_e intensité de la propriété émissive de la surface
- Pour chaque couleur,

$$\begin{aligned} I &= I_a + I_d + I_s + I_e \\ &= (\rho_a * I_a^{in}) + (\rho_d * I_d^{in} * (\mathbf{l} \cdot \mathbf{n})) + (\rho_s * I_s^{in} * (\mathbf{r} \cdot \mathbf{o})^f) + I_e \end{aligned}$$

- Source de lumière multiple :

$$I = (\rho_a * I_a^{in}) + (\rho_d * \sum_i I_d^{in,i} * (\mathbf{l}_i \cdot \mathbf{n})) + (\rho_s * \sum_i I_s^{in,i} * (\mathbf{r} \cdot \mathbf{o}_i)^f) + I_e$$

- Face non exposée pour une lumière et un œil : $\mathbf{l} \cdot \mathbf{n} < 0$ ou $\mathbf{r} \cdot \mathbf{o} < 0$.

Deux systèmes pour les lumières en OpenGL

- Une lumière omnidirectionnelle :
 - système par défaut,
 - état couleur défini par `glColor*`.
- Selon le modèle de Phong :
 - dégradé (ou shading) selon l'algorithme de Gouraud calculé avec la matrice de projection ;
 - activé/désactivé avec `GL_LIGHTING` ;
 - propriétés des différentes sources lumières ;
 - propriétés des matériaux pour les surfaces ;
 - normale associée au point par somme vectorielle des normales des surfaces définissant le point ;
 - application de `gluLookAt` sur la matrice de modélisation et visualisation.
 - Œil non local par défaut ; œil local avec :
`glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE)`

Lumières selon Phong en OpenGL

- une lumière ambiante par
`glLightModel*(GL_LIGHT_MODEL_AMBIENT, Couleurs);`
- 8 lumières (de 0 à 7) indépendantes
- $i^{\text{ème}}$ allumée/éteinte par
 `glEnable(GL_LIGHTi)/glDisable(GL_LIGHTi).`
- Position d'une lumière en coordonnées homogènes
 (x, y, z, w) :
 - $w=1$: (x, y, z) désigne un point
 - $w=0$: (x, y, z) désigne une direction (un vecteur)
- `glLight*(Lumiere, Etat, Params)` définit les paramètres d'une lumière pour un état.
- Etat : GL_POSITION, GL_AMBIENT, GL_DIFFUSE et GL_SPECULAR
- Les sources lumineuses peuvent se déplacer.

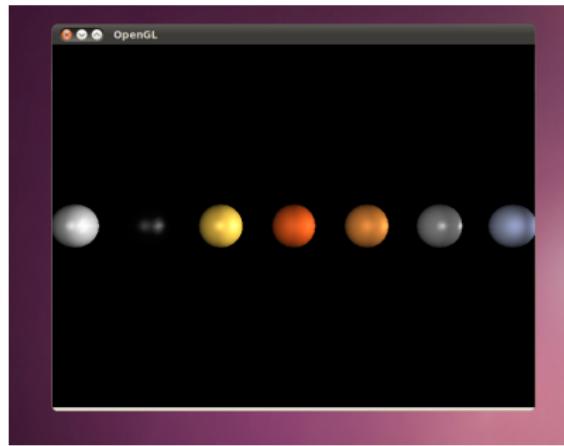
Atténuation et direction des lumières

- Atténuation de l'intensité de la lumière i par `glLightf(GL_LIGHTi , Etat, k)`
 - Etat : `GL_CONSTANT_ATTENUATION`, `GL_LINEAR_ATTENUATION` et `GL_QUADRATIC_ATTENUATION`,
 - k : facteur d'atténuation.
- Direction de la lumière i par `glLight*(GL_LIGHTi , GL_SPOT_DIRECTION, Param)`
- Cône de lumière i par `glLightf(GL_LIGHTi , GL_SPOT_CUTOFF, θ)`

Matériaux selon Phong en OpenGL

- propriétés de réflexivité spécifiées par `glMaterial*`(Etat1, Etat2, Param)
 - Etat1 : `GL_FRONT`, `GL_BACK`, `GL_FRONT_AND_BACK`;
 - Etat2 : `GL_AMBIENT`, `GL_DIFFUSE`, `GL_SPECULAR` et `GL_EMISSION`.
- propriété de brillance (ou shininess) spécifiée par `glMaterial`(Etat, `GL_SHININESS`, float f)
 - Etat : `GL_FRONT`, `GL_BACK`, `GL_FRONT_AND_BACK`;
 - f : le coefficient de brillance entre 0.0 (par défaut) et 128.0.

Quelques matériaux



	R	V	B
Or (shininess = 51.2)	0.24725	0.1995	0.0745
ambiante	0.75164	0.60648	0.22648
diffuse	0.628281	0.555802	0.366065
spéculaire			

Au final 3 modes pour l'illumination d'une scène

- Lumière omnidirectionnelle
 - les couleurs par `glColor*`
 - pas de gestion de la lumière
- Lumières et couleurs
 - les couleurs par `glColor*`
 - activation de `GL_COLOR_MATERIAL`
 - gestion des lumières
- Lumières et matériaux
 - rendu des couleurs par les matériaux
 - gestion des lumières.

Mélange des couleurs

- Blending : effets de transparence.
- Mélange des couleurs au niveau du buffer chromatique.
- `glEnable(GL_BLEND)/glDisable(GL_BLEND)` pour activer/désactiver le mélange.
- Transparence mais pas seulement :
 - le lissage.
 - les effets de brouillard.
 - Permet l'animation 2D.

Mélange des couleurs

- Prise en compte des paramètres chromatiques de la source (l'objet rendu) : $(C_s^r, C_s^v, C_s^b, C_s^a)$
- Prise en compte des paramètres chromatiques de la destination (le pixel du buffer chromatique) : $(C_d^r, C_d^v, C_d^b, C_d^a)$
- Prise en compte d'un facteur de blending de la source : $(B_s^r, B_s^v, B_s^b, B_s^a)$
- Prise en compte d'un facteur de blending de la destination : $(B_d^r, B_d^v, B_d^b, B_d^a)$

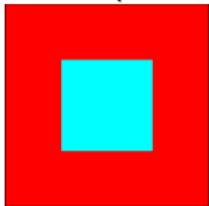
Mélange des couleurs

- `glBlendFunc(ParamSource, ParamDest)` pour choisir les paramètres du calcul du mélange pour chaque composante RVBA :
 - `GL_ZERO` : $B_*^* = 0$
 - `GL_ONE` : $B_*^* = 1$
 - `GL_SRC_ALPHA` : $B_*^* = C_a^s$
 - `GL_ONE_MINUS_SRC_ALPHA` : $B_*^* = 1 - C_a^s$
- `glBlendEquation(Func)` pour choisir la fonction de composition des paramètres chromatiques et des facteurs de blending $Func(C_s^*, B_s^*, C_d^*, B_d^*)$
 - `GL_FUNC_ADD` : $Func = C_s^* * B_s^* + C_d^* * B_d^*$ (fonction par défaut)

Transparence et opacité

- Transparent : $A = 0.0$ et opaque : $A = 1.0$
- Mélange de deux images avec facteur linéaire (selon l'alpha C_s^a de la source) :
 - `glBlendFunc(GL_ONE, GL_ZERO)`
 - Rendu de la première image (le fond est ignoré par `GL_ZERO`)
 - `glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)`
 - Rendu de la seconde image : $C^* = C_s^* * C_s^a + C_d^* * (1 - C_s^a)$
- Sujet (avec fond invisible) pour l'animation 2D
 - alpha $C_s^a = 0.0$ de la source pour le fond
 - alpha $C_s^a = 1.0$ de la source pour le sujet
- Transparence d'objets 3D
 - Rendu des objets opaques avec tampon de profondeur en lecture/écriture : `glDepthMask(GL_TRUE)`
 - Rendu des objets transparents avec tampon de profondeur en lecture seule : `glDepthMask(GL_FALSE)`

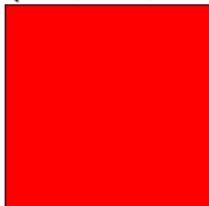
- Des : (1.0f, 0.0f, 0.0f, 1.0f) / Src : (0.0f, 1.0f, 1.0f, 0.5f)



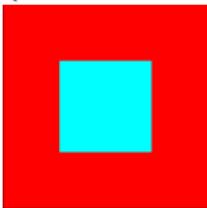
- Des : (1.0f, 0.0f, 0.0f, 1.0f) / Src : (0.0f, 1.0f, 1.0f, 0.5f)
 $glBlendFunc(GL_ONE, GL_ZERO)$

$$C_s^* * B_s^* + C_d^* * B_d^* =$$

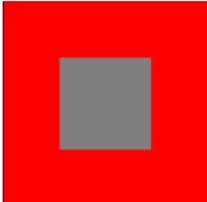
$$(0.0, 1.0, 1.0) * 1.0 + (1.0, 0.0, 0.0) * 0.0 = (0.0, 1.0, 1.0)$$



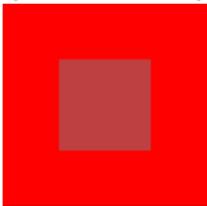
- Des : (1.0f, 0.0f, 0.0f, 1.0f) / Src : (0.0f, 1.0f, 1.0f, 0.5f)
 $glBlendFunc(GL_ZERO, GL_ONE)$
 $C_s^* * B_s^* + C_d^* * B_d^* = (0.0, 1.0, 1.0) * 0 + (1.0, 0.0, 0.0) * 1.0 = (0.0, 1.0, 1.0)$



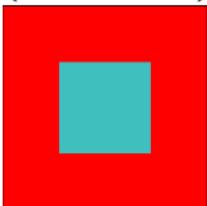
- Des : (1.0f, 0.0f, 0.0f, 1.0f) / Src : (0.0f, 1.0f, 1.0f, 0.5f)
 $glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)$
 $C_s^* * B_s^* + C_d^* * B_d^* = (0.0, 1.0, 1.0) * 0.5 + (1.0, 0.0, 0.0) * (1.0 - 0.5) = (0.5, 0.5, 0.5)$



- Des : (1.0f, 0.0f, 0.0f, 1.0f) / Src : (0.0f, 1.0f, 1.0f, 0.25f)
 $glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)$
 $C_s^* * B_s^* + C_d^* * B_d^* = (0.0, 1.0, 1.0) * 0.25 + (1.0, 0.0, 0.0) * (1.0 - 0.25) = (0.75, 0.25, 0.25)$



- Des : (1.0f, 0.0f, 0.0f, 1.0f) / Src : (0.0f, 1.0f, 1.0f, 0.75f)
 $glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)$
 $C_s^* * B_s^* + C_d^* * B_d^* = (0.0, 1.0, 1.0) * 0.75 + (1.0, 0.0, 0.0) * (1.0 - 0.75) = (0.25, 0.75, 0.75)$



Placage de texture



- Appliquer à une surface un motif non élémentaire
- ...sans en payer le coût algorithmique.
- Enrichir le rendu de détails macroscopiques.
- À base d'images bitmap.
- À la base de l'animation 2D.

Placage de texture en OpenGL

- Initialisation avec `glGenTextures`
- Création des index des textures
 - Déclaration d'un tableau de n textures :
`GLuint textures[n] ;`
 - Activation de la n^{eme} texture :
`glBindTexture` avec `GL_TEXTURE_2D` et `textures[n]` ;
 - Paramétrisation de la texture : `glTexParametri` avec
 - `GL_TEXTURE_2D`
 - soit `GL_TEXTURE_WRAP_S` ou `GL_TEXTURE_WRAP_T` associé à `GL_CLAMP` ou `GL_CLAMP_EDGE` ou `GL_REPEAT` ou `GL_MIRRORED_REPEAT`
 - soit `GL_TEXTURE_MAG_FILTER` associé à `GL_LINEAR` ou `GL_NEAREST`
 - soit `GL_TEXTURE_MIN_FILTER` associé à `GL_LINEAR` ou `GL_NEAREST` ou « `mipmap` »
- Envoie de la texture : `glTexImage2D` avec `GL_TEXTURE_2D`, `GL_RGB` (ou `GL_RGBA`) et `GL_UNSIGNED_BYTE`.

Placage de texture en OpenGL

- Association des textures aux surfaces lors du rendu :
 - activée/désactivée avec `GL_TEXTURE_2D`
 - `glTexEnvf` avec `GL_TEXTURE_ENV_MODE` pour choisir le type de placage : `GL_REPLACE` pour ignorer la couleur du support ou `GL_MODULATE` pour en tenir compte (par défaut).
 - Activation de la n^{eme} texture :
`glBindTexture` avec `GL_TEXTURE_2D` et `textures[n]` ;
 - Association à chaque sommet d'une coordonnée de texture avec `glTexCoord2f` (paramètres entre 0 et 1).