Plan du cours

1. Introduction
2. Modélisation d'objets 3D
3. Introduction à Povray
4. Rendu par lancer de rayons
5. Rendu temps réel
6. Introduction à OpenGL
7. Transformations géométriques
8. Modèles d'éclairage local

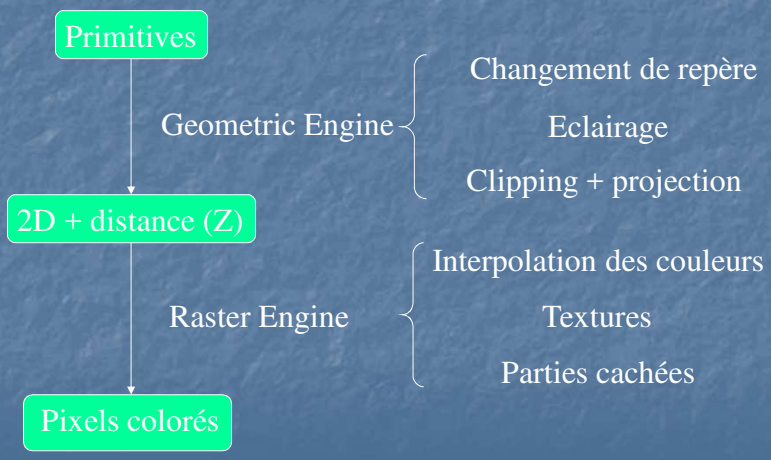
Rendu temps réel

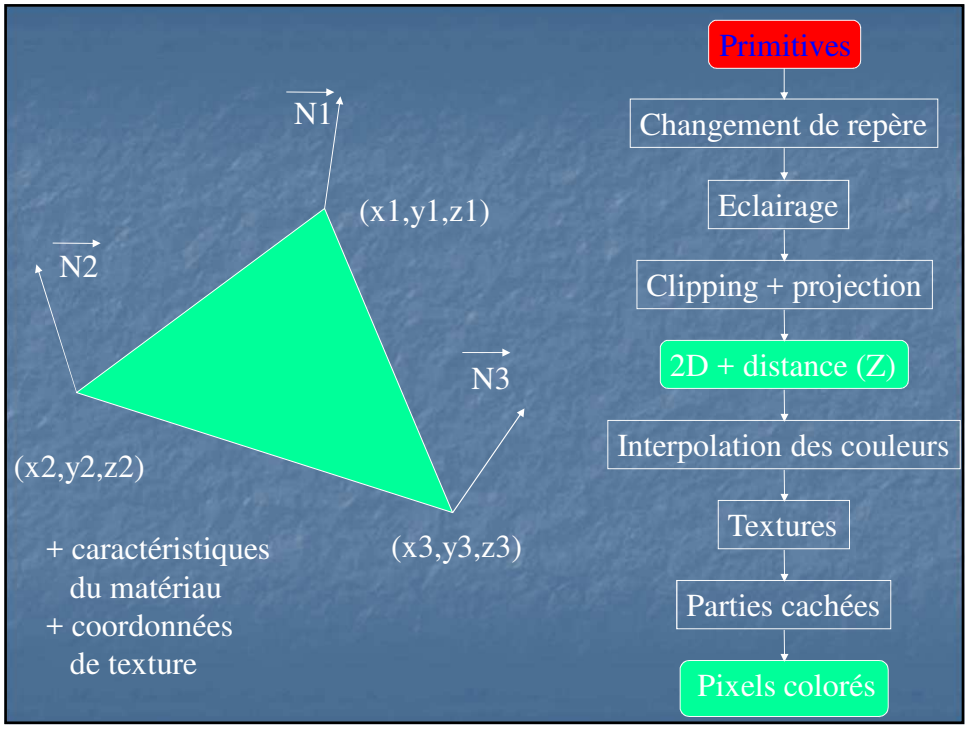
- Affichage temps réel :
  - Fréquence d'affichage des images > 25 images par seconde
    - Limite perceptive pour la perception de saccades
- Rendu temps réel
  - Calcul minimum de 25 images par seconde
  - Réduction de la fatigue visuelle :
    - ~60 images par seconde
  - Utilisation du pipeline de rendu

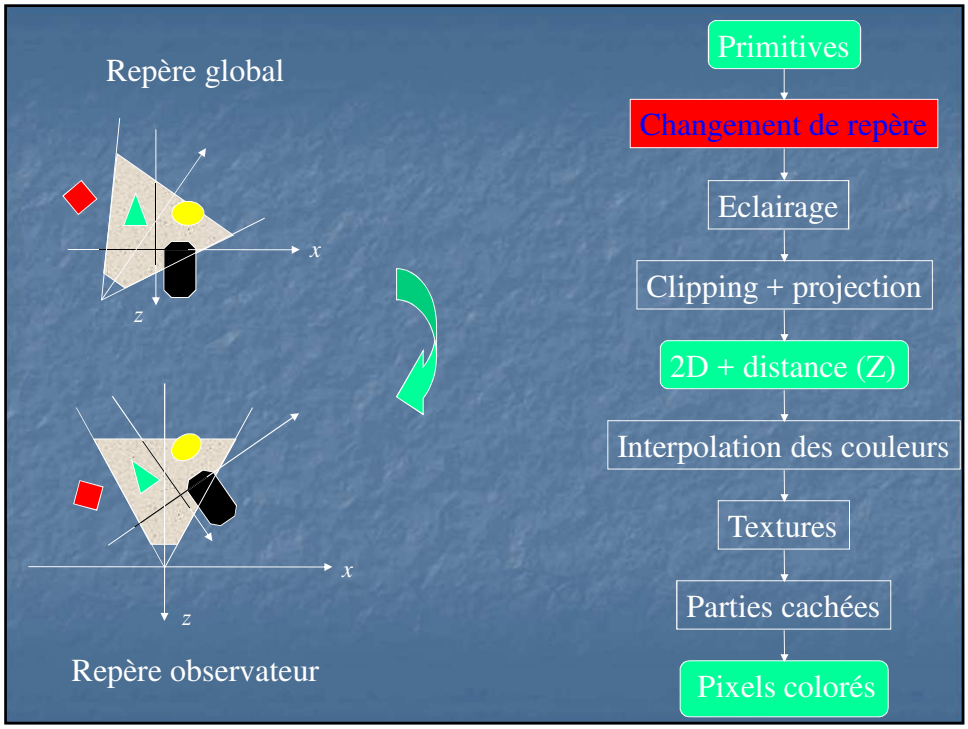
Le pipeline de rendu

- Succession des étapes nécessaires à l'affichage de chaque facette sur la grille de pixels représentant l'écran
- Hypothèse :
  - scène = ensemble de facettes triangulaires
- Exécution
  - software :
    - versions Mesa d'OpenGL
    - Librairie OpenGL
  - Hardware :
    - cartes accélératrices 3D

Le pipeline de rendu standard

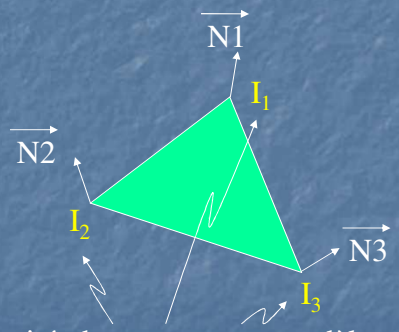



N1
(x1,y1,z1)
N2
N3
(x2,y2,z2)
(x3,y3,z3)
+ caractéristiques
du matériau
+ coordonnées
de texture
Primitives
Changement de repère
Eclairage
Clipping + projection
2D + distance (Z)
Interpolation des couleurs
Textures
Parties cachées
Pixels colorés

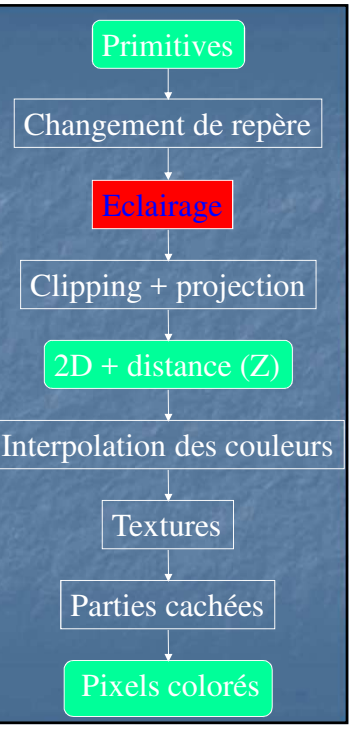

Repère global
x
z
x
z
Repère observateur
Primitives
Changement de repère
Eclairage
Clipping + projection
2D + distance (Z)
Interpolation des couleurs
Textures
Parties cachées
Pixels colorés

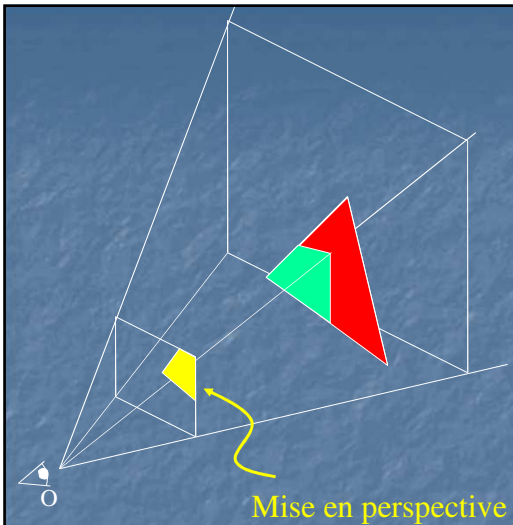
Calcul de l'effet lumineux produit sur chaque objet par chacune des sources présentes dans la scène

Calculs appliqués uniquement aux sommets de chaque triangle

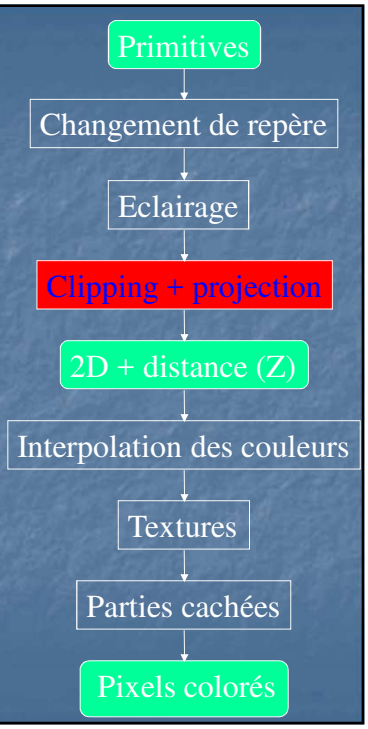


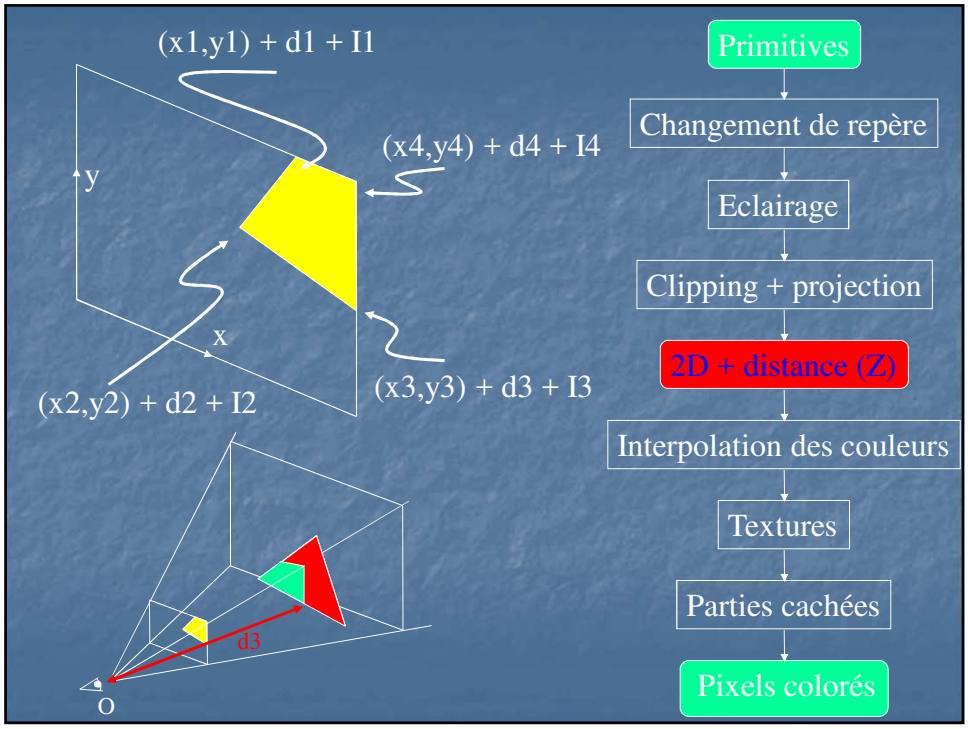
Intensité obtenues par un modèle d'éclairage quelconque
- Cf chapitre 6

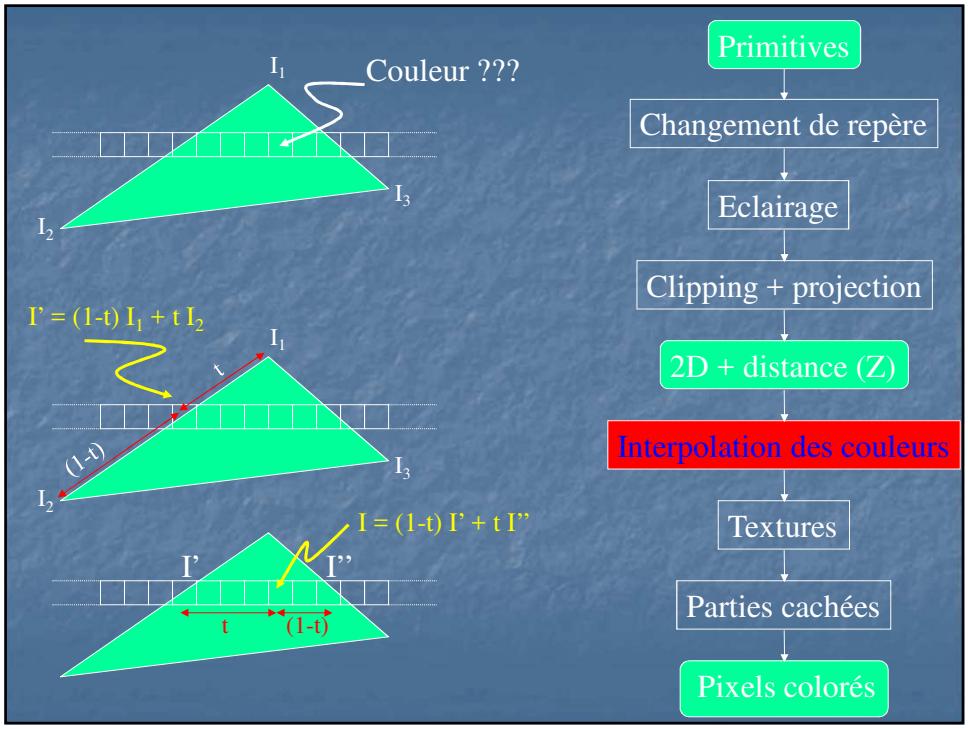


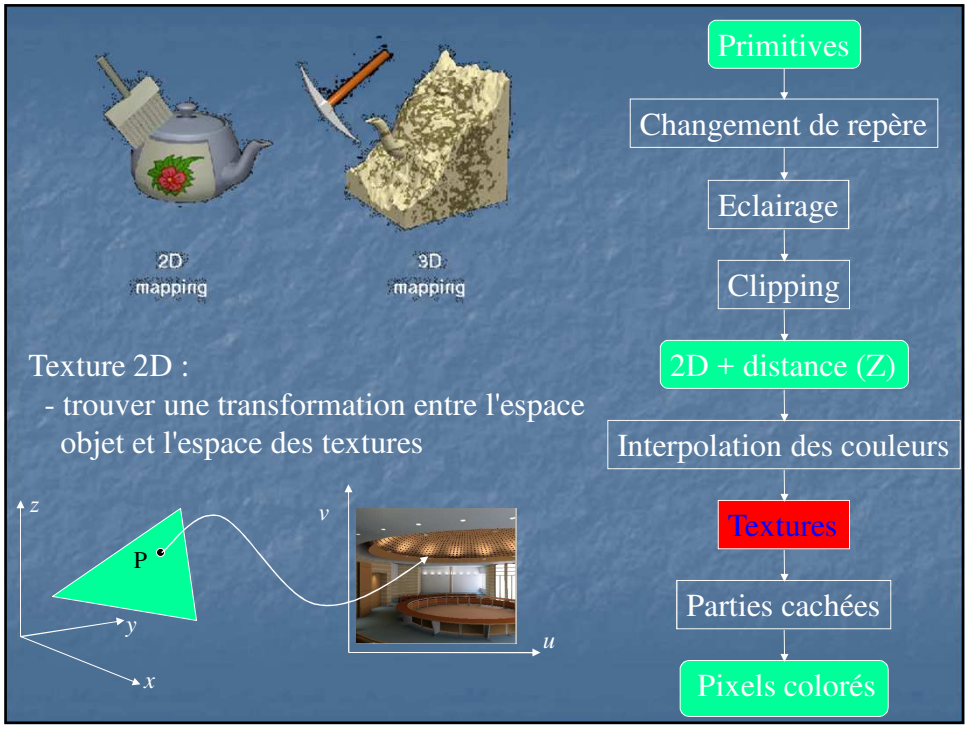


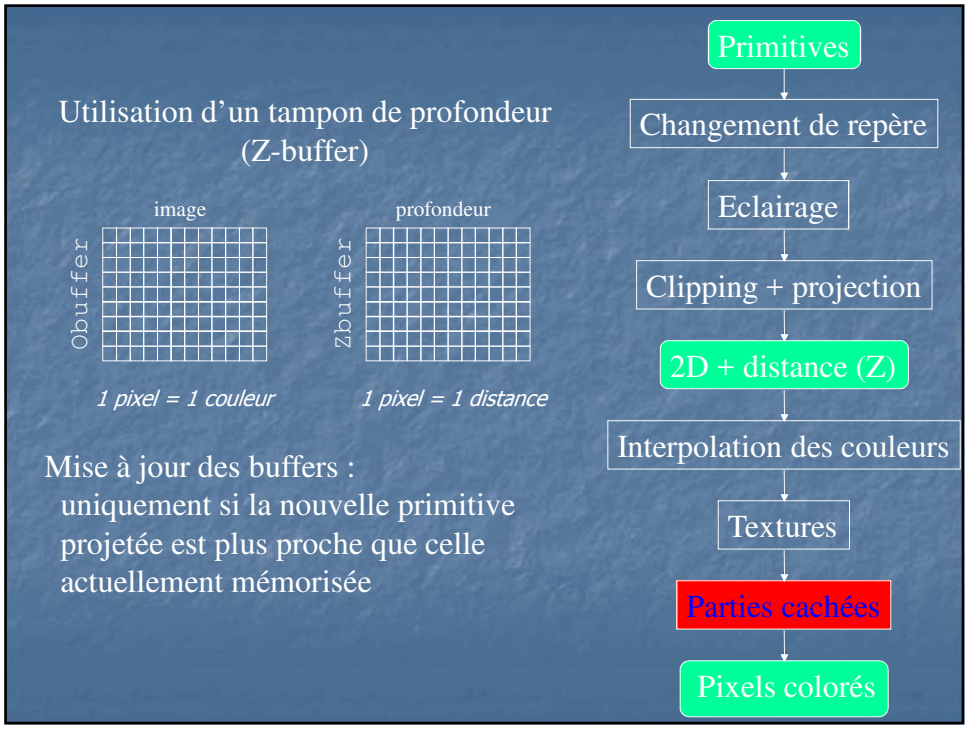
Pyramide de vision = 6 plans de clipping

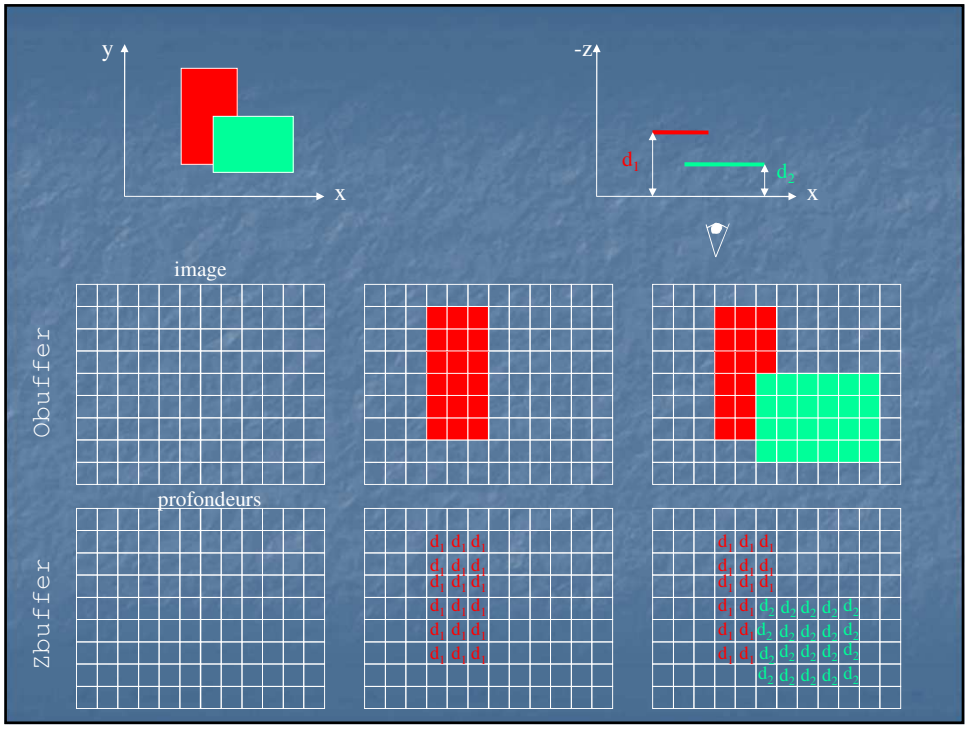



(x1,y1) + d1 + I1
(x4,y4) + d4 + I4
y
x
(x2,y2) + d2 + I2
(x3,y3) + d3 + I3
d3
O
Primitives
Changement de repère
Eclairage
Clipping + projection
2D + distance (Z)
Interpolation des couleurs
Textures
Parties cachées
Pixels colorés


I1
Couleur ???
I3
I2
I' = (1-t) I1 + t I2
I1
t
(1-t)
I3
I2
I = (1-t) I' + t I''
I'
I''
t
(1-t)
Primitives
Changement de repère
Eclairage
Clipping + projection
2D + distance (Z)
Interpolation des couleurs
Textures
Parties cachées
Pixels colorés


2D
mapping
3D
mapping
Texture 2D :
- trouver une transformation entre l'espace objet et l'espace des textures
z
y
x
P
v
u
Primitives
Changement de repère
Eclairage
Clipping
2D + distance (Z)
Interpolation des couleurs
Textures
Parties cachées
Pixels colorés


Utilisation d'un tampon de profondeur (Z-buffer)
image
profondeur
Obuffer
Zbuffer
1 pixel = 1 couleur
1 pixel = 1 distance
Mise à jour des buffers :
uniquement si la nouvelle primitive projetée est plus proche que celle actuellement mémorisée
Primitives
Changement de repère
Eclairage
Clipping + projection
2D + distance (Z)
Interpolation des couleurs
Textures
Parties cachées
Pixels colorés



Améliorations

- Cartes récentes offrent la possibilité de :
  - programmer l'étape d'éclairage (Vertex Shaders)
  - modifier la géométrie traitée (geometry Shders)
  - manipuler des pixels via des programmes plus complexes (Fragment shaders)
- Caractéristiques :
  - programmation en assembleur dédié ou en langages de + haut niveau (Cg nVidia)
  - programmes de taille moyenne (65K instructions)
  - boucles déroulables
- Programmation d'autres "algorithmes"
  - Langage et librairie Cuda (Nvidia)

126

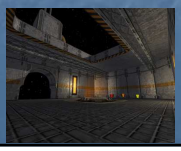
Plan du cours

1. Introduction
2. Modélisation d'objets 3D
3. Introduction à Povray
4. Rendu par lancer de rayons
5. Rendu temps réel
6. Introduction à OpenGL
7. Transformations géométriques
8. Modèles d'éclairage local

Introduction (1)

- OpenGL c'est quoi ?
  - ***Open G**raphics **L**ibrary*
  - Interface de programmation (API) multiplateforme
    - Conception d'application 2D / 3D
    - Regroupe plus de 200 fonctions
  - Très utilisée dans les applications
    - Scientifiques, Industrielles, …
    - Jeu vidéo (en concurrence avec Direct3D)

DOOM 3



Star Wars
Knights of the Old Republic

Introduction (2)

- Implémente le pipeline de rendu
- Définie pour être indépendante de toute plateforme
- Ne gère pas les aspects de haut niveau tels que le fenêtrage, l'interaction avec l'utilisateur, etc …
  - Utilisation de la librairie glut en TP (OpenGL utility toolkit)
- Ne fournit que des primitives graphiques de très bas niveau
  - Avoir recours à des librairies différentes pour construire des objets complexes
    - Exemple : libraire glu (OpenGL utility)

129

Une machine à états

- OpenGL utilise un « *Contexte Graphique* »
  - Ensemble de variables d'état consultées par les instructions de dessin
    - Couleurs de tracé, Épaisseur de trait
    - Matrices de transformation
    - Activation de certains algorithmes (back-face culling, z-buffer, éclairage, …)
  - Conservent leur état tant qu'elles ne sont pas explicitement modifiées

Nom de la variable d'état	*Épaisseur de trait*	*couleur*	*Éclairage actif*
valeur	`2.0`	`(1.0, 0.0, 0.0)`	`vrai`

Tous les tracés se font avec les valeurs actuelles des variables d'état, jusqu'à ce que le programme change explicitement ces valeurs

130

Exemple

Fixer la couleur à « rouge »
Fixer l'épaisseur de ligne à 1.0

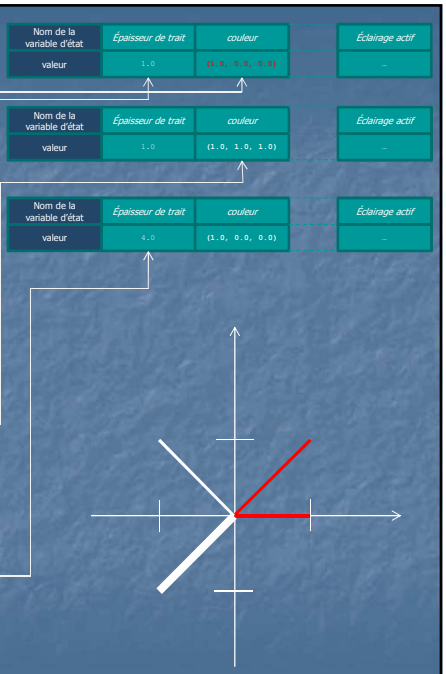
Tracer la ligne (0,0) - (1,0)
Tracer la ligne (0,0) - (1,1)

Fixer la couleur à « blanc »

Tracer la ligne (0,0) – (-1,1)

Fixer l'épaisseur de ligne à 4.0

Tracer la ligne (0,0) – (-1,-1)

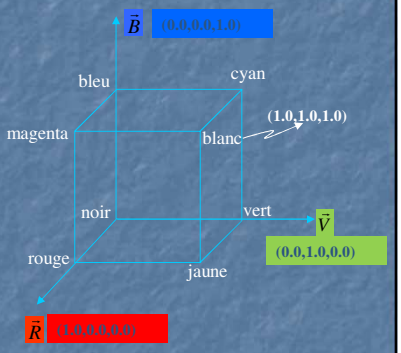


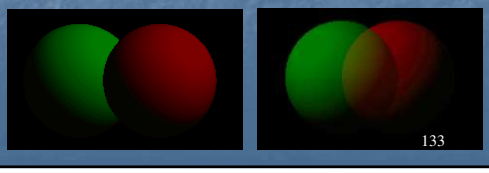
Quelques fonctions OpenGL

- Remarques préliminaires:
  - Fonctions OpenGL commencent toutes par gl
    - `glColor3f(…)`
  - Constantes & types OpenGL commencent par GL
    - `GL_LINES, GL_LIGHTING, GLenum, GLfloat, …`
- Affectation de l'épaisseur de trait:
  - glSetLineWidth(2.0)
- Récupération de l'épaisseur de trait:
  - GLfloat epaisseur;
  - glGetFloatv(GL_LINE_WIDTH, &epaisseur);

Couleurs en OpenGL

- Utilise le modèle (R,V,B,A)
  - (R,V,B) : couleur
  - A : Transparence (*alpha*)
- Chaque canal peut avoir une valeur :
  - Réelle dans [0.0, 1.0]
  - Entière dans [0, 255]
- Transparence
  - 0.0 : totalement transparent
  - 1.0 : totalement opaque





Utiliser les couleurs

- Fixer une couleur courante
  - `glColor3f(GLfloat r, GLfloat v, GLfloat b);`
  - `glColor3i(GLint r, GLint v, GLint b);`
  - Plusieurs autres formes de `glColor3`
- Récupérer la couleur courante
  - `GLfloat coulFloat[4]; GLuint coulInt[4];`
  - `glGetFloatv(GL_CURRENT_COLOR, coulFloat);`
  - `glGetIntegerv(GL_CURRENT_COLOR, coulInt);`
- Utilisation de la transparence
  - `glColor4f(GLfloat r,GLfloat v,GLfloat b,GLfloat a);`
    - `(r,v,b,a) dans [0.0, 1.0]`
  - `glColor4i(GLint r, GLint v, GLint b, GLint a);`
    - `(r,v,b,a) dans [0,255]`
  - Plusieurs autres formes de `glColor4`

Effacer la fenêtre d'affichage (1)

- La fonction glClearColor()
  - Permet de spécifier la couleur à utiliser pour effacer la fenêtre
  - glClearColor(GLfloat r, GLfloat v, GLfloat b, GLfloat a)
  - (r,v,b,a) dans [0.0, 1.0]
  - Mettre a (transparence) à 1.0
- Attention :
  - Cette fonction n'efface pas la fenêtre :
    - On fixe la couleur à utiliser quand on demandera à effacer le fenêtre
  - Exemple : glClearColor(1.0f, 1.0f, 1.0f);
    - La fenêtre sera effacée avec la couleur « blanc » quand on demandera l'effacement

135

Effacer la fenêtre d'affichage (2)

- La fonction glClear() permet d'effacer différents buffers, dont le buffer image (la fenêtre d'affichage)
  - Utilise les valeurs stockées dans le contexte graphique
- Prototype : glClear(GLbitfield mask);
  - mask = ensemble de bit (0 ou 1)
  - Chaque bit correspond à 1 buffer et est défini par une constante
  - Bit du buffer image : GL_COLOR_BUFFER_BIT
- Application :
  - glClear(GL_COLOR_BUFFER_BIT);
  - Efface le buffer image avec la couleur prévue dans le contexte graphique

136

Variables booléennes

- Quelques variables booléennes
  - Spécifient l'activation ou non de certaines fonctionnalités
    - `GL_LIGHTING : nom de la valeur de la fonctionnalité d'éclairage`
- Activation / Désactivation / Test
  - d'une variable d'état booléenne

glEnable(GLenum pname)
Variable d'état à activer (macro)
→ glEnable(GL_LIGHTING)

glDisable(GLenum pname)
Variable d'état à désactiver (macro)
→ glDisable(GL_LIGHTING)

glIsEnabled(GLenum pname)
Variable d'état à tester (macro)
→ glIsEnabled(GL_LIGHTING)

⇒ GL_FALSE ou GL_TRUE

GLUT

- OpenGL :
  - Fonctions de manipulation et de dessin 2D/3D
  - Pas de gestion de
    - l'interface utilisateur
    - de l'interaction avec l'utilisateur
- GLUT (***GL UT**ilities*)
  - Bibliothèque répondant à cette absence
  - Écrite par Mark J. Kilgard
  - Dernière version en mai 1998
  - Remplacée depuis par freeglut et OpenGLUT

Caractéristiques

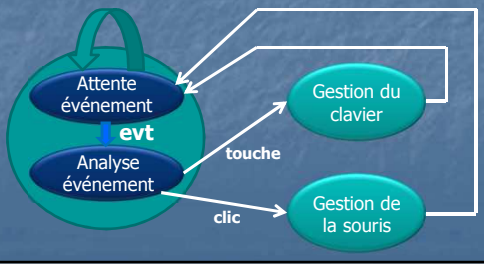
- Portabilité
- Simplicité
- Permet la gestion :
  - Du fenêtrage
  - Des événements utilisateurs (clavier, souris, …)
  - Des menus dynamiques
- Propose des routines de dessin de quelques objets 3d simples
  - sphères, cylindres, tores, …
- Utilise un modèle événementiel

Modèle événementiel

- Initialiser glut
- Connecter les fonctions associées à chaque événement
- Lancement de la boucle de capture des événements

→ Gestion de la souris
→ Gestion du clavier
→ Gestion de l'affichage
→ Gestion de l'animation

glutMainLoop() :



Primitives géométriques OpenGL

- OpenGL ne gère que quelques primitives de base :
  - Lignes, triangles, polygônes
- Déclaration des primitives géométriques :
  - Au sein d'un « bloc » spécialisé

Début du bloc → `glBegin(GLenum typeDePrimitive);`

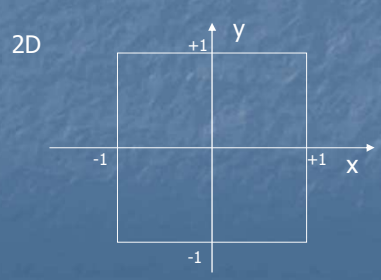
Définition des attributs géométriques (et autres) des facettes

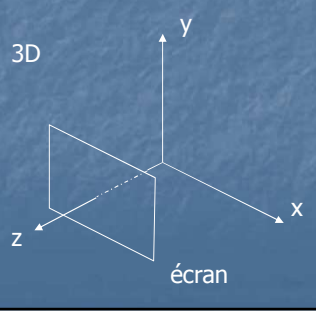
Fin du bloc → `glEnd();`

Différentes possibilités (lignes, triangles, polygones, …)

Attributs géométriques (1)

- Possibilité de travailler :
  - En 2D : dans le repère écran
  - En 3D : dans le repère de la scène
- Les repères OpenGL





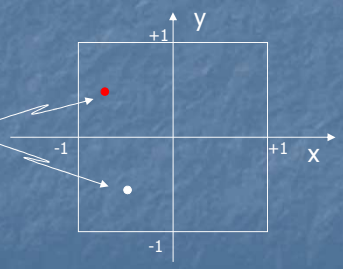
Attributs géométriques (2)

- Coordonnées des points 2D
  - glVertex2f(GLfloat x, GLfloat y);
- Coordonnées des points 3D
  - glVertex3f(GLfloat x, GLfloat y, GLfloat z);
- Existence d'autres formes …
- Peuvent également être spécifiées :
  - Couleurs
  - Normales
  - Coordonnées de texture

Types de primitives (1)

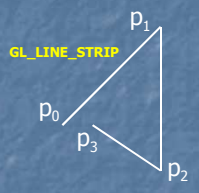
- Les points
  - GL_POINTS

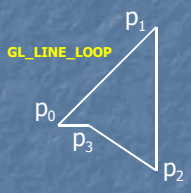
```
glBegin(GL_POINTS);
    glVertex2f(-0.5, -0.5);
    glColor3f(1.0, 0.0, 0.0);
    glVertex2f(-0.75, 0.5);
glEnd();
```



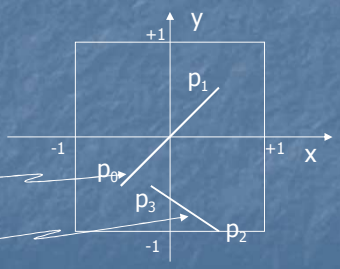
Types de primitive (2)

- Les lignes
  - GL_LINES
  - GL_LINE_STRIP
  - GL_LINE_LOOP



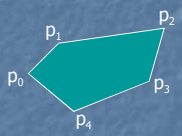



```
glBegin(GL_LINES);
    glVertex2f(-0.5, -0.5);
    glVertex2f(0.5, 0.5);
    glVertex2f(0.5, -1.0);
    glVertex2f(-0.25, -0.5);
glEnd();
```

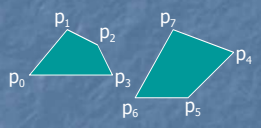


Types de primitive (3)

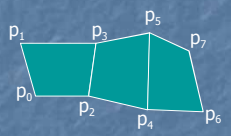
- Les polygones
  - uniquement convexes (retransformés en triangles)
  - 6 modes différents



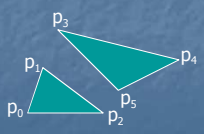
GL_POLYGON



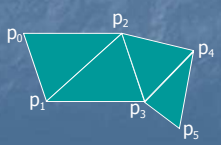
GL_QUADS



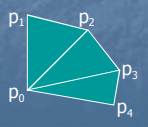
GL_QUAD_STRIP



GL_TRIANGLES



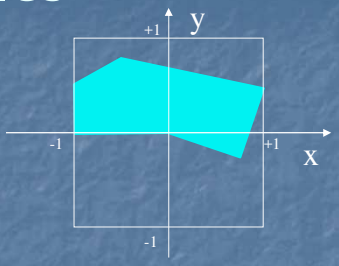
GL_TRIANGLE_STRIP



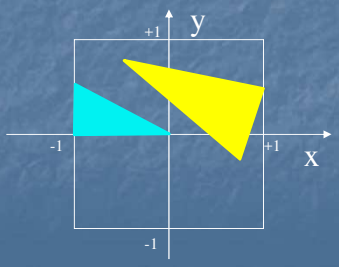
GL_TRIANGLE_FAN

Exemples

```
glBegin(GL_POLYGON);
        glVertex2f(0.0, 0.0);
        glVertex2f(-1.0, 0.0);
        gVertex2f(-1.0, 0.5);
        glVertex2f(-0.5, 0.75);
        glVertex2f(1.0, 0.5);
        glVertex2f(0.75, -0.25);
glEnd();
```

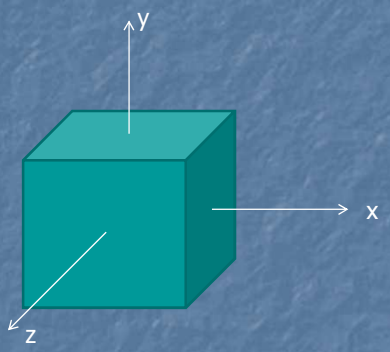



```
glBegin(GL_TRIANGLES);
        glVertex2f(0.0, 0.0);
        glVertex2f(-1.0, 0.0);
        gVertex2f(-1.0, 0.5);
        glVertex2f(-0.5, 0.75);
        glVertex2f(1.0, 0.5);
        glVertex2f(0.75, -0.25);
glEnd();
```



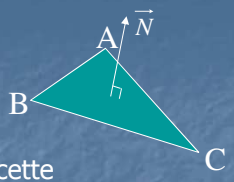
147

Application

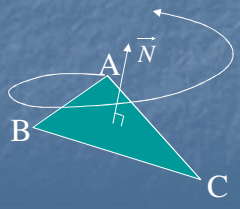


- Cube de centre O et de côté 2 unités, de couleur verte
- Idem, avec faces avant et arrière rouges

Les normales



- Normale à une facette (plane) :
  - Vecteur perpendiculaire au plan de la facette
- Propriété à assurer :
  - La normale doit toujours être orientée vers l'extérieur d'un objet facettisé
- Calcul d'une normale : $\vec{N} = \overrightarrow{AB} \wedge \overrightarrow{AC}$
- Remarque :
  - Lors de la modélisation, les points A, B et C doivent être spécifiés dans l'ordre direct (règle du tire-bouchon …)

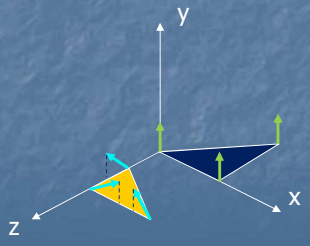


149

Normales OpenGL

- Par défaut
  - Normale calculée en fonction de l'ordre de spécification des sommets
    - Toujours s'assurer de l'ordre direct !!!
- Spécification « manuelle »
  - glNormal3f(GLfloat nx, GLfloat ny, GLfloat nz);
  - S'applique sur le(s) sommet(s) qui sui(ven)t

```
glBegin(GL_TRIANGLES);
  glNormal3f(0.0, 0.7, 0.7);
  glVertex3f(0.0, 0.0, 0.5);
  glNormal3f(0.7, 0.7, 0.0);
  glVertex3f(0.0, 0.0, 1.0);
  glNormal3f(-0.7, 0.7, 0.0);
  glVertex3f(1.0, 0.0, 1.0);
glEnd();
```




```
glBegin(GL_TRIANGLES);
  glNormal3f(0.0, 1.0, 0.0);
  glVertex3f(0.0, 0.0, 0.0);
  glVertex3f(1.0, 0.0, 0.0);
  glVertex3f(1.0, 0.0, -1.0);
glEnd();
```

Utilité

- Calculs d'éclairage
  - Voir partie 6
- Tests de visibilité

Extérieur (par rapport à l'objet)
pour chaque facette

L'observateur ne peut voir que les facettes dont l'« extérieur » est orienté vers lui

Test d'orientation

soient deux vecteurs $\vec{u}$ et $\vec{v}$

si $\vec{u}.\vec{v}>0$ alors $\vec{u}$ et $\vec{v}$ sont orientés dans la même direction

Back-face culling OpenGL

- Permet d'éliminer les facettes obligatoirement invisibles
  - Augmente les performances d'affichage
    - Les facettes éliminées ne passent pas dans le pipeline de rendu
- Activation :
  - glEnable(GL_CULL_FACE)
- Désactivation
  - glDisable(GL_CULL_FACE)
- Choix de l'orientation des facettes à supprimer
  - glCullFace(GL_BACK) (par défaut)
  - glCullFace(GL_FRONT)

153

Affichage des polygones

- Par défaut, les polygones sont affichés en mode « plein », avec la couleur active
  - Possibilité de choisir entre mode « plein » et mode « contour »
- Un polygone a deux faces
  - Face avant (face extérieure selon la normale)
  - Face arrière (face intérieure selon la normale)
  - Possibilité de fixer un mode d'affichage différent pour chaque face

void glPolygonMode(GLenum face, GLenum mode)

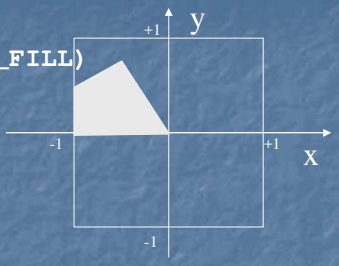
GL_FRONT ou
GL_BACK ou
GL_FRONT_AND_BACK

GL_POINT ou
GL_LINE ou
GL_FILL

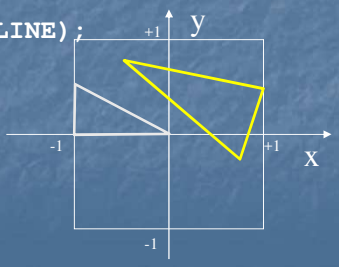
154

Exemples

```
glPolygonMode(GL_FRONT, GL_FILL)
glBegin(GL_POLYGON);
      glVertex2f(0.0, 0.0);
      glVertex2f(-1.0, 0.0);
      gVertex2f(-1.0, 0.5);
      glVertex2f(-0.5, 0.75);
glEnd();
```




```
glPolygonMode(GL_FRONT,GL_LINE);
glBegin(GL_TRIANGLES);
      glVertex2f(0.0, 0.0);
      glVertex2f(-1.0, 0.0);
      gVertex2f(-1.0, 0.5);
      glVertex2f(-0.5, 0.75);
      glVertex2f(1.0, 0.5);
      glVertex2f(0.75, -0.25);
glEnd();
```



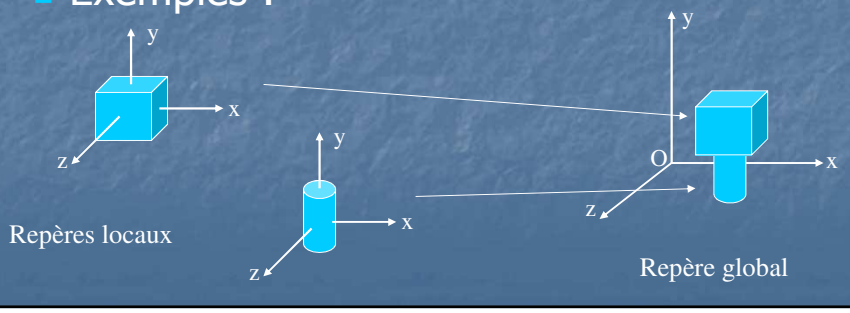
155

Plan du cours

1. Introduction
2. Modélisation d'objets 3D
3. Introduction à Povray
4. Rendu par lancer de rayons
5. Rendu temps réel
6. Introduction à OpenGL
7. Transformations géométriques
8. Modèles d'éclairage local

1. Objectifs

- Déplacer les objets de leur repère de définition (repère local) vers leur position dans la scène (repère global)
- Animer les objets dans la scène
- Exemples :



2. Rappels mathématiques

- Produit scalaire
- Norme d'un vecteur
- Distance euclidienne
- Produit vectoriel
- Équation cartésienne d'un plan

➡ Notions utiles pour le positionnement et l'animation

158

Produit scalaire

- Définition
  - L'espace est rapporté à un repère orthonormé $(O, \vec{i}, \vec{j}, \vec{k})$
  - Pour tout vecteur $\vec{u} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ et $\overrightarrow{u'} = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$ le produit scalaire de $\vec{u}$ et $\overrightarrow{u'}$ est défini par le réel : $\vec{u}.\overrightarrow{u'} = x.x' + y.y' + z.z'$
- Notation matricielle
  - un vecteur = une matrice uni-colonne
  - $u = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, u' = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} \Rightarrow \vec{u}.\overrightarrow{u'} = u^t.u' = [x \quad y \quad z].\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$

159

Produit scalaire - propriétés

- Le produit scalaire est une forme bilinéaire symétrique définie positive.
  Pour tout vecteur $\vec{u}$, $\vec{v}$, $\vec{w}$, pour tout réel λ :
  - $\vec{u}.\vec{v} = \vec{v}.\vec{u}$
  - $(\vec{u} + \vec{v}).\vec{w} = (\vec{u}.\vec{w}) + (\vec{v}.\vec{w})$
  - $(\lambda\vec{u}).\vec{v} = \lambda(\vec{u}.\vec{v})$
  - $\vec{u}.\vec{u} \geq 0$
  - $\vec{u}.\vec{u} = 0 \Leftrightarrow \vec{u} = \vec{0}$
- Les vecteurs $\vec{u}$ et $\vec{v}$ sont orthogonaux si et seulement si $\vec{u}.\vec{v} = 0$. On note $\vec{u} \perp \vec{v}$.

160

Norme d'un vecteur

- Définition
  - La norme du vecteur $\vec{u} = (x, y, z)^t$ est le réel positif ou nul

$$\|\vec{u}\| = \sqrt{\vec{u}.\vec{u}} = \sqrt{x^2 + y^2 + z^2}$$

- Propriété
  - Pour tout vecteur $\vec{u}$, $\vec{v}$, $\vec{w}$, pour tout réel λ :
    - $\|\vec{u}\| = 0 \Leftrightarrow \vec{u} = \vec{0}$
    - $\|\overrightarrow{\lambda u}\| = |\lambda|.\|\vec{u}\|$
    - $\|\vec{u} + \vec{v}\| \leq \|\vec{u}\|.\|\vec{v}\|$
  - Pour tout $\vec{u}$, $\vec{v}$:
    - $\vec{u}.\vec{v} = \|\vec{u}\|.\|\vec{v}\|.\cos(\vec{u}, \vec{v})$

161

Distance euclidienne

- Définition
  - La distance euclidienne de deux points A et B de l'espace est le réel : $d(A, B) = \|\overrightarrow{AB}\|$
  - Donc, si $A\begin{pmatrix} x_A \\ y_A \\ z_A \end{pmatrix}$ et $B\begin{pmatrix} x_B \\ y_B \\ z_B \end{pmatrix}$, alors

$$d(A, B) = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2 + (z_B - z_A)^2}$$

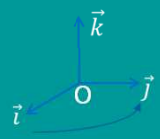
- Propriétés
  - Pour tous les points A, B et C :
    - $d(A, B) = 0 \Leftrightarrow A = B$
    - $d(A, B) = d(B, A)$
    - $d(A, C) \leq d(A, B) + d(B, C)$

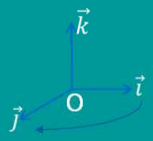
162

Produit vectoriel

- L'espace est rapporté à un repère orthonormé direct $(O, \vec{\imath}, \vec{\jmath}, \vec{k})$

Repère direct





Repère indirect

- Le produit vectoriel de deux vecteurs $\vec{u}$ et $\vec{v}$ est défini par :

$$\vec{u} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \vec{v} = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} \Rightarrow \vec{u} \wedge \vec{v} = \begin{pmatrix} yz' - y'z \\ x'z - xz' \\ xy' - x'y \end{pmatrix}$$

163

Produit vectoriel - propriétés

- Pour tous les vecteurs $\vec{u}$ et $\vec{v}$, le vecteur $\vec{u} \wedge \vec{v}$ est perpendiculaire à $\vec{u}$ et à $\vec{v}$.
- Si les vecteurs $\vec{u}$ et $\vec{v}$ sont normés et orthogonaux, alors $(\vec{u}, \vec{v}, \vec{u} \wedge \vec{v})$ est une base orthonormée directe.
- Pour toute base orthonormée directe $(\vec{\imath}, \vec{\jmath}, \vec{k})$, on a :

$$\vec{\imath} \wedge \vec{\jmath} = \vec{k} \quad , \quad \vec{\jmath} \wedge \vec{k} = \vec{\imath} \quad , \quad \vec{k} \wedge \vec{\imath} = \vec{\jmath}$$

- Le produit vectoriel est anti-symétrique :

$$\vec{\imath} \wedge \vec{\jmath} = -(\vec{\jmath} \wedge \vec{\imath})$$

- $\vec{u} \wedge \vec{v} = \vec{0}$ si et seulement si $\vec{u}$ et $\vec{v}$ sont colinéaires

donc A, B et C sont alignés ssi $\overrightarrow{AB} \wedge \overrightarrow{AC} = \vec{0}$

164

Produit vectoriel – propriétés (2)

- Le produit vectoriel est une application bilinéaire. Pour tous les vecteurs $\vec{u}$, $\vec{v}$, $\vec{w}$ on a :
  - $(\lambda\vec{u}) \wedge \vec{v} = \lambda(\vec{u} \wedge \vec{v}) = \vec{u} \wedge (\lambda\vec{v})$
  - $\vec{u} \wedge (\vec{v} + \vec{w}) = (\vec{u} \wedge \vec{v}) + (\vec{u} \wedge \vec{w})$
  - $(\vec{u} + \vec{v}) \wedge \vec{w} = (\vec{u} \wedge \vec{w}) + (\vec{v} \wedge \vec{w})$

- Pour tous les vecteurs $\vec{u}$ et $\vec{v}$ non nuls, on a :

$$\vec{u} \wedge \vec{v} = \|\vec{u}\|.\|\vec{v}\|.\sin(\vec{u}, \vec{v}).\vec{k}$$

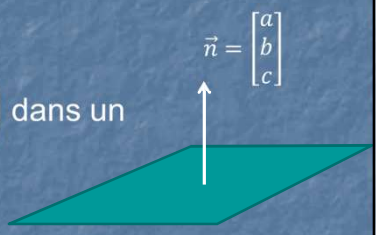
où $\vec{k}$ est un vecteur normé perpendiculaire à $\vec{u}$ et à $\vec{v}$.

165

Equation cartésienne d'un plan

- Soit $(a, b, c) \in \mathbb{R}^3$ non nul.
- Un plan (P) de vecteur normal $\vec{n}\begin{pmatrix} a \\ b \\ c \end{pmatrix}$ dans un repère orthonormal a pour équation cartésienne :

$$ax + by + cz + d = 0, \text{ où } d \in \mathbb{R}$$

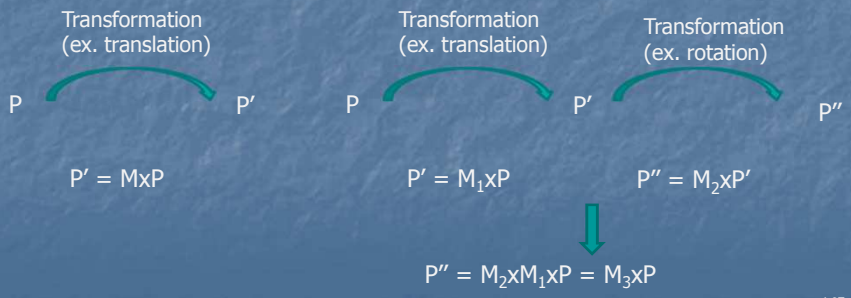


- Réciproquement l'ensemble des points de l'espace qui vérifie l'équation $ax + by + cz + d = 0$ est un plan de vecteur normal $\vec{n}\begin{pmatrix} a \\ b \\ \end{pmatrix}$.

166

Le produit matriciel

- En informatique graphique, le produit matriciel (produit d'une matrice par un vecteur ou le produit de deux matrices) est un outil de base
- Toutes les transformations sont représentées par une matrice
- Les matrices et vecteurs sont de dimension 3 ou 4



167

Le produit matrice-vecteur

$$M = \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,1} & m_{3,2} & m_{3,3} \end{bmatrix}, \qquad \vec{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$$

$$M.\vec{u} = \vec{v} \quad \Leftrightarrow \quad \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,1} & m_{3,2} & m_{3,3} \end{bmatrix} . \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

avec $$v_i = [m_{i,1} \quad m_{i,2} \quad m_{i,3}] . \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$$

« Produit scalaire de la ligne i de la matrice avec le vecteur u »

168

Exemple

Calculons : $\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 1 \\ -1 & 1 & -2 \end{bmatrix} . \begin{bmatrix} 2 \\ 3 \\ -1 \end{bmatrix}$

v_1= 1x2 + 1x3 + 1x(-1) = 4

v_2= 0x2 + 2x3 + 1x(-1) = 5

v_3= (-1)x2 + 1x3 + (-2)x(-1) = 3

donc $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 1 \\ -1 & 1 & -2 \end{bmatrix} . \begin{bmatrix} 2 \\ 3 \\ -1 \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \\ 3 \end{bmatrix}$

169

Le produit matrice-matrice

$A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix}$, $B = \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}$

$A.B = C \Leftrightarrow \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} . \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix} = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix}$

avec $c_{l,c} = [a_{l,1} \quad a_{l,2} \quad a_{l,3}] . \begin{bmatrix} b_{1,c} \\ b_{2,c} \\ b_{3,c} \end{bmatrix}$

« Produit scalaire de la ligne d'indice l de A et de la colonne d'indice c de B »

170

Exemple

Soit à calculer C = B.A, avec :

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 1 \\ -1 & 1 & -2 \end{bmatrix}, \quad B = \begin{bmatrix} 2 & 0 & 1 \\ 3 & 2 & 0 \\ -1 & 2 & 0 \end{bmatrix}$$

$$c_{1,1} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 1 \\ -1 & 1 & -2 \end{bmatrix} . \begin{bmatrix} 2 & 0 & 1 \\ 3 & 2 & 0 \\ -1 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 4 & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{bmatrix}$$

$$c_{2,1} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 1 \\ -1 & 1 & -2 \end{bmatrix} . \begin{bmatrix} 2 & 0 & 1 \\ 3 & 2 & 0 \\ -1 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 4 & ? & ? \\ 5 & ? & ? \\ ? & ? & ? \end{bmatrix}$$

$$c_{3,1} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 1 \\ -1 & 1 & -2 \end{bmatrix} . \begin{bmatrix} 2 & 0 & 1 \\ 3 & 2 & 0 \\ -1 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 4 & ? & ? \\ 5 & ? & ? \\ 3 & ? & ? \end{bmatrix}$$

171

Exemple (suite)

Soit à calculer C = A.B, avec :

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 1 \\ -1 & 1 & -2 \end{bmatrix}, \quad B = \begin{bmatrix} 2 & 0 & 1 \\ 3 & 2 & 0 \\ -1 & 2 & 0 \end{bmatrix}$$

$$c_{1,2} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 1 \\ -1 & 1 & -2 \end{bmatrix} . \begin{bmatrix} 2 & 0 & 1 \\ 3 & 2 & 0 \\ -1 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 4 & 4 & ? \\ 5 & ? & ? \\ 3 & ? & ? \end{bmatrix}$$

$$c_{2,2} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 1 \\ -1 & 1 & -2 \end{bmatrix} . \begin{bmatrix} 2 & 0 & 1 \\ 3 & 2 & 0 \\ -1 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 4 & 4 & ? \\ 5 & 6 & ? \\ 3 & ? & ? \end{bmatrix}$$

$$c_{3,2} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 1 \\ -1 & 1 & -2 \end{bmatrix} . \begin{bmatrix} 2 & 0 & 1 \\ 3 & 2 & 0 \\ -1 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 4 & 4 & ? \\ 5 & 6 & ? \\ 3 & -2 & ? \end{bmatrix}$$

172

Exemple (suite)

Soit à calculer C = A.B, avec :

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 1 \\ -1 & 1 & -2 \end{bmatrix}, \quad B = \begin{bmatrix} 2 & 0 & 1 \\ 3 & 2 & 0 \\ -1 & 2 & 0 \end{bmatrix}$$

$$c_{1,3} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 1 \\ -1 & 1 & -2 \end{bmatrix} . \begin{bmatrix} 2 & 0 & 1 \\ 3 & 2 & 0 \\ -1 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 4 & 4 & 1 \\ 5 & 6 & ? \\ 3 & -2 & ? \end{bmatrix}$$

$$c_{2,3} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 1 \\ -1 & 1 & -2 \end{bmatrix} . \begin{bmatrix} 2 & 0 & 1 \\ 3 & 2 & 0 \\ -1 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 4 & 4 & 1 \\ 5 & 6 & 0 \\ 3 & -2 & ? \end{bmatrix}$$

$$c_{3,3} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 1 \\ -1 & 1 & -2 \end{bmatrix} . \begin{bmatrix} 2 & 0 & 1 \\ 3 & 2 & 0 \\ -1 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 4 & 4 & 1 \\ 5 & 6 & 0 \\ 3 & -2 & -1 \end{bmatrix}$$

173

Exemple (fin)

Soit à calculer C = A.B, avec :

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 1 \\ -1 & 1 & -2 \end{bmatrix}, \quad B = \begin{bmatrix} 2 & 0 & 1 \\ 3 & 2 & 0 \\ -1 & 2 & 0 \end{bmatrix}$$

Le résultat est donc :

$$C = B.A = \begin{bmatrix} 4 & 4 & 1 \\ 5 & 6 & 0 \\ 3 & -2 & -1 \end{bmatrix}$$

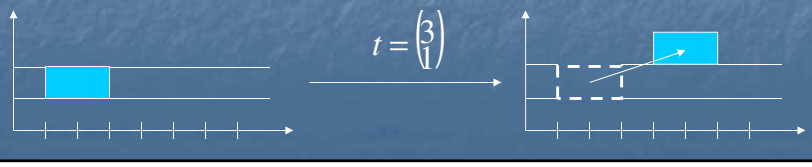
174

Transformations usuelles
les translations

$$t : R^n \rightarrow R^n$$

2D : $P\begin{pmatrix}x\\y\end{pmatrix} \xrightarrow{t} P'\begin{pmatrix}x'\\y'\end{pmatrix} \; avec \begin{cases} x' = x + t_x \\ y' = y + t_y \end{cases}$

3D : $P\begin{pmatrix}x\\y\\z\end{pmatrix} \xrightarrow{t} P'\begin{pmatrix}x'\\y'\\z'\end{pmatrix} \; avec \begin{cases} x' = x + t_x \\ y' = y + t_y \\ z' = z + t_z \end{cases}$

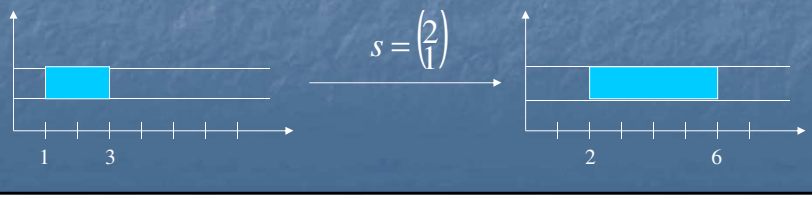


Transformations usuelles
les changements d'échelle

$$s : R^n \rightarrow R^n$$

2D : $P\begin{pmatrix}x\\y\end{pmatrix} \xrightarrow{s} P'\begin{pmatrix}x'\\y'\end{pmatrix} \; avec \begin{cases} x' = s_x \times x \\ y' = s_y \times y \end{cases}$

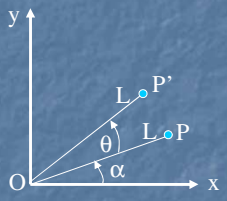
3D : $P\begin{pmatrix}x\\y\\z\end{pmatrix} \xrightarrow{s} P'\begin{pmatrix}x'\\y'\\z'\end{pmatrix} \; avec \begin{cases} x' = s_x \times x \\ y' = s_y \times y \\ z' = s_z \times z \end{cases}$



Transformations usuelles

les rotations 2D

$$R_{\theta,axe}: R^n \rightarrow R^n$$

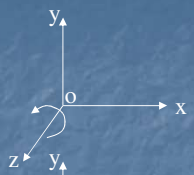


$$P = \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} L\cdot\cos(\alpha) \\ L\cdot\sin(\alpha) \end{pmatrix}$$

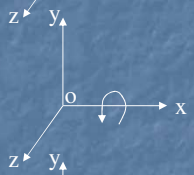
$$P' = \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x\cdot\cos(\theta) - y\cdot\sin(\theta) \\ x\cdot\sin(\theta) + y\cdot\cos(\theta) \end{pmatrix}$$

les rotations 3D

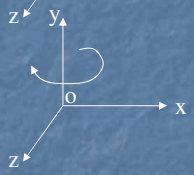
Rotation d'axe Oz : $P' = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x\cdot\cos(\theta) - y\cdot\sin(\theta) \\ x\cdot\sin(\theta) + y\cdot\cos(\theta) \\ z \end{pmatrix}$



Rotation d'axe Ox : $P' = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x \\ y\cdot\cos(\theta) - z\cdot\sin(\theta) \\ y\cdot\sin(\theta) + z\cdot\cos(\theta) \end{pmatrix}$



Rotation d'axe Oy : $P' = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x\cdot\cos(\theta) - z\cdot\sin(\theta) \\ y \\ x\cdot\sin(\theta) + z\cdot\cos(\theta) \end{pmatrix}$



Rotation d'axe quelconque :

- 3 rotations successives d'axe Ox, Oy, Oz
- angles des rotations = angles d'Euler

Représentation matricielle

- Objectifs
  - Uniformiser la représentation des transformations
  - Simplifier leur utilisation / composition
- Principe
  - Toutes les transformations usuelles peuvent s'écrire sous forme matricielle
  - M = matrice représentant la transformation
  - P' = M.P

Matrices de rotation

$$P' = R_{\theta,axe} \times P$$

Selon l'axe Oz $\quad R_{\theta,z} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Selon l'axe Oy $\quad R_{\theta,y} = \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix}$

Selon l'axe Ox $\quad R_{\theta,x} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}$

Matrice de changement d'échelle

$$P' = S \times P$$

$$S = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{pmatrix}$$

Le cas des translations

$$P' = P + T$$

$$T = \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}$$

- Problème :
  - Une translation ne se fait pas sous la forme P'=M.P
- Conséquences :
  - Non uniformité de la représentation matricielle
  - Multiplication des calculs en cas de composition des transformations (on ne peut pas calculer une seule matrice de passage)

Les coordonnées homogènes

- Espace des coordonnées homogènes = 4D
- Homogène : permet de représenter toutes les transformations usuelles sous forme de matrices

$$P=\begin{pmatrix}x\\y\\z\end{pmatrix}\longrightarrow P'=\begin{pmatrix}x\\y\\z\\1\end{pmatrix}$$

$$P=\begin{pmatrix}\frac{x}{\omega}\\\frac{y}{\omega}\\\frac{z}{\omega}\end{pmatrix}\longleftarrow P'=\begin{pmatrix}x\\y\\z\\\omega\end{pmatrix}$$

Remarque
ω=0 définit un point à l'infini dont (x,y,z) est le vecteur directeur

Application (1)
les translations

$$\vec{T}=\begin{pmatrix}t_x\\t_y\\t_z\\0\end{pmatrix}\qquad P=\begin{pmatrix}x\\y\\z\\\omega\end{pmatrix}\xrightarrow{\vec{T}}P'=\begin{pmatrix}x+t_x\\y+t_y\\z+t_z\\\omega\end{pmatrix}\qquad M_T=T=\begin{pmatrix}1&0&0&t_x\\0&1&0&t_y\\0&0&1&t_z\\0&0&0&1\end{pmatrix}$$

$$P''=T\times P'=\begin{pmatrix}x+\omega.t_x\\y+\omega.t_y\\z+\omega.t_z\\\omega\end{pmatrix}\xrightarrow{\text{retour dans } R_3}\begin{pmatrix}\frac{x}{\omega}+t_x\\\frac{y}{\omega}+t_y\\\frac{z}{\omega}+t_z\end{pmatrix}$$

Généralement : ω = 1

Application (2)
rotations et changements d'échelle

$$R_{\theta,z} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad S = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Autour de 0z

Composition des transformations

- Coordonnées homogènes :
  - Toutes les transformations usuelles peuvent se représenter sous forme matricielle
  - Application sous la forme : P' = M.P
  - M = matrice de transformation
- Composition de 2 transformations :
  - Écriture identique
  - Matrice M = **produit** des 2 matrices de transformation à utiliser

Remarque importante

- Le produit de 2 matrices de transformation doit être fait dans l'ordre **inverse** de l'ordre d'application des 2 transformations
- Exemple : $P \xrightarrow{M_1} P' \xrightarrow{M_2} P''$

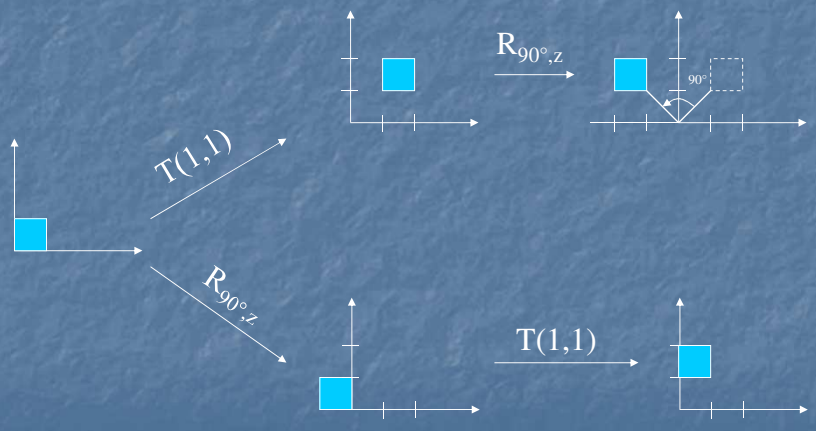
$$P' = M_1 \times P \qquad P'' = M_2 \times P' = M_2 \times M_1 \times P$$

  - Rappel :
    - Le produit de matrice n'est pas commutatif

$$M_2 \times M_1 \neq M_1 \times M_2$$

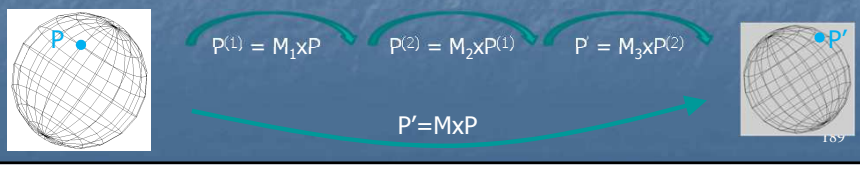
=> composition non commutative

Exemple



Avantage de la représentation matricielle

- Une transformation doit, en général, être appliquée sur de nombreux points de la scène
- Si $M = M_n \times M_{n-1} \times \ldots \times M_2 \times M_1$
  - On calcule une seule fois M
  - On applique M sur tous les points à transformer
- Sans composition :
  - On applique n produits de matrice pour chaque point à transformer …

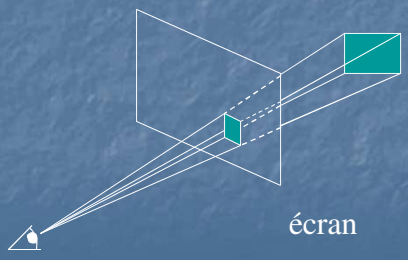


Les projections

- Transformations précédentes :
  - s'appliquent dans le repère de la scène
  - servent à configurer la scène
- Visualisation :
  - à l'affichage, il faut générer une vue de la scène
    - => projection des objets sur l'écran
  - dépend de plusieurs paramètres
    - type de projection
    - position de l'observateur
    - angle de vision (ouverture)

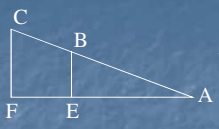
La projection perspective

- Observateur à distance finie de l'écran
  - il existe un point de fuite

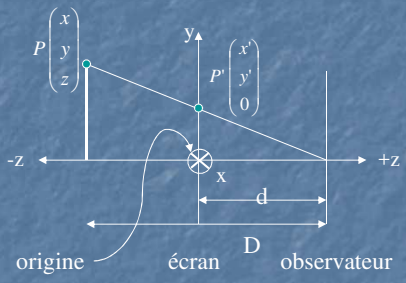


Équations de passage

Théorème de Thalès :



$$\frac{AE}{AF} = \frac{BE}{CF}$$



$$\begin{cases} \frac{x}{x'} = \frac{D}{d} \\ \frac{y}{y'} = \frac{D}{d} \end{cases} \quad D = z + d \quad \Rightarrow \quad \begin{cases} x' = \frac{d.x}{z+d} \\ y' = \frac{d.y}{z+d} \end{cases}$$

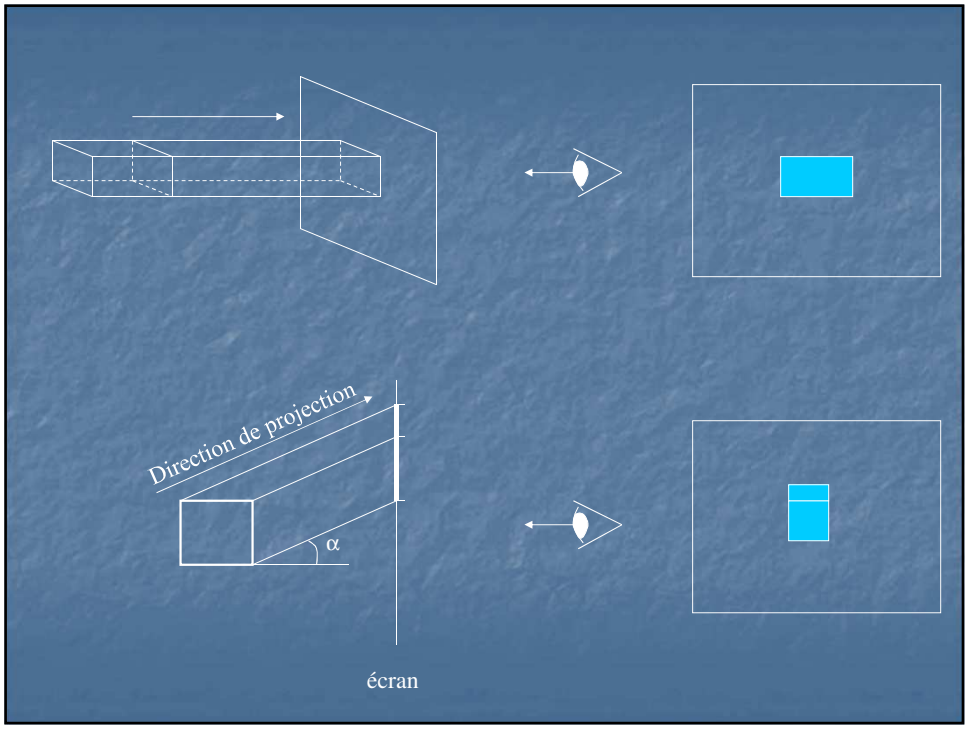
Représentation matricielle

$$P = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \rightarrow P' = \begin{pmatrix} \frac{d.x}{D} \\ \frac{d.y}{D} \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \\ \frac{D}{d} \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \\ \frac{z}{d}+1 \end{pmatrix}$$

$$M_{pp} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 1 \end{pmatrix}$$

Projections parallèles

- Observateur à l'infini
  - pas de point de fuite
  - tous les points se projettent suivant la même direction
- Deux types principaux
  - projection orthogonale
    - écran perpendiculaire à la direction de projection
  - projection oblique
    - écran incliné d'un angle α par rapport à la direction de projection
    - cas particulier : $\alpha = 45°$ (projection cavalière)



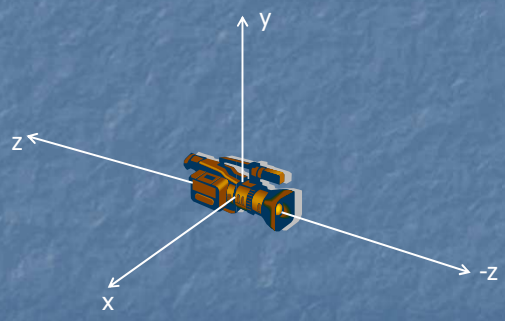
Les transformations OpenGL

- OpenGl utilise 3 transformations géométriques successives pour afficher la scène 3D sur l'écran :
  1. le déplacement des objets à la position que leur attribue le programme ;
  2. la projection des objets sur l'écran ;
  3. leur affichage dans la zone écran qui est attribuée à l'image (transformation en pixels).
- Ces actions sont exécutées successivement au cours du pipeline graphique (pipeline de de rendu)
- A chaque étape correspond une matrice différente
  - Chacune des matrice est stockée dans le pipeline graphique

195

Principe sous OpenGL

Par défaut, OpenGL considère que l'observateur (la caméra) est placé à l'origine du repère universel et qu'il regarde dans la direction des z négatifs



Pour que l'observateur puisse voir le contenu de la scène, ce sont les objets qui sont déplacés …

196

Sélection d'une matrice

- OpenGL gère plusieurs matrices de transformation géométrique
  - Il faut sélectionner celle que l'on souhaite modifier
  - À un instant donné, il n'existe qu'une seule matrice active dans le programme : **la matrice courante**

- Sélection d'une matrice de transformation :

```
void glMatrixMode(GLenum mode);
```

Matrice de transformation
- utilisée pour modifier les objets — GL_MODEL_VIEW

Matrice de visualisation
- utilisée pour projeter les objets sur l'écran — GL_PROJECTION

197

Initialisation

- La fonction suivante permet d'initialiser la matrice courante avec la valeur de la matrice identité :

```
void glLoadIdentity(void)
```

```
...
glMatrixMode(GL_MODEL_VIEW);
glLoadIdentity();
...
```

La matrice courante devient la matrice de transformation. On utilise alors les valeurs de cette matrice qui ont été stockées précédemment dans le contexte graphique.

La matrice courante est réinitialisée avec les valeurs de la matrice identité.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

198

Les translations

- Fonction :
  - glTranslatef(GLfloat tx, GLfloat ty, GLfloat tz)
    - Translation de vecteur (tx, ty, tz)
  - Multiplie la matrice de transformation courante par la matrice de translation correspondante
- Exemples

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(2.0, 0.0, 0.0);
...
```

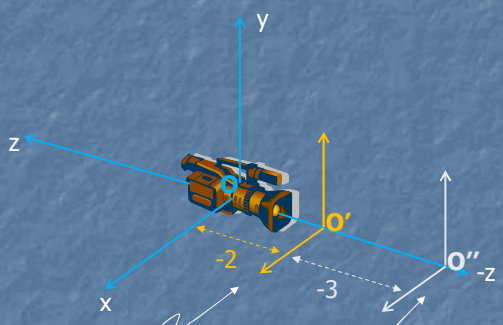
Tout ce qui suit sera translaté de 2 en x

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(2.0, 0.0, 0.0);
...
...
glTranslatef(3.0, 0.0, 0.0);
...
...
```

Tout ce qui suit sera translaté de **5** en x (composition des 2 translations)

Schéma de l'exemple

```
glMatrixMode(GL_MODEL_VIEW);
glLoadIdentity();
glTranslatef(0.0, 0.0, -2.0);
...
...
glTranslatef(0.0, 0.0, -3.0);
...
...
```



Repère dans lequel seront créés les objets 3D après le glTranslatef(0.0, 0.0, -2.0);

Repère dans lequel seront créés les objets 3D après le glTranslatef(0.0, 0.0, -2.0); *ET le* glTranslatef(0.0, 0.0, -3.0);

200

Les rotations

- Fonctions :
  - glRotatef(GLfloat alpha, GLfloat x, GLfloat y, GLfloat z);
    - Rotation
      - D'angle alpha (en degrés)
      - d'axe de vecteur directeur (x, y, z)
  - Multiplie la matrice de transformation courante par la matrice de rotation correspondante
- Exemples :

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glRotatef(45.0, 1.0, 0.0, 0.0);
...
```

Tout ce qui suit sera tourné de 45° autour autour de l'axe Ox

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glRotatef(45.0, 1.0, 0.0, 0.0);
...
...
glRotatef(30.0, 0.0, 1.0, 0.0);
...
...
```

Tout ce qui suit sera tourné de 45° autour de l'axe 0x, puis de 30° autour de Oy (composition des 2 rotations)

Les mises à l'échelle

- Fonctions :
  - glScalef(GLfloat sx, GLfloat sy, GLfloat sz);
    - Mise à l'échelle de facteurs sx, sy et sz
  - Multiplie la matrice de transformation courante par la matrice de mise à l'échelle correspondante
- Exemples :

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glScalef(1.0, 2.0, 1.0);
...
```

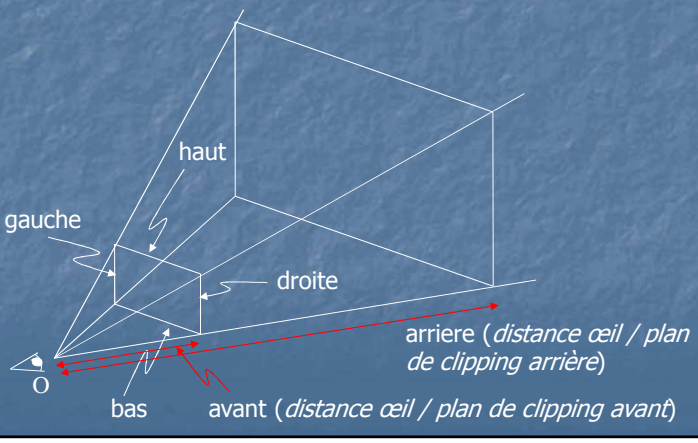
Tout ce qui suit verra sa hauteur multipliée par 2

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glScalef(1.0, 2.0, 1.0);
...
...
glScalef(1.0, 3.0, 1.0);
...
...
```

Tout ce qui suit verra sa hauteur multipliée par 6 (composition des 2 mises à l'échelle : 2.0 x 3.0)

Les Projections perspectives (1)

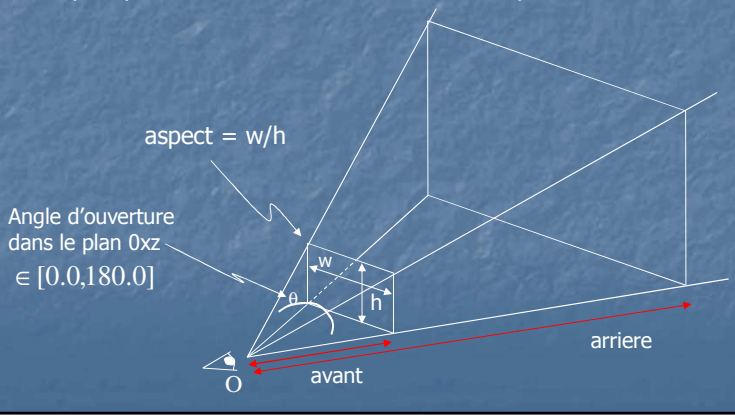
void glFrustum(GLdouble gauche, GLdouble droite,
GLdouble haut, GLdouble bas,
GLdouble avant, GLdouble arriere)



Les Projections perspectives (2)

void gluPerspective(GLdouble angle, GLdouble aspect,
GLdouble avant, GLdouble arriere)

Multiplie la matrice courante par une matrice de projection perspective, calculée en fonction des paramètres.

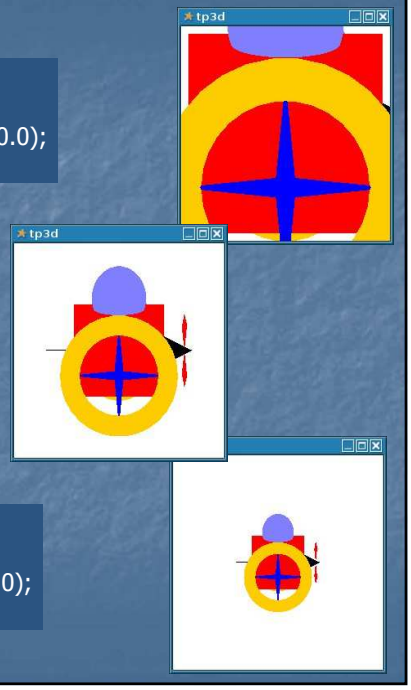


Exemples

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(30.0, 1.0, 1.0, 100.0);
...
```

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(60.0, 1.0, 1.0, 100.0);
...
```

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(90.0, 1.0, 1.0, 100.0);
...
```



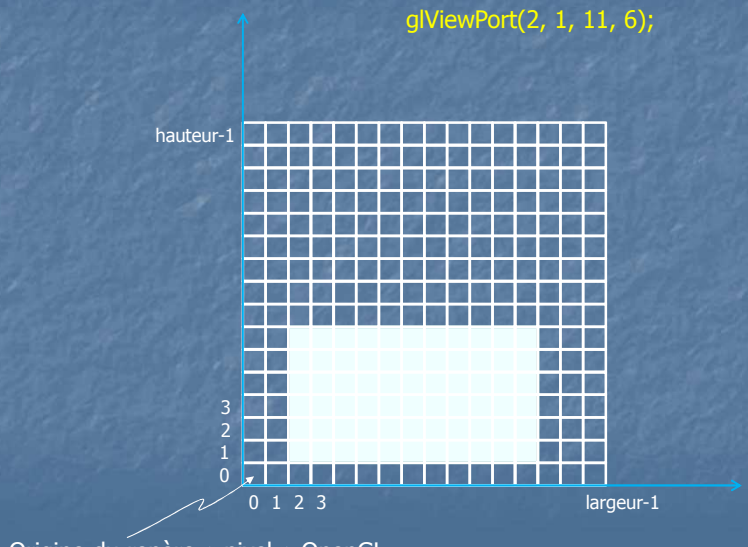
Définir une zone d'affichage

- OpenGL permet de restreindre la zone d'affichage à une sous-partie de la fenêtre d'affichage.
- Fonction à utiliser :

  void glViewPort(GLint xbas, GLint ybas,
  GLsizei largeur, GLsizei hauteur);
- Les paramètres représentent :
  - (xbas, ybas) : les coordonnées du coin inférieur gauche de la zone
  - largeur : la largeur de la zone
  - hauteur : la hauteur de la zone
- Tous les paramètres sont exprimés en coordonnées pixel.

206

Exemple



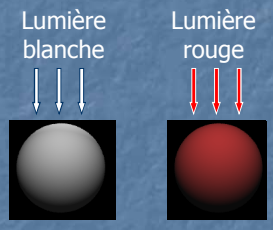
207

Plan du cours

1. Introduction
2. Modélisation d'objets 3D
3. Rendu temps réel
4. Introduction à OpenGL
5. Transformations géométriques
6. Modèles d'éclairage local
7. Rendu par lancer de rayons

Interactions lumineuses

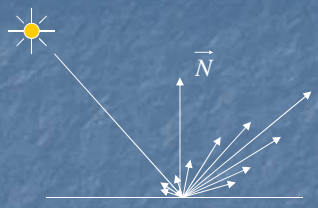
- Couleur observée dépend
  - Des sources de lumières
    - Distribution spectrale
    - Intensité
  - Des matériaux des objets
    - « couleur » de l'objet
    - Modes de réflexion
      - Mat (diffus)
      - Brillant (spéculaire)



Sphères blanches

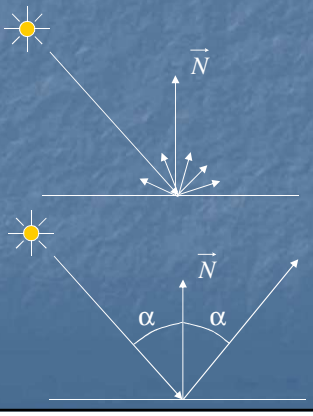
mat *brillant*

Modes de réflexion



- Cas général :
  - Réflexion dans toutes les directions
  - Intensité différente

- Approximations :
  - Réflexion diffuse
    - Même intensité dans toutes les directions
    - Intensité indépendante du point de vue
  - Réflexion spéculaire
    - Une seule direction de réflexion
    - Intensité dépendante du point de vue

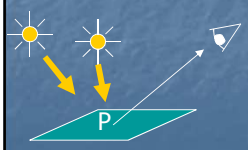


Calcul d'éclairage local

- Objectifs :
  - Calculer l'intensité de lumière reçue en un point P
  - Calculer l'intensité réfléchie vers l'observateur
- Simplifications
  - Ne pas tenir compte des autres objets
    - Pas d'ombres ni de réflexions
  - Modèle de réflexion de la lumière simplifié
    - Différents modèles (Gouraud, Phong, Ward, …)

Le modèle d'éclairage de Phong

- Modèle empirique (1973)
  - Résultat visuellement convaincant
  - Coût de calcul limité - Utilisé par OpenGL
- Expression
  - Soit I(P) l'intensité lumineuse réfléchie vers l'observateur au point P



$$I(P) = I_a(P) + I_d(P) + I_s(P)$$

Intensité réfléchie de manière spéculaire en P

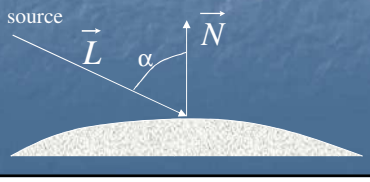
Intensité ambiante en P

Intensité réfléchie de manière diffuse en P

Intensité diffuse

- Hypothèses
  - L'intensité reçue depuis une source varie avec l'angle d'incidence
    - Lumière rasante : intensité faible
    - Lumière zénithale : intensité maximale
  - La quantité de lumière réfléchie
    - Est indépendante de l'angle de réflexion (réflexion diffuse)
    - Dépend des propriétés de réflectance du matériau

On pose : $I_d(P) = K_d . I . (\vec{N} . \vec{L}) = K_d . I . \cos(\alpha) \quad avec \quad 0 \le \alpha \le \frac{\pi}{2}$



Coefficient diffus de l'objet

Intensité de la source

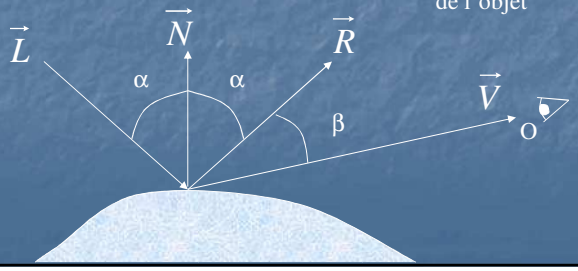
Intensité spéculaire

- Hypothèses :
  - La quantité de lumière réfléchie
    - dépend de l'angle de réflexion
    - Est maximale dans la direction spéculaire pure
    - S'atténue plus ou moins vite dans les autres directions en fonction du degré de brillance du matériau

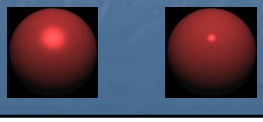
On pose : $I_s(P) = K_s . I . (\vec{V} . \vec{R})^s = K_s . I . \cos^s(\beta)$

Coefficient spéculaire de l'objet

Intensité de la source



S = indice de brillance
Matériau terne : s petit (1-10)
Matériau brillant : s grand (10-100)



Intensité ambiante

- Points non directements éclairés : noirs !!!
- Dans la réalité :
  - Nombreuses réflexions de lumière éclairent ces points
  - Difficiles à prendre en compte
- Eclairage ambiant :
  - permet d'associer un éclairage « global » aux parties d'objets non directement éclairées par une source

On pose $Ia(P) = Ka \cdot Isa$

Coefficient ambiant de l'objet (r,v,b)

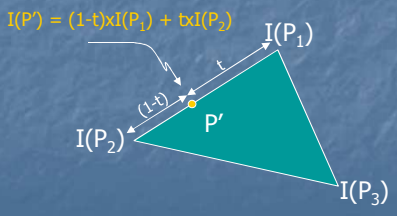
Intensité de la source ambiante (r,v,b) (constante commune à tous les objets)

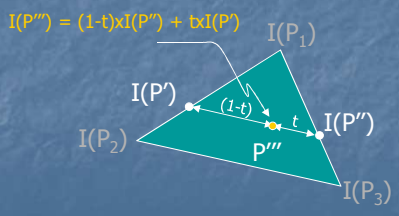
Méthodes d'interpolation

- Problématique
  - Calcul de I(P) en chaque point projeté sur l'écran
    - Calcul coûteux
    - Nombreux points invisibles
      - Élimination des parties cachées = dernière étape du pipeline de rendu
      - Calcul fait même pour les points qui ne seront pas visibles au final
  - Nécessité de réduire le coût de calcul
    - Faire un calcul exact au sommet des facettes
    - Interpoler les valeurs à l'intérieur des facettes

Interpolation de Gouraud (1)

- Principe
  - Calculer l'intensité au sommet des facettes
  - Interpoler les intensités à l'intérieur des facettes





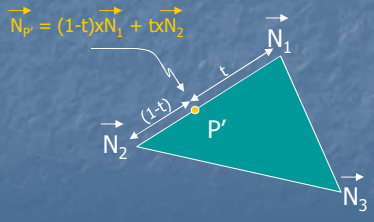
Interpolation de Gouraud (2)

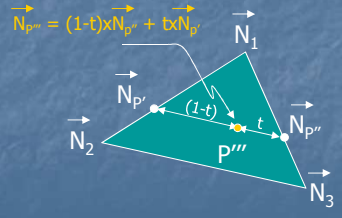
- Avantage :
  - Réduction importante du coût de calcul
- Inconvénients
  - interpolation linéaire sur les intensités

  => pas d'effets spéculaires
  - effets spéculaires :
    - effets non linéaires
    - localisés autour d'un point

Interpolation de Phong (1)

- Principe
  - Interpoler les normales
  - Effectuer le calcul exact en tout point





Interpolation de Phong (2)

- Avantage
  - Bonne représentation des réflexions spéculaires
- Inconvénient
  - Calcul beaucoup plus complexe
    - Évaluer $\cos^s(\beta)$ en tout point

⇒ OpenGL : Interpolation de Gouraud

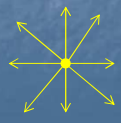
OpenGL et éclairage

- Par défaut :
  - OpenGL n'utilise pas les fonctionnalités liées à l'éclairage
  - Couleurs des objets définies par glColor3f (par exemple)
- Activation : glEnable(GL_LIGHTING)
  - Les couleurs définies par glColor ne sont plus utilisées
  - Nécessité de :
    - définir un matériau pour chaque objet
    - Définir une ou plusieurs sources
- Désactivation : glDisable(GL_LIGHTING)
  - Les couleurs définies par glColor sont utilisées

221

Les sources OpenGL

- OpenGL permet de définir :
  - 8 sources ponctuelles isotropes ou de type spot
    - Chaque source éclaire les points qui sont orientés vers celle-ci (utilisation de la normale)
    - Les points orientés en sens inverse ne sont pas éclairés
  - 1 éclairage ambiant
    - Permet d'éclairer les points non orientés vers une source
    - Agit même sans aucune des 8 sources ponctuelles
  - Modèle de couleur (rouge, vert, bleu)



Source isotrope

Spot

222

Source ambiante

- Valeurs par défaut : (0.2f, 0.2f; 0.2f, 1.0f)
- Modification :

  void glLightModelfv(GLint prop, GLfloat[] val)

  - Avec :
    - prop : la constante symbolique GL_LIGHT_MODEL_AMBIENT
    - val : un vecteur de 4 réels, qui représentent la couleur en (R,V,B,alpha)
      - Alpha = transparence ; non utilisée ici ; à positionner à 1.0

```
GLfloat valeur[] = {0.3, 0.3, 0.0, 1.0};
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, valeur)
```

Éclairage ambiant = « jaune pâle »

223

Les sources ponctuelles

- 8 sources distinctes possibles
  - Noms prédéfinis :

    GL_LIGHT0, GL_LIGHT1, … GL_LIGHT7
  - Activation :

    glEnable(GL_LIGHTn)
    - Avec n le numéro de la source à activer
  - Désactivation :

    glDisable(GL_LIGHTn)
    - Avec n le numéro de la source à activer

224

Les paramètres d'une source

- Retour sur le modèle de Phong

$$I(P) = K_a.I + K_d.\cos(\alpha).I + K_s.\cos^s(\beta).I$$

I = Intensité de la source – la même pour les 3 termes

- En OpenGL

$$I(P) = K_a.I_a + K_d.\cos(\alpha).I_d + K_s.\cos^s(\beta).I_s$$

Définir 3 intensités différentes pour chaque source

Initialisation d'une source

void glLightfv(GLenum source,
GLenum param,
float *valeurs)

Valeurs à affecter

GL_LIGHT0
...
GL_LIGHT7

« Nom » de la source à initialiser (8 sources disponibles)

GL_AMBIENT
GL_DIFFUSE
GL_SPECULAR

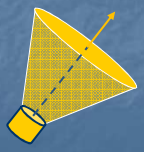
Intensités

+

GL_POSITION

Position

+



GL_SPOT_DIRECTION
GL_SPOT_EXPONENT
GL_SPOT_CUTOFF

Paramètres pour une source de type spot

GL_CONSTANT_ATTENUATION
GL_LINEAR_ATTENUATION
GL_QUADRATIC_ATTENUATION

Lois d'atténuation de l'intensité en fonction de la distance

Principales valeurs par défaut

GL_LIGHT0

GL_AMBIENT	(0.0, 0.0, 0.0, 1.0)
GL_DIFFUSE	(1.0, 1.0, 1.0, 1.0)
GL_SPECULAR	(1.0, 1.0, 1.0, 1.0)
GL_POSITION	(0.0, 0.0, 1.0, 0.0)

GL_AMBIENT	(0.0, 0.0, 0.0, 1.0)
GL_DIFFUSE	(0.0, 0.0, 0.0, 1.0)
GL_SPECULAR	(0.0, 0.0, 0.0, 1.0)
GL_POSITION	(0.0, 0.0, 1.0, 0.0)

GL_LIGHT1
à
GL_LIGHT7

Exemples

```
GLfloat pos[] = {0.0, 0.0, 2.0};
GLfloat diffus[] = {..., ..., ..., 1.0};
GLfloat specu[] = {..., ..., ..., 1.0};

glLightfv(GL_LIGHT0, GL_POSITION, pos);
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffus);
glLightfv(GL_LIGHT0, GL_SPECULAR, specu);

glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
```

```
GLfloat diffus[] = {1.0, 1.0, 1.0, 1.0};
GLfloat specu[] = {1.0, 1.0, 1.0, 1.0};
```

```
GLfloat diffus[] = {0.2, 0.2, 0.2, 1.0};
GLfloat specu[] = {1.0, 1.0, 1.0, 1.0};
```

```
GLfloat diffus[] = {1.0, 1.0, 1.0, 1.0};
GLfloat specu[] = {0.2, 0.2, 0.2, 1.0};
```

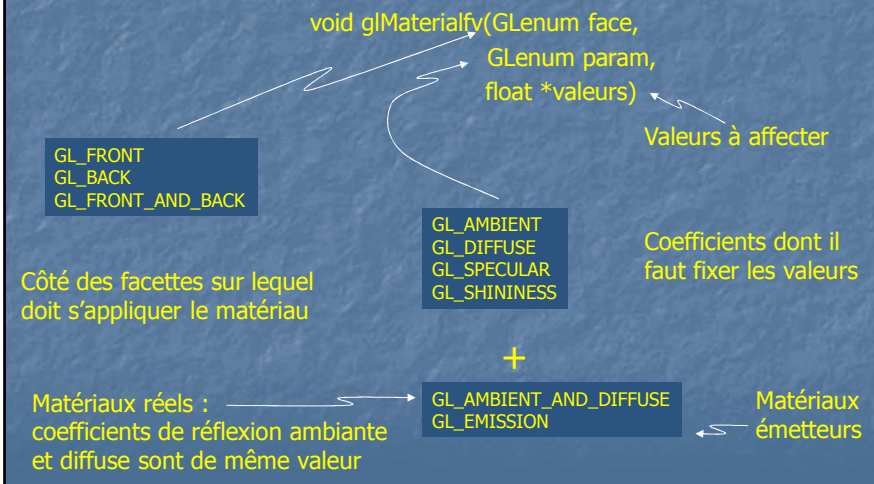
Les matériaux OpenGL

- Correspondent aux paramètres de réflexion du modèle de Phong

$$I(P) = K_a . I_a + K_d . \cos(\alpha) . I_d + K_s . \cos^s(\beta) . I_s$$

  - Kd : coefficient de réflexion diffuse
  - Ks : coefficient de réflexion spéculaire
  - Ka : coefficient de réflexion ambiante
  - S : indice de brillance
- Les coefficients de réflexion dépendent de la longueur d'onde
  - 3 valeurs dans le modèle (R,V,B)

Création d'un matériau



Exemples

```
GLfloat mat_amb_diff[] = {0.1, 0.5, 0.8, 1.0};
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, mat_amb_diff);
```

```
GLfloat mat_amb_diff[] = {0.1, 0.5, 0.8, 1.0};
GLfloat mat_spec[] = {1.0, 1.0, 1.0, 1.0};
GLfloat brillance[] = {10.0};
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, mat_amb_diff);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_spec);
glMaterialfv(GL_FRONT, GL_SHININESS, brillance);
```

Remarques

- Validité d'un matériau
  - Sur toutes les primitives géométriques qui suivent sa définition ; jusqu'à définition d'un nouveau matériau
- Normales
  - Nécessaires aux sommets
- Brillance

```
GLfloat brillance = 10.0;
glMaterialf(GL_FRONT, GL_SHININESS, brillance)
```

- Matériaux par défaut

GL_AMBIENT	(0.2,0.2,0.2,1.0)	GL_SPECULAR	(0.0,0.0,0.0,1.0)
GL_DIFFUSE	(0.8,0.8,0.8,1.0)	GL_SHININESS	0.0
GL_AMBIENT_AND_DIFFUSE	(0.8,0.8,0.8,1.0)	GL_EMISSION	(0.0,0.0,0.0,1.0)

Les normales OpenGL

- Les calculs d'éclairage supposent :
  - Que les normales soient connues
  - Qu'elles soient normées (leur norme doit être égale à 1)
- Une normale est une variable d'état dans le contexte graphique OpenGL
- Définition d'une normale courante :

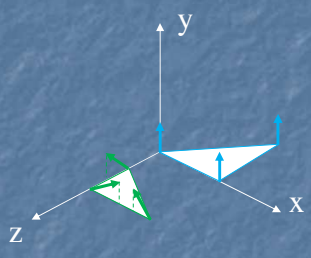
  void glNormal3f(GLfloat nx, GLfloat ny, GLfloat nz)

  void glNormal3fv(GLfloat *vecteur)
- Avec :
  - nx, ny, nz : les coordonnées réelles d'une normale
  - vecteur : un tableau de 3 nombres réels qui représentent les coordonnées d'une normale

233

Exemples

```
glBegin(GL_TRIANGLES);
  glNormal3f(0.0, 0.7, 0.7);
  glVertex3f(0.0, 0.0, 0.5);
  glNormal3f(0.7, 0.7, 0.0);
  glVertex3f(0.0, 0.0, 1.0);
  glNormal3f(-0.7, 0.7, 0.0);
  glVertex3f(1.0, 0.0, 1.0);
glEnd();
```




```
glBegin(GL_TRIANGLES);
  glNormal3f(0.0, 1.0, 0.0);
  glVertex3f(0.0, 0.0, 0.0);
  glVertex3f(1.0, 0.0, 0.0);
  glVertex3f(1.0, 0.0, -1.0);
glEnd();
```

Remarque

Pour conserver des normales normées automatiquement, il suffit, lors de l'initialisation de l'application, d'exécuter l'instruction suivante :

glEnable(GL_NORMALIZE)

234