# Développement mobile MIDP 2.0
# Mobile 3D Graphics API (M3G) – JSR 184

Frédéric BERTIN
fbertin@neotilus.com

# Présentaion :
# Mobile 3D Graphics API

# JSR 184 – M3G :présentation

- Package optionnel de l'api J2ME.
- Prend en considération la limitation des terminaux adressés par J2ME (beaucoup plus léger que Java3D)
- Assure un rendu assez rapide pour les jeux 3D, les animations, …
- Défini un format standard pour les graphes de scène (les fameux .m3g)
- De par la nature des calculs 3D, l'API nécessite de faire des calcules à virgule flottante
  - **Elle nécessite le CLDC 1.1**

# Les phases de la création 3D

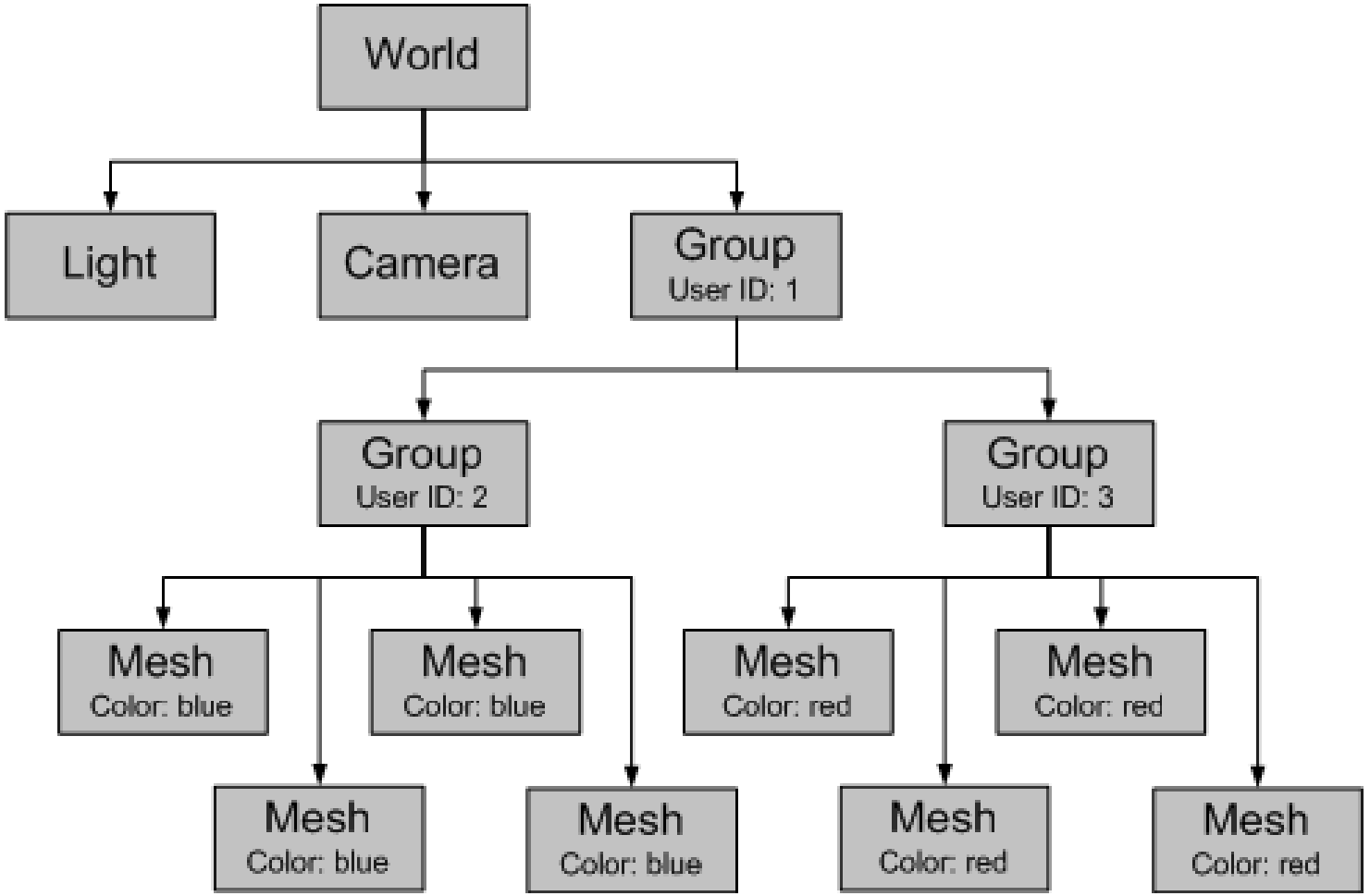
- Modeling
  - Créer tous les objets individuels qui seront utilisées ultérieurement dans la scène. On ajuste notamment :
    - Leur forme
    - Leur couleur
    - Luminosité
    - Transparence / opacité
    - …
- Aménagement de scène
  - Placer les objets sur la scène :
    - Les meshes
    - Les caméras
    - Les lumières
    - Les animations
- Rendering
  - Le processur final
  - « On filme la scène qu'on a préparé »

# Une scène 3D

- Un arbre, ou chaque feuille désigne un item physique ou abstrait dans un monde 3D :
  - Camera
  - Light
  - Mesh
  - …

- En J2ME :
  - Une scène peut contenir n'importe quel objet étendant : `javax.microedition.m3g.Object3d`
  - Une scène a toujours pour racine l'objet: `javax.microedition.m3g.World`

# Une scène 3D


World
Light
Camera
Group
User ID: 1
Group
User ID: 2
Group
User ID: 3
Mesh
Color: blue
Mesh
Color: blue
Mesh
Color: red
Mesh
Color: red
Mesh
Color: blue
Mesh
Color: blue
Mesh
Color: red
Mesh
Color: red

# Les modes : Immediate / Retained

- Immediate
  - Mode bas niveau
  - Fonctionne sur le même principe qu'OpenGL ou Direct3D
  - Permet le dessin direct d'objets

- Retained
  - Mode haut niveau
  - Utilise le graphe de scène (tous les éléments de la scène dans une arborescence)
  - Fonctionne sur le même principe que Java3D

# Les classes de base

# World

- Fourni un moyen de gérer tous les objets d'une scène.
- Peut être vu comme la « racine » d'une scène.
- Le nœud « world » ne peut avoir de parents.
- les transformations sont ignorées sur ce nœud lors du processus de rendering.
- En général un « World » est obtenu en lisant un fichier .m3G, en utilisant `javax.microedition.m3g.Loader`
- Un « World » peut avoir sa propre image de fond, définie par la classe `javax.microedition.m3g.Background`

# Loader

- C'est le « loader » des fichiers .m3g, implémenté par la plateforme sous-jacente.

- Permet de loader :
  - Les nœuds du graphe de scène
  - La scène entière

- Quand un fichier .m3g est chargé :
  - Toutes les classes étendant Object3D sont désérialisées
  - Un tableau d'objets désérialisés est retourné

```
Object3D[] o = null;
try
{
  o = Loader.load(name);
}
catch (Exception e) { }

World loadedWorld = (World) o[0];
```

# Object3D

- Classe abstraite, parente de tous les objets 3D
  - World
  - Tous les nœuds du graphe de scène
  - Animations
  - Textures
  - Meshes
  - …
- Fourni un ensemble de méthodes commun à toutes les classes
  - Sérialisation / désérialisation
  - Duplication d'objets
  - Définition d'ID et de paramètres
- Toutes les classes de l'API héritent de `Object3D` sauf `Graphics3D`, `Loader`, `Intersection` et `Transform`

# Graphics3D

- C'est le contexte graphique 3D, implémente le pattern Singleton
- On obtient l'instance par :

```
Graphics3D g3d = Graphics3D.getInstance()
```

- Doit être associé à une cible (un `Graphics` : `javax.microedition.lcdui.Graphics`)
- Le rendering est effectué par la méthode `render()`
  - Mode « immediate » : on doit explicitement définir une « Camera » et les « Lights » si il y'en a
  - Mode « retained » : « camera » et « lights » par défaut définis dans le « World »
- Pour effectuer un rendu de scène :
  1. On « bind » (lie) l'instance Graphics3D à une cible `Graphics` avec la méthode `bindTarget()`
  2. On effectue le rendu avec la méthode `render()`
  3. On relache la cible avec la méthode `releaseTarget()`

# Graphics3D : un exemple

```
public class MyCanvas extends Canvas {

   Graphics3D myG3D = Graphics3D.getInstance();
   World world= ...
   public void paint(Graphics g) {
        boolean bound = false;
        try {
                //…lie avec le Graphics…
                myG3D.bindTarget(g);
                bound = true;
                 // ... Rendu de la scène...
                myG3D.render(world);

        } finally {
                if (bound){
                        //relache la cible
                        myG3D.releaseTarget();
                }
        }
}
```

# Transformable

- Classe abstraite. Elle défini pour un node :
  - La translation
  - L'échelle
  - La rotation
  - Une matrice libre de transformation

- Multiplication de matrices pour avoir le vecteur p' par rapport à p

(p et p' : coordonnées homogènes) :

$$\mathbf{p' = T\,R\,S\,M\,p}$$

p'(x',y',z',w') et p ( x,y,z,w), deux vecteurs homogènes

x, x' : abscisse  y, y' : ordonnée

z, z' : cote  w, w' : échelle

T : Matrice de translation

R : Matrice de Rotation

S : Matrice d'échelle (Scale)

M : Matrice 4x4

# Node

- Classe abstraite, étend `Transformable`.
- Classe parente de tous les éléments d'un graphe de scène :
  - `Camera`
  - `Mesh`
  - `Sprite3D`
  - `Light`
  - `Group`
- Chaque `Node` défini un système de coordonnées qui peut être transformé relativement au système de coordonnée du nœud parent.

# Group

- Un noeud d'un graphe de scène qui contient un ensemble d'autres nœuds.

- Les groupes sont utiles quand :
  - On veut rendre visible ou invisible un grand nombre d'objets en même temps
  - On veut agir sur un ensemble d'objets (une voiture et ses 4 roues par exemple)

# Mesh

- « Maillage », C'est un ensemble :
  - de points 3d (x,y,z) : « vertices »
  - De connexion entre ces points : « polygons »
  - De description de textures à appliquer à ces polygones : « surfaces »

- La classe `Mesh` encapsule
  - un `VertexBuffer` (défini les vertices)
  - Un ou plusieurs `IndexBuffer` (défini les connexions entre les vertices)
  - Zero ou plusieurs `Appearance` (défini les attributs de rendu du maillage : lumière, texture, …)

- Dans sa forme la plus simple, un Mesh pouvant être rendu contient
  - Trois « vertices »
  - Un polygone « triangle »

- **Une restriction :** les polygones doivent être des triangles (la seule implémentation de `IndexBuffer` est `TriangleSplitArray !`)

- Plusieurs sous-mesh peuvent être définis dans un Mesh, mais tous les sous-mesh partagent le même `VertexBuffer`.

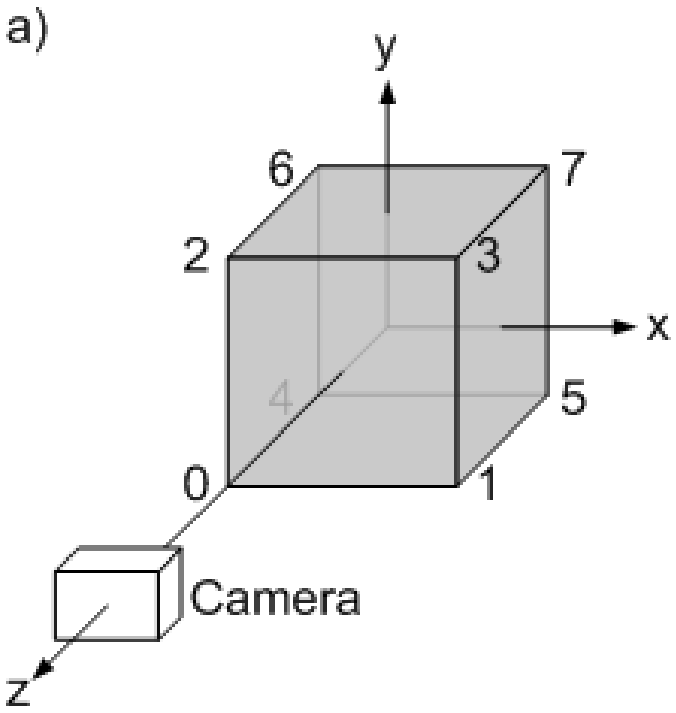
# Mesh : un cube

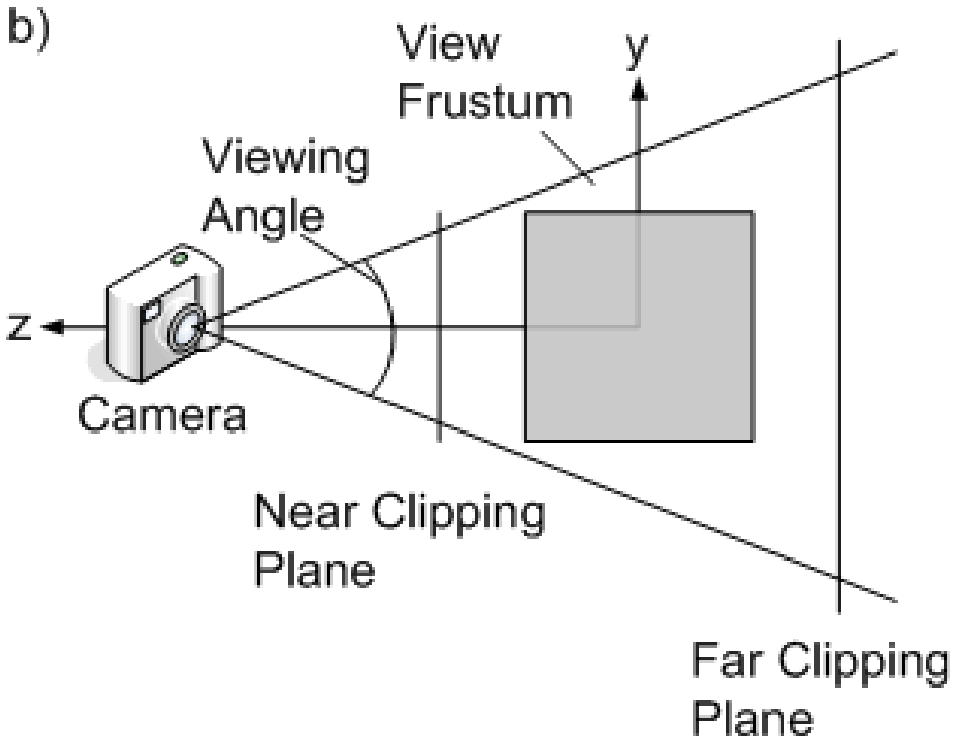
```
// The cube's vertex positions (x, y, z).
private static final byte[] VERTEX_POSITIONS = {
   -1, -1, 1,
   1, -1, 1,
   -1, 1, 1,
   1, 1, 1,
   -1, -1, -1,
   1, -1, -1,
   -1, 1, -1,
   1, 1, -1 };

// Indices that define how to connect the vertices to build triangles.
 private static int[] TRIANGLE_INDICES = { 0, 1, 2, 3, 7, 1, 5, 4, 7, 6, 2, 4, 0, 1 };

// Create vertex data.
VertexBuffer cubeVertexData = new VertexBuffer();
VertexArray vertexPositions = new VertexArray(VERTEX_POSITIONS.length/3, 3, 1);
vertexPositions.set(0, VERTEX_POSITIONS.length/3, VERTEX_POSITIONS);
cubeVertexData.setPositions(vertexPositions, 1.0f, null);


// Create the triangles that define the cube; the indices point to
// vertices in VERTEX_POSITIONS.
TriangleStripArray cubeTriangles = new TriangleStripArray( TRIANGLE_INDICES, new int[]
   {TRIANGLE_INDICES.length});

// Create a Mesh that represents the cube.
Mesh cubeMesh = new Mesh(cubeVertexData, cubeTriangles, new Appearance());
```

# Camera

- Un nœud du graphe de scène qui défini la position d'un « spectateur dans la scène »
- Une caméra effectue une projection de la 3D (la scène 3D) en 2D (de l'affichage sur écran).
- On peut définir autant de caméra que l'on veut par scène/

# Camera


a)
y
x
z
6
7
2
3
4
5
0
1
Camera

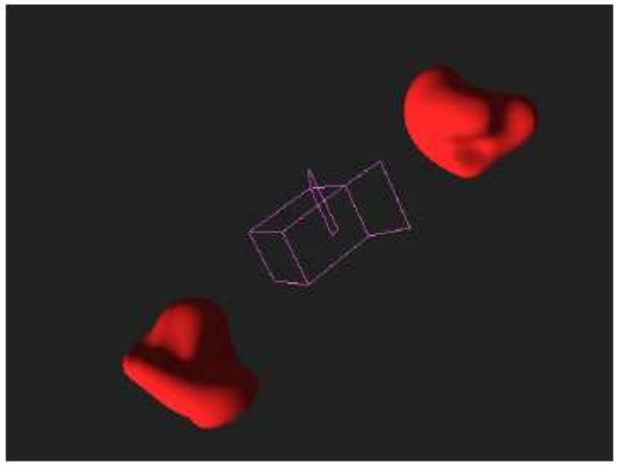

b)
View
Frustum
y
Viewing
Angle
z
Camera
Near Clipping
Plane
Far Clipping
Plane

# Light

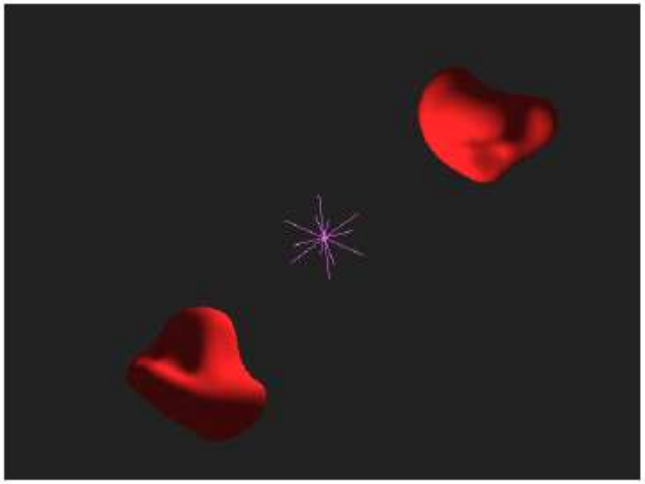
L'API M3G défini 4 types d'éclairage, avec des complexités de calcul différentes:

- éclairage d'ambiance (ambient light)
  - Défini l'intensité générale d'éclairage des objets sur la scène
  - Toute la scène est éclairée avec la même intensité
    - La position et direction de la lumière est ignorée au calcul de la lumière : économie de CPU
- Éclairage directionnel (directional light)
  - Défini d'où la lumière vient
  - Ne défini pas la position et la distance de la source de lumière aux objets
  - Utilisé pour simuler des sources de lumières lointaines (le soleil)
    - Usage moyen du CPU
- Eclairage omnidirectionnel (Omni Light)
  - Défini la source lumineuse comme un point
  - Tous les objets l'entourant sont affectés
  - Atténuation de la lumière en fonction de la distance des objets
    - Usage intensif du CPU
- Eclairage par spot (Spot Light)
  - Défini la position, la direction et le cône de la source lumineuse
  - Seuls les objets dans le cône de lumières sont affectés
  - Atténuation de la lumière en fonction de la distance des objets
    - Usage très intensif du CPU

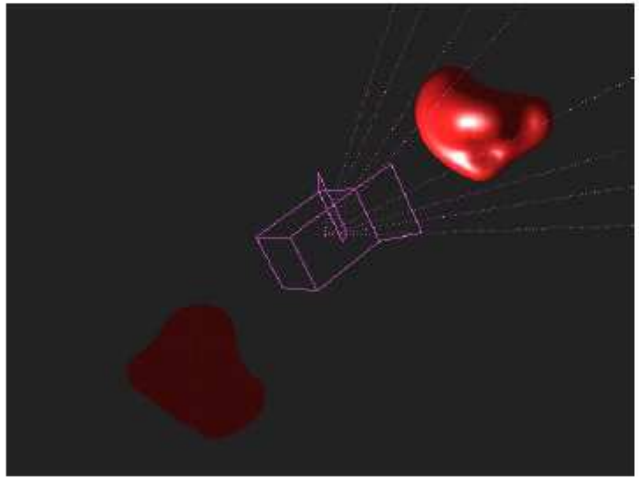
# Light : exemples



Eclairage directionnel



Eclairage omnidirectionnel



Eclairage par spot

# TP1 : le cube

- Dans la méthode `paint()` de la classe `VerticeSample` :
  - Mettre les trois méthodes :
    - `bindTarget()`
    - `render()` : attention, immediate mode !
    - `releaseTarget()`

# TP2 : Un texte 3D

**Dans Blender :**

- Démarrer avec un nouveau fichier, enlever le cube positionné par défaut (Ctrl-X)
- Positionner le curseur 3D à l'origine dans « front view » et « side view » (Shift-S-3) (snap to grid)
- Dans "Front View", ajoutez du texte (**Add > Text**)
- Basculez dans le mode d'édition (Tab), et affichez l' « editing context » (F9) , cliquez sur center, et placez la valeur Extrude à 0.100 : texte en relief
- Dans le mode « Object mode », sélectionnez le texte avec un clic droit, convertissez le texte (Alt-C) **font > curve**, puis **curve > mesh**
- Le texte toujours sélectionné, affichez l' « editing context » (F9), et créez un nouveau « material » dans « **Links and materials** » avec le bouton **new**
- Exporter au format M3G (**File > Export > M3G**). Sauvegarder votre fichier m3g

**Dans Eclipse :**

17. Dans la méthode `init()` de `Text3D`, coder le chargement du monde (World) par le `Loader` à partir du fichier m3g

# TP3 : Bebe

**Dans Blender :**

2. Ouvrir le fichier `bebe.blend`

- Exporter au format M3G (**File > Export > M3G**). Sauvegarder votre fichier m3g

**Dans Eclipse:**

7. Dans le `run()` de la classe `Bebe`, coder une rotation sur le mesh Bebe

**Indice :** le code de la transformation à effectuer toutes les 50 millisecondes sur le Mesh est :

```
postRotate(-2.0f, 0.0f, 0.0f, 1.0f)
```

11. Coder le démarrage du thread pour l'animation.