
# Transformations - Eclairage
# Textures - GLES 2.0

Jean-Philippe Farrugia@univ-lyon1.fr

# Plan

- Notions de bases de synthèse d'images.
- Le pipeline graphique et Open GL / GL ES.
- Modélisation et transformations.
- Eclairages et textures.
- Open GL ES 2.0 / Shaders.

# Transformations

- Nécessaires pour :
  - Modifier un objet.
  - Changer de repère.
- Notions d'algèbre linéaire nécessaires.
  - Vecteurs, matrices.

# Transformations

- Translations
- Rotations
- Mise à l'échelle
- Changement de repères ?

# Transformations

- Formulation matricielle des transformations.
  - Translation : Modélisable par addition d'un vecteur...
    - Pas souhaitable.
- Coordonnées homogènes
  - 4eme coordonnée : 1 pour point, 0 pour vecteur.
  - Pour toutes les transformations : produit de matrice.

# Translation

- Translation T d'un point p par un vecteur $\vec{u}$ $(u_1, u_2, u_3)$.
- T(x) = x
- T(y) = y
- T(z) = z
- T(O) = O + u

$$T = \begin{bmatrix} 1 & 0 & 0 & u_1 \\ 0 & 1 & 0 & u_2 \\ 0 & 0 & 1 & u_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Rotation

Cas particulier : rotation autour des axes du repère.
Axe quelconque ?

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Transformations

- Mise à l'échelle ?
- Projections ?
  - Paramètres ?
  - Matrice ?
- Composition de transformation ?

# Transformations OpenGL

- Dans GLES 1.0, les transformation usuelles sont prédéfinies.
- Transformation courante : état
  - Matrice MODELVIEW.
- Transformations empilées.
- Translation, Rotation, Mise à l'échelle pré-définies.
- Transformation quelconque : chargement d'une matrice 4x4.

# Transformations OpenGL

- Concrètement :
  - glTranslate, glRotate, glScale.
  - glLoadMatrix.
- Les transformations sont automatiquement composées.
- Pour empiler/dépiler la valeur courante :
  - glPushMatrix() / glPopMatrix().

# Exercice 3

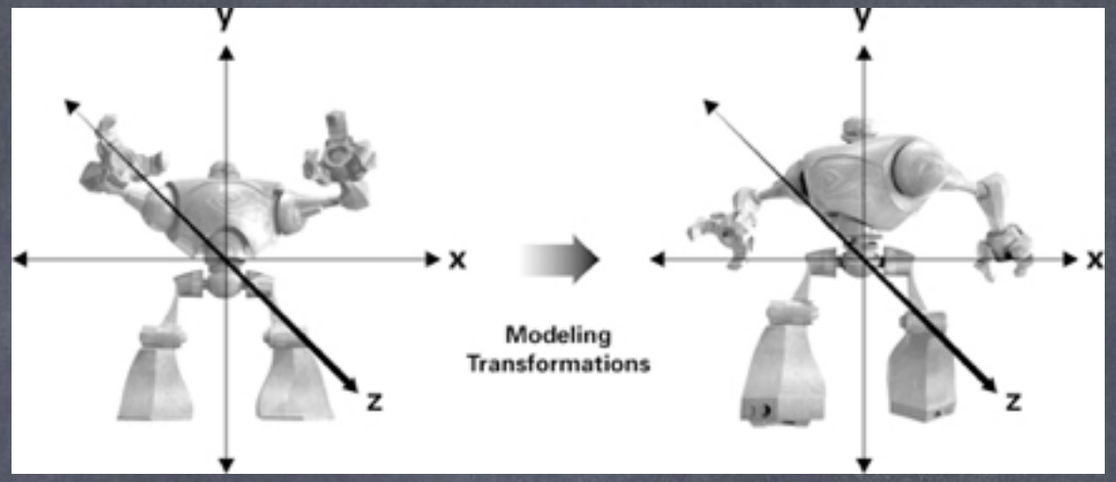
- Modifiez votre programme pour faire tourner l'objet affiché.
  - En touchant l'écran ?
    - Ajoutez l'instruction suivante dans l'initialisation :
      - gl.glDisable(GL10.GL_CULL_FACE);
- Modélisez un système solaire : un petit objet en rotation sur lui même, qui tourne autour d'un gros.

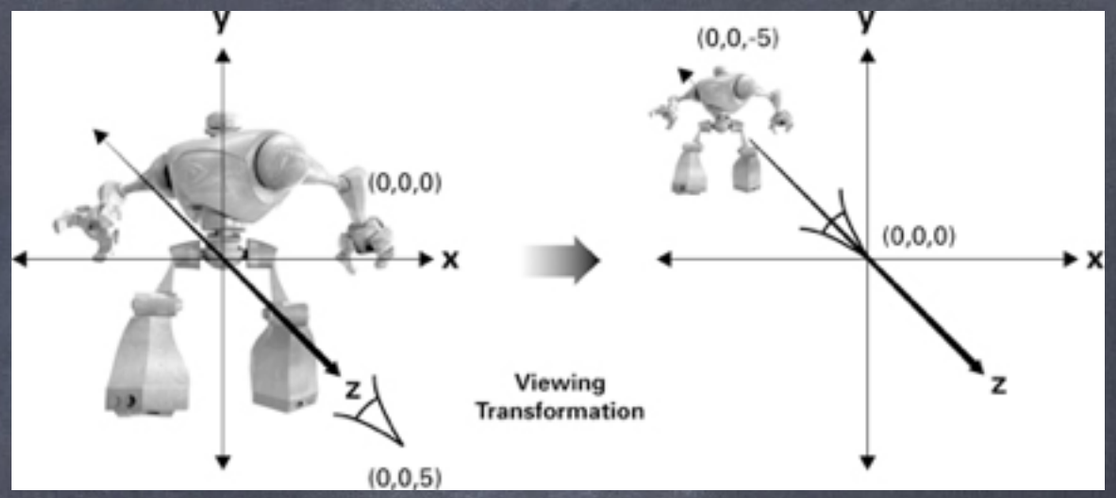
# Repères

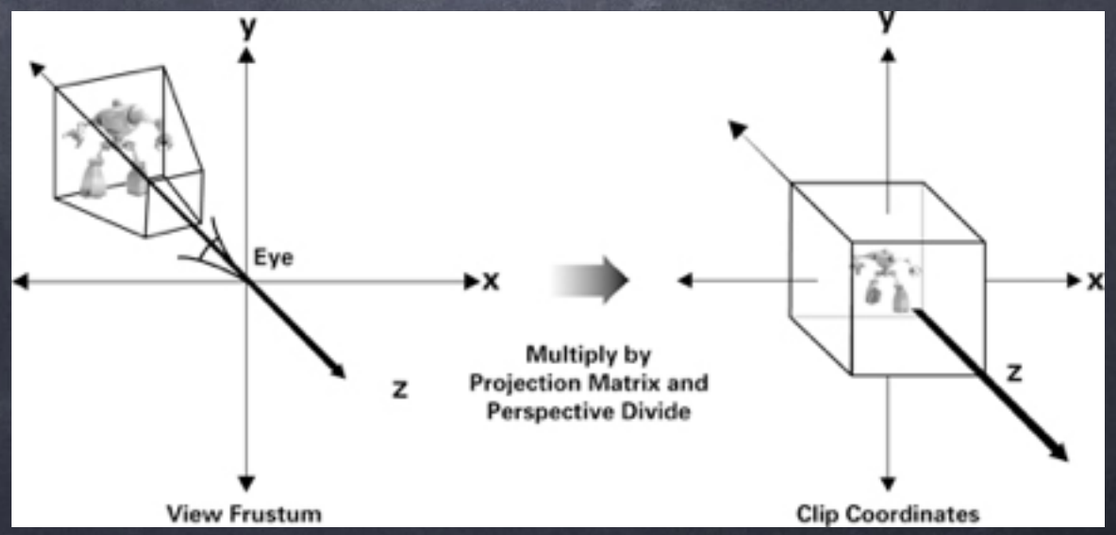
- Dans la définition d'une scène : plusieurs repères.
  - Object : celui de l'objet.
  - Scène : celui de... la scène.
  - Observateur : celui de la caméra.
  - Espace projectif : celui de l'écran, avant discrétisation.

# Repères

Les différents repères et leur transformations associées.



# Changements de repères

- Transformations pour passer d'un repère à l'autre :
  - Objet -> Scene : Model (modélisation).
  - Scene -> Observateur : View (visualisation).
  - Observateur -> Espace projectif : Projection.
  - Espace projectif -> Fenêtre : Viewport.
- NB : ces transformations sont matricielles.

# Dans OpenGL

- Transformations pour les changements de repères : fixées par un état...
  - GL_MODELVIEW pour model et view.
    - NB : OpenGL ne distingue pas les transformations de modélisation et de visualisation.
  - GL_PROJECTION pour projection.
  - glMatrixMode pour changer l'état.
- viewport fixée avec glViewport.

# Dans OpenGL

- Avec OpenGL seul : changements de repères définis «manuellement».
- Bibliothèque utilitaire : GLU
- Pour définir la transformation de visualisation : gluLookAt.
- Pour définir la projection : gluPerspective ou gluOrtho.
  - Eventuellement : glFrustum.

# Exercice 4

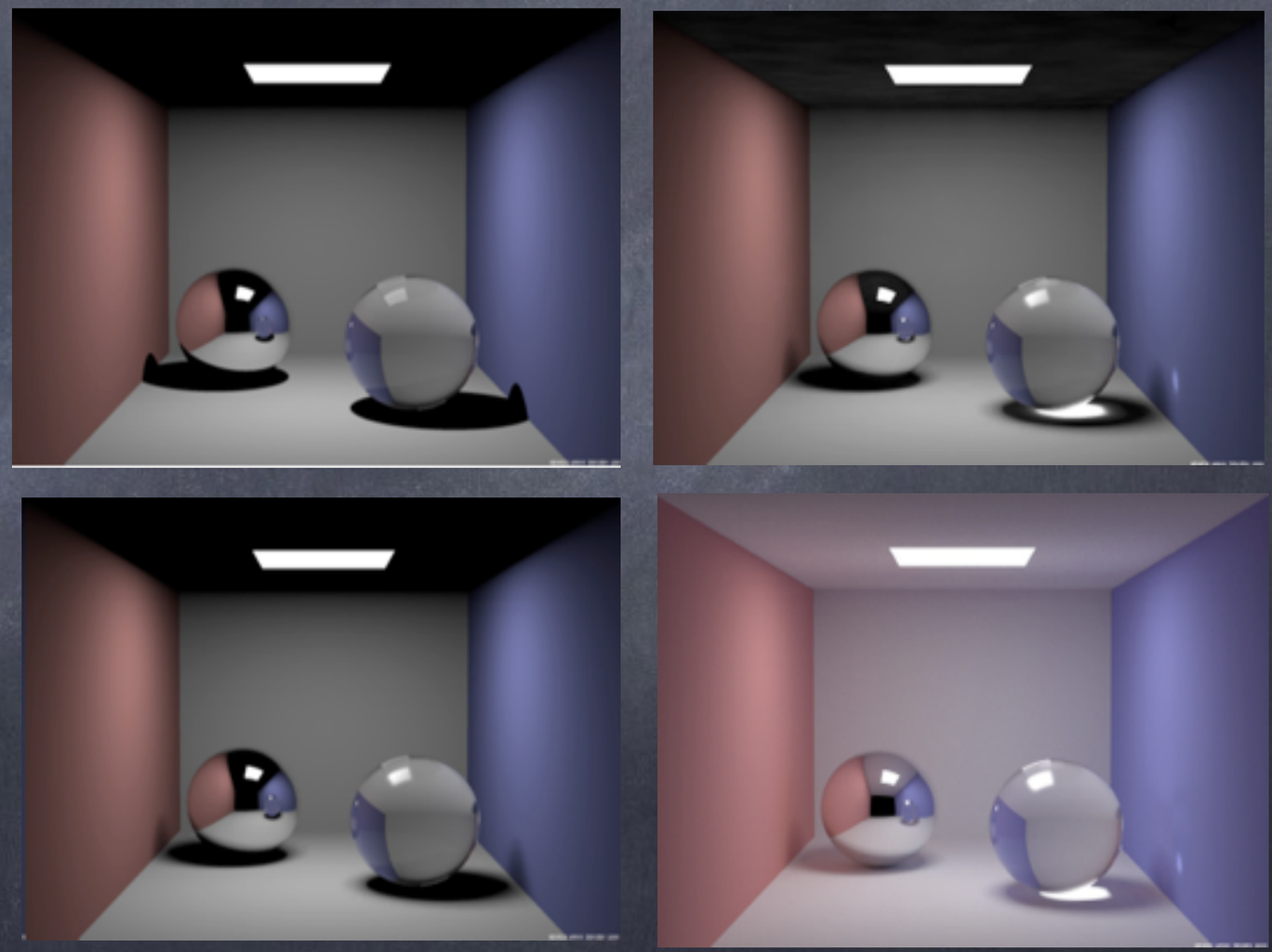
- Repérez l'affectation de MODELVIEW, PROJECTION et du viewport dans votre code actuel.
- Modifiez votre programme actuel pour déplacer la caméra autour de votre système solaire.

# Plan

- Notions de bases de synthèse d'images.
- Le pipeline graphique et Open GL / GL ES.
- Modélisation et transformations.
- Eclairages et textures.
- Open GL ES 2.0 / Shaders.

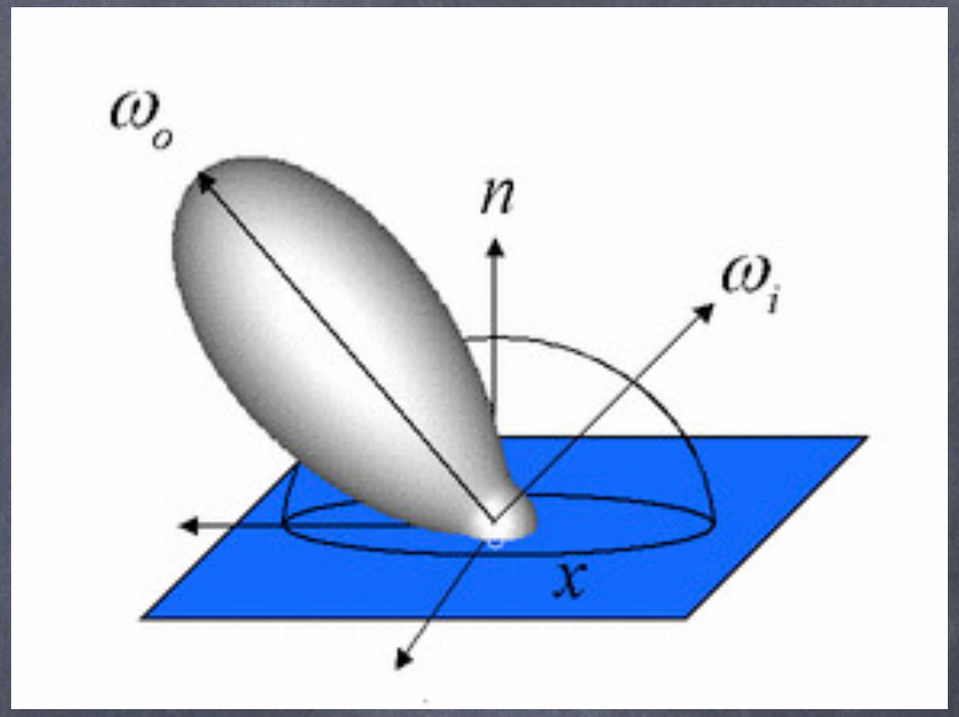
# Eclairage

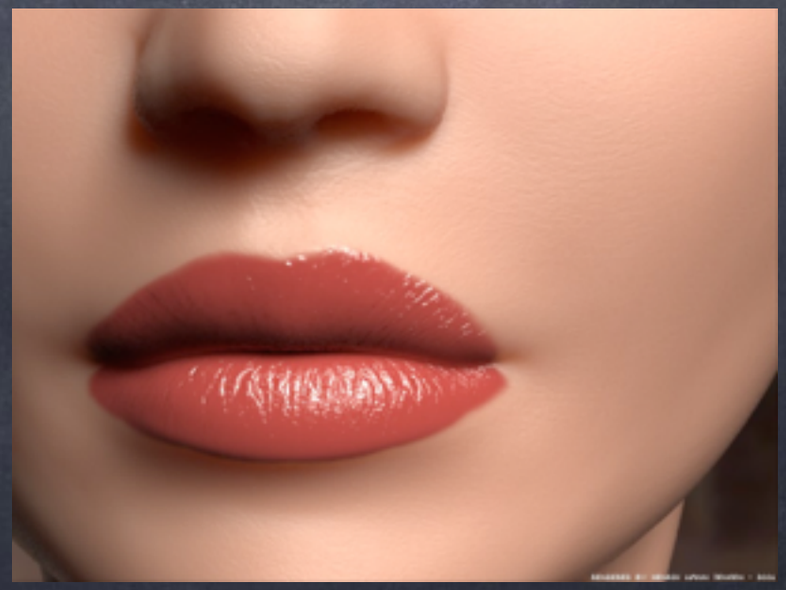
- Indispensable pour un rendu réaliste.
- Interaction lumière-matière.
- Implique :
  - La définition des sources de lumières.
  - La définition des matériaux.



# Matière

- Caractériser l'interaction.
  - Fonction de refléctance.
  - Fonction de transmittance.
- Permet la définition de matériaux.

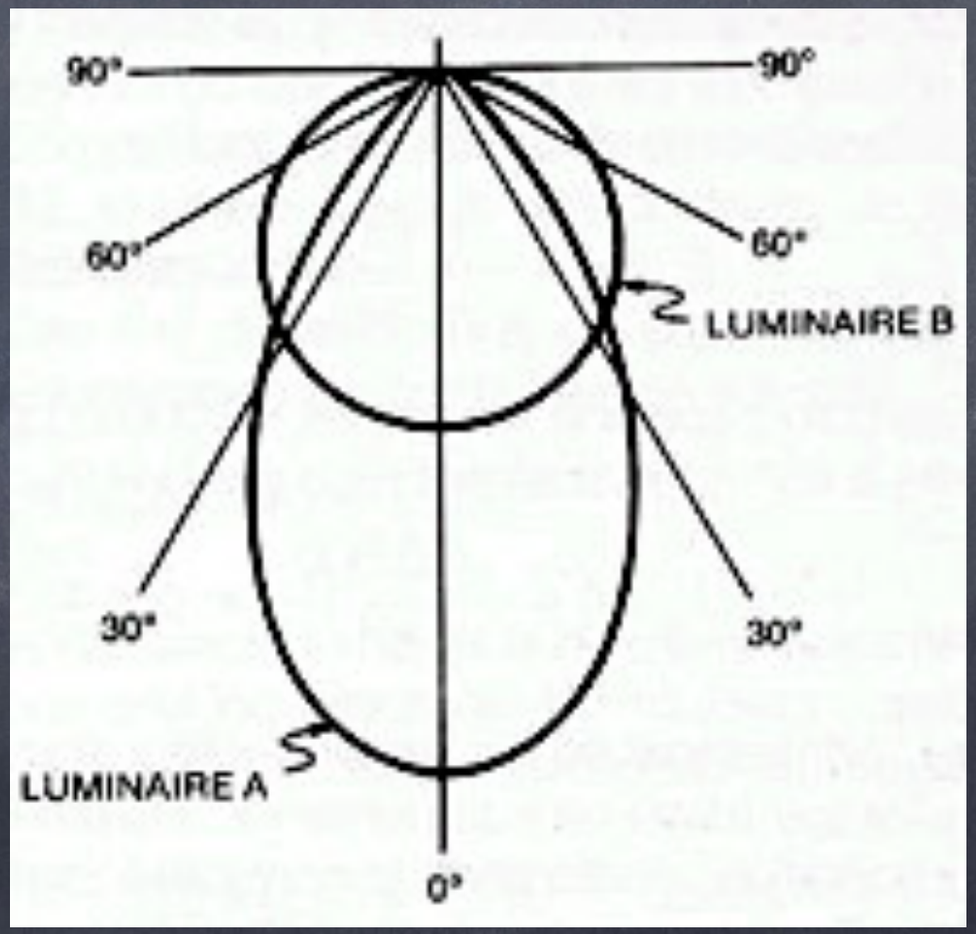


# Lumière

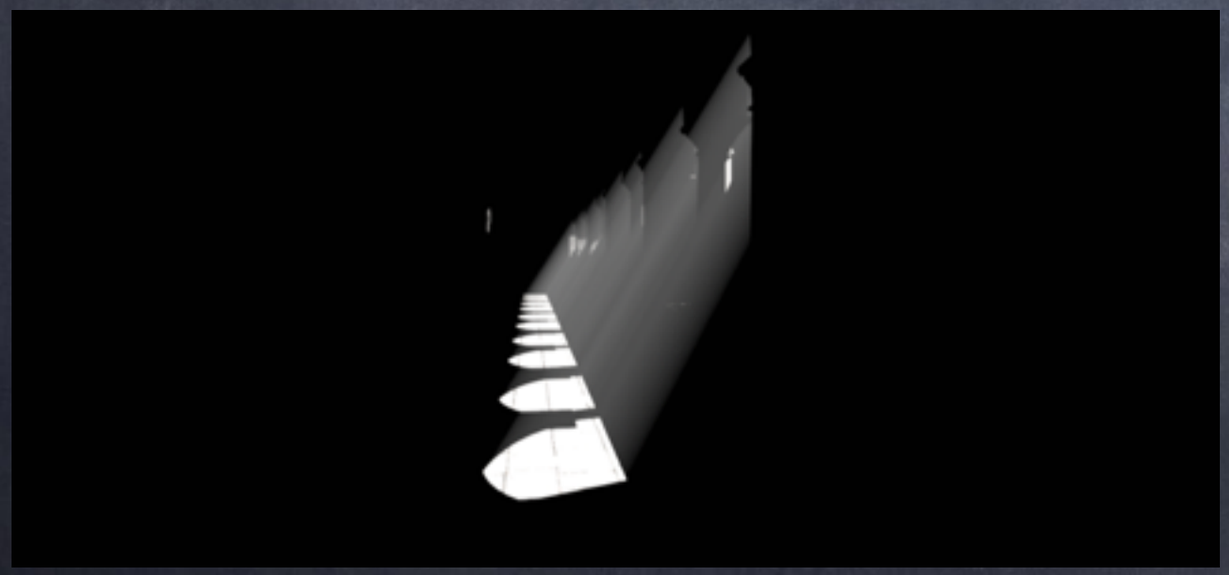
- Définition similaire à la BRDF.

- NB : n'importe quel élément de la scène peut être une source de lumière.



# Eclairer une scène

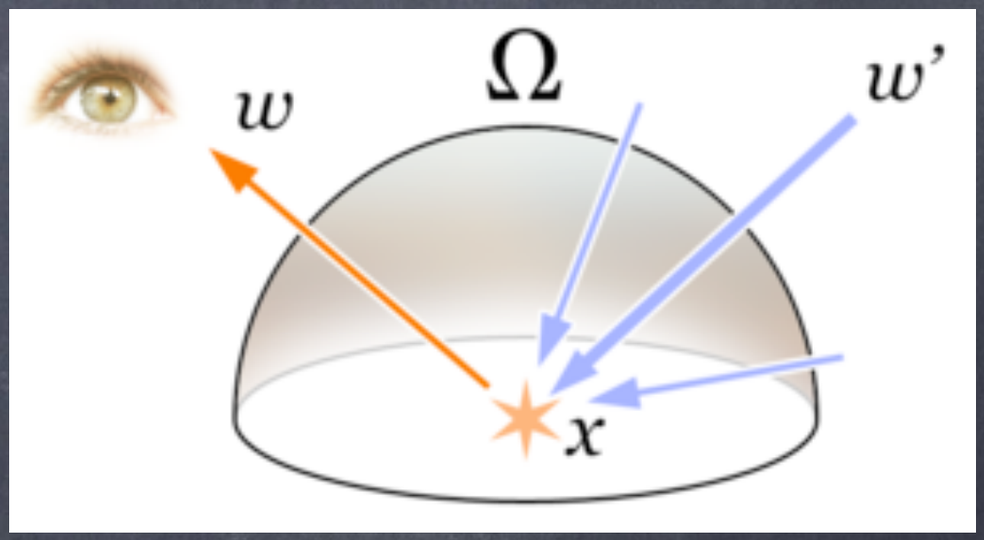
- Calculer l'ensemble des interactions entre les éléments de la scène.
- Problème ambitieux...



# Eclairer une scène

- Equation de Kajiya :

$$L_o(x, \mathbf{w}, \lambda, t) = L_e(x, \mathbf{w}, \lambda, t) + \int_\Omega f_r(x, \mathbf{w}', \mathbf{w}, \lambda, t) L_i(x, \mathbf{w}', \lambda, t)(-\mathbf{w}' \cdot \mathbf{n}) d\mathbf{w}'$$




- Lambda : longueur d'onde.
- t : temps.
- fr : BRDF de la surface.
- Impossible à résoudre en temps réel...

# En temps réel :

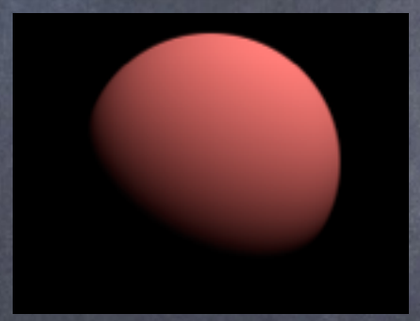
- Simplification :
  - BRDF décomposée en composantes simples.
  - Sources lumineuses ponctuelles.
  - Eclairement : interactions directes uniquement.
    - Et encore...

# Matériaux

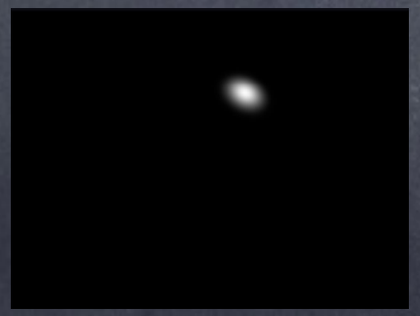
- BRDF définie selon trois composantes :
  - Ambiant, diffus, spéculaire.
  - Eventuellement, émission.

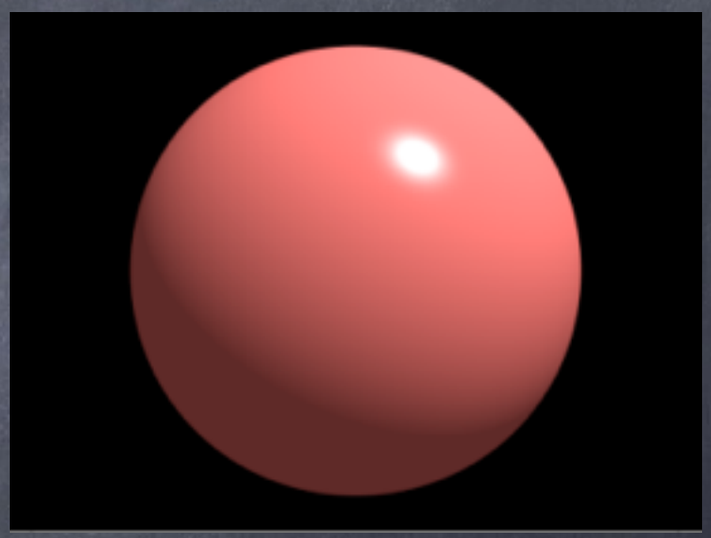
Ambiant

Diffus



Spéculaire





# Avec OpenGL

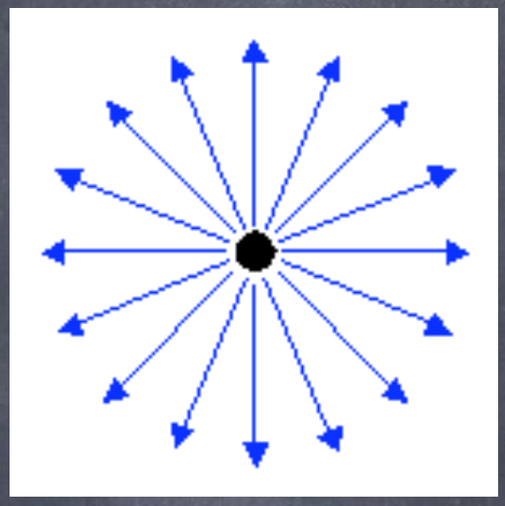
- Définition des matériaux :
  - Ambiant, diffus, spéculaire et émission.
  - Chacun est décrit par une couleur RGBA.
  - Défini avec glMaterial.
  - Activé avec glEnable en même temps que l'éclairage.

# Matériaux OpenGL

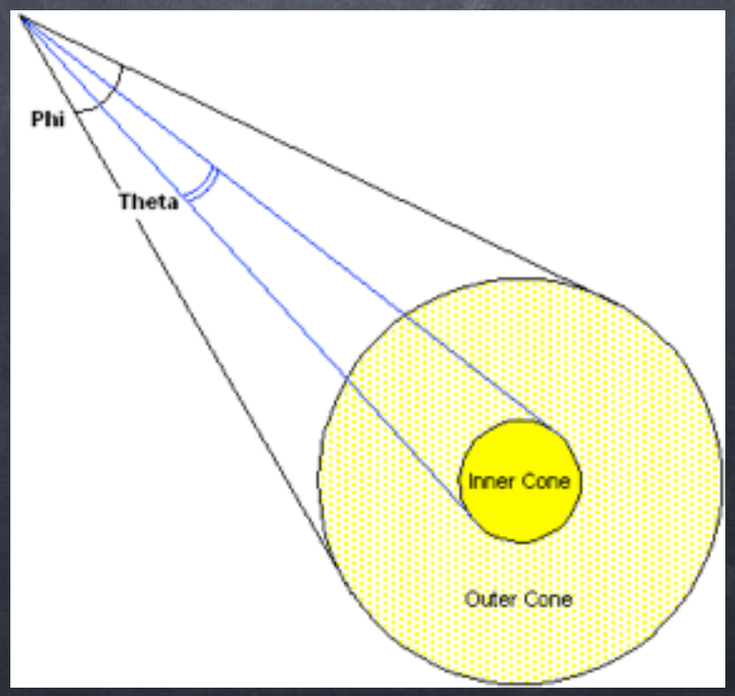
```
   // Definition des matériaux
private float mat_diffuse[] = { 1.0f, 1.0f, 0.0f, 1.0f };
private float mat_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
private float mat_shininess[] = { 20.0f };

gl.glMaterialfv(GL10.GL_FRONT_AND_BACK,
                GL10.GL_DIFFUSE,
                FloatBuffer.wrap(mat_diffuse));

gl.glMaterialfv(GL10.GL_FRONT_AND_BACK,
                GL10.GL_SPECULAR,
                FloatBuffer.wrap(mat_specular));

gl.glMaterialfv(GL10.GL_FRONT_AND_BACK,
                GL10.GL_SHININESS,
                FloatBuffer.wrap(mat_shininess));
```

# Sources de lumières

- Uniforme :
  - Lumière émise avec la même intensité dans toutes les directions.



- Directionnelle :
  - Lumière émise dans une direction et un angle précis.



# Avec OpenGL

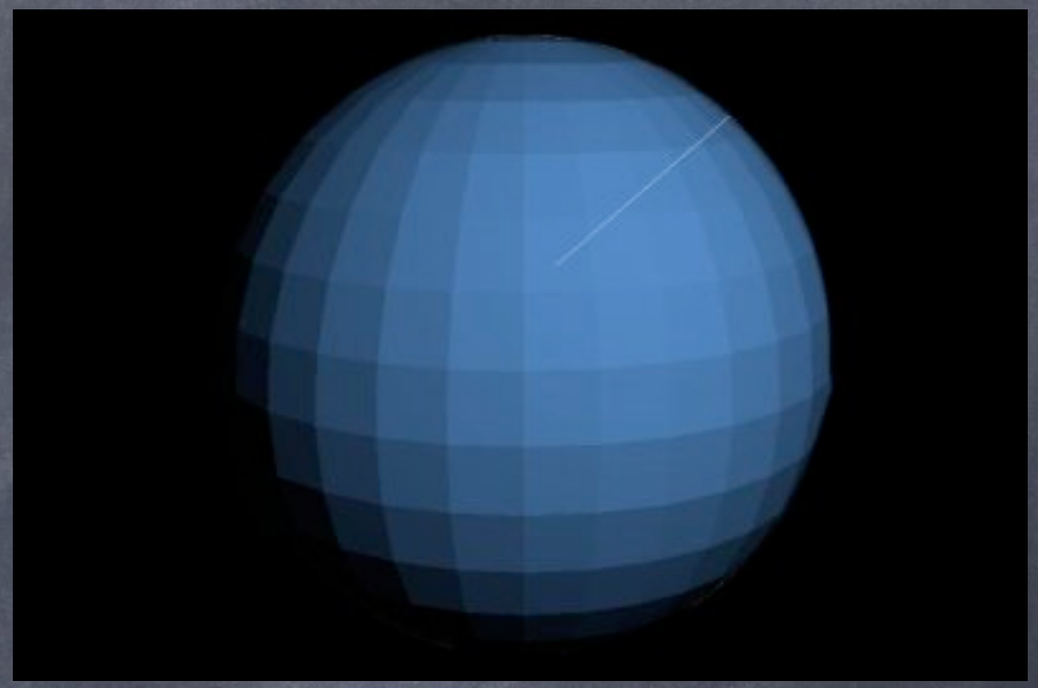
- Définition des lumières :
  - Définition avec glLight.
    - Position, couleur, type, atténuation...
  - Activation avec glEnable.
  - Possibilité de définir jusqu'à 8 sources.

# Avec OpenGL

```
// Definition d'une source de lumière
float light_position[] = { 0.0f,0.0f, 50.0f, 1.0f };
gl.glLightfv(GL10.GL_LIGHT0,
       GL10.GL_POSITION,
       FloatBuffer.wrap(light_position));

// Activation
gl.glEnable(GL10.GL_LIGHTING);
gl.glEnable(GL10.GL_LIGHT0);
```

# Calcul d'éclairage

- Nom usuel : «shading» :
  - Opération qui calcule la couleur des sommets.
- Nécessite généralement la normale au sommet.
- Souvent : interpolation bilinéaire entre chaque sommet.

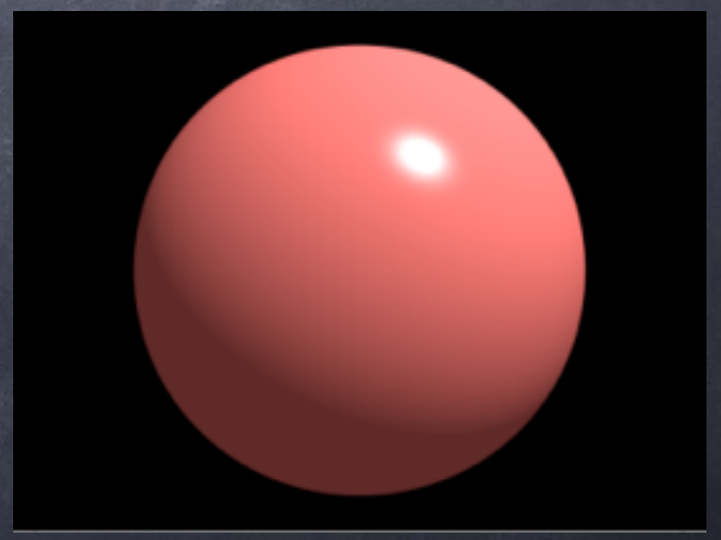


# Avec OpenGL

- Affectation de la normale :
  - En mode indexé : tableau d'attributs.

```
gl.glEnableClientState(GL10.GL_NORMAL_ARRAY);
gl.glNormalPointer(GL10.GL_FLOAT, 0, normalBuffer);
```

- Shading : pris en charge par le pipeline fixe.
  - Pour le modifier : shaders.
- Dans OpenGL : shading de «Phong».
  - Ambient, diffus, spéculaire.



# Exercice 5

- Pour cet exercice, il vous faut des objets avec normales
  - Instruction par l'intervenant.
- Modifiez votre programme précédent pour dessiner deux objets :
  - Le premier sera rouge et brillant.
  - Le deuxième sera vert et mat.
- Eclairez le tout judicieusement.

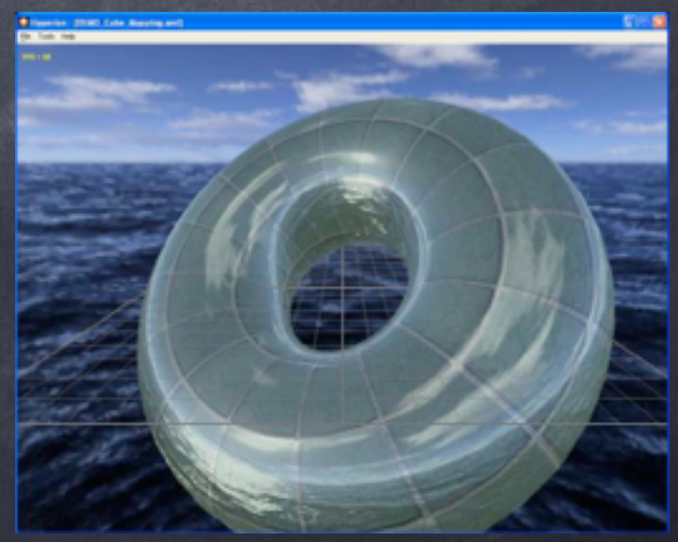
# Séance 2

- Transformations géométriques
- Visualisation et projections.
- Éclairage
- Textures

# Textures

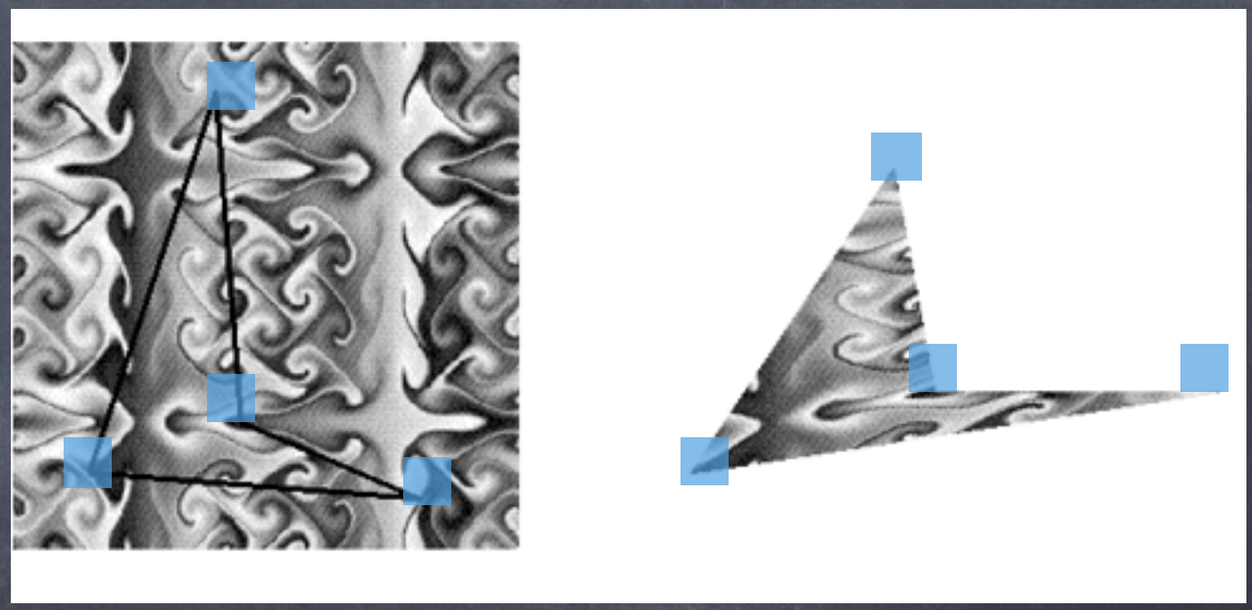
- Texture : procédé qui modifie la surface d'un objet pour lui donner une apparence particulière.


- Plusieurs types :
  - Textures algorithmiques générées.
  - Textures 2D, 3D, Cube map, surfaciques...



# Textures 2D

- Cas le plus courant :
  - Une texture est une image plaquée à la surface d'un objet.
  - Nécessite la paramétrisation de la surface de l'objet.
  - Pour chaque sommet : Coordonnée de texture.
    - Comprises entre 0 et 1.



# Textures

- Etapes :
  - Créer la texture.
  - Définir les coordonnées de textures pour l'objet.
  - Activer la texture pour l'objet concerné dans la boucle de rendu.

# Textures avec OpenGL

- Créer et parametrer une texture :

```
// Textures
IntBuffer myTexture = IntBuffer.allocate(1);
gl.glGenTextures(1, myTexture);
gl.glBindTexture(GL10.GL_TEXTURE_2D, myTexture.get(0));
gl.glTexImage2D(GL10.GL_TEXTURE_2D,
          0, GL10.GL_RGB, largeur_image, hauteur_image, 0,
          GL10.GL_RGBA, GL10.GL_UNSIGNED_BYTE, donnees_image);

// Paramétrage de la texture créée
gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MAG_FILTER, GL10.GL_NEAREST);
gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MIN_FILTER, GL10.GL_NEAREST);
gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_S, GL10.GL_CLAMP_TO_EDGE);
gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_T, GL10.GL_CLAMP_TO_EDGE);
```

# Textures avec OpenGL

- Alternative : utilisation de packages Android adaptés.

```
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.opengl.GLUtils;
```

```
// Chargement de l'image
Bitmap bitmapTexture = BitmapFactory.decodeFile("/sdcard/matexture.jpg");

myTexture = IntBuffer.allocate(1);
gl.glGenTextures(1, myTexture);
gl.glBindTexture(GL10.GL_TEXTURE_2D, myTexture.get(0));

// Paramétrage de la texture créée
gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MAG_FILTER, GL10.GL_NEAREST);
gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MIN_FILTER, GL10.GL_NEAREST);
gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_S, GL10.GL_CLAMP_TO_EDGE);
gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_T, GL10.GL_CLAMP_TO_EDGE);

GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmapTexture, 0);
```

# Textures avec OpenGL

- Dans la boucle de rendu :

```
gl.glEnable(GL10.GL_TEXTURE_2D);
gl.glBindTexture(GL10.GL_TEXTURE_2D, myTexture.get(0));

gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
// texCoordsBuffer a été initialisé précédemment avec les
coordonnées de textures
gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, texCoordsBuffer);
```

# Exercice 6

- Dessiner un quad avec une image plaquée dessus.
  - Quelles coordonnées de textures allez-vous affecter aux sommets ?
- Texturez un objet plus complexe.
  - Instructions données par l'intervenant.

# Exercice 7

- Reprenez votre système solaire de l'exercice 1 et faites les modifications suivantes :
- Texturez la surface des objets avec une image de «surface de planète».
- Tenez compte de l'éclairage : le «soleil» est une source de lumière.