Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Master Informatique 1

Projet Infographie 2022

Arbres de scènes en résolution adaptative

⓪ Modélisation des formes canoniques

Chaque forme de base (Cube, Sphere, Cylindre, Tore, Cône) est définie par les équations paramétriques caractérisant les points de leurs surfaces (vertex) ainsi que les vecteurs normaux (ou normales).

☞ ces équations sont présentées dans le TD0-Géométrie.

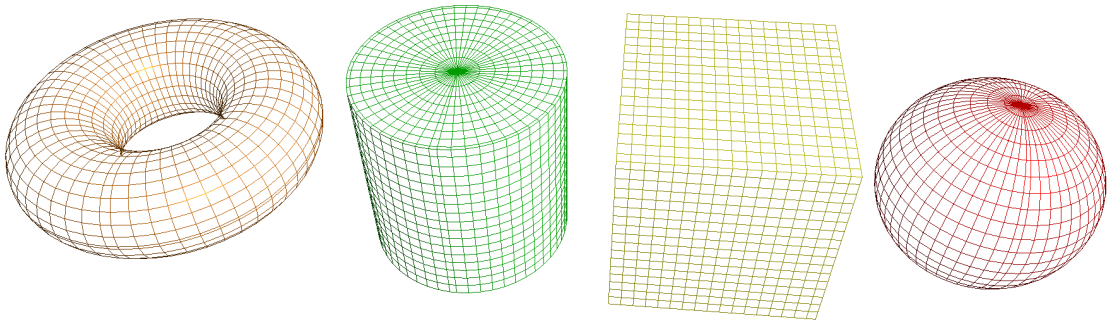
Comme vu en TD, chaque forme sera représentée par deux tableaux (les vertex et les normales) dont les tailles totales, identiques seront indexées sur une *résolution maximale* MAXRES (entre 100 et 1000). Cette résolution maximale correspondra à une forme dans sa taille maximale, correspondant à la taille de référence de l'objet canonique (le cube de côté 2, la sphère de rayon 1, le cylindre de rayon 1 et hauteur 2....).

Une forme canonique pourra donc être représentée simplement par quelque chose comme :

```
typedef struct _shape_
{
  int n1,n2,n3;    /* valeurs d'échantillonnage max - la plupart du temps 2 suffisent           */
  G3Xpoint  *vrtx; /* tableau des vertex   - spécifique d'une forme                              */
  G3Xvector *norm; /* tableau des normales - spécifique d'une forme                              */
  /*                  méthode d'affichage  - spécifique d'une forme                              */
  void (*draw_points)(struct _shape_*, G3Xvector scale_factor); /* mode GL_POINTS               */
  void (*draw_faces )(struct _shape_*, G3Xvector scale_factor); /* mode GL_TRIANGLES ou GL_QUADS */
} Shape;
```

D'autres champs pourront venir compléter cet objet ou ajuster les méthodes (cf.④).

☞ des indications pour déterminer ces tailles sont présentées en fin de TD3-Modélisation. Elles doivent être définies de sorte que l'échantillonnage de chaque forme produise des facettes de tailles *à peu prés équivalentes* d'une forme à l'autre.



Chaque forme sera crée en *1 seul exemplaire* (tableaux), à la résolution maximale, et une scène sera constituée d'objets qui seront des *instances* de ces formes, affichées avec une résolution réduite indexée principalement sur leur tailles (facteurs d'homothétie, supposés toujours ≤ 1.0).

☞ toutes les instances d'une forme données devront donc être plus "petites" que la forme canonique de référence.

① Affichage des instances

Cet affichage se fera par lecture des tableaux de vertex et normales avec les résolutions adaptées (les pas de parcours (1, 2 ou 3 selon la forme) ayant été déterminés dans l'étape de modélisation)

☞ là encore, les techniques de construction, de parcours des tableaux, de raccordement et d'affichage par points ou par `GL_QUADS` sont détaillées dans les TD2-Dessin et TD3-Modélisation

> ☞ à ce stade, l'évaluation portera essentiellement sur l'homogénité de représentation des instances des formes et des résolutions.

② Arbres de scène & Transformations géométriques

Une scène 3D est généralement représentée sous forme de *graphe*. A notre stade nous nous contenterons d'une représentation sous forme d'arbre. Cette structuration hiérachique est très puissante lorsqu'on l'associe à des transformations géométriques par matrices homogènes (cf. TD5-Transformations).

Un noeud de cet arbre contiendra, outre les champs classiques de chaînage vers les noeuds fils et frères, des champs "graphiques" qui pourront être utilisés ou non, selon les besoins, par exemple :

```
typedef struct _node_
{
  struct _node_ *down,*next;   /* chaînage                                    */
  G3Xhmat   Md;                /* matrice de transformation directe           */
  G3Xcolor  col;               /* couleur RGBA                                */
  Material  mat;               /* 4 réels dans [0,1] : (ambi, diff, spec, shine) */
  G3Xvector scale_factor;      /* facteurs d'échelles locaux en x,y,z,        */
  Shape    *instance;          /* une éventuelle instance d'objet             */
} Node, *SceneTree;
```

- par défaut, un `Node` ne contient pas d'`instance` mais hérite de tous les autres paramètres de son noeud père : `Md, col, mat, scale_factor` .
- si le `Node` ne contient pas d'`instance`, c'est juste un noeud interne qui joue un rôle d'intermédiaire pour, par exemple, configurer une transformation, une couleur, une matière... qui seront transmises à toute sa descendence.
- si le `Node` contient une `instance`, celle-ci se verra appliquer les transformations empilées dans la matrice `Md`, et sera affichée par les méthodes propres à l'`instance`, avec les résolutions locales déduites de `scale_factor` et les paramètres de couleurs/matières `col,mat`.
- un `Node` qui contient une `instance` n'est pas forcément une feuille de l'arbre (même si c'est souvent préférable), mais une feuille contient nécessairement une `instance`.

☞ c'est donc bien l'arbre qui porte toute l'information "graphique" (transformations, couleur, matière, facteurs d'échelle).

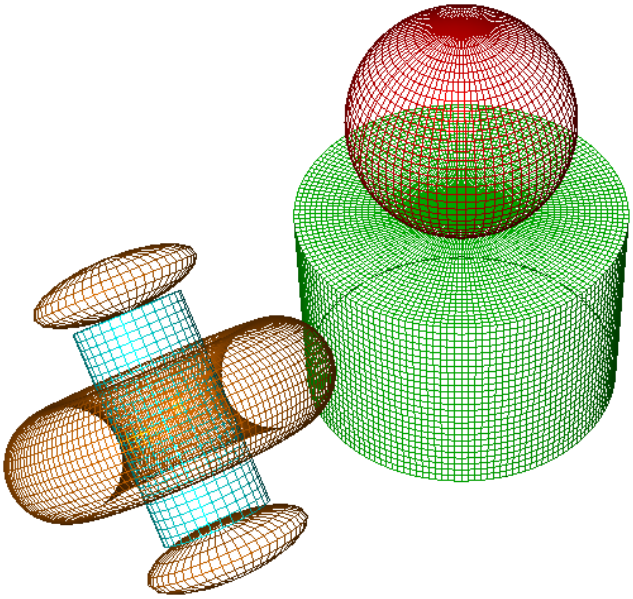
☞ appliquer une transformation à un `Node` revient donc à multiplier sa matrice (héritée du noeud père) par la matrice de la transformation souhaitée : `Md = g3x_Mat_x_Mat(Md,g3x_Translation3d(tx,ty,tz))`.

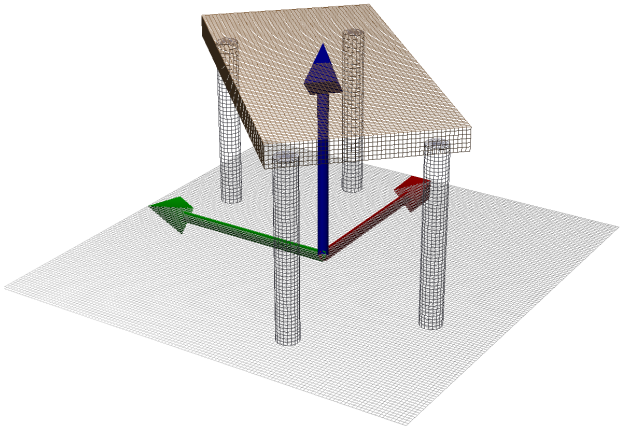
☞ l'affichage d'un arbre se fait donc simplement par parcours des `Node`. Si une `instance` est présente, on l'affiche, et on poursuit :

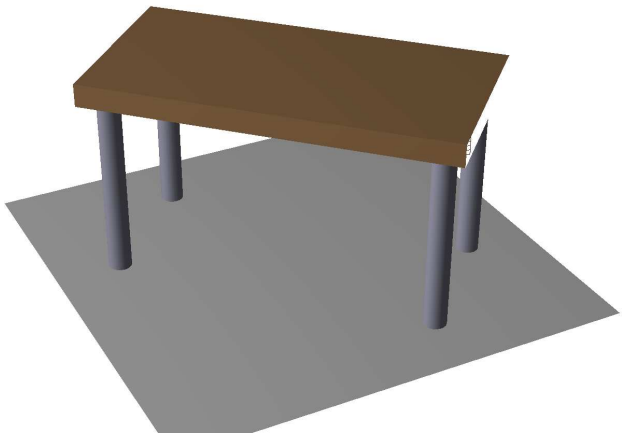
```
-------------------------------------------------------------------------
/* couleur, matière                                                     */
g3x_Material(node->col,node->mat[0],node->mat[1],node->mat[2],node->mat[3],0.);
glPushMatrix();              /* ouverture de la pile OpenGl             */
  glMultMatrixd(node->Md.m); /* chargement de la matrice locale         */
  /* affichage avec la bonne résolution                                 */
  node->instance->draw_faces(node->instance, node->scale_factor);
glPopMatrix();               /* déchargement & fermeture de la pile */
-------------------------------------------------------------------------
```

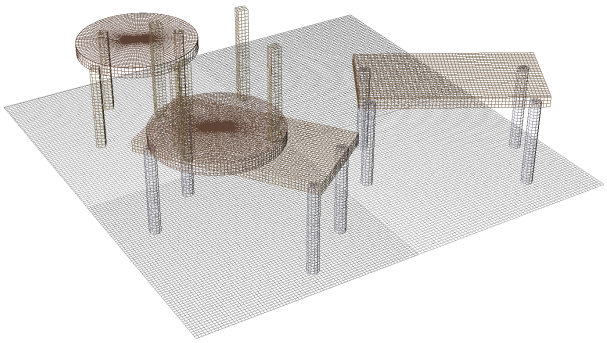
☞ le vecteur d'échelle `scale_factor` étant représentatif des déformations appliquées à la forme, il devra être mis à jour lorsque le `Node` sera soumis à une Homothétie.

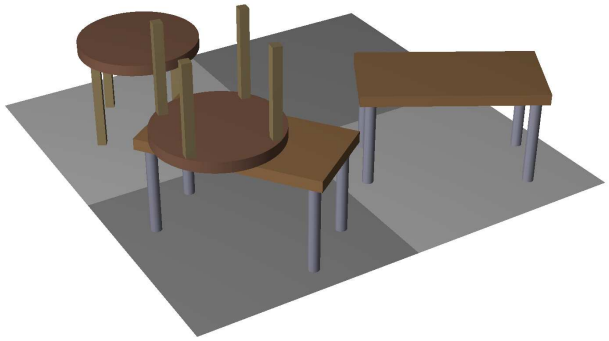
☞ pour cette seconde étape, la plus importante, l'évaluation portera sur la gestion des arbres, la propagation des transformations et des facteurs d'échelle.
☞ toutes les instances, quelles que soient leur forme et leur taille, doivent présenter, à l'affichage, une résolution à peu près homogène (taille de la "facette élémentaire").











③ Adaptation dynamique : prise en compte des positions relatives `Objet/Camera`

Pour cette dernière étape, il s'agit d'améliorer l'aspect adaptatif de la résolution d'affichage des `instances` en lui conférant une propriété dynamique, conditionnée par les positions relatives des objets dans la scène (proches de la caméra ou lointain).

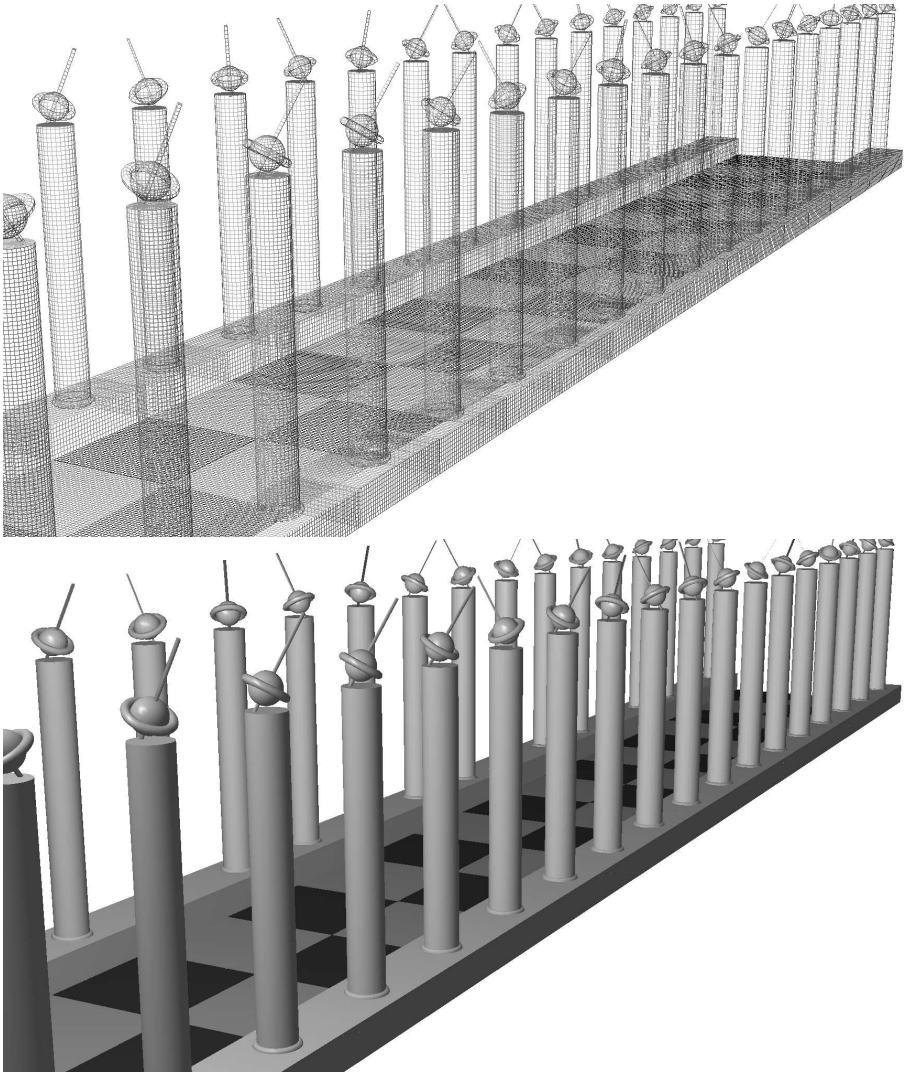
L'objectif est bien sûr de garder toujours une résolution (i.e. "qualité" d'affichage) *visuellement constante* : résolution élevée pour les `instances` en gros plan, faible pour les fonds de scène.

Pour cela il faut tout d'abord récupérer la position de la caméra : `G3Xpoint campos=g3x_GetCamera()->pos;` puis déterminer, pour chaque `Node` , à chaque vue (appel à la fonction `<draw()>`), quelle est sa distance à la caméra.

Deux options équivalentes pour cela (on partira du principe que la caméra vise *toujours* l'origine du repère, centre de la scène) :

- introduire dans `Node` un champs supplémentaire `double dcam`, mis à jour par héritage et lors des appels aux translations (comme le `scale_factor`)
- ou, au dernier moment, juste avant l'appel à la procédure d'affichage, récupérer la dernière colonne de la matrice de transformation locale `Md`, interpréter ces valeurs[(1)] `(Md.m[12],Md.m[13],Md.m[14])` comme *la* position du `Node` et calculer la distance à la caméra.

Bien sûr cette distance `dcam` devra être transmise, comme le `scale_factor` aux procédures d'affichage et intégrée dans le calcul des pas de parcours des tableaux (pondération de `scale_factor` par `1./dcam`).



[(1)] les rotations et homothéties n'influent pas sur ces valeurs – cf.TD5-Transformations