TP Synthèse d'images: Modélisation
CPE

durée - 4h

2011/2012

1 Prise en main de l'environnement

1.1 Classe d'affichage de la scene (scene.hpp)

La structure de base de l'affichage est expliquée en fig 1

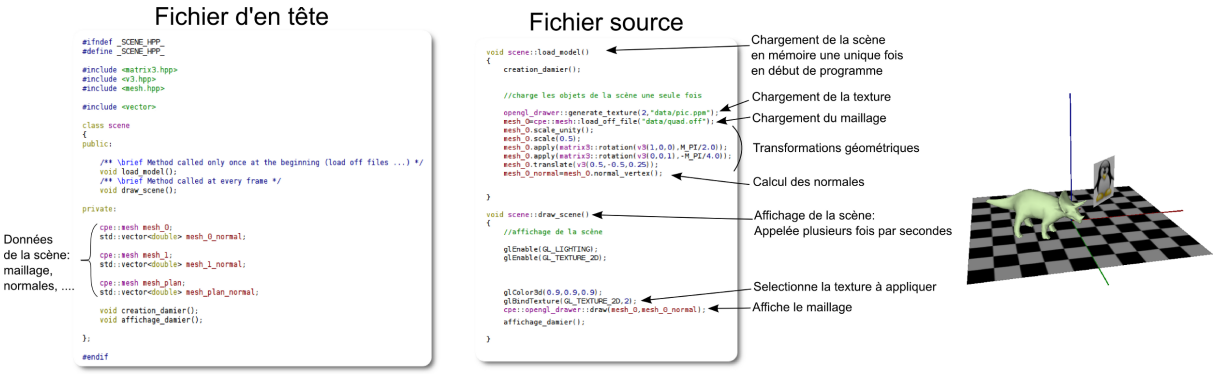

FIGURE 1 – Explication de l'affichage initiale.

Question 1 *Effectuez le chargement d'un autre maillage. Déplacez et modifiez l'orientation des maillages dans la scène lors de leur chargement. Observez la différence entre un affichage par normales exprimées par sommets/par triangles.*

1.2 Manipulation des classes de bases

Vous disposez de quelques classes de bases :
- v3 : Representant le vecteur (x, y, z).
- v4 : Representant le vecteur (x, y, z, w).
- matrix3 : Representant une matrice de transformation affine

$$m = \begin{pmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{pmatrix}$$

Les classes de bases disposent des opérations classiques. ex.

```
v3 x0(1,3,-2);                    \\x0=[1,3,-2]
v3 x1=v3(-1,-1,1)+x0;             \\x1=[0,2,1]
matrix3 m=matrix3::scale(5,5,1); \\m=diag(5,5,1)
double y=(m*x0).dot(x0);          \\y=<(m*x0),x0>
double x0_y=x0.y();               \\x0_y=3
```

La documentation complète des opérations disponibles est fournie ici : *doc/html/index.html*
Des classes plus spécifiques sont également fournies dans le dossier : *external/*
L'ensemble de ces classes sont accessibles depuis l'espace de nom *cpe*.

Question 2 *(Dans la methode scene : :loadModel) Effectuez la transformation du vecteur* $a = (1, 7, -5)/8$ *par la rotation* R *d'axe* $(1, 1, 1)/\sqrt{3}$ *et d'angle* $\pi/4$.

Calculez le vecteur b *normal à* a *et de son image par* R.

Construisez c *tel que* (a, b, c) *doit une base orthogonale de l'espace. Construisez la base orthonormale associée.*

1.3 Affichage en mode immédiat

La fonction *draw scene* peut être directement complétée pour un affichage OpenGL.

Voici la syntaxe pour tracer un segment jaune entre les positions $(0.1, 0.1, 0.2)$ et $(0.7, 0.8, 0.7)$:

```
glColor3d(1,1,0);
glBegin(GL_LINES);
glVertex3d(0.1,0.1,0.2);
glVertex3d(0.7,0.8,0.7);
glEnd();
```

Question 3 *Affichez la base orthonormale calculée précédemment et vérifiez visuellement son orthogonalité.*

L'affichage d'un triangle en mode immédiat suit la syntaxe suivante :

```
glBegin(GL_TRIANGLES);
glVertex3d(0,0,0);
glVertex3d(1,0,0);
glVertex3d(0,1,0);
glEnd();
```

Pour obtenir une interpolation des couleurs et des normales exprimées par sommet, on peut compléter cet appel :

```
glBegin(GL_TRIANGLES);

glColor3d(1,0,0);
glNormal3d(0,0,1);
glVertex3d(0,0,0);

glColor3d(0,1,0);
glNormal3d(0,0,1);
glVertex3d(1,0,0);

glColor3d(0,0,1);
glNormal3d(0,0,1);
```

```
glVertex3d(0,1,0);

glEnd();
```

Question 4 *Affichez en mode immédiat le carré unitaire de la fig. 2 (avec sa bordure) :*

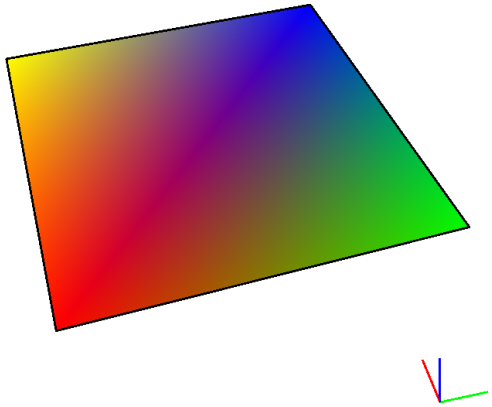

FIGURE 2 – Affichage d'un quad.

L'affichage en mode immédiat est excessivement lent. La concaténation sous forme de vecteurs de sommets et de connectivitée contigue en mémoire permet un accès rapide. La syntaxe est la suivante pour l'affichage du même quad :

```
//donnees: sommets, couleurs, normales, connectivitee
//  contigues en memoire
double p_vertex[3*4]={0,0,0 , 0,1,0 , 1,1,0 , 1,0,0};
double p_color[3*4] ={1,0,0 , 0,1,0 , 0,0,1 , 1,1,0};
double p_normal[3*4]={0,0,1 , 0,0,1 , 0,0,1 , 0,0,1};

unsigned int p_connectivity[3*2]={0,1,2 , 0,2,3};

//activation des modes d'affichages utilisees
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_NORMAL_ARRAY);
glEnableClientState(GL_COLOR_ARRAY);

//designation des pointeurs
glVertexPointer(3,GL_DOUBLE,0,p_vertex);
glColorPointer(3,GL_DOUBLE,0,p_color);
glNormalPointer(GL_DOUBLE,0,p_normal);

//affichage des triangles
glDrawElements(GL_TRIANGLES,3*2,GL_UNSIGNED_INT,p_connectivity);

//desactivation des mode d'affichages
glDisableClientState(GL_VERTEX_ARRAY);
glDisableClientState(GL_NORMAL_ARRAY);
glDisableClientState(GL_COLOR_ARRAY);
```

1.4 Structure mesh

La structure *mesh* stocke les sommets, les textures au besoin, et la connectivitee sous une forme compatible à leur affichage rapide.

Elle permet egalement de calculer les normales : soit par sommet, soit par triangles.

Les maillages complexes sont fastidieux à coder manuellement. Ils sont stockés dans des fichiers separés (ASCII). Un chargeur d'un format particulier (OFF) est fourni.

Une classe d'aide *opengl drawer* permet d'afficher aisément la structure *mesh*. Suivant le vecteur de normale envoyé (par sommet/triangles), la procédure d'affiche est modifiée. Une demande d'affichage sans specifier de normales donne lieu à l'affichage des arêtes.

Un exemple de construction directe et d'affichage à l'aide de la classe *mesh* pour un tetraèdre est proposé en fig. 3.

```
glEnable(GL_LIGHTING);
glColor3d(0.7,0.7,0.7);

cpe::mesh m;
m.add_vertex(0,0,0); m.add_vertex(0,1,0); m.add_vertex(1,0,0); m.add_vertex(0,0,1);
m.add_triangle(0,2,1); m.add_triangle(1,2,3); m.add_triangle(2,0,3); m.add_triangle(0,1,3);

opengl_drawer::draw(m,m.normal_polygon());
```

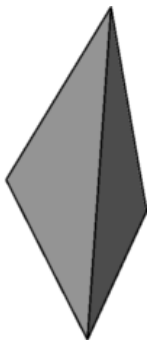

FIGURE 3 – Affichage d'un tetraedre a l'aide de la class *mesh*.

L'appel à la méthode *draw debug(m);* permet d'afficher à l'aide de l'interface graphique l'indice des faces et des sommets.

Question 5 *Réalisez cet appel à* draw debug *pour différent maillages grossiers.*

2 Maillage

Question 6 *Construisez un maillage représentant une structure 2D d'un terrain. C'est à dire modélisant une fonction donnée par $z := (x, y) \mapsto z(x, y)$. La hauteur z du terrain pourra être donnée par des fonctions trigonométriques (voir fig. 4).*

Notez l'ordonnancement de l'orientation des triangles afin de parcourir l'ensemble des indices des sommets. Vous pourrez commencer par un maillage très grossier dans un premier temps modélisant la fonction $z(x, y) = 0$. N'oubliez pas l'appel à la méthode de debug pour visualiser vos indices.

Question 7 *Si vous avez le temps, ajoutez des coordonnées de texutres et texturez vos terrain.*

Question 8 *Si vous avez le temps, ajoutez une variable temporelle s'incrémentant à chaque affichage. Réalisez alors un terrain dont les hauteurs sont animées. On pourra penser dans ce cas à la modélisation d'une surface fluide.*

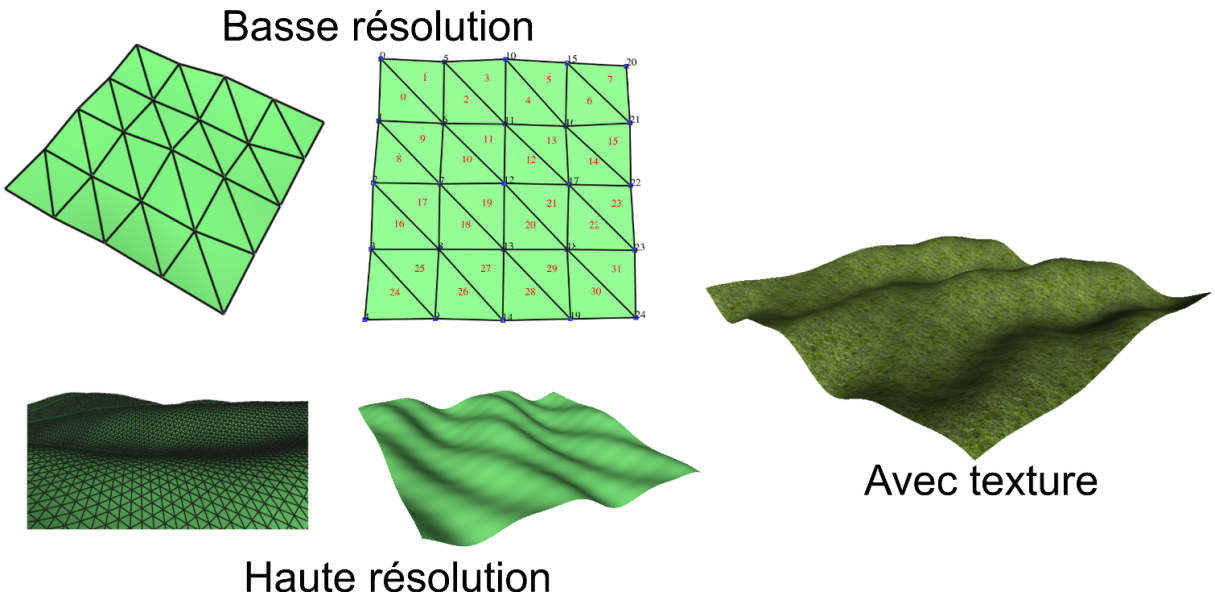

FIGURE 4 – Exemple de terrain possible.

Question 9 *Construisez une structure de maillage modélisant une sphère (voir fig. 5). On pourra pour cela, placer des sommets paramétrés par leurs coordonnées sphériques.*

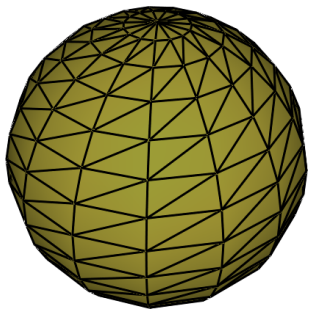

FIGURE 5 – Maillage de sphère paramétrée suivant les coordonnées sphériques.

Question 10 *Créez une méthode qui génère une ellipsoide dont les axes principaux seront donnés par les axes x, y, z et dont l'amplitude de ces axes sera donnée en paramètre.*

Question 11 *Généralisez la fonction précédente en permettant d'orienter les axes propres de l'ellipse suivant une base quelconque de l'espace qui sera donnée en paramètre (voir fig. 6). Pour cela, on appliquera une rotation spécifique sur l'ellipsoide créée précédemment (voir annexe rotation).*

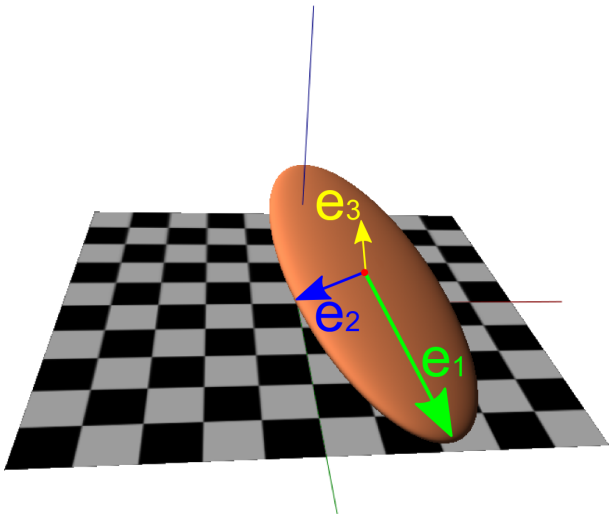

FIGURE 6 – Exemple d'ellipsoide orientée. Les vecteurs $\mathbf{e}_1$, $\mathbf{e}_2$ et $\mathbf{e}_3$ donnent l'information d'amplitude et d'orientation de la forme.

Question 12 *Créez un objet cylindrique aligné le long de l'axe z de rayon unitaire. Généralisez ensuite ce cylindre de manière à le paramétrer en donnant un vecteur directeur et un rayon (voir fig 7).*

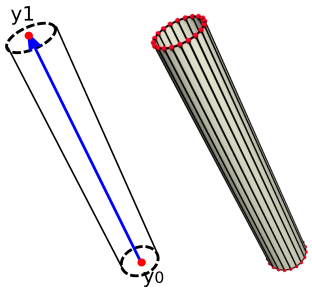

FIGURE 7 – Exemple de tube paramétré par $\mathbf{y}_0$, $\mathbf{y}_1$ et son rayon.

Question 13 *Si vous avez le temps, modélisez une scène avec un terrain et une soucoupe volante posée sur celui-ci par assemblage de primitives simples.*

Annexe rotation

On rappelle que la rotation du vecteur $\mathbf{v}$ autour de l'axe unitaire $\mathbf{u}$ et d'angle θ peut se calculer à l'aide de la formule de Rodriges :

$$\mathbf{v}_{\text{rot}} = \mathbf{v}\cos(\theta) + (\mathbf{u} \times \mathbf{v})\sin(\theta) + (\mathbf{u} \cdot \mathbf{v})(1 - \cos(\theta))\,\mathbf{u}\,. \tag{1}$$

De manière équivalente, il est possible d'exprimer cette rotation sous forme matricielle (voir fig. 8).

```
matrix3 matrix3::rotation(const v3& axis,const double& angle)
{
    v3 n=axis.normalized();

    double cos_t=std::cos(angle);
    double sin_t=std::sin(angle);

    return matrix3(cos_t+n[0]*n[0]*(1-cos_t),n[0]*n[1]*(1-cos_t)-n[2]*sin_t,n[1]*sin_t+n[0]*n[2]*(1-cos_t),
                   n[2]*sin_t+n[0]*n[1]*(1-cos_t),cos_t+n[1]*n[1]*(1.0-cos_t),-n[0]*sin_t+n[1]*n[2]*(1-cos_t),
                   -n[1]*sin_t+n[0]*n[2]*(1-cos_t),n[0]*sin_t+n[1]*n[2]*(1-cos_t),cos_t+n[2]*n[2]*(1-cos_t));

}
```

FIGURE 8 – Implémentation d'une matrice de rotation paramétrée par son axe et son angle.